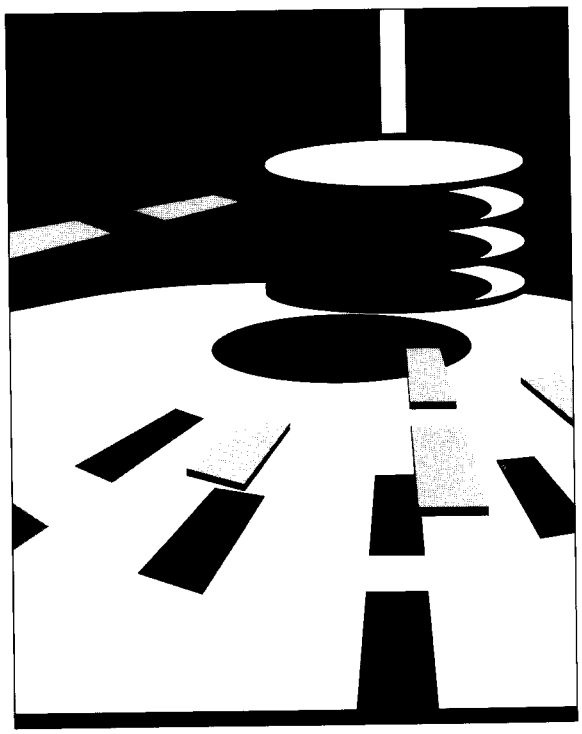
T A N D E M

SYSTEMS REVIEW

VOLUME 1, NUMBER 2

JUNE 1985



B00 Software

Data Communications

Workstations

Peripherals

Technical Paper

Editor's Note

I'd like to thank the many individuals and groups within Tandem that worked together to create this special issue. Some worked many extra hours to complete the articles and ensure their timely publication. Special thanks to: Mala Chandra, Joanne Danforth, Jim Eakin, Hank Hugeback, Kent Madsen, Sarah Rood, Carol Schaffer, Dick Thomas, Ann Whitesell, the 25 authors, the entire Illustration and Typesetting groups, and the technical reviewers in the Field Productivity Programs, Hardware Development, Installability, Large Systems Support, Product Management, Product Marketing, Software Development, and Systems Support groups.

Carolyn Turnbull White

Syntax notation used in this publication.

Notation	Meaning
UPPERCASE LETTERS	Represent keywords and reserved words.
lowercase letters	Represent variable entries supplied by the user.
Brackets [ ]	Enclose optional syntax items.
Braces { }	Enclose required syntax items.
Ellipses . . .	Follow syntax items that can be repeated any number of times.
Ellipses preceded by a comma , . . .	Follow syntax items that can be repeated any number of times and require a comma to separate each repetition.

Correction:

BASE24, an electronic funds transfer system, was mentioned in the article, "Using FOX to Move a Fault-tolerant Application," published in the February 1985 issue of the *Tandem Systems Review*. BASE24 is a software product offered by Applied Communications, Inc. (not Advanced Communications, Inc., as stated). The product has a multinodal, expandable structure analogous to Tandem's hardware structure and is designed for fail-safe, 24-hour processing seven days a week.

TANDEM SYSTEMS REVIEW

Volume 1, Number 2, June 1985

Editor
Carolyn Turnbull White

Technical Advisors
Mala Chandra
Dick Thomas

Associate Editors
Kent Madsen
Ann Whitesell

Assistant Editor
Sarah Rood

Art Director
Terri Hill

Designers
Joanne Danforth
Carol Schaffer

Production and Layout
Laurie Menden
Gayle Richardson
Tandem Illustration Group

Cover Art
Stephen Stavast

Typesetting
Tandem Typesetting Group

The *Tandem Systems Review* is published by Tandem Computers Incorporated.

Purpose: The *Tandem Systems Review* publishes technical information about Tandem software releases and products. Its purpose is to help programmer-analysts who use our computer systems to plan for, install, use, and tune Tandem products.

Subscription additions and changes: Subscriptions are free. To add names or make corrections to the distribution data base, requests within the U.S. should be sent to Tandem Computers Incorporated, Sales Administration, 19191 Vallco Parkway, Cupertino, CA 95014. *Requests outside the U.S. should be sent to the local Tandem sales office.*

Comments: The editor welcomes suggestions for content and format. Please send them to the *Tandem Systems Review*, 1309 So. Mary Avenue, Sunnyvale, CA 94087.

Highlights of the B00 Software Release

Tandem's B00 software runs on NonStop II, NonStop TXP, and NonStop EXT systems. Many new products and enhancements to existing products make up the release. This article highlights the major ones, including those listed below:

New products

Disc Process 2 (DP2)

EXCHANGE/SNA

Tandem Maintenance and Diagnostic System TMDS)

Enhanced or changed products

ATP6100 Asynchronous Terminal Process

BASIC

COBOL

FORTRAN

FOX 6700 Fiber Optic Extension

GUARDIAN 90 operating system

ENABLE program generator

EXPAND networking software

PATHWAY transaction processing system

Program Development Tools (PDT)

Transaction Application Language (TAL)

Transaction Monitoring Facility (TMF)

TRANSFER and TRANSFER/MAIL

With the B00 software release, Tandem has completed a new, more powerful version of the GUARDIAN operating system. To emphasize this milestone, the operating system software, including DP2, has a new name: GUARDIAN 90. This new version of the operating system yields a significant improvement in on-line transaction processing performance in a TMF environment. The GUARDIAN 90 operating system is upwardly compatible with previous releases of the software, allowing users to migrate to it without rewriting their applications.[1]

Data-base Software

Disc Process 2 (DP2)

DP2 is the new GUARDIAN 90 disc process for NonStop II, NonStop TXP, and NonStop EXT systems. (It is not available for NonStop 1+ systems.) The job of both DP2 and DP1 is to manage data and hardware within Tandem disc subsystems. Both disc processes implement the various ENSCRIBE file types, manage the space on discs, and manage the controllers and paths to the discs.

DP2, however, is a complete redesign and reimplementation of the disc process, aimed at achieving improved performance, throughput, recoverability, reliability, maintainability, and extensibility. All discs and disc controllers for NonStop II, TXP, and EXT systems are supported by DP2. TMF, the file system, and many utilities have been appropriately modified and enhanced in connection with its development.

[1]The only exceptions to this are applications that modify the environment (ENV) register and privileged applications that access the Destination Control Table (DCT). These applications must be modified to accommodate the expanded user and system code space, as stated in following sections.

To achieve the goals mentioned above for DP2, developers have changed the basic file structure. Thus, those user files on which DP2 is to be used must be changed from DP1 format to DP2 format. New utilities are provided for this purpose. Other architectural changes have been made to integrate the TMF and DP2 subsystems, resulting in increased performance in the TMF environment.

Note: Because of the changes in the structure of files, programs that open structured files in unstructured mode must be modified if DP2 is to be used with them. See the *ENSCRIBE Programming Manual* for the details of the new block structure.

ENABLE Program Generator

The B00 release includes major enhancements to ENABLE. ENABLE can now generate applications that access multiple files organized in a hierarchical fashion.

Another new feature is its ability to display and update multiple records within a file. Optionally, ENABLE can display this information in tabular format.

Finally, ENABLE makes the process of designing screens more flexible. Users can suppress different fields from the screen display for security purposes and can rearrange the layout of fields on the screen.

For a discussion of these new capabilities, see Chapman and Zimmerman, 1985.

PATHWAY Transaction Processing System

The B00 release of PATHWAY includes a new version of the Screen COBOL compiler. This new compiler runs only on NonStop II, TXP, and EXT systems because it takes advantage of extended memory. By using extended memory features, the new compiler permits a significant increase in the number of symbols allowed in Screen COBOL run units.

Several new command options are available:

- AUTORESTART, a new optional parameter, allows the PATHWAY system to restart terminals, TCPs, and servers automatically when abnormal errors are encountered.
- An exclamation mark (!) has been added to the SHUTDOWN command as an optional parameter. It is the equivalent of a STOP TERM *, followed by a STOP TCP *, followed by a SHUTDOWN.
- An ERRORS command has been added to PATHCOM to allow errors to be tolerated in IN/OBEY file processing.
- The INFO command now has an optional keyword, OBEYFORM, which formats the output as a syntactically correct PATHCOM command with the proper SET <entity type> prefix. One use of this option is to create new cold-start obey files after extensive modifications to an existing PATHWAY configuration.
- A new CONTROL PATHMON and CONTROL TCP command has been added to allow changing of object descriptions while the objects are active.
- New PRIMARY and SWITCH commands aid in the load balancing of a PATHWAY system.

PATHWAY now supports double-width screen sizes for Tandem 6520 and 6530 terminals, and it recognizes the BREAK key on conversational terminals. Also, it is now permissible for Screen COBOL programs of different terminal types to call each other.

Transaction Monitoring Facility (TMF)

B00 includes significant enhancements to TMF in the areas of recovery, fault tolerance, and operational flexibility.

TMF now includes autorollback, a fast mechanism for recovering a data base that is protected by TMF. Autorollback starts automatically at TMF start-up time and operates on all audited and logically inconsistent files. It does not require the mounting of tapes, and no operator intervention is needed once recovery is under way.

Thanks to autorollback, the loss of a disc-process pair no longer causes a TMF crash. If both processors are lost, all active transactions that the disc process might have been working on are aborted. When the processors are subsequently reloaded, autorollback recovers the volume. (For more detail on autorollback, see Pong, 1985.)

TMF has several new features that provide greater operational flexibility:

- A new command has been added to TMFCOM to allow the user to add one or more volumes to the set of volumes that are up for TMF. This command might be used if the volumes were down at START TMF time, if they went down while TMF was running, or if they failed to start earlier.
- Audit-trail-file, extent-size, and MAXFILES parameters can be altered while TMF is stopped, and new volumes can be added to a TMF configuration without reinitializing TMF.

A significant portion of TMF has been rewritten in conjunction with the development of DP2, and many of the new TMF features, including volume reintegration and autorollback, are available with DP1 as well.

TRANSFER and TRANSFER/MAIL
Changes or new features in the TRANSFER delivery system and T/MAIL include:

- Increased user friendliness and performance for T/MAIL.
- Support for T-TEXT.
- Support for FAXLINK.
- A Queue Manager for queuing items to agents.
- A repair facility for TRANSFER data bases.
- Dated item "unsave."
- Ordered folders.
- Return codes from agents.
- Staged item deletion.

The Operating System

User and System Code Space
In GUARDIAN 90, user-code, user-library, and system-library spaces have been enlarged. This change permits larger programs to run and allows large programs written for other vendors' systems to migrate more easily to Tandem systems.

User-code and user-library spaces can each now contain up to 16 code segments of 128K bytes each for a total of 32 segments of user code (4M bytes). In addition, 33 system-code segments (4.125M bytes) can be provided, 32 of which are available for the system library and one for system code. In connection with this expansion, internal and external changes have been made to BINDER, TAL, FORTRAN, DEBUG, LOBUG, INSPECT, XRAY, CRUNCH, and SYSGEN.

To accommodate these enhancements, application programs that modify the stack marker ENV Register must be modified, as must privileged programs that access the Destination Control Table, or DCT (including user-written I/O processes). All other programs run successfully under GUARDIAN 90 without recompilation.

FOX Protocol
Changes to operating-system software associated with the 6700 Fiber Optic Extension (FOX) provide automatic ring-topology generation, automatic path management, and support for on-line diagnostics. These changes require that all nodes in a FOX ring be upgraded to the GUARDIAN 90 operating system simultaneously if FOX communications are to be maintained. (This is explained further in a later section and in the accompanying article, "Changes in FOX.")

New Tandem Maintenance and Diagnostic System (TMDS)

TMDS provides a foundation for integrated, on-line fault diagnostics for the hardware components of Tandem systems. This first release includes a FOX diagnostic subsystem. Other products will be added in subsequent releases. Major features of this release are a centralized, diagnostic command interpreter with extensive "help" facilities, a diagnostic resource monitor, and automatic logging of hardware faults.

Command Interpreter (COMINT)

Several features have been added to enhance the Command Interpreter (COMINT):

- A HELP command is now available to aid users in the proper use of COMINT.
- Users can now control system access during start-up through a SYSGEN-defined input file to the start-up command interpreter.
- COMINT now runs as a nonprivileged process. Its privileged functions are implemented as procedures that can be called or processes that can be run. Also, password management, system logon, and the information passed to a $CMON process have been enhanced.
- The manner in which start-up messages are handled is different. Consult the Softdoc for details.

System Timing

Time services have been enhanced to include:

- Automatic Daylight Savings Time (DST) adjustments.
- Improved process timing.
- Julian date conversions.
- Timestamps based on Greenwich Mean Time (GMT).

Other system-time enhancements include CPU clock-rate averaging, clock-rate adjustments, and a callable procedure to set system clocks. This procedure makes it easier to keep accurate time when external clocks (e.g., that of the U.S. government radio station, WWV) are used.

Also, four-word, microsecond-resolution timestamps are now available, in addition to the old three-word timestamps. The GMT timestamps and new time-conversion routines are all based on the new four-word timestamp.

Other GUARDIAN 90 Enhancements

SYSGEN can now process configurations with a greater number of named entities (devices and named processes).

The performance of RELOAD has been improved, and thus, systems can be started more quickly.

The DCT has been moved to extended memory and has been increased in size. Also, it is now accessed via a hashing algorithm, which improves speed. New GUARDIAN 90 procedures have been provided to access the DCT.

Processes can now be started in parallel in any processor. This, combined with the enhancements to RELOAD, improves system start-up time dramatically. However, applications that previously serialized their nowait process initializations by directing them all to the same CPU can no longer rely upon that method of serialization.

Data Communications

ATP6100

Software is included in this release to support asynchronous point-to-point communications with the 6100 Communications Subsystem. Included in ATP6100 is support for both Line Interface Unit 1 (LIU1) and the new LIU4.

EXCHANGE/SNA

The new EXCHANGE/SNA is a logical and compatible extension to the EXCHANGE communications subsystem for handling SNA RJE communications. Basically, it provides an emulation of an IBM 3777-3 Multiple Logical Unit (MLU) Data Entry Subsystem, interfacing primarily to IBM operating systems that support JES2 and JES3.

EXPAND
A new network-access interface now improves communications between EXPAND and "network service-provider processes" (e.g., X25AM and FOX). The handling of X.25 Switched Virtual Circuits (SVCs) has been enhanced to include:

- Auto-establishment and take-down without operator intervention.
- Auto-recovery from SVC loss.
- SVC take-down and reestablishment during inactive periods, without an interruption of the EXPAND "session."

Languages

Transaction Application Language (TAL)
TAL supports the increased code space now available with the GUARDIAN 90 operating system. A program produced by a single compilation can have up to 16 code segments of up to 128K bytes each. No one procedure can exceed 64K bytes of code.

COBOL
COBOL (T9251) for NonStop™ systems is validated at the Federal Information Processing Standards (FIPS) high level. It is upwardly compatible with the original COBOL compiler (T9201), which is now available only for NonStop 1+ systems. The original compiler is validated at the FIPS low-intermediate level.

COBOL supports the increased code space now available with GUARDIAN 90. Code space available to an individual program unit can be as large as 128K bytes. A run unit can occupy up to 16 code segments, each containing 128K bytes. Data items greater than 32K bytes are permitted.

COBOL programs can use the sequential block-buffering provided by ENSCRIBE.

The CALL identifier and CANCEL identifier statements have been implemented (e.g., "CALL program-ID" where program-ID is a variable in working storage).

Additional SORT/MERGE features have been implemented, including:

- Sort/merge to/from a blocked tape.
- Sort/merge to/from a multireel tape file.
- Sort/merge to/from multifile tapes.
- Sort/merge by a user-specified collating sequence.

FORTRAN
FORTRAN now allows users to define large COMMON blocks that reside in extended memory, providing 128M bytes of data storage. FORTRAN programs can have up to 16 code segments of up to 128K bytes each. (GUARDIAN 90 support is implicit.)

BASIC
Most GUARDIAN 90 procedures can be called directly with the new CALL statement. The standard CROSSREF utility is used to produce cross-reference listings. The floating-point arithmetic package is automatically used if present on the system.

Program Development Tools (PDT)
PDT products provide the necessary support for increased code and library space. INSPECT and DEBUG have been changed to allow references to all 16 code segments in user-code and user-library spaces.

Notes on Installation, Conversion, and Tuning

System managers and application programmers at sites planning to migrate to the B00 Limited Customer Distribution should be aware of the following considerations.

Installation

Procedure. Users running A20 (and later) software releases can upgrade using the standard INSTALL procedure. Special procedures must be followed to upgrade from releases earlier than A20. See the Softdoc for details.

FOX Ring Upgrades. All nodes on an existing FOX ring must be upgraded to the GUARDIAN 90 operating system simultaneously. If this is not possible for some reason, then the nodes on the ring that cannot be upgraded must be removed from the ring, and if communications with the other nodes are still required, they must be made via EXPAND.

DP2 File-conversion Utilities. With DP2, the format of structured files has changed but the disc format has not. This feature allows greater flexibility for TMF. Conversion utilities are provided.

Conversion

Programs that Modify the ENV Register. Any programs that ran on previous releases and modify the ENV register saved in the stack marker do not run with GUARDIAN 90. The CODEY utility on the GUARD2 DSV identifies any object files that require modification and recompilation of their source because they address L-1 directly.

It does not identify those programs that modify the saved ENV register via indirect, relocatable, or absolute addressing, nor does it identify those programs that use PUSH, SETL CODE sequences to effect the modification. CODEY has a complete "help" file, and unless restricted by the start-up message, it searches all volumes and subvolumes for offending code.

Changes to ERROR 70 and STARTUP Messages. The manner in which COMINT handles ERROR 70 and STARTUP messages has changed. Refer to the Softdoc for details. The old mechanism generates an error message, but still works and should not negatively affect users. In a future release, the old mechanism will be removed.

Product Dependencies. A table of product interdependencies can be found in the Softdoc Highlights file. TAL programs should source in the proper EXTDECS in order to run successfully. If necessary (i.e., if the code was last compiled with A03 software), the proper EXTDECS should be retrieved from backup tape.

COBOL Compiling Considerations. COBOL allows "compile-time binding." Basically, the compiler detects a CLIBOBJ file in its subvol and directs BINDER to resolve external references from the CLIBOBJ file before forming the run-time library. At run time the SYSTEMMONITOR resolves all other external references from LIB and system code.

Any old versions of modules in CLIBOBJ that violate GUARDIAN 90 addressing rules for extended code space prevent the successful execution of run-time libraries that were with that CLIBOBJ. Therefore, users should first recompile (if necessary), or remove CLIBOBJ and then recompile all the COBOL run-time libraries affected.

Privileged Processes that Access the DCT. Privileged processes that update the DCT do not run under GUARDIAN 90. A conversion checklist is available from your Tandem analyst for any user-written privileged processes. Privileged code should be carefully reviewed against this document.

More ENFORM Reserved Words. The list of ENFORM reserved words has grown, which may cause unpredictable results when ENFORM queries are recompiled. Record names, file-name components, and section names that are the same as any question-mark (?) command keyword cause a syntax error message to be issued for the ?ASSIGN, ?ATTACH, ?COMPILE, ?EXECUTE, ?OUT, ?RUN, ?SHOW, and ?SOURCE commands. This affects only these commands.

The only solution for this syntax error is to change the name of the record, file, or section. For records, opening another record as a copy of the conflicting one and then using the new name suffices.

FUP INFO Output Modified. FUP INFO output has been expanded. Any application that depends on the output format should be reviewed and changed if necessary. An interim version of FUP that runs under GUARDIAN 90 but produces its output in the old format is available from Tandem analysts. This version will be supported for a limited time only, to permit customers to make an orderly transition to the new, enhanced format.

Capacity Planning and Tuning

More Pages Locked. Tests have shown that the GUARDIAN 90 operating system locks between 20 and 35 pages more than A20 or A06 GUARDIAN (depending on the configuration). (Note that as these tests did not include DP2, they do not reflect the additional pages locked when it is used.)

XRAY PRES PAGES and PRES PAGESM. The fields PRES PAGES and PRES PAGESM are not recorded with this release of the XRAY performance measurement tool. They will be added in a future release.

Other Sources of B00 Information

More information about the products discussed in this article is available elsewhere in this issue, in the appropriate software manuals, and in the B00 Software Documentation (Softdoc) included on the Site Update Tape (SUT). Users needing even more information should contact their Tandem analysts.

Acknowledgments
The authors would like to thank the many software developers and product managers who contributed information to this article.

References

Chapman, B. and Zimmerman, J. 1985. The ENABLE Program Generator for Multifile Applications. *Tandem Systems Review*. vol. 1, no. 1. Tandem Computers Incorporated.

Pong, M. 1985. TMF Autorollback: A New Recovery Feature. *Tandem Systems Review*. vol. 1, no. 1. Tandem Computers Incorporated.

Tandem Software Documentation. 1985. B00 Site Update Tape (SUT). Tandem Computers Incorporated.

Tandem software manuals for the B00 software release. 1985. Tandem Computers Incorporated.

Kevin Coughlin has been a systems analyst in the Installability and Quality group for the past year. He joined Tandem's Software Education Department three years ago as an instructor for the GUARDIAN Internals II course, bringing with him 12 years of experience with the products of other mainframe vendors. Before joining Tandem, he was a service-bureau programmer, a vendor representative, and a field analyst focusing on price/performance issues.

Robert Montevaldo is the systems product manager responsible for the B00 software release. He joined Tandem in September 1980 as a product manager for Tandem's 6530 terminal and later became the product manager for the Nonstop TXP system. Robert has over 15 years of experience in the computer industry.

The new optional disc process for Tandem systems, DP2, comprises a complete redesign and reimplementation of the earlier Tandem disc process, DP1. The design changes were made to improve performance, throughput, recoverability, reliability, maintainability, and extensibility. The new disc process supports all disc controllers and discs for Tandem Nonstop II, NonStop TXP, and NonStop EXT systems.

DP2's job is to manage data and hardware within the disc subsystems. It implements ENSCRIBE file types, manages the space on discs, and manages the controllers and paths to the discs.

The new disc process is accompanied by appropriately modified and enhanced utilities and File System. As the file structure has been changed for DP2, a new utility is provided with it to convert files from DP1 to DP2 format.

Other architectural changes have been made to integrate the Transaction Monitoring Facility (TMF) and DP2 subsystems, resulting in increased performance in the TMF environment.

DP2 handles takeover, crash recovery, and rollforward consistently, in that it physically backs out incomplete requests. Checkpointing for DP2 is analogous to auditing to the backup process. While DP1 checkpoints enough information to the backup to carry forward the interrupted updates, DP2 checkpoints enough information to the backup process so that the backup can back out any interrupted multistep updates. This technique considerably reduces the number of messages sent to the backup process.

Performance and Throughput Improvement

DP2 allows multiple processes to control a disc volume. The new disc process consists of a group of single-threaded disc processes, each having its own data space. The number of disc processes in a group is specifiable at SYSGEN. (The default is three; the maximum is eight.)

DP2 allows for overlapping CPU set-up execution and better utilization of the disc cache in memory. A request to the disc that requires physical I/O can now overlap with a request that is satisfied by a cache hit. Parallel execution also provides the opportunity for sequencing I/O requests for seek optimization.

A new attribute for open files, BUFFERED, allows write requests for audited and nonaudited files to be buffered in the disc-process cache rather than being forced to disc at each request. At file creation the default is to buffer audited files and not to buffer nonaudited files. The BUFFERED option provides a substantial performance improvement for applications that write sequentially to a file and for those using audited files in general.

The BUFFERED attribute can be altered/examined via a SETMODE call. In addition, the BUFFERED attribute has a file-label default used to select buffered writes for a nonaudited file without requiring the addition of a SETMODE call to the applications that access the file.

Other performance improvement features include the following:

- The number and size of checkpoints per data-base update sequence have been reduced.
- Auditing is more efficient and reduces the number of writes at commit.
- Audit checkpoint records are optionally compressed, reducing CPU and memory cycles, as well as audit-trail consumption.
- At the disc-driver level, the next-queued EIO is issued when the completion of the last EIO is handled. This results in higher device utilization and faster response times.
- Independent read requests may be satisfied concurrently by parallel reads issued against a mirrored pair. (This is only possible if there is a path to each half of the mirrored pair through a separate controller.)
- The selection of the disc drive (primary or mirror) used to satisfy a read request is now based on minimum seek cost. This should minimize head movement and improve response times. (Rotational delays are not accounted for in this algorithm.)
- When 3107 controllers are used, DP2 can perform data transfers up to 30K bytes in length in a single disc-transfer operation. This facility is available only to utilities supplied by Tandem, such as BACKUP2, RESTORE2, FUP DUP, and TMF TAPE. (Note that when a logical volume is controlled by a pair of disc controllers, both controllers must be 3107s for DP2 to support this feature.)

Recoverability Improvement

The structural integrity of nonaudited key-sequenced files and the volume directory is protected by a volume-resident "undo area." Before a multiblock update (e.g., B-tree block split or collapse) is begun, a highly compacted encoding of the intended steps is written to the undo area in one I/O. Using this undo area, the disc process backs out any multiblock operations that were interrupted by a double failure (e.g., failure of the primary and backup disc-process CPUs). This processing takes place automatically when the volume is brought up. The structural integrity of audited files is protected by TMF.

When a DP2 disc-process group is idle for a sufficient period of time, a new algorithm is invoked to flush dirty cache buffers (including dirty File Control Blocks, or FCBs) until the next user request is received. This helps minimize the risk of data loss and/or the recovery effort required should a system failure occur. In addition, if the disc is still idle when all dirty cache buffers have been flushed, DP2 periodically updates the volume-label timestamp. This facilitates early detection of hardware failures as well as providing a more accurate timestamp for use in detecting inconsistent mirrors.

Other Features and Changes

The following features are also included in DP2:

- The maximum record size of the directory and key-sequenced files is limited only by the usable block size (whereas DP1 limits the maximum record size to half the usable block size).
- The directory, the free-space table, and nonpartitioned files are dynamically extendable. The directory can now have up to 987 extents. The maximum number of extents for a nonpartitioned file is dynamically alterable and is limited by the space remaining in the file label after the alternate-key-file information is recorded. This allows for over 900 extents in most instances.
- Block sizes are limited to power-of-two multiples of the sector size (512, 1024, 2048, or 4096 bytes). For example, 3K-byte blocks are not supported by DP2.
- Index and data blocks in a DP2 key-sequenced file must be the same size. This restriction was introduced to simplify cache management for key-sequenced files.

Finally, the following file attributes have been added:

- *BUFFERED*, as described above, controls the use of write-through cache on a per-OPEN basis. A file-label default may also be set for each file.
- *ACCESSTYPE* specifies the type of access the user intends to use, allowing the disc process to optimize cache management appropriately. ACCESSTYPE values include *system-managed* (the default), which allows the disc process to determine the best buffering techniques; *random-access*, which puts the blocks on an LRU (least recently used) list in the cache; *sequential-access*, which protects the disc process from filling up the cache with blocks that will not be used again; and *direct-I/O*, which allows exclusive or protected read-only openers to bypass the cache for unstructured accesses.
- *AUDITCOMPRESS* specifies for a particular file that DP2 is to produce a representation of the change to the record (rather than an image of the entire record). It then puts the smaller of the change information or the record image in the audit record or checkpoint message.

Compatibility and Conversion Between DP2 and DP1

The use of DP2 should not require changes to existing applications; however, several DP2 features (such as buffered cache, file ACCESSTYPE, and AUDITCOMPRESS) require user action to enable and/or tune them. This action is described in the *ENSCRIBE Programming Manual*.

The volume label, the directory, and the internal structure of structured files on a DP2 volume are all different from those on a DP1 volume. This requires that a DP1 volume be converted to the DP2 format. Programs that access structured and unstructured files in the ordinary ways should not encounter any incompatibilities due to DP2. On the other hand, programs that read structured files with unstructured access will find that the block structure has changed.

The new BACKUP2 and RESTORE2 utilities support automatic conversions so that a backup tape from a DP1 volume can be restored to a DP2 volume and vice versa. As RESTORE2 can read tapes created by BACKUP, it can be used to convert any file. The File Utility Program (FUP) DUP command also performs the necessary conversions, based on the source and destination volume types.

The File Conversion Program (FCP) is provided to convert multiple files and multiple volumes in parallel. It was designed to convert volumes faster than BACKUP2 and RESTORE2 by making the conversions disc to disc. When FCP is used, all files and volumes must be on the local node.

FCP converts to and from DP1 and DP2, allowing a site to return to a DP1 volume if necessary. The conversion takes about 90 minutes for 240M bytes of data. (The conversion may be slower if most of the files are key-sequenced. Also, this time does not include the time to do a PUP REVIVE on the other half of the mirror.) The converted file may require more space than the original as a result of the DP1 block size being changed to meet the DP2 size rules (power-of-two multiples) and bit-map blocks being added (used for free-space allocation in structured files).

Within a single node, a system can contain both DP1 and DP2 volumes, with certain restrictions. All volumes configured for a controller must be of one type or the other. Also, if a volume is configured with a primary and a backup controller, all volumes configured for both controllers must be of the same disc-process type.

All partitions of a file must be of the same disc-process type. If a file has alternate keys, the primary file and the alternate-key files must all be of the same disc-process type. Unstructured access to a structured DP2 file from another node that is running an earlier (pre-B00) version of the File System is prohibited.

DP2 unstructured files are transparently blocked with one of four valid DP2 block sizes (512, 1024, 2048, or 4096 bytes, the default). This transparent block size, known as BUFFERSIZE, is the transfer size used against an unstructured file.

While BUFFERSIZE does not change the maximum unstructured transfer (4096 bytes), multiple I/Os may be performed to satisfy a user's request, depending on the BUFFERSIZE chosen. For example, if the BUFFERSIZE were 512 bytes, and a request were made to read 4096 bytes, at least eight transfers (each 512 bytes long) would be made. More than eight transfers would occur, in this instance, if the requested transfer did not start on a BUFFERSIZE boundary.

DP2's performance with unstructured files is best when requested transfers begin on BUFFERSIZE boundaries and are integral multiples of the BUFFERSIZE.

DP2 and TMF

One of the design goals for DP2 was to provide support for quick crash-open file recovery after a system crash. This new form of crash recovery, called autorollback, is done "in place" and requires no file or audit dumps to be loaded, as is required by the rollforward process.

Autorollback is initiated automatically as part of TMF start-up processing. No tapes need be mounted; all data needed to perform autorollback is kept on disc. The granularity of autorollback is an audited volume.

TMF on-line dump and rollforward both run much faster on DP2 because the data transfer for DP2 is physical rather than logical, as it is for DP1. Since autorollback handles double CPU failure, on-line dumps and rollforward should only be required for recovery from media failure.

DP2 implements a buffered cache that obeys the write-ahead-audit protocol. Before writing a modified data block to disc, the cache manager makes sure that the audit records for the updates to that block have been written to the audit-file disc. DP2 generates physical audit records based on block changes rather than generating logical audit records as DP1 does. This allows DP2 autorollback to recover the physical integrity of files, while DP1 cannot recover files crashed in the middle of block splits.

DP2 implements a control-point mechanism that limits the length of the audit trail to be processed during the "redo" phase of autorollback. The control-point mechanism enables autorollback to find, in each audit trail, a redo start point, such that the changes for all the redo audit records that precede the redo start point are guaranteed to be reflected in the corresponding data-file blocks on disc at the time of the crash.

B00 TMF software supports either a DP1 or DP2 configuration, but not a mixed configuration. When TMF is used, to make the transition from DP1 to DP2 (or vice versa), all volumes to be configured for TMF (including audit-trail volumes) must be converted to the same type (DP1 or DP2) and an INITIALIZE TMF must be used to purge all configuration and catalog information.

When a DP2 system is configured for TMF, audit trails cannot be configured on volumes that are to contain audited files. That is, volumes can contain either audited files or audit trails, but not both. This restriction was made to eliminate possible deadlocks with the new write-ahead-audit protocol and to make the DP2 software more reliable.

The DP1 Monitor Audit Trail contains only commit records. Within a DP2 configuration, however, the TMF command processor (TMFCOM) allows users to direct data-audit records to a Master Audit Trail, which contains both commit and data-audit records. By allowing all or most of the audit to be directed to the Master Audit Trail, DP2 reduces the number of writes required for a transaction commit, thereby increasing transaction throughput.

A new TMFCOM ENABLE VOLUMES command has been added to allow the manual initiation of the DP2 autorollback process.

Utilities such as BACKUP2, RESTORE2, and REVIVE are much faster with DP2 and 3107 controllers. DP2 provides double fault tolerance by protecting the structural integrity of files. Finally, DP2 provides compatibility in a network since it can coexist with DP1 on a system or in a network.

More detailed information can be found in the accompanying DP2 articles, the Softdoc, and the appropriate software manuals.

Conclusion

DP2 provides substantial performance, reliability, and flexibility improvements over the current disc process. Particularly for TMF, performance and recovery speed are dramatically improved, allowing TMF to be used more effectively, with much larger data bases and higher-performance transaction-processing applications than before. DP2 also provides substantial performance improvements for applications that use buffered cache.

Acknowledgments
The authors would like to thank the DP2 software developers for information used in this article. Special thanks to Mike Kenrich, who wrote the external specification.

Kay Carlyle, formerly the DP2 Product Manager, has recently become the manager of a Software Development data-base group. She joined Tandem in June 1981 as a regional data-base specialist in Falls Church, Virginia, after spending seven years in product management and development. She has a B.A. in Mathematics from the University of Kansas and Masters Degrees in both Computer Science and Business Administration from Arizona State University.

Larry McGowan joined Tandem in November 1983 as the DP2 Software Development Manager. He has over 20 years experience in operating-system development and management. Larry has a B.S. in Mathematics from the University of Washington.

DP2's Efficient Use of Cache

With the B00 release, Disc Process 1 (DP1) has been enhanced to use buffered cache memory for audited files. In the new Disc Process 2 (DP2), the use and management of cache is even more efficient. Also, DP2 makes it easy to examine and change the configuration of cache to meet varying processing loads.

This article briefly describes:

- How DP1 and DP2 use cache.
- How DP2 cache can be configured.
- How DP2 handles files.
- How DP2 manages cache.
- How DP2 cache-performance is reported.

For more information on buffered cache and DP2 performance, see the accompanying article, "DP2 Performance," by Jim Enright.

How DP1 and DP2 Use Cache

Write-through and Buffered Cache

By caching the most frequently used data so that it can be quickly accessed, a disc process makes efficient use of memory. When the disc process uses *buffered* cache (as opposed to *write-through* cache), it uses memory even more efficiently.

Using write-through cache, a disc process stores blocks of data in cache (where they are available to satisfy read requests quickly) but immediately writes to disc data blocks in which any data has been changed by a write or update request.

Using buffered cache, a disc process can leave the changed (or dirtied[1]) data blocks in cache for a period of time. In this way, it may be able to schedule the physical write for a time when it is not processing user requests. By delaying the write to disc, it may also reduce the number of physical writes it has to perform, as several records in the same block may be changed before the write takes place.

Versions of DP1 released before the B00 software release use write-through cache only. B00 DP1 has been enhanced to use buffered cache for audited files. The new DP2 uses buffered cache for audited files, by default, and, in addition, it allows users to decide whether unaudited files are to use buffered cache or write-through cache, or to bypass cache entirely. Table 1 summarizes these differences.

DP1 and DP2 Cache-block Sizes

Disc-process cache is a pool of memory buffers containing images of disc blocks. DP1 (B00 and earlier versions) uses one large cache that contains data and index blocks of various sizes as well as audit-trail information. Valid DP1 block sizes are 512, 1024, 1536, 2048, 2560, 3072, 3584, and 4096 bytes.

DP2 uses a separate buffer for each size of cache block and a separate area for audit blocks. It uses only four cache-block sizes: 512 bytes (sector size), 1024 (1K) bytes, 2048 (2K) bytes, and 4096 (4K) bytes (maximum transfer size).

[1]With buffered cache, when a record in a cache block is added, deleted, or modified, the block is considered "dirty." It is "cleaned" or "flushed" when the disc process writes it to disc.

Table 1.
Comparison of cache use by three versions of the disc process.

	Pre-B00 DP1	B00 DP1	DP2
Cache buffer sizes (in bytes)	512, 1024, 1536, 2048, 2560, 3072, 3584, 4096	512, 1024, 1536, 2048, 2560, 3072, 3584, 4096	512, 1024, 2048, 4096
Internal space management	Single buffer area for cache and audit	Single buffer area for cache and audit	Separate buffer area for each size of cache block and separate area for audit blocks
Dynamic configuration	None	None	Performed with PUP SETCACHE
Performance analysis tools	XRAY	XRAY	XRAY PUP LISTCACHE
Access modes	Least-recently used (LRU)	LRU	LRU Sequential Direct I/O System-managed
Type of search made by internal cache-search algorithm	Binary	Binary	Hash
Ability to have buffered, unaudited files	None	None	Specified with FUP SET, ALTER BUFFERED
Types of cache write for unaudited files	Write-through	Write-through	Write-through or buffered
Types of cache write for audited files	Write-through	Buffered	Buffered (default) Write-through (not recommended)
Type of cache read	Buffered	Buffered	Buffered
Autorollback recovery	None	Part of Transaction Monitoring Facility (TMF)	Part of TMF
Maximum number of disc processes accessing a volume at one time	One	One	One to eight (default is three)
SYSGEN parameters	WRITETHRUCACHE CACHEPAGES	WRITETHRUCACHE CACHEPAGES	SYSTEM_VOLUME_CACHE_SIZES
Buffer-size selection for unstructured files	None	None	Specified with FUP SET, ALTER BUFFERSIZE

Hashing vs. Binary-search Algorithm

DP2 uses a hash-code access to determine which blocks are in cache. DP1 uses a less efficient binary search to determine this.

Configuring Cache

With DP1, users can specify cache size only at SYSGEN. To change cache size, they must perform a SYSGEN and cold load the system. Also, the only tool available with DP1 for examining cache performance is the XRAY performance measurement tool.

When DP2 Cache Can Be Configured

DP2 is much more versatile at cache configuration and management. Users can specify the size of each of the four cache buffers at these times:

- When labelling the volume, with the Peripheral Utility Program (PUP) command LABEL.
- When setting up the SYSGEN configuration file for system volumes, in the ALL-PROCESSORS section with the keyword SYSTEM_VOLUME_CACHE_SIZES.
- While the system is running, with the new PUP SETCACHE and LISTCACHE commands. (LISTCACHE is used to examine the cache configuration first.)

Any DP2 cache size not configured with SYSGEN or PUP LABEL defaults to 16 blocks.

A DP2 Configuration Example

The command syntax allows the user to specify cache size as the number of bytes or as a multiple of 1024 bytes, where .5K represents 512 (decimal) bytes, 1K represents 1024 bytes, 2K represents 2048 bytes, and 4K represents 4096 bytes.

To set cache on volume $DATA to 16 512-byte cache blocks and 128 4096-byte cache blocks, and leave the number of 1K-byte and 2K-byte cache blocks the same, the user would use the following commands:

```
PUP SETCACHE $DATA, .5K = 16,4K = 128
```

or

```
PUP SETCACHE $DATA, 512 = 16,4096 = 128
```

Internal Cache-setting Algorithm

While users can request a certain number of cache blocks, an internal cache-setting algorithm ensures that there is enough cache for DP2 to do certain critical operations and that space is not wasted. There must always be room for at least two cache blocks for some complicated operations, such as block splits of a key-sequenced file.

Since each process in the group can be working on a different block-split operation, each disc process in a disc-process group must have a minimum of two blocks for each of the four block sizes for which it has buffers. Thus, if there are three processes in the DP2 disc-process group, even if the user were to request only one cache block for each of the four block sizes, the internal algorithm would calculate a minimum of six blocks in cache for each of the four block sizes.

DP2 also allocates cache in multiples of pages, so all cache allocations are rounded up to the next whole page to avoid wasting memory. A request for six 512-byte blocks, for example, is rounded up to eight blocks or two pages. The maximum number of cache buffers that can be allocated for DP2 cache is 8191.

How DP2 Handles Files

As mentioned earlier, A06 DP1 uses write-through cache for both audited and unaudited files, while B00 DP1 uses it for unaudited files only. Also, B00 DP1 requires audited files to use buffered cache, while in DP2 this is the default setting of a user option.

For more information about B00 TMF and DP2, see the accompanying article, "Improvements in TMF," by Tony Lemberger.

When Does DP2 Write Buffered Blocks to Disc?

When DP2 updates a record in a buffered-cache block, it waits for one of the following conditions to occur before writing the block to disc:

- Any opener closes the file.
- A control point is processed. (For more information about control-point processing, see Pong, 1985.)
- The user requests a flush (SETMODE 95).
- No more blocks are free and the block is the least recently used.
- The disc process is idle and cache is cleaned.
- The user changes the cache configuration with the PUP SETCACHE command.

Unaudited, Buffered Files and CPU Failure

If a disc-process takeover occurs as the result of a CPU failure and the file has been opened with a sync depth of zero, or if a volume is brought down incorrectly, all of the unflushed updates made to an unaudited, buffered file are lost. The application is returned a new error code, FEDATALOSS (122), with the next operation on that file.

How DP2 Manages Cache

DP1 manages cache blocks on a least-recently used (LRU) basis. New access modes enable DP2 to use cache more effectively. The access modes can be defined either programmatically by users (with a call to SETMODE) or determined by the system, based on actual usage. The access modes are described below:

- *Sequential access* reuses the same cache buffer. It is useful for sequential-batch-processing applications in which the data in the buffer is not likely to be needed again.
- *Random access* uses an LRU algorithm for selecting cache buffers. It ensures that frequently accessed disc blocks are in cache. This is the same method used by DP1.

- *System-managed access* determines whether to use sequential or random access based on actual file usage.
- *Direct I/O access* bypasses cache.

How DP2 Cache Performance Is Reported

There are two ways to examine DP2 cache performance. The DISC and DISCOPEN commands can be used with XRAYSCAN, just as they're used for DP1. Also, the new PUP LISTCACHE command can be used. In general, XRAY reports rates and LISTCACHE reports percentages, although some of the same statistics are reported by both.

New XRAY Counters

New counters have been added to the XRAY DISCOPEN and DISC reports to track disc-cache usage. They were added for both DP1 and DP2, but as DP1 and DP2 manage cache differently, they use some of the counters differently. A complete description of the XRAY counters can be found in the *XRAY User's Manual*.

Sample DISC Report

A sample of the DISC report for a single DP2 disc process and a controller appears in Figure 1. Four sets of cache counters are provided, one for each DP2 cache-block group. (For example, CHIT0, or cache hit 0, is the average number of cache hits per second for the cache that contains 512-byte blocks.) The counters represented include:

Counter name	Counter
CHITn RATE	Number of cache hits per second
MISSn RATE	Number of cache misses per second
CFLTn RATE	Number of cache faults per second
ABFNn RATE	Number of audit-buffer forces per second
CBKSn ALLOC	Total number of cache blocks allocated
CBKSn DIRTY	Average number of dirty cache blocks
MAXSn DIRTY	Maximum number of dirty cache blocks

Some counter names end in a number (0-3) that indicates the block size of the cache-block group. This number is sometimes called the *cache ID*. The four cache IDs and the block sizes they represent are:

Cache ID	Block size of cache group
0	512
1	1024
2	2048
3	4096

Figure 1

```
$SYSTEM   LDEV 4   PID 0,7   CTL 1   UNIT 0   DP2   FEB 5 08:31:00 FOR 130

DISC   REQ    REQ     REQ    READ   WRITE  SEEK   DISC   READ   WRITE  SEEK
BUSY   QLEN   QLENM   RATE   BUSY   BUSY   BUSY   RATE   RATE   RATE   RATE
4.21%  .277#  3.00#   13.3   1.90%  1.59%  .718%  3.08   1.41   1.02   .648

BYTE   IBYTE  CBYTE   SWAP   DP1CB  DP1CB  DP1AB  DP1AB  CHIT0
RATE   RATE   RATE    RATE   INUSE  MAX    INUSE  MAX    RATE

MISS0  CFLT0  ABF0    CBKS0  CBKS0  MAX0   CHIT1  MISS1  CFLT1  ABF1   CBKS1
RATE   RATE   RATE    ALLOC  DIRTY  DIRTY  RATE   RATE   RATE   RATE   ALLOC
.015                  16.0#                .254   .030                  16.0#

CBKS1  MAX1   CHIT2   MISS2  CFL2   ABF2   CBKS2  CBKS2  MAX2
DIRTY  DIRTY  RATE    RATE   RATE   RATE   ALLOC  DIRTY  DIRTY
       1.00#  .324    1.84                 16.0#         1.00#

CHIT3  MISS3  CFLT3   ABF3   CBKS3  CBKS3  MAX3   CP     CPWRT  FREE   STALL
RATE   RATE   RATE    RATE   ALLOC  DIRTY  DIRTY  RATE   RATE   RATE   RATE
11.0   .640                 16.0#  .718#  2.00#                .030
```

Figure 1.
Sample output from the XRAY DISC report, generated with the TABLE OFF command. Blank fields indicate a value of zero. (The report has been edited to fit this space).

Figure 2

```
DATE: JAN 9 1984, 11:33
CACHE STATISTICS : $DATA
COUNTERS INITIALIZED : JAN 9 1984, 11:00
ELAPSED TIME :                    :33
CACHE BLOCK SIZE :        512      1024      2048      4096

BLOCKS REQUESTED:          40        40        40        40
BLOCKS ALLOCATED:          40        40        40        40
BLOCKS DIRTY :              0%        0%        0%        0%

CACHE READ HITS :          84%       88%        7%       70%
CACHE READ FAULTS :         0%        0%        0%        0%
CACHE READ MISSES :        16%       12%       93%       30%

CACHE WRITES :             26%       21%        3%       44%
CACHE WRITE HITS :         16%        0%        0%        0%

CACHE CALLS :             7458      1545      4435     62208
AUDIT FORCES:                0         0         0         0

BYTES ALLOCATED TO CACHE:   300K     WRITES/CONTROL POINT  :  0.00
```

Figure 2.
Sample output from the PUP LISTCACHE command, generated with the STAT option.

New PUP LISTCACHE Command

The PUP LISTCACHE command, with the STATISTICS option, shows more clearly some of the information reported by XRAY. A sample PUP LISTCACHE report is shown in Figure 2. The *GUARDIAN Operating System Utilities Reference Manual* describes the report in detail.

The CACHE WRITE HITS information reported by the PUP LISTCACHE command represents the percentage of time the block was found dirty in cache when a write was performed. This gives an indication of the number of writes saved by the use of buffered cache. The XRAY CWHIT RATE (cache-write hit rate) in the DISCOPEN report gives the same information on a per-file basis.

Two useful TMF statistics in the LISTCACHE report are the number of audit forces and the number of writes per control point. AUDIT FORCES represents the number of times an audit trail had to be written to permit a data block to be written. A high value usually indicates insufficient cache memory. A similar statistic, ABFn RATE, is in the XRAY DISC report.

WRITES/CONTROL POINT represents the number of writes forced by control-point processing. In a system that is properly tuned and not overloaded, this counter should be zero.

Conclusion

The use of buffered cache is an important part of both B00 DP1 and DP2. DP2 improves on this basic enhancement by managing cache much more efficiently than DP1. DP2 also allows users to configure cache use for a particular application's needs. Finally, DP2 makes more information about cache utilization available, allowing users to adjust the use and size of cache to accommodate a variety of work loads.

References

Pong, Michael. 1985. TMF Autorollback: A New Recovery Feature. *Tandem Systems Review*. vol. 1, no. 1. Tandem Computers Incorporated.

Acknowledgments

The original design for DP2 was created by Andrea Borr and Franco Putzolu. The author would like to thank all of the DP2 developers for helping him to understand DP2. Thanks also go to Dick Thomas and Jim Tate for their help in preparing this article.

Ted Schachter joined Tandem in May 1983. He is a member of the Field Productivity Programs group and is currently providing technical support for DP2 by teaching Beta classes. Before joining Tandem, Ted spent nine years as a real-time systems programmer. He has an M.A. in math from the University of Denver.

The design of Disc Process 2 (DP2) has introduced several changes in the structure and processing of key-sequenced files. This article explains some of the new features resulting from these changes. It is intended for the technical reader who is already familiar with the file structures of Disc Process 1 (DP1).

Differences Between DP1 and DP2

The main differences between DP1 and DP2 key-sequenced files are outlined below:

1. DP2 blocks have a new header that can be used to distinguish a key-sequenced block from a relative or entry-sequenced block. While the block header for DP1 is the same for all blocks, the DP2 header is different for each block type. It contains information identifying the type of block and its relative position in the file, and an internal timestamp for the last update, called the Volume Sequence Number (VSN). The header varies in size depending on the type of structured block.
2. DP2 data blocks have forward and backward pointers, while DP1 data blocks contain only forward pointers. (The File System in the B00 release, however, uses only forward pointers.)
3. DP2 block pointers contain relative sector numbers (RSNs) that are 3 bytes long in place of the 4-byte relative-byte addresses (RBAs) used by DP1.
4. DP2 data records can be larger than one-half the block size. DP1 requires that a block be able to contain at least two records, but DP2 uses a new block-split algorithm that makes it possible to relax this restriction.
5. DP2 locates free blocks by using bit maps rather than by chaining blocks in a "free list" as DP1 does. DP2 can detect bit-map errors and correct them, while DP1 cannot repair the free-list chain when it breaks.
6. To simplify space allocation within files, DP2 requires index and data blocks in the same file to be of the same size, while DP1 permits them to be of different sizes.
7. There are no forward (horizontal) pointers in DP2 index blocks. These pointers are present in DP1, but are never used by it or by the File System.

Consistency

Consistency refers to the state of a data base after a failure. There are three kinds of consistency: structural, file, and data-base. The DP2 design, together with the closely integrated features provided by TMF, ensures all three.

Structural Consistency

Structural consistency applies to key-sequenced files that can break during the addition or deletion of a record as the result of a block split or block collapse. A key-sequenced file has structural consistency if the index pointers point to the correct blocks and if the forward and backward pointers in data blocks are consistent. If a failure occurs during a multiblock update, DP2 undoes any partial changes that have been made.

File Consistency

If a file has only structural consistency, its internal pointers are always consistent, but updates to the data may be lost. *File consistency* means that no updates are lost.

Data-base Consistency

File consistency and structural consistency may exist, but *data-base consistency* may be lacking because not all the file updates for a transaction are completed. A transaction can update multiple blocks in multiple files. TMF provides data-base consistency for audited files.

Block Structure of Files

DP1 key-sequenced files consist of index and data blocks that are either in use or are on a free list. These blocks can be of different sizes, but they have the same block format.

DP2 key-sequenced files consist of index blocks, bit-map blocks, data blocks, and free blocks. The headers of the different blocks are illustrated in Figure 1.

VSNs are internal timestamps used by DP2 to help implement the write-ahead-audit protocol, autorollback recovery, and rollforward recovery used by the Transaction Monitoring Facility (TMF). DP2 also uses VSNs when processing records in the "undo" area, an area used to ensure the structural consistency of unaudited DP2 key-sequenced files.

Block Splits

Block splits occur when the disc process tries to add a record to a block when there is not enough room. In the most complex case, the insertion of a new data record can cause the split of a data block, index blocks, and the root block. The DP1 algorithm is implemented so that splits are handled recursively, starting at the data level.

DP1 splits data blocks and then splits index blocks when necessary. DP2, on the other hand, looks ahead to see if an index-block split will be necessary and divides a complex split operation into a sequence of simpler split operations.

DP1 requires records in key-sequenced files to be less than one-half the size of the block so that the insertion of a record results in no more than two data blocks after the record is inserted (via a block split). DP2 relaxes this restriction: insertion of a large record can result in three data blocks after the record is inserted. Within DP2, all operations on a key-sequenced file are classified as either simple or complex.

Simple vs. Complex Split Operations

Simple operations are reads or writes on one disc block. Complex operations are write operations that modify two or more blocks, such as block splits, block collapses, insertions of the first record in an empty file, and deletions of the last remaining record in a file.

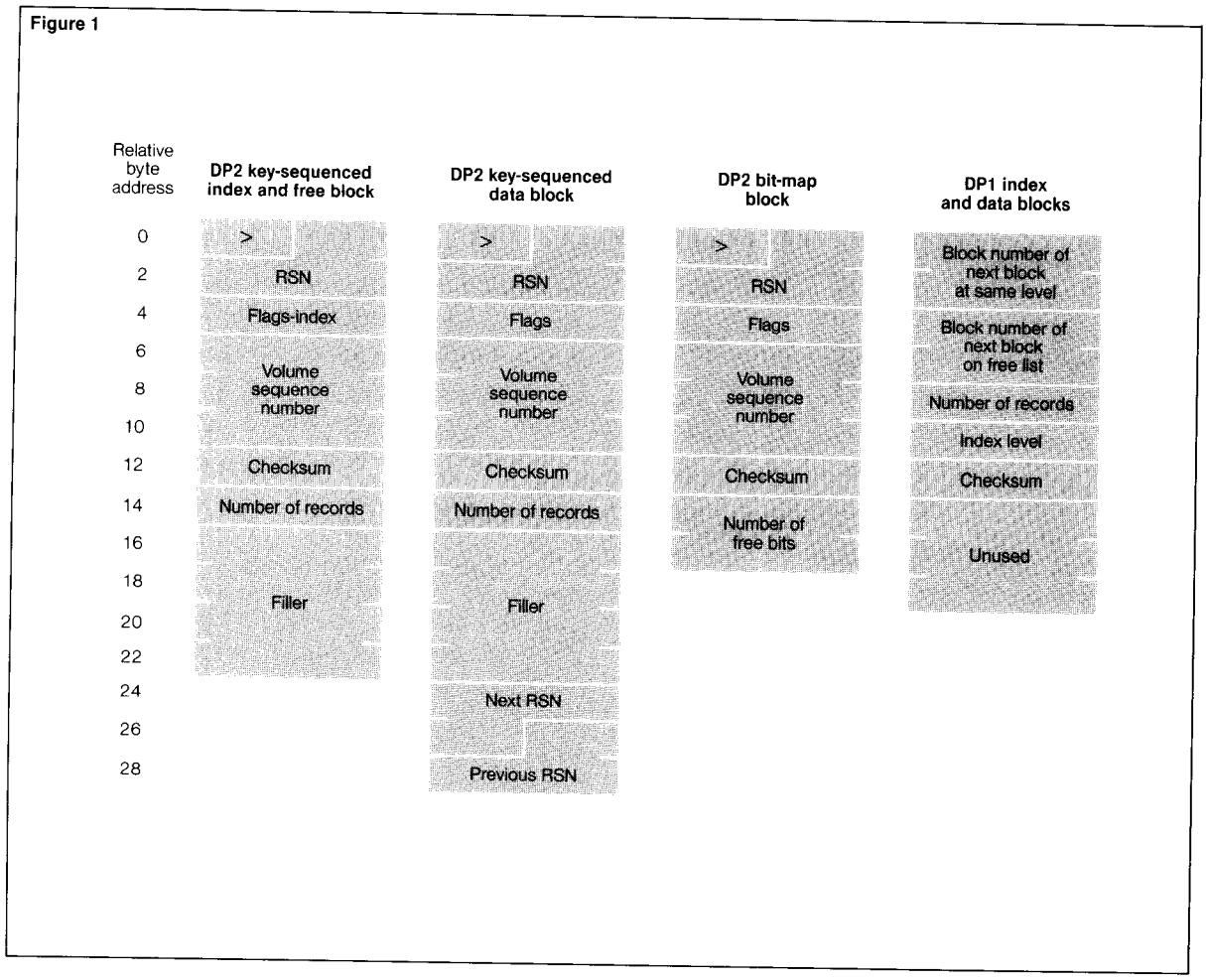


Figure 1.
The block headers for DP1 and DP2 key-sequenced files. The first 14 bytes of the DP2 block header are common to relative, entry-sequenced, and key-sequenced files. In the first byte of the header is an "eye catcher": a "greater than" symbol (>). The relative sector number (RSN) in the next 3 bytes numbers all the blocks in a file. The "Flags" word contains information about the type of file, type of block, and level of index.

A *block split* occurs when an attempt to insert or update a record in a data or index block requires the block to be split into multiple blocks because the inserted/updated record does not fit in the original block. A *block collapse* occurs when the last record of a data or index block is deleted, causing the block to be freed. The *insertion of the first record* in an empty file causes the formatting of three blocks (bit map, index root, and data block). The *deletion of the last and only record* causes the resetting of the end-of-file (EOF) pointer and the number-of-levels fields in the file label.

Block-split Decomposition

The DP2 block-split algorithm breaks up a complex split operation into a number of simpler operations. In general, a DP2 split is implemented as a sequence of internal operations, each of which (1) creates only one new block (affects only one level) and (2) ensures that consistency is maintained after the operation is completed. The DP2 split algorithm is more costly in disc activity than the DP1 recursive method *only* if the split of an index block occurs. The DP2 split algorithm, together with the recovery procedures provided by the undo area, auditing, and checkpointing, ensure that a file always has structural consistency.

Figure 2.
The structure of a file before a block split.

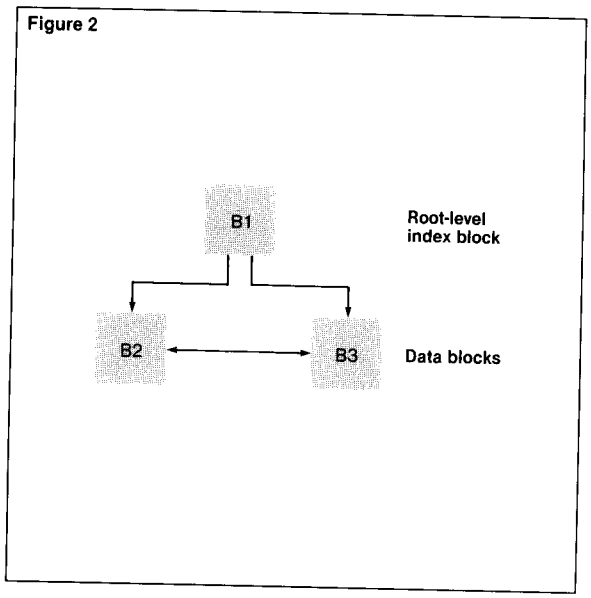
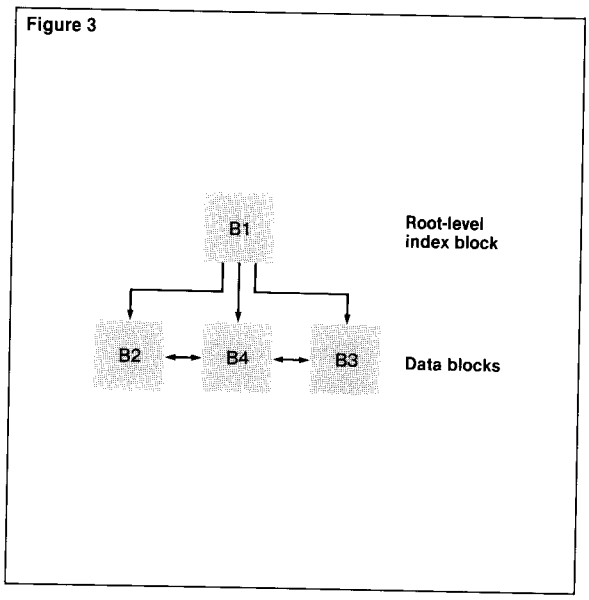


Figure 3.
The structure of the file in Figure 2 after the split of block B2 into blocks B2 and B4.



Data-block Split

Figure 2 depicts the structure of a sample DP2 file, and Figure 3 shows the file's structure after a block split. The steps involved in splitting the block are outlined below.

If there is not enough room to insert record R into block B2, a new block, B4, must be created. Half of the records are moved from B2 to B4, and then record R is inserted into block B2. In this example, it is assumed that the record is inserted into block B2 rather than block B4 (depending on the primary key, however, the record could have been inserted into either).

Before the split begins, it is necessary to verify that there is room in the parent index block for another record. If there is no room, one (or more) index-block splits must be done before proceeding.

Free Blocks and Bit Maps

The disc process finds a free block by scanning the bit maps that are stored in the bit-map blocks. DP2 attempts to find space for a new block "near" the block being split by beginning the bit-map search "near" the old block. Each bit in the bit map represents one block in the file. If the bit is set, the block is in use.

When the disc process finds a bit in the bit map that is not set, it reads the corresponding block header, whose status information indicates whether or not the block is indeed free. If the block is free, the disc process proceeds with the block split. If the block is not free, the disc process correctly sets the bit in the bit map and the search continues with the block indicated by the next free bit.

This bit-map search replaces the linked list of free blocks used by DP1. If the linked list breaks, DP1 cannot add new blocks to the file. If there are bits in the DP2 bit-map block that should be set but are not, however, DP2 is able to correct the situation as it is detected.

The Steps in Splitting a Block

The following steps are involved in splitting block B2:

1. Allocate block B4; move half of the records in B2 to B4.
2. Update the back pointer in B3 so that it points to B4 (instead of B2).
3. Insert a pointer record in the parent index block, B1.
4. Insert record R in B2, update the number of records in the header, and update the forward pointer to point to B4 (rather than B3).

Three-way Split

In DP2, a data block may have to be split into three blocks when a long record is inserted or updated and its position is not at the beginning or end of the block. For example, assume that the block size is 512 bytes

and a data block contains two records, R1 and R2, each 200 bytes long. The inserted record R has a length of 400 bytes, and its point of insertion is between R1 and R2. The three-way split is implemented as a sequence of two-way split operations, each of which preserves file consistency. The steps involved are:

1. Split the block into two blocks, at the point at which the new record is to be inserted or in front of the record that is to be updated.
2. Now the position of the inserted or updated record is at the beginning or end of the block, so only a two-way split is needed to complete the operation.

Index-block Split

An index-block split is similar to a data-block split except that index blocks, unlike data blocks, do not have horizontal pointers. Figure 4 shows the structure of a file before an index-block split, and Figure 5 shows the structure of the file afterward.

For record R to be inserted into B4, B4 must be split, requiring that a new index record be inserted into B2. Before B4 is split, B2 must also be split so that there is enough room to insert the new index record. To have enough room for the new index record, block B8 must be added and half of the records must be moved from B2 to B8.

The following steps are involved in splitting block B2.

1. Allocate block B8; move half the records in B2 to B8. (Because this is an index block, there is no back pointer.)
2. Insert into the parent index block, B1, a record pointing to B8.
3. Truncate B2. (Change the number of records in the block.)
4. Insert record R in block B4, causing a block split, as described above.

DP2 Block-split Algorithm

The undo area is a small, preallocated area on a volume that is reusable for every request resulting in a multiblock update. The block-split algorithm used by DP2 allows it to write the undo area to ensure the consistency of unaudited, key-sequenced files and the directory.

Figure 4

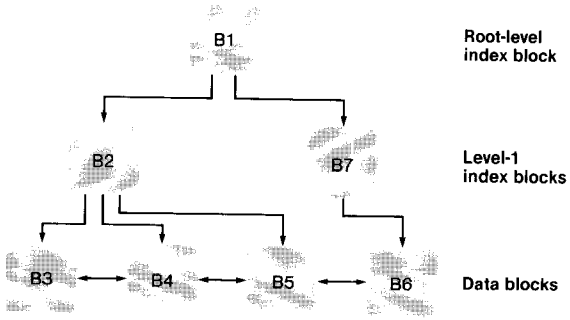


Figure 5

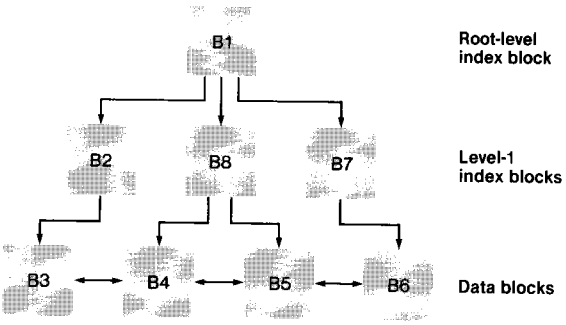


Figure 4. *The structure of a file that has two index levels.*

Figure 5. *The structure of the file in Figure 4 after the split of index block B2 into blocks B2 and B8.*

Before a multiblock update to an unaudited, key-sequenced file or the directory, DP2 writes into the undo area the information needed to undo the operation, in case the operation fails to complete. File consistency exists only if the operation is write-through or audited; otherwise, only structural consistency exists, and some data may be lost.

Table 1.
Source and level of consistency for DP2 files following (a) a takeover after the failure of the primary disc process' CPU and (b) a recovery from a multiple-CPU failure.

	(a) Consistency following takeover after failure of primary disc process' CPU		(b) Consistency following recovery after multiple-CPU failure	
	Source	**Level**	**Source**	**Level**
Audited files	Auditing	Data-base	Auditing	Data-base
Unaudited, buffered[1] files with sync depth > 0	Checkpointing	File	Undo area	Structural[2]
Unaudited, buffered[2] files with sync depth = 0	Undo area	Structural	Undo area	Structural
Unaudited[1] write-through files with sync depth > 0	Checkpointing	File	Undo area	File
Unaudited write-through[1,2] files with sync depth = 0	Undo area	File	Undo area	File
Directory write-through files with sync depth > 0	Undo area	File	Undo area	File

[1]For unaudited key-sequenced files, DP2 always uses file-label refresh even if the user specifies no refresh with the FUP SET command. This is to ensure that there is structural consistency following a system crash.

[2]For unaudited buffered files, unflushed updates may be lost. If a CPU failure causes the data loss, the application is returned FEDATALOSS (122). If FEPATHDOWN (201) is returned, the user is responsible for retrying the last operation.

For unaudited files with a sync depth greater than zero, the information in the undo area is checkpointed to the backup. (Setting the sync depth to zero is a way of turning off checkpointing in unaudited files.) When a takeover occurs, the backup disc process uses this information to undo incomplete multiblock updates and restore structural consistency.

TMF also takes advantage of the DP2 block-split algorithm when the primary disc process sends physical undo information to the audit disc process for audited files. The audit disc process writes this information to the audit trail for use by autorollback and rollforward recovery.

The DP1 audit trail contains only logical information, requiring a rollforward to recover from an incomplete multiblock update. The DP2 audit trail contains the physical undo information required to restore the structural consistency of a file; DP2 autorollback uses this physical information to restore the file.

Table 1 summarizes DP2 file consistency after (a) a takeover resulting from a failure of the primary disc process' CPU and (b) a recovery from a multiple-CPU failure.

Conclusion

DP2 has introduced changes in the structures and processing of key-sequenced files. The new key-sequenced block format, together with the block-split algorithm and the undo area, increase the reliability and functionality of key-sequenced files.

Acknowledgments
The author would like to thank all the DP2 developers for their help. Franco Putzolu was particularly helpful in explaining key-sequenced files. The reviewers of this article, especially Jim Tate, also deserve thanks for their comments and suggestions, as does Dick Thomas for the final revision.

Ted Schachter wrote this article, the article entitled, "DP2's Efficient Use of Cache," and the chart entitled, "A Comparison of the B00 DP1 and DP2 Disc Processes."

The B00 software release introduces many changes in the Transaction Monitoring Facility (TMF). This article provides an overview of the most significant ones, including:

- *Autorollback recovery*. This mechanism brings a data base back to a consistent state much more rapidly than the existing rollforward recovery mechanism (and without operator intervention). Autorollback recovery is possible because TMF now guarantees that audit information needed to perform recovery operations resides on disc.
- *Disc Process 2 (DP2) Implementation*. The new, high-performance disc process is designed to streamline accesses to disc volumes. The impact of TMF on disc-process performance has been reduced significantly as a result of this implementation.
- *Disc-volume flexibility*. TMF now provides users with much more flexibility in starting disc volumes. Selected volumes can be started initially and others activated later. Also, it is now possible to specify that a volume be available for nonaudited processing but unavailable for audited processing.
- *Improved crash protection for TMF*. With the B00 release, even the loss of the CPUs in which a disc-process pair resides does not generally result in a TMF crash.
- *Restartable TMF processes*. The loss of the backout or auditdump process no longer results in a TMF shutdown, because now these processes are automatically restarted when necessary.

This article provides information on all of the above. Refer to the TMF Softdoc for more details and for information on other changes to TMF software.

Autorollback Recovery

Before the B00 release, there was only one way of recovering a data base that had been contaminated as a result of a catastrophic failure: rollforward recovery. B00 TMF offers an additional method of recovery, autorollback, which is far more efficient. Rollforward recovery is now required only in the case of media failures that actually destroy the data base.

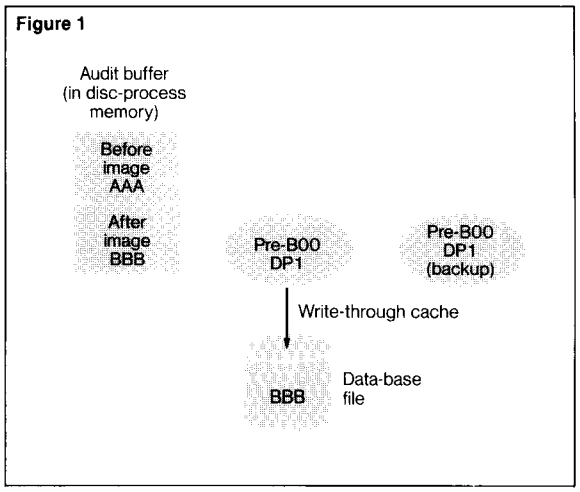


Figure 1.
Before the B00 software release, a change to an audited file resulted in the data-base change being written immediately to disc (write-through-cache). The audit that was generated as a result of this change was buffered in the memory of the disc process (audit buffer).

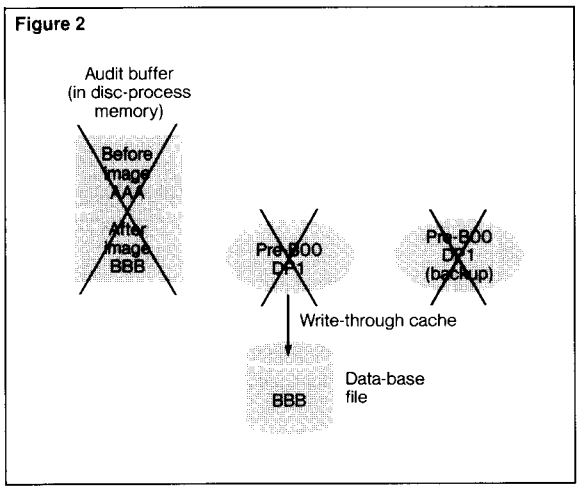


Figure 2.
Before B00, the failure of a disc-process pair before the end of a transaction destroyed the audit that was generated by the change to the audited file. To restore data-base consistency, the change to the data base had to be backed out. Because the audit was lost as a result of the failure, rollforward recovery was required.

To understand why autorollback has been so long in coming, it is important to note that Disc Process 1 (DP1) implemented a write-through-cache algorithm until the B00 release. With this algorithm, every file change was immediately written to disc, and, when updates were made to TMF audited files, the disc process buffered the before/after images of all changes made. (This audit information was written to disc during ENDTRANSACTION processing.)

Figure 1 shows the contents of an audit buffer generated by a change to an audited file. The original record in the data-base file was AAA, and it was changed to BBB. As explained above, A06 DP1 writes the change to the data-base file and buffers the before and after images for the change. Thus, if the disc-process pair controlling the data volume fails before the transaction ends, data-base changes that have been made (as part of the uncompleted transaction) have to be backed out to restore the logical consistency of the data base.

As shown in Figure 2, however, the audit information needed to back out the changes to the data base has been lost as a result of the failure of the disc-process pair. Thus, there is no recourse but to restore a known good copy of the data base and audit trails (from dump tapes) and use rollforward to recover.

If all the audit information needed to recover the data base were on disc, a fast recovery mechanism could be implemented that would make use of the existing data base. If the disc process could guarantee that all audit information generated while servicing audited requests was indeed recorded on disc, that audit information could be used to redo completed transactions. Uncommitted transactions, however, might or might not have audit information recorded in the audit trail.

If audit information were recorded on disc for these transactions, that data could be used to undo their effects in the event of a failure. The only way to ensure that such information is available on disc in the event of a failure is to require that it be written to disc before any data-base update associated with it. The scheme that ensures that audit information is on disc before the associated data-base updates is referred to as a *write-ahead-audit protocol.*

The most straightforward way of implementing a write-ahead-audit protocol is to force a write to the audit trail before the physical I/O to the data base is performed. However, the performance implications of an extra disc I/O for each write to the data base make this approach unworkable. Clearly, another solution must be found.

B00 TMF uses buffered cache to implement a write-ahead-audit protocol. Unless data blocks are buffered, audit cannot be buffered. Thus, when an application issues a logical I/O, the data is not immediately written to disc. It is buffered in cache, and it remains there until after the audit information reaches disc. This approach to implementing a write-ahead-audit protocol is attractive because it allows applications to access updated blocks in cache, resulting in a savings in physical I/O.

As a result of this change, a fast autorollback recovery mechanism can be implemented efficiently. Autorollback recovers the data base first by reading the audit trail forward (reapplying completed transactions) and then by reading the audit trail in reverse (backing out data-base updates associated with uncompleted transactions).

To perform an autorollback recovery, users start up TMF after the crash, and then TMF's Transaction Monitor Process (TMP) determines whether a recovery is needed. The existing copy of the data base is used, and all necessary audit-trail files are on disc. This alleviates the need for operators to mount tapes.

As explained above, autorollback reads the audit trail forward to the end and reapplies all changes made to the data base. This is called the *redo phase*. After all changes have been reapplied, any data-base changes made by transactions that were uncommitted at the time of the failure are backed out. This phase is called the *undo phase*.

During the undo phase, autorollback reads the audit trail in reverse, starting at the end. This phase ends when all changes made by transactions that were active at the time of the failure have been backed out. At the end of the undo phase, the data base has been restored to a logically consistent state, and application processing can begin. For more information about autorollback recovery, see Pong, 1985.

TMF is now capable of supporting either DP1 volumes or DP2 volumes, and TMF autorollback recovery is available for either DP1 or DP2.[1] However, the format of the audit trails, the format of many internal messages, and the behavior of TMF differs significantly for DP1 and DP2. Thus, there are many incompatibilities that preclude the audited operation of a DP1 volume and a DP2 volume at the same time. Users must specify which disc process is to be used for auditing when they initialize and configure TMF.

[1]In the rest of this article, unless specifically labelled as B00 or A06, the term *DP1* refers to both releases of DP1.

DP2 Implementation

DP2 is a new, high-performance disc process for Tandem systems. In this section, some of the differences in using TMF with DP1 and DP2 are discussed and then the performance improvements offered by DP2 are described.

DP2 vs. DP1

As explained above, there are a number of differences between DP1 and DP2 audit trails, and TMF itself operates differently in a DP2 environment.

Volumes Containing Audit Trails. Any DP2 disc volume may hold unaudited files. However, when a DP2 volume holds an audit trail, it may not hold audited files. This restriction stems from the very active role that DP2 plays in the movement of audit information to disc. In DP1, the audit trail is a simple unstructured file into which each of the auditing disc processes performs independent writes. In DP2, the disc process that owns the audit trail acts as a collector, buffering the contributions of many other auditing disc processes. These contributions are then written into the audit trail together. A single DP2 process does not act both as a collector and an audit generator.

Audit Trails. DP1 audit trails are different from DP2 audit trails. With DP1, TMF makes use of the Monitor and Disc Process Audit Trails. The Monitor Audit Trail is used to hold the commit and abort records for completed transactions. The Disc Process Audit Trail holds the before and after images of changes made to audited files.

DP2 makes use of the Master Audit Trail, which holds commit and abort records as well as the before and after images of audited changes.

Performance Improvements

DP2 is much faster than DP1. This is especially true when the files being updated are TMF files. A number of changes have been made in checkpointing and auditing strategies to achieve this. Also, the design of DP2 allows it to perform physical disc I/O while at the same time accessing data blocks in cache or performing other I/O set-up operations.

Reduced Checkpointing. DP1 incorporates the idea that any operation in progress at the time of the failure of the primary disc process should be carried forward to completion by the backup disc process. This idea leads inevitably to incremental checkpointing. That is, the primary disc process informs its backup of every step of an operation that changes a file. As a result, a single READUPDATELOCK-WRITEUPDATEUNLOCK sequence generates five checkpoint messages. If a failure of the primary disc process occurs, the backup has all the information it needs to continue the current operations to completion.

DP2 uses an entirely different approach. All operations in progress at the time of a failure are backed out (including TMF transactions being processed by the primary disc process at the time of its failure). Thus, the DP2 design allows for deferred checkpointing. This reduces the total number of checkpoints needed per READUPDATELOCK-WRITEUPDATEUNLOCK sequence to a maximum of two messages for an entire transaction, regardless of the number of physical I/Os or lock requests made by the transaction. The primary disc process needs only to inform the backup of the new transaction initially. If a failure of the primary disc process occurs, the backup takes over and knows which transactions are active. Those transactions can then be aborted and backed out.

Because the primary disc process does not tell its backup about all the work it is performing on audited files, DP2 spends much less time checkpointing. Only two checkpoints per transaction are required. Also, if the CPU in which the primary disc process is running fails, the backup disc process has no idea what the primary was doing.

The primary may have processed some requests on behalf of these transactions and promised to make some changes to the data base, which actually got no further than the primary's buffers. Those buffers are now gone, and consequently, the backup disc process cannot fulfill the failed primary's promises (because it doesn't know what they were). Thus, it aborts every active transaction that the primary had worked on. This ensures that the failed primary's promises don't harm the data base.

DP2's higher performance in an on-line environment has a cost. Recovering from CPU failures takes a little longer.

Overlapped Disc and CPU Processing. DP2 has been implemented in *process groups*, which consist of from one to eight disc processes, all of which work together to control a single disc volume. While one disc process in the group is in the process of performing physical I/O, another can be servicing a request by accessing data already in cache or performing other I/O set-up operations. This is a major performance advantage.

Also, DP2's use of buffered cache helps to reduce the total number of physical I/Os required. (For more detail on the advantages of using buffered cache, see the accompanying article, "DP2's Efficient Use of Cache," by Ted Schachter.)

Fewer Audit-trail I/Os. With DP1, whenever a transaction is in Phase One of abort or commit processing, each disc process that participated in the transaction must flush its audit to its associated Disc Process Audit Trail. This flush results in a physical I/O to the audit-trail. The DP1 disc process controlling the audit-trail volume has no knowledge of TMF audit trails.

Thus, the requests of each disc process are treated by the audit-trail disc process simply as a write to an unstructured file; there is one physical I/O for each disc process that participated in the transaction. In addition to the I/O performed to the Disc Process Audit Trail, a physical I/O is performed to the Monitor Audit Trail to write either the commit or abort record for every transaction.

With DP2, the disc process controlling the Master Audit Trail volume is aware of the audit-trail files. This doesn't change the requirement that each disc process that participated in a transaction flush its audit at Phase One abort or commit. However, it does mean that the Master Audit Trail disc process does not have to write each audit buffer to the audit trail immediately. Instead, it can buffer the audit from all the disc processes, and then, when the commit or abort record is written to the Master Audit Trail, append that record to the audit buffers already present in memory. Thus, all the audit information can be written with one physical I/O.

With DP2, the use of the Master Audit Trail to hold both before and after images as well as commit and abort records eliminates a great deal of extra physical I/O to the Monitor Audit Trail performed by DP1. Also, as just explained, the knowledge possessed by the Master Audit Trail disc process of the audit-trail files allows the buffering of audit information sent by many different disc processes and thus reduces dramatically the number of physical I/Os required.

Disc Volume Flexibility

With A06 TMF, a disc volume was considered either "up" for processing or "down." There was no intermediate state (e.g., that of being active for nonaudited processing and, at the same time, unavailable for audited work).

With B00, a disc volume can be in one of three different states:

- *Down* (e.g., in response to an operator's PUP DOWN command).
- *Up* (i.e., accessible for nonaudited processing but not for audited processing).
- *Up for TMF* (i.e., TMF has been started and audited work can be performed).

Changes to TMF were required to support the new intermediate state. The following sections examine this new flexibility.

Selected Volumes Can Be Started

New syntax has been added to the START TMF command, allowing the user to specify the set of disc volumes that are to be started. Only those volumes specified will be able to perform audited work. This means that users also need a mechanism that allows them to start separately any volumes that were omitted from the set of volumes specified in the START TMF command.

ENABLE VOLUMES Command

To fulfill the need described above, an ENABLE VOLUMES command has been added to TMFCOM. It allows users to add one or more volumes to the set of volumes that are up for TMF. The ENABLE VOLUMES command can be used to bring up any of the following:

- Volumes that were not requested at START TMF.
- Volumes whose CPUs were down at START TMF time.
- Volumes that went down while TMF was running.
- Volumes that failed to start earlier.

ENABLE VOLUMES initiates autorollback for any of the volumes that may require recovery. If autorollback is incapable of recovering the volume, a message is displayed on the operator's console. The operator should then rectify the problem and issue another ENABLE VOLUMES command.

The ENABLE VOLUMES command includes syntax to specify a set of volumes that should be enabled. This syntax is identical to that used with the START TMF command. Only the disc volumes specified in the command are enabled.

Down Volume Reintegration

If a disc volume goes down while it is up for TMF, it is possible to reintegrate the volume into TMF. This is accomplished by including the volume in the specified set of volumes for an ENABLE VOLUMES command.

When reintegrating such a volume, TMF examines all of the transactions that have worked on the volume. It must guarantee that all of these transactions are ended or aborted before the volume is enabled (to ensure that transaction backout and auto-rollback don't interfere with each other by working on the same records while undoing a transaction).

If an ENABLE VOLUMES command is received, the disc process looks to see if any old transactions (left over from before it went down) are still around. If any still exist, the disc process rejects the attempt to bring the volume up for TMF, and an Error 2 is returned. When these transactions are gone, the ENABLE VOLUMES command can be retried.

Adding New Volumes

With A06 TMF, the addition of new disc volumes to the system requires an initialization of TMF.

With B00, TMF makes use of a new file called VOLINFO. This file contains a list of all the volumes that are known to TMF. Should a new volume be added, the next START TMF command simply appends the new volume's name to the end of the VOLINFO file. The volume will then be enabled for TMF processing.

There are two restrictions on volumes that are not present when TMF is started:

- First, the volume must have been present, with that name, at some execution of START TMF. (This puts the name into the VOLINFO file.)
- Second, the volume's LDEV number must not change during a TMF session. For example, if $DATA was LDEV 6 and it has to be changed to LDEV 7, TMF must be stopped and then started again.

TMF Crashes

When a disc process does some work on behalf of a transaction, it must mark that transaction in such a way as to indicate that the transaction has audit information in the disc process' buffers. This marking is then used to determine which disc processes must flush their audit to end the transaction successfully.

TMF records which disc processes have audit to flush within the two CPUs that contain the disc process, eliminating the need to tell other CPUs.

When the two CPUs in which a disc-process pair are running fail (or, if only one was up, when it fails), TMF has no idea which transactions may have audit in the now lost disc-process buffers. Before B00, TMF simply gave up under these conditions, declaring a TMF crash. This forced the user to cold load the system and perform a rollforward.

B00 TMF still doesn't know which transactions might be in trouble in this situation, but if it can abort every transaction that might possibly have audit in the lost disc process' buffers, it can still ensure the correctness of the data base.

When TMF wants to end a transaction, each of the disc processes flushes the audit for that transaction to disc. When the last unflushed process in a particular CPU has flushed its audit to disc, a message that the disc process' CPU has finished flushing is sent to the BEGINTRANSACTION CPU.

If both CPUs in which a disc-process pair is running are lost, TMF needs to know whether any audit information has been lost. If the last CPU to fail has sent to the BEGINTRANSACTION CPU the message that the disc process' CPU is flushed, no audit has been lost. If that message hasn't been received, the transaction must be aborted because some audit may have been lost.

It should be noted that flushing occurs only while the transaction is ending (or aborting). Active transactions are not flushed; therefore, when TMF detects that the second CPU of a disc-process pair has failed, it marks every transaction on the system (with very few exceptions) as aborting.

Exception 1: DP2's Master Audit Trail

Above, the flushing of the audit to disc and TMF's management of the knowledge of who flushed was discussed. For DP1, the audit must be on disc before the disc is considered flushed. For DP2, however, any audit is considered flushed when it has been sent to the DP2 disc process that owns the Master Audit Trail.

Now, suppose that some audit has been sent to this DP2 collector. Normally, the commit record must be written to the end of the audit buffer. It is then written to disc. If the DP2 collector's CPUs are lost, whether the data audit or commit record reached the disc is unknown.

This poses a problem. If the data audit was lost, a commit record must not now be written. If the commit record has arrived at disc, the transaction must not now be aborted. The only way out of this dilemma is to look into the audit trail. If the commit record did reach the disc, the transaction committed. If not, the transaction aborted.

Since A06 TMF has no mechanism for handling this, when the dilemma arises, the B00 release forces a TMF crash, requiring a cold load and the execution of autorollback on every volume. Hence, on DP2 systems only, the loss of the CPU pair containing the Master Audit Trail's disc process causes a TMF crash.

Exception 2: The TMP

The TMP maintains many coordination functions, such as audit-trail management and TMF process management. Because of this, the loss of the TMP pair's CPUs causes a TMF crash.

This restriction is not serious. The TMP pair is placed by SYSGEN into the same CPUs as the operator process. While it is possible to SYSGEN a system in which the operator process is not placed in the same pair of CPUs as $SYSTEM, this configuration would make cold loading difficult and, hence, is not a serious option. This implies that the TMP resides in the same pair of CPUs as $SYSTEM.

Restartable TMF Processes

Before the B00 software release, the TMP was the custodian of three processes: backout, TMFTAPE, and catalog. The B00 release has added autorollback to this list.

The A06 release had backout, TMFTAPE, and catalog functioning as process pairs. Backout and TMFTAPE would sometimes fail, which would cause TMF to attempt to stop TMF because the processes were gone.

With B00, backout, TMFTAPE, and autorollback are not process pairs, but single processes. If the TMP sends a request to one of them, and they are gone, the TMP merely restarts them. Both the primary and backup TMP keep the current incarnation of the process open. If the TMP finds it necessary to restart one of these processes, it issues a message stating that the process was restarted.

The catalog process is still implemented as a process pair for two reasons:

- It maintains information across requests that would be lost in a restart.
- There wasn't a problem with the catalog failing to survive.

Should the catalog process pair fail, the TMP stops TMF. To recover from this situation, the operator can simply issue a START TMF command.

Conclusion

Several additional changes besides those mentioned in this article are also a part of B00 TMF. They relate to:

- TMP start-up and shutdown processing.
- Audit-trail management.
- New error messages.
- New TMF control files.

For details on the above changes, refer to the TMF Softdoc.

References

Pong, M. 1985. TMF Autorollback: A New Recovery Feature. *Tandem Systems Review*. vol. 1, no. 1. Tandem Computers Incorporated.

Acknowledgments

The author wishes to thank Pat Helland for his ideas, comments, and encouragement, all of which helped a great deal in the writing of this article.

Tony Lemberger is a senior systems analyst in the Large Systems Support group. In his four years at Tandem, he has been involved in application-design consulting, system performance, software education, and, most recently, research and support for DP2.

The performance of Disc Process 2 (DP2) is significantly better than that of the A06 version of Disc Process 1 (DP1). This article presents several types of DP2 performance data, as well as a performance comparison of A06 DP1 and B00 DP2. It discusses some of the new features responsible for the performance improvements and then describes specific performance results from several tests.[1]

Performance Features

The following new features of the disc process (listed in order of their impact) have improved its performance dramatically:

1. Writes can now be buffered in cache memory for both nonaudited and audited disc files.
2. Multiple disc processes can now service requests for a single volume.
3. "Bulk-I/O" service is now used by GUARDIAN 90 operating system utilities for sequential and "bulk" processing.

Another feature of the B00 release that has helped to significantly improve the performance of the disc process is the enhanced implementation of the Transaction Monitoring Facility (TMF).

Finally, three other new features are indirectly related to performance:

1. Users can examine and configure disc cache size on-line.
2. To ensure the integrity of the data residing on disc volumes, DP2 uses an "undo" area to keep information about the structural changes to the volumes until the changes have been completed.
3. Disc-volume labels are refreshed (rewritten) every 30 seconds when nothing has been written to or read from those volumes during that period.

All of these new features are discussed below.

Buffered Cache

Buffered cache, the opposite of write-through cache, provides a tremendous performance advantage to applications that write and update disc files. All applications audited by TMF can take advantage of this new feature.

[1] This article discusses the A06 version of DP1 only. For brevity, A06 DP1 is referred to simply as DP1 throughout the rest of the article.

For unaudited applications, the application itself must provide the data base with protection from failure and concurrency problems. To provide protection from processor failures of the primary disc process, TMF employs a write-ahead-audit protocol. This protocol, implemented in the disc process, ensures that the audit trail is written before the transaction commit is accepted. A06 TMF does not use the write-ahead-audit protocol.

It is worth noting that, in most cases, TMF audit-trail writes are done serially to mirrored-disc drives; a write to one disc drive completes before the write to the mirror is requested. This is true even when the disc has been SYSGENed in a parallel or simultaneous write configuration.

As would be expected, buffering can dramatically reduce the number of physical writes required to a disc volume. The most improvement is realized when an audited cache-resident file is buffered; that is, when sufficient cache is configured for the entire file to fit into the disc cache. Most data bases are too big to fit entirely in cache, however. The cache blocks are written to disc either in "spare time" (as decided by the disc process) or when the file is closed.

There is another means by which buffered writes are sent to disc. At selected intervals (currently every five minutes) a search is made through the disc cache for cache blocks that have been updated but not written to disc (dirty blocks). When a dirty block is found, it is marked for writing to disc during the next interval. If a block is found to already have this mark, it is immediately written to disc.

The reason for this "mark, then write" mechanism is to prevent a wholesale cache write, which could have a serious impact on requests to the disc volume. If a large cache were configured, say 1M byte or 2M bytes, several hundred writes would be required to flush out dirty blocks. With this method most updates can be made in spare moments between the five-minute control-point periods, and fewer writes are required at control-point time.

Most of the physical I/O savings realized from buffering are essentially free of additional system cost. Reads previously made from cache memory can now be followed by writes to the identical cache-memory locations. This saves a physical disc I/O or two writes when mirrored discs are used.

Multiple Disc Processes

DP2 provides the capability of multiple (one to eight) primary and backup disc processes per logical volume. It permits some disc processes in the disc-process group to prepare requests to the disc while other requests to the disc or cache are being serviced by other disc processes. This allows higher utilization of the disc hardware and disc cache. When physical reading is required (for a cache-read miss or a bulk I/O transfer) on a mirrored volume, eachof the discs in the pair can provide a separate access path to files on the volume simultaneously.

Bulk I/O

The new bulk-I/O features of DP2 allow Tandem utilities to transfer up to 30K bytes with a single request to the disc process. The utilities that use this feature are the File Conversion Utility (FCP), BACKUP2, RESTORE2, and the File Utility Program (FUP). TMF on-line dumps also use this interface to transfer up to 8K bytes of data per request.

With bulk I/O, the disc process bypasses cache to save time and avoid "swamping" the disc cache with blocks that will probably not be reused. In addition, the 3107 disc controller can request up to 30K bytes of data from the controller in a single request. For the 3106 controller, the disc driver issues several 4K requests from which the 30K bytes are assembled and, upon completion, the disc process gives a single response of 30K bytes to the application.

DP1 does not support the bulk-transfer capability of the 3107 controller. In DP1, a single request to the disc process can only provide 4K bytes of data.

TMF Changes

DP2 provides a tremendous savings in the resources required by TMF. Most of the savings result from reduction in the number of disc I/Os required to audit a transaction. This and other enhancements provide a major reduction in CPU busy rates per transaction.

One change in TMF for DP2 is the use of the write-ahead-audit protocol mentioned earlier. This method allows the data base to be restored after a crash by removing incomplete transactions (autorollback) rather than redoing all transactions from a previous on-line dump (rollforward).

A single audit trail can now provide full TMF functionality, whereas DP1 requires two separate TMF audit trails, one for commit records (the Monitor Audit Trail) and another for before and after images of the data-base records (the Data Audit Trail). In most cases a single mirrored physical write is all that is required to complete the auditing process.

Audit trails can also be compressed in DP2 to further reduce the amount of data checkpointed and written to the audit trail. With compression, only the changed portions of records are logged. A new constraint enforced by DP2 is that a volume cannot contain both audit trails and audited files.

DP2 combines the construction of auditing and checkpoint messages, which reduces the number and size of these messages significantly.

Finally, TMF autorollback provides recovery of an audited file by correcting inconsistencies rather than reconstructing the data base a transaction at a time. When TMF is restarted after a failure, an autorollback is automatically performed.

Improved Data Integrity

While data integrity may not be considered an attribute of performance, it does have a significant impact on it. First, DP2 maintains an undo area. Here, operations, which if not completed could result in a structural problem for the volume, are written before the file/directory is updated. If a failure occurs, the operation can be undone and attempted later.

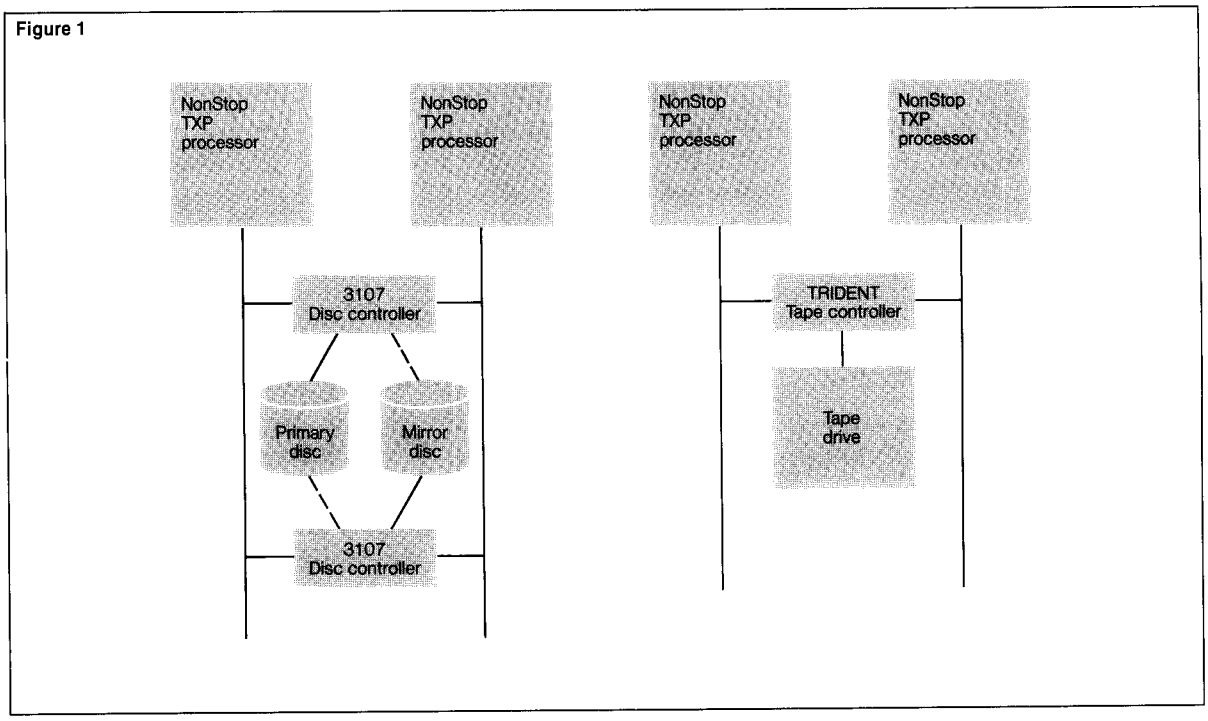
Another DP2 feature that ensures data integrity is the rewriting of disc volume labels every 30 seconds, if no other read or write requests have been issued to the disc within that time.

On-line Configuration

Users of DP2 can configure cache and examine cache performance data on-line, without using the XRAY performance measurement tool. DP2's cache-management scheme provides four fixed lengths for cache blocks. The available sizes are 512, 1024, 2048, or 4096 bytes per block.

The number of blocks for each size can be configured through an interface in the Peripheral Utility Program (PUP). PUP reports current cache performance for tuning. This fixed-length management scheme, with separate buffers for each block size, prevents fragmentation of the cache. The user now must configure four separate caches per volume to suit the application's distribution and access of file blocks.

Figure 1.
Configuration for the BACKUP2 and RESTORE2 tests. A four-processor NonStop TXP system used two 3107 disc controllers to access a single mirrored disc drive configured for parallel writes. It used a TRIDENT tape drive and controller combination to read and write the 6250 bpi tape during the tests.



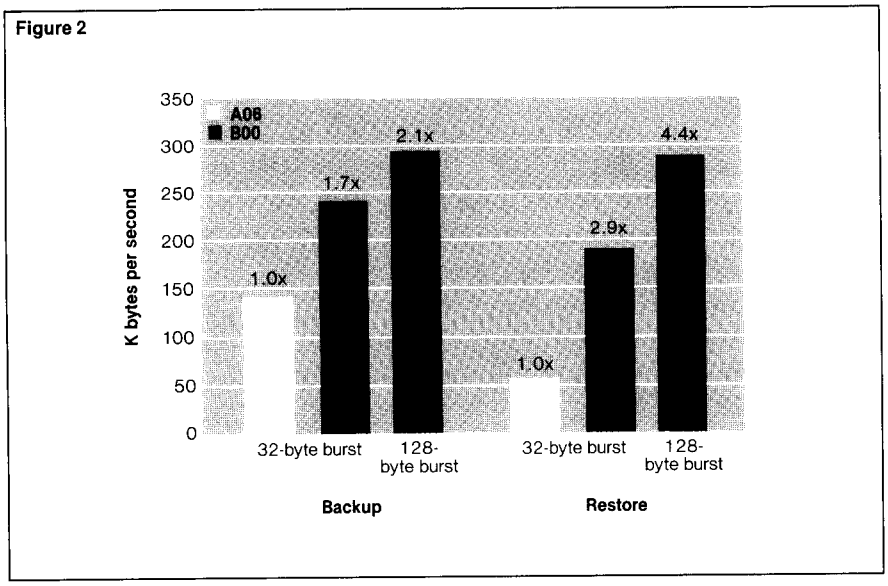


Figure 2.
Data rates for A06 BACKUP and RESTORE and B00 BACKUP2 and RESTORE2. A 25M-byte file with one thousand page extents was written from disc to tape and then restored from tape to disc.

Results of Performance Tests

The results of three major tests are reported here:

1. A backup and restore operation.
2. A sequential-copy operation.
3. An on-line transaction-processing (OLTP) application.

For each, the work load is described first; then a description of the configuration is provided; and finally, the results of the test are given and observations are made.

BACKUP2 and RESTORE2 Performance

A large (25M-byte) file, having one thousand page extents was written from disc to tape and then restored from tape to disc. Measurements were made with XRAY to confirm the results obtained. This was done first with DP1, under the A06 version of the GUARDIAN operating system, with the BACKUP and RESTORE utilities. The same operation was then performed with DP2, under GUARDIAN 90, with BACKUP2 and RESTORE2. The TAPEBLOCKSIZE parameter was used with BACKUP2, and 30K-byte blocks were written to tape by DP2.

Identical hardware was used for both tests. Figure 1 shows that the configuration consisted of a four-processor NonStop TXP system using two 3107 disc controllers to access a single mirrored disc drive configured for parallel writes. A TRIDENT tape drive and controller combination was used to read and write the 6250 bpi (bits-per-inch) tape during the tests.

In Figure 2, the per-second data rate for writing and reading back the tape are shown. The data rates relate directly to the amount of time taken to read and write the data to the tape drive. See the accompanying article, "Improved Performance for BACKUP2 and RESTORE2," by Anil Khatri, for more information on BACKUP2 and RESTORE2 performance. The bulk-I/O interface to DP2 and the algorithm changes in BACKUP2 and

RESTORE2 are responsible for most of the performance improvement. The bulk-I/O interface avoids copying the data unnecessarily.

While BACKUP can write only 540M bytes of data to tape in an hour, BACKUP2 can write 1047M bytes of data to tape in the same amount of time. Similarly, while RESTORE can write only 234M bytes of data to disc in an hour, RESTORE2 can write 1033M bytes in that amount of time.

Sequential-copy Performance

For the sequential-copy application, a 100K-byte file was copied from one file to another on the same disc. This was done from a program written in the Transaction Application Language (TAL). A single extent was used for each file. All DP2 tests were run with buffered-cache writes. (As mentioned earlier, buffered cache is not available in DP1.) A single mirrored volume contained both files. Sequential block-buffering was used for reads.

Results for three versions of this application are presented here. The first version used unstructured access to the file. For this version, 25 blocks of 4K bytes were read and written to move the 100K bytes. For the two versions using structured access, one thousand 100-byte records were read and then written to move the 100K bytes.

Identical hardware was used for all tests. Figure 3 shows that the configuration consisted of a four-processor NonStop TXP system using two 3106 disc controllers to access a single mirrored disc drive configured for parallel writes.

The results in Figure 4 show the elapsed time to copy the file. In the unstructured test, DP2 showed a 10% improvement in elapsed time over DP1. This modest improvement can be explained by the fact that unstructured access is very efficient under DP1 and, therefore, difficult to improve upon.

Figure 3

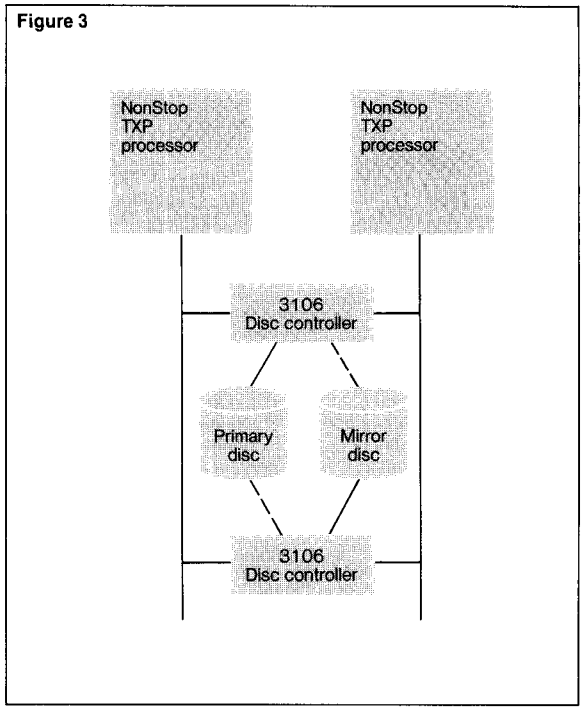


Figure 3.
Configuration for the sequential-copy tests. A four-processor NonStop TXP system used two 3106 disc controllers to access a single mirrored disc drive configured for parallel writes.

Figure 4

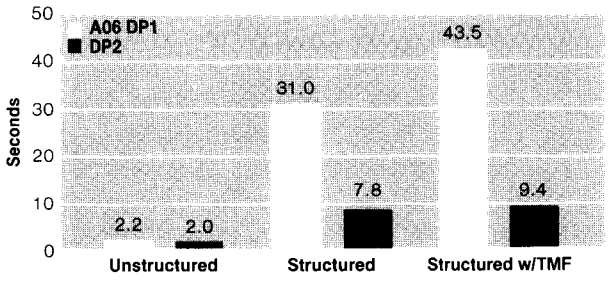


Figure 4.
The elapsed time required for A06 DP1 and B00 DP2 to copy 100 bytes sequentially from one file to another on the same disc.

Table 1.
The elapsed time required by A06 DP1 and B00 DP2 to copy 100 bytes sequentially from one file to another on the same disc.

	A06 DP1	B00 DP2
Unstructured access		
Read only	.7	.7
Read and write[1]	2.2	2.0
Read and write with TMF	3.8	4.1
Structured access		
Read only	4.3	4.0
Read and write[1,2]	31.0	7.8
Read and write with TMF[1,2]	43.5	9.4

[1]Buffered cache was used for all DP2 write tests.
[2]Sequential block-buffering was used for all structured file tests.

In the first structured test, DP2 reduced the elapsed time by 75%, to a factor of four. This is due primarily to the effect of buffering and, to a lesser extent, to more efficient checkpointing. In the second structured test, in which TMF was used, a still more significant performance gain was observed: the elapsed time was reduced from 43.5 seconds to 9.4 seconds. The increased efficiency of TMF auditing and checkpointing are the primary reason for this improvement. Table 1 shows the full sequence of all the file operations.

OLTP Performance with TMF

The performance of a large retail-banking application that used the full ENCOMPASS line of application-development products was tested. A Screen COBOL (SCOBOL) requester was used by the Terminal Control Process (TCP), a part of the PATHWAY transaction-processing system. The TCP sent requests to a server written in COBOL. All TCPs and disc processes were run as fault-tolerant processes.

All application files were audited by TMF. The ENCORE stress-test generator was used to simulate 500 terminals submitting transactions. XRAY was used to collect performance data. ENCORE provided response time and transaction-rate data. The ENFORM relational query language was used to process the performance data. The transaction flow is outlined below:

Requester flow

Accept 100 bytes.
Perform "depending on" algorithm.
Begin TMF transaction.
  Send 100 bytes to server with
    100-byte reply.
  Perform 10 move statements.
  Perform 10 if statements.
  Perform 10 simple calculations.
End TMF transaction.
Display 100 bytes.

Server flow

Read 100 bytes from TCP.
READ Account file (random, not cached).
UPDATE Account file.
WRITE History file (sequential, cached).
READ Teller file (random, cached).
UPDATE Teller file.
READ Branch file (random, cached).
UPDATE Branch file.
Send 100-byte reply to TCP.

Data-base Description. The application data base in this test consisted of over 100M bytes of application data in four files. Three of the application data-base files were accessed randomly. The fourth file, an entry-sequenced log, was written sequentially, one record for each transaction. All three structured-file types were used. Table 2 contains a complete description of each application file.

Table 2.
Description of the data-base files used in the OLTP tests.

File type	Name	Record size (bytes)	Number of records	Type of access	Other Information
Key-sequenced	Account	100	1,000,000	Random	Key-sequenced, not cached
Relative	Branch	100	100	Random	Cached
Relative	Teller	100	500	Random	Cached
Entry-sequenced	History	50	1 per transaction	Sequential	

System Description. In Figure 5 is a diagram of a processor pair. The system consisted of NonStop TXP processors with 4M bytes of memory each. There were four 3106 disc controllers per processor pair. Sixteen disc drives were used to make eight mirrored volumes. Five hundred simulated terminals were equally distributed on the system.

Process Layout. Processes were evenly distributed to balance the processing load. Two disc volumes were "primaried" in each CPU. An equal number of TCPs per CPU were used (five for DP1 and three for DP2). The TCP configuration under DP2 was changed to provide additional memory for the disc processes. Approximately 100 pages of memory were saved by reducing the number of TCPs in a CPU. Three DP2 disc processes per volume (the default) were SYSGENed. Simulators and servers were evenly distributed.

Performance Results. The number of physical I/Os required to complete a transaction was reduced by 12.3 with DP2. In the test, 18.9 disc I/Os were required by DP1, while only 6.6 disc I/Os were required by DP2 for the same transaction. This is a 65% reduction in total disc I/Os.

The amount of CPU time required per transaction was reduced by 25% for DP2.

In the test, the DP2 system required a 10% increase in the total amount of memory over that required by the DP1 system. (Disc caches were not included in this figure. If users increase cache sizes, this increase would be in addition to the 10% increase required by DP2.)

All disc caches were between 64K bytes and 128K bytes. For both the DP1 and DP2 systems, adequate cache was provided.

Figure 5

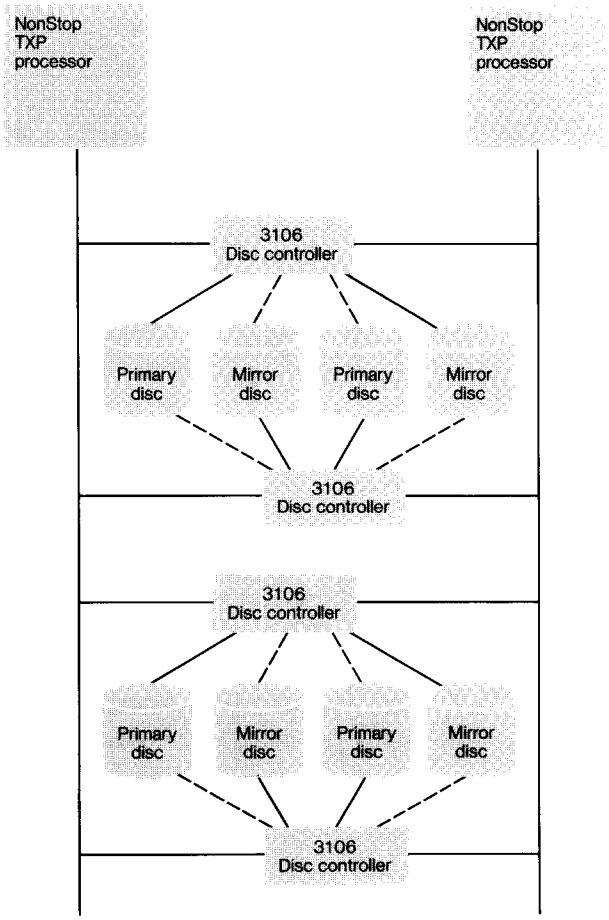
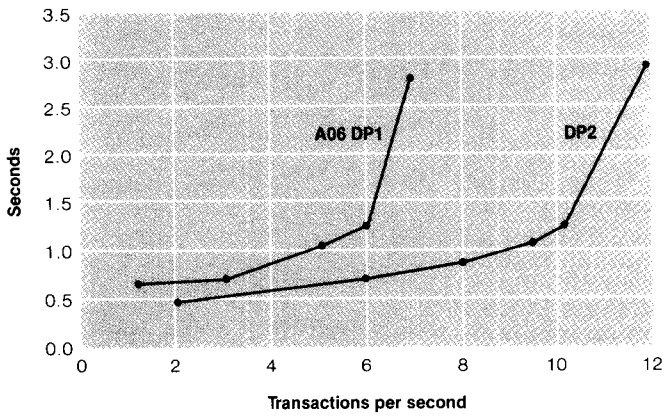


Figure 5.
A processor pair. In the on-line transaction-processing (OLTP) tests, the system consisted of NonStop TXP processor pairs with 4M bytes of memory in each processor. There were four 3106 disc controllers per processor pair. Sixteen disc drives were used to make eight mirrored volumes. Five hundred simulated terminals were equally distributed on the system.

Figure 6.

Response time vs. throughput for the OLTP application. The test application, patterned on a large retail-banking application, used the full ENCOMPASS line of application-development products, including TMF. The data base consisted of over 100M bytes in four files; three were accessed randomly, and one was written sequentially. All three structured-file types were used. Processes were evenly distributed to balance the load.

Figure 6



Response Time and Throughput. Response time and throughput improvements are presented in Figure 6. For applications with a particularly stringent response-time requirement, DP2 offers a significant reduction. In the test, at six transactions per second (for 95% of the transactions processed), DP2 provided a response time of .75 seconds, while, at the same transaction rate, DP1 yielded a response time of 1.25 seconds.

A more interesting and meaningful observation can be made by comparing the work load that the system can process at nearly identical response times. If a 1.25-second response time is required for 95% of the transactions, A06 DP1 provides six transactions per second (TPS). DP2 can deliver 10.2 TPS at this same response time. This is a throughput improvement of approximately 1.7 times, when identical hardware is used. The improvement is similar at higher and somewhat lower transaction rates and for average response times.

I/O Reduction. The TMF audit-trail write activity has the most dramatic reduction of I/Os per transaction. Where DP1 required nine audit writes to three files, DP2 required two writes to a single file (i.e., a single write to a mirrored disc). This I/O reduction, in combination with the buffering of the application data base (three mirrored writes to cached files or six I/Os), accounts for the reduction in disc-busy time per transaction of 130 ms. DP1 required 300 disc ms per transaction, DP2 only 170 ms.

Decrease in Messages. More efficient checkpointing of messages to the backup disc process, audit disc process (ADP), and backup ADP is the reason for the balance of the CPU-busy reduction. The decrease in message traffic had many other side effects as well.

Since the dispatcher is responsible for the sending portion of an interprocess communication (IPC), a reduction in dispatches per transaction was measured. The dispatches per transaction were reduced from 367 to 222 (40%) as a direct result of the overall reduction of 3.7 to 1 in message bytes per transaction. The disc process experienced a 5 to 1 reduction in message bytes per transaction.

Table 3 shows the message traffic in more detail. DP2 sends one-half as many messages as DP1; furthermore, the remaining messages are one-half their previous size. (These figures were obtained without audit compression.)

Interrupts. Interrupts serve many purposes; one of these is the completion of interprocess communications via BUSRECEIVE interrupts, and another is the processing of previously requested EXECUTE I/O (EIO) requests to and from the channel. Because of the reduction in disc I/O and message traffic, CPU time spent processing interrupts has been reduced by 33%. A breakdown of CPU time for both DP2 and DP1 appears in Figure 7.

Surprises and Items of Interest. System balancing and tuning with DP2 was quickly accomplished. Whereas with DP1, resource consumption depended upon many physical file characteristics, DP2 displayed a "generic access" pattern. Thus, in DP2, an access was simply an access. A sequential or random write to a cached file required approximately equivalent resources. This made disc-process balancing with DP2 easy.

Processes and files were carefully balanced in the tests for both DP1 and DP2. Substantial balancing and rebalancing (such as tuning cache, moving processes to less busy CPUs, and moving files to less busy discs) is normally required to gain maximum system performance, but with DP2, only the initial balancing after installation was necessary.

The first DP2 benchmark showed all processor-busy times within 2% of each other. The last benchmark, with a CPU-busy of 90%, showed a response time of less than three seconds for 95% of the transactions, and the CPU-busy was still balanced within 3%.

Table 3.
Number of messages and message bytes dispatched per transaction in the OLTP tests.

	Messages per transaction		Message bytes per transaction	
	A06 DP1	**B00 DP2**	**A06 DP1**	**B00 DP2**
Disc process	87	36	54,771	10,120
TCP	8	8	4,364	4,562
Servers	8	8	913	912
Replay (simulator)	3	3	677	721
Remainder	1	1	362	272
Total	108	56	61,088	16,587

Figure 7

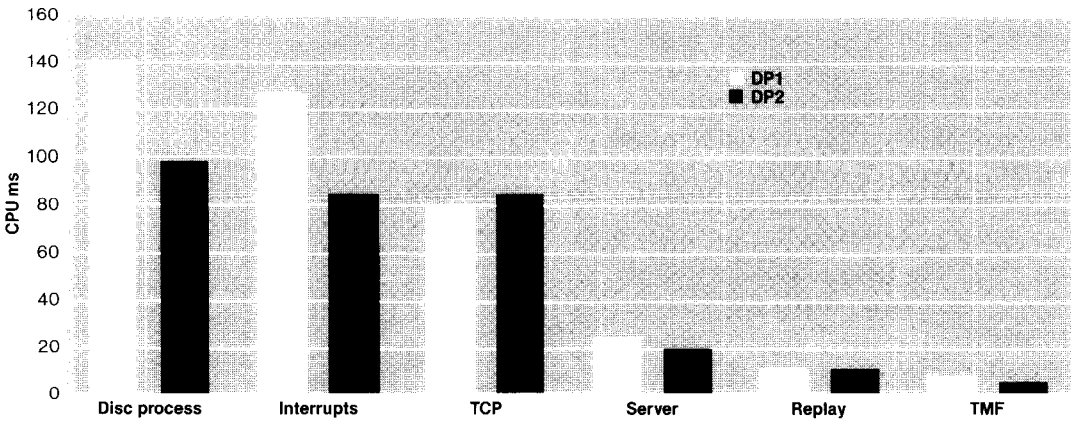


Figure 7.
A breakdown of CPU time for A06 DP1 and B00 DP2, as measured in the OLTP application tests. TMF was used with both disc processes.

The throughput improvement of 1.7 times for equivalent response times measured for DP2 can be attributed to several factors. As shown in Figure 6, at a three-second response time (for 95% of the transactions) the average CPU-busy for DP1 was 75%. As a result of cache buffering and disc-process multithreading, DP2 operated in excess of 90% utilization and still delivered the same response times. This ability to operate at higher CPU utilizations, coupled with the decreased CPU requirements per transaction, explains the improvements measured for DP2.

In other tests, configured with three DP2 disc processes on a single volume, two interesting observations were made. When physical access was required to retrieve records from disc, three disc processes could swamp the disc with requests. When cache access was provided, these three disc processes could swamp the CPU with requests. It should also be noted that the number of requests per second that could be satisfied under these conditions improved from 1.3 to 7.5 times, depending on the file type and access mode.

Use of a single audit trail improves the performance of most applications as long as the entire audit-trail write (per transaction) is less than 4K bytes. For applications requiring more than 4K bytes of before and after images and commit records, audit compression may shrink the audit buffer down to less than 4K bytes per transaction. This 4K-byte boundary is important because the entire buffer is written for each transaction commit. When more than 4K bytes must be written for each transaction, an auxiliary audit trail should be configured on a separate volume. (Also, auxiliary audit trails should be configured if parallel recovery is desired.)

Configuration Issues

For most applications running on NonStop TXP processors, three DP2 disc processes fully utilize the disc and CPU hardware. When buffering is fully utilized, process-balancing provides maximum system throughput. File balancing (in accordance with the new physical-access requirements of the system) is also required to improve throughput.

As DP2 uses buffered cache for writes, small to medium files that are randomly accessed should now be entirely cached, and adequate cache should be configured for them.

Also, in general, larger caches for all files should be provided. There are two reasons for this. One is that writes can now be "hits" in cache. In DP1, a physical write would have been required to flush the write, no matter how large a cache was configured. The second reason is that the additional disc servers provide more access to the cache, making a larger cache more useful. The enhancements to PUP mentioned earlier make it easy to configure and measure larger caches on-line. (Note that when cache is reconfigured, it is flushed.)

Also, cache can be more effectively utilized when multiple servers are used. The same is true of disc drives when more application files are placed on a disc volume.

Application Issues

The performance of many different types of user application will improve with DP2. Applications that will benefit most are those that write sequentially and use buffered cache. Those that write randomly and use buffered cache will also benefit greatly.

Any application audited by TMF will experience major performance improvements, and those requiring periodic backups or restorations will find the amount of time these require to be dramatically reduced.

The performance of read-intensive applications that use mirrored-disc and multi-threaded requests will improve, since the mirror copy can now service a separate read request.

Applications that will benefit least from DP2 are those that cannot use any of its new features. If an application does not use TMF or buffered cache, has only one request outstanding to a volume at one time, or is already CPU-bound, its performance should not be significantly improved with DP2. Even for this type of application, however, reliability and recoverability are improved with DP2.

Future Work

DP2 performance in many different modes of operation has been examined. As performance studies continue, more will be learned about DP2's operational characteristics. Some of the performance areas being studied are:

1. B00 DP1 vs. DP2 performance.
2. DP2 performance on OLTP applications that do not use TMF.
3. DP2 performance on the NonStop II processor.
4. The time and resources required by File System atoms (basic units of work).
5. Audit-trail compression and its effects on CPU-busy, message traffic, and audit trail size.

Also, as more user applications convert to DP2, more data about its performance will be available for analysis.

Conclusion

DP2 performance is substantially improved over that of A06 DP1, as a result of the significant improvements in its implementation. Applications using DP2 with B00 TMF will realize tremendous savings in disc I/O. Those applications using cached or sequential writes will see a dramatic reduction in disc I/O.

In the tests described in this article, less CPU time was required by DP2 for equal work. With DP2, fewer messages are sent, fewer CPU interrupts are generated, and more CPU time is available for use. Also, BACKUP2, RESTORE2, and FUP provide substantial performance gains. These improvements in performance are accompanied by the improved reliability, recoverability, and functionality provided by DP2.

Acknowledgments
Much of the information in this article was derived from over six months of performance measurement and analysis by the Performance Analysis and Measurement group in Software Development. The members of the group are Pat Beadles, Anil Khatri, Dennis Markt, Praful Shah, Susan Truong, and the author. Many other groups contributed information as well.

Jim Enright, Manager of the Performance Analysis and Measurement group in Software Development, joined Tandem in August 1983. His group has recently been involved in testing and analyzing the performance of DP1 and DP2, along with that of other Tandem software and hardware products.

A Comparison of the B00 DP1 and DP2 Disc Processes[1]

	DP1	DP2
Maximum extents per file:		
Nonpartitioned	16	Variable (more than 16)[2]
Partitioned	16 × the number of partitions	16 × the number of partitions
Maximum number of partitions	16	16
Number of directory extents	1	978
Largest extent	65535 pages, 134,215,680 bytes	65535 pages (includes bit maps not available for data storage)
Legal block sizes	512, 1024, 1536, 2048, 2560, 3072, 3584, 4096 bytes	512, 1024, 2048, 4096 bytes
Block-header length:		
Relative	20 bytes	20 bytes
Entry-sequenced		
Key-sequenced data		30 bytes
Key-sequenced index		24 bytes
Bit-map blocks (Relative, key-sequenced)	Do not exist	18 bytes
Maximum number of records in a block (N) for different record lengths[3]	N = (B-22)/(R + 2) where B = block length T = record length	N = B-X/(R + 2) where X = 22 for relative files X = 22 for entry-sequenced X = 32 for key-sequenced

[1] A more detailed version of this chart appears in the *ENSCRIBE Programming Manual*, Part No. 82583 A00.

[2] In DP2, extent and alternate-key information share the same storage.

[3] The maximum number of records in any block is 511.

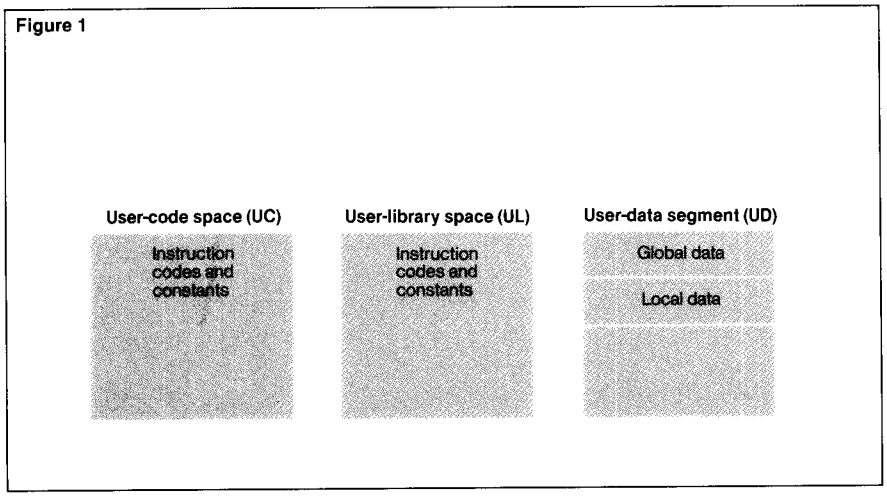
	DP1	DP2
Maximum record length (Block size = 4096):		
Relative Entry-sequenced	4072 bytes	4072 bytes
Key-sequenced	2035 bytes	4062 bytes
Unstructured	4096 bytes	4096 bytes
Controllers	3106 3107 (treated like 3106)	3106 3107 (long transfers)
Key-sequenced index and data blocks	May be different sizes	Must be the same size
Lock search	Sequential search on locks by file	Hash-code access to lock table
Audited files and audit-trail files	Audited files and audit-trail files may exist on the same volume	Audit-trail files cannot exist on the same volume as audited files
Audit-trail file contents	Monitor Audit Trail: Commit and abort records Data Audit Trail: Data records	Master Audit Trail: Commit, abort, and data records Optional Auxiliary Audit Trails: Data records
Cache	Binary search Not dynamic Write-through, buffered for audited files LRU Access Mode	Hash-code access Dynamic Buffered option LRU, sequential, direct I/O access modes

Increased Code Space

Tandem NonStop II, NonStop TXP, and NonStop EXT systems were originally designed with a limit of 64K words of user code and 64K words of user-library code (128K words in total) per process. With the B00 software release, this limit has increased to 2M bytes for user code and 2M bytes for user-library code (4M bytes total) per process. Also, system-code area has been expanded from 128K words to 4.125M bytes.

These enhancements were added in response to user requests for more user-code space and in response to the need for system-code expansion. They are an integral part of the GUARDIAN 90 operating system and are not an optional product. DEBUG, INSPECT, and BINDER are the only products in which the user interface has been changed as a result of the enhancements.

Figure 1.
A user process consists of user-code space (UC), user-library space (UL), and a user-data segment (UD).



Users having programs that require more code space will find creating and managing multisegment program files very simple. File creation and maintenance for programs with single-segment files have not changed. Multisegment program files are just an extension of single-segment program files.

User Code and Library Code

A running process consists of compiled instructions in code space in memory that manipulate data contained within a separate data segment in memory. Code for a process consists of user-code space (UC) and optional user-library space (UL). The user-code space contains the code from the program file and the user-library space contains the code from the library file specified in the RUN command. Figure 1 illustrates this.

In the GUARDIAN 90 release, user-code space and user-library space have been increased from one 64K-word segment each (i.e., 128K words of total code per process) to up to sixteen 64K-word segments each. That makes 2M bytes (1024K words) of user-code space and 2M bytes (1024K words) of user-library space, adding up to 4M bytes (2048K words) of total code space per process.

As illustrated in Figure 2, each of the 16 segments in the user-code space or user-library space is defined by a space identifier or SPACEID. The SPACEID is made up of an identifier (UC for user code or UL for user library) and an octal index number in the range of %0 to %17. For example, a SPACEID of UC.4 specifies segment %4 in the user-code space.

Each segment within a code space consists of sections of compiled source code called code blocks. Some code blocks are limited to 32K words because of language restrictions. Code blocks cannot cross segment boundaries. Each language defines code blocks and their sizes differently.

COBOL defines a code block as the *main program* or a *subprogram*. The maximum size of a code block is now 64K words, making this the maximum size of a main program or a subprogram. (A20 COBOL is limited to a code block of 32K words.)

FORTRAN also defines a code block as a main program or subprogram, but it calls them *program units*. The maximum size of a FORTRAN code block is 32K words.

TAL defines a code block as a *procedure*. The maximum size for a TAL code block is 32K words.

Considerations for Multisegment Programs

The code-space enhancements do not affect the designing of application programs. The only change is that now more code blocks can be added to a program file. General rules for creating multisegment-program files are discussed below; they apply to all languages.

Code-block Location

Multiple code blocks can reside in the same segment as long as the total size is less than 64K words.

Code blocks that are frequently called should reside within the same segment as the calling code block. Arranging code blocks with this in mind increases the performance of the application (see the section called "Performance").

Figure 2

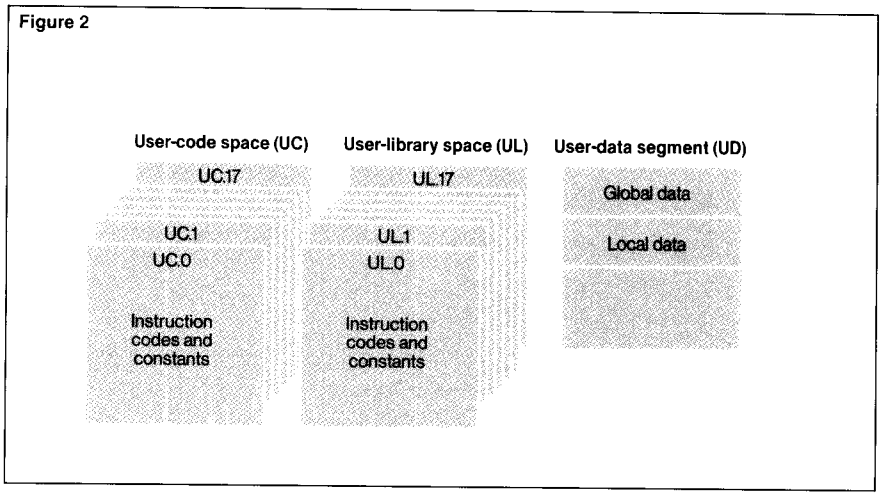


Figure 2.
With multisegment program files, the user-code space and user-library space can contain up to 2M bytes of code (sixteen 64K-word code segments) each.

P-relative Code Arrays

Two guidelines exist for handling P-relative code arrays in multisegment program files:

- Each global P-relative code array is replicated in every segment that contains a reference to the array by BINDER. Code-space requirements should be evaluated with this in mind.
- The passing of code arrays in a call to another code block is not supported; however, existing single-segment programs will continue to work.

Maximum Code-block Size

As mentioned earlier, the maximum size for code blocks is different for each language. For FORTRAN and TAL, it is now 32K words; for COBOL it is 64K words.

Code-block Names

The names of code blocks cannot be the same within a multisegment program.

Figure 3

ENTRY POINT MAP BY NAME

SP	PEP	BASE	LIMIT	ENTRY	ATTRS	NAME
17	002	001000	174627	001403		SPACESA
16	002	001000	174627	001403		SPACESB
15	002	001000	174627	001403		SPACESC
14	002	001000	174627	001403		SPACESD
13	002	001000	174627	001403		SPACESE
12	002	001000	174627	001403		SPACESF
11	002	001000	174627	001403		SPACESG
10	002	001000	174627	001403		SPACESH
07	002	001000	174627	001403		SPACESI
06	002	001000	174630	001403		SPACESJ
05	002	001000	174630	001403		SPACESK
04	002	001000	174630	001403		SPACESL
03	002	001000	174630	001403		SPACESM
02	002	001000	174630	001403		SPACESN
01	002	001000	174630	001403		SPACESO
00	002	001000	174630	001403		SPACESP
17	003	174630	174702	174640	M	T16UC

← The segment number in octal

BINDER - OBJECT FILE BIND - T9621B00 - (28JAN85) SYSTEM \TEST
Object file name is $DISC1.COBTEST.RUNUNIT
Number of Binder errors = 0
Number of Binder warnings = 0
Primary data = 16 words
Secondary data = 11527 words
Code area size = 1008 pages ← Number of pages required for the code
Resident code size = 0 pages
Data area size = 35 pages
Number of code segments = 16 segments ← Number of segments used

COBOL - T9251B00 - (28 JAN 85)
The object file is executable only on a NonStop II processor
Number of COBOL errors = 0
Number of COBOL warnings = 0
Number of source lines read = 27570
Maximum symbol table size = 4480 words
Elapsed time - 0:31:39

Figure 3.
The compiler listing of this multisegment program file contains 16 segments. Each subprogram, SPACESB through SPACESP, resides in its own segment and consumes approximately 63,895 words (decimal) of each segment. Subprogram SPACEA also contains the main code block, T16UC, which consumes 63,938 words (decimal) of segment %17.

CURRENTSPACE Procedure

A program can determine which segment it is currently executing by calling the new procedure CURRENTSPACE. This procedure returns the stack-marker environment (ENV) register and an ASCII version of the SPACEID. The syntax for this procedure is:

CALL CURRENTSPACE (<ASCII-space-id>);

ASCII-space-id contains an ASCII string in the form of <space>.<#> where <space> equals UC, UL, SC, or SL, and <#> equals the octal segment number in ASCII characters. <Spaceid> is an integer that contains the SPACEID in ENV register format, as follows:

SPACEID	Bit(s)
ENV.<4>	Library bit
ENV.<7>	System-code bit
ENV.<11:15>	SPACEID bits

Example of a Multisegment Program File

Figure 3 contains an example of a multisegment program file created by COBOL. The source code for the main program and all subprograms were contained in one EDIT file. The command COBOL/IN <edit file>, OUT $S.#LP/ was used to compile the source and produce the compiler and BINDER reports.

The total size of the code area is 1008 pages or 2.064M bytes, contained in 16 segments. In the BINDER report, "Entry Point Map by Name," the segment in which a code block resides is shown under the heading "SP." (The segment numbers listed are in octal.)

Calling Subprograms and System Procedures

A nonprivileged process' view of virtual memory is divided into six short-address spaces (SASs). The SASs contain the current executing user-process environment and part of the operating-system environment. They are defined in Figure 4.

Before a process can execute, the minimum number of required pages of the user-code segment and user-data segment are placed into the appropriate SAS. The user-library segment is mapped only if the user has specified a library file and it is referenced (via an XCAL). System data and system code are always mapped (because each is contained in a single segment). In GUARDIAN 90, the system library has been expanded to two segments (SL.0 and SL.1).

When a compiler calls a code block that resides within the same segment, it generates an internal-procedure reference (PCAL) instruction. When it calls a code block outside of its segment, it generates an external-procedure reference (XCAL) instruction.

Executing an XCAL instruction for a code block that is not mapped takes longer than it does for one that is mapped. In order for code to execute, it must be mapped into one of the SASs. If a code block in UC.0 were to call a code block in a UC segment other than UC.0 (e.g., UC.1 through UC.17) or in an unmapped UL or SL segment, the call would take longer than it would if the called code block resided in UC.0 or if the UL or SL segment were mapped.

For example, if the called code block resided in UC.1, UC.1 would first have to be mapped into SAS 2 (overwriting UC.0 with UC.1). Once SAS 2 contained the code for UC.1, the called code would execute. On the return to UC.0, UC.0 would have to be remapped into SAS 2. This would obviously require more time than that required for UC.0 to call a code block that was already mapped in an SAS (e.g., UL, SC, or SL).

Performance

The performance of an application can be affected by the placement of code blocks within segments in a multisegment program. The general rule for the placement of code blocks within segments is: code blocks that frequently call each other should be placed in the same segment.

Figure 4

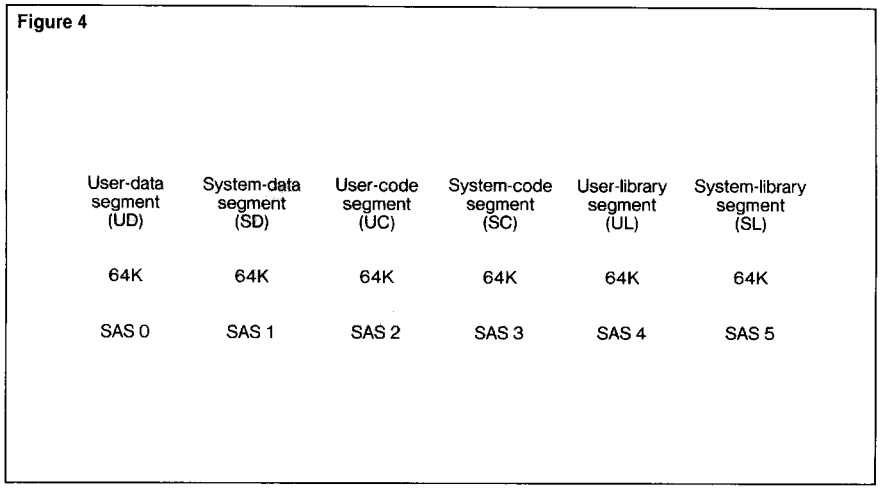


Figure 4.
For a nonprivileged process, virtual memory is divided into six short-address spaces (SASs). The user process can execute code within UC, UL, SC, or SL spaces. The code can manipulate data in the UD or SD segments.

To illustrate the effect of code-block placement on the performance of the NonStop II and NonStop TXP processors, two TAL programs with nearly identical code were written. The only difference between the two was that the main program and subprogram of the first program (PROGA) were contained in the same segment (requiring only a PCAL instruction) and the main program and subprogram of the second program (PROGB) were placed in different segments (necessitating an XCAL instruction).

Because PROGB's subprogram resided in a different segment, segment switching in SAS 2 occurred between the main program and the subprogram (creating a longer execution time). Each program ran alone, passing no parameters, and the called subprogram contained no code, causing an immediate return to the main program. The code for the two programs was as follows:

Main loop

```
Call TIME(time1);
while count
< 50000D do begin
  count: = count + 1D;
  call subprog^test;
  end;
call TIME(time2);
```

Subprogram

```
PROC SUBPROG^TEST;
begin end;
```

Table 1.
Execution time for internal-procedure (PCAL) and external-procedure (XCAL) reference instructions for two TAL programs running under GUARDIAN 90.

NonStop II processor

Program	Type of call	Execution time: Microseconds per call	Execution time: Calls per second
PROGA	PCAL	16.0	62,000
PROGB	XCAL	51.0	19,500

NonStop TXP processor

Program	Type of call	Execution time: Microseconds per call	Execution time: Calls per second
PROGA	PCAL	5.4	185,000
PROGB	XCAL	10.0	96,000

The performance timings for the execution of PROGA and PROGB on the NonStop II and NonStop TXP processors are reported in Table 1.

The results for both processors show that an XCAL instruction takes longer to execute than a PCAL instruction. Programmers concerned about the increased execution time caused by inefficient code-block placement should use BINDER to place code blocks that frequently call each other within the same segment.

Note that the NonStop TXP processor executes PCAL and XCAL instructions faster than the NonStop II processor. This is mainly because the NonStop TXP processor maps segments differently in the SASs. Further information about this can be found in the *System Description Manual* for NonStop systems, in the section entitled, "Addressing and Memory Access."

Binding a Multisegment Program File

A multisegment program file can be built by default during compilation (the compiler invokes BINDER) or manually by using BINDER to place the code blocks within the appropriate segments.

Below are a few considerations for using BINDER with multisegment program files:

- The main code block (COBOL, TAL, or FORTRAN main program) does not have to reside within the first segment.
- An unused segment cannot reside between two used segments (e.g., segment 2 cannot contain code blocks if segment 0 contains code blocks but segment 1 is empty).
- The STRIP command is not allowed on a multisegment program file. To produce a multisegment program file without symbols, the compiler directive ?NOSYMBOLS must be used or the SYMBOLS OFF parameter must be specified with the BINDER SET command.

Figure 5 contains an example of a COBOL application in which the code blocks have been arranged to maximize performance. A diagram of the code-block arrangement is shown in Figure 6.

The program consists of six code blocks, one main program and five subprograms, SUB1 through SUB5. SUB3 is called frequently from the main program, SUB2 usually calls SUB5, and SUB1 and SUB4 are called only when errors occur. Each of the six code blocks are 25K words in length. For maximum performance, MAIN and SUB3 should reside in the same segment, SUB2 should reside with SUB5, and SUB1 and SUB4 can reside anywhere.

Compatibility

The use of multisegment program files has introduced one incompatibility: NonStop 1+ programs using procedure labels that were compiled with a pre-E08 compiler do not execute on NonStop II, TXP, or EXT systems.

Procedure labels are a third mechanism for calling external or internal code blocks. They are coded in this way:

```
STACK @ <code-block-name>;
CODE (DPCL);
```

The first line loads the code block's address into the register stack (register A). Register A now contains a procedure label. The second line makes a dynamic procedure call. Control is transferred to the code block specified in the register stack.

When the above code is compiled on a pre-E08 NonStop 1+ compiler, the STACK operation of the code-block name does not generate a set-code-map (SCMP) instruction. Because of this, the procedure label is not in proper form and does not execute correctly on a NonStop II, TXP, or EXT system. Both B00 and pre-B00 NonStop II, TXP, or EXT compilers automatically generate SCMP instructions after the STACK code-block-name operation.

The following NonStop 1+ programs compiled with a pre-E08 compiler are not transportable to B00 NonStop II, TXP, or EXT systems without being recompiled by a B00/E08 compiler:

- FORTRAN and TAL programs containing external-procedure labels.
- FORTRAN and TAL programs containing internal procedure labels used as a user library or programs combined with other program files to create a multisegment program file.
- COBOL programs that have had the pre-B00 COBOL run-time library bound into the program file.

Figure 5

```
BINDER - OBJECT FILE BIND - T9621B00 - (28JAN85) SYSTEM \TEST
                          <-------- Adding code in segment 0 (default)
@ADD CODE * FROM MAIN
@ADD CODE * FROM SUB3
@ADD SPACE                <-------- Advances to segment 1
@ADD CODE * FROM SUB2
@ADD CODE * FROM SUB5
@ADD SPACE                <-------- Advances to segment 2
@ADD CODE * FROM SUB1
@ADD CODE * FROM SUB4
@BUILD PROGFILE

ENTRY POINT MAP BY NAME

SP   PEP   BASE     LIMIT    ENTRY    ATTRS  NAME
00   002   000006   060656   000570   M      MAIN
02   002   000006   060656   000327          SUB1
01   002   000006   060656   000327          SUB2
00   003   060656   141526   000327          SUB3
02   003   060656   141526   000327          SUB4
01   003   060656   141526   000327          SUB5
```

Figure 6

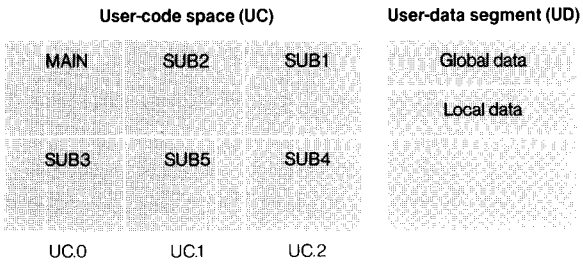


Figure 5.
The ADD SPACE command is used to increment the segment number in which code blocks are added. The BINDER "Entry Point Map by Name" report shows where each code block was placed.

Figure 6.
A diagram of the example COBOL application in Figure 5.

The transportability of multisegment program files is restricted in the following way:

- Multisegment program and library files do not execute on a NonStop 1+ system. They fail with an *Illegal Program File Format* error message at run time.
- The condition code (CC) and register pointer (RP) fields of the ENV register in the stack marker now contain the encoded space identifier. Therefore, any program that reads the CC or RP from the stack-marker ENV register must be recoded to obtain those values from the hardware ENV register. (Trap handlers, the only exception to this rule, are discussed in the following section.)

  The B00 TAL compiler reports warning messages when it detects an equivalence to L[-1] (the location of the saved ENV register in the stack marker) in the declaration of variables.
- TAL programs that contain private trap handlers (by calling ARMTRAP) may have to be modified. Trap handlers also require that the stack be one word larger.

Compatibility problems within existing program files can be detected with CODEY, a program supplied with the B00 release. CODEY is a utility that detects programs that inspect or modify the CC or RP bits in the stack-marker copy of the ENV. It searches program files for references to variables at L[-1] and displays a warning if the CC or RP (bits 10-15) are used. Any warnings reported should be closely examined to determine if the use of the ENV works correctly on a B00 system.

CODEY has the option of checking one file, a subvolume, or all program files on a volume. It also has the ability of dumping selected code or displaying procedure names within a program file. CODEY can be run from $<volume>.GUARD2.CODEY. Documentation is on $<volume>.GUARD2.CODEYDOC. The HELP command within CODEY is also useful.

Trap Handlers

DEBUG is the default trap handler for all processes; however, a program can run with a private trap handler by calling ARMTRAP, as follows:

```
Call ARMTRAP (<traphndlr-addr>,
              <trapstack-addr>);
```

In multisegment program files, the call to ARMTRAP to initially arm a trap handler must be in the same segment as the trap handler code. A simple way to insure this is to put the call to ARMTRAP in the trap-handler procedure:

```
PROC arm^the^traphndlr;
BEGIN
  ENTRY trap^handler; !Entered here to initially
                        set up the trap handler.
  CALL ARMTRAP(@trap^handler,
               $LMIN( LASTADDR,%77777 )
               - 350 );
  RETURN;
trap^handler:
   .
   .
   .
END; !arm^the^traphndlr
```

Existing trap handlers work on B00 NonStop systems unchanged. A trap handler might need to be modified to run on these systems only if the programmer wants to access the SPACEID located at L[-5] in the trap handler's stack. The pseudomarker for the trap handler is allocated as follows:

Trap-handler pseudomarker	Allocation
Li[-6]	<trapstack-addr>
Li[-5]	Stack marker ENV with SPACID at trap
Li[-4]	Mask register
Li[-3]	S at the time of trap
Li[-2]	P at the time of trap
Li[-1]	Hardware ENV at the time of trap
Li[0]	L at the time of trap
Li[+1]:Li[+8]	Registers (R0 through R7)

Because one word was added to the pseudomarker in the trap handler's stack, the amount of space available to the trap handler is reduced by one word. This has an impact only on those programs that allocate exactly the number of words necessary in the trap handler's stack.

To insure that a trap handler can execute on all systems, the word referenced by <trapstack-addr> must not be used.

DEBUG, INSPECT, and XRAY

DEBUG, the operating-system debugging facility, has been enhanced to support multisegment program files.

INSPECT, the interactive symbolic debugger, also handles multisegment program files. A low-level-command syntax has been extended to allow references to specific segments within the user-code and user-library spaces. The high-level syntax is unchanged. References to code blocks are mapped to the correct user-code or user-library segment.

The XRAY performance measurement tool has been changed to adapt procedure- or code-range measurement to user-code or user-library spaces. All existing key words except SYSTEM LIBRARY CODE MAP are accepted. New key words allow the measurement of a specific segment within the user-code or user-library space.

SYSGEN

The system-code area now consists of up to 32 segments in the system-library space and one segment in the system-code space. GUARDIAN 90 has used only three segments of the possible 33 segments: SC.0, SL.0, and SL.1. The number of segments required will change with each release of the operating system.

In earlier versions of GUARDIAN, SYSGEN builds one work file for each of the two system-code spaces, SYSGEND and SYSGENE. SYSGEN (using BINDER) now builds as many work files as necessary to contain all of the system procedures. The single system-code segment (SC.0) is built in the work file SYSCOD00. The system-library segments (SL.0 - SL.37) are built in the work files SYSLIB00 through SYSLIB37.

SYSGEN builds the system code and library segments from the object files named in TANDEM^SYSTEM^LIBRARY^FILES in the CONFAUX file. SYSGEN uses the order of the object files in this list to determine the segment to which the code should be added (SL.0 or SL.1). The most frequently referenced procedures are in object files at the front of the list; infrequently called procedures are at the end of the list. With infrequently called procedures located in SL.1, the number of map switches is minimized since SL.0 is usually mapped.

Conclusion

The support of multisegment programs in the GUARDIAN 90 operating system has greatly increased the maximum size of a program (to 4M bytes). This new capability obviates the need to break up large applications into several processes that must communicate via the Message System.

The enhancement to the GUARDIAN 90 operating system is a standard one, preserving the forward compatibility of existing applications. The implementation has minimal impact on the way users currently create and maintain applications, requiring little user education.

Acknowledgments
Dennis Aiello, Marsha Brewer, Mala Chandra, Jeff Lichtman, Don Meyer, Bob Montevaldo, Carl Niehaus, and Ann Whitesell provided valuable review comments. A special thanks to Cindy Sidaris who provided technical expertise and was largely responsible for this article.

Arthur Jordan has supported the GUARDIAN operating system in the Corporate Support Services group since 1983.

New GUARDIAN 90 Timekeeping Facilities

Extensive enhancements to the timekeeping services offered by the GUARDIAN 90 operating system have been added in the B00 software release.[1] These features include:

- Four-word, microsecond-resolution timestamps based on the Julian date (GMT).
- CPU clock-rate averaging.
- Clock-rate adjustment.
- Automatic Daylight Savings Time (DST) adjustments.
- Julian-date conversion routines.
- A callable procedure to set system clocks.
- An optional IN file for the cold-load Command Interpreter.

This article discusses these new timekeeping features and the rationale behind them.

[1]The new process-timing features of the B00 release are not part of the GUARDIAN 90 timekeeping facilities, and thus, are not discussed here. See the article, "New Process-timing Features," by Sunil Sharma, for a discussion of this feature.

Terminology

Knowledge of the following definitions is helpful for understanding the discussion:

- *Greenwich Mean Time (GMT)* is the popular name for UTC, Coordinated Universal Time.
- *Local Standard Time (LST)* is GMT adjusted by the time-zone difference for the local time zone. It does not include DST.
- *Local Civil Time (LCT)* is Local Standard Time, including the adjustment for DST.
- *Daylight Savings Time (DST)*, also Summer Time, is a system that extends the amount of daylight in the evenings by advancing the civil time. Usually, but not always, this is done periodically in hour increments. In the United States, DST begins at 2:00 A.M. on the last Sunday in April and ends at 3:00 A.M. (DST) on the last Sunday in October. The United States advances the time by one hour.
- *Julian Day Number (JDN)* is the integral number of days since B.C. 4713 January 1, according to the Julian proleptic calendar. The formal definition of the JDN states that the JDN starts at noon, Greenwich. For simplicity, the GUARDIAN 90 timekeeping features assume the JDN starts at midnight local or Greenwich time, depending on the base of the timestamp.
- *Gregorian Calendar* is the common civil calendar. It was instituted in 1582 by Pope Gregory XIII. It has been adopted by and is in use as the civil calendar of almost all countries.

In addition, the following terms are used in this article:

- *GUARDIAN 90 timekeeping facilities* refers to one or all of the enhanced timekeeping features provided in the B00 release of GUARDIAN 90. It does not refer to a system process; there is no single GUARDIAN 90 timekeeping process.
- *Three-word timestamp* or *old timestamp* refers to the three-word timestamp used by the TIME and CONTIME procedures. This timestamp represents the number of centiseconds (.01 second) since 00:00 December 31, 1974.
- *Julian timestamp, four-word timestamp*, or *new timestamp* refers to the four-word, microsecond-resolution timestamp based on the Julian Date that is provided by the new timekeeping facilities. Its value represents the number of microseconds since B.C. 4713 January 1, 12:00 (Julian proleptic calendar), GMT.

Motivation for the New Facilities

As Tandem's software products and customer applications have become more sophisticated, and particularly as networks have been implemented across continents and around the world, a need has evolved for more extensive timekeeping facilities than those offered by the GUARDIAN operating system. The GUARDIAN 90 timekeeping facilities are Tandem's response to this need.

The new four-word timestamps, in addition to providing greater resolution, are easier to manipulate. Computations using the new timestamps are performed simply by treating the timestamps as decimal (quad) quantities.

Applications can now enjoy the support of GMT, conversion to and from Local Civil Time or Local Standard Time, automatic correction of Local Civil Time for Daylight Savings Time adjustments, and the ability to determine the local time at other nodes.

A procedure to set the system clock provides for the interrogation of an external clock and the setting of system time accordingly. It may be used simply to avoid having an operator initialize the system time after a cold load. Electronic clocks that monitor government broadcasts of standard time (WWV in the United States) are available.[2] Use of such a clock provides a method for synchronizing the system clocks of geographically distributed Tandem systems.

Compatibility with Existing Facilities

The GUARDIAN 90 timekeeping facilities are upwardly compatible; no changes are necessary to existing programs that use earlier GUARDIAN timekeeping facilities.

Support of Existing Procedures

The existing GUARDIAN TIMESTAMP and CONTIME procedures (which use the old three-word timestamp) and the TIME procedure will continue to be supported.

Use of the RCLK Instruction

The RCLK instruction returns a four-word integer, representing the number of microseconds that have elapsed since 00:00 December 31, 1974. Some programs attempt to use this instruction for elapsed-time measurements. This method is potentially unreliable, as processes are subject to interruption, unless they are privileged and running under interruption. Users are encouraged to employ the new B00 process-timing facilities, specifically the procedure MYPROCESSTIME, instead of using RCLK.

[2] WWV are the call letters of the U.S. government radio station. This station broadcasts the time, both in an audio message and in binary representation.

New JULIANTIMESTAMP Procedure

One of the key additions to the timekeeping facilities is the JULIANTIMESTAMP procedure. It returns a four-word, microsecond-resolution timestamp based on the Julian date (GMT). The external declaration for this procedure is defined in Figure 1.

Figure 1

```
FIXED PROC JULIANTIMESTAMP (TYPE, TUID) EXTENSIBLE, CALLABLE;
   INT          TYPE;
   INT          .TUID;
   EXTERNAL;

FIXED PROC COMPUTETIMESTAMP (DATE^N^TIME, ERRORMASK) EXTENSIBLE;
   INT      .DATE^N^TIME;
   INT      .ERRORMASK;      ! Optional, Returned
   EXTERNAL;
```

Figure 1.
The external declarations for the new procedures JULIANTIMESTAMP and COMPUTETIMESTAMP.

TYPE is an optional parameter; if supplied, it indicates which timestamp the caller is requesting:

Type	Julian timestamp returned
0	Current GMT (default)
1	System load GMT
2	SYSGEN GMT

TUID is also an optional parameter; if it is supplied, a "time-update ID" is returned in TUID. The ID can be used with the procedure SETSYSTEMCLOCK, described later. The time-update ID is used internally for consistency checking; it has no external use. (The timestamp base conversion provided by the new timekeeping facilities is described later.)

Clock Adjustments

The new timekeeping systems differ significantly from previous GUARDIAN systems in how clocks are set and synchronized, how the user obtains and interprets timestamps, and how processes can obtain clock-adjustment information.

Clock Setting

A20 GUARDIAN provides only one (nonprivileged) method for setting system clocks: the Command Interpreter (CI) command SETTIME. GUARDIAN 90 timekeeping facilities include a callable procedure, SETSYSTEMCLOCK. Also, the SETTIME command has been enhanced to allow an additional field with which the user can specify the time base, GMT or LCT. The SETSYSTEMCLOCK procedure sets the system clock only if called by a process running under the SUPER group ID. Similarly, the SETTIME command can be executed only by SUPER group users.

Clock Averaging

A20 GUARDIAN systems synchronize the system clock in the CPUs within a system by making forward adjustments of the slower clocks. This has the effect of having the system time follow the fastest clock in the system. Since one expects a normal distribution of clock rates around the nominal, one expects this method to result in fast-running system clocks.[3]

In the GUARDIAN 90 timekeeping systems, the clocks of the various CPUs in a given system are averaged. The result should be system-clock rates much closer to nominal, especially in systems with a large number of CPUs.

Clock-rate Adjustment

The new timekeeping systems provide a user-callable procedure, SETSYSTEMCLOCK, to set and/or adjust the system clocks. If the system time is set, a SETTIME message is sent to system processes and those user processes that have requested "new" system messages via MONITORNEW. These processes can then respond appropriately to the setting of the clock.

[3] *Nominal* means that which is named, the named value, the value used for design purposes. For example, crystals used in crystal-controlled electronic oscillators are designated as having a certain nominal frequency and precision, e.g., 180 MHz ± .01%. The nominal clock rate of most clocks is one second per second.

If the requested time change is small, i.e., less than two minutes, the timekeeping facilities may decide to temporarily adjust the clock rates to achieve the same result. Adjustment of clock rates is transparent to all processes.

Users who have an external clock will find this feature useful. A process can poll the external clock at regular intervals (every hour, for example), and call SETSYSTEMCLOCK to synchronize the system clocks with the external clock. Presumably, the clocks will differ very little, and the new timekeeping facilities will adjust the system clocks rather than reset them.

Daylight Savings Time Adjustment

A20 GUARDIAN timekeeping systems require the system operator to set the system clock. Most sites set their clocks to the Local Civil Time, and when a Daylight Savings Time transition occurs, it is necessary for someone to remember to use SETTIME and reset the system clocks.

With the GUARDIAN 90 timekeeping facilities, the system can be configured to automatically apply the DST correction, by using either the standard U.S. 1966 rules or a table of arbitrary DST transitions defined by the user.

The Local Civil Time, which includes any DST adjustments in effect, can be retrieved via the CONVERTTIMESTAMP procedure. (The results of an RCLK instruction executed in the vicinity of a DST change are unpredictable, however. RCLK's result is adjusted for the DST transition by the SYSTEM MONITOR soon after the DST change. It is not possible to guarantee exact correspondence to the microsecond.)

SETTIME Message Format

Five words have been added to the SETTIME (message type -10) system message to allow processes to determine the magnitude of and reason for a time change. The words added are:

```
<sysmsg>[2] FOR 4 = signed change in micro-
                    seconds (FIXED integer).
<sysmsg>[6]       = reason
```

The reason codes are:

0 = Initial SETTIME, GMT & local changed
1 = Subsequent SETTIME, GMT & local changed
2 = DST change, local only changed.

Time and Date Conversion Procedures

Time-conversion Procedures

The new timekeeping facilities provide users of the GUARDIAN 90 operating system with three conversion routines for manipulating Julian timestamps. The first computes a Julian timestamp from an integer array representing the date and time. The second converts a Julian timestamp to an integer array representing the date and time. The third routine converts Julian timestamps from GMT to local time or vice versa.

The first two procedures reference an integer array that represents the date and time. The array has eight elements containing:

Element	Contents
Element[0]	Gregorian year (1984, 1985, ...)
Element[1]	Gregorian month (1-12)
Element[2]	Gregorian day of month (1-31)
Element[3]	Hour of the day (0-23)
Element[4]	Minute of the hour (0-59)
Element[5]	Second of the minute (0-59)
Element[6]	Millisecond of the second (0-999)
Element[7]	Microsecond of the millisecond (0-999)

Note that the range of the year is restricted: 1 <= Year <= 4000.

COMPUTETIMESTAMP Procedure. This procedure (see Figure 1) computes a Julian timestamp from an integer array that represents a Gregorian date and the time of day.

DATE^N^TIME is a required parameter supplied by the caller; it is an array of eight elements containing the year, month, day, etc., as described above.

ERRORMASK is an optional parameter. If supplied, COMPUTETIMESTAMP checks each element of DATE^N^TIME for validity and sets a bit corresponding to each element of DATE^N^TIME that was out of range; e.g., ERRORMASK = %100000 means the year was out of range.

INTERPRETTIMESTAMP Procedure. This procedure (see Figure 2) converts a Julian timestamp to an array of integers representing the same Gregorian date and time of day. In addition, it returns (as its value) the 32-bit Julian day number.

JULIAN^TIMESTAMP is a required parameter supplied by the caller; it should contain a valid Julian timestamp.

DATE^N^TIME is an array of eight elements containing the year, month, day, etc., as described above. It is returned by INTERPRETTIMESTAMP and contains the equivalent of the JULIAN^TIMESTAMP value.

The value returned by the procedure is the Julian day number of the Julian timestamp.

CONVERTTIMESTAMP Procedure. This procedure converts a Julian timestamp to or from a local Julian timestamp at any accessible node in the network. The external declaration is given in Figure 2.

JULIAN^TIMESTAMP is a required parameter supplied by the caller; it should contain a valid Julian timestamp.

DIRECTION is a parameter optionally supplied by the caller. It specifies which conversion is requested and may take the following values:

Value	Conversion
0	GMT to LCT (default)
1	GMT to LST
2	LCT to GMT
3	LST to GMT

NODE is an optional parameter supplied by the caller. It is used to specify the node for which the conversion is requested; it defaults to the local node.

ERROR is an optional parameter. If it is supplied, CONVERTTIMESTAMP returns the following values:

Value	Error
-2	Impossible LCT
-1	Ambiguous LCT
0	OK
1	DST Range in doubt
2	DST Table not loaded
>2	FS error to NODE

CONVERTTIMESTAMP returns a Julian timestamp in the requested base.

Julian-date Conversion Procedures

The new timekeeping facilities provide users of GUARDIAN 90 with two conversion routines for converting from Gregorian calendar dates to Julian day number and vice versa.

COMPUTEJULIANDAYNO Procedure. This procedure computes the Julian day number from a Gregorian calendar date on or after 01 January 0001. The Gregorian calendar date must be valid. The external declaration of the procedure is given in Figure 2.

YEAR, MONTH, and DAY are required parameters and contain the Gregorian year, month, and day of month, respectively.

ERROR is an optional parameter. If supplied, COMPUTEJULIANDAYNO checks each element of DATE^N^TIME for validity and sets a bit corresponding to the parameter whose value was out of range. Bit 0 indicates YEAR was out of range, bit 1 corresponds to MONTH, and bit 2 to DAY.

COMPUTEJULIANDAYNO returns a Julian day number.

INTERPRETJULIANDAYNO Procedure. This procedure (see Figure 2) converts a Julian day number to the year, month, and day in the Gregorian calendar. The Julian day number must be greater than 1,721,119 (00 March 0000).

JULIANDAYNO is initialized by the caller; it should contain a valid Julian day number.

YEAR, MONTH, and DAY are used by INTERPRETJULIANDAYNO to return the equivalent Gregorian year, month, and day respectively.

Setting the System Clocks

The GUARDIAN 90 timekeeping facilities provide two methods of setting the system clocks. The SETTIME command for the CI has been modified, and a new callable procedure, SETSYSTEMCLOCK, has been added to the system libraries.

SETTIME Command

An optional field, the timebase, has been added to the SETTIME command of the CI. The valid values are GMT, LCT, and LST. They may be upper, lower, or mixed case. The default continues to be LCT.

Two new error messages have been added: *Ambiguous Time Specification* and *Impossible Time Specified*. These messages are output if the LCT time specified is ambiguous or impossible because of a DST transition. (In U.S. areas where DST is used, 02:30 on the last Sunday in April is impossible and 2:30 on the last Sunday in October is ambiguous.)

In addition, users can specify the time to the second, e.g.:

SETTIME 10 Aug 1984, 12:13:14 GMT

SETSYSTEMCLOCK Procedure

The new SETSYSTEMCLOCK procedure (see Figure 2) allows the SUPER group or privileged caller to change the system clock.

JULIANGMT is supplied by the caller. It contains the GMT, in JULIANTIMESTAMP form, to which the system clock is to be set.

MODE is supplied by the caller. It describes the mode and source:

Value	Mode	Source
0	Absolute GMT	Operator input
1	Absolute GMT	Hardware clock
2	Relative GMT	Operator input
3	Relative GMT	Hardware clock

The relative mode implies that the parameter JULIANGMT contains a microsecond correction, not an actual timestamp. This is useful for very precise time synchronization with a hardware clock or for a moderately precise method of operator time adjustment.

Figure 2

```
INT(32) PROC INTERPRETTIMESTAMP (JULIAN^TIMESTAMP, DATE^N^TIME);
    FIXED       JULIAN^TIMESTAMP;
    INT         .DATE^N^TIME;
    EXTERNAL;

FIXED PROC CONVERTTIMESTAMP (JULIAN^TIMESTAMP, DIRECTION, NODE, ERROR)
                                        EXTENSIBLE, CALLABLE;
    FIXED       JULIAN^TIMESTAMP;
    INT         DIRECTION;
    INT         NODE;
    INT         .ERROR;
    EXTERNAL;

INT(32) PROC COMPUTEJULIANDAYNO (YEAR, MONTH, DAY, ERROR) EXTENSIBLE;
    INT         YEAR;
    INT         MONTH;
    INT         DAY;
    INT         .ERROR;
    EXTERNAL;

PROC INTERPRETJULIANDAYNO (JULIANDAYNO, YEAR, MONTH, DAY);
    INT(32)     JULIANDAYNO;
    INT         .YEAR;
    INT         .MONTH;
    INT         .DAY;
    EXTERNAL;

PROC SETSYSTEMCLOCK (JULIANGMT, MODE, TUID) EXTENSIBLE, CALLABLE;
    FIXED       JULIANGMT;
    INT         MODE;
    INT         TUID;
    EXTERNAL;
```

TUID is optional. If supplied by the caller, it should contain a time update ID obtained by calling JULIANTIMESTAMP. TUID is recommended when modes 2 and 3 are used to avoid conflicting changes. The resulting condition codes imply:

Code	Meaning
>	TUID mismatch; retry after redetermining relative error.
=	Time set as requested.
<	Insufficient capability.

Figure 2.
The external declarations for the new procedures INTERPRETTIMESTAMP, CONVERTTIMESTAMP, COMPUTEJULIANDAYNO, and SETSYSTEMCLOCK.

Daylight Savings Time Considerations

The new timekeeping facilities require knowledge of Daylight Savings Time transitions for the system's location. This information allows them to compute the Local Civil Time from GMT and vice versa. For example, if an operator cold loads the system and does a SETTIME using the Local Civil Time, the system calculates GMT.

One of three options can be specified at SYSGEN: NONE, TABLE, or USA66. They are described below:

Option	Effect
NONE	DST is not observed at the system's location.
USA66	The rules used in the United States since the adoption of the Uniform Time Act of 1966 are to be followed.
TABLE	A table of DST transitions are to be loaded at system-cold-load time.

DST Table Loading Procedure

The new facilities include the callable procedure ADDDSTTRANSITION (see Figure 3) that allows the SUPER group caller to add an entry to the DST Transition table.

Figure 3

```
PROC ADDDSTTRANSITION (LOWGMT, HIGHGMT, OFFSET) EXTENSIBLE, CALLABLE;
   FIXED     LOWGMT;
   FIXED     HIGHGMT;
   INT       OFFSET;
   EXTERNAL;
```

Figure 3.
The external declaration for the new procedure ADDDSTTRANSITION.

LOWGMT contains a Julian timestamp that is the GMT when OFFSET is first applicable.

HIGHGMT contains a Julian timestamp that is the GMT when OFFSET is no longer applicable.

OFFSET contains the difference in seconds between Local Civil Time and Local Standard Time (LCT = LST + OFFSET).

The DST Table must be loaded in time sequence and with no gaps; i.e., except for the first call, the LOWGMT of each call must be the same as the HIGHGMT of the previous call. Thus, many calls have an OFFSET parameter of zero.

The DST table must be initialized with at least one DST transition that is less than the current date and time, and at least two DST transitions that are greater than the current date and time. If this is not done before the SETTIME is entered, the error message *Incorrect Daylight Savings Time Conversion* is displayed when the SETTIME is entered.

Note that CONVERTTIMESTAMP assumes that transitions are separated by one day or more.

ADDDSTTRANSITION Command

A new command, ADDDSTTRANSITION, has been added to the CI to allow SUPER group users to add entries to the DST Transition Table. The syntax of the new command is shown in Figure 4. The limits and sequence restrictions for the ADDDSTTRANSITION procedure also apply for this command.

SYSGEN Changes

Three new clauses have been added to the ALLPROCESSORS paragraph of the SYSGEN input:

- INITIAL_COMINT_INFILE (Optional)
- TIME_ZONE_OFFSET (Required)
- DAYLIGHT_SAVINGS_TIME (Required)

Cold-load Command Interpreter IN File

An optional input file specified at SYSGEN has been added for the initial (start-up) Command Interpreter. If this option is specified in the SYSGEN input, the file must exist. It is copied to the SYS*nn* subvolume and named CIINFILE. The syntax of the clause is:

INITIAL_COMINT_INFILE <filename>

Sites having an external clock that choose to use the CI IN file option can automate all of the system-restart activity. After the cold load is complete, the CI IN file is executed by the cold-load CI, running under the SUPER.SUPER user ID.

The CI IN file typically performs the following activities:

1. Sets the system time by executing a program that reads the external clock and sets system time. A sample program, SCLOCK, is provided. It is recommended that the system clock be set before any other activity is started, only to ensure that other processes retrieve valid timestamps.
2. Performs various system start-up functions, such as starting the spooler, starting Command Interpreters, and perhaps starting the Transaction Monitoring Facility (TMF) and the PATHWAY transaction processing system.
3. Starts another CI on the Operations and Services Processor (OSP). This is because the initial cold-load CI terminates when it has finished reading the CI IN file.

Note that using a CI IN file does not leave a CI running and logged on under SUPER. SUPER, as is the case when the CI IN file configuration option is not used.

Figure 4

ADDDSTTRANSITION <start-date-time>, <stop-date-time>,<offset>

<Start-date-time> is the beginning of the period when <offset> is applicable and <stop-date-time> is the end of the period when <offset> is applicable. The format of <start-date-time> and <stop-date-time> is:

{ <month-name> <day> } <year> , <hour>:<min>[:<sec>] { GMT }
{ <day> <month-name> } { LST }

<offset> is the difference between standard time and Daylight Savings time and is of the form:

[+] <hour>:<min>
[-]

Note that <offset> must be between - 8:59 and + 8:59.

Examples

ADDDSTTRANSITION 28 OCT 1984, 02:00 LST, 28 APR 1985, 02:00 LST, + 0:00

ADDDSTTRANSITION 28 APR 1985, 02:00 LST, 27 OCT 1985, 02:00 LST, + 1:00

ADDDSTTRANSITION 27 OCT 1985, 02:00 LST, 27 APR 1986, 02:00 LST, + 0:00

Figure 4.
The syntax of the new ADDDSTTRANSITION command for the Command Interpreter. This command allows SUPER group users to add entries to the DST Transition Table.

LST Offset from GMT

The GUARDIAN 90 timekeeping facilities require that the specification of the offset of LST from GMT be made in hours and minutes. The offset must be specified in the ALLPROCESSORS clause as

TIME_ZONE_OFFSET + hh:mm

or

TIME_ZONE_OFFSET - hh:mm

where *hh* is an unsigned integer less than 24 and *mm* is an unsigned integer less than 60. The following are some examples:

Offset as specified in ALLPROCESSORS clause	City
TIME_ZONE_OFFSET + 01:00	Paris
TIME_ZONE_OFFSET + 05:30	Bombay
TIME_ZONE_OFFSET + 09:00	Tokyo
TIME_ZONE_OFFSET - 08:00	San Francisco
TIME_ZONE_OFFSET - 05:00	New York

Daylight Savings Time Selection

With the new facilities, the selection of one of three options for Daylight Savings Time is required: none, a table-driven method, or the rules followed in the United States since 1966. As other algorithmic rules are required, the appropriate options will be added.

The new clause to the ALLPROCESSORS paragraph is:

```
DAYLIGHT_SAVINGS_TIME { NONE | USA66 |
TABLE}
```

Command Interpreter Changes

The new timekeeping facilities have affected the Command Interpreter in four areas:

- An additional field in the SETTIME command has been added.
- The ability to specify the initial CI's input file has been added.
- A new command to add an entry to the DST Transition Table has been added.
- Command execution is now restricted before the first SETTIME.

All but the last modification have already been discussed.

The initial (cold-load) CI executes only ADDDSTRANSITION, SETTIME, and HELP commands until the first SETTIME command has been entered successfully. Other commands result in the message, *Please Set System Time.* If the system was configured with an initial CI IN file (using the INITIAL_COMINT_INFILE SYSGEN parameter), however, there is no restriction on commands executed from the IN file. This allows the IN file to start a process that communicates with an external clock and then calls SETSYSTEMCLOCK to set the system time.

Sample External Clock Reading Process

TAL source code for a sample user process that reads an external clock is supplied with GUARDIAN 90. This program, SCLOCK, reads the external clock and sets the system clock immediately, every five minutes, and whenever a CPU is powered on.

Benefits of the New Facilities

The new timekeeping facilities significantly enhance the timekeeping services of GUARDIAN 90 in several areas.

Application Design

The use of GMT timestamps facilitates the design of applications in a geographically distributed network. Procedures that convert timestamps between GMT, LST, and LCT, on both local and remote nodes, as well as being aware of Daylight Savings Time, simplify the design of global-network applications.

The enhancements to the SETTIME message permit applications to detect when and why the system clock was reset and determine the magnitude of the change.

The microsecond resolution of Julian timestamps makes them more useful for identification of events. If two events occur within .01 second, the old TIMESTAMP procedure might have returned the same value to both, making the timestamp useless for determining the sequence of events.

Programming
Four-word Julian timestamps and Julian date conversion routines simplify programming tasks that involve timekeeping.

System Management
Automatic adjustment for Daylight Savings Time changes eliminates the need for an operator to do a SETTIME when these changes occur, reducing potential operator errors.

The ability to read an external clock and set system time completely eliminates the possibility of the operator's making an error when setting the system time. Users that have an external clock and choose to use the CI IN file option can automate all system restart activity.

Finally, the new algorithm for CPU clock-rate averaging should provide more accurate system times than those provided by previous releases.

Conclusion

GUARDIAN 90 timekeeping services contribute significantly to the effectiveness of Tandem systems for applications that require accurate timekeeping and for geographically distributed networks.

Acknowledgments
Glenn Peterson is responsible for the design, documentation, and initial implementation of the GUARDIAN 90 timekeeping facilities.

Eric Nellen joined Tandem in February 1979 as a member of the Software Quality Assurance group. He designed and implemented the first Tandem terminal simulator, which was used for QA testing and benchmarks. He has worked in operating systems development for several years and is currently a member of the OS Kernel group.

New Process-timing Features

The GUARDIAN 90 operating system incorporates greatly enhanced features for process timing.[1] Earlier versions of GUARDIAN provide two facilities for process timing: the XRAY performance measurement tool and the procedure SETLOOPTIMER. Both have shortcomings:

- XRAY is cumbersome to use, and the overhead of starting an XRAY measurement is not justified for small measurements such as obtaining the execution time of a single process. Moreover, XRAY does not have a programmatic interface.
- SETLOOPTIMER, a programmatic interface, is designed only for detecting loops in a program. In addition, in earlier versions of GUARDIAN, the SETLOOPTIMER mechanism includes interrupt time and is inaccurate due to rounding errors.

GUARDIAN 90 provides process-timing facilities in addition to XRAY and vastly improves the accuracy of the SETLOOPTIMER mechanism. Process-execution time is now a fundamental part of the process state. This permits the following additional functionality:

- Any process can query the operating system for the process time of any process, including its own.[2]
- A process can specify that it be notified via a system message when it has executed for a certain amount of time.
- Some processor-utilization statistics (amount of time spent by the processor executing processes, servicing interrupts, and remaining in the idle state) can be obtained.
- Accuracy of the SETLOOPTIMER mechanism has been improved because interrupt-servicing time is excluded and microsecond resolution is used.

User Interface

The following procedures are now available for the timing of processes: SETLOOPTIMER, SIGNALTIMEOUT, SIGNALPROCESSTIMEOUT, MYPROCESSTIME, and PROCESSTIME. In addition, a new procedure called CPUTIMES is available to provide utilization statistics for a processor. (See the *System Procedure Calls Reference Manual* for details).

[1] This article describes the GUARDIAN 90 features that can be used to time individual processes. New GUARDIAN 90 features for maintaining the system date and time of day are described in an accompanying article, "New GUARDIAN 90 Timekeeping Facilities," by Eric Nellen.

[2] In this article *process time* signifies the length of time during which the process executes on the processor. It does not include the time taken by software-interrupt handlers to process interrupts that occur while the process is running. The unit of measure for process time is microseconds.

The functions of the procedures are briefly described below:

- *SETLOOPTIMER* now sets a timer that is based on process time. When the timer times out, it triggers a *process-loop-timeout trap* (trap number 4). Control then transfers to the user trap handler (if specified) or to DEBUG/INSPECT.
- *SIGNALTIMEOUT* sets an elapsed-time timer. When the timer times out, a system message (see the *System Messages Manual*) is queued on the $RECEIVE queue of the process (see the *System Procedure Calls Reference Manual*). The functionality and the underlying mechanism for this procedure have not changed in GUARDIAN 90.
- *SIGNALPROCESSTIMEOUT*, a new procedure, is similar to SIGNALTIMEOUT but is based on process time instead of elapsed time. It sets a process-time timer. When the timer times out, a system message is queued on the $RECEIVE queue of the process.
- *MYPROCESSTIME*, a new procedure, gives the process time (in microseconds) of the calling process.
- *PROCESSTIME*, a new procedure, gives the process time (in microseconds) of any process in the network that runs on a system operating under GUARDIAN 90. The procedure PROCESSINFO can also be used to obtain the process time of a process.
- *CPUTIMES*, a new procedure, gives the following processor utilization statistics (in microseconds, since the last processor load) of any processor in the network that is operating under GUARDIAN 90: the amount of time since the processor loaded and the amount of time spent by the processor executing processes, servicing interrupts, and remaining in the idle state.

Implementation

The GUARDIAN 90 kernel and data structures maintain accurate process-execution time as an intrinsic part of the process state. They incorporate an internal facility to cause an event to occur, based on the amount of process-execution time that has passed. This facility is used to support the following:

- *SETLOOPTIMER* procedure (whose corresponding event is to cause a trap after a specified amount of process-execution time).
- *SIGNALPROCESSTIMEOUT* procedure (whose corresponding event is to cause a system message to be queued on the $RECEIVE queue of the calling process after the process has executed for a specified amount of time).
- *Process-priority adjustment mechanism.*

The facility may also be used in the future to support additional capabilities. For example, the parent process could request the operating system to notify it when a specific child process has executed for a certain amount of time. (This feature is not present in GUARDIAN 90 currently but is an example of something that could now be easily implemented.)

Each process has a 32-bit counter (referred to in this article as PCBPTIMER) that is part of the process state. When a process is inactive, PCBPTIMER is stored in the field PCBPTIMER in the Process Control Block (PCB) for the process. (The layout for the PCB is given in the GUARDIAN 90 file DPCTL, listed in CDECLARE.) When the process is dispatched, the dispatcher uses the new Set Process Time (SPT) instruction to give PCBPTIMER to the firmware. The cached firmware value, which represents the current value of PCBPTIMER, can be read via the new Read Process Time (RPT) instruction. The SPT and RPT instructions are defined in the *System Description Manual*.

PCBPTIMER accumulates process-execution time as the process executes and is adjusted for any processor time used by software interrupt handlers. If PCBPTIMER overflows from a positive to a negative number, a *process-timeout* dispatcher interrupt occurs.

Every time the dispatcher deactivates a process, the dispatcher accumulates the elapsed process time for the last dispatch to a quad-word counter (PCBXPTIME) in the Process Control Block Extension (PCBX) for the process. The dispatcher also accumulates the elapsed process time to a quad-word counter called SUMPROCESSBUSY in the system data segment. This counter contains the total processor time spent in executing processes. (The elapsed process time is the difference between the cached value of PCBPTIMER and the saved value in the PCBPTIMER field in the PCB.) The dispatcher also saves the cached value of PCBPTIMER in the field PCBPTIMER of the PCB for the process.

Thus, the purposes of PCBPTIMER are to cause a dispatcher interrupt after a specified amount of process-execution time and to track elapsed process time in a dispatch. The purpose of PCBXPTIME is to accumulate process-execution time as the process executes. It will overflow only if the process executes for over .29 million years.

The unit of measure for both counters is microseconds. At any instant, the process time of the process is given by the sum of the PCBPTIMER and PCBXPTIME. When the process is active, PCBPTIMER is cached in a processor register and can be accessed via the new instructions; otherwise, the PCBPTIMER is stored in the PCB field PCBPTIMER.

Each process has a doubly-linked list, called the process-time list (PTL), composed of process-time-list elements (PTLEs). A PTLE contains the process time at which the process should time out. The list header of the PTL is a field in the PCBX. The PTLEs are listed in order of increasing process-time value. Two PTLEs, one for process-priority adjustment and one for the SETLOOPTIMER procedure, are allocated in the PCBX. PTLEs for use by SIGNALPROCESSTIMEOUT are allocated out of the PTLE table, which is in the same absolute segment as the PCBXs.

For a process-timeout dispatcher interrupt, the dispatcher (1) processes each PTLE that has timed out, (2) deletes the PTLE from the PTL, and (3) takes action based on the event indicated by the PTLE type:

- If the PTLE was set by a call to SETLOOPTIMER, a trap #4 is signaled.
- If the PTLE was set by a call to SIGNALPROCESSTIMEOUT, a system message is queued on the $RECEIVE queue of the process. This message indicates to the process that the timer set by a call to SIGNALPROCESSTIMEOUT has timed out.
- If the PTLE is for system-loop timing, the dispatcher adjusts the priority of the process and requeues the PTLE on the PTL with a time value of the current process time of the process plus two seconds (the time when the priority is to be adjusted next).

The dispatcher then initializes the 32-bit counter (PCBPTIMER) so that it will overflow when the current leading PTLE in the PTL times out, and then it dispatches the process.

IDLEPROCESS executes on the processor when there is no process on the ready list. Although IDLEPROCESS is actually a pseudoprocess because it does not have a PCB and other resources allocated to it, the processor maintains execution time for it because the firmware views it as a process. The PCBPTIMER and PCBXPTIME entries for IDLEPROCESS are allocated in the system-data segment. These entries and the SUMPROCESSBUSY counter are used to support the CPUTIMES procedures.

Applications

SETLOOPTIMER can be used to (1) detect whether or not the process is looping, (2) perform a certain set of operations (defined in the user trap handler) after the process has executed for a specified amount of time, and (3) perform a set of operations periodically by having the user trap handler always call SETLOOPTIMER before returning to the interrupted operation.

SIGNALTIMEOUT can be used for scheduling real-time operations since it notifies a process when a certain amount of real time has elapsed.

SIGNALPROCESSTIMEOUT can be used for scheduling operations based on the amount of time the process has executed. It queues a system message on the process' $RECEIVE queue to notify that process after it has executed for a specified amount of time. The process can call the procedure READ to read the message whenever it wants to. (SETLOOPTIMER can also be used for this purpose; however, it causes a trap that interrupts the process and immediately transfers control to the trap handler.)

MYPROCESSTIME can be used by a process to monitor its activity and to determine the time spent by the process in performing various operations. For example, a Command Interpreter could print the time it took to execute the last command before prompting for the next command.

PROCESSTIME can be used to (1) monitor the activity of any process running on a GUARDIAN 90 system in the network, (2) detect a looping process, and (3) detect whether or not a process is a recent invocation of the program after an old invocation died, especially if the recent invocation has the same Process ID (PID) as the old process and the same name. (For unnamed processes, the timestamp form of the PID can be used for assigning a process a unique PID). See the *GUARDIAN Operating System Programmer's Guide* for information on process names.

CPUTIMES can be used to display a processor's utilization. This would help in balancing the load across the processors in the system. For example, a high interrupt-busy time in a processor might indicate that a great deal of I/O was being done through that processor. PEEK can be used to confirm this.

The information obtained from the above routines may indicate whether or not the substantial capabilities of XRAY should be used for a performance evaluation of the system or of a process that is a heavy user of processor time.

Conclusion

The process-timing features of the GUARDIAN 90 operating system are a significant enhancement. Process-execution time is maintained with microsecond resolution and interrupt-processing time is excluded. Procedures are provided to retrieve the process time of processes and to enable a process to be notified when it has executed for a specified amount of time. Also, the accuracy of the SETLOOPTIMER has been greatly improved. Finally, a procedure is now available for retrieving processor-utilization characteristics.

References

GUARDIAN Operating System Programmer's Guide. Part no. 82357 A00. Tandem Computers Incorporated.

System Messages Manual. Part no. 82409 A00. Tandem Computers Incorporated.

System Procedure Calls Reference Manual. Part no. 82359 A00. Tandem Computers Incorporated.

Acknowledgments

The basic ideas for the GUARDIAN 90 process-timing enhancements were conceived by Richard Carr. The author would like to thank Richard for his suggestions during the implementation phase, Gary Campbell and Richard Harris for making the firmware changes, and last but not least, Heidi Kuehn for greatly improving the readability of this article.

Sunil Sharma joined Tandem in August 1983 as a software developer in the Operating Systems group. He has an M.S. in Computer Engineering from Rensselaer Polytechnic Institute in Troy, New York.

Writing a Command Interpreter

Tandem has received many requests to extend the capabilities of its command interpreter, COMINT. These requests have been for a variety of capabilities, including:

- Additional commands.
- Changes in the syntax and/or functionality of existing commands.
- A history buffer of previous commands (for their reexecution or later examination).
- Macro substitution within commands.
- Assignment of commands to function keys.

Custom Command Interpreters

It is not possible to implement these new features while maintaining the compatibility between old and new versions of COMINT, however. Also, the need or desire for these new capabilities varies from customer to customer. Therefore, users requiring special features should consider developing their own command interpreters. In fact, several users have already written their own.

One drawback to most of the command interpreters written by users so far, however, is that they are *nonprivileged* and, thus, do not have the full functionality of COMINT (until now a *privileged* process).[1] Their command interpreters cannot modify the USERID file, or take a bus dump, for example. Until now, these users had to switch from their custom command interpreters to COMINT when they needed to execute a privileged function.

New Nonprivileged COMINT

With the GUARDIAN 90 operating system, COMINT no longer contains privileged code and does not have to be FUP licensed. COMINT's privileged processing has been moved into new system-library procedures, existing system procedures have been modified, and new privileged server programs have been written. COMINT now invokes these new procedures or server programs to execute privileged functions.

Users can now take advantage of these new procedures and server programs to develop their own custom command interpreters, complete with all the functionality of Tandem's COMINT.

The following sections:

- Present an overview of the flow of control in COMINT.
- Explain how to get a sample skeletal command interpreter that can be expanded and customized as needed.
- Describe COMINT (including its nowait operations and new server programs) in detail.

[1] A privileged process has direct access to the internal structures or privileged procedures of the operating system. Nonprivileged processes do not. (They cannot cause halts.) Nonprivileged programs do not need to be modified and/or recompiled when the internal structures and privileged procedures of the operating system change.

Flow of Control

COMINT was designed as a transaction-oriented process. It does the following:

1. Gets the transaction (command).
2. Deciphers the command and applies user constraints (e.g., checks to see if a user can issue the ADDUSER command).
3. Invokes the appropriate procedures or dispatches the proper program (COMINT server, Tandem subsystem, or user-developed program) to complete the processing of each command.

Figure 1 charts this flow of control.

All command interpreters should have a top-level controlling procedure. COMINT's version of this procedure, COMMAND-INTERPRETER, loops continuously on calls to read and process each command.

Command interpreters should also have a centralized procedure for getting commands. COMINT's PROMPT contains code for reading from a terminal, a disc file, or another process.

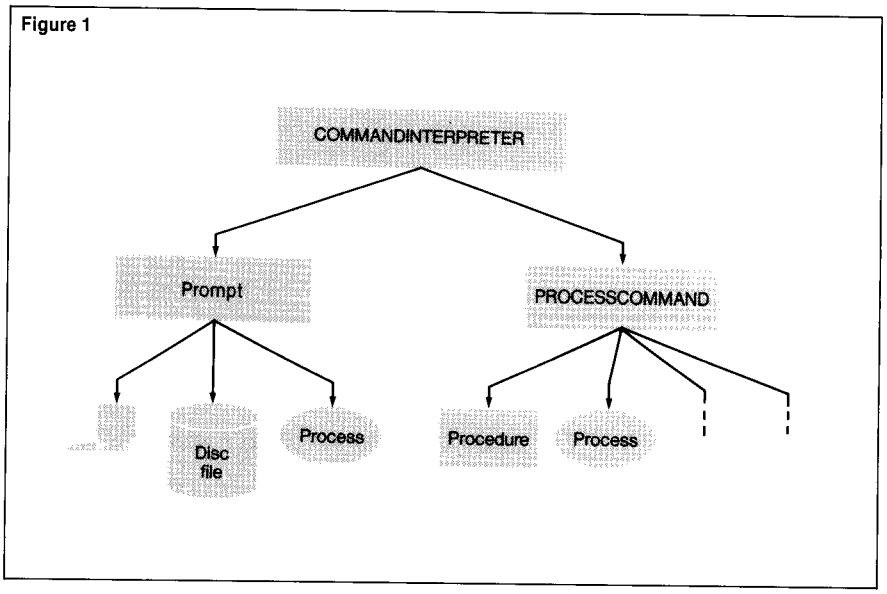
Finally, COMINT has a separate procedure, PROCESSCOMMAND, to process each command, although this is not required of all command interpreters. PROCESSCOMMAND does some initial parsing and validation checking and then invokes the appropriate procedure or program. It is then the responsibility of each procedure or program to fully parse and validate the command string or RUN parameter string before processing the command.

Simple Command Interpreter

A simple command interpreter that recognizes the EXIT, TIME, and WHO commands is available on tape from your Tandem analyst.

Note that, although the sample command interpreter was written in TAL, command interpreters can be written in any language. It is advisable, however, to develop the command interpreter in a language that provides easy access to GUARDIAN 90 system procedures, as it is through these procedures that the command interpreter will do most of its work.

Figure 1



Nowait Operations

Perhaps the most difficult aspect of developing a command interpreter is the handling of its operations in a "nowait" manner. One of the design goals of a command interpreter is that it be able to respond to messages and maintain a fault-tolerant configuration while:

- Prompting for a command.
- Communicating with another process.
- Waiting for a process to terminate.
- Writing to its OUT file.

To accomplish this, all the above operations must be done in a nowait manner, thus requiring the command interpreter to be a multithreaded process.

COMINT's solution to the problem is to perform all the above as nowait operations and to have one centralized procedure to wait for completion of these operations. This procedure calls AWAITIO with a file number of -1 so that AWAITIO returns when any outstanding I/O request completes. The centralized procedure then determines which I/O has completed and does the proper processing.

Figure 1.
The flow of control in COMINT. COMMAND-INTERPRETER, the main procedure, calls PROMPT to get a command. This command is then passed to PROCESSCOMMAND which calls/runs the appropriate procedure(s)/program to process the command. (The dotted lines indicate additional procedures/programs that process other commands.)

For example, after starting a new process, COMINT opens the process in a nowait manner. COMINT then calls a procedure to wait for the completion of the OPEN before sending the start-up message. If the user presses the BREAK key before the open completes, AWAITIO completes on $RECEIVE before completing on the OPEN. This causes COMINT to cancel the open request and terminate the start-up sequence.

COMINT Server Programs

Although major changes were made to make COMINT nonprivileged, externally little has been changed. All the commands from the previous version of COMINT are still available and perform the same function. Internally, however, the following commands now invoke the corresponding COMINT server programs:

Command	Server program
ADDUSER	ADDUSER
{X|Y}BUS{DOWN|UP}	BUSCMD
DEFAULT	DEFAULT
DELUSER	DELUSER
PASSWORD	PASSWORD
RECEIVEDUMP	RCVDUMP
RELOAD	RELOAD
REMOTEPASSWORD	RPASSWRD
USERS	USERS

When these programs are run from a user-written command interpreter or from COMINT, the following rules apply:

- All of the server programs should reside on the same subvolume as the command interpreter. (For user-written command interpreters this is not necessary, but for COMINT it is.)
- All server programs must be FUP licensed to run because they contain privileged code.
- The server programs cannot be run remotely because they contain code to prevent execution by remote users.
- All RUN options are allowed, except that NOWAIT should not be used with any of the server programs (as the results of running the servers would be unknown).

COMINT Commands

In Table 1, all of the commands recognized by COMINT are listed. For each, the function of and the GUARDIAN 90 procedures or COMINT server programs invoked by the procedure are given. (Procedures used to parse and validate command parameters are not included in the list.)

Although the list of COMINT commands is extensive, user-written command interpreters need not implement every command. It may be best to leave out the rarely used commands such as ADDUSER, BUSCMD, DELUSER, and SETTIME. These can then be issued from a running COMINT.

Note that if an error occurs at any point while the command is being processed, the command is aborted and PROMPT is called.

Note also that command strings not recognized as valid COMINT commands are treated as implicit RUN commands. For a description of COMINT's commands, refer to the *GUARDIAN Operating System Utilities Reference Manual*. For a description of each GUARDIAN 90 procedure, refer to the *System Procedure Calls Reference Manual*.

Conclusion

With the GUARDIAN 90 operating system, all of COMINT's privileged functions have been incorporated within the system or supplied in separate programs. Users can now develop their own command interpreters with all the functionality of COMINT and add extra features tailored to their needs.

References

GUARDIAN Operating System Utilities Reference Manual. Part no. 82403 A00. Tandem Computers Incorporated.

System Procedure Calls Reference Manual. Part no. 82359 A00. Tandem Computers Incorporated.

David Wong joined Tandem in April 1981, beginning work on automated release products, such as INSTALL, FACTRLSE, and CUSTRLSE. He has since transferred to the Operating Systems group where he worked on COMINT and is now working on low-level kernel code. David received a B.S. in Physics in 1974 and an M.S. in Applied Mathematics in 1978, both from Santa Clara University. Before joining Tandem, he did contract work for various research projects, including Pioneer 10 and 11 data reduction, IRAS telescope simulation, and airborne data acquisition.

Table 1.
The commands recognized by COMINT, the procedures they use, and the functions of the commands.

COMINT command	Procedures it uses	Functions of the command
ACTIVATE	ACTIVATEPROCESS	1. Parse and validate the name or <cpu,pin> of the process to activate. 2. Call ACTIVATEPROCESS on the process.
ADDDSTTRANSITION	ADDDSTTRANSITION, COMPUTETIMESTAMP, CONVERTTIMESTAMP	1. Parse and validate the limits and offset of the Daylight Savings Time period. 2. Call COMPUTETIMESTAMP to convert limits into Julian microseconds. 3. Call CONVERTTIMESTAMP to convert limits to Greenwich Mean Time. 4. Call ADDDSTTRANSITION to add the period to the Daylight Savings Time table.
ADDUSER	ADDUSER	Cause the ADDUSER server program to run. Pass ADDUSER command parameters directly as a RUN parameter string to the ADDUSER process.
ALTPRI	ALTERPRIORITY, PROCESSINFO	1. Parse and validate the name or <cpu,pin> of the process whose priority is to be changed. 2. Call PROCESSINFO for information to be sent to $CMON to log/validate the ALTPRI request. 3. Call ALTERPRIORITY to change the priority of the process.
ASSIGN		1. Parse and validate the new ASSIGN. 2. Save the information in the ASSIGN data buffer on COMINT's data stack.
BACKUPCPU	CHECKOPEN, CHECKPOINT, GETCRTPID, NEWPROCESS, PROGRAMFILENAME, STOP, SWAPFILENAME	1. Check for a backup CPU number. 2. If no CPU number is given, call STOP to stop COMINT's backup. 3. If CPU number is given, call GETCRTPID, PROGRAMFILENAME, and SWAPFILENAME to get COMINT's process name, object-file name, and swap volume so that NEWPROCESS can start the backup. 4. If the backup is successfully started, call CHECKOPEN and CHECKPOINT to synchronize the backup.
BUSCMD	BUSCMD	Cause the BUSCMD server program to run. Pass command parameters directly as a RUN parameter string to the BUSCMD process.
CLEAR		1. Parse and validate the ASSIGN(s)/PARAM(s) to clear. 2. Delete the ASSIGN(s)/PARAM(s) from their data buffer(s) on COMINT's data stack.
COMMENT		Ignored by COMINT. COMINT prompts for the next command.
CREATE	CREATE	1. Parse and validate the name of the file to be created. 2. Call CREATE to create the file.
DEBUG	DEBUG, DEBUGPROCESS, GETCRTPID, LOOKUPPROCESSNAME, MYTERM	1. Parse and validate the name or <cpu,pin> of the process to be debugged. 2. If the TERM parameter is entered without a terminal name, call MYTERM to get the name of the current home terminal. 3. If a process name was specified, call LOOKUPPROCESSNAME to find the primary process. 4. Call GETCRTPID, if the current command is being debugged, call DEBUG; otherwise, call DEBUGPROCESS to debug the specified process.
DEFAULT	DEFAULT	Cause the DEFAULT server program to be run. Pass the command parameters directly as a RUN parameter string to the DEFAULT process.
DELUSER	DELUSER	Cause the DELUSER server program to be run. Pass the command parameters directly as a RUN parameter string to the DELUSER process.
EXIT	GETREMOTECRTPID, GETSYSTEMNAME, MYPID, STOP	1. Call MYPID, GETREMOTECRTPID, and GETSYSTEMNAME to format a termination confirmation prompt. 2. Call STOP to stop COMINT (primary and backup).
FC	FIXSTRING	1. Display the old command and prompt for modifications. 2. Call FIXSTRING to update the old command. 3. Process the new command.
FILES	NEXTFILENAME	1. Parse and validate the name of the subvolume containing the files to be listed. 2. Repeatedly call NEXTFILENAME, to get the list of files in the specified subvolume.
HELP		1. Parse and validate the name of the command for which "help" information is to be displayed. 2. Display the help message for the specified command.
INITTERM	SETMODE	If the input device was a terminal, call SETMODE function 28 to reset it.
LIGHTS	SENDLIGHTS	1. Parse and validate the new lights parameters. 2. Call SENDLIGHTS to reset the lights with the new parameters.
LOGOFF	CHECKPOINT, CONTROL, SETMODE, STOP	1. Print any user messages which have been sent to COMINT. 2. Notify $CMON of a logoff and close $CMON. 3. Clear local buffers and call CHECKPOINT to synchronize the backup. 4. Simulate the TIME command to display the time. 5. If the input device was a terminal, call SETMODE function 28 to reset it. 6. If the input device was a terminal, call CONTROL request 12 on the input terminal to disconnect. 7. If this was a remote COMINT, call STOP to stop the primary and backup. 8. If the input device was a terminal, call CONTROL request 11 on the input terminal to request a reconnect.

Continued on next page.

Table 1. (Continued)
The commands recognized by COMINT, the procedures they use, and the functions of the commands.

COMINT command	Procedures it uses	Functions of the command
LOGON	CHECKPOINT, VERIFYUSER	1. If necessary, prompt for and then parse and validate the user name and password. 2. Call VERIFYUSER to log on. 3. If the user was previously logged on, notify $CMON of an implicit logoff. 4. Check with $CMON to see if logons are allowed. 5. Call CHECKPOINT to synchronize the backup. 6. Display the logon banner. 7. Simulate the TIME command to display the time.
O[BEY]	EDITREADINIT, OPEN	1. Parse and validate the new input file. 2. Call OPEN to gain access to this new file. 3. If the new input file is an EDIT file, call EDITREADINIT on it.
PARAM		1. Parse and validate the new PARAM. 2. Save the information in the PARAM data buffer on COMINT's data stack.
PASSWORD	PASSWORD	Cause the PASSWORD server program to run. Pass the command parameters directly as a RUN parameter string to the PASSWORD process.
PAUSE		1. Parse and validate the name or <cpu,pin> of the process on which COMINT is to wait. 2. Post a read on $RECEIVE and wait for a completion (STOP or ABEND) message on the specified process.
PMSG		1. Parse and validate the "on-off" flag. 2. Set the internal PMSG flag accordingly.
PPD	GETPPDENTRY, GETSYSTEMNAME	1. Parse and validate the name or <cpu,pin> of the process for which the directory is to be displayed. 2. Call GETPPDENTRY on the specified process, or, if no process was specified, on all named processes. 3. Call GETSYSTEMNAME to convert system numbers to names and then display the results.
PURGE	PURGE	Loop through the list of file names, parsing, validating, and calling PURGE on each one.
RCVDUMP	RCVDUMP	Cause the RCVDUMP server program to run. Pass the command parameters directly as a RUN parameter string to the RCVDUMP process.
RECEIVEDUMP	RCVDUMP	1. Parse and validate the dump parameters. 2. Reformat the parameters into a RUN parameter string for the RCVDUMP process. 3. Run the RCVDUMP server program.
RELOAD	RELOAD	Cause the the RELOAD server program to run. Pass the command parameters directly as a RUN parameter string to the RELOAD process.
REMOTEPASSWORD	RPASSWRD	Cause the RPASSWRD server program to run. Pass the command parameters directly as a RUN parameter string to the RPASSWRD process.
RENAME	CLOSE, OPEN, RENAME	1. Parse and validate the old and new names of the file. 2. Call OPEN to access the file. 3. Call RENAME to rename the file. 4. Call CLOSE on the file.
RPASSWRD	RPASSWRD	Causes the RPASSWRD server program to run. Pass the command parameters directly as a RUN parameter string to the RPASSWRD process.
[RUN[D]]	CLOSE, NEWPROCESS, OPEN, WRITE, WRITEREAD	1. Parse and validate the program name and RUN options. 2. Set up the parameters for and call NEWPROCESS to start the specified program. 3. Call OPEN (nowait) to access the new process. 4. Call WRITEREAD (nowait) to send the start-up message. 5. If requested, call WRITE (nowait) to send the ASSIGN messages. 6. If requested, call WRITE (nowait) to send the PARAM message. 7. Call CLOSE on the new process. 8. If this is a waited run, post a read on $RECEIVE to wait for this process to complete.

Continued on next page.

Table 1. (Concluded)
The commands recognized by COMINT, the procedures they use, and the functions of the commands.

COMINT command	Procedures it uses	Functions of the command
SET		1. Parse and validate the INSPECT parameters. 2. Save the INSPECT parameters in internal flags.
SETTIME	COMPUTETIMESTAMP, CONVERTTIMESTAMP, SETSYSTEMCLOCK	1. Parse and validate the current time. 2. Call COMPUTETIMESTAMP to convert the specified time into Julian microseconds. 3. Call CONVERTTIMESTAMP to convert the specified time to Greenwich Mean Time. 4. Call SETSYSTEMCLOCK to set the new time.
SHOW		1. Parse and validate the INSPECT flag(s) whose state is to be displayed. 2. Display the state of the INSPECT flag(s).
STATUS	CONVERTPROCESSTIME, GETSYSTEMNAME, PROCESSINFO, USERIDTOUSERNAME	1. Parse and validate the name or <cpu,pin> of the process or CPUs for which the status is to be displayed. 2. Get the search criteria. 3. If a process is specified, call PROCESSINFO on it with the search criteria. 4. If one or more CPUs are specified, call PROCESSINFO with the search criteria on every process in the CPU. 5. Call GETSYSTEMNAME to convert system numbers to names. 6. Call USERIDTOUSERNAME to convert user IDs to user names. 7. Call CONVERTPROCESSTIME to convert process time to hours, minutes, seconds, and milliseconds. 8. Display the status information.
STOP	STOP	1. Parse and validate the name or <cpu,pin> of the process to be stopped. 2. Call STOP to stop the specified process.
SUSPEND	SUSPENDPROCESS	1. Parse and validate the name or <cpu,pin> of the process to be suspended. 2. Call SUSPENDPROCESS on the process.
SWITCH	CHECKSWITCH	Call CHECKSWITCH to switch the roles of the primary and backup command-interpreter processes.
SYSTEM		1. Parse and validate the name of the new default system. 2. Save the new default system name.
SYSTIMES	CONVERTTIMESTAMP, INTERPRETTIMESTAMP, JULIANTIMESTAMP	1. Call JULIANTIMESTAMP to get the current Greenwich Mean Time and cold-load time. 2. Call CONVERTTIMESTAMP to convert the times to Local Civil Time. 3. Call INTERPRETTIMESTAMP to convert the times to year, month, day, etc. 4. Display the times.
TIME	CONVERTTIMESTAMP, INTERPRETTIMESTAMP, JULIANTIMESTAMP	1. Call JULIANTIMESTAMP to get the current Greenwich Mean Time (GMT). 2. Call CONVERTTIMESTAMP to convert GMT to Local Civil Time (LCT). 3. Call INTERPRETTIMESTAMP to convert LCT to year, month, day, etc. 4. Display the date and time.
USERS	USERS	Cause the running of the USERS server program. Pass the command parameters directly as a RUN parameter string to the USERS process.
VOLUME	PROCESSFILESECURITY, VERIFYUSER	1. Parse and validate the new default volume, subvolume, and/or file security. 2. Call VERIFYUSER to get the omitted values. 3. Reset COMINT's internal default volume and subvolume. 4. Call PROCESSFILESECURITY to change the security for file creation.
WAKEUP		1. Parse and validate the wakeup on-off flag. 2. Save the WAKEUP on-off flag.
WHO	GETSYSTEMNAME, MYPID, PROCESSINFO	1. Call PROCESSINFO on MYPID to get the CRTPID, process ID, home terminal, and system number. 2. Call GETSYSTEMNAME to convert the system number to the system name. 3. Display the WHO information.
XBUSDOWN, XBUSUP, YBUSDOWN, YBUSUP	BUSCMD	1. Reformat the parameters into a RUN parameter string for the BUSCMD process. 2. Run the BUSCMD server program.

The Tandem Global Update Protocol

The Tandem Global Update Protocol is an efficient mechanism for synchronizing and broadcasting updates to a collection of independent CPUs. It guarantees atomic updates to replicated information in a system without shared memory. Information consistency is preserved despite any number of CPU failures.

Consistency in the Tandem System

Tandem's approach to fault tolerance and ease of expansion is based upon independent CPUs in coordinated operation. Correct operation of applications requires a consistent view of key system components. For example, access to an I/O device depends on locating the process responsible for the device, which depends on a consistent description of the device-process mapping.

In many multiple-CPU systems, consistency is achieved through shared memory, but in a Tandem system, the absence of shared memory is a fundamental aspect of the system design. Shared memory is avoided for fault tolerance, expandability, and simplicity.

One might ensure consistency by centralizing information in a single CPU; this is a useful method when the information is not accessed too often and when the system has some capability to recover from the loss of the CPU. In order to achieve single-fault tolerance, the information can be replicated in a second CPU.

In the Tandem system, the goal is to maintain consistent copies of various system information in all CPUs. Such information is accessed frequently, but infrequently updated. Through this approach, both better performance and some measure of multi-fault tolerance, particularly at the lowest level (kernel) of the operating system, can be achieved.

Consistency of an update to information replicated in every CPU depends upon *atomicity* (see Gray, 1978). As applied to the Tandem system, an atomic update operation has the following characteristics:

1. The update is completed within some maximum time boundary.
2. Either every CPU is successfully updated or no CPU is updated.
3. The updates occur serially; if the same data is updated several times, the results are the same in all CPUs.

Replicated Information

In many cases, the Tandem system does centralize system information. For example, a data-base file or record is locked by recording the lock information in a single location: the disc process for that file or record.

The system information that is of interest in this article is that which is replicated in every CPU. Examples of this information include:

1. *Device-process mapping*. To access a terminal, a file on a disc, or a communications line, for example, a program must communicate with the process responsible for that device. In every CPU, the operating system maintains a system table, the Destination Control Table (DCT), that contains the name of each I/O device and the process(es) that control the device.

   A device is often referred to by its relative location in the DCT, so a given device name must occupy the same location in the DCT of every CPU.

2. *Process pairs*. The process pair is a basic building block of fault tolerance on a Tandem system. Each fault-tolerant system process or user application is a pair of processes, the *primary* and the *backup*, which reside on different CPUs. The DCT contains the name of each process pair and location of each process in a pair. Each change in the system, such as a process creation or stop, CPU failure or reload, or a "switch" of function between primary and backup processes, causes an update to the DCT.

3. *Notification of process failure*. Although a process pair is single-fault-tolerant, multiple faults can cause the process pair to fail. For multifault tolerance, the process' ancestor must be notified when the process is no longer executing. When the ancestor-descendent relationship is replicated in every CPU, ancestor notification is guaranteed even in the presence of multiple CPU failures.

4. *System messages*. Various system events (e.g., CPU ups and downs and time-of-day changes) are logged in a system-status buffer in every CPU. It is important that entries in this buffer be in the same order in all CPUs, so updates to the status buffer must be atomic.

5. *Time of day*. The current time of day should agree closely in all CPUs. Any operation to change the time of day must be atomic.

6. *Process-reload coordination*. Multiple CPUs can be reloaded in parallel (by one or more RELOAD programs), but a critical section of each reload must be performed serially and atomically. This is performed by notifying each CPU when this operation is about to occur; notification must be atomic.

Global Updates

Replicated system information must be changed only by broadcasting an update to every CPU. This operation is referred to as a *global update*.

A typical situation requiring a global update is the creation of a named process. To make the process name known throughout the system, the operating system must allocate an entry in the Destination Control Table (DCT) of every CPU and place the name and process identifier in the entry. Since a process may be referred to by its relative location in the DCT, the process name must occupy the same relative location in the DCT in every CPU.

It is possible for two or more CPUs to attempt to create processes with the same name; all but one of these attempts must fail with a *Name Already Exists* error. Thus, the global update must be atomic, so that the update attempts occur in the same order on all CPUs.

In the past, global-update operations on the DCT were required to be single-fault-tolerant. Experience has shown that some critical operations, especially those that affect overall system consistency, should be multifault-tolerant if possible. Thus, it is important that the DCT (as well as other information replicated in every CPU) remain consistent despite any number of CPU or bus failures.

Although the interprocessor bus is fast and does not constitute a bottleneck, the global-update operation should use a minimal number of messages. DCT updates are not very frequent, but that might not be the case for some future use of the global-update mechanism.

This article describes the method used in the GUARDIAN 90 operating system to update replicated system information by broadcasting an update to all CPUs. The following sections introduce those facilities of the Message System that are important to the discussion of global updates. For a more detailed description of the Message System, see Chandra, 1985.

GUARDIAN 90 Message System

The GUARDIAN 90 Message System manages traffic across the Interprocessor Buses, a pair of 100M-bit/second transfer media that connects all CPUs in a single Tandem system. Using a pair of these buses, any CPU can transfer a message to any other CPU at speeds approaching the speed of a memory-to-memory transfer within a CPU. Various levels of protocol to control message flow cause the effective transfer rate to be somewhat less than the raw bus speed, so the conservation of messages is an important factor in system performance.

Three types of message are supported by the Message System: the interprocess message, the PIO message, and the unsequenced packet. Each has different levels of protocol, and therefore, each has a different cost.

The interprocess message is the most general form of communication, supporting a full requester-server protocol in which a requester process sends a variable-length request to a server process and receives a variable-length reply. A process may be sending many interprocess messages at one time.

In order to provide fault tolerance and to manage resources effectively, a total of nine control packets, data blocks, and acknowledgments may be transmitted for each full message; in addition, as many as five process dispatches are required.

The PIO message allows a process to communicate with the GUARDIAN 90 kernel of some CPU. This message has a fixed size, and the only reply is a simple ACK/NACK acknowledgment. A process may send only one PIO message at a time and must wait for its completion, but such messages usually require a very small time to be processed and acknowledged. Within the receiving CPU, mutual exclusion of PIO messages is guaranteed; the CPU must complete processing of one PIO message before it can receive another one.

The unsequenced packet is a simple message consisting of one physical bus packet. It is used to acknowledge the receipt of other messages and for "I'm Alive" messages (see below). The unsequenced packet requires no explicit acknowledgment.

The Message System guarantees delivery of both interprocess messages and PIO messages. An unacknowledged message is retried until it is acknowledged or until the receiving CPU is declared down.

Message Broadcasting

The GUARDIAN 90 operating system contains a new Message System facility, the *N+1 Broadcast*, that allows a process to request that up to N+1 PIO messages (where N is the number of CPUs in the system) be sent by the Message System kernel.

When a process wishes to send an N+1 Broadcast, it specifies a starting CPU. The PIO message is sent to each CPU in a predetermined order (see below), including the CPU that is sending the message. Finally, the message is again sent to the starting CPU.

While the messages are being sent, the process must wait and has no control over the operation. In particular, it may not cancel the operation before it is completed. The only action that prevents the messages from being sent is the failure of either the sending or receiving CPU.

When the message is complete, a set of N+1 acknowledgment codes (ACKs or NACKs) are returned to the originating process. As described below, the N+1 Broadcast facility also performs a few special operations that are required for sending global updates.

Although "broadcasting" often implies sending messages to all CPUs at once, the N+1 Broadcast is actually a serial operation that appears to the sending process as an atomic operation. Unless the CPU to which the message is being sent is declared down, each PIO message must be accepted and acknowledged before it is sent to the next CPU. This serialization is required to provide fault tolerance of the Global Update Protocol.

The N+1 Broadcast is more efficient than normal messages. With the latter, two processes would be dispatched for each message. With the N+1 Broadcast, each receiving CPU accepts and acknowledges each message in an interrupt routine. The sending CPU receives each acknowledgment and immediately sends the next message from an interrupt routine. Entering an interrupt routine suspends execution of other processing, but does not cause expensive process switching.

Processor-status Consistency

A fundamental basis for guaranteeing consistency in the Tandem system is that all operating (or "up") CPUs must agree as to which CPUs are up. This is referred to as *processor-status consistency*. Each CPU maintains a list of other CPUs that are considered to be up. The system's processor status is consistent if every up CPU has an identical list of "up CPUs." Processor-status consistency is achieved through the Message System's "I'm Alive" protocol. Once a second, each up CPU in a system sends an unsequenced "I'm Alive" message to every other up CPU. Every two seconds, each CPU brings down every CPU from which it has not received an "I'm Alive" message in the last two-second interval.

Since a CPU never sends an "I'm Alive" message to a CPU that it considers down, this simple protocol ensures symmetry. Except during a short period following a CPU failure, any two CPUs have identical status information about each other; either each considers the other to be up or each thinks the other is down.

Once a CPU has been declared down by the system, it must be reloaded in order to rejoin the system as an up CPU.

Although it is expected that processor status may be inconsistent for a few seconds, the system eventually achieves a consistent state. The short period of inconsistency is an important consideration when global updates are performed atomically.

Overview of the Global Update Protocol

The Global Update Protocol (GLUP) is Tandem's multifault-tolerant mechanism by which an atomic update can be broadcast to every CPU in an efficient manner. GLUP components include state information maintained in each CPU, a standard global-update message format, and a set of rules that specify how global updates should be processed and what steps should be taken when failures occur.

A primary element of the GLUP state is the identity of a unique CPU, known as the *locker*. As is described below, all CPUs must agree as to which CPU is the locker.

The fundamental operation of the GLUP is to use the N+1 Broadcast mechanism to send the same global-update message to every CPU. The first message is always sent to the locker, which ensures that only one global update is in progress at any time. The locker either allows the update to proceed or notifies the sender that an update "collision" has occurred. A collision terminates the broadcast; the sender must retry the update.

If the update is allowed to proceed, the message is sent to every other CPU and, finally, it is sent once more to the locker. The locker recognizes the second copy of the update as notification that the update is complete.

The following sections explore various aspects of the GLUP, including how consistency is maintained when (1) multiple senders attempt an update at the same time, (2) one or more CPUs fail during an update, and (3) a CPU is reloaded and must be synchronized with the other CPUs.

Technical Description

The GLUP state consists of the following elements:

1. A designated CPU, known as the *GLUP-Locker*, to coordinate global updates. The first-loaded CPU is the initial locker. It remains the locker until it fails; the remaining CPUs then select a new locker, as described below.
2. A well-defined ordering of all CPUs, known as the *GLUP-Order*. This order must begin with the GLUP-Locker and include each other CPU exactly once. Any ordering would do, but for simplicity, the chosen sequence is the natural ordering:

   L, L+1, L+2, ..., N-1, 0, ..., L-1

   where L is the number of the locker CPU and N is the number of CPUs. GLUP-Order is illustrated in Figure 1.
3. In each CPU, a one-word semaphore, *GLUP-Lock*, which contains either -1, to indicate that the update lock is not held, or a process ID (PID), to indicate that the lock is currently held by that process.

   The value of GLUP-Lock in the locker CPU determines whether the update lock is held. The GLUP-Locks in the other CPUs are, however, essential for recovery of failures.
4. In each CPU, a one-word GLUP sequence number, *GLUP-Seq*. Whenever the GLUP-Lock is not held (in the GLUP-Locker CPU), all CPUs have identical GLUP-Seqs. During an update, each CPU that receives and processes the update message increments its GLUP-Seq by 1.
5. In each CPU, a small array, *GLUP-Update*, that contains the last update processed by the CPU. This is simply a copy of the PIO message that was sent by the updating CPU.

A standard update message contains four elements:

1. The process ID of the updating process.
2. A "lock bit," to distinguish a "locking update" from a nonlocking update.
3. A GLUP-Seq copied from the sending CPU.

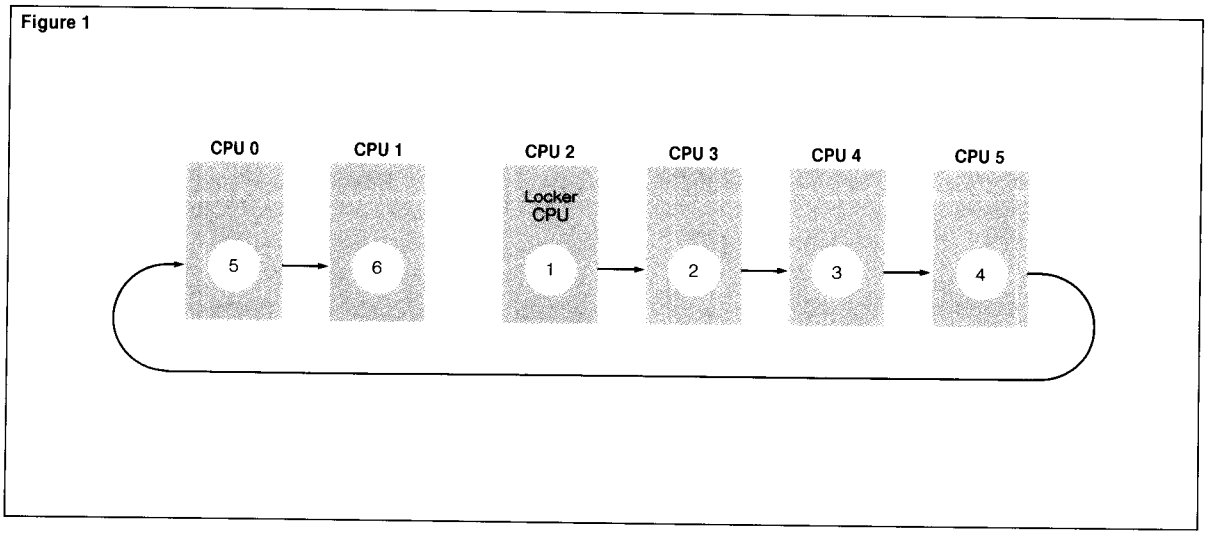


Figure 1.
GLUP-order, the order of CPUs for a global update. This order begins with the GLUP-Locker and includes each CPU. The sequence is L, L+1, L+2, ..., N-1, 0, ..., L-1, where L is the number of the locker CPU and N is the number of CPUs.

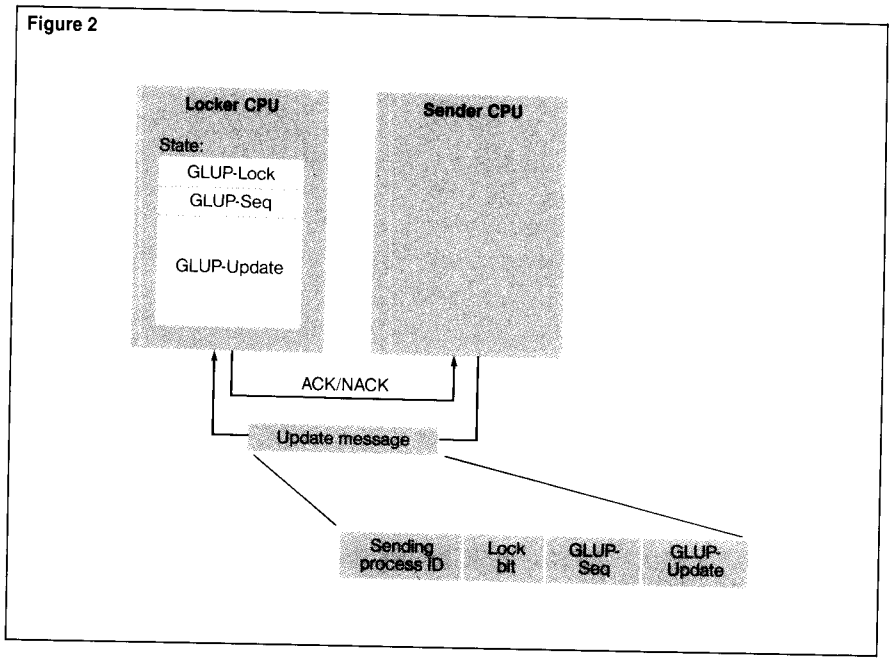
4. A fixed-length array containing a description of the update; the format of this array depends on the type of update.

The update message is a special type of the PIO message described above. The GLUP state and the sending of an update message are illustrated in Figure 2.

Standard Update Sequence

The normal steps (in the absence of failures) for performing a global update are:

1. A process (in the "sending" CPU) constructs a message containing an update and a GLUP-Seq copied from the sending CPU. The message is given to the sending CPU's Message System as an N+1 Broadcast. The process is blocked.
2. The sending CPU sends the update message to the locker CPU. This first message of an N+1 Broadcast is marked as a "locking update" message.
3. The locker CPU examines the GLUP-Lock. If the semaphore is already held, the message is simply NACKed; the N+1 Broadcast is terminated in the sending CPU and the sending process is notified. Typically, the process delays for one hundredth of a second and reattempts the update.
4. The locker compares the GLUP-Seq in the message with its own GLUP-Seq. If they do not match, the message is NACKed, which also terminates the broadcast. Finally, the locker ensures that the update is a locking update; a nonlocking update is also NACKed.
5. The locker CPU sets the sending process' PID in the GLUP-Lock, increments its GLUP-Seq by 1, and saves the update message in its GLUP-Update storage. It then processes the update and ACKs the message.



6. The sender CPU sends a nonlocking update to every other CPU in GLUP-Order. When the update is received, each CPU sets the sending process' PID in the GLUP-Lock, increments its GLUP-Seq by 1, and saves the update message in its GLUP-Update storage. It then processes the update and ACKs the message.
7. Finally, the sender sends a nonlocking update message, recognized as the update-termination message, to the locker. The locker stores -1 (meaning *unlocked*) in GLUP-Lock and ACKs the message.
8. The originating process is unblocked and allowed to execute. The update is complete.

Figure 2.
The GLUP state and the sending of an update message. The message consists of the process ID of the sending process, the "lock bit" which distinguishes a locking update from a nonlocking update, a GLUP-Seq copied from the sending CPU, and a fixed-length array containing a description of the update.

The message flow for updating a four-CPU system is illustrated in Figure 3. Message 1 is the locking update that is sent to the locker CPU. If it is NACKed, the update is terminated and must be retried; otherwise, messages 2, 3, and 4 are sent to update the other CPUs. Note that the sender CPU updates itself in GLUP-Order. Finally, Message 5, a nonlocking update, is sent to the locker to signal the end of the update.

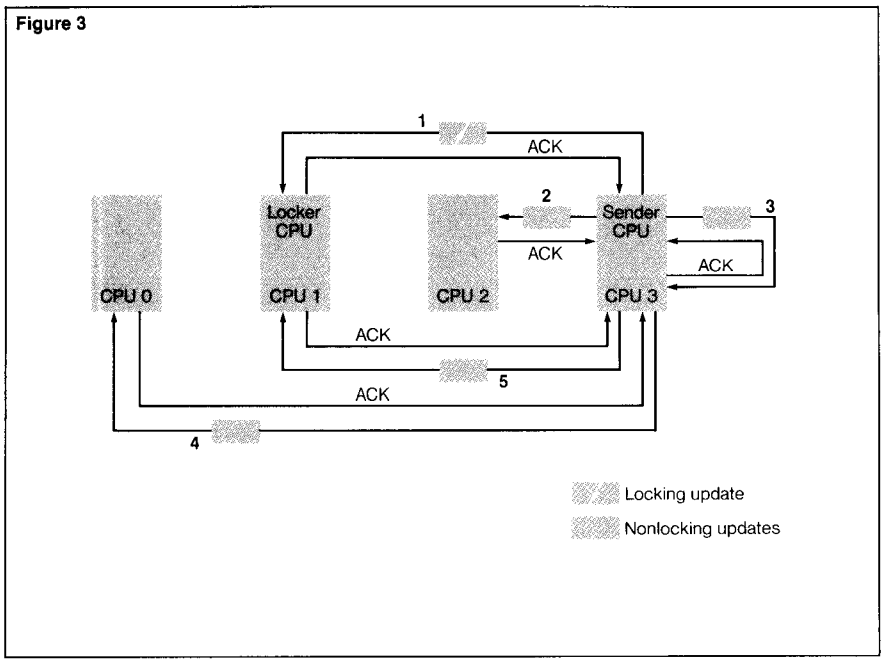
Global-update Sequence Number

The global-update sequence number has a number of important uses; primarily, it eliminates the need for a semaphore when a shared-data update depends upon the current value of the shared data.

Any process that wishes to make such an update performs the following sequence:

1. Copy the local CPU's GLUP-Seq.
2. Construct an update description that may be a function of the current state of shared data.
3. Instruct the Message System to broadcast the update, including the copied GLUP-Seq.
4. If the update fails due to a colliding update, go to step 1.

Figure 3.
The message flow for updating a 4-CPU system. Message 1 is the locking update sent to the locker CPU. If it is NACKed, the update is terminated and must be retried; otherwise, Messages 2, 3, and 4 are sent to update the other CPUs. Note that the sender CPU updates itself in GLUP-Order. Finally, Message 5, a nonlocking update, is sent to the locker to signal the end of the update.



For each possible GLUP-Seq, one and only one global update succeeds. Furthermore, successful updates are processed in the numerical order of their GLUP-Seq. Thus, the shared data could not have changed between the time the GLUP-Seq was copied and the time the global update was processed in each CPU.

This method eliminates the need for the process to obtain the GLUP-Lock semaphore before it constructs an update based on the current value of shared data. It sharply reduces the time that any global semaphore is held and eliminates the opportunity for a process to obtain a semaphore and "forget" to release it.

This method also eliminates the possibility of deadlock on the GLUP-Lock semaphore when a failure requires a change in the system configuration before processes can be executed. Such failures can require a global update; if some process holds the GLUP-Lock, a deadlock occurs. In the past, deadlocks were avoided only through very careful analysis of asynchronous events and exhaustive testing to discover the errors in the analysis.

The GLUP-Seq is also used to detect and discard duplicate updates. Some updates, such as "advance the system clock by *n* microseconds," must be applied exactly once in every CPU. Various failures cause the latest update to be re-sent to ensure that every CPU receives it, but those CPUs that have already received the update detect the sequence mismatch and ignore the duplicate.

CPU Failures and Global-update Consistency

CPU failure is the most common cause of difficulty in maintaining the consistency of global updates. If a sending CPU were to fail after updating a subset of the CPUs, the remaining CPUs would be inconsistent with the updated CPUs. If a locker CPU were to fail while an update were in progress, the system would lose its ability to prevent concurrent updates to shared data.

Sender CPU Failure

Recovery from a sender CPU failure is the responsibility of the locker CPU. If a CPU failure is detected by the locker, it checks the GLUP-Lock. If it is not held (i.e., is -1), then there is no update in progress and no recovery is necessary.

If the GLUP-Lock contains the PID of a process that was running in the failing CPU:

1. The locker constructs an update message from the saved message in GLUP-Update and the current GLUP-Seq minus 1.
2. The locker uses the N+1 Broadcast to resend the message to all CPUs. The first locking update message is not re-sent because this message was already processed by the locker.
3. Those CPUs that processed the update before the sending CPU failed reject the global update because the GLUP-Seqs do not match. The remaining CPUs accept and process the update.
4. Finally, the locker sends the update to itself and the message is recognized as an update termination, resetting GLUP-Lock to -1 and allowing other global updates to be processed.

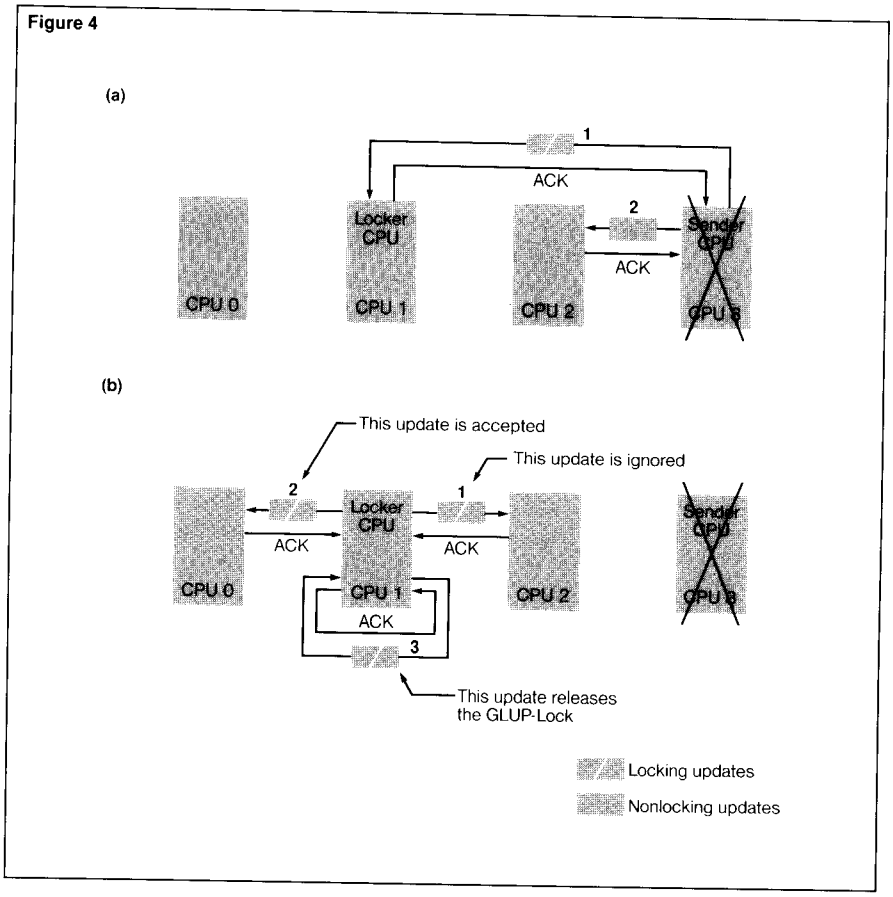
Thus, once an update message is accepted by the locker CPU, it is "committed" and is sent to every up CPU at least once unless, possibly, the locker CPU fails.

Figure 4a illustrates the failure of a sender CPU (CPU 3) after it has updated both the locker and another CPU (CPUs 1 and 2). In Figure 4b, the locker resends the update to all up CPUs. The update is discarded by CPU 2 because it has already received it. CPU 0 accepts the update, and the locker CPU releases the GLUP-Lock when it receives the update.

Failure of the Locker CPU

When the locker CPU fails, the failure is detected in every other CPU at different times, but eventually all remaining CPUs detect the failure. Each CPU chooses the new locker to be the CPU following the failed locker in GLUP-Order.

Figure 4



The new locker realizes that it must take over from the old locker. When it does so:

1. The new locker's GLUP-Lock is set to the process ID of the last update that it processed, preventing any new updates from being accepted. Since the new locker CPU cannot examine the old locker's GLUP-Lock, it cannot tell if the last update was completed or not. The simplest way to ensure that the last update was completed is to send it once more.
2. The new locker reconstructs the last update it received using the saved GLUP-Update and the current GLUP-Seq minus 1.
3. The new locker performs an N+1 Broadcast to the up CPUs. Any CPUs that had not received the last update accept and process it; the others reject it.

Figure 4.
(a) CPU 3 dies after updating both the locker and another CPU (CPUs 1 and 2). (b) The locker CPU resends the last update to all up CPUs. CPU 2 discards the update because it has already received it. CPU 0 accepts the update and the locker CPU releases the GLUP-Lock when it receives the update.

4. When the last message of the broadcast is re-sent to the new locker, it resets the GLUP-Lock semaphore to -1. New updates can now be accepted by the new locker.

Since the new locker is always the next CPU in GLUP-Order following the old locker, no CPU other than the old locker has received any update that has not been received by the new locker.

Figure 5

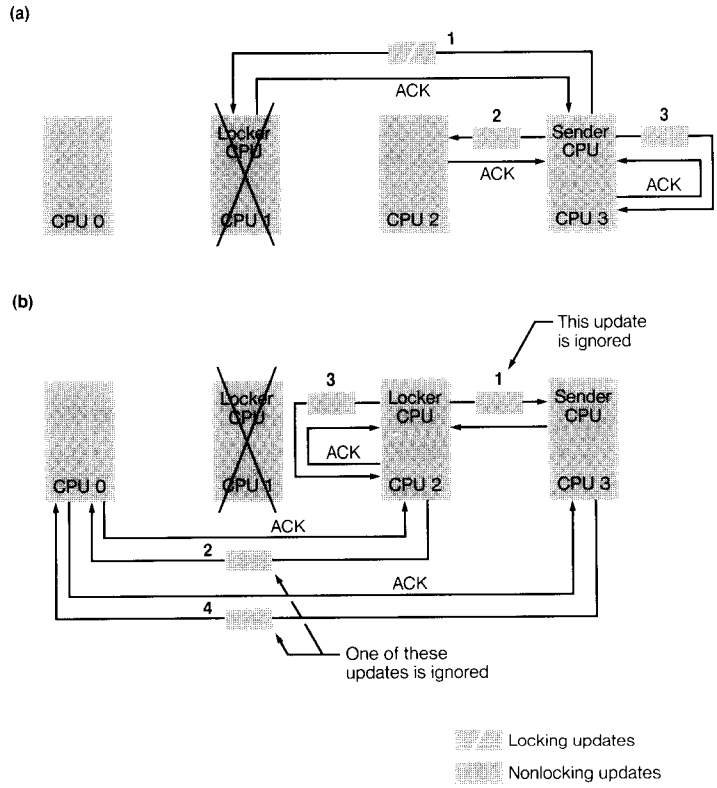


Figure 5.

(a) The locker CPU dies after CPUs 1, 2, and 3 have received an update from CPU 3. (b) The new locker (CPU 2) resends the last update to all up CPUs. CPU 3 ignores the update because it is a duplicate of the previous update. CPU 0 accepts either Message 2 from CPU 2 or Message 4 from CPU 3, depending on which arrives first; it then ignores the other message. When the new locker sends Message 3 to itself, it releases the GLUP-Lock.

When the new locker broadcasts the last update, the originating CPU may still be sending the update. Updates from the new locker may overtake updates from the original sender; some of the sender's updates may get NACKed due to mismatching GLUP-Seqs and duplicate updates. Thus, the sender ignores NACKs to broadcast messages other than the locking-update message. Once the sender is through broadcasting, the update has been completed.

On the other hand, other CPUs may send locking updates to the new locker before the new locker discovers the old locker is down. These updates are terminated immediately, because any CPU NACKs a locking update if it doesn't consider itself to be the locker. The updating processes continue to retry their updates until the new locker discovers its true identity, resends the last update, and permits new updates to be processed.

Figure 5a illustrates the failure of the locker CPU after CPUs 1, 2, and 3 have received an update from CPU 3. In Figure 5b, the new locker (CPU 2) resends the last update to all up CPUs. CPU 3 ignores the update because it is a duplicate of the previous update. CPU 0 accepts either Message 2 from CPU 2 or Message 4 from CPU 3, depending on which arrives first; it then ignores the other message. When the new locker sends Message 3 to itself, it releases the GLUP-Lock.

Failure of Another CPU

If a CPU other than the sender or the locker fails, the update proceeds normally. The Message System either succeeds or fails to transmit the update to the failing CPU, but in either case, it continues to send the update to all other CPUs.

Multiple CPU Failures

The most interesting cases of multiple CPU failure are (1) concurrent failure of the sender and locker and (2) concurrent failure of the current locker and the CPU that should become the new locker. The remaining cases are simply dealt with and are left as an exercise for the reader.

In the event of a concurrent failure of the sender and locker, if the sender fails before it has successfully updated the locker, the update does not succeed because no up CPU ever accepts the update. If the sender fails after it has updated the locker, the locker knows about the update and tries to resend it to the remaining CPUs. If the update is received by the CPU that will become the new locker, either from the original sender or by the locker when the sender fails, the update commits; when the old locker fails, the new locker resends the last update to all other CPUs. If both the sender and old locker fail before either of them send the update to the new locker, the update is confined to down CPUs and might as well have not happened.

The concurrent failure of an old and new locker is simply handled because updates are always sent in GLUP-Order and CPUs become the locker in GLUP-Order. The CPU next in order following the new locker becomes the locker, and if the sending CPU (or one of the locker CPUs following the failure of the sending CPU) updates this CPU before it takes over as the new locker, the update commits; otherwise, no up CPU is updated.

Failures of three or more CPUs are easily broken down into the previously described cases. The essential rules are:

1. If the sender CPU does not fail, the update is successful.
2. If the sender CPU fails, the update is successful if it is transmitted to a CPU that becomes the locker and does not fail; otherwise, the update is confined only to down CPUs and has no effect on the rest of the system.

Reloading and Synchronizing a Down CPU

In the Tandem system, a down CPU is reloaded in the following manner:

1. The CPU is reset and placed in a state to receive a memory image across the interprocessor bus.
2. A RELOAD process is executed in an up CPU, which sends a memory image of the GUARDIAN 90 kernel and some special processes to the down CPU. It also sends a message to every other CPU notifying it to expect the down CPU to come up.
3. The RELOAD program notifies the CPU to begin executing; its Message System is activated to send "I'm Alive" messages and to receive other messages, particularly global updates. Every other CPU recognizes that the down CPU is now up and begins to exchange "I'm Alive" messages with it.
4. The RELOAD program sends some additional shared data messages (e.g., the DCT) to synchronize the reloaded CPU with the system. When the CPU is fully synchronized, it starts executing processes.

When a down CPU is reloaded, operations must be carefully sequenced to ensure that the CPU has exact copies of all shared data and that no significant updates are missed during the synchronization phase.

The reloading strategy used in the GUARDIAN 90 operating system assumes that (1) the copying of shared data from the reloading CPU to the reloaded CPU is an operation that can be retried and (2) shared-data copy operations require less time than the typical time between global updates.

In the sending (reloading) CPU, the RELOAD process performs the following operations:

1. It obtains a copy of the current GLUP-Seq.
2. It sends shared-data copy messages to the reloaded CPU. These are not global-update messages and are not subject to the GLUP processing described above.
3. It sends a global update to all CPUs (including the reloaded CPU) that contains the copied GLUP-Seq and the update operation code to make the shared data valid.
4. If the global update fails due to an expired GLUP-Seq or update collision, it returns to step 1.

In the reloaded CPU, the GUARDIAN 90 kernel performs the following operations:

1. If a normal global update is received, it consults a local flag, *shared data is valid*. This flag is not set when the CPU is reloaded; if it is still not set, the kernel discards the global update but ACKs it as if it were acceptable.
2. If a shared-data copy message is received from the RELOAD process, the CPU stores the shared data in the proper table. The CPU must be prepared to receive duplicate shared-data copy messages and cancel the effect of all such duplicates except the last.
3. When the make-shared-data-valid update is received, the CPU sets the shared-data-is-valid flag and processes all subsequent global updates in a normal fashion. Note that it is the responsibility of the RELOAD program (by using a global update) to determine that the reloaded CPU has received the current replicated data.

Basically, the shared-data copy operation is protected by the GLUP-Seq mechanism. No global updates can occur between the beginning of the shared-data copy and receipt of the make-shared-data-valid update in the reloaded CPU. At that time, the replicated data in the reloading CPU and the reloaded CPU agree and are kept in agreement by the standard global-update mechanism.

If the amount of replicated data is too large, then it will be necessary to divide the shared data into sections, and each section will require its own shared-data-is-valid flag and make-shared-data-valid global-update operation.

One must be careful that the CPU being reloaded does not reference shared data before it is validly copied from the reloaded CPU. In GUARDIAN 90, this is easily accomplished by disabling process dispatching until all shared data is valid.

Finally, if some other CPU dies during a reload operation, the CPU being reloaded kills itself unless all of its shared-data-is-valid flags have been set.

Performance

Figure 6 compares the performance of the DCT update operation for both the A20 GUARDIAN and B00 GUARDIAN 90 operating systems. The A20 version obtains a global semaphore before sending the update to every CPU. In a single-CPU system, no messages are required to obtain the semaphore, so its performance is equivalent to that of GUARDIAN 90. In a multiple-CPU system, obtaining the semaphore causes the fixed overhead of A20 GUARDIAN to be much greater than that of GUARDIAN 90.

Conclusion

The GUARDIAN 90 operating system incorporates a new algorithm for updating replicated data in a distributed system with no shared memory. This algorithm is very efficient and provides consistency and atomicity despite any number of CPU or link failures. This algorithm performs considerably better than a conventional, semaphore-based algorithm that survives only a limited number of failures.

Figure 6

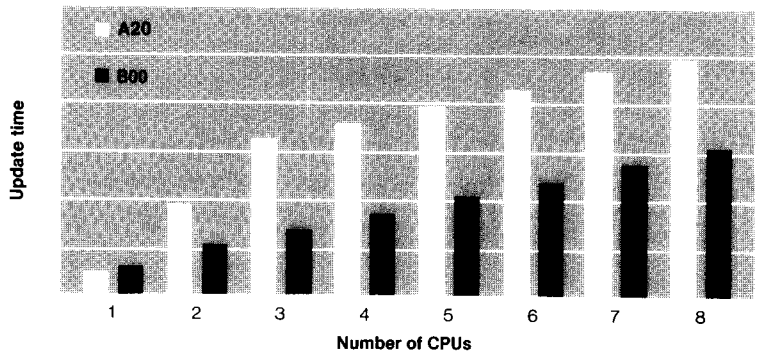


Figure 6.
A comparison of the elapsed time for a global update on the A20 GUARDIAN and B00 GUARDIAN 90 operating systems. The A20 version obtains a global semaphore before sending the update to every CPU. In a single-CPU system, no messages are required to obtain the semaphore, so its performance is equivalent to that of GUARDIAN 90. In a multiple-CPU system, obtaining the semaphore causes the fixed overhead of A20 GUARDIAN to be much greater than that of GUARDIAN 90.

References

Chandra, M. 1985. The GUARDIAN Operating System and How to Design for It. *Tandem Systems Review*. vol. 1, no. 1. Tandem Computers Incorporated.

Gray, J. N. 1978. Notes on Data Base Operating Systems. In *Operating Systems—An Advanced Course*, eds. R. Bayer, et al., pp. 393-481. Springer-Verlag.

Acknowledgments

Wendy Bartlett made many suggestions that improved this article immeasurably; her efforts are deeply appreciated.

Richard Carr is the staff technical consultant for the T16 Systems group in Software Development. Since joining Tandem in 1981, he has designed and implemented SYSGEN2, the NonStop II memory manager, DSAP, DCOM, and the GUARDIAN facilities described in this article. Before coming to Tandem, Richard was associated with Stanford University. As a staff member, he designed and implemented the time-sharing system for the university's large-scale IBM computers, as well as the interactive FORTRAN compiler for that system. He also obtained a Masters and Ph.D. in Computer Science from Stanford.

Changes in FOX

The low-level handshaking protocols used by the 6700 Fiber Optic Extension (FOX) have been completely reimplemented in the B00 software release to provide better monitoring of available paths to different clusters. As the changes make B00 FOX incompatible with the A06 and A20 versions of the GUARDIAN operating system, it is necessary for users to upgrade *all* of the systems on a FOX ring to GUARDIAN 90 (B00 release) at the same time. (Future modifications to FOX will remain compatibile with GUARDIAN 90.)

This article describes some of the features of the new FOX protocols and their effect on FOX users.

FOX Handshaking Protocols

The software that controls the operation of a FOX ring includes the IPB monitor process, EXPAND line handlers, and the Message System. The interprocessor bus (IPB) monitor processes in the various systems in a ring communicate with one another using low-level handshaking protocols. The network line handlers on A06 and A20 GUARDIAN/EXPAND systems also communicate using handshaking protocols. These protocols are used to establish a network connection between FOX systems.

Original Protocols

The original (A06 and A20) protocols do not consider any characteristics of the underlying physical hardware of a system, such as the bus controllers and the ring topology. The information available to the IPB monitor process and FOX line handlers do not permit the software to deduce the layout of the FOX ring. These protocols are adequate to establish a network connection, but they do not permit the localization of failing components when hardware failures occur, especially for rings with more than two or three systems or for rings with systems physically far apart.

New Protocols

The B00 protocols provide a better model of the ring configuration of a FOX network to aid in the diagnosis of problems and also to provide better recovery from errors. Each IPB monitor process monitors all four paths (X LEFT, X RIGHT, Y LEFT, and Y RIGHT) and maintains a view of the FOX ring consisting of an ordered list of all systems that are up on each path. Since a FOX ring configuration may change as clusters are cold loaded or taken down, each IPB monitor process builds up the view dynamically from the information exchanged by the low-level protocols.

Each IPB monitor also monitors the bus controllers and the FOX links for errors. The bus-controller microcode makes information available to the IPB monitor about the status of the links and error counts. The IPB monitor uses this information to determine whether the link is usable or not. For example, if it detects a link that is cross-wired

(from the X bus to the Y bus) or if certain error counts (such as that for dropped packets) exceed a certain threshold, it declares the link unusable. When this happens, the information is propagated over the entire FOX ring so that no more message traffic is sent on the bad link, minimizing error resends.

This mechanism also enables a link to be removed from service for maintenance or to insert a new system onto the ring without causing an excessive number of errors and resends at other systems on the ring. The IPB monitor process also automatically declares the link to be usable if the error condition disappears; for example, if the cross-wired link is rewired properly. The error information is also made available to users through Communications Management Interface (CMI) status-display commands.

An additional benefit of the logical-ring configuration information is that the Message System attempts to use the path with the smallest number of intervening systems for message communications.

TMDS FOX Diagnostic Subsystem

B00 FOX also supports the Tandem Maintenance and Diagnostic System (TMDS). The FOX diagnostic subsystem allows a Tandem customer engineer (CE) to take a bus controller (LBU) out of service and diagnose errors, using downloadable microdiagnostics. Normal message traffic continues through the other controller. This is useful in localizing hardware faults.

Need for Simultaneous Upgrade to GUARDIAN 90

The most important effect of the new protocols on FOX users is that, in order to use FOX with GUARDIAN 90, all systems on the ring must be upgraded to GUARDIAN 90 simultaneously. This is necessary because the new protocols are incompatible with the old ones. As explained above, the information exchanged in B00 FOX is much more elaborate than that exchanged in A06 and A20 FOX. Also, the new protocols now require the cooperation of other IPB monitors.

Direct-connect EXPAND Line Handlers

Users who do not wish to upgrade all systems to GUARDIAN 90 simultaneously can use other communications lines (e.g., direct-connect EXPAND line handlers) to maintain communications until the complete upgrade is made.

Leaving B00 EXPAND Line Handlers Down

A GUARDIAN 90 system cannot communicate over a FOX ring with an A06 or A20 GUARDIAN system. Similarly, a GUARDIAN 90 system cannot communicate with another GUARDIAN 90 system if the two are separated from each other on the FOX ring by an A06 or A20 GUARDIAN cluster.[1]

If the EXPAND line handlers in the GUARDIAN 90 systems on a ring are not brought up (with the Peripheral Utility Program, PUP), however, GUARDIAN systems can use the FOX ring to communicate with each other. This works even if these clusters are separated from each other by GUARDIAN 90 clusters. Similarly, if the GUARDIAN 90 systems on a ring are adjacent to one another (i.e., there are no intervening GUARDIAN systems) and the EXPAND line handlers at the GUARDIAN systems are not brought up with PUP, the GUARDIAN 90 systems can communicate over the FOX ring.

Users can thus use the ring in a limited fashion before they have upgraded all systems to the GUARDIAN 90 operating system. Caution is recommended to users who do this, however, since unpredictable results can occur if both GUARDIAN/EXPAND and GUARDIAN 90/EXPAND line handlers are brought up with PUP simultaneously. Users should also note that some CMI display information may be spurious if a ring contains both GUARDIAN 90 and GUARDIAN operating systems.

[1] In the rest of this section, *GUARDIAN* is synonomous with *A06/A20 GUARDIAN*.

Other External Changes

Other external changes in the B00 release of FOX include changes to SYSGEN modifiers for the IPB monitor and the FOX line handlers, and changes to CMI commands and displays. The SYSGEN modifiers have been changed to delete certain modifiers for the line handler that are specific to FOX (e.g., NEXTSYSCLUSTER).

The CMI commands and displays are modified to include the new information available to the IPB monitor process. For example, the CMI command STATUS SUBNET now displays an ordered view of the ring from the local system on each bus and direction, as well as an indication of whether the ring is continuous or there is a break. If the ring is broken, the reason for the break is also displayed (e.g., DOWNED BY OPERATOR). This information is useful to operations and service personnel.

Conclusion

The new FOX protocols make the FOX ring easier to diagnose and maintain. Error handling and recovery are also improved. These protocols can now be enhanced in future releases while their compatibility with the GUARDIAN 90 operating system is maintained. The implementation of these protocols results in an incompatibility between A06 and A20 GUARDIAN and GUARDIAN 90, but this one-time incompatibility is justified by the extended functionality and future expandability of FOX.

Acknowledgments
The author would like to acknowledge Rich Larson's help in designing the B00 FOX protocols.

Nitin Donde joined Tandem in July 1982. He works in the Operating Systems group in Software Development. Nitin has an M.S. in Computer Science from the University of Wisconsin in Madison.

Improved Performance for BACKUP2 and RESTORE2

New high-performance versions of the BACKUP and RESTORE programs are available as part of the GUARDIAN 90 operating system. BACKUP2 is used to copy disc files onto magnetic tape; RESTORE2 is used to return those files to disc.

BACKUP2 and RESTORE2 handle both DP1 and DP2 files, but their significant performance improvements are most evident with the latter. Still available, BACKUP and RESTORE handle only DP1 files.

The performance improvements achieved with BACKUP2 and RESTORE2 are a result of:

1. Improved software (DP2, BACKUP2, and RESTORE2).
2. Improved hardware (3107 disc controller).
3. Improved microcode (3206 tape controller).

Results of performance tests show that BACKUP2 performs up to 2.5 times faster on DP2 files than BACKUP performs on DP1 files. RESTORE2 (on DP2 files) is up to 4.5 times faster than RESTORE (on DP1 files). Also, using an improved algorithm, RESTORE2 performs up to 1.6 times faster on DP1 files than RESTORE does. BACKUP2 and BACKUP have identical performance on DP1 files. These results are illustrated in Figure 1a and b.

This article discusses the performance analyses made on BACKUP, RESTORE, BACKUP2, and RESTORE2 and quantifies the improvements resulting from each of the hardware and software features mentioned above. It ends with a mention of areas for possible future improvement.

Figure 1

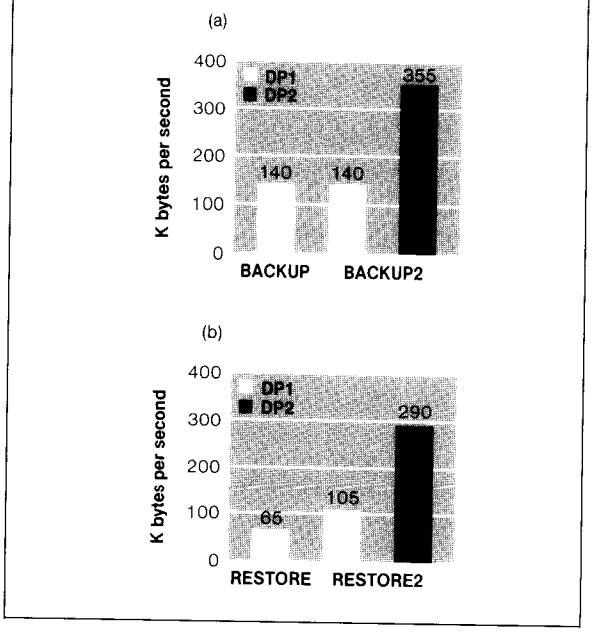


Figure 1.
(a) Performance comparison of backup operations on large files. BACKUP was run on DP1 files, and BACKUP2 was run first on DP1 and then on DP2 files. (b) The same comparison for restore operations.

BACKUP2 Performance

Backing up a disc file involves:

1. Opening the file.
2. Writing the file label to tape.
3. Copying the contents of the file to tape.
4. Closing the file.

The performance analysis of BACKUP was restricted to the backing up of large files, as that is the area in which the greatest improvement could be made. Also, for the purposes of this performance study, the backup processing was considered to be the copying of the contents of the disc file to tape. The overhead associated with opening and closing files was not measured.

Dump-loop Algorithm

The *dump loop* is the code within BACKUP that is responsible for copying the file contents. The algorithm it uses (minus the details of loop termination and buffer management) can be described as follows:

```
LOOP
  READ( Discfile, Buffer, Wait );
  AWAITIO( Tape );
  ! Compute tape record checksum for "Buffer".
  WRITE( Tape, Buffer, Nowait );
ENDLOOP;
```

The performance analysis was based on a comparison of the time required to read a block of data from disc and the time needed to write the same data block to tape. Since DP1 disc reads (and writes) are limited to 4K bytes, eight disc reads were required to fill a 30K-byte tape record.

The BACKUP dump-loop algorithm can be viewed as a combination of two semi-independent processing loops, the DISC^LOOP and the TAPE^LOOP. The DISC^LOOP consists of all the processing required to get a data block (which was to be written to tape) from the disc; the TAPE^LOOP comprises the processing between two tape writes. These loops operate at different speeds and are synchronized by the AWAITIO procedure.

For the DISC^LOOP, it was assumed that the "nowait" tape write would be completed when AWAITIO was called. The DISC^LOOP time would be its best-case time.

The DISC^LOOP consists of the "wait" disc read and other components that are executed serially with it in the dump loop. These other components are the processing time required by AWAITIO, the checksum calculation of the tape record, and the nowait tape write call (but not the time required by the tape process to write the record).

Similarly, for the TAPE^LOOP, it was assumed that the disc read would be finished before the tape write finished. The TAPE^LOOP consists of the nowait tape write call, the time required by the tape process to complete the tape write, the processing time required by AWAITIO, and the checksum calculation of the tape record.

Software measurements for various components of the disc and tape loops, along with calculated timings for the channel, disc, and tape hardware, were used to obtain the times required to execute the disc and the tape loops for 30K-byte blocks. With a TRIDENT tape drive, 3106 disc controller, 4104 disc drives, NonStop TXP processors, and B00 software, the time required to execute the DISC^LOOP was 211.7 ms and the time to execute the TAPE^LOOP was 125.4 ms.

For BACKUP, since the DISC^LOOP takes more time to process a block than the TAPE^LOOP, the rate at which the data can be backed up is determined by the rate at which the DISC^LOOP can get data from the disc. It took 211.7 ms to process one 30K-byte block; therefore, the predicted

backup data rate was 145,000 bytes/second. This compared favorably with the measured data rate of 140,000 bytes/second, validating the model of the backup operation used in the test.

DP2 BULKIO

From the foregoing analysis, it was clear that the DISC^LOOP needed to be improved to speed up backup processing. The dump-loop algorithm was examined, and it was noted that, although the tape record was written with one File System write, the disc record was read with eight reads. One obvious way to speed up the DISC^LOOP would be to change the algorithm to use only one disc read for every iteration of the dump loop. BACKUP2 was designed to do this by making use of the DP2 BULKIO feature, which allows DP2 files to be read and written in 30K-byte record sizes.

In addition to allowing large record sizes to be read and written, DP2 BULKIO provides fast sequential data access. The DP2 disc process bypasses cache when it is performing DP2 BULKIO. It is assumed that the number of cache hits would be too low for DP2 BULKIO to justify the performance penalty of searching cache.

Also, the File System does not buffer DP2 BULKIO records. For normal reads and writes, the File System sends and receives records from temporary buffers in its private data space. For DP2 BULKIO, the File System sends and receives records directly from the program's data space, eliminating the time required to copy the data into its own buffers. (Programs that use DP2 BULKIO must be privileged and must obey certain system rules.)

Note that both the DISC^LOOP and the TAPE^LOOP include time to compute the tape-record checksum. With DP2, this checksum calculation can be eliminated, however, because DP2 computes the checksum of every sector it reads. For BULKREAD, it combines the checksums of all the sectors read and passes the resulting checksum back. Thus, the checksum returned by BULKREAD to BACKUP2 speeds up both the DISC^LOOP and the TAPE^LOOP.

With the DP2 BULKIO feature, the DISC^LOOP time was 170.5 ms and the TAPE^LOOP time remained 125.4 ms. This made the backup speed 180,000 bytes/second, providing approximately a 30% improvement.

Note that even with DP2 BULKIO, the DISC^LOOP time was longer than the TAPE^LOOP time; hence, the DISC^LOOP time needed to be reduced even further for better performance.

Longer Transfers

An analysis of the different components of the DISC^LOOP showed that a long time was spent in getting the data from the disc to the processor's memory. Also, although the use of DP2 BULKIO reduced the number of logical reads of the disc file from eight to one for every tape block write, it still took eight disc accesses to get the data for one tape record from the disc. The amount of time required for each disc access had the following components:

1. *Seek time*, the amount of time required to get the disc head to the correct cylinder. It is usually zero for backup, since the data is read sequentially from disc.
2. *Latency time*, the amount of time spent waiting for the correct sector to be positioned under the disc head.
3. *Data-transfer time*, the amount of time required to transfer the data from disc to the disc-controller buffer.
4. *Channel time*, the amount of time required to burst the data from the controller buffer to the processor over the channel.

An analysis of the components revealed that the amount of time required to access data from disc could be reduced considerably by reducing the number of accesses required to get 30K bytes from disc. If the number were reduced from eight to one, the latency time for seven accesses would be eliminated.

It was possible to do this with the longer transfer capability of the 3107 disc controller. DP2 uses this capability when performing BULKIOs. With a 3107 controller, the DISC^LOOP took 112.4 ms, making it faster than the TAPE^LOOP. This meant the backup speed, now determined by the TAPE^LOOP time, was 243,000 bytes/second.

Larger Channel Bursts

The bottleneck had shifted from the DISC^LOOP to the TAPE^LOOP, requiring a reduction in the time taken by the latter. An examination of the components of the TAPE^LOOP revealed that 78% of the TAPE^LOOP time was consumed by hardware components, in transferring the data from the CPU to the tape-controller buffer and writing the tape record to tape.

The time required to transfer the data from the processor to the tape controller could be reduced considerably by transferring the data in larger bursts, since the tape controller uses 16-word bursts and has a large hold-off time after every burst. A channel burst size of 128 words resulted in a TAPE^LOOP time of 78.5 ms while the disc time remained at 112.4 ms, making the backup speed 273,000 bytes/second.

Redesign of the Dump Loop

The bottleneck had shifted back to the DISC^LOOP. Measurements showed the disc drive to be idle about half the time, when any of the following occurred:

1. The record was sent from the disc process to BACKUP.
2. The previous tape write was awaited.
3. The record was written to tape.
4. The next disc read request was sent to the disc process.
5. The disc process prepared the read request.

DISC^LOOP throughput would improve if the next read request were queued at the disc process when the current read finished. This would allow steps 2 and 3 to proceed in parallel with steps 4 and 5. In BACKUP2, the dump loop was redesigned to accomplish this.

BACKUP2 opens the file twice and issues nowait disc reads on both OPENs. (It is necessary to open the file twice, as only one nowait operation can be outstanding against a disc-file OPEN at one time.) A nowait read is always outstanding on one of the OPENs for the next record while the current record is awaited and used. The BACKUP2 DP2 dump-loop algorithm is as follows:

```
BULKREAD( Discfile1, Buffer1, Nowait );
LOOP
  BULKREAD( Discfile2, Buffer2, Nowait );
  AWAITIO( Discfile1 )
  AWAITIO( Tape );
  WRITE( Tape, Buffer1, Nowait );
  BULKREAD( Discfile1, Buffer1, Nowait );
  AWAITIO( Discfile2 )
  AWAITIO( Tape );
  WRITE( Tape, Buffer2, Nowait );
ENDLOOP;
```

With this new dump-loop algorithm, the DISC^LOOP time for BACKUP2 is 83.3 ms and the TAPE^LOOP time remains 78.5 ms, making the expected backup speed 369,000 bytes/second. The measured throughput is 351,000 bytes/second, which is near the predicted throughput.

RESTORE2 Performance

Restoring a disc file involves:

1. Reading its tape-file label.
2. Purging the old disc file (if necessary).
3. Creating the file.
4. Opening the file.
5. Preallocating its extents.
6. Copying the contents of the file from tape to disc.
7. Closing the file.

As with the BACKUP2 measurements, the RESTORE2 measurements were restricted to large files, and the overhead associated with such operations as purging, creating, and opening was not included.

Dump-loop Algorithm

The RESTORE dump-loop algorithm is as follows:

```
LOOP
  READ( Tape, Buffer, Wait );
  ! Verify tape record checksum.
  AWAITIO( Discfile );
  WRITE( Discfile, Buffer, Nowait );
ENDLOOP;
```

Redesign of Dump Loop

Because of the DP1 4K-byte limitation, the nowait write of a 30K-byte record in this dump loop consists of seven wait disc writes and only one nowait write. This limits RESTORE's performance, as described below.

The RESTORE dump loop is essentially the BACKUP dump loop with the roles of the tape and disc file reversed. Measurements showed, however, that DP1 BACKUP is over twice as fast as DP1 RESTORE (140,000 bytes/second vs. 65,000 bytes/second). The difference was commonly attributed to the longer amount of time required for a disc WRITE (especially if the drive were mirrored) than for a disc read.

Measurements showed that parallel mirrored-disc writes took only 43% longer than disc reads. A major part of the difference in speed between BACKUP and RESTORE was attributable to the seven wait disc writes that were executed serially with the wait tape read (and, thus, were part of the TAPE^LOOP).

RESTORE2's dump loop was redesigned to perform a nowait tape read in addition to performing the last disc write as a nowait write. This improved the performance of RESTORE2 62% over that of RESTORE with DP1.

DP2 BULKIO, Longer Transfers, and Larger Channel Bursts

RESTORE2 takes advantage of DP2 BULKIO. It uses one nowait write for a 30K-byte record, thus eliminating the seven waited 4K-byte writes required with DP1. With the longer transfers of the 3107 disc controller and the larger bursts of the TRIDENT tape controller, the data can now be restored at 290,000 bytes/second, resulting in a 346% improvement over the performance of RESTORE.

Conclusion

The use of DP2 BULKIO, longer data transfers available with the 3107 disc controller, larger channel bursts available with the TRIDENT tape controller, and the redesign of the BACKUP and RESTORE dump loops have resulted in significantly improved performance for the new GUARDIAN II BACKUP2 and RESTORE2 programs.

The performance of these two programs can be improved even further with improvements to the microcode and software. Enhancements under consideration for future releases include larger channel bursts on the 3107 controller and further improvements to the RESTORE2 algorithm.

Acknowledgments

The authors would like to thank to Jim Enright, Jim Gray, Praful Shah, Andrea Borr, Mitch Butler, Tim Hallock, Franco Putzolu, Pat Beadles, Dennis Markt, Susan Troung, Larry McGowan, Peter Oleinick, and Gil Siegel for their help in collecting the BACKUP2 and RESTORE2 performance information and for reviewing this article.

Anil Khatri is a member of the Performance group in Software Development. Before joining Tandem in 1983, he obtained a B.S. in Electrical Engineering from the Indian Institute of Technology at Kanpur and an M.S. in Computer Science from the University of Maryland.

Matt McCline joined Tandem Software Development in September 1982. At that time, he enhanced and maintained the File Utility Program (FUP), Peripheral Utility Program (PUP), and BACKUP and RESTORE utilities. In the past year, he worked on BACKUP2 and RESTORE2. Recently, he began working in the Transaction Monitoring Facility (TMF) group. Matt has a B.S. in Electrical Engineering and Computer Science from U.C. Berkeley.

VIEWSYS: An On-line System-resource Monitor

A new Tandem utility program, VIEWSYS, now allows users to monitor system resources while their system is running. VIEWSYS displays the percentage used of selected resources in bar-graph form on a Tandem 6520 or 6530 terminal. Users can display a variety of screens showing the usage of one resource across all processors in a system or all resources within a set of processors. Users define which processors in a system are to be monitored.

VIEWSYS is useful as a first-cut system-balancing tool because it provides a dynamic view into the system while applications are running. It is especially helpful for monitoring the effects of program or file relocation. This article provides a brief overview of the functions and commands available with VIEWSYS.

System Monitoring

To assure the best performance possible from its systems, Tandem has made available a number of balancing and tuning tools. At one end of the spectrum is XRAY, a very detailed performance-analysis tool. An XRAY measurement is taken on an active system, and the subsequent output is studied to determine bottlenecks or imbalances within that system. Although users can view XRAY output as it makes the measurements, common practice is to interpret the results after the measurement so that the process of viewing the output has no effect on the statistics gathered.

At the other end of the spectrum are the lights on the processor panel, which users can view to obtain statistics on processor performance. From the command interpreter (COMINT), users can issue the LIGHTS ON command to cause the percentage of processors busy to be displayed by the panel lights. Lights one through ten indicate the percentage of time that that processor was not idle during the polling period (generally one second); each light indicates 10 percent.

PEEK is another utility for monitoring processor performance. It reports statistical information about GUARDIAN and GUARDIAN 90 control blocks, pools, and physical memory. PEEK produces single or multiple "snapshots" containing current and maximum resource use within a specified processor. Its *maximum* counters, initialized with the PEEK INIT command, report the maximum use of those resources over time. For statistics such as page faults, users can generate a "snapshot" at a known time, generate another "snapshot" later, and determine average use during the elapsed time.

The need for a utility that monitors and displays current resource usage on-line led to the creation of VIEWSYS. Intended as an addition to the above tools, it reports a subset of the statistics they provide, in a graphical, interactive, on-line fashion. VIEWSYS is available for NonStop II, TXP, and EXT systems running the GUARDIAN 90 operating system.

Support

VIEWSYS is an economically priced, limited-support software product. There is no local Tandem office support for this product; instead, users can submit questions or problems by filling out the form in the back of the VIEWSYS manual and mailing it directly to Tandem headquarters. Generally, a reply to a query is returned within two weeks.

VIEWSYS is an added-value utility that has no potential impact on the operating system or user applications and therefore does not require locally based systems-analyst support. Offering the product with this limited support permits Tandem to offer it at an attractive price.

Customers contractually agree to the support terms and are licensed to use VIEWSYS for a one-time fee. VIEWSYS is part of the B*nn* software-release series, but there is no guarantee that it will be part of subsequent release series.

Program Function

VIEWSYS is designed to dynamically reflect system-resource use. Users can view the current or maximum allocation of control blocks, system pools, and physical memory. Information is available on page-fault rates, CPU-busy rates, dispatch rates, disc-I/O rates, and interprocessor send-busy rates. Processor queues and page-fault queues are displayed.

Users can display the HELP screen (shown in Figure 1) by pressing a function key. This screen displays function-key assignments and the current attributes of the VIEWSYS program. Users can select individual resources with unshifted function keys and individual processors with shifted function keys. They can alter other VIEWSYS program attributes with special function keys.

Figure 1.
The VIEWSYS HELP screen. Users control the running program by pressing function keys. The top section shows the keys used to control program attributes. The center section lists the entities reported by the program and the function keys depressed to select the entities. The bottom section depicts the current program attributes.

Figure 1

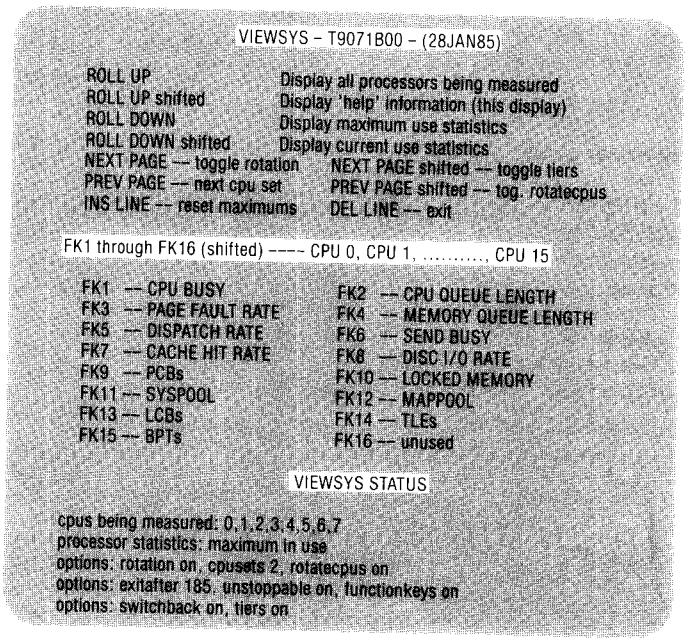


Table 1.
Entities measured by VIEWSYS.

Name	Meaning
CPUBUSY	The percentage of time a processor is not idle during the polling period (processor-busy + interrupt-busy time).
CPUQUEUE	The average number of processes on the ready list during the polling period (processes that are ready to run).
PAGEFAULT	The average number of page faults that occurred, per second, during the polling period.
MEMQUEUE	The average number of processes awaiting page-fault servicing during the polling period.
DISPATCH	The average number of dispatches, per second, during the polling period.
SENDBUSY	The percentage of time during the polling period that a send was being performed within a processor.
CACHEHITS	The average number of disc-cache hits, per second, during the polling period.
DISCIO	The average number of disc I/Os, per second, during the polling period.
PCB	The number of Process Control Blocks allocated at the time the processor is polled.
LOCKEDMEM	The number of pages of physical memory that are locked at the time the processor is polled.
SYSPOOL	The number of pages of SYSPOOL allocated at the time the processor is polled.
MAPPOOL	The number of pages of MAPPOOL allocated at the time the processor is polled.
LCB	The number of Link Control Blocks allocated at the time the processor is polled.
TLE	The number of Time List Elements allocated at the time the processor is polled.
BPT	The number of break points allocated at the time the processor is polled.

Figure 2.
VIEWSYS command syntax. Users issue keyword commands to determine the program attributes. They can set the items highlighted in gray at run time or during program execution with function keys.

Figure 2

```
VIEWSYS [ / <run-option> [ , <run-option ]... / ]
                [ <command> [ ; <command> ]... ]

    <run-option> is one of:

        NAME [ $<process-name> ]
        CPU <cpu-number>
        PRI <priority>

    <command> is one of:

    BACKUPCPU <cpu-number>
    CPUSETS <number>
    DELAY <number-seconds>
    DISPLAY <display-type>
    EXITAFTER <number>
    FUNCTIONKEYS { ON | OFF }
    MULTITYPE { ON | OFF }
    NUMCPUS <number>
    RESERVELCBS { ON | OFF }
    ROTATECPUS { ON | OFF }
    ROTATION { ON | OFF }
    SWITCHBACK { ON | OFF }
    TIERS { 1 | 2 }
    UNSTOPPABLE { ON | OFF }
    UPCPUS { ON | OFF }
    USERCPUS <cpu-number> [ <cpu-number> ]...
```

Measurement Techniques

Table 1 shows the entities measured by VIEWSYS. The values displayed are taken from GUARDIAN 90 counters also used by XRAY and PEEK. VIEWSYS does not alter any of these counters and, therefore, can be run concurrently with these programs. More than one VIEWSYS program can run concurrently on a system.

Using VIEWSYS

The three standard run options for VIEWSYS are: NAME, CPU, and PRI. Following the run options, optional commands can be entered to configure VIEWSYS (see Figure 2). The run commands allow users to specify program attributes such as the processors to be monitored (USERCPUS 1 2 3), the initial display format (DISPLAY CPUBUSY), the polling period in seconds (DELAY 5), and whether to rotate from display to display (ROTATION ON).

Once VIEWSYS is running, its attributes can be altered with function keys. In Figure 2, attributes that can be set with both run-time commands and function keys are highlighted in gray.

Current/maximum Mode

When first started, VIEWSYS reports *current usage*. This is the percentage and amount of each resource in use during that polling period; each display reflects the average use since the last display.

Users can press a function key to alter the display to *maximum mode*. For this mode, internal maximum counters are kept for each resource in each processor being monitored. When users select maximum mode, VIEWSYS displays the highest percentage and amount of each resource recorded. These counters are initialized when the process starts, and users can also initialize at any time by pressing a function key. This allows them to reset the maximums at a recorded time and then display the maximum values occurring since that time. This can be useful in finding under-utilized resources.

ROTATION and ROTATECPUS Commands

Users can set the program to rotate through all possible display types by setting ROTATION ON. Setting it ON causes the sequential display of all processors being monitored. Setting it OFF causes the currently selected display to be repeated. Function keys are used as toggles to alter the state of these two attributes.

Bar-graph Percentages

Figure 3 shows the screen display generated by the USERCPUS command. Nine of the percentages reported are based on the actual configured amounts of the resource. System-pool sizes are determined at system generation (SYSGEN) time, as are the number of control blocks allocated. These amounts are used to determine the percentage used. CPUBUSY and SENDBUSY percentages are determined by dividing the busy time by the total elapsed time.

Six entities have no absolute maximums, so maximums have been established for them for use by VIEWSYS. Table 2 lists these maximums for the NonStop II and NonStop TXP processors. The values have been selected so that the display of a high percentage indicates potential problems resulting from contention for processor resources. Because of the faster processing speed of the NonStop TXP processor, the estimated maximums for its dispatch and disc I/O rates are set higher than those for the NonStop II processor.

Figure 3

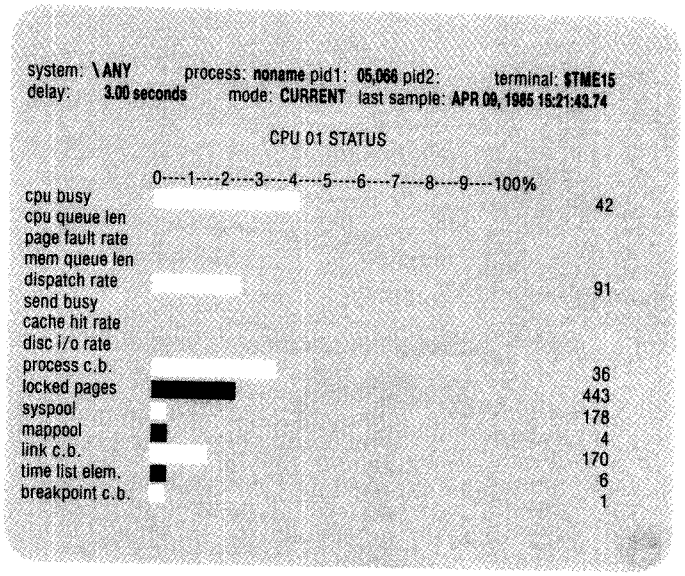


Figure 3.

A screen generated by the VIEWSYS USERCPUS command. The bars represent percentages of utilization. The numbers to the right of the bars represent either the average number of control blocks, memory pages, etc., or the rate (per second) of page faults, cache hits, etc.

Table 2.
Values for entities used as maximums in VIEWSYS.[1]

	NonStop II processor	NonStop TXP processor
CPU queue	10	10
Page-fault rate	25	25
Memory queue	6	6
Dispatch rate	400	800
Cache-hit rate	100	100
Disc-I/O rate	50	100

[1]Number in queue or number per second.

Conclusion

VIEWSYS extends the capability of Tandem's system-monitoring facilities by providing flexible, real-time, graphical displays of system-resource usage. It is useful for both locating problems and balancing resource use within a system.

Dale Montgomery is an account analyst in the Seattle district. When he wrote this article a few months ago, he was a staff analyst for the Field Productivity Programs group in Sunnyvale. His previous duties at Tandem included instructing both System Management and GUARDIAN Operating System courses and programming in the Tandem Application Language (TAL).

Introducing TMDS, Tandem's New On-line Diagnostic System

The Tandem Maintenance and Diagnostic System (TMDS), a new diagnostic software system for Tandem hardware, is available in the B00 release. The FOX diagnostic, a TMDS subsystem for diagnosing problems with the 6700 Fiber Optic Extension (FOX), is also available.

TMDS aids Tandem customer engineers (CEs) and users in many ways. To CEs, TMDS provides a uniform interface to its diagnostic subsystems and an extensive "help" facility for using these diagnostics. The diagnostics themselves provide a safe on-line mechanism for finding and repairing broken hardware. TMDS can run concurrently with user applications, causing fewer interruptions in system activity. System managers who have TMDS running on their systems will find that less time is now required by CEs to determine the location and causes of hardware faults.

System-management considerations for installing and running TMDS software are minimal. The installation instructions, in a step-by-step format, are included in the TMDS software documentation (Softdoc). It is important that system managers install TMDS as soon as possible, instead of waiting until a CE needs to use it. As described in this article, two processes ($ZENO and $ZLOG) should be running at all times.

For Tandem users interested in diagnostics, the rest of this article describes the design of TMDS and the facilities it provides for Tandem CEs. A brief description of the FOX diagnostic is included as an example of a TMDS application. The article concludes with a discussion of potential enhancements to TMDS.

On-line Diagnostics for Fault-tolerant Systems

On-line diagnostics are important to the smooth and uninterrupted functioning of fault-tolerant systems. Users of Tandem systems demand high availability of their applications. Having to curtail all or even a portion of their normal activity to analyze a failure presents a major hardship to their operations staff.

The need for concurrent execution of diagnostics and user applications places several restrictions on on-line diagnostics. The

diagnostics must not usurp any computer resource, from CPU cycles to disc-access paths, without good reason and user permission. In addition, the running of these diagnostics must not compromise data security: control must be given only to authorized users.

Fault-tolerant machines ease the diagnostic task. In these machines, normal system activity continues even after a hardware component fails. The faulty component freezes its state, allowing diagnostics to search for the cause of the fault.

To find the fault, on-line diagnostics must have support from hardware controllers. When possible, fault detection and fault isolation should occur in the hardware, firmware, or software where the fault occurs. Many times, the location of the fault is known at the controller level and should simply be reported to the user. On-line diagnostic systems must support a method of getting this information to the user.

Other faults, such as disc retries, may not stop a system from continuing, but information about their causes can be lost when the system continues. In any case, pertinent information about faults must be stored for future reference by diagnostics.

The TMDS Approach

The design goals for TMDS were to:

- Support diagnostics that allow a CE to diagnose hardware in the shortest possible time while not interfering with normal system activity.
- Decrease the time required by a CE to learn how to use a new or infrequently used diagnostic.
- Minimize the potential for CE diagnostic errors and provide a mechanism for explaining the errors that do occur.
- Create a facility that records information about unexpected hardware faults, providing the CE with all information relevant to a particular failure.

The design of TMDS includes the following features associated with these goals:

- Utilization of hardware and firmware features that allow diagnostics to test one part or path of the hardware while leaving the other parts available for normal system activity.
- Testing that identifies the correct field-replaceable unit (FRU) to be replaced.
- A standard command interpreter for all TMDS diagnostics that provides consistent syntax and extensive help for all TMDS commands.
- A mechanism that stores information on abnormal events that occur during the system's activity. This information is accessible both by the CE and, more importantly, by the diagnostic code, to determine the cause of the problem, and, in certain cases, the action required to fix it.
- Use of NonStop processes and fault-tolerant Tandem hardware to insure that TMDS is always available. If any component of a Tandem system or network fails, the system continues and a CE can use TMDS to diagnose the faulty hardware. In addition, TMDS ensures that diagnostics cannot leave a device in a diagnostic state.

Implementation

TMDS, by itself, does no direct diagnostic work. This is done by its diagnostic subsystems and the CEs that use them. It provides the CEs with a single, powerful, consistent command interpreter for all diagnostics that run under the TMDS umbrella.

TMDS is designed to handle two types of activity. The first is activity requested by the CE through diagnostic commands. The second is the activity initiated when TMDS is notified of a failure in the system.

Diagnostics Run by the CE

Figure 1 shows the interaction of the TMDS components initiated when a CE uses TMDS. The components are described below.

End-user Interface. The end-user interface (EUI) is the TMDS process started by a CE. It is a general-purpose command interpreter, which, as a front end to all TMDS diagnostic subsystems, guarantees that diagnostic commands have a consistent syntax.

The commands consist of a command name and optional parameters. A parameter has two parts: an optional name and a mandatory value. For example, the following are valid commands in TMDS:

```
TMDS> TEST LBU x, TESTNAME all
TMDS> TEST x, all
```

where TEST is the command name, LBU and TESTNAME are parameter names, and *x* and *all* are parameter values. (TMDS> is the TMDS prompt.)

Figure 1

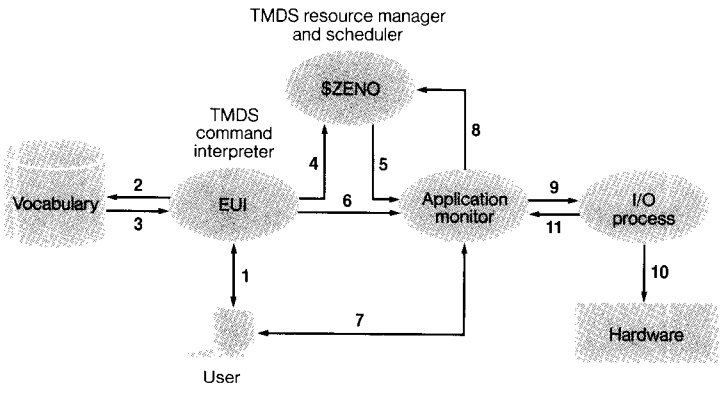


Figure 1.

Components of TMDS. A diagnostic command is entered by the CE with the end-user interface (EUI) and is verified against the vocabulary. Next the resource manager and scheduler ($ZENO) starts the application monitor (AM) in which the command is executed. Then the test microcode is downloaded into the device and the test is run. The resource manager is a NonStop process pair that returns resources allocated during diagnostic work to their initial state if an error occurs.

A benefit of the two-part parameters used in TMDS is that a parameter's name can be given in any position in the command. The following example, which has its parameters specified in a different order from that of the first example above, is also valid:

```
TMDS> TEST TESTNAME all, LBU x
```

Another benefit of two-part parameters is that a *default parameter* can be set. In TMDS, it is possible to specify that all commands having the parameter PARM are to have the value VALUE for that parameter. This mechanism is useful when a parameter is used frequently in several different commands or in the repetition of a single command.

The command DEFAULT is used to establish this relationship. The name of the parameter in the DEFAULT command is the parameter name to be defaulted; the value is the default value. For example,

```
TMDS> DEFAULT LBU x
```

associates the parameter name LBU to the value *x*. After this DEFAULT command is issued, the LBU parameter defaults to the value *x* for all commands that use that parameter unless the user specifies a value for LBU on the command line. The following example is now equivalent to the first example given above:

```
TMDS> TEST TESTNAME all
```

The EUI also presents a unified "help" facility for all diagnostics. There are two ways to obtain help information in TMDS. Use of the help operator, *?*, is one way. Anywhere in a command line, the help operator can be used to ask, "What are my options?" For example, if CEs cannot remember the values for the LBU parameter in the TEST command, they can use the help operator to obtain this information as shown in the following example:

```
TMDS> TEST LBU ?
```

If they cannot remember the name or values of the second parameter for TEST, they can type *?* in the second parameter location to get this information:

```
TMDS> TEST LBU x, ?
```

If the help operator is used in place of the command name, a list of all available commands is displayed. The help operator is best applied to obtain a particular piece of information.

The HELP command and the ERROR command can also be used to get assistance in issuing commands and analyzing errors. The HELP command supplies information in many areas, including general information, rules, and commands for TMDS, and general information about the FOX diagnostic.

For each of these areas, specific information on many subjects is available. One can get help on entering commands by asking for the rules about a general command:

```
TMDS> HELP RULES command-entry
```

One can get an overview of the commands available to diagnose FOX hardware by asking for the FOX overview:

```
TMDS> HELP FOX overview
```

Typing HELP without any parameters displays an introduction to TMDS. Included is assistance on all aspects of HELP. One can always find the complete details of any particular command by typing:

```
TMDS> HELP COMMAND <command-name>
```

or

```
TMDS> HELP <command-name>
```

The ERROR command explains error messages. All TMDS errors are printed in this format:

```
****FOX999: Example FOX error message
```

The number of asterisks (one to four) indicates the severity of the error. The asterisks are followed by an error identifier (in this example, FOX999) and a short description of the error.

A more complete description of the error can be obtained by using the ERROR command with the error identifier as a parameter. For the above error, one would enter the following to get a description of the error:

```
TMDS> ERROR FOX999
```

Vocabulary. When Tandem developers create a TMDS diagnostic subsystem, they define its commands in the TMDS *vocabulary*, a data base for information about command names, command parameters, and the location of the diagnostic code that executes a particular command. When the CE enters a command, the EUI asks the vocabulary to validate the command. The command is compared with the information stored in the vocabulary, and if it was incorrectly or incompletely specified, the EUI displays an explanation of the error. If the command is valid, the information associated with that command is used to run the command.

Commands for a particular hardware device are grouped together within the vocabulary into *subsystems* and can only be used within that subsystem. For example, the FOX TEST command for testing FOX hardware can only be started in the FOX subsystem. TEST commands for other hardware devices are available in other subsystems. The commands provided by the EUI can be executed in all subsystems.

TMDS provides two ways to switch between subsystems. The first is to specify the subsystem on the TMDS run line. The command sequence below starts TMDS in the FOX subsystem where all FOX commands can be run:

```
:TMDS FOX
```

The other way to switch to a different subsystem is to use the SWITCHTO command:

```
TMDS> SWITCHTO fox
```

Resource Manager and Scheduler ($ZENO). When a resource is being diagnosed, it should not be available to normal users or other diagnostics; if it were, the diagnostic might act unpredictably. To eliminate contention for the resource being diagnosed, the resource is always *allocated* to a diagnostic before it is used. Once the allocation is made, the diagnostic can be run. When it completes, the resource must be deallocated. (For the FOX hardware, allocation consists of putting the hardware into the diagnose state; deallocation returns the device to the stopped or started state.)

The device must be deallocated regardless of whether the diagnostic succeeds or fails. If a diagnostic ends abnormally, the TMDS resource manager and scheduler ($ZENO) makes the required deallocation. $ZENO handles all allocation and deallocation of resources.

$ZENO is a licensed NonStop process pair. It can recover from any single hardware failure. If a diagnostic process stops prematurely, $ZENO stops all associated diagnostic work on that resource and restores the resource to a known state (e.g., the stopped state with the standard microcode loaded.) The CE can then restart testing. $ZENO must run at all times for TMDS to work.

$ZENO also handles all TMDS process creation. In later releases of TMDS, this may include the ability to schedule TMDS commands to run at a particular time.

Application Monitors. TMDS runs all diagnostics separately from the EUI. Each of these processes is called an *application monitor (AM)*. An AM consists of TAL procedures.

When a command is validated, the name of the AM and the procedure that implements the command (as stored in the vocabulary) are returned to the EUI. The AM is created, the parameters of the command are passed to this AM process as a parameter stack stored on the AM's stack, and the specified procedure is called. The mechanism used to pass parameters to a procedure in a different process is called a *remote-procedure call.*

Executing the diagnostic as a process separate from the EUI has the advantage that, if a diagnostic ends abnormally for any reason, the EUI remains unaffected, allowing the CE to continue. This advantage would be outweighed by the disadvantage of having to create and communicate with an AM every time a command is run if it were not for the remote-procedure call that handles this work. The remote-procedure call mechanism uses $ZENO to save information on AMs. In this way, AMs can be reused, thus minimizing the number created.

The procedure name to execute, the values for the parameters for that procedure, and the AM where that procedure's code can be found must be supplied to the remote-procedure-call mechanism. This mechanism creates the stack frame (identical to that created by the TAL compiler when a normal procedure call is made) by using BINDER information about the specified procedure call. This stack frame is sent (via a standard GUARDIAN 90 message) to the remote-procedure-call mechanism in the AM where the specified procedure is to be run. In this AM, the remote-procedure-call mechanism receives the stack frame and adds it to the stack. A call is then made to the appropriate procedure.

Standard utility procedures bound into every AM simplify the allocating of resources, formatting of text output, and printing and formatting of standard errors. Only one copy of these procedures need be present in an AM for all commands in an AM to use them. Having these utility routines common throughout all device diagnostics creates coherence and consistency, thus making it easier for the CE to troubleshoot problems.

AMs also provide a convenient way of securing diagnostics. As it is unacceptable to allow just any user to run a diagnostic, TMDS uses GUARDIAN 90 operating-system security to limit diagnostic use. The object code of the EUI and the AM associated with a diagnostic command can be secured so that only a certain set of users can run a particular diagnostic. (For example, while the FOX STATUS command can be executed by anyone, only SUPER.CE can execute the TEST command.) The system administrator controls access to these diagnostics through the allocation of user passwords.

Executing a TMDS Command. The following is an example of what happens when a TMDS command is executed. In this example, a FOX LBU has encountered an error and the CE uses TMDS to diagnose it. The CE starts TMDS in the FOX subsystem, obtains help on the TEST command, and tests the interprocessor-controller (IPC) board in LBU X by looping it through the appropriate microcode test five times.

First, the CE runs the FOX diagnostic:

:TMDS FOX

TMDS responds with:

Welcome to Tandem's Maintenance and Diagnostic System...

The CE asks for help on the TEST command:

TMDS> HELP test

TMDS responds with an explanation of the TEST command:

The TEST command downloads and runs microcoded diagnostics in the FOX LBU. The TEST command has...

Before running the TEST command, the CE defaults the LBU parameter so that the TEST command will test LBU X. (This parameter now need not be specified in any further commands.) The CE enters:

TMDS> DEFAULT LBU x

Then, the CE uses the TEST command:

TMDS> TEST TESTNAME ipc, LOOP 5, DETAIL on

As is illustrated in Figure 1, first the EUI checks the syntax of the command entered at the terminal (1). Command and parameter names in TMDS consist of an alphanumeric sequence of 36 characters or less. Since this command has a command name and parameter names that look valid to the EUI, the command is passed to the vocabulary (2) for a semantics check.

The vocabulary checks the FOX subsystem for a command named TEST (3). It fills the defaults set by the user and checks each parameter to see that the TEST command has a specified or defaulted value for that parameter. The vocabulary also checks the value of each parameter against its acceptable range of values. If the EUI check or vocabulary check fails, a message explaining the error is displayed and the EUI reprompts the CE.

In this example, the command validation succeeds, causing the EUI to make a remote procedure call to $ZENO (4) to have it start the appropriate AM that implements the TEST command (5). The appropriate procedure in the AM is then called (6) with the parameter values *x*, *ipc*, *5*, and *on*.

The TEST command first tries to allocate LBU X through $ZENO (8). If the LBU were already being diagnosed, the allocation would fail, and $ZENO would inform the CE (via the AM) that someone else was already diagnosing that LBU (7).

In this example, the allocation succeeds, and a check of LBUs X and Y is made to verify that bringing down LBU X wouldn't isolate the system. As the system *would* be isolated if LBU X were down, the CE is warned (7) and confirmation for continuing the test is requested.

The CE confirms that the test should continue. The LBU is stopped, and the microcode that implements the IPC test is downloaded into it (9,10). All errors found while the FOX microcode is downloaded and executed are displayed for the CE (11,7).

When the diagnostic completes, the LBU is returned (deallocated) (8) to either the stopped or started state as specified by the CE. If the diagnostic ends abnormally for any reason, $ZENO returns the LBU to the stopped state.

TMDS Activity Initiated Automatically

When a device fails during normal system activity, it is important to be able to save information about the failure for later examination. Figure 2 shows the TMDS components that save this information; their interaction is described below.

Figure 2.
In addition to performing diagnostic activity initiated by the CE, TMDS records and stores information about unexpected hardware failures for later examination. The information is passed from an I/O process to the TMDS logger and is stored in an event log.

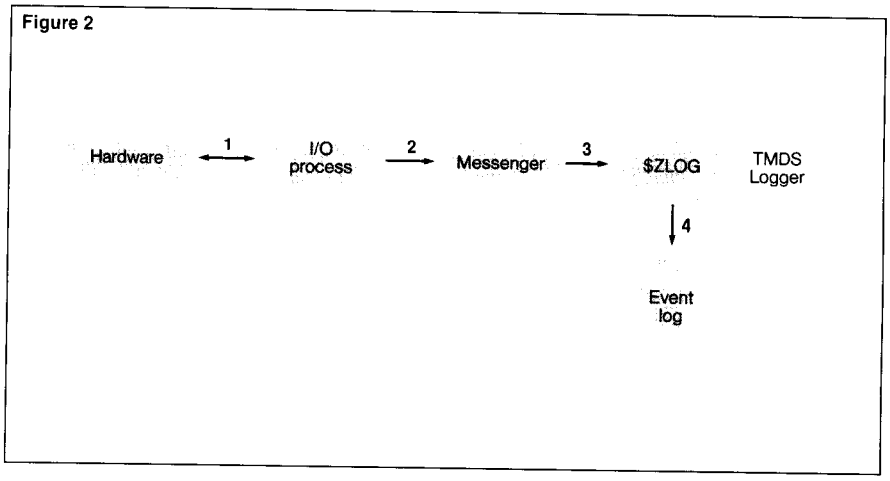


Table 1.
Syntax for the FOX TEST command.

Parameter	Values	Default
LBU	{ X | Y }	(None; a parameter is required.)
TESTNAME	<testname>	ALL. (All tests are run.)
LOOP	<loop>	Once
SYSTEM	<system>	The local system
DETAIL	{ ON | OFF }	OFF. (No detailed output is printed.)

In TMDS, an *event* is any occurrence about which information must be retained for examination by a CE or diagnostic. An occurrence of a hardware error and the running of a test are both considered to be events. All information about an event is encapsulated in an *event signature (ES)*, a standard, variable-length message.

An I/O process that detects an error (1) uses procedures supplied by TMDS to create an event signature. To send this information to TMDS, the I/O process passes the ES to the GUARDIAN 90 operating system messenger process (2) via a procedure call. The I/O process then continues execution.

The messenger process implements the communications protocol needed to pass the event signature to the TMDS logger process, $ZLOG (3); for example, it handles path retries when necessary. This minimizes the overhead the I/O process incurs when sending messages.

The TMDS logger receives the event signatures and stores them in a TMDS event log (4). They can be retrieved by the TMDS commands FIND (used to view event signatures in the event log) and PURGE (used to delete event signatures).

The logger is a nonprivileged NonStop process pair. It should be run at all times to gather as much fault information as possible.

The FOX Subsystem

The FOX diagnostic is used in diagnosing FOX hardware faults. It has four commands: REPAIR, REVUPDATE, STATUS, and TEST. Descriptions of these commands are available from the TMDS HELP facility. The parameters for the TEST command, their possible values, and their defaults are listed in Table 1.

As the previous example has shown, the FOX diagnostic's TEST command is used to download microcode tests into the FOX LBU once that LBU is in the stopped state. Tests for each FOX board are available; a list of TEST names is available in the HELP information.

Note that the TEST command uses the TMDS two-part notation. Each parameter has a name (e.g., DETAIL) and a set of allowable values (e.g., ON or OFF). This feature allows the CE to enter the DETAIL parameter in the third position instead of the fifth position where it is defined to be. (See the TEST command used in the previous example.)

The TEST parameters are used for other FOX commands as well. Common parameter names allow CEs to default a parameter for a number of commands in a subsystem.

Figure 3 is an example of the information displayed with the TEST command. First, the TEST command prints a header (within a box) that summarizes the command the CE specified, including the estimated amount of time it will take the command to complete. After the header, short error messages are displayed; in Figure 3, the messages are for errors FOX400, FOX450, and FOX402. The TEST command finishes with a summary of the tests run and errors returned.

In Figure 3, the CE requests more complete information about the error condition displayed as ***FOX450 and finds that the error message identifies the board that probably caused the error. The CE can now replace that board and try the TEST command again.

TMDS Today

The FOX diagnostic subsystem is a good example of the power and functionality of TMDS because it uses TMDS extensively. All command interaction is managed by TMDS.

FOX diagnostics use the abilities of $ZENO to allocate and deallocate the FOX hardware. The FOX diagnostics and the EUI both use the remote-procedure-call mechanism to minimize the work involved in starting AMs.

Future Enhancements

Future changes in TMDS will focus on three major areas: simplifying the CE's use of TMDS, adding new diagnostic functionality, and providing fault analysis.

Command entry to TMDS can be made easier. An optional menu interface to TMDS is under consideration. Also, a method of having TMDS request command parameters (instead of requiring the user to supply them with the command name) is being examined. As more commands are added to TMDS, this would keep command entry uncomplicated and straightforward.

Commands for maintaining hardware (not just diagnosing problems) may be included in future TMDS releases. With this type of command, a hardware device could be checked periodically to verify that it is in running order. Scheduling maintenance commands to run at a particular time might be supported, allowing the CE to pinpoint a problem area before a problem occurs.

Also, a fault analyzer might be developed to monitor the event log. When possible, the fault analyzer would determine the solution to a problem when it occurred. This could provide the CE with much useful information. The analyzer could include an alarm that would alert the CE at CE headquarters of failures at a user site.

Most importantly, TMDS will be used for all future Tandem diagnostics. Diagnostics specifications have been proposed for discs and CPUs, as well as other subsystems.

Figure 3

```
TMDS> TEST LBU x, TESTNAME ipc
TMDS FOX DIAGNOSTIC - 28JAN85 - (19DEC84)
FOX TEST COMMAND              INPUT INFORMATION
LBU          :   X
TESTNAME :   IPC
LOOP         :   1 TIME
SYSTEM       :   \LAB3
Time for 1 test set iteration =  0:02:41
Tests will complete at about   16:20:38 (or when BREAK is hit).

   FOX400: Test set pass number 1 started at 16:18:12.
***FOX450: Test error 173 in module IPCDIAG at 3,1.  FRUS: IPC
   FOX402: TEST RESULTS SUMMARY FOR SYSTEM \LAB3, LBU X:

            Test set name      = IPC
            Stop time          = 16:20:45   3 JAN 1985
            Start time         = 16:17:51   3 JAN 1985
            Total test time    =  0:02:54
            # complete repeats = 1
            Total errors       = 1

\LAB3 lbu X is currently in the STOPPED state.
Do you wish to put it on line (Y/N)? y
```

```
TMDS> ERROR fox450

A microdiagnostic has detected an error in the hardware. The module name is
given, along with the test and subtest where the error was found. The
"FRUS:" part of the message gives an ordered list of the FRUs which are the
likely cause of the error.
```

Figure 3.
The output of the TEST command can include information about a number of errors. To conserve space, short messages about each error are displayed. When the ERROR command is entered for any of the error messages, a more detailed explanation of the error message is displayed. In this example, the ERROR command was entered for FOX450. (The commands entered by the CE are represented in bold type. FRU stands for field-replaceable unit.)

References
Tandem Maintenance and Diagnostic System Reference Manual. Part no. 82386 A00. Tandem Computers Incorporated.

Acknowledgments
The author would like to thank Brenda Troisi, Rikki Westerschulte, and Tom Mathieson. Without their help, this article could not have been written.

Jim Troisi joined Tandem in September 1982 as a software designer. Since then, he has worked on the design and implementation of TMDS. Before working for Tandem, Jim was a graduate student at the Massachusetts Institute of Technology.

B00 Software Manuals

With the B00 release, Software Publications offers you a new manual set: 85 software manuals sporting a new design, presenting information in a new format, and representing the continuing efforts of the writers, editors, designers, and staff to produce a quality product.

The manual covers are now black, displaying the new Tandem logo and a stripe of color to represent the manual library to which they belong.

To better aid users, Software Publications has produced separate user's guides and reference manuals for many of the Tandem products it supports. The Operating System writers have reworked much of the GUARDIAN 90 information, moving system messages, utilities, and procedure calls into separate manuals.

The Languages writers have reorganized the EXTENDED BASIC and COBOL reference manuals. They've added examples, a second color to highlight sample programs, and even more system information to improve the new EXTENDED BASIC and COBOL user's guides and reference manuals.

For new system users, there is the TGAL tutorial. Experienced TAL programmers have a revised TAL manual. There are also new reference manuals for users of the PATHWAY transaction-processing system and Screen COBOL.

You'll find many more additions and improvements to the software manuals. Table 1 lists the title, part number and version, and status of all the software manuals pertaining to the B00 software release.

Sunny Olds was a lead editor for Software Publications when she wrote this article. She coordinated editorial efforts for the B00 Software Release, edited the Languages and Operating System libraries, and authored the March 1985 *Catalog of Software Publications and Related Products*.

Table 1.
B00 software manuals (NonStop II, TXP, and EXT systems).

Title	Part no. and version	Status	Title	Part no. and version	Status
Operating system					
GUARDIAN Operating System Pocket Guide	82506 A00	New manual	System Description Manual	82507 A00	New edition
GUARDIAN Operating System Programmer's Guide	82357 A00	New manual	System Management Manual	82069 A00	New manual
GUARDIAN Operating System User's Guide	82396 A00	New manual	System Messages Manual	82409 A00	New manual
GUARDIAN Operating System Utilities Reference Manual	82403 A00	New manual	System Operator's Guide	82401 A00	New manual
Introduction to Tandem Computer Systems	82503 A00	New edition	System Procedure Calls Reference Manual	82359 A00	New manual
Spooler Programmer's Guide	82394 A00	New manual	Tandem Maintenance and Diagnostic System Reference Manual	82386 A00	New manual
			XRAY User's Manual	82078 D00	Updated
Tools and utilities					
BINDER Manual	82514 A00	New edition	INSPECT Interactive Symbolic Debugger User's Guide	82315 A00	Unchanged
CROSSREF Manual	82516 A00	New edition	SORT/MERGE User's Guide	82091 A00	Unchanged
DEBUG Manual	82598 A00	New edition	TGAL Manual	82526 A00	New edition
EDIT Manual	82079 B00	Unchanged	Tools and Utilities Pocket Guide	82504 A00	New edition
ENTRY Screen Formatter Operating and Programming Manual	82020 A00	Unchanged	UPDATE/XREF Manual	82080 B00	Unchanged
ENTRY520 Screen Formatter Operating and Programming Manual	82053 A00	Unchanged			
Languages					
COBOL Pocket Guide	82575 A00	New edition	FORTRAN Pocket Guide	82577 A00	New edition
COBOL Reference Manual, Volume 1	82589 A00	New manual	Interim Supplement to the FORTRAN 77 Reference Manual	82436 A00	Supplement
COBOL Reference Manual, Volume 2	82590 A00	New manual	MUMPS Pocket Guide	82578 A00	New edition
COBOL User's Guide	82389 A00	New manual	MUMPS Reference Manual	82542 A00	New edition
EXTENDED BASIC Pocket Guide	82379 A00	New manual	TAL Pocket Guide	82376 A00	New manual
EXTENDED BASIC Reference Manual	82580 A00	New edition	Tandem FORTRAN 77 Reference Manual	82030 A00	Unchanged
EXTENDED BASIC User's Guide	82385 A00	New manual	Transaction Application Language (TAL) Reference Manual	82581 A00	New edition
Data management					
Data Definition Language (DDL) Reference Manual	82534 A00	New edition	PATHAID Reference Manual	82428 A00	New manual
Data Management Pocket Guide	82302 D00	Unchanged	PATHWAY SCREEN COBOL Reference Manual	82424 A00	New manual
ENABLE Reference Manual	82560 A00	New edition	PATHWAY System Management Reference Manual	82365 A00	New manual
ENABLE User's Guide	82571 A00	New edition	Transaction Monitoring Facility (TMF) Reference Manual	82541 A00	New edition
ENCORE User's Guide	82350 A00	Unchanged	Transaction Monitoring Facility (TMF) System Management and Operations Guide	82543 A00	New edition
ENFORM Reference Manual	82348 B00	Updated	TRANSFER Delivery System Management and Administration Guide	82522 A00	New edition
ENFORM User's Guide	82349 B00	Updated	TRANSFER Delivery System Programming Guide	82525 A00	New edition
ENSCRIBE Programming Manual	82583 A00	New edition	TRANSFER/MAIL User's Guide	82599 A00	New edition
Introduction to ENFORM	82313 B00	Unchanged			
Introduction to PATHWAY	82339 A00	New manual			
Introduction to the Transaction Monitoring Facility (TMF)	82528 A00	New edition			
Introduction to TRANSFER Delivery System	82323 C00	Unchanged			

Continued next page

Table 1. (Concluded)
B00 software manuals (NonStop II, TXP, and EXT systems).

Title	Part no. and version	Status	Title	Part no. and version	Status
Data communications					
6100 ADCCP Programming Manual	82411 A00	New manual	EXCHANGE Reference Manual	82568 A00	New edition
6100 MPS-B Programming Manual	82413 A00	New manual	EXPAND Reference Manual	82370 A00	New manual
6100 BSC Programming Manual	82412 A00	New manual	Introduction to Tandem Data Communications	82511 A00	New edition
6100 TINET Programming Manual	82414 A00	New manual	Introduction to the Tandem 6100 Communications Subsystem (CSS)	82373 A00	Unchanged
Communications Management Interface (CMI) Operator's Guide	82544 A00	New edition	Reference Manual for ATP6100 and Non-6100 Terminal/Printer Processes	82435 A00	New manual
Communications Utility Program (CUP) Reference Manual	82430 A00	New manual	SNAX Programming and Network Management	82596 A00	New edition
CP6100 I/O Process Programming Manual	82410 A00	New manual	SNAX Reference Summary	82591 A00	New manual
Device-Specific Access Method—AM3270/TR3271	82432 A00	New manual	Tandem Hyper Link (THL) Reference Manual	82056 A00	Unchanged
Device-Specific Access Method—AM6520	82433 A00	New manual	Tandem-to-IBM Link (TIL) Reference Manual for IBM Users	82050 B00	Unchanged
ENVOY Byte-Oriented Protocols Reference Manual	82582 A00	New edition	Tandem-to-IBM Link (TIL) Reference Manual for Tandem Users	82055 B00	Unchanged
ENVOYACP Bit-Oriented Protocols Reference Manual	82588 A00	New edition	X.25 Access Method—X25AM	82431 A00	New manual
Terminals					
6VI Voice Input Option for 653X Terminal Family Installation and Operation Guide	82671 A00	New manual	Getting to Know T-TEXT	82655 A00	New manual
6530 Alternate Input Option Installation and Operation Guide	82652 A00	Unchanged	Model 6820 Terminal Cluster Concentrator (TCC)—Installation and Operation Guide	82651 B00	Unchanged
6530 Video Monitor Option Installation and Operation Guide	82653 A00	Unchanged	Printer Option for 6530 Terminal—Installation and Operation Guide	82650 A01	Unchanged
653X Integrated Options Installation and Operation Guide	82677 A00	New manual	T-TEXT User's Manual	82656 A00	New manual
653X Multi-Page Terminal Installation and Operation Guide	82508 A00	New edition	Video Display Units Programming Manual	82047 A00	Unchanged
653X Multi-Page Terminal—Programmer's Guide	82310 B00	Unchanged	Video Display Units Operating Guide	82048 A00	Unchanged
EM3270 Option for 653X Family—User's Guide	82668 A00	Unchanged	Word Processing Option for 653X Installation and Operation Guide	82654 A00	New manual
DYNAMITE 654X workstation					
654X Workstation—GW-BASIC User's Guide	82664 A00	New manual	654X Workstation—MS-DOS User's Guide	82661 A00	New manual
654X Workstation—Information Xchange Facility	82669 A00	New manual	654X Workstation—Operations Guide	82658 A00	New manual
654X Workstation—Macro Assembler Programmer's Guide	82662 A00	New manual	654X Workstation—PCFORMAT User's Guide	82679 A00	New manual
654X Workstation—MS-DOS Programmer's Guide	82660 A00	New manual	654X Workstation—Technical Reference	82665 A00	New manual
Supplemental publications					
Catalog of Software Publications and Related Products	82552 A00	New edition	Master Index for the NonStop Systems	82586 A00	New edition

Tandem's Corporate Education Group has added eight interesting new courses to its software education curriculum. It has also revised and improved the PATHWAY course, one of the most popular software classes.

The new courses offered in 1985 are described below. Following their descriptions is a list of all Tandem Education Centers.

As more new courses are offered throughout the year, information about them will be made available. To obtain course information, enroll, or order any self-paced course, contact your Tandem sales representative or your nearest Tandem Education Center.

PATHWAY (38404-B00)

The popular PATHWAY course for application developers has been improved. The new class still covers all the topics described in the *Software Education Catalog*, and it contains the following new material:

- Use of the INSPECT symbolic debugger to isolate problems.
- Stress testing and tuning applications.
- More comprehensive labs so that each student develops a complete working PATHWAY system.

DP1-DP2 Conversion Course (38449-A00)

This five-day course provides a description of the differences between Disc Process 1 (DP1) and Disc Process 2 (DP2), and a description of the new features of DP2. Topics include conversion issues and strategies, new features of the Transaction Monitoring System (TMF), and DP2 support tools.

At the end of the course students will be able to plan for a conversion of a system from DP1 to DP2, do the appropriate SYSGENs in order to implement the conversion, and be able to configure and test a DP2-TMF system.

Audience

The course is designed for analysts and system managers who will be supporting a DP2 installation.

Prerequisites

A working knowledge of TMF and experience as a system manager are required.

Multifile ENABLE — Self-paced (38455-A00)

This is an intensive independent study course to familiarize analysts and programmers with the newest version of the ENABLE program generator. Students will learn the differences between single and multifile ENABLE, the ENABLE syntax, its capabilities, and how to integrate programs generated by ENABLE into a unified PATHWAY

environment. The course provides an in-depth look at the generated Screen COBOL code and the benefits of making easy (and not so easy) code changes with EDIT. The material also reviews the ENABLE skeleton command language, alteration of the skeleton file, and use of the SET BOX FLAG command.

Audience
This course is designed for programmers and application developers who will be generating interactive applications with ENABLE.

Prerequisites
A knowledge of FUP, EDIT, DDL, and PATHWAY would be beneficial.

TRANSFER (38440-A00)

This five-day course covers installation and programming of TRANSFER, Tandem's delivery system for staged information distribution. Topics include: processes that comprise the TRANSFER environment, installation procedures, and programming clients and agents. Lab sessions supplement classroom lectures and provide hands-on practice.

Audience
This course is intended for system administrators and application programmers. The modular design of the course permits each group to attend that portion of the class relevant to its job functions.

Prerequisites
Prerequisites include:

- Concepts and Facilities (38401)
- PATHWAY (38404)
- Transaction Monitoring Facility (38413)
- System Resource Management (38411 or 38419). This is a prerequisite for system administrators.

TAL Syntax — Self-paced (38433-A00)

This independent-study course covers the fundamentals of TAL (Tandem's systems programming language) syntax structure and use. Topics include: data types and expressions, statements, addressing procedures and parameter passing, program organization, debugging programs with INSPECT, and the TAL programmatic interface to the GUARDIAN operating system.

Audience
This course is designed for systems programmers, application developers, system operators, or anyone involved in writing programs in a high-level systems language. It is recommended as a prerequisite to the TAL Programming course for people with no previous experience of block-structured languages.

Prerequisites
The prerequisites include:

- Knowledge of at least one other programming language.
- Familiarity with Tandem EDIT.
- Completion of the Concepts and Facilities course. (This would be helpful, but is not mandatory.)

T-TEXT — Self-paced (38456-A00)

This self-paced tutorial course complements the *T-TEXT User's Manual* by providing hands-on instruction in the basics of document preparation using Tandem's word processing software. The course can be completed in just two or three hours.

Ten lessons step the user through the most commonly used T-TEXT functions. The T-TEXT menus are explained, along with the options available from each menu. Also included is a handy quick-reference guide on T-TEXT commands.

Audience
This course is aimed at nontechnical personnel who use word processing in their jobs.

Prerequisites
There are no prerequisites for this course.

An Introduction to SNAX — Self-paced (38438-A00)

Tandem's System Network Architecture Communications Services (SNAX) enables SNA devices and host computers to communicate and share applications with Tandem systems and EXPAND networks. This self-paced course introduces SNAX and how it relates to the IBM environment.

Students learn how to perform a SYSGEN to set up an SNA environment, how to use CMI to alter the SNAX environment, and how to perform first-level troubleshooting.

Audience
This course is intended for system managers, operators, analysts, data communications specialists, and anyone who needs to set up or maintain a SNAX application. The course does not teach or assume a knowledge of SNA.

Prerequisites
The only prerequisite is an ability to perform a SYSGEN.

6100 Communications Subsystem Primer — Self-paced (38441-A00)

This is an intensive independent-study course which introduces the 6100 subsystem to system operators and system managers. Students learn the required components of SYSGEN and a naming convention.

This course provides a comprehensive overview of CMI/CMP as they relate to the 6100, an overview of system-management procedures, and the use of DIAG6100 as a system management tool. The material also provides an overview of TRACE and PTRACE functions, and a detailed overview of the 6100 hardware.

Audience
This course is intended for system operators and system managers.

Prerequisites
There are no prerequisites.

Introduction to the DYNAMITE Workstation — Self-paced (9405)

This self-paced tutorial course includes a getting-started booklet and a floppy disc. It allows students to learn about and experiment with the DYNAMITE 654X workstation in an interactive, protected environment.

Topics include: the DYNAMITE keyboard, starting DYNAMITE, an introduction to the Microsoft Disk Operating System (MS-DOS), how to name files, how to use the 6530 terminal emulator, and how to enter and exit BASIC. The course takes about two hours.

Audience
This tutorial is designed for nontechnical users who have had no previous experience with a personal computer.

Prerequisites
There are no prerequisites.

Marilyn Janow is the Manager of Corporate Education Services. She is responsible for publication of software and hardware training courses and education administration. She is also responsible for producing education catalogs, schedules, and brochures about Tandem training; supporting special education requirements for certain customer projects; and administering the Alliance coupon program for software houses. Marilyn transferred to the Corporate Education group in September 1982 and has been with Tandem since April 1981.

Tandem Worldwide Education Centers

United States

Arizona
3300 N. Central, Suite 700
Phoenix, AZ 85012
(602) 264-2206

California
6160 Bristol Parkway
Culver City, CA 90230
(213) 417-3922

2820 San Tomas Expressway
Santa Clara, CA 95051
(408) 970-4324

1309 South Mary Avenue
Sunnyvale, CA 94087
(408) 730-3700

Colorado
5300 DTC Parkway
Englewood, CO 80111
(303) 779-6766

Florida
1408 N. Westshore, Suite 804
Tampa, FL 33607
(813) 877-7466

Georgia
100 Galleria Parkway, Suite 680
Atlanta, GA 30339
(404) 951-0199

Illinois
Hamilton Lakes
500 Park Blvd., Suite 1400
Itasca, IL 60143
(312) 773-1750

Massachusetts
7 Wells Avenue
Newton, MA 02159
(617) 964-6500

Michigan
3001 State Street, Suite 1010
Ann Arbor, MI 48104
(313) 622-2200

32540 Schoolcraft Avenue
Livonia, MI 48150
(313) 425-4110

Minnesota
3050 Metro Drive, Suite 205
Minneapolis, MN 55420
(612) 854-5441

New Jersey
777 Terrace Avenue, Second Floor
Hasbrouck Heights, NJ 07604
(201) 288-6050

New York
One Penn Plaza, Suite 4334
250 W. 34th Street
New York, NY 10119
(212) 760-8440

Pennsylvania
Commerce Court Office Building
4 Station Square, 7th Floor
Pittsburgh, PA 15219
(412) 562-0262

Texas
4225 Wingren, Suite 100
Irving, TX 75062
(214) 257-9744

Virginia
12100 Sunrise Valley Drive
Reston, VA 22091
(703) 476-3154

Washington
14711 NE 29th Place, Suite 100
Bellevue, WA 98007
(206) 881-8636

Washington, D.C.
5201 Leesburg Pike, Suite 700
Falls Church, VA 22041-3280
(703) 379-7900

International

Australia
3 Bowen Crescent
Melbourne, Victoria 3004
011-61-3-267-1577

22 Atchinson Street
St. Leonards, Sydney, NSW 2065
011-61-2-438-4566

Belgium
Louizalaan 306-310
1050 Brussels
011-32-2-6487330

Canada
7270 Woodbine Avenue, Second Floor
Markham, Ontario L3R-4B9
(416) 475-8222

6500 Route Trans-Canadienne
St. Laurent, Quebec H4T 1X4
(514) 342-6711

Denmark
Helgeshoj Alle 55
DK-2630 Taastrup
011-452-5252-88

England
Tandem House, Mendy Street
High Wycombe, Buckinghamshire
HP112NZ
011-44-494-21277

France
2 - 4 Rue Victor Noir
92200 Neuilly Sur Seine, Paris
011-33-1-738-29-29

Germany
Geschaeftsstelle Duesseldorf
Heinrich-Hertz-Strasse 2
4010 Hilden
011-49-2103-572-100
011-49-2103-572-101

Berner Strasse 34
6000 Frankfurt/Main 56
11-49-69-50003-0

Hong Kong
448 Wing On Plaza
Tsim Sha Tsui East, Kowloon
011-852-3-7218136

Italy
Viale del Ghisallo, 20
20151 Milano
011-39-2-3087386

Japan
Yasukuni Kudan Minimi Bldg.,
3rd Floor
2-13-14 Kudan Minami, Chiyoda-Ku
Tokyo 102
03-237-9531

Netherlands
"Plus Point"
Jupiterstraat 146-3
2132 HG Hoofdorp
02503-30294

New Zealand
Level 20, Williams City Centre
Boulcott Street, Wellington
011-644-723286

Norway
O. H. Bangsvei 51
N-1322 Hoevik
2-123330

Sweden
Norgegatan 1
Box 1254
S-163 13 Spaanga
011-46-8-7507540

Switzerland
Zweierstrasse 138
8036 Zuerich
011-41-1-461-3025

Distributors

Finland
Oy Dava AB
Box 458
00101 Helsinki 10
011-358-0-42021

Mexico
Tandem Computers de Mexico,
S.A. de C.V.
Ave. Juarez No. 14-2 piso
06050 Mexico, D.F.
(905) 585-8688

Philippines
Mini Systems, Inc.
LBH Building, 4th Floor
1431 A. Mabini Street
Ermita, Metro Manila
722-27316

Taiwan
Syscom Computer Engineering Co.
9th Floor
53 Jen Ai Road, Sec. 3
Taipei, Taiwan, R.O.C. iso
(02) 7731302-9

SNAX/HLS: An Overview

A significant addition to SNAX (Tandem's fault-tolerant interface to the Systems Network Architecture, SNA), is available with the B00 software release. This is SNAX High Level Support (SNAX/HLS), an easy-to-use interface that allows programmers with little knowledge of SNA to develop SNA-related applications on Tandem networks. SNAX/HLS dramatically reduces the time required to develop such applications.

This article describes the design and implementation of the SNAX/HLS system. Topics include:

- The components of the SNAX/HLS system.
- The SNA capabilities it supports.
- Its potential applications.
- An application benchmark employing SNAX/HLS.

This information is intended for technical managers, application programmers, systems analysts, and others who need an overview of SNAX/HLS capabilities. Readers are assumed to have a general understanding of SNA, IBM SNA products, and SNAX. Information on IBM SNA products and devices, SNAX, and SNAX/HLS is available in the references listed at the end of the article.[1]

[1]In the interest of brevity, *HLS* is used in place of *SNAX/HLS* throughout the remainder of the article except when the product's full name is appropriate.

Design Goals

With the introduction of SNAX, Tandem systems acquired the ability to communicate directly with SNA software products, such as the Information Management System (IMS), and SNA devices, such as the IBM 3624. SNAX is unique within the industry in providing both a gateway to SNA software products and SNA device support on a CPU that is not IBM-compatible.

SNAX offers two application interfaces:

- *The SNA3270 interface*, a high-level interface connecting the Command Interpreter and Spooler subsystems with SNA display stations and printers.
- *The SNALU interface*, a general-purpose, low-level interface similar to the application interface of IBM's Virtual Telecommunications Access Method (VTAM). The SNALU interface was designed to function as the base interface for new SNAX products as well as to provide the basic framework for specialized offerings from Tandem software houses.

SNAX also supports SNA Session Passthrough, a feature that permits SNA devices attached to a Tandem network to access SNA application programs resident in an IBM host without changes to or reprogramming of the application.

The SNA3270 interface and SNAX Passthrough gained quick acceptance by Tandem customers. Many Tandem installations worldwide use them.

Tandem users and software houses also saw the generality of the SNALU interface as a positive feature. Using SNALU, applications could control the flow of SNA messages

and acknowledgments and could send and process special SNA commands, making the interface well suited for custom applications. SNALU application development required that programmers be highly knowledgeable in SNA message formats and protocols, however, making program development for simple business applications complex.

HLS was developed to fill the need for a high-level addition to the SNALU interface for generic business applications. Its design resulted directly from meetings between Tandem developers and users of Tandem systems. The design goals were:

- *High-level support for SNA communications.* HLS was to handle all aspects of SNA protocols and formats. The programmer interface was to be simple and easily understood by applications programmers with little knowledge of SNA or SNAX. The SNA protocol had to be consistent with the published standards on SNA communications architecture.
- *Consistent interface to the PATHWAY system and other environments.* HLS was to support an interface that allowed both those applications based on the PATHWAY transaction processing system and those not based on it to take full advantage of SNAX capabilities.
- *Support for SNA gateways and SNA devices.* HLS was to support logical sessions to IBM subsystems (e.g., IMS) as well as advanced IBM devices (e.g., the 4700, 3624, and 3650).
- *A full complement of control and support tools.* HLS was to provide easy-to-use utilities to configure, control, and trace the operational environment. Significant interest in a prototyping/simulation tool was also expressed.

Implementation

HLS is a system of processes that addresses the product requirements stated above. The implementation of HLS takes full advantage of Tandem products such as the PATHWAY system and the ENABLE program generator to provide high-level SNA communications.

The HLS process is implemented as an intermediate process between the user application and the SNALU interface of SNAX. This makes the power of the SNALU interface available to programmers while shielding them from the SNA complexities of the interface. Figure 1 illustrates a conceptual overview of HLS in a Tandem network environment.

Verbs

HLS implements an application interface that is independent of requester language. A set of *verbs* at the interface level handles all SNA communications functions requested by the application. The verbs, in essence, make up the high-level application language.

An HLS verb is merely a formatted message from the application program that describes to HLS the function to be performed on behalf of the application. Specifically, it is a set of indicator fields that precede optional user data. The indicator

Figure 1

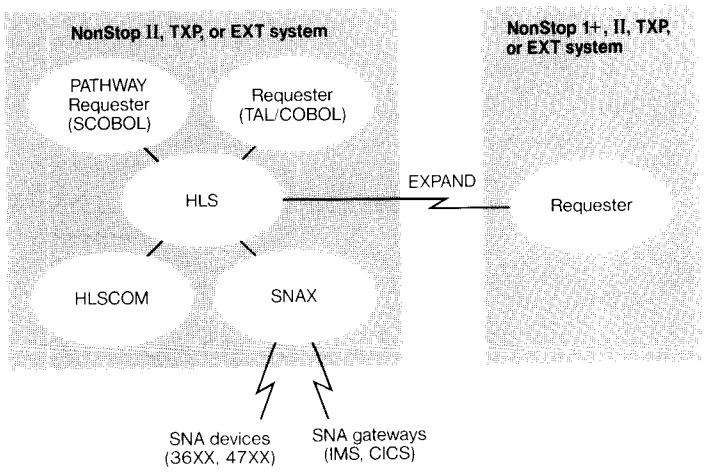


Figure 1.
The SNAX/HLS environment in a Tandem network. HLS supports both those applications based on the PATHWAY transaction processing system and those not based on it. The requesters can be located anywhere in the Tandem network. HLS also supports commands from the command and status processor HLSCOM. The HLS process must execute on a Tandem NonStop II or TXP processor.

fields can be either verb requests (requesting the function to be performed) or verb modifiers (modifying the action of the verb). To ensure that HLS functions in a PATHWAY environment, the indicators are always byte values, typically *Y* or *N*, and not bit masks.

Each verb is associated with a reply. The verb reply is a formatted message from HLS to the application program that indicates the result or status of the verb request. Verb replies can also contain indicator fields and optional data. The most important field of the verb reply is the RETURN-CODE field. The RETURN-CODE is a generalized mechanism for standard error and status reporting, much like the condition code or file error in the Tandem operating systems.

Since verbs elicit a two-way message between the requester and HLS, the operating-system call WRITEREAD is used for message delivery. Any Tandem language that supports the equivalent of WRITEREAD can communicate to HLS. Currently, all languages supplied by Tandem support WRITEREAD or its equivalent.

Figure 2

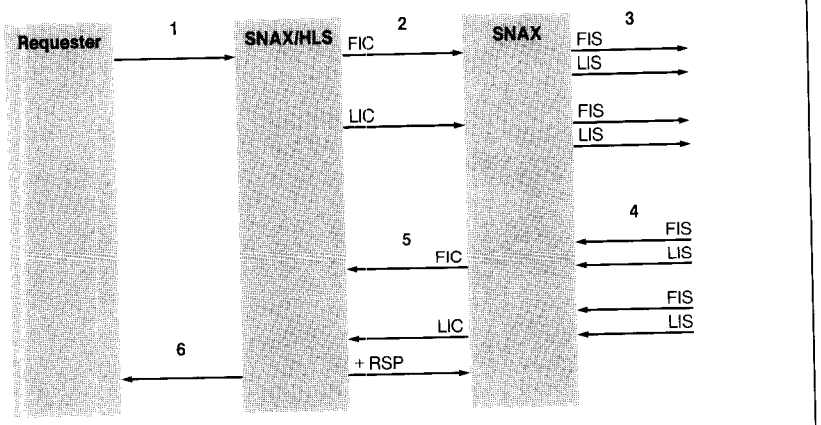


Figure 2.
The processing of a verb that sends a request to the session partner and receives a reply. FIC and LIC are SNA indicators relating to message-chain element groupings. FIS and LIS are SNA indicators relating to message-segment element groupings. +RSP is an SNA acknowledgment. The processing steps are described in detail in the text.

To ensure that HLS would be readily accepted and easily used by applications programmers, considerable attention was paid to the naming conventions for the verb requests and modifiers. SNA communications are replete with arcane constructs such as *brackets*, *chains*, and *request shutdown sequences*, and much of the difficulty in understanding SNA has stemmed from this jargon. Most of these constructs relate directly to important and easily understandable concerns of transaction processing, however: brackets are uninterruptable units of work (i.e., transactions), chains are logical messages, and request shutdown sequences are orderly terminations of communications. HLS, whenever possible, uses the most understandable and meaningful name for each verb modifier. For example:

- When a bracket is in progress, HLS sets the indicator TRANSACTION-IN-PROGESS to *Y*.
- If a complete chain is received from the session partner, HLS sets the indicator MESSAGE-COMPLETE to *Y*.
- Whenever programmers wish to terminate the session in an orderly fashion, they invoke the PREPARE-TO-CLOSE verb.

The Verb Message Sequence

The HLS system was designed to be efficient. A single request from an HLS application can send a transaction to the session partner as well as receive the reply. Figure 2 illustrates this as follows:

1. The requester delivers a "message" (i.e., a verb request with data) to HLS for transmission to the session partner.
2. HLS formats the message into appropriate SNA message units (chain elements), handles required aspects of SNA protocol, and delivers each chain element to SNAX with a SNAX header.
3. SNAX formats each chain element into the appropriate Synchronous Data Link Control (SDLC) frames, handles SDLC addressing, and transmits each frame on the data link.

4. SNAX receives a series of SDLC frames constituting the session partner's reply. SNAX assembles these frames into chain elements and delivers them to HLS.
5. HLS receives chain elements from SNAX, assembles the chain elements into a logical message, and delivers an SNA acknowledgment to the session partner, if required.
6. The HLS requester receives a verb reply consisting of a "message" (i.e., a verb reply with data).

Compatibility with SNA Environments

HLS supports a wide range of SNA communications environments, including the following:

- FM and TS profiles 2, 3, 4, and 7.
- Multiple RU chains managed by HLS or the user.
- Half-duplex flip-flop and contention send/receive mode.
- Full-duplex send/receive mode.
- Immediate and delayed request mode.
- Bracket support under both termination rules.
- Primary-to-secondary and secondary-to-primary pacing.
- MAXRU up to 4096 bytes.
- LU types 0, 1, 2, 3, 4, and 7.
- Pipeline LUs.

HLS has been implemented in strict accordance with the architectural specifications of SNA, as documented in the *SNA Format and Protocol Reference Manual*. Its internal usage checks and state machines are equivalent to those documented in the *Protocol Reference Manual for SNA*, ensuring that HLS state interpretations match exactly those provided by IBM. Figure 3 illustrates the similarity between HLS source code and published SNA documentation.

Figure 3

Finite state machine

SNA

STATE NAMES → / INPUTS	BETC 01	INC 02	PURGE 03
R,RQ,-CANCEL, BC,EC	-	>(R)	>(R)
R,RQ,-CANCEL, BC,-EC	2	>(R)	>(R)
R,RQ,-CANCEL,-BC, EC	>(R)	1	1
R,RQ,-CANCEL,-BC,-EC	>(R)	-	-
R,RQ, CANCEL	>(R)	1	1
S,-RSP,TO_CURRENT_CHAIN	-	3	-
'RESET' /* FROM DFC_RESET */	-	1	1

OUTPUT CODE	FUNCTION
R	RECEIVE_CHECK_SENSE = X'2002' /* CHAINING ERROR */

END FSM_CHAIN_RCV;

SNAX/HLS

INT CHAIN^RCV^ = 'P' : = [

STATE NAMES → / INPUTS	BETC 01	INC 02	PURGE 03
R,RQ,-CANCEL, BC,EC	1	2 + R	3 + R
R,RQ,-CANCEL, BC,-EC	2	2 + R	3 + R
R,RQ,-CANCEL,-BC, EC	1 + R	1	1
R,RQ,-CANCEL,-BC,-EC	1 + R	2	3
R,RQ, CANCEL	1 + R	1	1
S,-RSP,TO_CURRENT_CHAIN	1	3	3
'RESET' /* FROM DFC_RESET */	1	1	1

OUTPUT CODE	FUNCTION
R	RECEIVE_CHECK_SENSE = X'2002' /* CHAINING ERROR */

];

Figure 3.

A finite state machine (FSM) as specified for SNA (documented in the SNA Format and Protocol Reference Manual*) and as implemented in SNAX/HLS. (The punctuation of the HLS code has been modified slightly to fit the figure format.) An FSM is used in SNA to "remember" past events and control the operation of SNA processes. The memory of the FSM is its current state (the current state maps to one of the FSM's columns). When presented with an input (shown on rows) the FSM produces a new current state and output (shown at the intersection of the input row and current state column). The striking similarity between the two FSMs ensures that SNAX/HLS correctly enforces SNA protocol.*

Table 1.
SNAX/HLS verbs.

Verb	Function
Session establishment and termination	
OPEN-SESSION	Establishes communications with the session partner.
PREPARE-TO-CLOSE	Performs an orderly session shutdown.
CLOSE-SESSION	Performs a forced session close.
Data transfer	
SEND-DATA	Sends data to the session partner.
RECEIVE-DATA	Retrieves data from user's receive queue.
SEND-AND-RECEIVE-DATA	Sends data to the session partner and then queues a RECEIVE-DATA verb.
Status request	
REQUEST-SEND-STATE	Requests permission to send data (used only in half-duplex flows).
SEND-STATUS	Sends an SNA LUSTATUS to session partner.
RECEIVE-CONTROL	Queries the top element of the receive queue.
Utility	
CONVERT-ERROR-CODE	Converts hexadecimal fields to display format and decodes SNA sense codes into English messages.
GET-ATTRIBUTES	Obtains profile and BIND information.
SET-ATTRIBUTES	Sets profile options dynamically.

It is important to note that while HLS eliminates SNA communications protocol from the application program, it does *not* eliminate data presentation and formatting requirements. Message formats are the responsibility of the HLS application. In other words, SNA/RJE (LU Type 1) support is possible under HLS, but users must format the data buffer with appropriate SNA Character String (SCS) control codes before transmission. Similarly, word-processing support (LU Type 4) is supported under HLS, but the user is responsible for the format requirements for the device.

System Components

HLS Process

The HLS process is a multithreaded TAL process that acts as an interface between the HLS application and the SNAX SNALU interface. Its primary activity is to perform verb requests and produce verb replies. Typically, verb requests result in one or more SNA messages flowing to the session partner on behalf of the HLS application. HLS also provides a message queue for each application. Incoming messages are buffered by the HLS process if the application does not have an outstanding verb.

HLS verbs do the following:

- Establish communications with a session partner.
- Manage the transmission and reception of data.
- Terminate communications with a session partner.

Table 1 lists the verbs supported by the HLS process.

HLS also supports a "command" interface. Using command-interface verbs, a user process can programmatically control and inspect the HLS operating environment. Command-interface verb replies contain bit masks and are primarily intended for applications written in TAL.

Finally, HLS has been designed as a generic, self-configuring system applicable to a wide range of SNA environments. HLS always examines the session-establishing SNA message termed the BIND. The BIND contains a variety of option fields that detail the specifics of a given logical communications channel between two session partners. If the BIND is within the range of acceptable values to HLS, internal control blocks are customized automatically to fit the specifics of the current session.

Resource Definition Table (RDT)

The Resource Definition Table (RDT) describes the characteristics of the HLS environment. It is a table wherein all session profile information, SNA message formats, and global data are stored. At HLS initialization, the RDT is opened and read into an extended data segment. While this means that every HLS process has the complete RDT in virtual memory, the size of the RDT is relatively small (less than 2K bytes) for most configurations. The RDT in virtual memory can be dynamically replaced without requiring the HLS process to be stopped or affecting active sessions.

When HLS receives an OPEN-SESSION request from the application program, the profile section of the RDT is scanned for the specified profile name. Pointers in the profile point to the location of special SNA messages that are used to establish the session. The application programmer need only know the profile name to specify the details of the SNA session.

The RDT is an unstructured file produced via the "compilation" of an ENSCRIBE file called RDTKSFLE. The maintenance of the RDTKSFLE is performed by a set of programs supplied with HLS that are generated by ENABLE. Thus, the PATHWAY system and the ENABLE program generator are required for the user to use the standard RDT maintenance system.

HLS is delivered with a sample RDTKSFLE, containing several useful profile, BIND, and INITSELF records. The user is free to install the sample RDTKSFLE or create a null RDTKSFLE and add only those records needed for planned applications.

HLSCOM Process

HLS provides a command interface to user processes. A standard command process, HLSCOM, is delivered with the product. Users are also free to design their own command processes using the verbs available with the command interface.

HLSCOM is designed to support a wide range of commands suitable for operator control, error recovery, and system tuning. Optionally, HLSCOM can configure trace files for HLS traces. HLSCOM supports "wild-card" name constructs so that commands can affect:

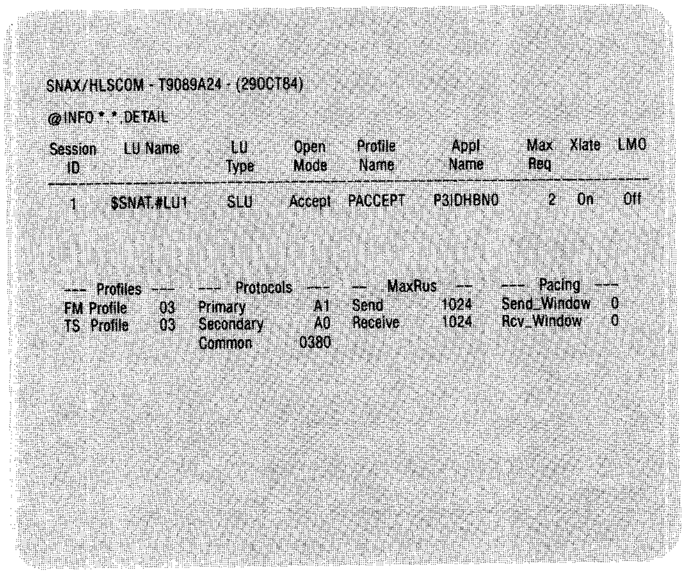
1. A single LU.
2. All LUs on a given SNAX line.
3. All LUs running under an HLS process.

Table 2 lists the HLSCOM commands.

Table 2.
HLS command processor (HLSCOM) commands.

Command	Function
ABEND	Abends current SNAX/HLS process.
ABORT	Aborts one or more sessions.
ASSUME	Sets default line name.
CMDVOL	Sets defaults for file-name expansion in commands.
EXIT	Exits current obey file or program.
FC	Fixes last command line of current input file.
HELP	Displays help information.
INFO	Returns session configuration information.
LISTOPENS	Lists all process openers for specified session(s).
OBEY	Reads commands from specified file.
OBEYVOL	Sets defaults for expansion of obey file names.
OPEN	Specifies name of current HLS process.
PEEK	Shows current SNAX/HLS process information.
QMSG	Broadcasts message to selected HLS applications.
SHOW	Shows current CMDVOL and OBEYVOL settings.
STATUS	Returns session-status information.
SWITCH	Dynamically replaces the current RDT in virtual storage.
TRACE	Configures, starts, and stops session tracing. (Individual LUs can be traced.)

Figure 4



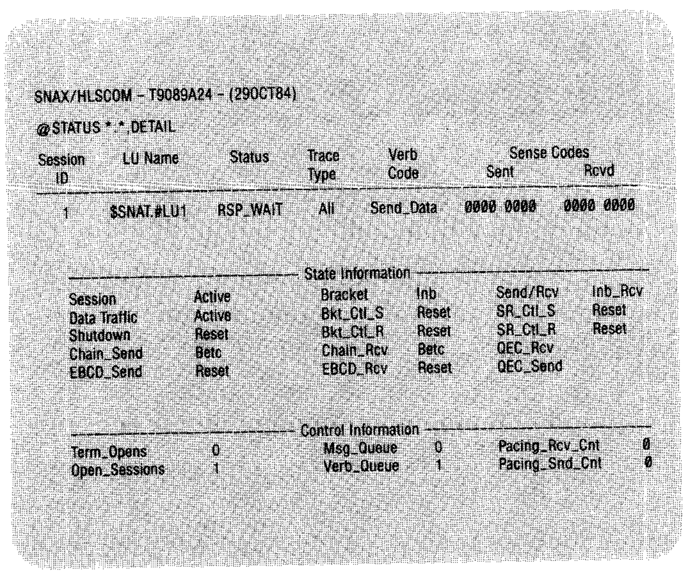
```
SNAX/HLSCOM - T9089A24 - (29OCT84)
@INFO *.*,DETAIL
Session  LU Name      LU    Open    Profile   Appl      Max  Xlate  LMO
  ID                  Type  Mode    Name      Name      Req
--------------------------------------------------------------------------
  1      $SNAT.#LU1   SLU   Accept  PACCEPT   P3IDHBNO    2  On     Off

 --- Profiles ---   --- Protocols ---   -- MaxRus --    --- Pacing ---
 FM Profile   03    Primary      A1     Send     1024   Send_Window   0
 TS Profile   03    Secondary    A0     Receive  1024   Rcv_Window    0
                    Common     0380
```

Figure 4

A sample INFO, DETAIL screen from HLSCOM. The operator requests information for all LUs (.*) with this INFO command, although the information for only one LU is shown in this figure. The session ID, the LU name, and all critical session-profile information is displayed. Note also that all relevant BIND fields are displayed.*

Figure 5



```
SNAX/HLSCOM - T9089A24 - (29OCT84)
@STATUS *.*,DETAIL
Session  LU Name     Status    Trace  Verb        Sense Codes
  ID                           Type   Code        Sent        Rcvd
----------------------------------------------------------------------
  1      $SNAT.#LU1  RSP_WAIT  All    Send_Data   0000 0000   0000 0000

 ------------------------- State Information -------------------------
 Session        Active       Bracket     Inb       Send/Rcv   Inb_Rcv
 Data Traffic   Active       Bkt_Ctl_S   Reset     SR_Ctl_S   Reset
 Shutdown       Reset        Bkt_Ctl_R   Reset     SR_Ctl_R   Reset
 Chain_Send     Betc         Chain_Rcv   Betc      QEC_Rcv
 EBCD_Send      Reset        EBCD_Rcv    Reset     QEC_Send

 ------------------------ Control Information ------------------------
 Term_Opens     0            Msg_Queue   0         Pacing_Rcv_Cnt    0
 Open_Sessions  1            Verb_Queue  1         Pacing_Snd_Cnt    0
```

HLS supports extensive SNA-protocol state interpretation with the DETAIL option of the STATUS and INFO commands. Using DETAIL, users involved in systems programming and technical support can examine each state machine and major control block of an HLS process. Figure 4 illustrates a sample INFO command. All critical configuration information is displayed for the session.

Because on-line inspection of SNA state information is critical to rapid problem isolation and determination, the STATUS command is provided to display the current state of the HLS session. Figure 5 contains a sample of the information displayed by the STATUS command. In this particular sample, the HLS application is sending a message to its session partner (the current verb is SEND-DATA), and the HLS application is blocked, waiting for message acknowledgment from the session partner (the current status is RSP_WAIT). This type of display greatly accelerates problem resolution.

Application Prototyping and Simulation

Using the Application Prototyping and Simulation (APS) system, applications programmers can interactively execute verb requests and immediately see verb replies. Using this feature, programmers can create a prototype of an application in minutes without writing any code. They can easily simulate error recovery, message contention, and verb execution conditions.

Figure 5.

A sample STATUS, DETAIL screen. The operator requests information from all LUs (.*) with this STATUS command, although only one LU is shown here. Note that the current verb for the HLS application is SEND-DATA (shown in the Verb Code field) and completion of the verb is blocked: SNAX/HLS is waiting for message acknowledgment from the session partner (shown in the Status field). The state information section displays the status of all SNA protocol control blocks. The control information shows the value of various counters (e.g., Msg_Queue is the number of messages queued for the application).*

APS is also a useful educational tool. Programmers unfamiliar with HLS can be shown verb dynamics in a few hours of instruction.

APS is a Screen COBOL requester, presenting one screen per verb request. The APS user merely selects the function key representing the verb to be executed, fills in the verb indicators, directs APS to send the verb to an HLS process, and observes the result.

APS includes extensive display fields to help identify errors and problems. It interprets SNA sense codes in both hexadecimal notation and English. The display fields are quite useful for learning the HLS system. Figure 6 represents an APS screen after an OPEN-SESSION failure has occurred. The SNA sense code for the failure (hex 0801) is automatically decoded to message text (Resource Not Available). With this feature, problem resolution can begin immediately; the programmer does not have to decode the sense code by using one of several SNA manuals.

Figure 6

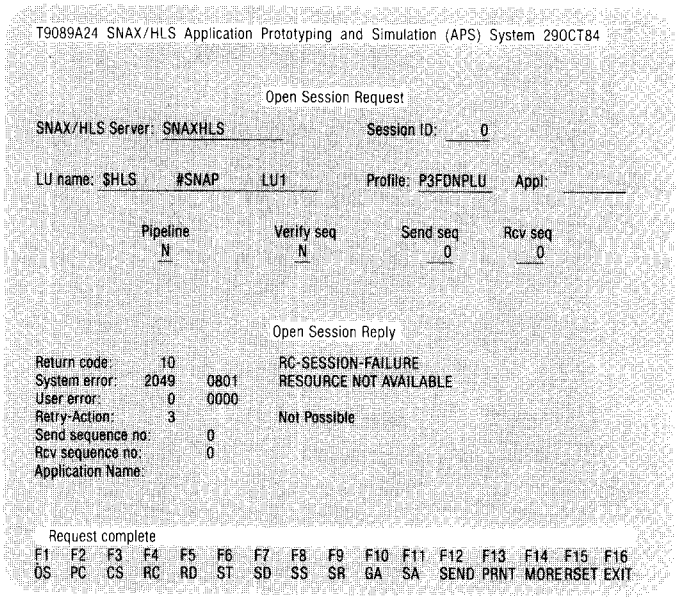


Figure 6.
A screen from an APS session in which a session failure occurred. The top half of the screen shows the OPEN-SESSION verb parameters supplied by the user. The bottom half of the screen shows the verb reply from HLS. Note that the return code is RC-SESSION-FAILURE. The system error field shows the reason for the session failure. A negative response from the session partner was received, indicating that the session partner could not be contacted (RESOURCE NOT AVAILABLE). The retry-action field provides information on whether the OPEN-SESSION verb is retryable; in this instance no verb retry is possible.

HLS Trace Analysis Process

The HLS Trace Analysis Process (HLSTAP) allows the user to format and display trace information in a meaningful, high-level format. First, the user configures trace requirements using HLSCOM. Four levels of trace information can be configured:

1. *Verb input and output*. At this level, a trace of all verb requests into HLS and all verb replies out of HLS can be configured. This trace is intended for use in determining application-program problems.
2. *LU service calls*. At this level, a trace of internal scheduling events that HLS executes on behalf of the user can be configured. This is, in essence, a "dispatcher" trace and is intended for HLS internal-problem determination.
3. *Data flow control (DFC)*. At this level, a trace of "before and after" DFC finite state machines can be configured during the course of message processing. This is intended for HLS internal-problem determination.
4. *SNALU I/O*. At this level, a trace of messages to and from SNAX can be configured.

Figure 7

```
SNAX/HLSTAP - T9089A24 - (29OCT84)

>>>>>> DFC^RESET      ,record 00001,19Dec84,09:24:29.31 <<<<<<

LU Name:   $SNAT.#LU1          Session ID: 1

------------------------------ State Information ------------------------------

Session          Active      Bracket     Inb      Send/Rcv     Inb_Rcv
Data Traffic     Active      Bkt_Ctl_S   Reset    SR_Ctl_S     Reset
Shutdown         Reset       Bkt_Ctl_R   Reset    SR_Ctl_R     Reset
Chain_Send       Betc        Chain_Rcv   Betc     QEC_Rcv
EBCD_Send        Reset       EBCD_Rcv    Reset    QEC_Send

------------------------------ Bind Information -------------------------------

  --- Profiles ---    --- Protocols ---      -- MaxRus --  --- Pacing ---
  FM Profile   03   Primary      A1       Send      1024  Send_Window   0
  TS Profile   03   Secondary    A0       Receive   1024  Rcv_Window    0
                    Common       0380

---------------------------- Session Parameters -------------------------------

DFC_BID_RCV       Allowed      DFC_QEC_RCV       Not allowed  SC_CLEAR   Allowed
DFC_BID_SEND      Not allowed  DFC_QEC_SEND      Not allowed  SC_RQR     Not allowed
DFC_BIS_RCV       Not allowed  DFC_RELQ_RCV      Not allowed  SC_SDT     Allowed
DFC_BIS_SEND      Not allowed  DFC_RELQ_SEND     Not allowed  SC_STSN    Not allowed
DFC_CANCEL_RCV    Allowed      DFC_RSHUTD_RCV    Not allowed  SC_CRV     Not allowed
DFC_CANCEL_SEND   Allowed      DFC_RSHUTD_SEND   Allowed
DFC_CHASE_RCV     Allowed      DFC_SBI_RCV       Not allowed
DFC_CHASE_SEND    Allowed      DFC_SBI_SEND      Not allowed
DFC_LUSTAT_RCV    Not allowed  DFC_SHUTC_RCV     Not allowed
DFC_LUSTAT_SEND   Allowed      DFC_SHUTC_SEND    Allowed
DFC_QC_RCV        Not allowed  DFC_SHUTD_RCV     Allowed
DFC_QC_SEND       Not allowed  DFC_SHUTD_SEND    Not allowed
DFC_RTR_RCV       Not allowed  DFC_SIG_RCV       Allowed
DFC_RTR_SEND      Allowed      DFC_SIG_SEND      Allowed
USING_BRACKETS    Yes          THIS_HALF_SESSION_RQ_MODE      Immediate
FIRST_SPEAKER     Yes          PARTNER_HALF_SESSION_RQ_MODE   Immediate

>>>>>> VERB^REPLY      ,record 00002,19Dec84,09:24:29.85 <<<<<<

LU Name:   $SNAT.#LU1          Session ID:   1

Verb Reply:   Open_Session

---- Indicators -----
Return_Code:          0        (RC-OK)
System_Error:      0000
User_Error:        0000
Retry_Action          0        (NOT APPLICABLE)
Send_Seq_No:          0
Rcv_Seq_No:           0
Application:      P3IDHBN0
```

Figure 7.
Two records from a SNAX/HLS trace file, formatted with HLSTAP. The first record (record 1) shows the initialization of session control blocks for LU $SNAT.#LU1. The State Information section shows the initial settings for SNA control flows. The Bind Information section shows the values of the session-establishing SNA message (i.e., the BIND message). The Session Parameters section shows which types of SNA message SNAX/HLS is allowed to send and receive under the rules of the current session. The second record shows the completion of an OPEN-SESSION verb for the same LU. Note that all verb reply indicators are formatted by HLSTAP.

Once the trace of selected LUs is complete, the user can stop the trace and begin analysis with HLSTAP. HLSTAP allows individual LUs or groups of LUs to be selected for processing. Formatted displays decode all verb indicators, the session BIND, DFC states, SNA request/response headers, and user data. Several convenience commands are also included in HLSTAP. Figure 7 represents a page from a sample HLSTAP report.

Potential Applications

HLS was designed to function in a wide range of environments, including SNA device support, SNA gateway applications, selective passthrough applications, and "intelligent-networking" applications. The following is a description of how HLS fits into each environment.

Device-support applications

In this type of application, an SNA device is connected to a Tandem application (e.g., the PATHWAY transaction processing system). SNAX includes native support for IBM 327X display devices and 328X printers. The HLS system adds support for the following IBM products:

- 3600 Financial System.
- 3624 Financial System.
- 3630 Plant Communications System.
- 3640 Manufacturing System.
- 3650 Retail Store System.
- 3660 Supermarket System.
- 3680 Programmable Store System.
- 3767 Communications Terminal.
- 3770 Data Entry System.
- 4700 Financial System.
- 5520 Information Display System.
- 6670 Information Distributor.
- 8100 Information System.
- Series/1 General Purpose DP System.
- Series/32 General Purpose DP System.
- Series/34 General Purpose DP System.
- Series/36 General Purpose DP System.
- Series/38 General Purpose DP System.

This SNA device support allows the business application to reap the benefits of the Tandem system (i.e., availability, expandability, and programmability) while still making use of the SNA terminal or device.

It is important to note that HLS provides only the SNA support for these devices. Specific device control and data formatting is the responsibility of the HLS application.

Gateway Applications

Gateway applications are processes resident on a Tandem system that are in communication with a foreign computer/network. HLS provides an easy-to-use interface for Tandem applications to communicate with the following SNA applications:

- Customer Information Control System (CICS).
- Host Command Facility (HCF).
- Information Management System (IMS).
- Job Entry System 2 (JES2).
- Job Entry System 3 (JES3).
- Network Communications Control Facility (NCCF).
- Network Routing Facility (NRF).
- Time Sharing Option (TSO).

In gateway-application environments, HLS provides SNA protocol support for applications running on Tandem systems. In such environments, Tandem transaction processes can have on-line access to data located on the IBM system and vice versa. In other words, each system can function as a data-base "server" for the other. Critical data bases can be located on the Tandem system for fast, reliable access while batch-processed data can be located on the IBM system. Thus, neither system need be isolated from the data bases.

Often SNA device-support applications are closely integrated with SNA gateway applications. For example, in a shared automated-teller (ATM) network, Tandem systems could support a network of SNA ATMs using the device-support features of HLS. The routing of ATM transactions would be done by Tandem applications, and transaction delivery to SNA hosts would be supported by the gateway SNA features of HLS.

Gateway sessions to the Network Routing Facility (NRF) deserve special mention. NRF is an IBM software product that runs in an SNA front-end processor (e.g., the 3725). It allows transactions to be routed through an SNA network with no IBM host-application involvement. HLS supports NRF sessions, an example of which is given in the next section.

Selective Transaction-passthrough Applications

SNA gateway support can also involve a version of transaction processing termed *selective passthrough*. In this type of application, the critical data bases are strategically located on Tandem and IBM computers. HLS applications provide the gateway access to the IBM systems. Transaction programs on the Tandem system control the device (Tandem, SNA, or other) and route requests for data to the appropriate source. Terminal operators are unaware of the location of data and/or programs. The demonstration of HLS in October 1984 to the International Tandem Users' Group (ITUG) in Orlando, Florida, featured selective transaction passthrough. In it, terminal operators could retrieve and update records on either a Tandem ENCOMPASS or an IBM CICS data base.

Intelligent-network Applications

Although the term *intelligent network* is ambiguous, there is a consensus that an intelligent network provides, between diverse terminals and hosts, an interface that contains value-added network functions. HLS provides the SNA gateway and SNA device support this type of endeavor requires. For example, one HLS user is supporting custom X.25 terminals with a network of Tandem processors. The user employs HLS to connect the terminals to TSO running in IBM SNA hosts. In essence, the HLS application bridges the gap between the SNA application in the IBM host and foreign X.25 terminals.

SNAX/HLS in an Application Benchmark

SNAX/HLS was used in an application benchmark of a large retail credit-authorization system. The section below describes the HLS design used in the benchmark and the HLS performance results.

The user had a large SNA network in place and requested a demonstration of Tandem's SNA transaction-processing capabilities before undertaking full-scale application development.

Credit transactions from point-of-sale (POS) devices were to be concentrated by IBM Series/1 minicomputers at the store level. Credit-card transactions from the store's POS devices were to be routed through the SNA network to the Tandem systems for credit authorization or rejection.

NRF was selected to route transactions between the Tandem systems and the Series/1 computers. It provides transaction switching with no IBM host-program involvement. NRF also supports transaction pipelining, permitting the transactions from several POS devices to be multiplexed to the Tandem application through a single LU in the Series/1. Figure 8 shows the elements of the planned network.

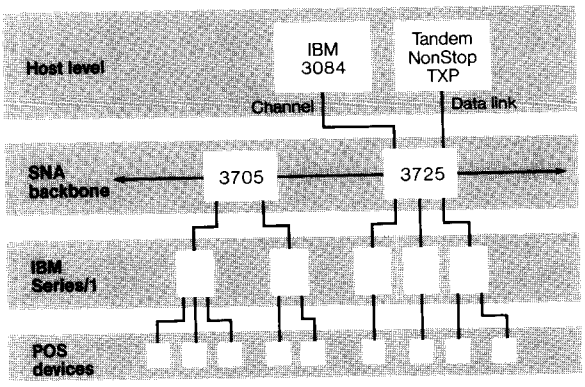


Figure 8.
A schematic diagram of the elements in the retailer's network as simulated in the benchmark. SNA and non-SNA point-of-sale (POS) devices are concentrated by IBM Series/1 minicomputers at retail store locations. Tandem systems running SNAX and HLS connect to the SNA backbone network and perform credit authorizations for customers.

The PATHWAY transaction processing system was selected to control and support credit-authorization processing on the Tandem system. Using HLS, PATHWAY programs were to accept transactions from NRF, perform credit processing, and reply to NRF with approval or rejection.

The detailed design focused on three major aspects of the HLS application: session establishment, data transfer, and session termination. Session establishment and termination were straightforward. The OPEN-SESSION and CLOSE-SESSION verbs were to be used to start and stop SNA communications. The data transfer phase was to use an initial "priming" RECEIVE-DATA verb to acquire the first transaction, followed by a SEND-AND-RECEIVE-DATA verb to reply to the first transaction (the SEND portion of the verb) and prepare for a new transaction (the RECEIVE portion of the verb). The SEND-AND-RECEIVE-DATA verb was to be used in this fashion to minimize the number of interprocess messages between the HLS application and the HLS process.

A summary diagram of the application logic is presented in Figure 9. (In the diagram, the verb structures have been simplified to enhance clarity. More information about HLS verbs can be found in the documentation accompanying the software.)

Figure 9 represents the following dialogue and actions:

1. The application issues the OPEN-SESSION verb, which defines the SNAX LU name to be used for session communication. The PROFILE field of this verb points to an entry in the RDT that specifies the session parameters. The PIPELINE=YES option field indicates to HLS that the LU is to be handled in a pipeline manner.

   The execution of the OPEN-SESSION verb causes HLS to send a special SNA message, INITSELF. The INITSELF message requests NRF to establish a session (i.e., send the SNA BIND message). After the session is established, HLS completes the OPEN-SESSION verb with RETURN-CODE=OK (i.e., the session was successfully established).

Figure 9

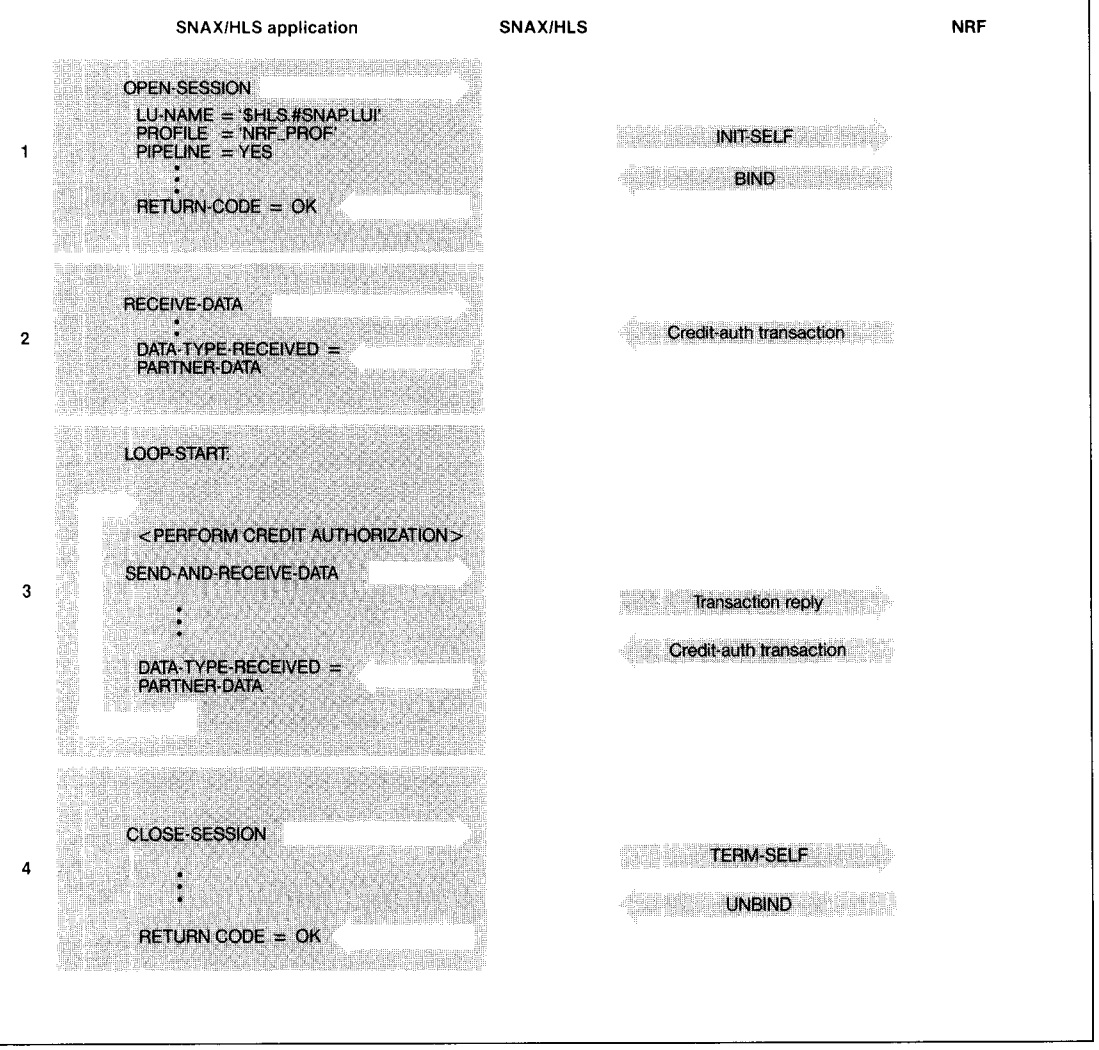


Figure 9.
An overview of the SNAX/HLS processing logic for the benchmark credit-authorization application. Verbs have been simplified for clarity. There are four major steps: (1) the establishment of a session between the SNAX/HLS application and NRF, IBM's Network Routing Facility, (2) an initial "priming" RECEIVE-DATA to obtain the first transaction, (3) an iterative loop of credit processing and execution of a send/receive verb, and (4) termination of the session with NRF. Verb execution is described in detail in the text.

2. The application issues a RECEIVE-DATA verb to accept the first transaction. HLS then accepts a transaction from NRF, delivers SNA acknowledgment if necessary, and completes the RECEIVE-DATA verb with DATA-TYPE-RECEIVED= PARTNER-DATA to indicate the source of the message.
3. The application now enters its main processing loop, continuing the processing until the session is ended or the application is terminated.

   First, the transaction is subjected to credit processing. Next the application issues a SEND-AND-RECEIVE-DATA verb. The SEND portion of the verb sends the transaction reply to NRF for transmission to the requesting terminal. The RECEIVE portion of the verb accepts the next transaction for processing.
4. When the HLS application wishes to terminate the session, it issues the CLOSE-SESSION verb. This verb causes HLS to send the SNA message TERMSELF to NRF. The session is ended when the SNA UNBIND message from NRF is received by HLS.

The credit authorization system described above was constructed by CICS application programmers using Tandem program-development tools. The system was subjected to extensive tests by the Teleprocessing Network Simulator (TPNS),

an IBM software product. TPNS testing showed that SNA sessions could be established between IBM processors and Tandem systems and that the HLS application correctly processed SNA pipeline transactions.

The system was subjected to a significant performance benchmark. Along with the transaction-throughput and response-time measurements, some SNAX/HLS measurements were taken. Based on measurements made with the XRAY performance-measurement tool, the cost of a SEND-AND-RECEIVE-DATA verb in a pipeline environment was calculated at 11 ms per transaction on the NonStop TXP processor.[2] This figure can be interpreted in the following ways:

- A NonStop TXP processor is capable of processing almost 100 SEND-AND-RECEIVE-DATA verbs per second in the environment measured.
- HLS processing overhead is very small. The measurement above was obtained from a system of 32 NonStop TXP processors, at 120 transactions per second. At the measured rate, only 4.1 percent of the total processing power of the system was consumed by HLS verb processing.

Conclusions

SNAX/HLS brings the power of Tandem on-line transaction processing to SNA networks. Tandem users can use the features of SNAX/HLS to support advanced SNA terminals and/or communicate to IBM SNA subsystems. Critical data bases and applications can now be positioned where business needs dictate.

SNAX/HLS support tools such as the APS system bring application programmers up to speed rapidly, allowing applications to be developed in a minimum amount of time. In a production environment, tools such as HLSCOM and HLSTAP assist in rapid problem isolation, determination, and resolution.

With SNAX/HLS, Tandem solutions to critical business problems can be more easily designed, developed, and maintained in an SNA environment.

References[3]

Kirk, D. 1984. A SNAX Passthrough Tutorial. *Tandem Journal*. vol. 2, no. 2. Tandem Computers Incorporated.

SNAX Programming and Network Management. Part no. 82326 A00. Tandem Computers Incorporated.

Systems Network Architecture — Sessions between Logical Units. GC20-1868. IBM Corporation.

Systems Network Architecture Concepts and Products. GC30-3072. IBM Corporation.

Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic. SC30-3112. IBM Corporation.

Watson, L. and Madsen, K. 1984. *Capacity Planning for Tandem Computer Systems*. Tandem Application Monograph Series. Part no. 83904. Tandem Computers Incorporated.

Acknowledgments

The author would like to thank Allen Maxwell and Dick Kline for their painstaking reviews and constructive comments on this article.

[2]The CPU millisecond figure was obtained by summing the CPU BUSY figures for all HLS processes, dividing the sum by 100 to yield total processing time, and, finally, dividing by the transaction rate per second.

[3]In addition to the references listed, several Tandem software manuals (or sections of existing manuals) on SNAX/HLS will be published. Refer to these when they become available.

Steven E. Saltwick is an advisory analyst on the South Central Regional Staff in Dallas, Texas. Since joining Tandem in 1982, Steve has supported major accounts in the Dallas District and the South Central Region. He was a member of the group responsible for the design and implementation of SNAX/HLS. He is currently a member of Tandem's Intelligent Network Task Force. Before joining Tandem, he was a systems engineer and SNA developer for another computer vendor.

The DYNAMITE Workstation: An Overview

Designed as an integrated part of the Tandem system, the DYNAMITE 654X workstation combines the local processing capability of a personal computer and the capabilities of the Tandem 653X terminal. This allows Tandem users to use their Tandem system, transfer information to and from the system and their workstations, and use the workstations locally, taking advantage of the business software currently available for personal computers.

Standard Software and Hardware Components

The workstation comes in two models: the 6541 workstation, which includes two diskette drives, and the 6546 workstation, which includes one diskette drive and one 10M-byte Winchester (hard) disc. Both models include all of the following standard software and hardware components:

- MS-DOS operating system (Microsoft).
- GW-BASIC language.
- 653X terminal-emulation software.
- 256K-byte memory.
- 8086 processor (8 MHz).
- 26 function keys (IBM PC and Tandem 653X).
- 12-inch monitor.
- Low-profile keyboard.
- Tilt-and-swivel terminal base.
- Serial printer interface.
- RS-232-C, RS-422, or Tandem current-loop communications ports.
- Audio speaker.
- Power-up diagnostics.

Options

Several product options are also available to complement the workstation. First, a new, low-cost printer (available in two models) can be used either as a screen printer or as an output device for local applications. Both printer models have two switch-selectable print formats: normal 9 × 9 format or the near letter-quality 18 × 17 dot-matrix format. The models are:

- *Model 5540*, a serial-matrix printer that prints 80 columns at 158 cps, when the 9 × 9 matrix is selected (94 cps, when the 18 × 17 format is selected).
- *Model 5541*, a serial-matrix printer that prints 132 columns at 158 cps, when the 9 × 9 matrix is selected (94 cps, when the 18 × 17 format is selected).

A second option is the bit-mapped graphics board that runs third-party, IBM-compatible graphics, offering high resolution and a mouse interface.

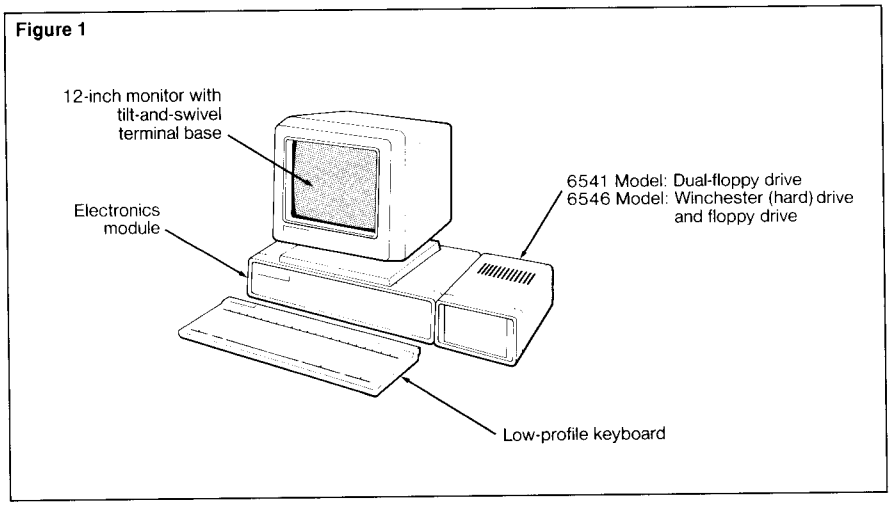
Finally, two system-integration software packages, the Information Xchange Facility (IXF) and PCFORMAT, are also available. IXF software transfers information between system and local files. PCFORMAT converts Tandem system files to one of several formats used by third-party PC software.

Features

The DYNAMITE workstation enhances the on-line information capabilities of Tandem systems. Its benefits for the Tandem user include:

- *Economy*. The workstation provides the user with several information tools in one: (1) An on-line 653X transaction terminal, (2) a personal computer, (3) a 3270 terminal, when the optional EM3270 software is used, (4) a graphics terminal, when the graphics option is used, and (5) a word-processing terminal, when third-party software is used.
- *IBM PC compatibility*. Most IBM PC applications can be run on the DYNAMITE workstation.
- *Modular, ergonomic design*. The electronics module can be placed off the work space to save space, and the terminal can be swiveled and tilted.
- *High-resolution graphics*. The DYNAMITE workstation provides 800 x 300 resolution (in pixels).
- *Display of both alphanumerics and graphics* on a standard video screen. Unlike some personal computers and workstations, it does not require a separate graphics terminal.
- *Third-party software* that performs significantly faster than other third-party emulation software (up to ten times faster for some operations). Also, the DYNAMITE workstation third-party software is designed to work with future, as well as existing, Tandem software.

Figure 1.
The hardware components of the DYNAMITE workstation. The electronics module can be placed away from the rest of the components to save desk space.



Hardware Components

The DYNAMITE 6546 workstation is shown in Figure 1. Both Model 6546 and Model 6541 have a 12-inch monitor, electronics module, keyboard, and disc module. The difference is in their disc drives: the 6541 disc module contains two diskette drives, each having a capacity of 360K bytes, while the 6546 disc module contains one diskette drive and one 10M-byte Winchester disc.

Figure 2 illustrates the basic components and three option-board slots of the electronics modules for the two models. For both models, the upper slot of the electronics module contains the controller board, including the following:

- Processor.
- Monitor interface (display controller).
- Keyboard interface.
- 128K bytes of memory.
- Communications interface.
- Serial printer interface.
- Bus interface to the additional options.

The middle slot contains a disc controller board (either the dual-floppy controller for the Model 6541 or the floppy/hard disc controller for the Model 6546). This board controls disc drives and contains another 128K bytes of memory. If the user needs more than 256K bytes of memory, additional

memory can be added to this board in blocks of 128K bytes. There is room for three more blocks of memory (384K bytes), for a maximum workstation memory size of 640K bytes.

The bottom slot is available for option boards. Currently, the only option boards available are the multifunction board and the graphics board.[1] The multifunction board contains:

1. An IBM parallel printer port used with IBM-compatible printers.
2. An IBM serial port used with third-party communication packages.
3. A real-time clock (battery-backed).

The graphics option board contains all the capabilities of the multifunction board in addition to graphics.

Note that options currently used with the 653X terminal family (i.e., the alternate-input and voice options) are not compatible with the DYNAMITE workstation. The main reason for this is that the architecture of the 6530 is based on the Z80 chip while that of the DYNAMITE workstation is based on the 8086.

Compatibility with the IBM PC

The DYNAMITE workstation is compatible with applications written to run on MS-DOS or IBM's PC-DOS operating systems. Applications written within the constraints of the MS-DOS or PC-DOS system are hardware-independent and compatible. Most popular third-party applications follow these constraints and, thus, run on the DYNAMITE workstation. If software has been written to address specific hardware attributes of the PC, it may not be compatible with the workstation.

Figure 2

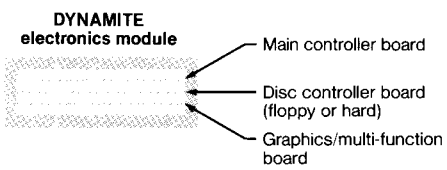


Figure 2.
Board slots in the electronics module of the DYNAMITE workstation. The main controller board contains the processor, monitor interface, keyboard interface, 128K bytes of memory, communications interface, serial printer interface, and bus interface to the options. The disc controller board contains the controller for the floppy diskette or hard disc drives and another 128K bytes of memory. The bottom slot is for option boards. Currently the multifunction and graphics boards are available.

Levels of Compatibility

The article, "How Compatible is Compatible?" (Cook, 1983), describes seven levels of compatibility for the IBM PC. Below is a brief description of these levels and information about the compatibility of the DYNAMITE workstation at each level. (Note that as the level number increases, the degree of compatibility also increases.)

Level 1: Media compatibility. This is the ability to read and write discs in the format used by the IBM PC. A disc formatted on the PC can be read and written to on the DYNAMITE workstation and vice versa.

Level 2: Processor compatibility. This is instruction-set compatibility (8088, 8086, 80188, and 80186 chips). Although the instruction set of the DYNAMITE workstation is compatible with that of the PC, the processor clocks of the PC and the workstation differ. The PC processor has a 4.77 MHz clock while the workstation has an 8.00 MHz clock. This means that programs that implement processor-based timing loops on the PC may not work on the DYNAMITE workstation. If, however, the timing loop is based on the counter timer (the same for both systems), the program should work.

[1]The A10 (or later) release of the firmware and software is required for both option boards.

Table 1.
Compatibility of the DYNAMITE workstation with the IBM PC.

DYNAMITE workstation component	Compatible with IBM PC?
Processor instruction set	Yes
Processor clock	No
Operating system	Yes
Option boards	No
Character set and keyboard	Yes (plug-compatible)
Video display, superset	Yes
System architecture	Yes (partially)

Level 3: Operating-system compatibility. This means compatibility with MS-DOS or other popular PC operating systems. The DYNAMITE workstation is compatible with MS-DOS. Tandem has licensed MS-DOS (version 2.11), which has external commands equivalent to those of the IBM PC, compatible function calls, identical file-protection schemes, and equivalent run-time libraries.

Level 4: Option-board compatibility. This level refers to the use of PC option boards. PC option boards cannot be used in the DYNAMITE workstation. Hardware options for DYNAMITE workstation electronics modules are designed and provided by Tandem. Tandem does, however, provide some functionally equivalent options (e.g., the graphics board, which is compatible with the IBM optional graphics board). The workstation graphics board can be run in monochrome or color mode. Third-party software packages that directly access monochrome or color display hardware should work on the DYNAMITE workstation.

Level 5: Character-set and keyboard compatibility. This level of compatibility requires that the product use the same 256 display codes and the same keys used by the PC. The DYNAMITE workstation has a compatible keyboard; in addition, it has other function keys for 653X terminal emulation.

Level 6: Video compatibility. This level of compatibility requires that the video interface used by the PC be used, including memory mapping and controller addresses. The DYNAMITE workstation's video interface is compatible with that of the IBM PC. The character-display controller on the processor board is compatible with the IBM monochrome alphanumeric graphics, and it has color capability for alphanumeric and bit-mapped color graphics.

Level 7: System compatibility. This means duplication of the entire PC architecture, including random-access memory (RAM), read-only memory (ROM), I/O addresses, and the PC Basic Input/Output System (BIOS). The system compatibilities and incompatibilities of the DYNAMITE workstation with the IBM PC are listed in Table 1.

The DYNAMITE workstation is partially compatible with the PC at this level. It uses the same interrupt system, direct-memory-access (DMA) system, and timer counter. In addition, the key I/O addresses are the same as those for the PC (including the addresses for the keyboard, display, and diskette). Finally, the memory of the DYNAMITE workstation has the same layout as that of the PC.

The portions of the DYNAMITE workstation system architecture that differ from the PC include, for example, the system bus, the processor clock, and the soft-configuration menu stored in nonvolatile RAM (the PC uses dip switches).

Input/Output Compatibility

A brief discussion of the IBM PC software called BIOS would be helpful before examining compatibility further. BIOS is contained in ROM. Its primary function is to handle low-level aspects of I/O (such as interrupts) to the display controller, floppy, hard disc, keyboard, and printer on behalf of an application. To state this in a different way, an application can make calls to BIOS to perform I/O, and BIOS handles all direct communications to the device.

Using BIOS, the application does not have to know the physical characteristics of a device. It simply passes the data to BIOS, which knows the device's physical characteristics and how to communicate with it. As long as the application's calls to BIOS remain consistent, the application is able to access the device, regardless of changes in the hardware interface.

As a general rule, if a workstation's BIOS is compatible with PC BIOS, a third-party application accessing the hardware through BIOS should run successfully on both units. Note that, for an application to run successfully, the interface between the application and BIOS must be the same, while the interface between BIOS and the device can be different. For example, while the PC printer interface is parallel, the printer interface for the DYNAMITE workstation is serial. As long as a third-party package calls BIOS to use the printer, however, the printer interface works.

The Tandem serial printer port, therefore, is equivalent to the IBM parallel port when BIOS is used. Third-party configuration instructions for serial printers should only be followed when a serial printer is attached to the IBM PC COM1 or COM2 serial ports.

A PC application can, however, bypass the BIOS and access the hardware directly. The DYNAMITE workstation was designed so that its key hardware elements (the display controller, diskette, and keyboard) are compatible with those of the IBM PC. Applications that access these components directly should run successfully on the DYNAMITE workstation.[2]

Table 2 lists the software products that have been tested on the DYNAMITE workstation and have run successfully. This list is by no means exhaustive. More applications are scheduled for testing. (Note that the list is not a commitment from Tandem to warrant or support the software.)

Table 2.
Third-party software products that have been tested successfully on the DYNAMITE workstation.

Application	Vendor
Wordstar	MicroPro
ThinkTank	Living Videotext
SuperCalc 2	Sorcim
VisiCalc IV	Software Arts
Lotus 1-2-3	Lotus Development
MultiPlan	Microsoft
Symphony	Lotus Development
R:BASE 4000	Microrim
dBASE II	Ashton-Tate
dBASE III	Ashton-Tate
EasyWriter (1.1)	Info Unlimited
VisiWord	VisiCorp
WORD	Microsoft
Displaywriter 2	IBM
Multimate	Multimate Intl.
Framework	Ashton-Tate
Managing Business with Lotus 1-2-3	Lotus Development
EDIX Text Processor	Emerging Technology
Harvard Project Manager	Harvard Software
IBM Personal Editor	IBM Corporation
Microsoft Macro Assembler (1.25)	Microsoft
Norton Utilities	Peter Norton
PC Tutor (for MS-DOS 2.0)	Comprehensive SW
PFS:FILE	Software Publishing
PFS:WRITE	Software Publishing
Prokey	Rosesoft
Sidekick	Borland SW
PC Master	Courseware
Dow Jones Reporter	Dow Jones Software
Verbatim Disk Analyzer	Verbatim
Copy II PC	Central Point SW
VEDIT	CompuView
CPM/86 Operating System	Digital Research
Concurrent CPM/86	Digital Research
Turbo Pascal (2.0)	Borland SW
Lattice C Compiler (2.0)	Lattice Corp.

DYNAMITE Workstations in the Tandem Environment

While the DYNAMITE workstation provides local processing and is compatible with the IBM PC, it is much more than a personal computer. The basic product includes all the hardware and software necessary to communicate with a Tandem system.

Included in the basic product is Tandem 653X terminal emulation for both conversational mode and block mode, in an asynchronous, TERMPROCESS environment. The workstation can be attached to any port already configured for an asynchronous 653X terminal, and it can be used in any manner appropriate for that terminal. (More detailed information about DYNAMITE host integration is given in the accompanying article by Stan Kosinski.)

EM3270

Communication with an IBM 3270 application via a DYNAMITE workstation can be accomplished using EM3270 software (not included in the basic product). This requires the 6530 emulation software (a standard software component).

[2]Support for an IBM hardware-compatible serial communications port and parallel printer port is planned for the DYNAMITE workstation multifunction and graphics board.

Information Xchange Facility

The Information Xchange Facility (IXF), which is used for file transfer, is also available at an additional cost. With it, both ASCII and binary files can be transferred between the Tandem system and DYNAMITE workstations. File transfer is useful for data integrity (copying the workstation data to a mirrored volume for safety) and for security (copying a file to the host and purging it from the floppy).

IXF is easy to use. To transfer files one follows these steps:

1. Boot MS-DOS.
2. Start 653X emulation.
3. Log on to COMINT.
4. Return to MS-DOS.
5. Initiate file transfer with the MS-DOS external command.

Note that, to accomplish the file transfer, software executes on both the DYNAMITE workstation and the host system. (See Figure 3.)

IXF provides:

- Powerful line-oriented commands to direct operations.
- Command origination from the command line or from a file.
- Sophisticated pattern matching for file names on the DYNAMITE workstation and the host.
- Multiple file transfers with one command.
- A full set of options for customized tailoring.
- A sophisticated communications protocol for data integrity.
- Process-to-process communications between the host and workstation applications.
- A protocol that is transparent to applications.
- Translation of nongraphic data.

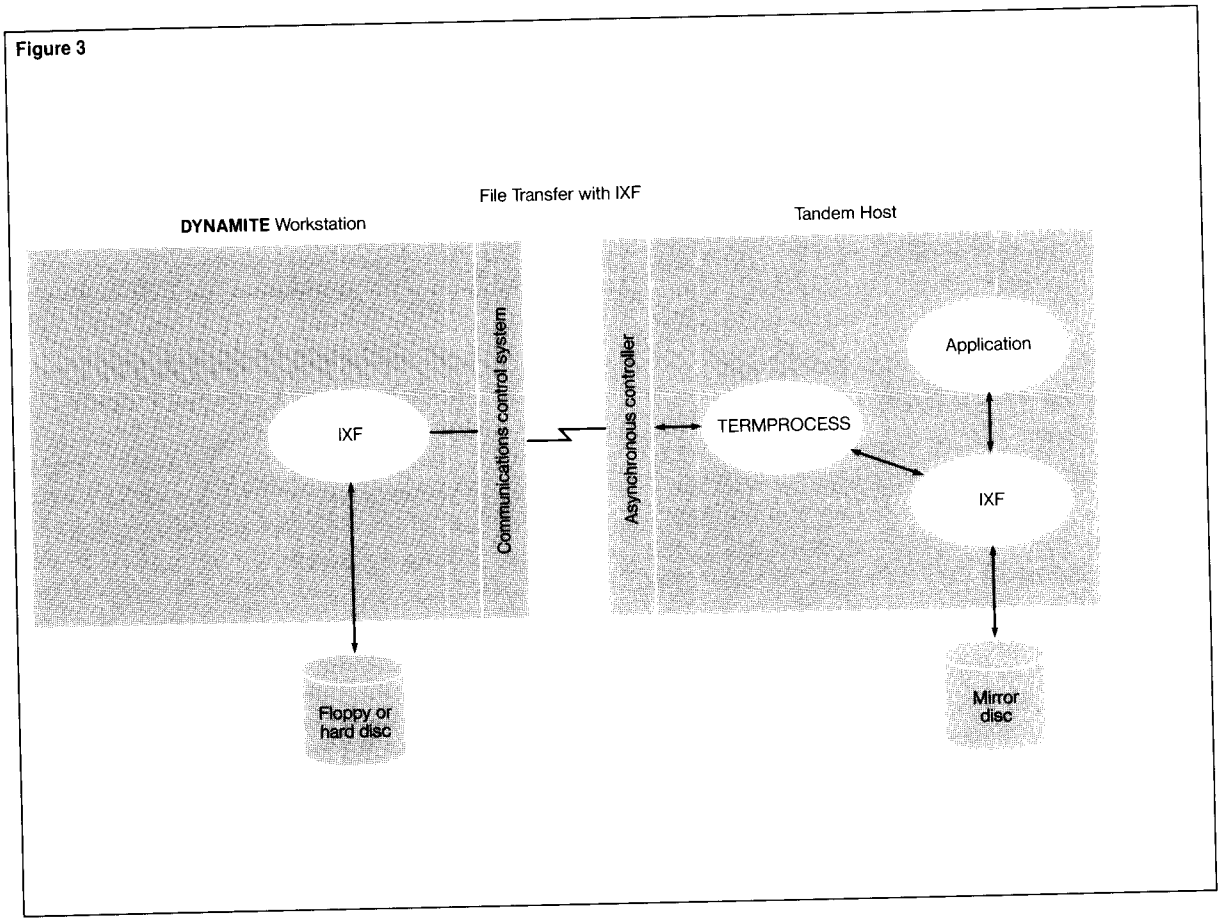


Figure 3.
File transfer between a Tandem system and a DYNAMITE workstation with the Information Xchange Facility (IXF). (The communications control system is to a set of software components on the workstation that provide a high-level, software interface to the data-communications hardware.)

PCFORMAT

Finally, an even greater degree of integration can be achieved between the DYNAMITE workstation and the Tandem host with PCFORMAT, a data-extraction utility. PCFORMAT runs on any Tandem host, accepts data from any Tandem file, and converts the data into any of these IBM PC formats: DIF (Data Interchange Format), SYLK (Symbolic Link), BASIC, and ASCII.

Once converted, the file can be downloaded to the workstation with IXF. The downloaded file can then be accessed by any PC program that recognizes the format. This process is illustrated in Figure 4.

Input to PCFORMAT is typically a file described by the Tandem Data Definition Language (DDL). The input file can be unstructured, entry-sequenced, relative, or key-sequenced.

End-user Support

Tandem provides end-user support for the DYNAMITE workstation via its Customer Assistance Center (CAC) in Austin, Texas, and the Customer Focal Point (CFP) employee at the customer site. Figure 5 shows the relationship between the CAC, CFP, and end user.

Customer Assistance Center (CAC)

The CAC is a source of information, expertise, and assistance to workstation users for problems for which the users can find no solution in the product manuals and educational materials.

Assistance provided by the CAC includes:

- Hardware and software trouble-shooting.
- Problem isolation and resolution.
- Operational assistance.
- Information about product use.

The CAC maintains a data base of known problems, customer calls, commonly asked questions, and third-party software that has been tested and run successfully on the DYNAMITE workstation. With its expert staff, data base, testing program, and support facilities, its goal is to ensure that users of the DYNAMITE workstation are fully supported.

Figure 4

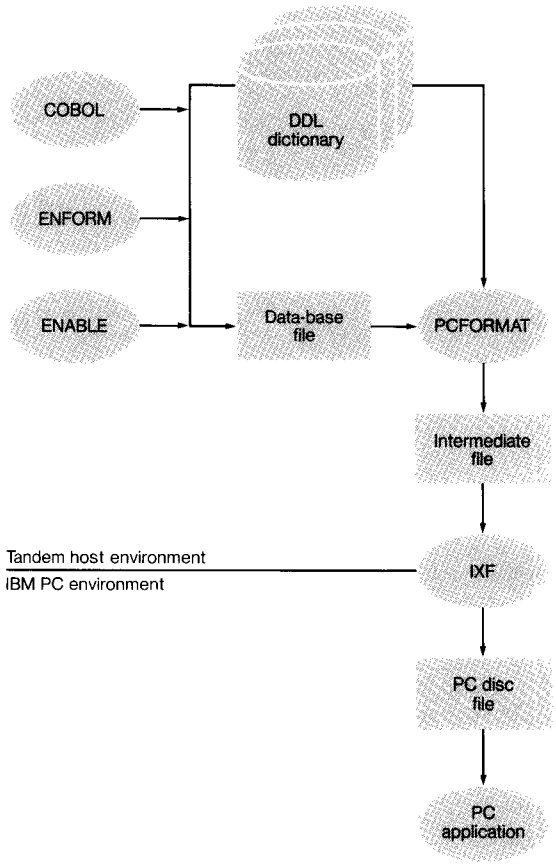


Figure 4.

Data extraction with PCFORMAT. PCFORMAT runs on any data from any Tandem file, and converts the data into a format usable by IBM PC programs. File types can be unstructured, entry-sequenced, relative, and key-sequenced. Formats for PC disc files include DIF, SYLK, BASIC, and ASCII.

Customer Focal Point (CFP) Employee

Instead of individual end users calling the CAC directly, customers choose at least one end-user employee to be Tandem's CFP. The CFP is the liaison between Tandem and the customer for all support needs for the DYNAMITE workstation.

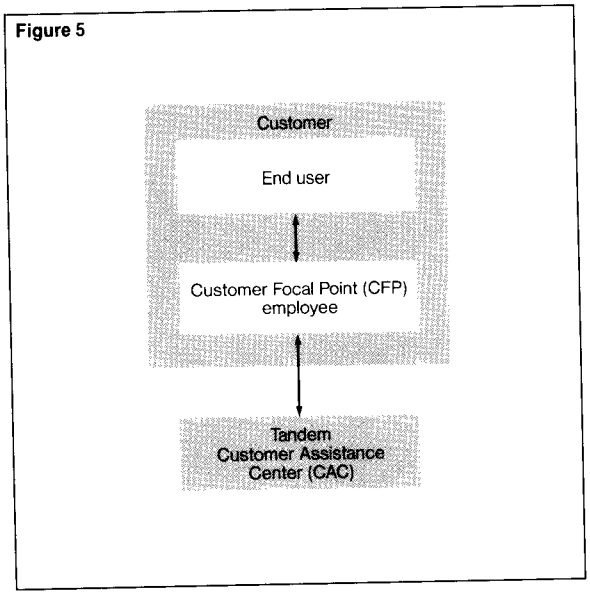


Figure 5.
The relationship between Tandem's Customer Assistance Center (CAC), Customer Focal Point (CFP) employee, and end users of the DYNAMITE workstation. The CFP is the liaison between the end user and the Tandem CAC. Large customer sites may have more than one CFP.

When an end user comes to the CFP with a workstation question or problem that does not require the expertise of the CAC, the CFP is responsible for answering the question or solving the problem. When expert help is required, the CFP contacts the CAC via a toll-free 800 telephone number, and the CAC provides the assistance needed. The CFP then communicates the information or solution to the end user.

Software Supported by the CAC

Note that, while Tandem CAC support analysts have a working knowledge of the popular third-party software that runs successfully on the workstation, they do not support this software. Their function is to support the hardware and software supplied with the DYNAMITE 654X workstation. If they find that a user's problem results from a problem in third-party software, they refer the CFP to the appropriate vendor.

Conclusion

The DYNAMITE 654X workstation is several information tools in one. It provides Tandem users with a single-vendor solution for connecting microcomputers with a mainframe system. It has many features, including high-resolution graphics; high performance; fast access; compatibility with the IBM PC; a modular, ergonomic design; and popular third-party software.

The DYNAMITE workstation can:

- Emulate the Tandem 653X terminal.
- Execute PCFORMAT to convert data from Tandem host format to IBM PC format.
- Transfer files between a Tandem system and the local environment with IXF.
- Communicate with IBM 3270 applications with EM3270.
- Run third-party business software for personal computers.

All of these capabilities make it an excellent business tool for users of Tandem systems.

Tandem has a continuing commitment to integrate workstations into its systems. Future products and enhancements are planned to further automate the sharing of host and workstation information.

References

Cook, S. 1983. How Compatible is Compatible? *PC World*. vol. 1, no. 1.

654X Workstation—PCFORMAT User's Guide. Part no. 82679 A00. Tandem Computers Incorporated.

Ginny Smith is a senior systems analyst in the Data Communications group of Software Education, where she develops courses for customers and systems analysts. Before this, she specialized in Tandem terminal products in the Large Systems Support group. During that time, one of her contributions was the support strategy for the DYNAMITE workstation. She joined Tandem in October 1980 as an instructor and course developer for Tandem Application Language (TAL) and GUARDIAN operating system courses. Before joining Tandem, Ginny was a systems engineer for another mainframe vendor.

An Introduction to DYNAMITE Workstation Host Integration

Tremendous advances in microelectronics during the 1970s have come to fruition in the '80s. Digital watches, video games, and even automobiles with synthesized voice warnings are now commonplace. Perhaps the most exciting development has been that of the personal computer. Its rapid proliferation, both at home and in the workplace, has brought an entire society into the computer age almost overnight. This revolution prompted *Time* magazine to select the computer recently as its first nonhuman "Man of the Year."

Few institutions have remained untouched by the PC revolution, least of all the information industry. Large, centralized machines are giving way to smaller computers, located in individual offices and homes. We are witnessing an Industrial Revolution in reverse.

The personal computer has allowed end users to design and custom-tailor applications to meet their unique needs. Such machines increase productivity because they need not be shared with other users. However, the speed with which personal computers have proliferated has created some problems as well.

In business, government, and education, personal computers are largely isolated, cut off from important applications and information resources still resident on mainframes. Many data bases cannot be distributed. (They may be too large to install on every personal computer that needs access. Also, the nature of the application may make distribution difficult, if not impossible, because of data-base consistency problems.)

PC users have difficulty exchanging information with one another, and they long for the convenience of mainframe-based electronic mail systems. They also need access to expensive peripherals, such as laser printers, which cannot practically be attached to every personal computer. Finally, personal computers cannot provide the peace of mind that comes with fault-tolerant file storage.

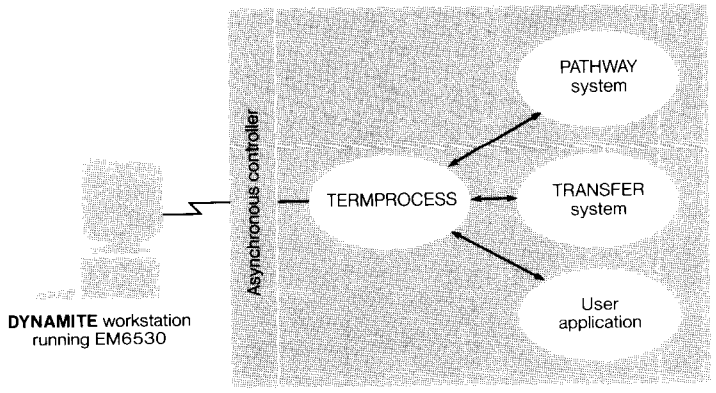
The Tandem DYNAMITE 654X workstation was designed from the beginning to provide users not only with substantial local computing power, but also with full access to existing Tandem mainframes. This flexibility allows a business to take advantage of the possibilities presented by personal computing without abandoning the substantial investment already made in host applications, data bases, and hardware.

The EM6530 Terminal Emulator

The first important element in the DYNAMITE host-integration package is the 6530 terminal emulator, EM6530. As shown in Figure 1, EM6530 is a program residing on the DYNAMITE workstation that can transform the workstation into a 6530 terminal. This emulation capability allows the workstation to access all facilities associated with the local Tandem mainframe, including the ENCOMPASS data-base system, TRANSFER delivery system, and the EXPAND network to which the mainframe is attached. Data residing in the host network may, of course, be accessed just as easily by other DYNAMITE workstations equipped with the EM6530 program.

Figure 1.
Host access through the EM6530 terminal emulator.

Figure 1



Data-base Format Conversion

The EM6530 terminal emulator is very useful because it provides access to a Tandem host. Its ability to move information from the host environment to the workstation environment (and vice versa) is very limited, however. Information to be sent to the host must be entered manually while the workstation is operating as a terminal. Information from the host may be displayed, but it cannot be stored permanently at the workstation (for access and manipulation later, when the workstation is operating in a stand-alone mode). These limitations not only prevent information coming from the mainframe from being processed locally (e.g., by a spreadsheet program), but they also prevent the output of a workstation program from being entered into the host environment so that it can be accessed by other workstation users.

Thus, the next element in our DYNAMITE host integration package is a facility that allows the host to transfer information in bulk to the workstation so that the workstation can process that information locally. Such a transfer is not as easy to effect as one might think. One big problem is the fact that the format of data-base files on a Tandem mainframe is quite different from that of the files used by third-party programs that run on the DYNAMITE workstation.

The Tandem PCFORMAT program solves the data-base-format transformation problem. PCFORMAT is a host-resident program that can convert Tandem data-base files into files that are structured so as to be usable with common third-party PC programs. Various conversion formats can be specified, including:

- *ASCII.* This format, typically used by word-processing programs, consists of text lines separated by carriage return/linefeed couplets.
- *BASIC.* This format is compatible with the INPUT statement in BASIC. It may also be used by such third-party programs as Lotus 1-2-3 and dBASE II.

- *DIF (Data Interchange Format).* This format is used by third-party programs such as Lotus 1-2-3 and VisiCalc.
- *SYLK (Symbolic Link).* This format is used by certain Microsoft programs.

The use of the PCFORMAT program is illustrated in Figure 2. The terminal emulator program, EM6530, can be used first to access the host. Then, the PCFORMAT program is invoked. It uses the DDL data dictionary to establish the proper format for the records, thus eliminating the need for a separate user specification. The output of the PCFORMAT program is an unstructured host file in one of the above-mentioned formats.

If there is a need to convert a restricted set of data-base records (for example, those for all the married employees in an employee data base), the ENFORM program can be used to extract them from the data base before the PCFORMAT program is run.

Figure 2

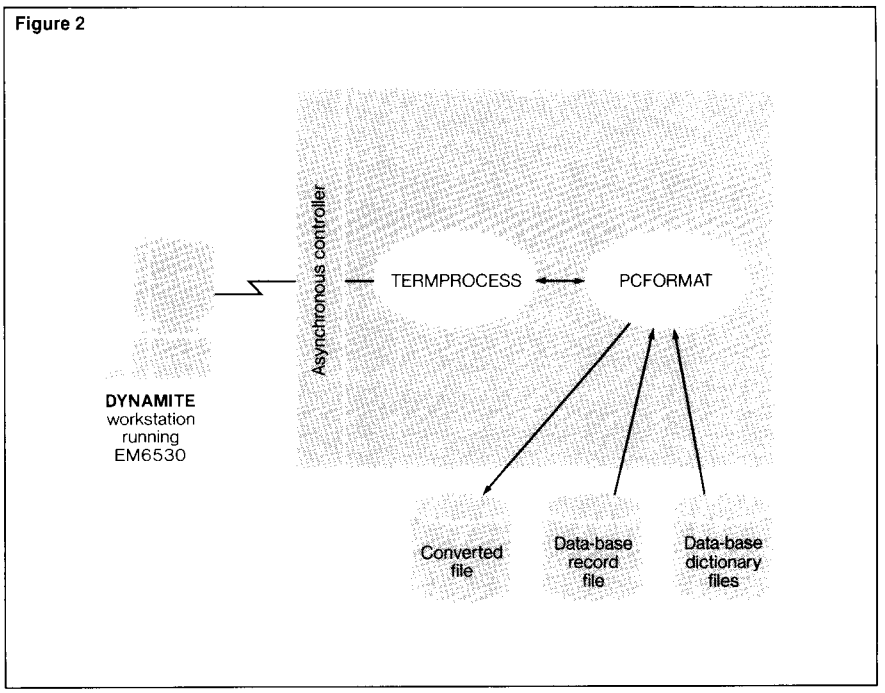


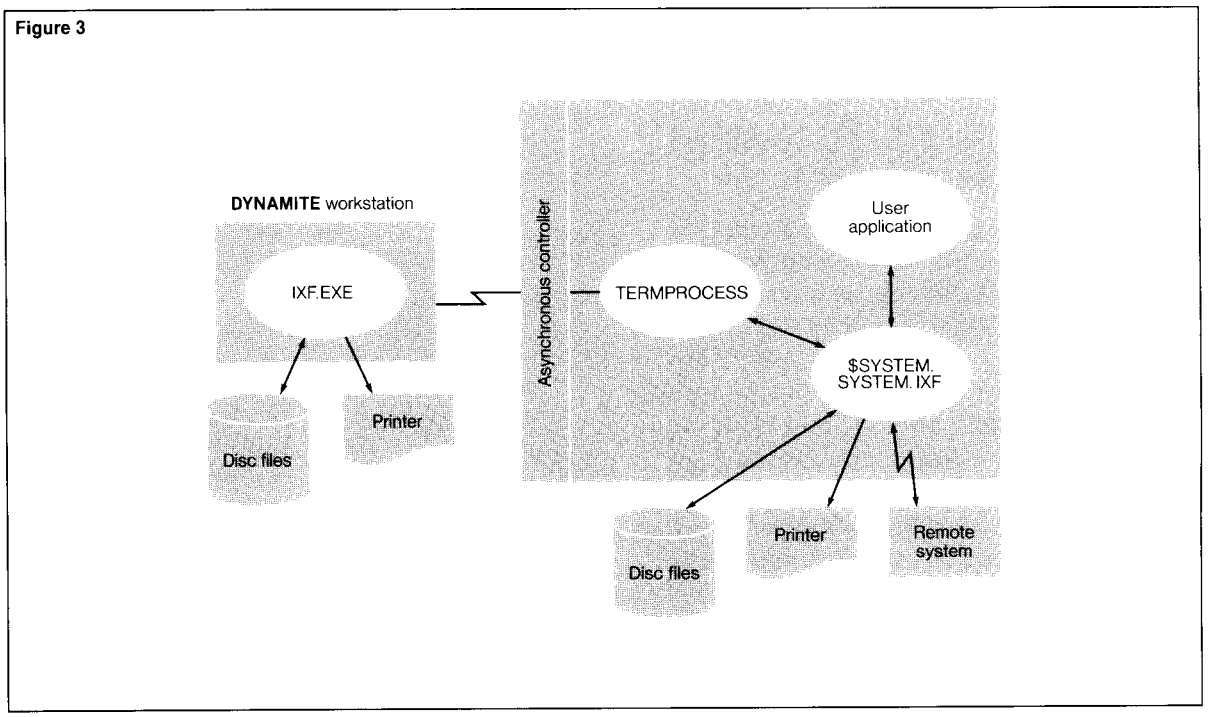
Figure 2.
Data-base conversion with PCFORMAT.

The Information Xchange Facility

Once the information is in a format suitable for workstation processing, the problem becomes one of access. To solve the problem, Tandem has developed the Information Xchange Facility (IXF) program. IXF allows a user to transfer files between a DYNAMITE workstation and a Tandem mainframe over asynchronous communications links (TERMPROCESS) or X.25 via an X.3 PAD (packet assembler/disassembler). IXF features:

- Powerful, line-oriented commands, which can come either from the command line or a command file.
- Sophisticated pattern matching, which can be used to restrict file searches.
- The ability to move multiple files to the same or different locations with one command.
- A rich set of options for tailoring commands.
- A sophisticated communications protocol, which not only ensures end-to-end integrity, but also increases the information-transfer rate by condensing streams of duplicate characters.
- The ability to transfer files directly between devices. For example, a disc file on a Tandem system may be printed directly on a printer attached to a DYNAMITE workstation.
- The ability to transfer binary (8-bit) and communications control characters.
- Data compression to increase the transfer rate.
- Automatic conversion of EDIT files to text files when they are moved to a workstation (and vice versa).
- The ability to print workstation files containing workstation-printer control characters on a Tandem system printer.

Figure 3.
The Information Xchange Facility (IXF).



IXF Components

As shown in Figure 3, the Information Xchange Facility consists of two components: the workstation portion (IXF.EXE) and a host portion ($SYSTEM.SYSTEM.IXF). For security, IXF always requires that transfer operations be initiated from the workstation. The workstation portion of IXF processes most of each command and, by user option, initiates the host portion to commence the transfer. Errors that occur at either end are automatically reported to the other end (and therefore to the user).

Three commands are currently supported: GET, PUT, and PRINT. These commands are discussed in more detail below.

GET Command. The IXF GET command is used to import files or data from the host to a workstation. The general format of the GET command is:

```
GET [( option [, ...])] Tandem_filelist
  AS DOS_filename [, ...]
```

where Tandem_filelist is a list of Tandem file names separated by commas, and DOS_filename is an MS-DOS file name.

Each of the Tandem file names may designate a disc file, a process, or a device located anywhere within the host system to which the workstation is attached (or within the network to which the host system is attached). The Tandem files are read sequentially to the end-of-file (EOF) and written to the workstation device or disc file specified. Note that IXF allows a host process to be specified. This process might be a user-written preprocessor that tailors the information to be transferred. For example, such a process might be programmed to fetch from a personnel data base only records for employees with children.

All disc files are read according to their structure. That is, unstructured files are read unstructured, EDIT files (code 101) are read as EDIT files, and ENSCRIBE files are read as structured files. This default can be overridden by means of the BINARY option (described below). Also, by default, every file is read with a record size of 132 bytes. Thus, in structured files with record sizes longer than 132 bytes, records are truncated unless the default is overridden by means of the REC option.

In the command syntax outlined above, users can insert pattern-matching or "wild-card" characters into the name of any disc file in the Tandem file list, and IXF will use them to select specific files from a larger set. The characters used are either an asterisk (*) or a question mark (?), where an asterisk signifies zero or more characters and a question mark signifies exactly one character.

For example, assume that a subvolume on the host system contains the following files and that those file names containing the letter *C* followed by at least one character are employees' "children files":

C0
EMPA
EMPB
EMPC
EMPD
EMPC1DEP
EMPC2TMP
EMPTITLE
SMRHLPC3

Under these circumstances, if users wanted to transfer only the children files to the workstation, they could give **C?** as the file name, and in response to that designation, IXF would select those files that contain a *C* (followed by at least one character) anywhere in the name. Thus, the files C0, EMPC1DEP, EMPC2TMP, and SMRHLPC3 would be selected. (File EMPC would not be selected because the question mark in the file-name designation indicates that the *C* must be followed by another character.)

There are many different ways of combining wild-card characters. The result is a very powerful file-selection tool. Also, since the syntactic group

Tandem_filelist AS DOS_filename

can be repeated any number of times, a virtually limitless transfer of information can be effected with one command. All IXF commands apply the default volume and subvolume (when necessary) to Tandem file names, and likewise, they apply the current directory to DYNAMITE file names.

Another feature of IXF is automatic file-name mapping. Mapping occurs when an asterisk is specified as the file-name part of the DOS file name, as in *A:*.DAT*, *B:**, or ***. In response to the above DOS file-name designations, the Tandem file name would be used as the DOS file name. For example, if *A:*.DAT* were given as the DOS file name in the wild-card example above, the resulting DOS files would be A:C0.DAT, A:EMPC1DEP.DAT, A:EMPC2TMP.DAT, and A:SMRHLPC3.DAT.

Yet another mapping feature is the ability to preserve DOS file-name extensions. As will be explained in connection with the PUT command, DOS file-name extensions are normally appended to the end of the DOS file name to form the Tandem file name when mapping is requested. These extensions may then be extracted during GET operations by specifying a file-name extension consisting entirely of question marks. For example, if one wanted to move the EMPA, EMPB, EMPC, and EMPD files mentioned above, one could use the command,

GET EMP? AS *.?

which would result in the DOS files EMP.A, EMP.B, EMP.C, and EMP.D.

Options associated with the GET command are: BINARY, PURGE, REC, and WAIT. The BINARY option overrides the structured reading of structured files. As a result, EDIT and ENSCRIBE files are read and transferred "unstructured." Note that the BINARY option does not control the type of data transferred, but rather the manner in which structured files are read.

The PURGE option allows IXF to purge files when it must overwrite existing ones. The REC option is used to specify the input record size.

Perhaps most interesting is the WAIT option, which allows the transfer to be delayed until the host portion of IXF is initiated separately. This option could be used if workstation users wanted to have files that were updated during overnight administrative processing transferred to their workstations when they were ready. To make this happen, they could simply run IXF before leaving for the evening, specifying the appropriate GET command with the WAIT option. Then, when the host files have been updated, the host portion of IXF could be automatically invoked by the host application that updated the files.

If the workstation IXF initiation were included as part of a "batch" file, it would even be possible to post-process the transferred information automatically on the workstation, where it could be ready for immediate use when the user returned the next morning. The asynchronous initiation of the host portion of IXF is discussed later in this article.

PUT Command. The IXF PUT command is used to export files or data from the workstation to the host. The general format of the PUT command is:

```
PUT [( option [, ...])] DOS_filelist
  AS Tandem_filename [, ...]
```

where DOS_filelist is a list of MS-DOS file names separated by commas, and Tandem_filename is a single Tandem file name.

Each of the DOS file names may designate a disc file or device on the workstation. The Tandem file name may specify a disc file, a process, or a device. Each DOS file is read sequentially to the EOF and written to the Tandem file specified. Since a process may be specified by the Tandem file name, a user-provided post-processor could be written under that name to custom-tailor the transferred information. For example, such a process might be programmed to update a data base automatically in response to the file transfer.

If the destination file specifies a disc file, an EDIT file is created by default, since there is no file code associated with DOS disc files. Note that when writing EDIT files, IXF automatically resets the high-order bit of each byte, interprets and discards some control characters, and simply discards the remaining control characters. Also, trailing blanks in each record are discarded. Thus, data will probably be lost if a user transfers a nontext file without invoking the BINARY option (described below).

Horizontal tab characters are treated specially, however. By default, a tab stop is set every eight characters. When a horizontal tab character is encountered, blanks are inserted up to the next tab stop.

Carriage return/linefeed couplets are also treated specially and interpreted as record separators. This means that IXF will not start a new EDIT file record until a record separator is encountered. Since EDIT files are limited to 255-byte records, truncation may occur. Also, trailing blanks are deleted from EDIT files, which means that DOS text files that are sent to the host and retrieved will not compare if they originally contained trailing blanks in any records. To avoid the EDIT file characteristics or to move nontext information, the users should invoke the BINARY option (described below).

To select a group of files for transfer from the workstation to the host, users can use pattern-matching or "wild-card" characters. As in the GET command, these characters are either an asterisk or a question mark, where the asterisk denotes zero or more characters and the question mark denotes exactly one character. However, DOS-style pattern matching is used, which means that the workstation interprets these characters somewhat differently than the host did. In particular, it is important to realize that when a question mark is used at the end of a file name designation, the question mark will also match zero characters.

The following example illustrates the differences between DOS and host pattern matching. Assume that a directory on the workstation contains the following files:

```
C.DAT
C0.DAT
COMP.TXT
EMP.A
EMP.B
EMP.C
EMP.D
EMPC1DEP.DAT
EMPC2TMP.DAT
EMPTITLE
SMRHLPC3.DAT
```

In such a case, the pattern *C?.** would select both C0.DAT and C.DAT (but it would not select COMP.TXT). The pattern matches C.DAT because the question mark matches to zero characters in that case. A question mark only maps characters to zero when it is at the end of a file-name part (e.g., C?.DAT or C.?). Thus, *COM?P.TXT* would not select COMP.TXT from the above files.

Also, the pattern **C?**, illustrated in the GET command example, could not be used with the PUT command, because the asterisk may only be used as the last character in a pattern with DOS-style pattern matching. The pattern *C?*.** would select C.DAT, C0.DAT, and COMP.TXT. (*C*.** would also select the same files). The pattern **.DAT* could be used to select all files that have an extension of DAT. As with the GET command, the syntactic group of

```
DOS_filelist AS Tandem_filename
```

can be repeated any number of times.

The PUT command also provides file-name mapping. Mapping occurs when an asterisk is specified as the file-name part of the Tandem file name (for example, *$VOLUME.SUBVOL.**, *SUBVOL.**, or simply *). In each case, the DOS file-name extension (if any) is appended to the DOS file name (after the DOS file name has been truncated as needed to preserve the extension and yet produce an eight-character Tandem file name). For example, a seven-character file name with a two-character extension (EMPFILE.DB) would be truncated to form the eight-character name (EMPFILDB).

When the preservation of file-name extensions is not desired, it may be suppressed with the use of the NOEXTS option described below. The combination of differences in pattern matching and possible truncation of DOS file names means that careful thought should be given to file-naming conventions with an eye to simple transitions between DOS and Tandem file names.

The available options for the PUT command include BINARY, NOEXTS, NOTABS, PURGE, REC, TABS, and WAIT.

The BINARY option causes an odd-unstructured file to be created when the host destination is a disc file. Odd-unstructured is used to avoid adding data when odd record lengths or the amount of information transferred results in an odd byte count. BINARY also causes all control characters to be ignored and simply passed through.

The NOEXTS option inhibits the concatenation of file-name extensions when host names are formed. In the above example, EMPFILE.DB would then be formed as EMPFILE.

NOTABS inhibits horizontal-tab-character interpretation. Horizontal tabs are simply stripped out (except with the BINARY option).

The PURGE, REC, and WAIT options are identical in function to the corresponding options associated with the GET command, except that REC also affects the output (host) record size.

The TABS option provides for the selection of tab-stop locations.

PRINT Command. The IXF PRINT command is similar to the PUT command, except that it is tailored for printing files. The general format of the PRINT command is:

```
PRINT [( option [, ...])] DOS_filelist
  TO Tandem_filename [, ...]
```

where DOS_filelist is a list of MS-DOS file names separated by commas, and Tandem_filename is a single Tandem file name.

Each of the DOS file names may designate a disc file or device on the workstation. The Tandem file name is restricted to either a process or device. Each DOS file is read sequentially until the EOF and written to the Tandem file specified. The control characters (form feed, backspace, carriage return, and linefeed) are interpreted as printer control characters in the same way they would be interpreted by a printer attached to a workstation. In addition, horizontal tab characters are interpreted (or, optionally, ignored) in the same way they are interpreted by the PUT command.

Pattern matching to direct the file search is available and is identical in its functions and restrictions to that described for the PUT command. File-name mapping is also allowed and works just as it did with the PUT command. File-name mapping can be useful for tagging files sent to a spooler process with the source file name. For example, the command

```
PRINT P1.LST, P2.LST, P3.LST, P4.LST
  TO $S.#HOLD.*
```

would cause the names $S.#HOLD.P1LST, $S.#HOLD.P2LST, etc., to be formed. These names would show up in the JOB command of PERUSE.

The available options for the PRINT command include NOEXTS, NOSKIP, NOTABS, TABS, and WAIT. The NOEXTS, NOTABS, TABS, and WAIT options are identical in function to the corresponding PUT options.

The NOSKIP option causes a SETMODE 5 to be performed (see the *GUARDIAN Operating System Programmer's Guide*). Essentially, this option inhibits the automatic form feed at the bottom of each page, provided the printer carriage-control tape is set up properly. This option is useful for printing many documents produced by third-party programs that do not expect automatic form feed.

Running IXF

IXF must be initiated from the workstation (i.e., by an external command). The workstation must be connected to a Tandem host through an asynchronous connection using TERMPROCESS. The host must be in conversational mode, and it must be running a command interpreter (COMINT). IXF attempts to solicit a prompt from COMINT and, when it is successful, attempts to initiate the host portion of IXF (if the WAIT option is not being used).

Once the host portion is running, it switches into its information-exchange protocol to transfer data. When contact is successfully made with the host IXF, a banner containing the host version is displayed. Following that are displays of the files copied or any error messages. If an existing file is overwritten, a "purge" message is displayed, along with the file name.

If the WAIT option is invoked, the host portion of IXF must be initiated separately. This can be performed from a different terminal (or DYNAMITE workstation running EM6530) connected to the host and running a Command Interpreter. The device name of the DYNAMITE workstation must be passed as a parameter (for example, IXF $TNT01). The OUT file is used for error messages, and it may be any disc file, process, or device (e.g., IXF /OUT IXFLOG/ $TNT01).

The host portion of IXF can also be initiated by any process running on the host. All that is required is to call NEWPROCESS and pass a "start-up" message to IXF containing the information required in the output-file-name and parameter-string fields.

The destination for error messages should be specified as the output file name, and the device name of the DYNAMITE workstation waiting for the host should be in the parameter string (terminated with a null byte). If the home terminal of the process initiating IXF is the same as the DYNAMITE workstation, however, the key word REMOTE should be used in place of the device name in the parameter string. The output file name passed in the start-up message may be the process that is doing the NEWPROCESS call.

The format of the start-up message is described in the *GUARDIAN Operating System Programmer's Guide*. The NEWPROCESS procedure, process control, and interprocess communications are also described in detail in this manual. Remember that the workstation portion of IXF must always be initiated first (with the WAIT option).

Regardless of the manner in which IXF is initiated, its status may be checked at the workstation by typing CONTROL-Q (hold the CTRL key down and press Q). The resulting display gives information on the number of packets exchanged and communications errors encountered. A normal exchange causes the number of packets to steadily increase.

An occasional communications error is no cause for alarm, as the automatic detection and retry mechanisms of IXF recover from nearly all error conditions. If, however, a very high number of errors is noted, or if IXF terminates with an EXCEEDED RETRIES error, this may indicate that the communications link is broken or sufficiently error-prone that an exchange is difficult or impossible. In such cases, it is best to retry the exchange a few times to see if the problem persists. Dial-up modem connections are particularly susceptible to poor line conditions. Often, hanging up and redialing resolves the problem.

When IXF is exchanging information with a process on the host, the host process must function as a server. That is, IXF calls the GUARDIAN routine OPEN to open the server process as a file. The server process receives an OPEN system message through its $RECEIVE file and must reply to it with the GUARDIAN REPLY routine. The server process must also expect to receive I/O requests through its $RECEIVE file, interpret them, perhaps perform other I/O or computation, and respond to the requests. To terminate a GET function from a server process, the process should respond to a READ request with an EOF indication (File System error 1).

When IXF is finished, it severs the connection to the server process with a CLOSE system message. In the case of a PUT function, the CLOSE message is the only indication the server receives to mark the termination of the exchange. Note that the server process may be the object of multiple exchanges and so should be prepared to handle multiple OPEN-I/O-CLOSE sessions. For example, multiple exchanges would occur to process $SRVR1 with the following IXF command issued at a DYNAMITE workstation:

```
PUT EMPTITLE,SMRHLPC3.DAT
  AS $SRVR1.#PUT.*
```

Full details on interprocess communications and $RECEIVE file handling can be found in the *GUARDIAN Operating System Programmer's Guide.*

Conclusion

The DYNAMITE workstation host-integration package closes many of the gaps between workstations and host systems. The EM6530 terminal emulator allows workstation users to see host data and interact with other host users. The PCFORMAT file-conversion utility allows host data bases to be converted into a form more acceptable to workstation software packages. Finally, the Information Xchange Facility provides the means to efficiently exchange data between host and workstation and, with user-written pre- and post-processing programs, can be tailored to many different applications. In short, the DYNAMITE workstation can provide the best of both worlds: efficient, personal, local computing with quick access to powerful, centralized, and fault-tolerant mainframe computing.

References

GUARDIAN Operating System Programmer's Guide. Part no. 82357 A00. Tandem Computers Incorporated.

654X Workstation—Information Xchange Facility. Part no. 82669 A00. Tandem Computers Incorporated.

654X Workstation—Operations Guide. Part no. 82658 A00. Tandem Computers Incorporated.

654X Workstation—PCFORMAT User's Guide. Part no. 82679 A00. Tandem Computers Incorporated.

Stan Kosinski has an M.S. in Electrical Engineering from the University of California. Currently, he is a member of the DYNAMITE development staff in Austin, Texas, where he developed the Information Xchange Facility. Since joining Tandem in December 1980, he has worked on a variety of projects, including the development of the ENCORE stress-test generator. Before joining Tandem, Stan spent ten years working in the areas of performance prediction and analysis, computer hardware and software architectures, operating systems, and data communications.

The V8 Disc Storage Facility: Setting a New Standard for On-line Disc Storage

On January 15, Tandem introduced the V8 Disc Storage Facility. The V8 uses a unique new packaging design to optimize disc performance for high-volume on-line transaction-processing applications. It provides both large storage capacity and high throughput by packaging eight 168M-byte disc drives in a single cabinet (see Figure 1). This adds up to a total of 1.3G bytes for storing large amounts of information with the added benefit of eight actuators for high performance.

The V8 speeds data access by minimizing disc-access time and queuing. This makes the cost per disc access per second lower (and the number of transactions serviced per second higher) than that obtained with standard disc-storage facilities.

The V8 provides parallel paths to data through multiple actuators. When a file is partitioned across eight discs (and overlapping seeks are accounted for), eight I/O requests can be serviced simultaneously. When a file resides on one disc volume of a conventional disc, concurrent disc accesses must be queued and serviced separately, resulting in slower response time and reduced system throughput.

The average seek time per drive for the V8 is only 20 ms, which, when added to latency, gives an average time-to-data of only 28 ms.

To ensure data availability, disc mirroring on the V8 allows duplicate data to be stored on an independent disc drive so it can be accessed even if one drive should fail. Since the V8 contains multiple disc drives and two power cords, one for each of four drives, mirrored discs can reside in the same cabinet.

The V8 sets new standards of reliability with very high mean time between failure for both the disc drive and the power supply. The sealed head and disc assemblies of the Winchester drives require no preventive maintenance.

When service is required, the V8's field-replaceable drives and power supplies result in fast and efficient service. Since each drive has its own power supply, a malfunction in one does not shut down the others. On-line service also allows a unit to be replaced with no interruption to current operations.

The high reliability and easy serviceability of the V8 result in a significantly lower cost of ownership and higher system availability for users. Even if a drive without a mirror fails, only a small part of the data base becomes unavailable, whereas a failure in a conventional disc drive results in a substantial loss of data.

The V8's unique design packs eight disc drives in a cabinet that occupies only ten square feet of floor space (including service clearance), making it the most efficient user of computer-room space in the industry. It stores 134M bytes per square foot, compared to 100M bytes per square foot for units with a comparable storage capacity. Even more significantly, it houses eight actuators in ten square feet of floor space, making it five times more space-efficient than competitive drives.

Consistent with Tandem's system modularity, the V8 Disc Storage Facility is offered with four, six, or eight drives. Additional Winchester drives can be added in increments of two to a maximum of eight per cabinet. Expansion is easy because each drive is a separate plug-in unit that can be quickly added on-line. Price/performance is optimal since users need only buy the capacity they need when they need it.

Figure 1

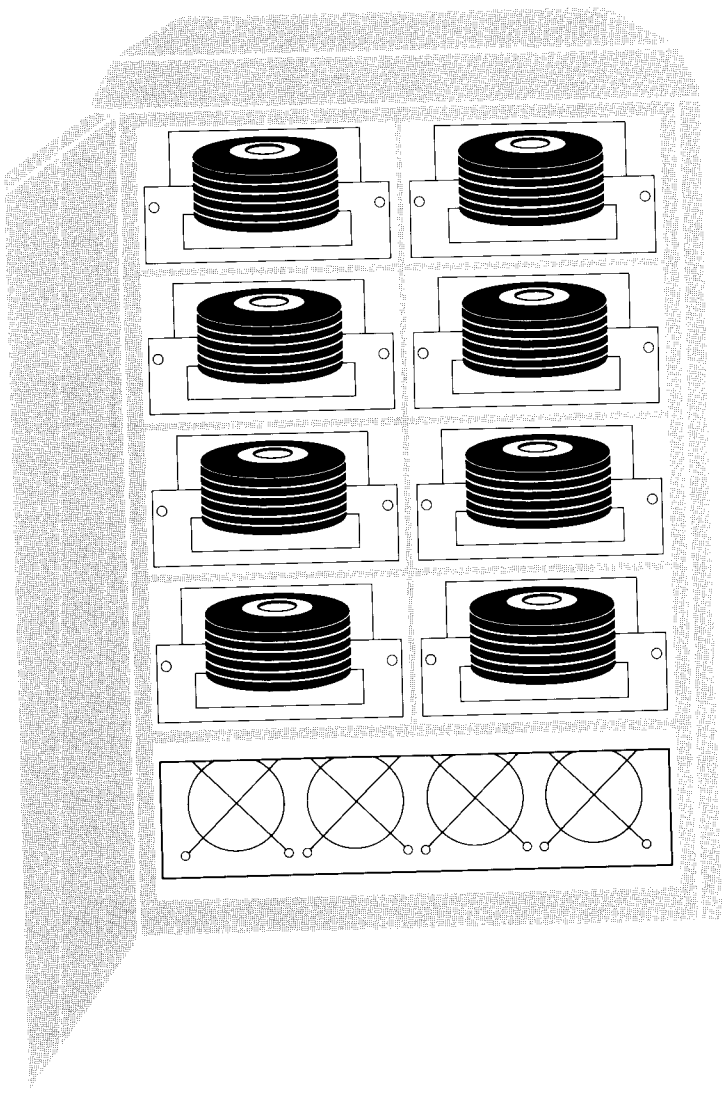


Figure 1.
By packaging eight disc drives in a single cabinet, the V8 provides the large capacity and high throughput needed for high-volume on-line transaction-processing applications.

Mary Ann Whiteman is a product marketing manager for data base and peripheral products. She joined Tandem in October 1984 and, since then, has coordinated the announcement programs for the V8 Disc Storage Facility and DP2. Before joining Tandem, she worked in application development, systems engineering, and product marketing/management for data-base, communications, and other software products. Mary Ann has a B.A. in Mathematics from Douglas College of Rutgers University and is currently working on an M.B.A. degree at Santa Clara University.

Introducing the 3207 Tape Controller

The 3207 Tape Controller is a new-generation I/O controller designed to provide complete data integrity for any single-point fault and to locate faults as a part of its normal operating sequence. Its features include:

- Lock-stepped microprocessors.
- A fully protected internal bus.
- Firmware containing embedded fault-isolation sequences in the operational code and power-on diagnostic sequences.
- Loop-back checking in the device interface.
- Self-checking logic in state machines, counters, registers, and other logic elements.

For this new, more complex design, the number of basic controller gates has increased by at least 30% and the number of interconnects by at least 20% over those of its predecessor. Even with this complexity, power consumption and real estate remain low and reliability high.

These seemingly conflicting design goals were met by the use of state-of-the-art VLSI gate-array technology. Most of the logic in the 3207 Tape Controller is packed into ten gate-array modules; the rest is contained in approximately 190 conventional MSI/SSI chips, memory, and two Motorola 68000 microprocessors. In terms of basic complexity, this is equivalent to approximately 30,000 gates (excluding microprocessors and memory).

Functional Description

A Tandem data-base I/O controller controls the transfer of data between a mass storage device and one of two I/O channels. The 3207 Tape Controller supports two 45- or 125-ips (inches-per-second) NRZI/PE tape drives. The concepts and design methodologies used are easily adaptable to other controller designs. Figure 1 shows a block diagram of the 3207 Tape Controller.

Figure 1

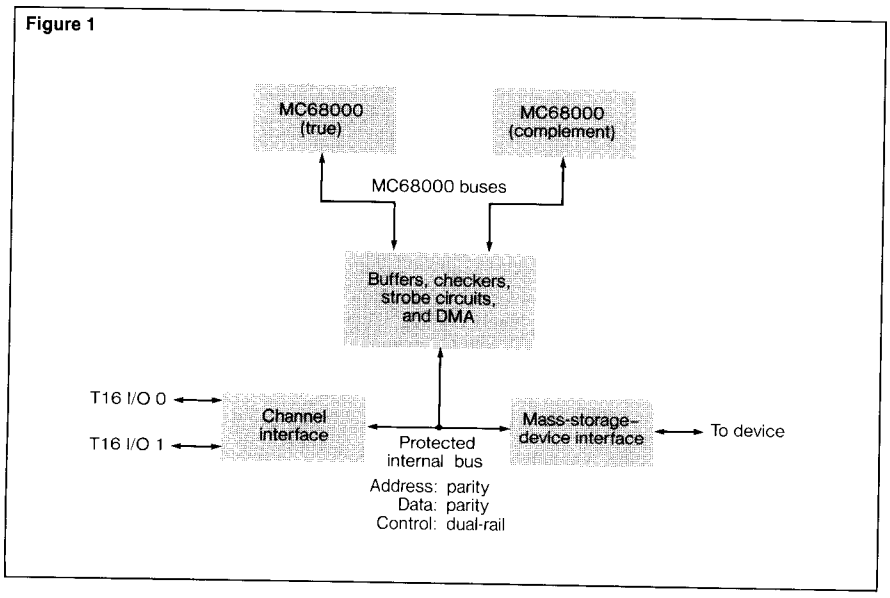


Figure 1.
A functional block diagram of the 3207 Tape Controller. The four primary logic groups of the 3207 are the microprocessors and their checking logic, a protected internal bus, a dual-ported Tandem I/O controller channel interface, and a device formatter and interface.

One of the design requirements for the 3207 Tape Controller was the use of off-the-shelf microprocessors. The Motorola 68000 microprocessor was chosen because it has these characteristics:

1. A 16-bit data bus, allowing word-oriented architecture for data transfers.
2. A powerful instruction set suited to a high-level-language implementation of the firmware.
3. Built-in protection to separate memory spaces (code vs. data or supervisory vs. user) based on external function-control pins.
4. A powerful error-exception capability using an error-exception input pin.
5. Existing in-house program-development tools for firmware, based on the Tandem Programming Language (TPL).

Off-the-shelf microprocessors seldom implement built-in fault-detection circuits, and the MC68000 is no exception. To provide fault detection, the 3207 Tape Controller uses two of the MC68000 processors in lock-stepped mode. This allows it to isolate faults simply and accurately by localizing them in time and place. It checks the data bus, the address bus, the control strobes for data read/write, the interrupt lines, and the bus-arbitration lines independently, on each bus cycle. This approach is a comprehensive one, differing from any other known method of lock stepping.

The Motorola 68000 microprocessor is a 16-bit processor (externally) with 24 address lines. The strobe lines it uses for memory-mapped I/O reads and writes are AS*, UDS*, LDS*, DTACK*, and R/W*. It uses the arbitration lines, BR*, BG*, and BGACK*, for direct-memory-access (DMA) activities. It supports seven levels of interrupts by using IPL0*, IPL1*, and IPL2*.

The MC68000 also has a RESET* line and two other lines, BERR* and HALT*, which are very effectively used in the 3207 Tape Controller for fault detection on the lines mentioned above. The architecture of the 3207 is closely coupled with the firmware to process data from the error-detection logic.

The two types of error processing used by the 3207 Tape Controller are:

1. Lock-stepping MC68000s and associated logic. The primary motive in this architecture is to create a protected, modified MC68000 bus and perform data transfers on it. This is accomplished by using two MC68000s and self-checking logic. Any miscompare between the two MC68000s or any error in the associated self-checking logic is considered fatal and the processor is halted.
2. Acting as an interface to the protected internal bus. If the two MC68000s lock step without errors and the associated checking logic functions normally, the internal bus is good. Memory-mapped registers and other logic that make up the interface to the internal bus may detect errors caused, for example, by faults in internal logic, interface buffers, and printed wiring.

   The checking logic for such faults is part of the external circuitry (particularly the VLSI module), and such errors are reported to the microprocessors. They process these errors as nonfatal ones and enter into a diagnostic mode of exception processing to locate the faults. These faults can be accurately located more than 90% of the time.

Definitions

In this article, the terms *check* and *test* are used frequently. In order to understand the context in which these terms are used, the following definitions are provided:

- *Error*. A condition in observable output that is abnormal for normal input at the time of observation.
- *Fault*. An abnormal condition in a physical element of a logic circuit.
- *Check*. To detect faults made through the observation of errors. In highly checkable circuits, every physical fault should result in abnormal output for some normal input.
- *Test*. To locate faults. (The input need not necessarily be normal.)

Figure 2

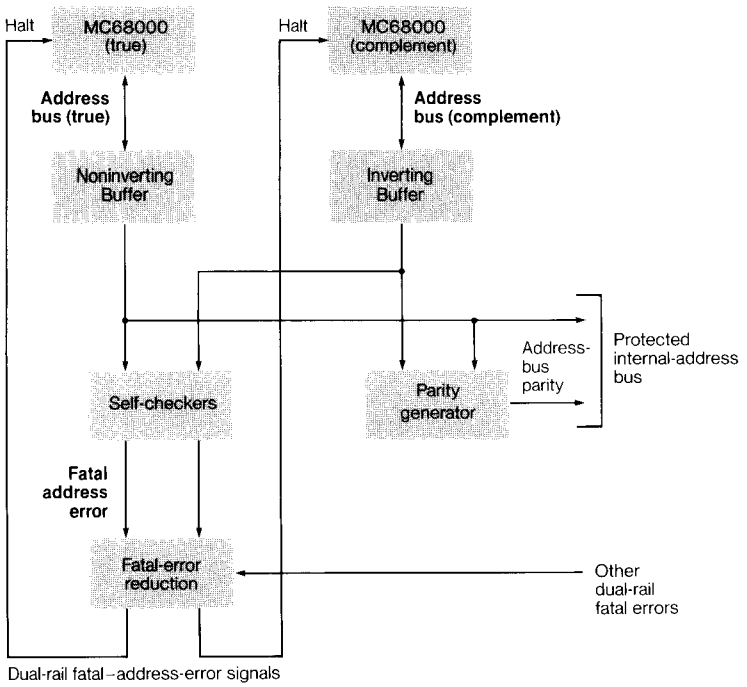
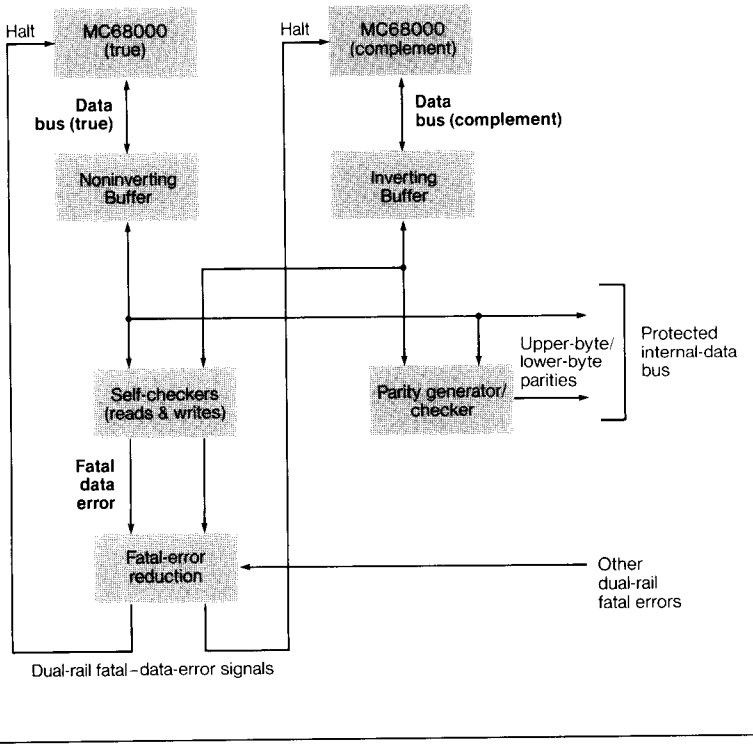


Figure 3



Protected Address Bus

Figure 2 shows the implementation of the protected address bus. The address lines from the MC68000 called the *true* processor are buffered with noninverting buffers, while the address lines from the other MC68000, called the *complement* processor, are buffered with inverting buffers. The 3207 Tape Controller's checking logic uses morphic-reduction circuits to check these two sets of address lines.

This checking logic produces a pair of signals called *Fatal Address Errors*. These signals are gated along with other fatal errors (explained later). The output of this logic appears on a pair of *Fatal Error* lines, which are respectively connected to the halt lines of the two processors. Then, if an address mismatch occurs, the processors are halted. All address errors are detected on the same bus cycle and are considered fatal, causing a halt on the following bus cycle.

The checked true or complement address lines are used to generate odd parity across the address bus. The true address bits, with an odd-parity bit, constitute the internal address bus of the controller. All the other peripheral circuits are designed to check parity on the address bus with every read and write. These address-parity errors are considered nonfatal and are processed by the microprocessor as exceptions in order to locate the faulty circuit.

Protected Data Bus

The comparison of true and complement data buses with true and complement microprocessors is similar to the address comparison described in the previous section. All data miscompares between the microprocessors, and errors from the associated checking logic are considered fatal. Figure 3 contains a diagram of a protected data bus.

Figure 2. *Implementation of the protected address bus.*

Figure 3. *Implementation of a protected data bus.*

The checked data-bus lines are used to generate a lower-byte, even-data-parity bit and an upper-byte, even-parity bit. Two data-parity bits are used both to improve the detection of data errors and to allow for byte or word operations on data.

All I/O-mapped registers check for correct data parity while being written to and report errors back to the microprocessor on the same bus cycle. Also, an external parity-checking circuit monitors the data bus for errors during reads by the microprocessor and reports the errors to the microprocessor on the same bus cycle. The data-parity errors are treated as nonfatal errors, and the firmware processes them on an exception basis to locate the faulty circuit.

Dual-railed Control Strobes

Five control strobes (AS*, UDS*, LDS*, DTACK*, and R/W*) are related to data reads and writes from each microprocessor. The two copies of each of these signals should be synchronous within the tolerance limits of the two microprocessors. These control lines are compared through the use of self-checkers; any miscomparison is considered fatal.

The independent checking of the control strobes, one of the unique features of the 3207 Tape Controller's lock-stepping design, achieves these design goals:

1. Isolation of errors to the control bus. This makes it easy to troubleshoot during faulty controller operation.
2. Prevention of error propagation. The context under which the error occurred is saved. This is an important aid to locating transient and intermittent error conditions.
3. Ability to disable the lock-step circuit. This allows for the isolation of the failing microprocessor and the debugging of the design and code with an in-circuit emulator.

The checking of the control strobes presented several design challenges. Even though both microprocessors are timed with the same clock, one of the processors may be operating at the minimum delay specification while the other operates at the maximum delay specification because of process variations in parts fabrication.

The effect of the difference in delay on data reads and writes is to necessitate more stringent set-up and hold margins. Modified strobes are generated to take care of the new stringent timing requirement for reads and writes. The generation itself is done in a self-checking manner and is dual-railed. All these challenges are met in the 3207 Tape Controller by unique circuit-implementation techniques.

All peripheral circuits that have an interface to these modified control strobes are designed so that they check these lines on a read or write. Any error detected is reported to the microprocessor for nonfatal exception processing.

Protection Techniques Used in Interrupt Handling

Three interrupt lines exist for each processor, making seven levels of interrupt possible. All the interrupt lines are dual-railed, a unique feature of the lock-stepping technique.

Interrupts to the processors are normally considered asynchronous. When two microprocessors receive asynchronous interrupts caused by tolerance differences in set-up times, one of the processors may see the interrupt and respond, while the other may not. This can result in address, data, or control-strobe errors. To eliminate this possibility, all interrupts are handled synchronously, and each interrupt signal is dual-railed and handled independently by each processor. The only types of error that can result in an address, data, or control error are transient or permanent faults in the interrupt lines themselves.

Protection Techniques Used in Bus Arbitration and DMA

There are three bus-arbitration lines in the MC68000 microprocessor. They are *Bus Request*, *Bus Grant*, and *Bus Grant Acknowledge*. The first and the last are input to the microprocessor; the second is output from it. The DMA controller is designed so that dual-railed bus-request and bus-grant-acknowledge signals are given independently to the two processors, and bus-grant signals received from both processors are checked. Bus-request and bus-grant-acknowlege signals are synchronized with the processor clock.

The DMA-controller state machine is duplicated and checked on every state-machine clock cycle. DMA address generation is protected by parity-predicted binary counters. Any error detected in any of the logic described above is logged into an internal register and forces the DMA machine into an error state. When an error state occurs, the DMA gives up the bus.

During DMA, the data bus is also monitored by an independent parity-checking circuit, and any error is logged into a register. After the operation completes, the microprocessor reads the DMA status and data-parity status to see if the DMA completed normally. This helps isolate the fault quickly and minimizes the chance of contaminating massive blocks of data. DMA handshake lines that have faults are handled through a time-out mechanism by which the bus is forced back to the microprocessor.

Checking and Testing of Gate-array Modules

Earlier sections of this article outlined various aspects of checking in both the 3207 Tape Controller and the microprocessor, and summarized how a protected internal bus is produced. This section summarizes the checking circuits needed in the peripheral circuitry to check both the internal bus and other logic dependent on the function of each circuit.

The three major functional blocks in this controller are:

1. *The processing element*, consisting of two MC68000 microprocessors, a DMA gate-array module, and a Processor Support Module (PSM) containing the self-checking and strobe-generation circuits.
2. *The channel interface*, consisting of logic for dual-channel ports. Each port contains a port-register module (PRM), a port-control module (PCM), and conventional MSI buffer chips.
3. *The device interface*, consisting of interface and formatter circuits to support two 45- or 125-ips NRZI/PE tape drives, phase-locked loop circuits, tape interface buffers, and the following four gate-array modules:
   a. Formatter timing module (FTM).
   b. Write formatter module (WFM).
   c. Read control module (RCM).
   d. Read formatter module (RFM).

Checking Techniques Used in 3207 Gate Arrays

The checking techniques used by the Internal Bus Interface include the following:

- Address parity check on data reads and writes.
- Data parity check on writes.
- Control strobe check on reads and writes.

Those for the T16 Channel Interface include:

- Sequence check on T-Bus (done with a sequence detector).
- Parity-predict protection on all counters.
- Parity-predict protection on state machines.
- DMA-handshake-interlock monitoring.

Finally, those for the Device Interface include:

- Parity-predict protection on state machines.
- Parity-predict protection on counters and registers.
- Loop-back capability for device-interface signals, used by firmware for in-line testing.

Testing Techniques Used in 3207 Gate Arrays

The two primary testing techniques used with the 3207 Tape Controller are:

1. *Scan*. The internal registers of five of the gate arrays can be scanned. The scanning technique is primarily used to test gate arrays at the component level.
2. *Read/write*. Storage elements that cannot be scanned can be written and read back by the microprocessors over the internal bus. Read-only status registers can be initialized with master reset.

Other standard testability practices, such as the ability to clock from external sources and soft pull-up/pull-down, have also been adopted.

Checkability and Testability Coverage in VLSI Modules

Table 1 shows checkability and testability coverage in the VLSI modules of the 3207 Tape Controller. The coverage is expressed as the percentage of the total number of gates used in the design that are checkable or testable.

Hardware self-checking is defined as the ability of the chip to detect an error in normal operation that results in one or more of the following actions by the chip:

1. An error flag is set in an internal register.
2. To locate the fault, the bus-error line is pulled active for error-exception processing by the MC68000s.
3. The halt line is pulled active to freeze the microprocessors, indicating fatal errors.

Table 1.
Checkability and testability in 3207 Tape Controller VLSI modules.

Chip	Scan	Testability of Storage Elements		Hardware self-checking	Stuck-node coverage
		Read	Write		
PRM	0%	100%	100%	95%	100%
PCM	10%	90%	30%	80%	100%
DMA	0%	95%	95%	100%	99%
PSM	30%	70%	50%	95%	96%
FTM	20%	100%	100%	95%	98%
WFM	100%	100%	40%	100%	99%
RFM	100%	0%	0%	100%	97%
RCM	100%	100%	12%	100%	96%

Conclusion

The development of the 3207 Tape Controller is a pioneering effort in the design and implementation of VLSI-based products. By using VLSI technology, the design engineers were able to use the complex design techniques required to improve the data-integrity and fault-isolation capabilities of the basic I/O controller without incurring the penalties of reduced reliability, increased real estate, and power consumption that would have accompanied similar efforts a few years ago. Many of the techniques outlined in this article will also be used in future Tandem products.

Acknowledgments
The 3207 Tape Controller was primarily developed by four design engineers, including the author. The author wishes to acknowledge the contributions made by the other members of the design team: Ed Rhodes, Albert Lui, and Mark Walker.

Sri Chandran is a member of the Hardware Engineering group. He is the leader of the 3207 Tape Controller development project.

Robustness to Crash in a Distributed Data Base: A Nonshared-memory Multiprocessor Approach

Since attention first turned to the problem of data-base recovery following system crash, computer architectures have undergone considerable evolution. One direction such evolution has taken is toward fault-tolerant, highly available, distributed data-base systems. One such architecture is characterized by a single system composed of multiple independent processors, each with its own memory. This paper examines the inadequacy of both the traditional definition of system crash and the conventional approaches to crash recovery for this architecture. It describes an approach to recovery from failures that takes advantage of the multiple independent processor memories and avoids system restart in many cases.[1]

Introduction

With the emergence of on-line update in transaction-processing applications, log-based data-base–recovery techniques have evolved to provide robustness to crash or system failure. Log-based crash-recovery techniques have received considerable attention in the literature [4,6,8,9,10].

The strategies adopted by the proponents of these techniques fall into two basic categories. Both postulate the existence of two types of memory [4]:

1. *Main memory,* which is volatile, hence does not survive system failure.
2. *Secondary storage,* which is stable or nonvolatile, hence usually survives system failure.

In the first strategy, a transaction writes an *intentions list* rather than updating data-base pages in real time. The application of a transaction's intended *updates* to the actual data-base pages is deferred until transaction commit, at which time the transaction's intentions list is written to a secondary storage log, following which the updates are applied to the actual data-base pages. If a failure occurs during the application of the intentions list, the recovery procedure consists of restarting the application of the intentions list from the beginning. This technique has been described by Lampson and Sturgis in [8].

In the second strategy, a transaction effects its data-base updates in real time, but a so-called *write-ahead-log* protocol governs the migration of the updated data-base pages from a memory buffer pool to secondary storage. According to this protocol, described by Gray in [4], no updated data page is permitted to be written to secondary storage before the log records describing the updates to that page have been written to the secondary-storage log. At commit time, transaction recoverability is achieved by forcing to stable storage all log records related to the committing transaction.

Using either of the above strategies, data-base recovery following a crash is characterized by having recourse to the log stored on secondary storage in order to ensure that committed transactions are applied and uncommitted transactions are removed from the data base. A difference between the two strategies lies in the type of log information required for crash recovery. In the case of deferred update, only redo information need be logged. In the case of real-time update with write-ahead log, both undo and redo information must be logged [5].

Reexamination of the Term *Crash*

Central to the strategies used in the conventional approaches to crash recovery is the definition of a *crash* or *system failure* as the loss of the contents of main memory [9]. The inadequacy of this definition of system failure becomes evident when applied to a nonshared-memory multiprocessor architecture. The concept of *main memory* as a unique and shared resource constituting a single point of failure is inappropriate for multicomputer systems. In a system architecture in which multiple independent processors, each with its own memory, are connected to form a single system or node via interprocessor buses or local area network, the use of the term *crash* to denote an all-or-nothing state of the system loses its validity.

The term becomes even less meaningful when applied to a long-haul network consisting of multiple shared-memory nodes, or even of multiple multicomputer nodes. Such configurations raise the possibility of partial crashes caused by individual processor failures within a node or caused by node failures within a network. A fault-tolerant system design may allow certain failures within a node to be handled without requiring system restart. If a partial failure does not require system restart, neither should it require full data-base restart. However, the problem of the total failure or crash of a multicomputer node still remains and must be handled.

A corollary to the generalization of the concept of crash is the generalization of the concept of crash recovery. If, as in the above definition, secondary storage is viewed as the only storage that survives failures, then crash recovery must be based on a secondary-storage log and system restart is required.

If, on the other hand, a processor failure does not imply the failure of other processors, then recovery techniques not requiring system restart or recourse to secondary storage are possible. If a portion of the "log" were copied from the memory of one processor to that of another during normal processing, and one of these processors survived the failure of the other, recovery from the partial system failure could be effected using the log information from the memory of a surviving processor while system operation continued "on-line."

Tandem Computers has implemented a multiprocessor architecture using the above concepts. The next section presents a brief description of Tandem's system architecture in order to motivate a more general approach to identifying and recovering from both partial and total system failures. Subsequent sections define robustness to single and multiple processor failures in a Tandem system. A discussion of Tandem's implementation of fault tolerance and the evolution of its design follows.

Figure 1

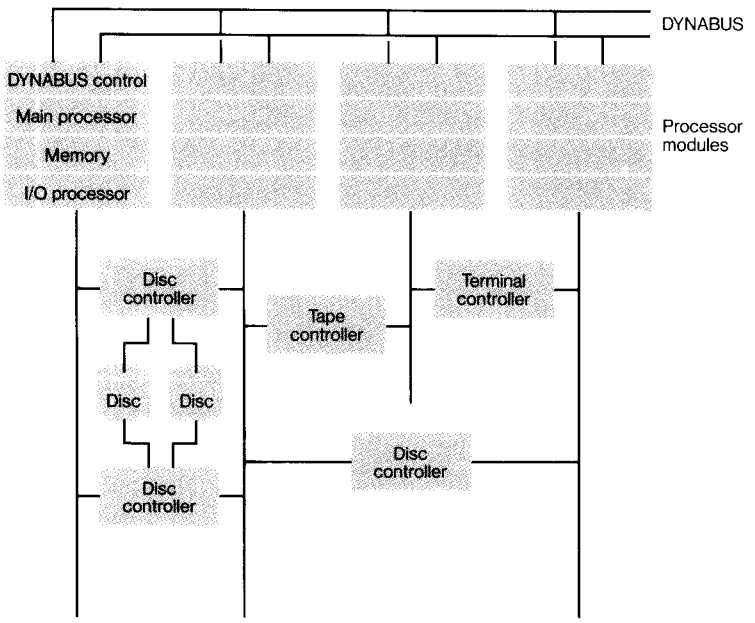


Figure 1.
Tandem hardware architecture. A system consists of from 1 to 16 processor modules, each with its own memory, interconnected via the duplexed DYNABUS. The hardware configuration provides redundant paths to the peripherals.

Architectural Overview

The hardware architecture of a Tandem system is described in [7]. Illustrated in Figure 1, it is based on multiple independent processors that are interconnected by dual high-speed buses to form a single *system (node)*. The goals of the architecture are fault tolerance, high availability, and modularity. Hardware redundancy is provided such that the failure of a single module does not disable any other module or disable any intermodule communication.

Normally, all components are active in processing the work load. When a component fails, however, the remaining system components automatically take over the work load of the failed component. Each of the (up to 16) processors in a system has its own power supply, memory, and I/O channel. Memory has battery backup power capable of saving system state for several hours in the event of power failure. Each I/O controller is connected to the I/O channels of two processors, and each I/O device, such as a disc drive, may be connected to two controllers. A given disc volume is directly accessible from two processors. Disc-volume availability, despite media failures, is provided by optional duplication, or *mirroring*, of drives.

System resources are managed by a message-based operating system, described in [1]. The Message System, a component of the operating system, provides communication between processes executing in the same or different processors, making the distribution of hardware components transparent to processes. Through its Message and File Systems, the operating system makes the multicomputer structure appear as a unified multiprocessor to higher levels of software.[2]

Built on this architecture is a distributed data-management and transaction-management system called ENCOMPASS. Described in [3], ENCOMPASS allows data to be distributed across multiple processors and discs within a single node, or even within multiple nodes of a Tandem long-haul network. It supports the *transaction* concept [5] in this distributed environment. The transaction concept is implemented by means of a log and real-time (as opposed to deferred) update. Transactions can span multiple discs (connected to multiple processors) within the same node or on multiple nodes of a Tandem long-haul network.

Updates to a file may or may not be protected by transaction auditing, depending on the value of the file attribute called *audited*. (Henceforth, the terms *log/logging* and *audit trail/auditing* will be used interchangeably).

ENCOMPASS supports three kinds of structured file organization: (1) key-sequenced, (2) relative-record, and (3) entry-sequenced. A key-sequenced file is organized as a B-tree on the primary key field. All three file organizations can have alternate keys. Alternate keys are implemented as separate key-sequenced files that "point" to primary file records via a field that contains the value of the primary key. Alternate key files and the primary files that they index can reside on separate disc volumes. Partitioning files, by primary-key value range, across multiple disc volumes (possibly on multiple nodes) is also supported.

[2]For more information on the Message System, see Mala Chandra's article, "The GUARDIAN Message System and How to Design for It," in the *Tandem Systems Review*, vol. 1, no. 1, February 1985.

One of the basic implementation components of ENCOMPASS is a process that acts as a server for files on a particular disc volume. This process, designated the *disc process*, is an example of an I/O process pair [2]. An I/O process pair is a mechanism that provides fault-tolerant systemwide access to I/O devices. It consists of two cooperating processes that run in the two processors physically connected to a particular I/O device.

One of these processes, designated the primary process, controls the I/O device, handling all requests to perform I/O on the device. The other process, designated the backup process, functions as a standby, ready to take over control of the device in case of failure of the primary path to the device. The processor in which the primary I/O process resides is an integral constituent of the primary path to the device. Should the primary's processor crash, the backup process must have information sufficient to take over control of the device. This critical information is sent from the primary process to the backup process during the course of normal processing in the form of so-called checkpoint messages.

The process pair that controls a disc volume is called the disc-process pair, or simply the disc process. Its primary and backup members run in the "primary" and "backup" processors for the disc volume, respectively. The disc process has an active rather than a passive backup process. The term *active backup process* refers to the fact that the information the backup process receives via checkpoint messages drives its execution control flow. This is in contrast to a possible alternative design in which the backup process passively receives copies of recently dirtied portions of the primary process' memory. The active backup concept is central to the design of single-fault tolerance, as described below. Figure 2 illustrates the concept.

From the point of view of a given disc process, a *file* is a single partition of an ENCOMPASS "file" (if, indeed, the latter is partitioned). Partitions of key-sequenced primary data files and of alternate key files look alike to the disc process: each is structured as a single B-tree. The higher-level concept of a file with partitions and/or alternate keys is implemented by the File System. The File System is a set of user-callable procedures that execute in the environment of the user process. These procedures (e.g., OPEN, READ, KEYPOSITION, LOCKREC, WRITE, etc.) accomplish an operation by sending one or more request messages to the appropriate disc process(es). In a requester-server model, the invoker of the File System is the requester and the disc processes are the servers.

The primary interface to the disc process is record-oriented, although a block-oriented interface is also provided. Most update requests result in the updating of a single record within a single block of a given file. In the case of key-sequenced files, however, the possibility that a single request message from the File System could cause a B-tree split or collapse means that the request may be executed as a *series of micro-update steps*. Since an incomplete series of micro-update steps leaves a file structurally inconsistent, robustness to crash requires a method of assuring its atomicity. This atomicity is provided for both audited and non-audited files, but the means differ, as explained later.

Figure 2

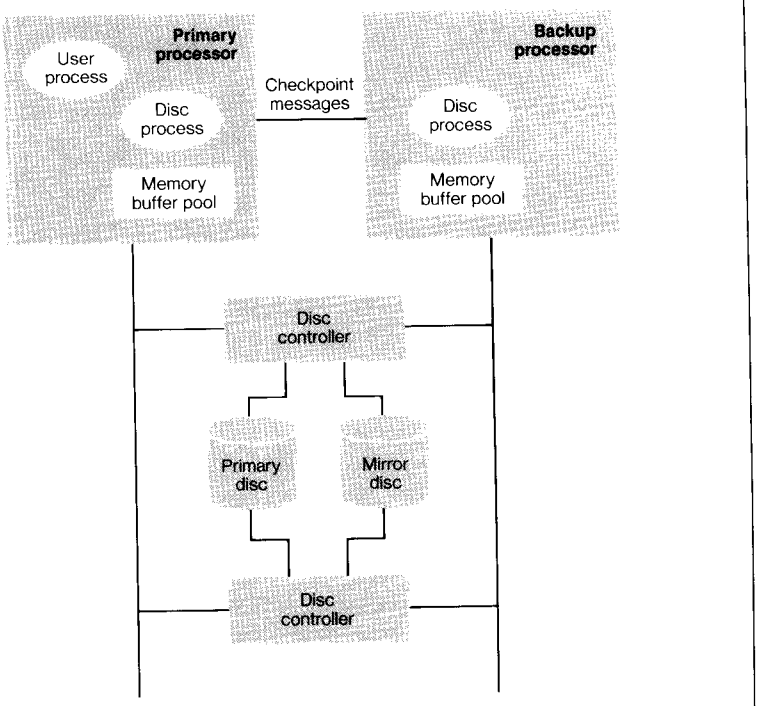


Figure 2.
The hardware configuration for a disc-process pair. In the primary and backup processors are the primary and backup disc processes for a mirrored disc volume. The primary disc process performs I/Os to move pages to and from its memory buffer pool. Reads go to the disc closest to the cylinder; writes go to both discs. The backup disc process maintains the backup buffer pool, based on checkpoint messages received from the primary disc process.

Failure Modes

The system architecture described supports fault tolerance for a variety of failure modes other than processor crash. Fault tolerance extends from failures of single hardware components (discs, I/O channels, I/O controllers) to failures of system or application software (programmatic processor halt, user process error, transaction abort). The current discussion, however, will be limited to failures that result in the loss to a single multiprocessor node of one or more of its constituent processors. *Loss* in this context means the invalidation of everything stored in the failed processor's memory. This could actually be caused by the failure of any hardware or software component associated with that processor.

The failure model supported can be characterized as *fail fast*. Consistency checks are an integral part of the system hardware and software. If such a check fails, the bad component is halted. This approach makes failures "clean" and makes it unlikely that a failed component would contaminate other components [2,5].

Definition of Robustness to Single-processor Failure

The failure of a single processor in the described environment results in the *takeover* of its functions by the remaining processors. In particular, the failure of a primary disc process' processor results in the takeover of its function by the backup disc process' processor. If the failed processor contained other primary disc processes with different backup processors, the failed processor might have its work taken over by several other processors.

The disc process is designed to provide robustness to single-processor failure. This robustness is implemented by means of (1) checkpoint messages sent from the primary process to the backup process during normal processing and (2) a takeover algorithm described later.

The following elements constitute robustness to single-processor failure:

1. "Sessions" between the disc process and requesters calling the File System survive the failure of the disc process' primary processor. Thus, any file open before takeover still appears open after the takeover.

   When updates are not protected by transaction auditing (i.e., updates to nonaudited files), a mechanism of tagging messages between the File System and the disc process with sequence numbers can optionally be used to guarantee that a request message is never lost during the takeover and that a nonidempotent operation is never duplicated [2].

   When updates are protected by transaction auditing (i.e., updates to audited files), the file-open session survives the takeover, but updates executed under that open by a given transaction survive the takeover if and only if that transaction committed before the takeover.

   The tolerance of sessions to single-processor failure obviates the need to perform system restart in the event of such a failure. For nonaudited files, the takeover is transparent to the caller of the File System. For audited files, the takeover is not transparent to the caller of the transaction management system (since transactions may be aborted), but higher-level software makes the abort and restart of such a transaction transparent to the end user [3].

2. The structural integrity of both audited and nonaudited files on the volume is guaranteed. Thus, if the primary's processor fails in the middle of performing a series of micro-update steps to a file, takeover processing restores the file's structure to a consistent state by backing out the steps performed before the failure.

3. The transactional consistency of the data base as a whole is guaranteed. Thus, if a transaction that was uncommitted at the time of takeover had updated audited files on the failed primary disc process' volume, takeover processing aborts the transaction and backs out its changes everywhere (on other volumes on this or other nodes). It should be noted that

transaction backout does *not* include undoing a *completed* B-tree index operation. In this sense, transaction backout is *logical* rather than *physical*.

Definition of Robustness to Disc-process-pair Crash

A *disc-process-pair crash* is defined as the simultaneous failure of both its primary and backup processors. The crash of a disc process pair and the failure of its primary and backup processors are viewed as equivalent because the disc process is an integral part of the operating system, and, as such, becomes operational whenever the the processor is restarted. Conversely, whenever a disc-process primary or backup process detects an internal consistency-check failure, it halts its processor in accordance with the fail-fast principle. While such a measure might be deemed Draconian in a conventional architecture, this aspect of the design is predicated on the principle that system availability is not compromised by the loss of a single processor.

The underlying assumption is that processors fail independently and that the primary and backup disc processes have independent failure modes. Of course, this assumption would be invalidated by the presence of a "hard" (i.e., nontiming-dependent) algorithmic bug present in code that would inevitably be executed by either member of the process pair. The elimination of such bugs has not proven to be an impractical goal, however. This might not be so were the primary and backup processors running in lockstep, or were the backup process passively receiving copies of recently dirtied portions of the primary process' memory.

When a disc-process pair crashes, the situation is similar to the state described earlier as the crash of a shared-memory system. Information stored in memory (in this case the memories of *both* primary and backup processors) is lost. Any method of recovery must resort to secondary storage. Furthermore, since "sessions" between the crashed disc-process pair and requesters calling the File System have been broken, there is the operational requirement of "restart." The analogy between the elements of robustness to single-processor failure and robustness to disc-process-pair crash is as follows:

1. "Sessions" between the disc process and requesters calling the File System do not survive the disc-process-pair crash.
2. The structural integrity of both audited and nonaudited files on the volume is guaranteed. Thus, if the disc-process pair crashes in the middle of performing a series of micro-update steps to a file, crash recovery restores the file's structure to a consistent state.
3. The transactional consistency of the data base as a whole is guaranteed. Thus, if a transaction that was uncommitted at the time of the disc-process-pair crash had updated audited files on the crashed disc-process pair's volume, crash recovery backs out that transaction's changes everywhere (on other volumes on this or other nodes).

   Conversely, if a transaction that was committed at the time of the disc-process-pair crash had updated audited files on the crashed disc-process pair's volume, but those updates were still in memory buffers (rather than reflected in the corresponding data-base pages on secondary storage) at the time of the crash, crash recovery retrieves those updates (from the log) and applies them to the data-base pages on secondary storage.

   As in the case of the single-processor failure, transaction backout does not undo *completed* B-tree index operations.

Evolution of the Disc-process Design

The above description reflects a rearchitecture of the disc process. The goals of the new design were to provide quick recovery from disc-process-pair crash and less costly tolerance of single-processor failure. The old disc process provided robustness to single-processor failure as described above. However the old implementation of single-processor-failure tolerance made a trade-off in favor of fast takeover recovery from single-processor failure at the expense of long recovery in the event of disc-process-pair crash.

The only method of recovery from disc-process-pair crash was the time-consuming technique of reloading previously archived copies of audited data-base files and "rolling forward" these files to a state of transactional consistency by the application of after-images from the audit trail. The duration of volume unavailability implied by this procedure was justified by the assumption that double-processor failure is rare. In fact, however, double failures are more common than would be predicted by consideration of hardware mean time between failures. Most processor failures are in fact caused by software bugs or operational errors.

Two characteristics of the original design dictated the "roll forward" approach to crash recovery and tolerated single-processor failure at the expense of extra disc I/Os and extra checkpoint messages during normal processing. These were as follows:

1. The decision to synchronously *write through* to disc all updated data-base pages rather than buffering them in memory.
2. The technique of *incremental checkpointing* (sending messages from primary to backup disc process during normal processing), which provided the backup process with the information needed in the event of the primary processor's failure to carry forward any interrupted series of micro-update steps and to continue forward processing on transactions active on the disc volume.

Write-through cache was originally conceived as a means of simplifying the implemention of single-processor-failure tolerance. However it made the write-ahead-log protocol [4] infeasible because unacceptable performance would result if every data-base update resulted in two writes: first, the before-image log necessary for undo in case of failure; second, the modified data-base page. The absence of write-ahead log made the fast crash-recovery technique of in-place rollback of crashed transactions impossible. Writing through every data-base update also had negative implications for throughput and response time. Rather than allowing the "piggy-backing" of several in-memory modifications on the same I/O, it meant that each time a page was "dirtied" in memory, it would be written out synchronously (while the application process waited).

Incremental checkpointing is necessary if the backup process is to be prepared (in the event of the primary processor's failure at any instant) to *carry forward* an interrupted series of micro-update steps or an interrupted transaction. In the rearchitected disc process, the approach to takeover is to provide the backup process with enough information to enable it to *back out* rather than to carry forward any interrupted series of micro-update steps, and to *abort* rather than to continue processing forward any transaction active on the disc volume. With this approach, *deferred checkpointing* is possible. According to this technique, the information that would have been sent as synchronous, incremental checkpoint messages in the old architecture is instead buffered on the primary process' side and sent as a batch at such times as transaction commit.

This technique reduces considerably the cost of single-processor-failure tolerance in terms of number of messages. In particular, it saves sending checkpoint messages that inform the backup process of memory-only changes in the primary's processor that will not reach secondary storage and will be backed out anyway in case of takeover. An example of such a change is a buffer dirtied in memory by a transaction that has not yet committed and for which the audit has not yet been forced. The backup process need have no knowledge of such a change since the transaction that caused it will be aborted and backed out (globally) in case the primary disc process' processor fails.

In explaining the *takeover algorithm* used by the new disc process to recover from single-processor failure, it is useful to draw an analogy between the use of log records by conventional crash-recovery algorithms [4] and the use of checkpoint records during takeover processing. Checkpointing for the new disc process is analogous to logging to the backup process. Audit and checkpoint records have a common format; for this reason, they are known as *audit/checkpoint records*. A typical audit/checkpoint record contains identification of the file, page number within file, record number within page, and the before and after content of the changed record. A version number of the

change is stored in both the page header and the audit/checkpoint record to provide idempotence during recovery.

Just as conventional log-based crash-recovery algorithms use the redo information in the log to bring the data-base pages up-to-date with the information that had been logged by the time the system crashed, so the takeover algorithm uses the redo information from checkpoint records received to bring the backup process' memory-buffer pool up-to-date with the information that had been checkpointed by the time the primary's processor failed. Similarly, just as conventional crash recovery proceeds to use logged undo information to back out incomplete requests and uncommitted transactions, so the takeover algorithm uses checkpointed undo information to back out any incomplete series of micro-update steps.

At this point, the takeover algorithm terminates and the former backup process begins operation as a primary process by accepting new request messages. Uncommitted transactions have not yet been recovered, however. Any partially completed series of micro-update steps belonging to an uncommitted transaction has been backed out, and the transaction is prevented from continuing forward processing, but at the completion of takeover such a transaction's updates have not yet been backed out. Locks needed for backout are still held, however.

Such a transaction will eventually be backed out by means of logically compensating disc-process request messages sent by the backout process, a system process that extracts the information needed for such requests from the log. The logically compensating operations requested by the backout process are made idempotent by tolerating a "record not found" condition when deleting a record (compensating for an insert) or a "duplicate key" condition when inserting a record (compensating for a delete). Compensating update operations are automatically idempotent.

Crash Recovery for the Rearchitected Disc Process

The new disc process uses separate mechanisms to provide robustness to disc-process-pair crash for nonaudited and audited files. As previously stated, robustness to crash for nonaudited files implies the restoration of structural integrity. For audited files, on the other hand, it implies not only the restoration of structural integrity to individual audited files, but, in addition, the guarantee of transactional consistency for the data base as a whole.

In the case of nonaudited files, updates are not protected by transaction auditing. However, loss of structural integrity due to a series of micro-update steps being interrupted by disc-process-pair crash is prevented by use of the so-called *undo area* on the disc volume. This is a small preallocated area on the volume that is reuseable for every request. Before a series of micro-update steps on a nonaudited file (e.g., B-tree block split) is begun, a highly compacted encoding of the intended steps is written to the undo area using one I/O. Then if the disc-process pair crashes before the operation completes, this undo information is used to back out the incomplete operation when the volume's processors are restarted.

The algorithm used to recover audited files from disc-process-pair crash is summarized below. It is analogous to typical data-base crash-recovery algorithms used for conventional architectures [4].

Following disc-process-pair crash, users first restart the volume's primary and backup processors. They then initiate the autorollback process. Autorollback obtains a list of those audited files on the crashed volume that were open for write access at crash time. These are the files that are recovered from the log. Log processing during crash recovery consists of a forward and a backward pass.

The forward pass begins at the *redo start point*. This is a location in the log before which all logged updates (redo images) are guaranteed to be reflected in the data base. Existence of such a point within a short distance of the end of the log is guaranteed by the periodic execution by each volume's disc process of *control points*. At each control point, currently dirty buffers are flagged. During any spare time between control points, flagged buffers are written out. At the occurrence of the next control point, any flagged buffers not yet written are forced out and newly dirtied buffers are flagged. (Other systems term this mechanism a *checkpoint*; see [6]). The locations in the log

of the latest two control-point records are remembered at a known place on the disc volume.

When recovering a given crashed disc volume, autorollback finds that volume's redo start point by obtaining the pointer to its next-to-last control point. When recovering a set of crashed volumes, autorollback starts its forward pass of the log at the earliest redo start point for any of the crashed volumes. Autorollback then sends to the disc process of a crashed volume all redo log records it finds from that volume's redo start point through the end of the log.

After the redo phase, the backward pass begins. Reading the log backwards from the end, autorollback sends to the appropriate disc process those undo log records that represent incomplete series of micro-update steps. When all of the changes represented by these log records have been physically undone, all audited files open on the crashed volume(s) will have been restored to a state of structural integrity.

During the same backward pass, autorollback sends to the appropriate disc process those undo log records that represent logical operations on data blocks (e.g., record insert, modify, or delete) that were executed by transactions uncommitted at crash time. When all of the changes represented by these log records have been logically backed out (i.e., using compensating operations at disc-process-request level), global transactional integrity will have been achieved.

Conclusion

The concepts of *crash* and *crash recovery* have been seen to require generalization in order to find applicability to a nonshared-memory multiprocessor architecture, in which some processors may survive the crash of other processors in the system. The architecture of the Tandem computer system was described as a case in point. A technique of logging to another processor's memory that tolerates single-processor failure and obviates the need to perform system restart was described. An analogy was drawn between the technique used in a Tandem system to recover from a single-processor failure and conventional crash-recovery techniques that rely on a secondary-storage-resident log.

References

1. Bartlett, J. F. 1978. A "NonStop" Operating System. In *Proceedings of the Eleventh Hawaii International Conference on System Sciences.*
2. _____. 1981. A NonStop Kernel. In *Proceedings of the Eighth Symposium on Operating System Principles,* ACM.
3. Borr, A. J. 1981. Transaction Monitoring in ENCOMPASS: Reliable Distributed Transaction Processing. In *Proceedings of the Seventh International Conference on Very Large Data Bases*, September 1981. Republished as Tandem TR 81.2.
4. Gray, J. N. 1978. Notes on Data-base Operating Systems. IBM Research Report, RJ 2188.
5. _____. 1981. The Transaction Concept: Virtues and Limitations. In *Proceedings of the Seventh International Conference on Very Large Data Bases*, September 1981. Republished as Tandem TR 81.1.
6. Gray, J. N., et al. 1979. The Recovery Manager of a Data Management System. IBM Research Report, RJ 2623.
7. Katzman, J. A. 1978. A Fault-tolerant Computing System. In *Proceedings of the Eleventh Hawaii International Conference on System Sciences.*
8. Lampson, B., and Sturgis, H. E. 1976. Crash Recovery in a Distributed Data Storage System. Xerox Palo Alto Research Center. Also appears in *Lecture Notes in Computer Science: Distributed Systems — Architecture and Implementation*. ed. B. W. Lampson. vol. 105. 1981. Springer-Verlag.
9. Menasce, D. A., and Landes, O. E. 1980. On the Design of a Reliable Storage Component for Distributed Data-base Management Systems. In *Proceedings of the Sixth International Conference on Very Large Data Bases*, October 1980.
10. Verhofstad, J. S. M. 1978. Recovery Techniques for Database Systems. *Computer Surveys*. vol. 10, no. 2.

Acknowledgments

Many of the ideas incorporated into the new disc-process design were originated by Franco Putzolu. Thanks are due to Jim Gray, Chris Duke, and John Nauman for editorial suggestions whose implementation improved the presentation of this material.

Andrea Borr has worked on the design and development of the ENCOMPASS products ENFORM and TMF, and on the new GUARDIAN 90 disc process, DP2, since she joined Tandem in 1978. She is currently engaged in a project associated with DP2. Before joining Tandem, she spent 8½ years as a software developer and field analyst for two other mainframe vendors. Andrea holds a Bachelor's Degree in Mathematics from the University of Chicago and a Master's Degree in Computer Science from the University of Wisconsin. She was also a doctoral candidate at Stanford University.

TANDEM PUBLICATIONS ORDER FORM

The *Tandem Systems Review* and the Tandem Application Monograph Series are combined in one subscription. Use this form to subscribe, change a subscription, and order back copies.

For requests *within the U.S.*, send this form to:

Tandem Computers Incorporated
Sales Administration
19191 Vallco Parkway
Cupertino, CA 95014-2599

For requests *outside the U.S.*, send this form to your local Tandem sales office.

Check the appropriate box(es):

- ☐ New subscription (# of copies desired ___ ___)
- ☐ Subscription change (# of copies desired ___ ___)
- ☐ Request for back copies. (Shipment subject to availability.)

Print your current address here:

_____
ADDRESS

_____

_____

_____

_____

_____
ATTENTION

_____
PHONE NUMBER (U.S.)

If your address has changed, print the old one here:

_____
ADDRESS

_____

_____

_____

_____

_____
ATTENTION

_____
PHONE NUMBER (U.S.)

To order back copies, write the number of copies next to the title(s) below.

NUMBER OF COPIES

Tandem Journal

_____ Part No. 83930, Vol. 1, No. 1, Fall 1983

_____ Part No. 83931, Vol. 2, No. 1, Winter 1984

_____ Part No. 83932, Vol. 2, No. 2, Spring 1984

_____ Part No. 83933, Vol. 2, No. 3, Summer 1984

Tandem Systems Review

_____ Part No. 83934, Vol. 1, No. 1, February 1985

_____ Part No. 83935, Vol. 1, No. 2, June 1985

Tandem Application Monograph Series

_____ Part No. 83907, *Designing a Network-Based Transaction-Processing System,* April 1982, SEDS-002

_____ Part No. 83900, *Developing TMF-Protected Application Software,* March 1983, AM-005

_____ Part No. 83901, *Designing a Tandem/Word Processor Interface,* March 1983, AM-006

_____ Part No. 83902, *Integrating Corporate Information Systems: The Intelligent-Network Strategy,* March 1983, AM-007

_____ Part No. 83903, *Application Data Base Design in a Tandem Environment,* August 1983

_____ Part No. 83904, *Capacity Planning for Tandem Computer Systems,* October 1984

TANDEM EMPLOYEES: PLEASE ORDER YOUR COPIES THROUGH YOUR MARKETING LITERATURE COORDINATOR.

06/85

TANDEMCOMPUTERS

Part No. 83935

400096 06/85 Printed in USA