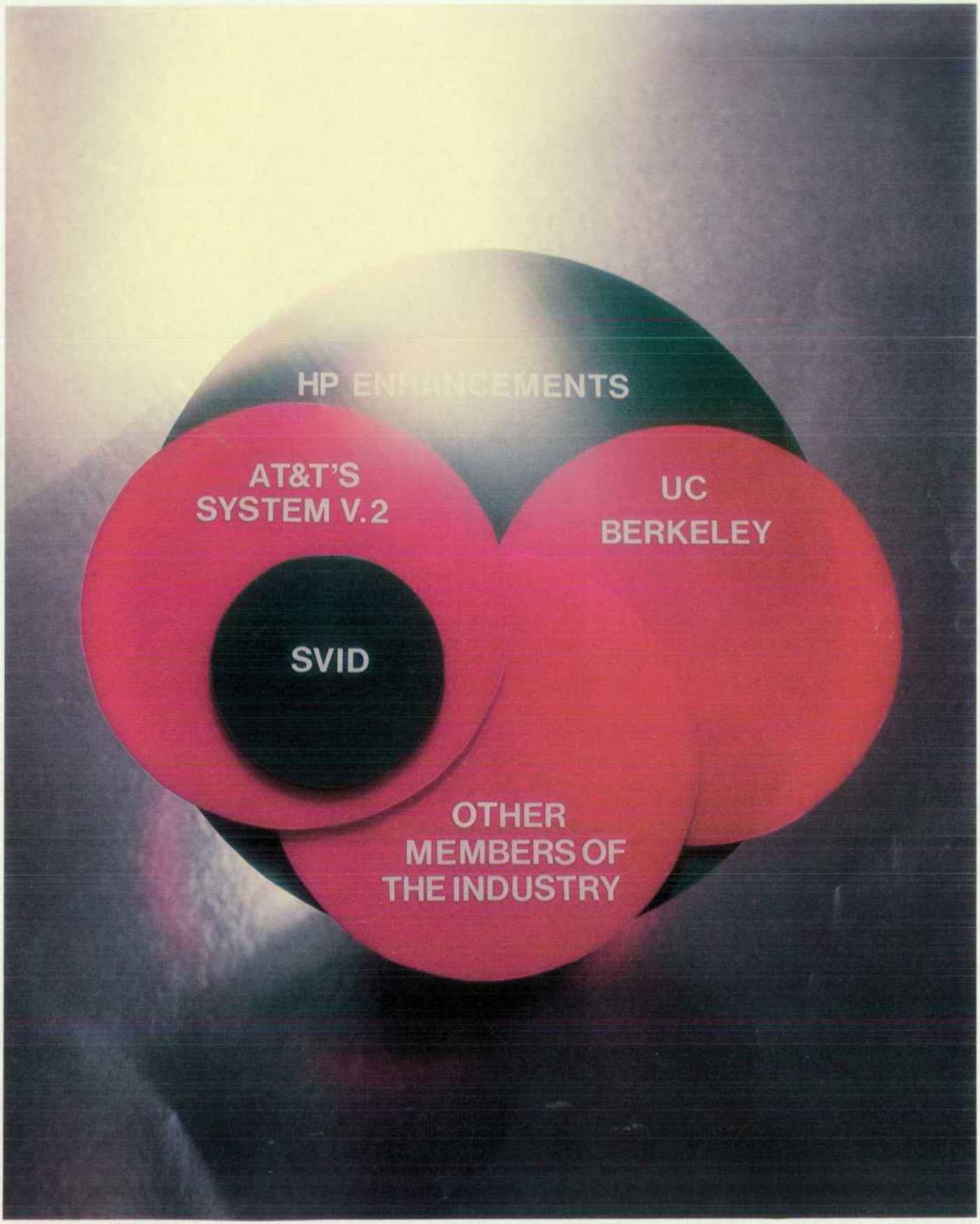
HEWLETT·PACKARD
JOURNAL

DECEMBER 1986



HEWLETT-PACKARD JOURNAL

December 1986 Volume 37 • Number 12

Articles

4 **The HP-UX Operating System on HP Precision Architecture Computers,** *by Frederick W. Clegg, Gary Shiu-Fan Ho, Steven R. Kusmer, and John R. Sontag* *HP's version of AT&T's UNIX System V.2 operating system has the extensions needed for effective support of CIM and CAE applications.*

9 **A UNIX System V Compatible Implementation of 4.2BSD Job Control**

13 **Decreasing Real-Time Process Dispatch Latency Through Kernel Preemption**

33 **Data Base Management for HP Precision Architecture Computers,** *by Alan S. Brown, Thomas M. Hirata, Ann M. Koehler, Krishnan Vishwanath, Jenny Ng, Michael J. Pechulis, Mark A. Sikes, David E. Singleton, and Judson E. Veazey* *ALLBASE provides both the network method of data access and the relational method.*

46 **Data Storage in ALLBASE**

Departments

3 **In this Issue**

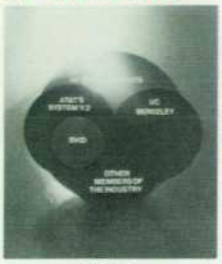
3 **What's Ahead**

23 **1986 Index**

30 **Authors**

32 **Reader Forum**

In this Issue



This issue continues the series of papers on major hardware and software components of HP's new generation of computers, products of the HP Precision Architecture development program. Earlier issues have presented the motivation and framework for the program (August 1985), the optimizing compiler technology (January 1986), and the processor and input/output architecture together with the measurement and simulation methodologies which guided their development (August 1986).

Although the low-level processor and I/O architecture have been the focus of much attention, most users will see the system through higher-level interfaces. This issue presents two such facilities: the HP-UX operating system and the ALLBASE data base management subsystem. Each represents a very substantial software development effort, and each has been designed to make maximal use of the speed, large address space, and I/O capabilities provided by HP Precision Architecture.

The first paper (page 4) presents the implementation of HP-UX, a real-time extension of AT&T's UNIX System V.2 operating system for the HP 9000 Series 800 Model 840 processor. The ways in which the implementation exploits the capabilities of HP Precision Architecture are described, with particular attention given to real-time extensions, memory mapping, and the I/O subsystem.

The second paper (page 33) describes the multilevel implementation of ALLBASE, a new data base management system that fully supports both a relational access model and the more traditional network model of data access. ALLBASE presents the same interface and supports the same data representations on both HP-UX and MPE XL operating systems.

Future papers in this series will treat hardware realizations of the architecture, the MPE XL operating system, and other topics.

Michael J. Mahon
Manager, Computer Language Laboratory
Guest Editor

Cover

Origins of the HP-UX operating system (Fig. 3, page 5) sculpted in plastic.

What's Ahead

The January issue begins with four articles about the HP 3562A, a low-frequency dynamic signal analyzer with built-in curve-fitting and synthesis features. The articles discuss the design of the HP 3562A and its measurement modes, curve fitter, and synthesis capability.

Concluding the issue is an article detailing how performance measurements pointed to the design changes that provide the increased performance of the HP 3000 Series 70 Business Computer.

The HP Journal encourages technical discussion of the topics presented in recent articles and will publish letters expected to be of interest to our readers. Letters must be brief and are subject to editing. Letters should be addressed to: Editor, Hewlett-Packard Journal, 3200 Hillview Avenue, Palo Alto, CA 94304, U.S.A.

The HP-UX Operating System on HP Precision Architecture Computers

HP-UX is the technical operating system for HP Precision Architecture processors. It's an extension of AT&T's UNIX System V.2.

by Frederick W. Clegg, Gary Shiu-Fan Ho, Steven R. Kusmer, and John R. Sontag

HP-UX IS HEWLETT-PACKARD'S standard version of AT&T's UNIX* System V operating system.[1-6] It is currently supported on the HP 9000 family of computers, including its most recent addition, the Series 800 Model 840. HP-UX is one of the two operating systems offered on the HP Precision Architecture family of processors. The HP-UX implementation on the Model 840 provides all of the functionality needed for full support of both computer integrated manufacturing (CIM) and design automation (CAD/CAE).

Earlier HP Journal articles have covered implementations of HP-UX on HP 9000 Series 300 and 500 Computers[7,8] and on the HP Integral Personal Computer.[9,10,11] Comparison of those articles with this one will reveal a strong similarity between those implementations of HP-UX and the HP Precision Architecture implementation described in this article. This is not by accident. HP's corporate strategy calls for the HP-UX operating system to appear the same from the user's point of view, no matter what underlying architecture is used.[12,13] Therefore, after a quick summary of HP-UX, this article will stress the contributions to HP-UX made by the HP Information Technology Group project teams responsible for implementing HP-UX on HP Precision Architecture. These contributions include kernel preemption, job control, native language support, and real-time enhancements.

Hewlett-Packard has chosen to support a UNIX operating system because it has several assets:

- Existing standards and a means to standardize further
- Productive software development environment
- Portability of software
- Easy access to existing applications software
- Hardware and vendor independence
- Multivendor networking
- Ability to run on micros, minis, and mainframes.

A UNIX operating system can offer these benefits because it is flexible and powerful. It is constructed of a collection of tool-like programs, each of which performs a general-purpose task. These programs can be combined in various ways so that myriad complicated tasks can easily be accomplished. In addition, new programs can be added without affecting the existing operating system.

One of the more powerful features in this system is that the output from one program can become the input for another without the user's creating an intervening temporary file. This allows speedy, powerful commands to be constructed without the unwanted overhead necessary in other operating systems.

Three major parts make up the core of a UNIX operating system (see Fig. 1): the kernel, which controls the resources of the computer's hardware, the file system, which is the means for organizing the layout of data storage, and the shell, which is the command interpreter.

A large number of tools are also available in a UNIX operating system, including programming languages, text processors, and many more. A traditional UNIX operating system is interactive, which means that the user's input is immediately responded to by the system. In addition, a UNIX system is multitasking, so the user can instruct the computer to run background tasks while continuing to work interactively. It is also multiuser in that many people can use it at once.

The features of a UNIX operating system are traditionally explained in a large reference manual, which documents each core program, tool, and interface. This large tome is organized much like a dictionary and is available on-line for quick reference. It is divided into sections by type of

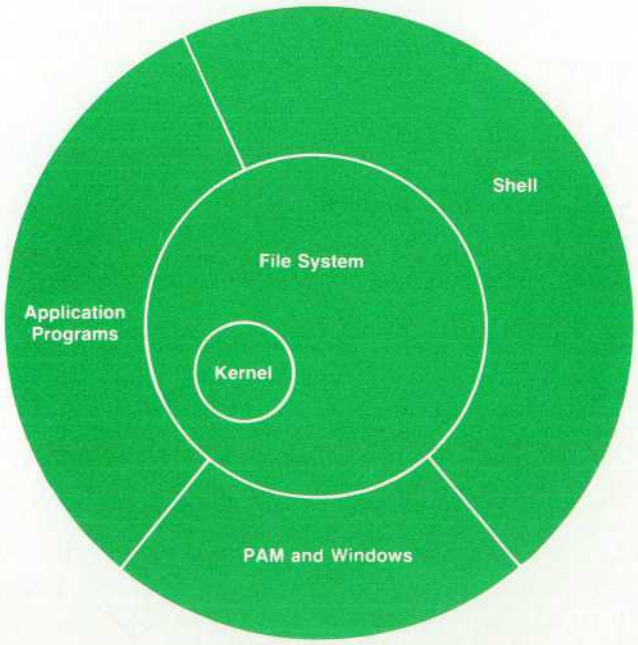


Fig. 1. *Organization of the HP-UX operating system.*

*UNIX and System V are registered trademarks of AT&T Bell Laboratories in the U.S.A. and other countries.

program. For example, all utilities are in section 1 and all C libraries are in section 3c.

History of HP-UX

The shape and contents of the HP-UX system result primarily from three historical forces:

- Development and standardization efforts by AT&T
- Enhancements added by the University of California at Berkeley
- Development and standardization efforts by HP.

The UNIX operating system was developed at Bell Laboratories in 1969 by programmers who wanted a simple, flexible, and powerful environment in which to write their programs. One of their first projects was to develop text processing and formatting tools for internal documentation needs at AT&T. Because of its power, flexibility, and ease of use, the UNIX operating system quickly found widespread acceptance by computer scientists throughout the Bell Labs organization. In 1975 the first public version, Version 6, was released to universities. Divergence began as the university students and professors added features, while the AT&T engineers continued on their own path. By Version 7 in 1978 the portability of the system had been improved with the rewriting of most of the operating system in the high-level language C. In 1979 Berkeley's 3.0BSD (for "third Berkeley software distribution") added virtual memory. Meanwhile, HP purchased source code from AT&T in 1980 and began to enhance the operating system as well.

In 1981 AT&T released System III, which incorporated many Version 7 features. Between 1980 and 1981, Berkeley added job control, tuning, long variable names, and different hardware support. Because they had established themselves as a leader in development, Berkeley received a grant from the U.S. Department of Defense Advanced Research Projects Agency (DARPA), to provide support for networking. By the next release in 1983, 4.2BSD, Berkeley had also added a faster file system and a means of interprocess communication.

In 1983 AT&T released System V, which improved performance and added semaphores and shared memory. By System V, Release 2, Issue 2 in 1985, AT&T had added shell layers (permitting a single user to work on multiple jobs simultaneously), flexnames (accommodating longer names for identifiers), and virtual memory.

To prevent various versions of the operating system from diverging any further, AT&T published the *System V Interface Definition, Issue 1*, (called the SVID) in 1985. The SVID defines the *de facto* industry standard interfaces for System V, so that portability between similarly compliant UNIX operating systems is possible. The interfaces are defined individually in the SVID, much as they are in the traditional UNIX operating system reference manual. The input needed and resultant output are explained, but the method of implementation is not. Therefore, each computer is free to implement the interfaces in its own way, yet it can still comply with the SVID and be compatible with other UNIX operating system implementations.

While the SVID is the most important standard in the UNIX operating system community to date, standardization efforts have been spearheaded by the IEEE, /usr/group (an international UNIX users group), X/OPEN (a consortium of European and U.S. manufacturers), and others. HP has participated heavily in these efforts and continues to be a leader.

HP-UX Defined

HP-UX complies with the SVID, Issue 1, and is a superset of it (see Fig. 2). In addition to SVID features, HP-UX includes some features from Berkeley versions and other versions in the industry.

For example, virtual memory management and local-area

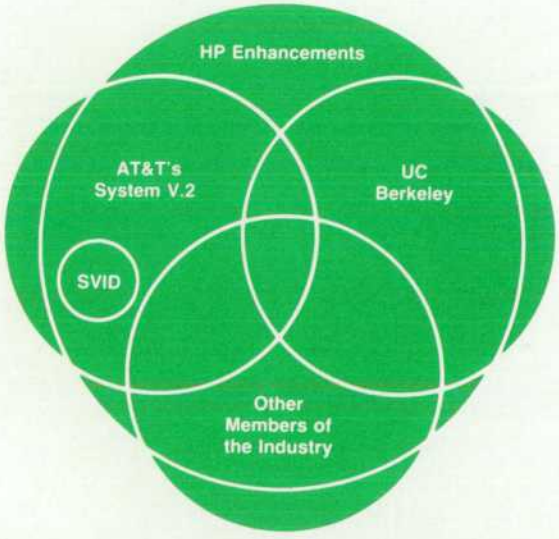


Fig. 2. *Origins of the HP-UX operating system.*

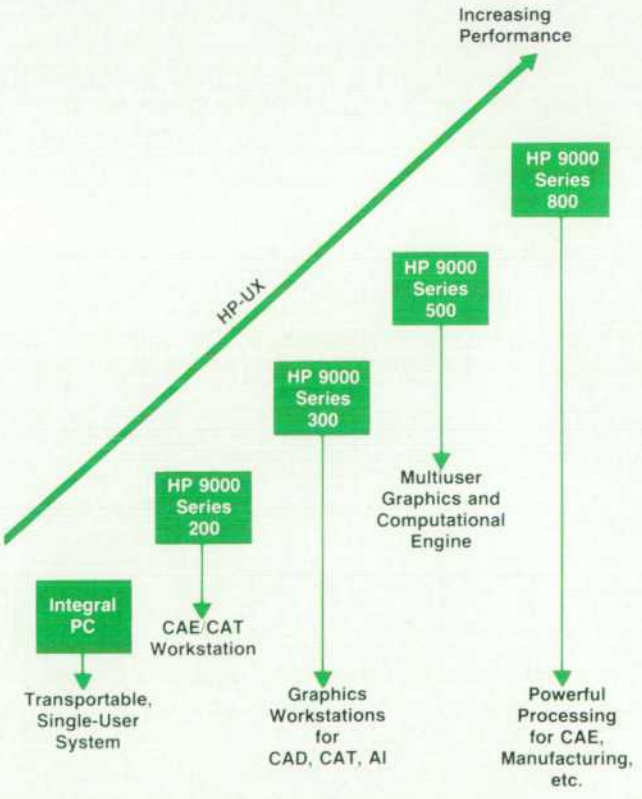


Fig. 3. *The HP-UX computer family and its applications.*

networking have been derived from the Berkeley 4.2BSD implementation. In other areas, such as those of real-time support and native language support (internationalization), new capabilities have originated at HP. HP is actively working with other UNIX system vendors and with standards organizations such as IEEE and X/OPEN toward the goal of having HP extensions to UNIX operating system capabilities accepted as industry standards.

Recognizing the importance of UNIX operating systems as an emerging standard, HP management convened the first meeting of a UNIX operating system working group in May of 1981. This working group, composed of representatives of all HP entities working on or contemplating a product based on a UNIX operating system, was formed to facilitate communication between implementors in different locations in an effort to minimize differences at the user application interface level between their respective implementations.

The working group identified three fundamental objectives that have guided its efforts since its inception:

- Effortless application and user migration from the latest AT&T UNIX system and easy migration from other popular UNIX system environments
- Effortless application and user migration between any HP-UX systems
- Added value without impairing ease of migration.

"Effortless application migration" means that code can be recompiled and run without change on a new system. This level of compatibility is commonly referred to as source code compatibility. This encourages importation of applications from other vendors' machines, serves to preserve the valued investments of our customers, and facilitates the leverage of HP's development efforts across HP-UX systems. Object code compatibility is supported across systems of the same architecture. For example, code for the Model 840 will run on other HP-UX members of the HP Precision Architecture family without recompilation.

The UNIX operating system working group evolved into the HP-UX Steering Council and created several subcommittees, which include the management council, technical working group, marketing working group, documentation working group, and support working group. The councils and working groups strive to develop a consistent approach to all aspects of the product, as well as a common operating system interface across multiple product families and corporate divisions.

Compliance with the HP-UX Standard

The current focal point of the standardization effort at HP is the two-volume *HP-UX Standard Specification, Version B.1*, published in January 1986. The standard, as prescribed by this specification, closely follows the SVID and documents all features in HP-UX. The reference manuals customers receive with their computers beginning in late 1986 will be derived directly from the standard specification.

Hewlett-Packard computing products that wish to offer an operating system derived from, or similar to, AT&T's UNIX system must comply with this standard. Waivers for any deviation from the standard require approval by the HP-UX management council before the deviating product can be placed on the corporate price list.

Compliance with the HP-UX standard is verified by running the HP-UX verification test suite on a system. This is a highly automated, extensive collection of "black box" tests that systematically verify that the behavior of each operating system service and library call matches that called for by the entries in the standard. The HP-UX verification suite is the product of over twenty engineering years of effort in various HP organizations over the past five years to produce the yardsticks necessary to implement a solid standard.

The HP-UX verification suite was recently complemented by HP's purchase of the AT&T System V verification suite (SVVS). The HP-UX standard and the HP-UX verification suite have, of course, been carefully designed to be compatible. Availability of the SVVS is an important milestone, nonetheless, since it provides for an additional guarantee of SVID compatibility. HP intends to have all HP-UX products pass the SVVS. After January 1987, UNIX system implementations passing the SVVS will be eligible for certification by AT&T that the implementation is SVID compatible.

One other factor helps ensure that HP-UX implementations on various HP computer products are compatible and conform to the HP-UX standard: most of the source code for these implementations at the commands and libraries level is shared across all implementations. A single shared source repository is accessed by the engineering staff working on all HP-UX products. To ensure that updates to this software are properly coordinated, they are supervised by a shared source administrator. Before a module may be modified by a member of any engineering team, that module must be checked out from the shared source data base. No one else may access that module for the purpose of changing it while it is checked out. Needless to say, this approach not only helps keep different HP-UX products compatible, but also affords HP tremendous engineering leverage whereby the refinements of a single engineer on

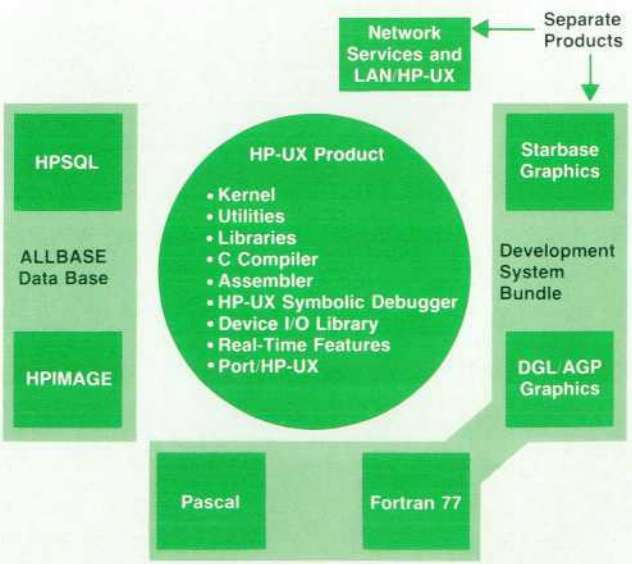


Fig. 4. *Software products available for HP-UX systems.*

one product team can be quickly shared across the entire HP-UX product line.

Operating System Enhancements

Because the original UNIX operating system was written by programmers for programmers, certain biases are apparent in it. Most of them are of benefit, but some are perceived as drawbacks in certain circumstances. Early UNIX operating systems were designed for general-purpose, timeshared applications in an English-speaking environment. New needs and new markets have developed and HP-UX has had to address them. These have been accommodated by adding enhancements to HP-UX with the philosophy of adopting existing industry standard interfaces (e.g., SVID, System V.2, Berkeley 4.2BSD, /usr/group, and IEEE P1003) whenever possible.

One apparent drawback of traditional UNIX operating systems has been the difficulty in learning them. The programmers who devised the user interface were interested in keeping their typing to a minimum and so created extremely terse and cryptic command names. For example, grep stands for global regular expression printer, which probably would not mean much to a novice. HP-UX has enhanced the operating system by adding interfaces that are menu-driven and provide for windowing on the screen.

HP-UX on the HP 9000 Model 840 is implemented directly atop the new HP Precision Architecture hardware, that is, it is a native mode implementation, not a layered implementation. It combines the features necessary to support the design automation and computer integrated manufacturing markets, but shares most features with the other members of the HP-UX family (see Fig. 3), which run on widely differing architectures.[14] Examples of HP-UX family features (see Fig. 4) include support for graphics, local area networking, and native language support (internationalization).

Native Language Support

Native language support (NLS) is an area in which HP has taken a leadership position in expanding the capability of UNIX operating systems (see Fig. 5). More than 50% of HP's sales are to international customers. In addition, the international community is demanding that computer systems have the capability to interact with users in their respective native languages. Much of the support for this capability lies within the operating system. HP's engineers began investigating how to extend the UNIX operating system to make this possible over four years ago.[15,16,17] The objective was to provide a satisfactory base for end users who want their application programs to run in languages other than "USASCII." Recently, the European standards group X/OPEN adopted HP's technology as their standard for NLS interfaces.

The User Interface

The user interface to any UNIX system is the shell. This is essentially a command interpreter program that accepts user inputs and turns them into requests to the underlying kernel, libraries, and other utilities to accomplish desired actions. Additionally, the shell performs housekeeping functions such as keeping track of the current working directory for a given user within the file system and other environment variables. A fundamental distinguishing characteristic of UNIX operating systems is that the shell is just another user program, that is, it is not embedded in the kernel, which is the heart of the operating system. A reasonably sophisticated user not happy with any of the shells provided can write a custom shell and use that instead of the one provided by the system's manufacturer.

Currently, HP 9000 systems (including the Model 840) are shipped with two shells. The more traditional Bourne shell, sh, is a direct descendant of the shells used with the earliest UNIX systems at Bell Laboratories. The C shell, csh, was developed at Berkeley and adds such capabilities as command substitution based on a stack of previously executed commands, a pushdown stack of file system directories currently being used, and commands to move multiple simultaneous jobs between background and foreground modes in conjunction with the supported Berkeley job control functionality (see box, "A System V Compatible Implementation of 4.2BSD Job Control," page 9). Csh as currently supplied by HP has been enhanced with the addition of features (in the past found in the public domain program tcsh) originally found in Digital Equipment Corp.'s TENEX operating system, including command and file name completion. With this capability, a user can type only as many leading characters of a command or file name as needed to distinguish that command or file from the other valid ones in that context and then strike the **ESC** key to request the shell to fill in the missing keystrokes. Also adopted from tcsh is an autologout feature that automatically logs users out of the system after they have been inactive for a specified period of time (as when a programmer goes home for the night having forgotten to log out).

Other shells are available, as well. For example, the Korn shell, ksh, recently released by AT&T is available to those users who purchase a license for its use from AT&T.

The ability to use arbitrary shells is one of the features contributing to the high degree of flexibility for which UNIX systems are so highly praised. As an example, in

Supported Language	Language Set	Comments
American English	ASCII	Supported by HP terminals, printers, and plotters
Canadian English	Roman8	
Danish	Roman8	
Dutch	Roman8	
English (U.K.)	Roman8	
Finnish	Roman8	
French	Roman8	
German	Roman8	
Italian	Roman8	
Norwegian	Roman8	
Portuguese	Roman8	
Spanish	Roman8	
Swedish	Roman8	
Greek	Greek8	Supported by HP terminal
Turkish	Turkish8	
Japanese	Japan15	Katakana is supported by HP printers and plotters. Katakana, Kanji, and Hiragana are supported by a special terminal available from Yokogawa-Hewlett-Packard in Japan

Fig. 5. *Languages supported on HP-UX systems.*

1983, HP R&D engineers implementing HP-UX on a new processor wrote a simple shell to mimic the command interpreter of HP's MPE V operating system on the HP 3000. Another special shell, rtesh, is part of the software migration aids package discussed later in this article. It provides a user command interpreter that looks and acts like that of the RTE operating system for HP 1000 Computers.

In addition to serving as the interpreter of user commands entered interactively, most UNIX operating system shells have the capability of executing scripts, which consist of files of commands. Shell scripts, in turn, can contain conditional execution and looping constructs, permitting the user to write entire simple programs without the need to resort to C, BASIC, Pascal, or other more traditional programming languages. The C shell, csh, purportedly derives its name from efforts to give its programming constructs a syntax as consistent as possible with the C language.

The Program Development Environment

Traditional UNIX systems are generally acknowledged to be the most productive software development environment available on a widespread basis. Virtually every aspect of a software engineer's job is aided by one or more tools of the system.

Ex, vi, and other editors (full-screen and otherwise), as well as numerous other text manipulation tools (grep, awk, sed—the list is very long), speed the entry of everything from design proposals to source code to maintenance documentation. Many of these tools "know" about the structure of specific programming languages so that a single keystroke can be used to move the cursor to the beginning of the next block, for example.

Source code preprocessors such as lint, cpp, cxref, and cb are available to provide early screening of errors and to improve program readability. Compilers for C, Fortran, and Pascal are currently available on HP-UX products. Lisp, Ada, and BASIC will be available soon from HP. Third-party vendors offer a wealth of other languages, including Forth and COBOL, for HP-UX systems.

For dealing with larger software systems, tools such as make are available to ensure that the proper version of each component is employed in the generation of a system. SCCS (the source code control system) and other utilities from AT&T's *Programmer's Workbench* are available to facilitate administration of multiple versions of a given software system. For engineers building software that must recognize and/or translate commands or some other formal language input, HP-UX includes lex (a lexical analyzer) and yacc (yet another compiler compiler). For getting new programs to work well at run time, two very powerful debuggers are provided. The traditional adb (assembler-level debugger) has been carefully tailored to HP Precision Architecture on the Series 800 (a major challenge on a RISC architecture). Higher-level (source-level) debugging is facilitated by xdb, the latest in a lineage of symbolic debuggers including sdb and cdb in earlier UNIX systems. Xdb has been generalized to support Fortran and Pascal, as well as C programs. It is sufficiently powerful to tackle even such difficult jobs as analyzing a memory dump of the kernel after a crash. Finally, HP-UX offers tools such as prof and vmstat to aid in monitoring the run-time behavior of a program and in performance tuning.

Real-Time Commands and Libraries

The implementation of HP-UX on HP Precision Architecture is noteworthy in its extensions and alterations that make it suitable for real-time applications.[18] Most of the work to accomplish this was done within the kernel of the operating system. Command and library support for the real-time feature set consists of the datalock routine and the getprivgroup, setprivgroup, prealloc, and rtprio commands.

Real-time privileges are the capabilities given to processes to become real-time processes and/or lock themselves in memory. These privileges give users direct control over scarce system resources and must be controlled carefully. The approach chosen by HP-UX is an extension of the Berkeley access groups concept. Each real-time privilege is controlled by one or more real-time access groups, which are set up by the superuser. A process is given a real-time privilege only if it is a member of the corresponding real-time access group. This approach allows dynamic revocation of real-time privileges by the superuser by redefining the real-time access groups. A user process can also remove a real-time privilege from its child process by deleting the corresponding real-time access group from its child process before it passes control to the child process.

Process Management

Process Scheduling. A typical UNIX system scheduling algorithm was designed to provide equitable access to the CPU and memory in a timeshared environment. Process scheduling priority is recalculated periodically and process execution is time-sliced to ensure a fair share of the CPU for each process.

The concept of real-time priorities has been added to HP-UX to satisfy the real-time needs of the computer integrated manufacturing market. All real-time priorities are

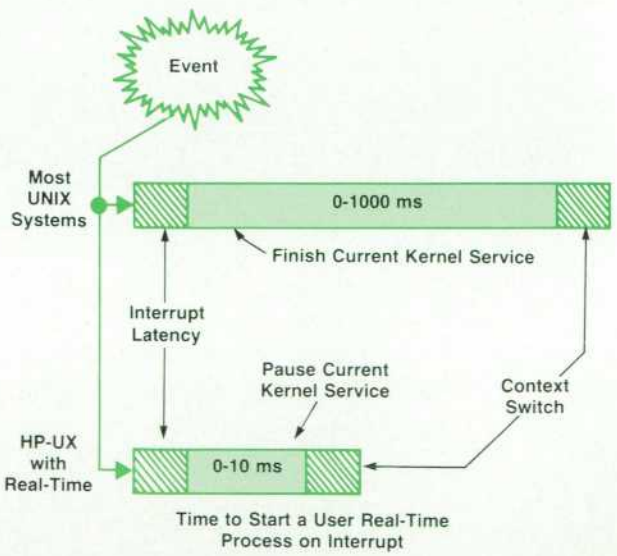


Fig. 6. *HP-UX real-time extensions achieve an order of magnitude improvement in preemption latency over traditional UNIX systems.*

of higher scheduling priorities and are not modified by the operating system. A process with the appropriate real-time privilege can set its own real-time priority and the real-time priorities of its offspring. In the extreme case, the process with the highest real-time priority can monopolize the processor and have full control of the system. Real-time processes with the same priority are executed under a round-robin, time-sliced policy.

Time-Based Scheduling. Processes can be scheduled by firing an alarm clock in traditional UNIX systems. UNIX System V supports an alarm clock with a resolution of one second. That is sufficient for most timeshared applications but is too coarse for time-critical applications. HP-UX has adopted the Berkeley 4.2BSD timer interface, which supports microsecond resolution of three types: real time (wall clock time), virtual time (time spent in the user code) and prof time (time spent in both the kernel and user code). The timer is nondrifting in the sense that a process can schedule itself for execution at regular intervals independently of the dispatching and rescheduling overhead of the process.

Preemption Latency. In a traditional UNIX system, a process executing in user code can be preempted immediately. However, when executing in the kernel, the process gives up the CPU only voluntarily and explicitly, for example

A UNIX System V Compatible Implementation of 4.2BSD Job Control

The job control functionality first introduced into UNIX operating systems by Jim Kulp of IIASA and later provided by the 4.2BSD version has become a *de facto* industry standard.[1,2] It allows the user to control multiple simultaneous tasks from a single terminal easily. However, this job control facility, as implemented in 4.2BSD, is incompatible in several respects with System V. This is typical of many cases where new features desired by customers must be carefully engineered to fit comfortably into HP-UX without violating industry standards for compatibility.

4.2BSD job control allows users to stop (suspend) the execution of processes and continue (resume) their execution at any later point. This only works easily for processes that are stopped and continued during the same login session.

The user almost always employs this facility via the interactive interface jointly supplied by the system tty driver and a job control shell such as csh. The tty driver recognizes a user-defined suspend character which causes all current foreground processes to stop and the user's job control shell to resume. The job control shell provides commands that continue stopped processes in either the foreground or the background. The tty driver will also stop a background process when it attempts to read from or write to the user's terminal. This allows the user to finish or suspend the foreground task without interruption and continue the stopped background process at a more convenient time.

Some of the System V incompatibilities of 4.2BSD job control that are resolved in HP-UX are discussed in the following sections.[3]

SIGHUP Changes

System V semantics state that when a process group leader dies, all processes in the same process group are sent the SIGHUP signal which, by default, kills all the processes. Job control shells execute a command by making all processes in the pipeline belong to the same (brand new) process group and by making the first program in the pipeline be the process group leader. Typically, the first program in a pipeline terminates before the other programs. Under System V semantics, this would cause the premature death of the remaining pipeline. Because of this, 4.2BSD does not generate SIGHUP on process group leader death. To support System V semantics and still allow job control to function properly, HP-UX makes a distinction between a "System V process group leader" and a "job control process group leader." A System V process group leader is given System V semantics (SIGHUP is generated) and a job control process group leader is given 4.2BSD semantics (SIGHUP is not generated).

SIGCLD Changes

Under System V, the SIGCLD signal is sent to a process whenever one of its immediate child processes dies. Under 4.2BSD, SIGCLD (or its variant, SIGCHLD) is also generated when a process changes state from running to stopped. Since a System V application would not expect to receive SIGCLD under these new circumstances and since a job control shell would not be able to function properly without such notification, a compatible compromise was developed. The (parent) process wishing to trap SIGCLD may set a flag when calling the HP-UX sigvector routine to establish a signal handler. This flag will cause SIGCLD to be sent for stopped children, in addition to terminated children. A System V application using signal will see the System V compatible SIGCLD semantics.

Controlling Terminal Changes

Under System V, whenever a process group leader dies, the controlling terminal associated with that process group (if any) is deallocated (disassociated from that process group). 4.2BSD does not deallocate controlling terminals on process group leader death for the following reason: job control shells make the lead process in every pipeline a process group leader. If the controlling terminal for each pipeline were deallocated whenever the lead process terminated, then the remaining processes would effectively become background processes (assuming they were currently in the foreground) and would stop when any of them attempted subsequent I/O to the terminal.

To allow both semantics, controlling terminals are only deallocated when a System V process group leader dies and not when a job control process group leader dies. (See the discussion of SIGHUP changes above.)

tty Driver Considerations

For System V compatibility, the suspend and delayed suspend characters are defaulted to a disabled value (0377). This means that job control is inactive by default when a user logs on. The user must explicitly activate job control by defining either or both of these characters via stty or some similar interface.

References

1. W. Joy, *An Introduction to the C Shell*, Computer Science Division, University of California at Berkeley, November 1980.
2. *UNIX Programmer's Manual*, 4.2 Berkeley Software Distribution, Virtual VAX-11 Version, Computer Science Division, University of California at Berkeley, August 1983.
3. D. C. Lennert, "A System V Compatible Implementation of 4.2BSD Job Control," *USENIX Conference Proceedings*, Summer 1986, pp. 459-474.

David C. Lennert
Member of the Technical Staff
Information Technology Group

by blocking for some unavailable resource or by completing a system call. This may significantly delay the execution of a high-priority real-time process and is unacceptable to critical real-time applications. For a detailed description of how the HP-UX kernel on the Model 840 has been modified to allow preemption in the kernel, see the box "Decreasing Real-Time Process Dispatch Latency Through Kernel Preemption," on page 13. The effects of these modifications are shown in Fig. 6.

Job Control. Job control allows users to suspend the execution of processes selectively and resume their execution at any later point. Job control, as implemented in Berkeley 4.2BSD, is incompatible with the UNIX System V semantics in several areas: signaling mechanism, process group, and controlling terminal semantics. A job control interface that supports the Berkeley functionality and is compatible with the UNIX System V specification has been added to HP-UX and implemented on the Model 840. This interface has been accepted in the working draft of the IEEE P1003.1 POSIX, the *Portable Operating System Interactive Executive* standard. For a detailed description of the HP-UX implementation of job control, see the box "A UNIX System V Compatible Implementation of 4.2BSD Job Control," on page 9.

Process Synchronization

Reliable Signals. A signal is the software interrupt mechanism supported in a UNIX system to relate asynchronous events to user processes. Traditional UNIX systems including System V suffer from the race condition, which is that a signal can be lost or the receiving process can be killed if the signal is sent to a process when the process is in the middle of processing another signal of the same type and has not rearmed its signal handler. The Berkeley system recognized this problem and defined a set of signal system calls to eliminate it. This interface has been adopted by HP-UX and implemented on the Model 840 with minor modifications so that it is compatible with System V.

Shared Memory, Semaphores, and Messages. Shared memory, semaphores, and messages are interprocess communication mechanisms added to UNIX System V.2. The same mechanisms have also been added to HP-UX and implemented on the Model 840.

Memory Management

For a process to have predictable response time, HP-UX allows a process to lock its text segment, data segment, or both in memory. When a process locks its data segment, its stack and heap segments are also locked in memory at the same time. A process can also reserve additional heap and stack space when it locks itself in memory to avoid future page faults. If it outgrows its reserved space, the additional page is locked in memory automatically. Shared segments can be locked in memory by using the shared memory system calls. The above capabilities (through the same interface) have been added to the SVID and are currently supported on UNIX System V.2. The only difference is that HP-UX allows a process to lock memory segments in memory only if it possesses the memory locking privilege. This allows tight control of the memory locking capability. Naive or malicious users who do not have the memory locking capability cannot consume most of the system memory, either intentionally or unintentionally, by locking themselves in memory. When full UNIX System V compatibility is needed, it can be achieved by giving the memory locking privilege to every process. Berkeley 4.2BSD does not support any of the above capabilities.

The swapping policy of the HP-UX implementation on the Model 840 has been modified in such a way that it favors real-time processes. In general, timeshared sleeping processes are swapped out first, then real-time sleeping processes, then timeshared runnable processes, and finally real-time runnable processes. This policy is transparent to the users. The only difference is that timeshared processes may be swapped out more frequently and thus run slower.

File System

Because a UNIX operating system uses its file system very heavily, the traditional UNIX file system is a performance bottleneck. The HP-UX implementation on the Model 840 is based on the Berkeley fast file system, with

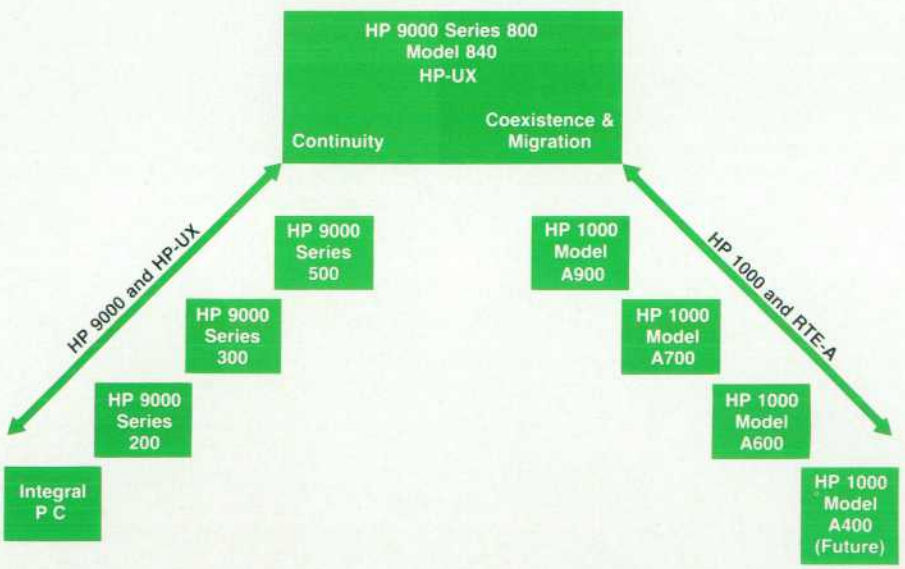


Fig. 7. *Migration paths for HP-UX and RTE systems.*

minor modifications for UNIX System V compatibility. File access rates up to ten times faster than a traditional UNIX file system have been achieved in the fast file system.

The fast file system partitions the disc into one or more cylinder groups, each of which consists of one or more consecutive cylinders on a disc. Inodes, which store information such as file types and locations, are allocated in each cylinder to reduce disc head movement, rather than at the beginning of a file system as in a traditional UNIX operating system. Large data blocks with small fragments are supported to maximize disc transfer rates and minimize file system fragmentation. New data blocks are allocated on the same cylinder as the previous block, whenever possible, with rotational latency taken into consideration. Files belonging to the same directory are placed in the same cylinder group whenever possible. The fast file system user interface is fully compatible with that of a traditional file system.

File Locking. File locking has been added to HP-UX and implemented on the Model 840. Both advisory and enforcement mode locks are supported. An advisory lock allows a process to lock a region of a file to achieve exclusive use of the region among cooperating processes. A process can still have access to an advisorily locked region if it chooses. Enforcement lock enforces exclusive access by the locking process. Other processes that attempt to access the locked resource either return an error or sleep until the resource becomes unlocked. The same interface has been accepted into the SVID and /usr/group standard.

Scatter Read and Gather Write. Scatter read allows a user to read from a file and scatter the data into multiple buffers in one system call. Gather write allows a user to gather data from multiple buffers and write it to a file in one system call. It is extremely useful for data acquisition applications to minimize the system call overhead. The interface was first defined in Berkeley 4.2BSD, added to HP-UX, and implemented on the Model 840.

User Control of Buffering. In traditional UNIX operating systems, the file systems store the most recently used data in their buffer cache to reduce the file system access time and improve the file system bandwidth. The buffer cache is periodically flushed out to disc to ensure data integrity in a system crash. UNIX System V supports the capability of flushing out all of its buffer cache. The operation incurs significant overhead to the system and, therefore, is done infrequently. The Berkeley system added the capability of flushing out only the buffer cache of a specified file to allow users better control of the buffering. This interface has been adopted into HP-UX and implemented on the Model 840. In addition to flushing out the buffer cache of the specified file, HP-UX on the Model 840 also flushes out the inode and indirect blocks of the file so that data integrity of the file is guaranteed.

Preallocation of Disc Space. In a traditional UNIX operating system, the disc space of a file is allocated as needed on write operations. This implies that performance on write is slower since time may be spent allocating space during the write operation. This may be unacceptable to some critical real-time applications. The capability of preallocating file space has been added to HP-UX and implemented on the Model 840.

Powerfail Recovery. The HP-UX operating system on the Model 840 also supports powerfail recovery. On a temporary power failure, the CPU state and data stored in the cache are flushed out to memory backed up by battery. If power is restored within a short time (15 minutes for a 24M-byte system), all I/O devices are reset, I/O transactions ongoing at the time of the power failure are restarted, CPU and cache states are restored, and a signal is sent to each process informing it of the power failure. A process can then take appropriate recovery actions.

Asynchronous I/O

In a traditional UNIX operating system, a process cannot continue its execution until the completion of the requested I/O operation. The delayed write feature in a UNIX operating system is an attempt to gain more parallelism in the system. However, many applications in the computer integrated manufacturing market want to start multiple I/O operations, continue their execution, and then wait for completion of all of the I/O transactions. Other applications may want to perform multiple asynchronous I/O operations and be signaled on the completion of any of the I/O transactions. The Berkeley 4.2BSD system supports asynchronous I/O activities of this type. This capability has been added to HP-UX and implemented on the Model 840. In particular, a process can request that it be signaled when a certain driver state occurs. Alternatively, it can poll a driver to determine if the driver is ready for reading or writing, or if it has an exceptional condition pending. It

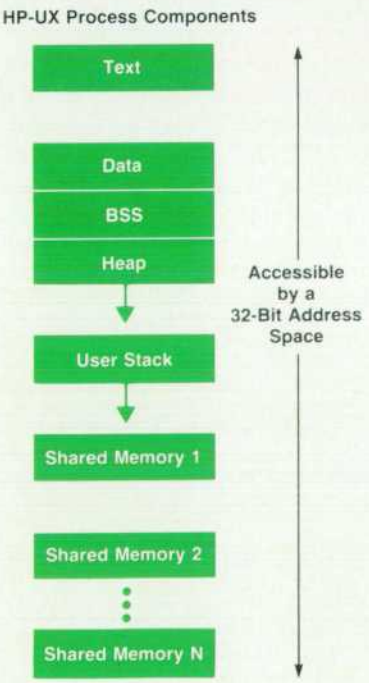


Fig. 8. *AT&T's* UNIX System V Interface Definition *implies that processes have the following components: text, data, unitialized data, a user stack, a heap, and zero or more shared memory segments. The user stack may grow automatically to suit the procedure call nesting of the process. The heap will grow (or shrink) when the process requests more (or less) memory using the* brk *or* sbrk *system calls.*

can also make nonblocking requests to drivers.

Custom I/O Drivers

In the computer integrated manufacturing market, application developers may want to write their own device drivers to have direct control of their equipment. The HP-UX implementation on the Model 840 supports a configuration tool to allow users to add drivers to the system without requiring the acquisition of a UNIX system source license.

Debugging tools used by the lab developers are also available to users for their driver development: a kernel debugger for debugging the kernel on the hardware, an off-line driver debugging environment to allow logical debugging of software in a UNIX system user environment before testing it on the hardware, and driver skeletons to help users in designing their own drivers. HP's Data Systems Division also plans to release the above tools as products shortly after the first release of the Model 840, together with an HP-UX source product with a technical specification of the internal operation of the HP-UX operating system.

The device I/O library is a set of libraries in HP-UX that allows users to have direct control of I/O devices. It was first supported on the HP 9000 Series 500, then on the Series 300, and is now supported on the Model 840. With first release of the Model 840, users have, through this library, direct access to the HP-IB card, which is used for instrumentation, and the high-speed parallel I/O card from HP's Roseville Networks Division.

Software Quality

Extensive measures have been taken to ensure that the HP-UX software on the Model 840 meets the high quality standard of HP. In particular, the HP-UX software meets the defect density, breadth, and depth coverage requirements as discussed in reference 19. Two software tools have been developed to measure the test coverage of the software: the path flow analyzer (PFA) and the instruction coverage analyzer (ICA). The PFA operates by adding instructions to each path of a program to count the number of times a path has been executed. It was used on all software in the system except the kernel to confirm the extent to which tests have been comprehensive. The ICA operates by replacing all of the instructions of a program at the very beginning of an execution with break instructions. A special trap handler puts the original instruction back when a break instruction is encountered. The program runs very slowly at the beginning but soon runs at its normal speed when most of the break instructions have been replaced. The ICA tool has been used extensively in kernel testing since it does not alter the execution timing of the system. It has proven to be a very valuable tool.

Software Migration Aids

HP's computer products strategy presently calls for a convergence upon three operating systems for all products. These are MS-DOS for personal computer products, MPE for commercial and business data processing markets, and HP-UX for most other computer markets to which HP caters. These include scientific and engineering computation, real-time process control, computer-integrated manufacturing, and general-purpose computing.

HP's strategy calls for migrating the established base of RTE customers and application software, where appropriate, to HP-UX beginning with the advent of HP Precision Architecture (see Fig. 7) and continuing through the next decade as HP Precision Architecture systems become available for all types of applications. To facilitate this process, HP has developed an entire package of migration tools called Port/HP-UX, which is included in the base software package for the HP 9000 Series 800.

Limited tools are provided for previous members of the HP 9000 family to facilitate migration of HP 1000 software to those machines. The Model 840, however, is the first HP-UX implementation to offer sufficiently complete real-time functionality and sufficiently fast real-time performance to accommodate the vast majority of HP 1000 applications. For this reason, a much more extensive set of migration tools was deemed appropriate for the Series 800 than those offered for Series 200, 300, or 500 members of the HP 9000 family. Of course, no tools are needed for migration from Series 200, 300, or 500 systems to Series 800 machines. This is because of the inherent portability of software based on UNIX operating systems.

Port/HP-UX provides an extensive library of emulation

(continued on page 15)

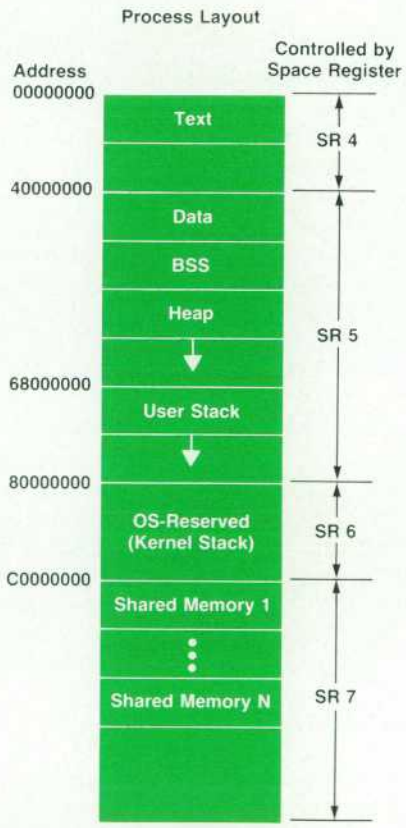


Fig. 9. *The components of an HP-UX process are mapped onto HP Precision Architecture as shown. The kernel stack is a stack used only while the kernel is executing on behalf of the process, and it is otherwise inaccessible to the process. The kernel is not included in the short-pointer address space of the process. Rather, it executes out of a separate space (0) and it accesses the memory of the process using a handful of functions that use long-pointer addressing mode.*

Decreasing Real-Time Process Dispatch Latency Through Kernel Preemption

A key measure of a real-time system is how quickly a waiting process can be dispatched in response to some event (for example, I/O completion). One major component of this is the time it takes to preempt the currently executing process. In a traditional UNIX operating system, a process executing in user code can be preempted immediately. However, when executing in the kernel, the process gives up the CPU only voluntarily and explicitly (for example, by blocking for some unavailable resource or by completing a system call). The kernel can therefore execute for a significant period of time before giving up the processor to another process. This period of time is called preemption latency and, when significant, it is unacceptable in a real-time system.

This article describes modifications to the HP-UX kernel that substantially reduce this time. Measurement results are presented that quantify these times and the improvements that have been made.

Alternative Solutions

The goal is to decrease the amount of time the kernel executes before it gives up the processor to a waiting higher-priority real-time process. To achieve this goal there are two basic alternatives: 1) the kernel can be made to execute all of its functions more quickly, or 2) the kernel can be made to tolerate interrupting its execution in deference to the waiting process (preemption).

The former is clearly a superior approach, because it has the side benefit of causing the entire system to execute faster and with less kernel overhead. It can be achieved through a combination of faster hardware and algorithmic changes. In addition to algorithm changes that reduce total execution time, one can shift code from the kernel into the user program. One example would be to implement the file system manipulation code in a user library and leave only the code supporting basic device access in the kernel. This allows more of the "kernel" to execute in user mode where it is readily preemptable. The problems with moving kernel algorithms into user mode are a loss of reliable security checking and a loss of atomicity of operation with respect to other processes. In addition, there is only so much kernel code that can be reasonably moved into user mode. In the final analysis, preemption latency is typically left unacceptably high. So the second alternative, increasing the preemptability of the kernel, is explored.

The problem with making the kernel arbitrarily preemptable is a loss of atomicity. Kernel data structures can be viewed as memory shared among all the user processes. Each process makes requests of the kernel that update this shared data. There must be a mechanism that ensures that these updates are performed atomically; otherwise, faulty operations and a system crash usually result. Simultaneous (multiprocessor) and interleaved (uniprocessor) data structure access is prevented either through one or more semaphores (which reduces the problem to updating shared semaphores) or by preventing even the possibility of contending access. The latter approach is usually provided by atomic hardware instructions and/or architecting the system so that such colliding accesses never happen.

The mechanism used in a traditional UNIX operating system is this latter approach: nothing interrupts a process while it is running in the kernel. (The exception to this, I/O interrupt processing, will be discussed later.) This implementation has too coarse a granularity. That is, the data structure lock can be held for a long period of time, and this can prevent other processes from running even if they don't access the same data structures being currently updated. Thus the lock covers more data structures and lasts for a longer period of time than is usually needed. It is this drawback that gives rise to the poor preemption latency of UNIX operating systems.

Solution Implementation

The preferred solution is to use multiple semaphores and have each semaphore control access to an independently used data structure. No other process will access the data structures the preempted process is using since no other process has the necessary semaphores locked. The kernel can then be immediately preempted at any point in its execution. This results in the fastest preemption time but requires that the entire kernel be modified to adhere to semaphoring conventions. Just sorting through the various data structures and assigning semaphores can be a large amount of work. This approach is typically employed in multiprocessor systems. For a description of one such implementation and the effort required see Bach[1] and Felton.[2]

An approach that is easier to implement is to find places in the kernel where it is already safe to preempt and only allow preemption there. Such a safe place is a spot or region in kernel code where all kernel data structures are either updated and consistent or locked via semaphore. This does not require modifying the entire kernel to conform to a new data access philosophy. It does have several drawbacks, however. Rather than occurring immediately, preemption is held off until the next safe place. Also, our experience has shown that these safe places are not found but made. This approach can be viewed as a generalized extension of a technique described by Ferrin.[3]

Because implementation schedule was of strong importance, our solution combines both these preemption styles: there is a synchronous method, which allows preemption at a specific point during kernel execution, and an asynchronous method, which allows preemption anywhere during a region of kernel execution.

The synchronous method is useful when places can be identified in the kernel where data structures are either in a consistent state (i.e., between an access transaction) or all required resources are locked via some semaphoring mechanism. The synchronous method is invoked by placing a call to the macro KPREEMPTPOINT at such a safe place in the kernel. This macro merely checks a global flag, reqkpreempt, which indicates the presence of a higher-priority real-time process that is ready to run and calls a function, kpreempt, to cause a swtch to the process. The reqkprempt flag is similar in function to the runrun flag used in typical UNIX operating systems to indicate that a higher-priority timeshare process is ready to run.

There is also a function variant of KPREEMPTPOINT called IFKPREEMPTPOINT which returns true if a pending preemption was serviced; otherwise it returns false. This is useful if lengthy algorithms need to allow preemption, but if preemption occurs, there are assumptions that may have been invalidated and now must be rechecked.

The asynchronous method is useful when preemption can be tolerated over a region of execution and synchronous polling via KPREEMPTPOINT would incur unacceptable overhead (for example, large memory copies during fork, exec, or user I/O). This method is implemented via a software-generated interrupt which is recognized by hardware. Hardware causes an asynchronous transfer of control to the trap handling routine (similar to a page fault taken inside the kernel when accessing user pages), which in turn calls kpreempt to preempt the kernel.

In total, approximately 180 synchronous preemption points and 20 asynchronous preemption regions were added to the HP-UX kernel.

Overcoming Limitations

There is one overriding limitation on what kernel preemption can accomplish. Kernel preemption can only preempt (suspend via swtch) an operation being executed within a process context. It cannot preempt interrupt processing code and allow a process to execute because the UNIX system does not support this type of operation. Therefore, all interrupt processing is implicitly considered to be of higher priority than any (real-time) process. This means that no matter how quickly preemptable one makes the kernel, if interrupt processing becomes unacceptably time-consuming then timely kernel preemption cannot be achieved. So the only option is to reduce interrupt processing overhead to an acceptable level. Note that, even if all individual interrupt servicing operations are short, kernel preemption can be held off for an arbitrarily long time by many quickly arriving (back-to-back) interrupts during heavy I/O activity. There is nothing that can be done in this situation since interrupt processing, by definition, has priority over all process execution. In addition to the typical I/O driver code, the UNIX system allows non-I/O code to be executed in an interrupt processing context. This facility, called the callout queue, causes a kernel procedure to be executed at a specified time offset. The procedure is invoked from an interrupt processing context during clock interrupt servicing. This is usually done at a weaker interrupt priority than all other I/O interrupts. (See Straathof[4] for a more detailed discussion of the callout queue mechanism.)

To minimize callout queue execution overhead, a separate system process was created to provide a preemptable process context in which to execute some lengthy callout queue code. This process, the stat daemon, is a lightweight kernel process similar to the swapping daemon (sched) or the pageout daemon. It waits for the lightning bolt event (the lightning bolt event is a standard UNIX operating system event that occurs frequently, for example, every second) and then executes a standard set of statistics gathering routines. These routines represent the lengthy portion of the schedcpu function. (Among other things, schedcpu recomputes process priorities every second; see Straathof[4] for a discussion of its operation.) A new routine, sendlbolt, is now scheduled on the callout queue in place of schedcpu. Sendlbolt performs the quick functions of schedcpu including generating the lightning bolt event.

General Performance Improvements

In addition to the preemption specific modifications, kernel preemption times were improved by several general performance improvements. These include making the process table multithreaded and placing entries in different states on different lists, and using hashing techniques to speed data structure searches. See Feder[5] or McKusick[6] for a discussion of similar improvements.

Measurements

To tell how long the kernel executes without blocking or preempting, and where in the kernel these long execution paths are, the kernel was instrumented to collect timing measurements. To obtain sufficiently accurate times, a new kernel routine was introduced to obtain clock time accurate to a microsecond.

To determine the improvement made in real-time process dispatch time, two sets of measurements were taken, one set with kernel preemption enabled and one set with it disabled. The same workload was run during both measurements.

The workload consisted of a suite of tests that validate the correct working of all kernel functions. Because the instrumented kernel precisely measures each section of the kernel every time it is executed, it is not necessary to execute a kernel code path more than once.

The results are summarized in Table I.

Table I
Nonpreemptable Kernel Time

	Preemption Off	Preemption On	Improvement
90% kernel	40 ms	1.4 ms	×28
99% kernel	129 ms	3.4 ms	×37
Maximum kernel	1127 ms	14.6 ms	×77

Note that these results are for a particular run of a particular workload. Results will vary from run to run and from workload to workload.

Table I shows that HP-UX kernel preemption has provided significant improvements in real-time process dispatch time. In the worst case observed with our workloads the improvement was well over 50-fold. With preemption enabled it was found that nonpreemptable kernel code paths were significantly shorter, and more consistently so, than in the traditional case. This resulted in better and more reliable timely dispatch of real-time processes regardless of background workload.

Acknowledgments

The following people from Hewlett-Packard contributed to this work: James O. Hays introduced preemption points and regions into the memory and process management systems, and also off-loaded interrupt processing functions into the newly created stat daemon. Sol F. Kavy introduced preemption points and regions into the file system. Suzanne M. Doughty edited early versions of this article.

References

1. M. J. Bach and S. J. Buroff, "Multiprocessor UNIX Operating Systems," *AT&T Bell Laboratories Technical Journal*, Vol. 63, no. 8, October 1984, pp. 1733-1749.
2. W. A. Felton, et al, "A UNIX System Implementation for System/370," *AT&T Bell Laboratories Technical Journal*, Vol. 63, no. 8, October 1984, pp. 1751-1767.
3. T.E. Ferrin and R. Langridge, "Interactive Computer Graphics with the UNIX Time-Sharing System," *Computer Graphics*, Vol. 13, no. 4, February 1980, pp. 320-331.
4. J.H. Straathof, A.K. Thareja, and A.K. Agrawala, "UNIX Scheduling for Large Systems," *USENIX Conference Proceedings*, Winter 1986, pp. 111-139.
5. J. Feder, "The Evolution of UNIX System Performance," *AT&T Bell Laboratories Technical Journal*, Vol. 63, no. 8, October 1984, pp. 1791-1814.
6. M.K. McKusick and M. Karels, "Performance Improvements and Functional Enhancements in 4.3BSD," *USENIX Conference Proceedings*, Summer 1985, pp. 519-531.

David C. Lennert
Member of the Technical Staff
Information Technology Group

(continued from page 12)

routines that permit most HP 1000 applications programs written in Fortran or Pascal to be moved to HP-UX on HP Precision Architecture with only a simple recompilation. To provide early assessment of the likely magnitude of a contemplated HP 1000-to-Series 800 port, Port/HP-UX includes a migration analysis utility (or MAU).

The MAU reads the source code of HP 1000 applications programs and automatically identifies any constructs that may need to be changed to work properly in the Series 800 environment. To assist current HP 1000 owners in anticipation of transitions of their own enterprises from HP 1000 systems to the HP Precision Architecture family, the MAU is also available on HP 1000s.

HP 1000 Programs on HP-UX

HP 1000 programs run as native HP-UX programs whenever they are in user application code. This gives the immediate benefit that application code runs significantly faster on the Series 9000 Model 840 than it does on an HP 1000. Ported programs also can take advantage of features of HP Precision architecture, including 32-bit address space, paged virtual memory, and a rich set of registers.

HP 1000 programs are recompiled to run on HP-UX using the standard HP-UX compilers. The Fortran compiler in particular includes a number of extensions to facilitate compiling HP 1000 programs. Programs are linked with the standard HP-UX linker and libraries containing essentially all commonly used RTE entry points, such as EXEC, FMP, the RTE system library, Image, AGP, DGL, and F/1000.

These libraries of RTE entry points make use of HP-UX facilities to provide an environment suitable for RTE programs. Key RTE tables such as the ID segment table, I/O tables, and resource tables are maintained in shared memory and are accessed by the EXEC and FMP calls. An FMP format file system is maintained using HP-UX facilities, and a set of utility programs is provided for manipulating the RTE environment. These programs include a command interpreter called rtesh that provides most of the commands available from CI on RTE, and an editor called ed1000, which simulates EDIT/1000 on RTE, including screen mode and regular expressions.

The RTE libraries attempt to provide complete emulation of their RTE equivalents; differences are documented, and are identified by the MAU. Performance of the emulated RTE calls is such that most application programs will still see a net performance improvement when they are moved from an HP 1000 Model A900 to an HP 9000 Model 840. Exceptions are programs that are very operating system dependent. Some of the calls, such as resource number operations and HP-IB transfers, can take more time on the 840 than they do on the A900.

Implementation Highlights: The Kernel

The implementation of the HP-UX operating system on the HP Precision Architecture HP 9000 Series 800 Computers is fully compatible with the *AT&T UNIX System V Interface Definition* (SVID) and with the *HP-UX Standard Specification Version B.1*. It is an operating system tuned for high system throughput in a multiuser environment and fast real-time response.

Conforming to the SVID has many implications, among which is the support of a process model. This process model defines how processes are created, how they perform input and output, how they communicate with one another, and how they are destroyed. It also implies what components make up a process, and how they may be accessed. The UNIX operating system process model assumes that each process has its own separate 32-bit address space, although certain components in that address space may be shared between processes. HP Precision Architecture has a 64-bit or 48-bit address space, but, as will be described later, it also has an addressing mode that allows support of the 32-bit addressing model.

Each process is assumed to have the following components (Fig. 8, page 11):

- Text—the instructions of the program. Most UNIX systems separate instructions from data not only to enforce good programming practice, but also to allow for the nonwritable instructions to be shared among many processes. The text must be not only executable, but also accessible as data because constants are often loaded here to increase memory sharing.
- Data—the data of a program. This component is initialized from the program file during an exec system call.
- Bss—the data of a program that was not initialized in the program's source code. The SVID process model assumes that bss is initialized to zero during the exec system call. It is kept separate from the data component and does not take up space in the program file. (By the way, bss is a time-worn mnemonic from a computer long since dead. It means "block started by symbol.")
- Heap—a data area that is allocated at run time. The heap will grow or contract when requested by the process through a brk or sbk system call. Many application programs assume that heap starts at the end of the data component and grows toward higher-numbered address-

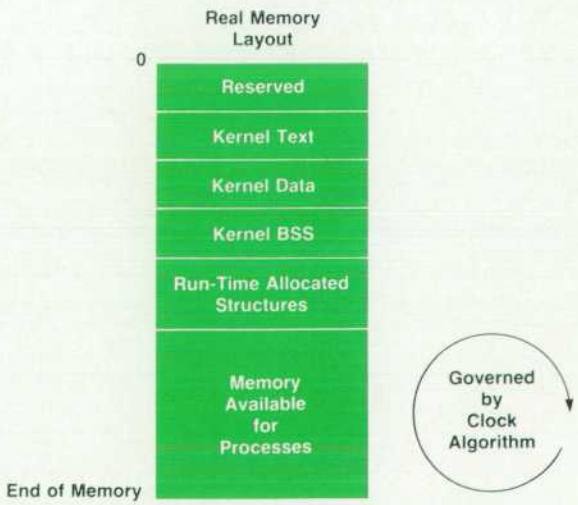


Fig. 10. *This is how the real memory of the computer is organized. First is the area reserved by the architecture, followed by the kernel text, data, and bss. Next are kernel data structures that are sized during system initialization. The remaining memory of the computer is available for processes.*

es, so this convention has become part of the process model.

- Stack—the procedure stack. This stack automatically grows to suit the procedure nesting of the process.
- Shared memory segments. Shared memory segments allow processes to share data quickly and easily, without system call overhead. Each process may have zero or more shared memory segments, and may control who may read or write them based on user and group numbers.

Mapping the UNIX Process Model onto HP Precision Architecture

For any UNIX operating system implementation, the process model must be mapped onto the underlying computer architecture. HP Precision Architecture is well-suited to support the SVID process model through its state-of-the-art virtual memory architecture. The architecture has two basic modes of memory access: real (physical) and virtual.[20] While running in real mode, the processor accesses memory using a 32-bit address generated from registers and displacements selected in the instruction. While running in virtual mode, the processor accesses memory using a 64-bit address. (Actually, initial releases of HP Precision Architecture computers implement a 48-bit address. The upper 16 bits are not required to meet current market needs.) Only small portions of the operating system execute in real mode; most of the operating system and all processes execute in virtual mode.

The 64-bit virtual address is also generated from registers and displacements selected in the instruction, but because the processor's general registers are 32 bits wide, more bits are needed. These bits are taken from separate registers called space registers, of which there are eight. The space registers must be loaded with the desired space number before the instruction that accesses memory is executed.

There are two different ways by which a space register is selected, and these define the two basic virtual addressing modes of the architecture: long-pointer and short-pointer.[20] Long-pointer addressing mode is selected when a two-bit field in the instruction is 1, 2, or 3. In this case, space register 1, 2, or 3 is selected, respectively. The contents of this space register (called a space number) are concatenated with the 32-bit address generated from general registers and displacements selected in the instruction to produce a 64-bit address. The 64-bit virtual address is then converted to a 32-bit real address through a combination of hardware and software.

Previously it was mentioned that the process model implies that a process is accessed using addresses that are 32 bits wide. The short-pointer addressing mode provides this functionality, and is used exclusively to implement the process model for HP-UX. In this addressing mode, the lower 32 bits of the 64-bit address are generated (as usual) from general registers and displacements selected in the instruction. The upper two bits of the address contained in the register are used to select one of space registers 4, 5, 6, or 7, whose contents are then concatenated with the lower 32 bits to form a complete 64-bit virtual address.

To sum up so far, the short-pointer virtual addressing mode is used exclusively by processes in HP-UX. The long-pointer address mode is only used by the operating system, and only to a limited extent. All long-pointer addressing is limited to routines written in assembly language in the operating system.

The components of a process are accessed by the process using the short-pointer addressing mode (see Fig. 9 on page 12). The 32-bit address space is effectively broken into quadrants whose space numbers are determined by space registers 4, 5, 6, and 7. The text of the process is governed by space register 4, and it begins at the start of that quadrant. The data, bss, heap, and user stack of the process are governed by space register 5. Data, bss, and heap are contiguous and begin at the beginning of the quadrant. User stack begins in the middle of the quadrant. The shared memory segments are governed by space register 7, and are located throughout the quadrant.

The quadrant governed by space register 6 is reserved for operating system use, and is currently used to contain the stack that the kernel uses when executing on behalf of the process.

Memory Management

HP-UX on the HP 9000 Series 800 has full virtual memory support for all process components, and the operating system has been tuned for both multiuser and real-time environments. Memory management is done through a combination of paging and swapping. Paging is implemented through a process called the pageout daemon, which scans through memory, pushes dirty pages to disc memory called the swap device, and tries to keep a reasonable amount of memory on a list of pages that are free. When the paging system becomes overloaded, a swapper process (sched) will push whole processes to the disc and keep them from executing until the overload has subsided. Swapping has two positive effects: 1) it frees up large amounts of memory very quickly, and 2) it prevents the swapped process from page faulting and keeping the paging system swamped. The paging system is overloaded when all of the processes cannot keep their working sets in memory.

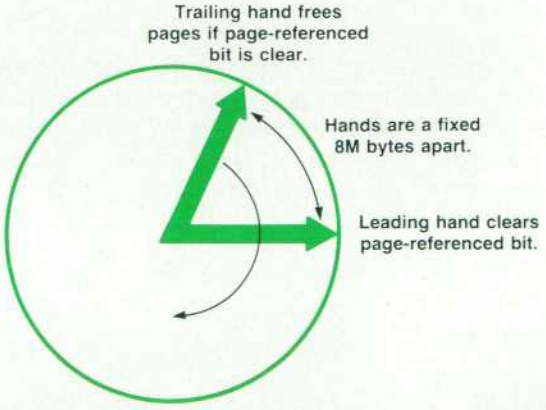


Fig. 11. *The global clock algorithm is used to approximate a least recently used page-replacement algorithm. In this algorithm, the pages of real memory are depicted as wrapped around the edge of a clock. A pair of hands sweeps around the clock, with the leading hand clearing the page-referenced bit, and the trailing hand freeing pages that have not been referenced.*

The following are features of memory management on the HP 9000 Series 800:

- Demand paging. A process may reside in memory or on disc with a granularity of the 2048-byte page of HP Precision Architecture.
- Page reclaims. After pages have been unmapped and put on the free list, they can still be reclaimed when the process faults on that page.
- Page clustering. When paging from the swap device, up to eight contiguous virtual pages around the faulted page may be brought in in one disc transfer. When paging to the swap device, up to eight contiguous dirty virtual pages will be pushed to the swap device in one transfer.
- Paging from file. The text and data of the process will initially be paged directly from the program file. Thereafter, it is paged to and from the swap device. This means that the whole program need not be brought into memory on exec, and pages will only be brought in as needed. Even though text is nonwritable, it is moved to the swap device to take advantage of page clustering later.
- Zero fill on demand. Virtual pages in the bss, stack, heap, and shared memory segments will be allocated real memory and zeroed on first demand.
- Multiple swap devices. The system supports more than one swap device through the swapon system call and the swapon system administration command. The swap area is considered to be interleaved among the swap devices, leading to increased bandwidth for swapping because there are multiple disc heads and spindles. This also means that virtual memory limits are unconstrained by the maximum disc size available.
- Text page reattaches. Between executions of a program, the pages of text may be on the free list. Before the operating system decides to bring in text pages from disc, it checks to see if the page is in the free list by searching in a hashed list of pages sorted by text disc block number.

Real Memory Layout

The real memory of the computer contains both the operating system and the application programs as shown in Fig. 10. A few pages are used at the beginning of real memory for the architecturally reserved functions. Thereafter, the kernel text, data, and bss are placed. After that are data structures that are allocated at run time, such as the file system buffer cache, whose size is a fraction of the memory in the machine. Lastly comes memory that is available for applications programs and is controlled by the paging and swapping algorithms.

The area available for paging and swapping is governed by an algorithm called the clock algorithm.[21,22] This algorithm approximates a least recently used algorithm. The only hardware assistance required are page-referenced and page-dirtied bits, which is why the algorithm has been ported to many architectures.

The clock algorithm is implemented by the pageout daemon. In the clock algorithm, the real pages available for paging are represented as being wrapped around the edge of a clock (Fig. 11). The clock has two hands, and they are kept a fixed distance apart. The hands move around the clock at a speed that is dependent on the fraction of memory in the free list of pages. The lower the fraction of free memory, the faster the hands move around the clock. The pageout daemon only bothers to run when less than a quarter of memory is free.

The pageout daemon clears the page-referenced bit for the real page to which the leading hand points, and checks the page-referenced bit for the real page to which the trailing hand points. If the page has been referenced, then it is left alone. If the page has not been referenced, then the page-dirtied bit is checked. If the page is not dirty, it is unmapped and placed onto the free list where it can be used for a new purpose or reclaimed. If the page is dirty, then it is placed on a list of pages to be copied to the swap device and given to the swap device driver. After the page has been copied the pageout daemon will place it in the free list.

Architectural Accommodations

Many of the innovations of HP Precision Architecture have implications for operating system designers. Two important ones are 1) delayed branches and signals and 2) the virtual caches.

The architecture allows for a delayed branch concept of instruction execution that exposes a two-level pipeline to the compiler designers and the operating system designers. After an interrupt or trap, this pipeline can be restored

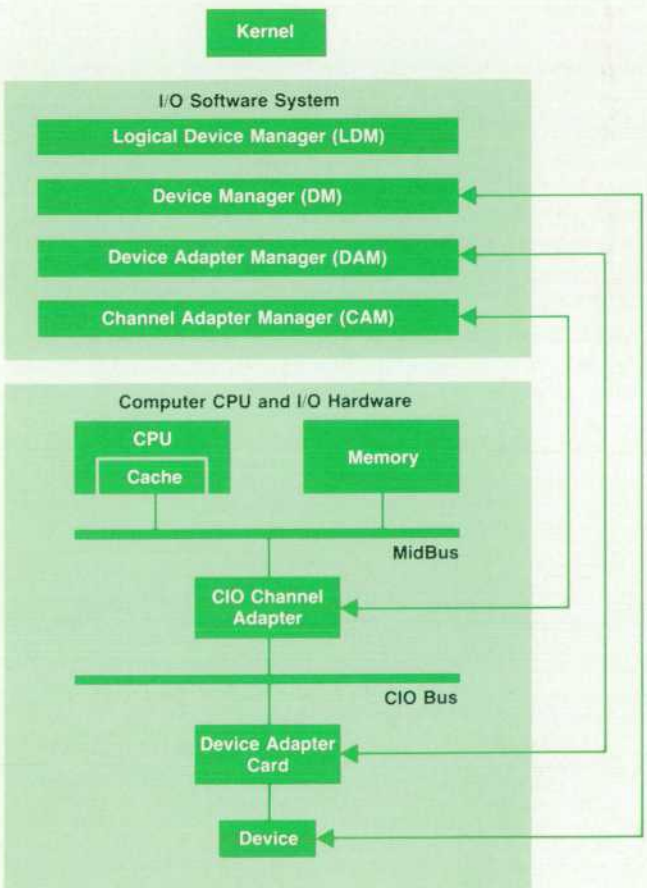


Fig. 12. *The hierarchical HP-UX I/O system on the HP 9000 Model 840 Computer uses the HP CIO system. There are three levels of hardware. The software structure reflects the hardware structure; there is a manager (driver) for each hardware element.*

only by using a privileged instruction, and this impacts how the signal software interrupt mechanism of HP-UX is implemented. In other architectures, the signal stack is set up by the operating system, the signal function is executed, and then user-level code restores the state of the process at the time of the signal. Because the HP Precision Architecture pipeline can only be restored by a privileged instruction, a special (hidden) system call has been added to implement the signal return.

Virtual Cache

One of the most unconventional aspects of HP Precision Architecture is the area of memory cache design. Several of the architecture's features require more work on the part of the operating system. These include:

- No requirement for a write-through data cache
- Separate code and data caches
- Virtually addressed caches.

Because HP Precision Architecture does not require implementations to have a write-through cache, physical memory will not always be consistent with the data cache. One effect of this is that I/O transactions, which access physical memory, must have data flushed from the cache before the transaction begins (more about this later). Also, separate code and data caches imply that when code is modified by the processor, it must write the data to the data cache, purge any oustanding cache lines in the instruction cache, and flush the data from the data cache to main memory, where it can then be accessed consistently from the instruction cache. One example of where this occurs is in the ptrace system call, which is used to set breakpoints in the text of a process. Although these innovations require more work on the part of the operating system, the overall performance improvement far outweighs the effort.

HP Precision Architecture supports virtually addressed caches, which allows for parallelism in cache access and virtual translation. While the virtual addressing hardware is generating the physical address and checking page access rights, the cache is using the virtual address to select a cache line and present it to the processor. Cache misses are determined in parallel with the processor's acting on the data in the cache line.

This feature leads to higher performance for the architecture, but it also leads to interesting problems for the operating system. One rule of thumb is that the operating system is never able to point two different virtual addresses to the same physical address. This would lead to data inconsistency, because the different virtual addresses might address different cache lines. Thus, HP Precision Architecture does not support a feature loosely termed address aliasing.

One impact of this restriction is that the full SVID shared memory definition cannot be implemented. In UNIX System V, a process can attach a shared memory segment at an address it specifies in its virtual space. If two or more processes want to access the shared memory segment at addresses that they specify, the result is address aliasing. HP-UX on HP Precision Architecture only supports shared memory attaches at an address that the operating system chooses, which will be the same for all processes sharing that shared memory segment. This is the most often used method for attaching shared memory segments, so the lack of address aliasing is not critical.

Implementation Highlights: The Input/Output System

HP Precision Architecture supports a memory mapped I/O architecture. I/O devices are accessed by normal read and write operations. The I/O subsystem for the HP 9000 Model 840, the first HP Precision Architecture computer that runs the HP-UX operating system, uses HP's channel I/O (CIO) system for its initial implementation. The choice of CIO allows device adapters designed for HP 9000 Series 500 Computers to be used in the Model 840. The decision to leverage existing device adapter cards reduced the time required to design the I/O software, and allowed support of a large number of prototype systems.

I/O Software Design Concepts

The HP-UX I/O system is designed to provide an interface to the I/O hardware that conforms to the *HP-UX Standard Specification Version B.1*. The system is also designed to make use of common interface routines to handle driver requirements, and to be easily configured to adapt to a wide range of hardware configurations. The underlying structure of the I/O subsystem is designed to follow the natural hierarchy of the hardware. A manager (driver) exists for each piece of hardware in the system. A message-based protocol is used to communicate between layers in the hierarchy. Each manager in the software hierarchy is presented with a serial stream of messages. While the module is processing a message, it cannot be reentered.

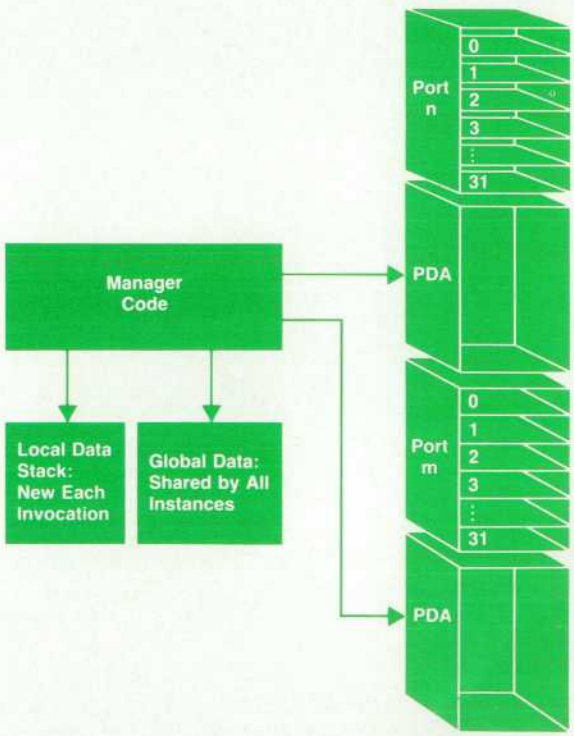


Fig. 13. *An instance of a manager is made up of four data areas.*

In addition to normal I/O functionality, a uniform support interface for the system is provided. To make HP-UX more supportable in the field, an error logging interface and an interface for diagnostic programs were defined. The error logging interface allows system histories to be maintained.

Hardware Hierarchy

HP Precision Architecture supports a hierarchical I/O hardware subsystem.[23] In the case of the CIO subsystem on the Model 840, there are three levels of hardware in the system.

A CIO channel adapter, also called a bus adapter, resides on the midbus, where it converts midbus requests to CIO requests, and presents a memory mapped view of the CIO device adapter cards (see Fig. 12). On the CIO bus, up to 12 device adapter cards are multiplexed by the CIO channel. Some device adapter cards, like the HP 27110B HP-IB card, can handle up to eight devices simultaneously. Each level imposes some constraints on the type, size, and number of operations that can be supported. HP Precision Architecture requires that DMA transactions be cache line aligned, up to a maximum cache line size of 64 bytes. On the Model 840, the midbus requires that DMA transfers be 32-byte aligned, that caches be flushed by software before I/O starts, and that I/O buffers be a multiple of 32 bytes long. CIO requires that transaction starts be synchronized in a certain way, and each device adapter card has its own particular limitations. The I/O software is designed to reflect this hierarchy of hardware and solve the problems of each hardware level in the associated code.

Software Hierarchy

At the lowest level, the CIO channel adapter manager (CAM) handles the multiplexing of starts and completions on the CIO channel. Device adapter managers (DAMs), such as the HP-IB device adapter manager, send requests to and receive completion notification from the CAM. Above the DAMs, device managers (DMs) handle device specific protocols. Finally, logical device managers (LDMs) handle the interface. Any of these levels can be collapsed for simple devices or high performance.

For each occurrence of a piece of hardware in the system, there exists an instance of a manager to control it. An instance of a manager is made up of four parts (see Fig. 13): the manager code which is shared by all instances of the manager, global data which is shared by all instances of the manager, the port data area (PDA) which is private static storage for a single instance of the manager, and a local data stack which is new for each invocation of the manager. Each instance of a manager also has a port associated with it, which maintains the relationship between an instance of a manager and its PDA, and provides the connection to other managers.

HP-UX Interface

The HP-UX I/O system presents a traditional UNIX operating system interface to the user. The HP-UX I/O system appears to the user as standard files which may be accessed through open, close, read, write, and ioctl calls. HP-UX provides security for the I/O system through the standard file ownership and protection mechanisms.

An LDM has access to several HP-UX kernel functions to lock down memory and validate addresses, allocate and deallocate kernel buffers, set and reset timers, move data to and from user space, and sleep on an event. An LDM has the choice of doing DMA directly to or from user space, allocating a system buffer to cache data, or using a character list to manage byte FIFOs.

When the LDM is ready to touch the hardware, it uses I/O services to make a request to the next lower manager.

I/O Services

I/O services provide the functions needed to configure managers, communicate between managers, and log errors reported by managers.

During the configuration process, I/O services build the software hierarchy to match the configuration, allocate port numbers to managers so that a parent or child manager in the hierarchy can invoke the correct instance of a manager, and provide memory allocation for managers so that private, static data (PDA) can be allocated for an instance of a manager. I/O services also provide timer services for managers to use, as well as an error logging facility.

The major role of I/O services is to route requests and replies from one manager to another (see Fig. 14). Managers request service from other managers by sending a message. A reply is a message from a manager that has completed servicing a request. I/O services use a port number passed with each request or reply to invoke the correct instance of a manager. With each invocation, I/O services pass a message from a sending to a receiving manager, and guarantee seriality of execution through an instance of a manager. This means that a particular instance of a manager that has been invoked cannot be reentered, but higher-priority interrupts can be processed. If I/O services find that an instance of a manager is busy when a message is ready to be delivered, I/O services queue the message on the port for that

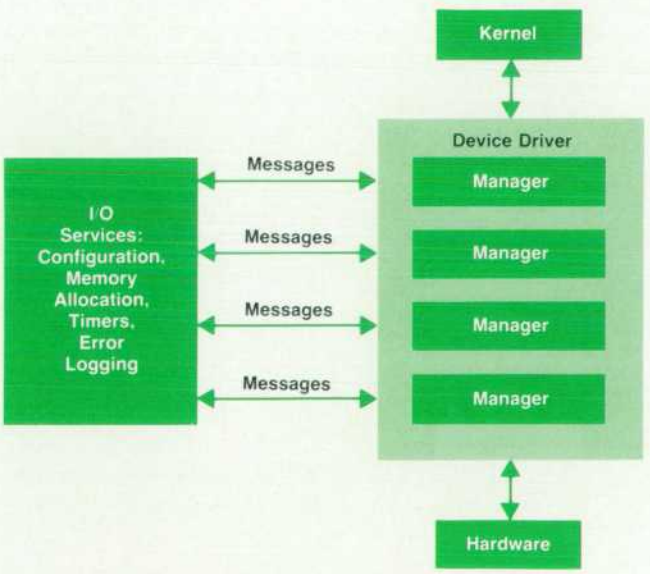


Fig. 14. *I/O services provide functions needed to configure managers, log errors, and route messages from one manager to another.*

instance of the manager, and deliver it when the manager returns. Managers save state and exit while they are waiting for a request to be serviced. The manager will be invoked again by the reply coming from the serving manager.

Supported Interfaces

At first release, the Model 840 supports a set of CIO device adapter cards that allows connection to most existing HP peripherals and to customer black boxes (see Fig. 15).

The HP-IB device adapter card supported on the Model 840 is the HP 27110B. It is supported as a standard IEEE 488 bus for instrumentation, as a high-speed bus for HP CIPER printers (HP 2563/64/65/66/67) and magnetic tape drives (HP 7974/78), and as a high-speed dedicated bus for HP CS-80 and SS-80 disc and tape drives.

To optimize performance of the disc subsystem, a monolithic LDM, a manager that interfaces to the HP-UX kernel and the CIO CAM, was written to support an HP 27110B device adapter with only CS-80 discs/tapes on it. The result is a very high-performance disc interface capable of supporting four discs simultaneously at full speed, or eight discs at a slightly lower rate.

Instrument control is possible on the Model 840 through the use of the Device Independent Library (DIL), the HP-UX standard interface library for HP-IB instruments. This library gives the user complete control over the HP-IB device adapter, making it possible to talk directly to any device that can connect to the HP-IB.

It is also possible for users migrating from the HP 1000 Computer RTE operating system to use the RTE HP-IB libraries provided by Port/HP-UX for RTE programs that control instruments.

The HP 27140A 6-channel terminal multiplexer is the supported RS-232-C interface on the Model 840. The HP 27140A supports full modem control on all six ports, as well as XON/XOFF flow control. The HP 27140A provides the connection mechanism for terminals and for serial printers such as the HP LaserJet. Each multiplexer in the system is polled every 30 milliseconds to pick up any incoming characters and to send out any characters in the outbound queue.

The HP 27125B interface allows the Model 840 to connect to either Ethernet or IEEE 802.3 local area networks. The Model 840 supports both AdvanceNet, HP's proprietary networking strategy, for communication with HP systems, and Berkeley/ARPA services for communication with other machines using a UNIX operating system. The LAN card is armed to interrupt whenever a packet arrives, and is then polled to move the packets into the system.

For users who need to connect to 16-bit general-purpose parallel devices, the HP 27114A is supported on the Model 840. This card supports both single-ended and differential input/output lines and has three sense and three control lines. The DIL library is also supported for this interface, giving the user low-level control of the HP 27114A in a standard manner.

Software Control Flow

To show how the I/O system works on the Model 840, the process for generating, booting, and using the the I/O system will be described, along with an example of how a disc transaction would flow through the system.

The I/O system for the Model 840 is configured with a simple kernel building process. A C-like description of the hardware and software hierarchy is entered as input to the build program, uxgen. The user includes all of the devices that may be connected, even though some of the devices may not be present at any given boot time. Uxgen creates a table which the boot process uses to configure the I/O system. References to all the managers required are generated, and a kernel is linked containing all of the necessary software, all in just a few minutes. The user moves the new kernel to the root partition, / directory and reboots the system.

The system is brought into memory and begins to configure the I/O system and perform other tasks required to make the system functional. The I/O configuration software creates the software hierarchy, calling each instance of a manager and allocating any PDA space that it needs. Managers that cannot find their associated hardware log an error to the diagnostic system and then go into an idle state. The manager attempts to configure the hardware again on a user request for devices or on a powerfail recovery for device adapter cards. As soon as a path from the hardware

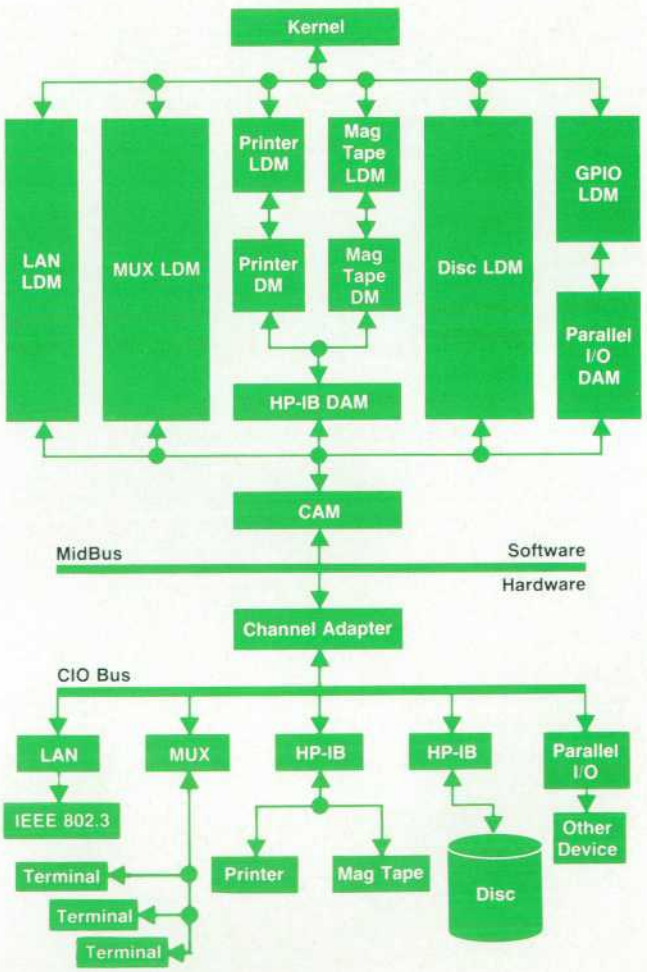


Fig. 15. *HP-UX I/O system structure for the HP 9000 Model 840 Computer.*

to the kernel interface is configured, that path becomes available for I/O transactions.

Request Initiation

A request is initiated by a user process. The LDM checks the validity of the user's request, locks down the affected pages in memory, and performs any queuing that may need to be done.

In the case of the disc manager's performing a user request directly to the disc, the buffer address is checked for validity, the pages are locked in memory so that they cannot be paged out, the disc addresses are computed, and the request is inserted into the disc queue. When the request gets to the head of the queue, a message is sent to the CAM, giving a request chain and the address of the device adapter that the request is for.

The CAM has four major tasks to perform for each request. It translates all of the virtual addresses to real addresses for the hardware, flushes all of the request buffers out of the cache, guarantees the alignment of buffers so that DMA does not affect data outside of the user request, and initiates the request on the hardware.

The CAM builds a request chain paralleling that sent down by the calling driver. The requesting manager sends all of its requests down with virtual addresses. The CAM builds a similar list, but all of the addresses are in real mode, which the hardware understands. In the Model 840 implementation, buffers must be 32-byte aligned, so if the buffer is not aligned, the CAM allocates a 32-byte buffer, called a buflet, for temporary storage. A separate request block is built for each page of the request, since virtually contiguous pages may be physically separated. Finally, a buflet is allocated to cover the last bytes of the transaction if the buffer does not end on a 32-byte boundary. On a read operation, the data is transferred by DMA into the buffer addresses in the real chain. Data in buflets must be copied back to the user's buffer after DMA is completed on reads, and copied to the buflets before DMA starts on writes. Data in the aligned buffers goes directly into the buffer.

Finally, the CAM starts the transaction on the hardware and returns. The LDM returns to the kernel, the requesting user process is put into a sleeping state, and the kernel causes a switch to another process which will run.

DMA Transaction

Fig. 16 shows an example of a DMA transaction and how the CAM changes it before it is started on the hardware. The first quad[23] in the chain (the write) is already 32-byte aligned, so the CAM simply translates the DMA buffer address to a real address. Note that the DMA will be done directly from the buffer passed down to the CAM. The second quad in the chain (the read) is not 32-byte aligned. The CAM must divide this quad into several new quads to ensure correct buffer alignment. The first quad the CAM allocates is 8 bytes long and will go into a CAM-allocated buflet. Effectively, this aligns the rest of the buffer on a 32-byte boundary. The next buffer is 672 bytes long and accounts for the rest of this page. The next buffer is 2048 bytes long (a page). The next buffer is 1344 bytes long and includes all of the remaining buffer until the last 32-byte boundary. The final buffer is 24 bytes long and points to a buflet. This preserves the end of the user's buffer. Finally, the last quad in the chain is processed; it is 32-byte aligned and therefore only an address translation is required.

Interrupt Servicing

When the hardware has finished processing a request, an interrupt is signaled and the CAM is called to complete the transaction processing. On request completion, the CAM copies any buflets that need to be copied back to user space, the real chain elements are returned to a pool, and a reply message is sent to the requesting manager.

The LDM cleans up its resources, sends a wakeup to the process that requested the data, and wakes up its initiation section to start any queued requests.

Powerfail Recovery

After power has returned, the I/O system begins to recover by sending a message to the CAM notifying the lowest-level manager of the power failure. The CAM in turn sends a message to each of the managers above it with the same notification. Each manager notifies the managers above it that power has failed, and replies to managers to acknowledge receipt of the powerfail message. Messages move up through the system, against the flow of normal requests. A lower manager will return every message it

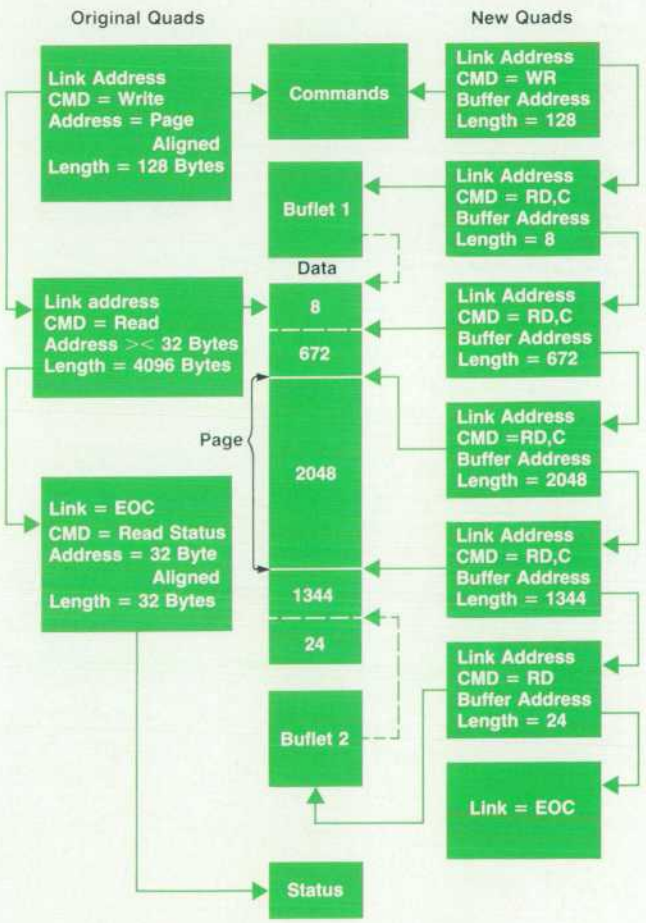


Fig. 16. *An example of a DMA transaction, showing how the channel adapter manager changes it before it is passed on to the hardware.*

sees from a higher manager with a powerfailed status until it sees a powerfail reply. In that way, all queued messages are cleared out, and each manager can then reinitialize its piece of hardware.

Error Logging and Diagnostics

In the case of hard or frequent soft errors, the manager sends a log message to a special diagnostic port. A user process reads the error messages from the diagnostic port and logs them to both the console and a file.

To facilitate hardware fault diagnosis, special hooks were added to the Model 840 I/O managers. Unlike traditional systems, this design has a set of read, write, and ioctl calls built into a diagnostic section of each manager. When the manager is opened with a special mode bit set, controlled by file security, the diagnostic mode is enabled. In this mode, card status can be queried, cards can be reset, and special diagnostic sequences can be sent down to the device.

Conclusions

HP Precision Architecture is the foundation for a full product line of machines running the HP-UX operating system. Many features of the architecture, combined with accommodations by the operating system, have led to a high-performance system able to support the *UNIX System V Interface Definition* and the *HP-UX Standard Specification*.

Acknowledgments

We'd like to acknowledge Martin Liu, who led the initial effort for porting HP-UX to HP Precision Architecture. Julie Banks, Karen Barnes, Steve Boettner, Don Bollinger, Larry Dwyer, Judy Guist, Doug Hartman, Jim Hays, Gregg Kellogg, David Lennert, Michael Mahon, Dave Snow, Bruce Thompson, and Kevin Wallace were major contributors to the implementation of HP-UX on HP Precision Architecture. Thanks to our own Bill Bryg and the other architects for their work in creating these innovative concepts. Linda Kyrnitzke and Dan Epstein provided invaluable aid in the preparation of this article. Finally, we extend our emphatic thanks to all the engineers in the HP-UX laboratory. Without their outstanding efforts, the accomplishments detailed in this article would not have been possible.

References

1. M.L. Connor, "What is UNIX?," *Hewlett-Packard Journal*, Vol. 35, no. 3, March 1984, pp. 9.
2. S.R. Bourne, *The UNIX System*, Addison-Wesley, 1983.
3. M.G. Sobell, *A Practical Guide to UNIX System V*, Benjamin/Cummings, 1985.
4. H. McGilton and R. Morgan, *Introducing the UNIX System*, McGraw-Hill, 1983.
5. B.W. Kernighan and R. Pike, *The UNIX Programming Environment*, Prentice-Hall, 1984.
6. B.W. Kernighan and D.M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978.
7. A.G. Anderson, et al, "New HP-UX Features for HP 9000 Series 300 Workstations," *Hewlett-Packard Journal*, Vol. 37, no. 7, July 1986, pp. 34-41.
8. S.W.Y. Wang and J.B. Lindberg, "HP-UX: Implementation of UNIX on the HP 9000 Series 500 Computer Systems," *Hewlett-Packard Journal*, Vol. 35, no. 3, March 1984, pp. 7-15.
9. T.J. Williams and N.A. Mills, "A Multitasking Personal Computer System for the Technical Professional," *Hewlett-Packard Journal*, Vol. 36, no. 10, October 1985, pp. 4-6.
10. R.M. Fajardo, et al, "A UNIX Operating System Adapted for a Technical Personal Computer," *ibid*, pp. 22-28.
11. J.A. Brewster, et al, "A Friendly UNIX Operating System User Interface," *ibid*, pp. 28-33.
12. M.V. Hetrick, "HP-UX: A Corporate Strategy," *Hewlett-Packard Journal*, Vol. 35, no. 3, March 1984, pp. 12-13.
13. J.S. Birnbaum and W.S. Worley, "Beyond RISC: High-Precision Architecture," *Hewlett-Packard Journal*, Vol. 36, no. 8, August 1985, pp. 4-10.
14. M.V. Hetrick and M.L. Kolesar, "A New 32-Bit VLSI Computer Family: Part II—Software," *Hewlett-Packard Journal*, Vol. 35, no. 3, March 1984, pp. 3-6.
15. J.E. Bale and H.E. Kellogg, "Native Language Support for Computer Systems," *Hewlett-Packard Journal*, Vol. 36, no. 6, June 1985, pp. 27-32.
16. J.L. Bidwell and D.W. Palermo, "Advanced Multilingual Computer Systems for Measurement Automation and Computer-Aided Engineering Applications," *Hewlett-Packard Journal*, Vol. 33, no. 5, May 1982, pp. 3-7.
17. K.S. Barnes, and D.S. Epstein, "Changing Character: Technical Problems in the Multilingual Support of UNIX," *UNIX Review*, December 1985.
18. R.M. Lenk, "Real-Time Functionality in HP-UX," *TC Interface*, Vol. 5, no. 1, January/February 1986.
19. S. Davey, "The HP Precision Program Software Quality Plan," *Hewlett-Packard Journal*, to be published.
20. M. Mahon, et al, "Hewlett-Packard Precision Architecture: The Processor," *Hewlett-Packard Journal*, Vol. 37, no. 8, August 1986.
21. O. Babaoglu, W. Joy, and J. Porcar, *Design and Implementation of the Berkeley Virtual Memory Extensions to the UNIX Operating System*, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California at Berkeley, December 10, 1979.
22. R.W. Carr and J.L. Hennesy, *WSClock—A Simple and Effective Algorithm for Virtual Memory Management*, ACM 0-89791-062-1-12/81-0087, December 1981.
23. D.V. James, S.G. Burger, and R.D. Odineal, "Hewlett-Packard Precision Architecture: The Input/Output System," *Hewlett-Packard Journal*, Vol. 37, no. 8, August 1986.

HEWLETT-PACKARD JOURNAL

Volume 37 January 1986 through December 1986

Index

Hewlett-Packard Company, 3200 Hillview Avenue, Palo Alto, California 94304 U.S.A.
Hewlett-Packard Central Mailing Dept., P.O. Box 529, Startbaan 16, 1180 AM Amstelveen, The Netherlands
Hewlett-Packard (Canada) Ltd., 6877 Goreway Drive, Mississauga, Ontario L4V 1M8 Canada
Yokogawa-Hewlett-Packard Ltd., Suginami-ku, Tokyo 168 Japan

PART 1: Chronological Index

January 1986

Compilers for the New Generation of Hewlett-Packard Computers, *Deborah S. Coutant, Carol L. Hammond, and Jon W. Kelley*
Components of the Optimizer
An Optimization Example
A Stand-Alone Measurement Plotting System, *Thomas H. Daniels and John Fenoglio*
Eliminating Potentiometers
Digital Control of Measurement Graphics, *Steven T. Van Voorhis*
Measurement Graphics Software, *Francis E. Bockman and Emil Maghakian*
Analog Channel for a Low-Frequency Waveform Recorder, *Jorge Sanchez*
Usability Testing: A Valuable Tool for PC Design, *Daniel B. Harrington*

February 1986

Gallium Arsenide Lowers Cost and Improves Performance of Microwave Counters, *Scott R. Gibson*
Creating Useful Diagnostics
Manufacturing Advances
A New Power Transformer
Optimum Solution for IF Bandwidth and LO Frequencies in a Microwave Counter, *Luiz Peregrino*
Seven-Function Systems Multimeter Offers Extended Resolution and Scanner Capabilities, *Scott D. Stever, Joseph E. Mueller, Thomas G. Rodine, Douglas W. Olsen, and Ronald K. Tuttle*
Advanced Scalar Analyzer System Improves Precision and Productivity in R&D and Production Testing, *Jacob H. Egbert, Keith F. Anderson, Frederic W. Woodhull II, Joseph Rowell, Jr., Douglas C. Bender, Kenneth A. Richter, and John C. Faick*
Filter Measurement with the Scalar Network Analyzer
Scalar Analyzer System Error Correction
Calibrator Accessory
Voltage-Controlled Device Measurements

March 1986

An Introduction to Hewlett-Packard's AI Workstation Technology, *Martin R. Cagan*
HP's University AI Program
A Defect Tracking System for the UNIX Environment, *Steven R. Blair*
A Toolset for Object-Oriented Programming in C, *Gregory D. Burroughs*
Tools for Automating Software Test Package Execution, *Craig D. Fuget and Barbara J. Scott*
Using Quality Metrics for Critical Application Software, *William T. Ward*
P-PODS: A Software Graphical Design Tool, *Robert W. Dea and Vincent J. D'Angelo*
Triggers: A Software Testing Tool, *John R. Bugarin*
Hierarchy Chart Language Aids Software Development, *Bruce A. Thompson and David J. Ellis*
Module Adds Data Logging Capabilities to the HP-71B Computer, *James A. Donnelly*
System Monitor Example

April 1986

A Data Acquisition System for a 1-GHz Digitizing Oscilloscope, *Kenneth Rush and Danny J. Oldfield*
General-Purpose 1-GHz Digitizing Oscilloscopes
High-Performance Probe System for a 1-GHz Digitizing Oscilloscope, *Kenneth Rush, William H. Escovitz, and Arnold S. Berger*
Waveform Graphics for a 1-GHz Digitizing Oscilloscope, *Rodney T. Schlater*
Hardware Implementation of a High-Performance Trigger System, *Scott A. Genther and Eddie A. Evel*
1-GHz Digitizing Oscilloscope Uses Thick-Film Hybrid Technology, *Derek E. Toeppen*
A Modular Power Supply, *Jimmie D. Felps*
Program Helps Teach Digital Microwave Radio Fundamentals, *Christen K. Pedersen*

May 1986

Low-Cost Automated Instruments for Personal Computers, *Charles J. Rothschild, 3rd, Robert C. Sismilich, and William T. Walker*
PC Instruments Modules
Instrumentless Front-Panel Program Demonstrates Product Concept
Versatile Microcomputer is Heart of PC Instruments Oscilloscope Module
Mechanical and Industrial Design of the PC Instruments Cabinet
PCIB: A Low-Cost, Flexible Instrument Control Interface for Personal Computers, *William L. Hughes and Kent W. Luehman*
A Custom HQMOS Bus Interface IC
Interactive Computer Graphics for Manual Instrument Control, *Robert C. Sismilich and William T. Walker*
Mouse in Danger: Managing Graphics Objects
Oscilloscope Software Leverages Previous Concepts and Algorithms
Automated Testing of Interactive Graphics User Interfaces
Industrial Design of Soft Front Panels
HP-IB Command Library for MS-DOS Systems, *David L. Wolpert*
Case Study: PC Instruments Counter Versus Traditional Counters, *Edward Laczynski and Robert V. Miller*
Reciprocal Counting in Firmware
Salicide: Advanced Metallization for Submicrometer VLSI Circuits, *Jun Amano*

June 1986

Integrated Circuit Procedural Language, *Jeffrey A. Lewis, Andrew A. Berlin, Allan J. Kuchinsky, and Paul K. Yip*
Knowledge-Assisted Design and the Area Estimation Assistant
Software Development for Just-in-Time Manufacturing Planning and Control, *Raj K. Bhargava, Teri L. Lombardi, Alvina Y. Nishimoto, and Robert A. Passell*
Comparing Manufacturing Methods
The Role of Doppler Ultrasound in Cardiac Diagnosis, *Raymond G. O'Connell, Jr.*
Doppler Effect: History and Theory, *Paul A. Magnin*
Johann Christian Doppler
Power and Intensity Measurements for Ultrasonic Doppler Imaging Systems, *James Chen*
Extraction of Blood Flow Information Using Doppler-Shifted

Ultrasound, *Leslie I. Halberg and Karl E. Thiele*
Continuous-Wave Doppler Board
Observation of Blood Flow and Doppler Sample Volume
Modifying an Ultrasound Imaging Scanner for Doppler Measurements, *Sydney M. Karp*
Digital Processing Chain for a Doppler Ultrasound Subsystem, *Barry F. Hunt, Steven G. Leavitt, and David C. Hempstead*

July 1986
Design of HP's Portable Computer Family, *John T. Eaton, Carl B. Lantz, Clifford B. Cordy, Jr., James W. Pearson, Michael J. Barbour, Courtney Loomis, and Ella M. Duyck*
Inside the LCDs for The Portable and Portable Plus
Low-Power Modes for Portable Computers
Hollow Studs for Package Assembly
I/O and Data Communications in Portable Computers, *Andrew W. Davidson and Harold B. Noyes*
Personal Applications Manager for HP Portable Computers, *Robert B. May and Alesia Duncombe*
Memory Management for Portable Computers, *Mark S. Rowe*
A Hybrid Solution for a 25-Line LCD Controller, *Glenn J. Adler*
Creating Plug-in ROMs for the Portable Plus Computer, *William R. Frolik*
Structure of a Plug-In ROM
New HP-UX Features for HP 9000 Series 300 Workstations, *Andrew G. Anderson, David L. Frydendall, Robert D. Gardner, Robert M. Lenk, Robert J. Schneider, Bonnie Dykes Stahlin, and Ronald G. Tolley*
A Protocol Analyzer for Local Area Networks, *Gordon A. Jensen, Stephen P. Reames, Jerry D. Morris, Jeffrey H. Smith, Jeffrey Tomberlin, and James M. Umphrey*

August 1986
Hewlett-Packard Precision Architecture: The Processor, *Michael J. Mahon, Ruby Bei-Loh Lee, Terrence C. Miller, Jerome C. Huck, and William R. Bryg*
Floating-Point Coprocessor
HP Precision Architecture Caches and TLBs
Hewlett-Packard Precision Architecture: The Input/Output System, *David V. James, Stephen G. Burger, and Robert D. Odineal*
Hewlett-Packard Precision Architecture Performance Analysis, *Joseph A. Lukes*
The HP Precision Simulator, *Daniel J. Magenheimer*
Remote Debugger

September 1986
Advanced Modular Engineering Workstations, *Gilbert I. Sandberg, Daryl E. Knoblock, John C. Keith, Michael K. Bowen, and Ronald P. Dean*
Modular Computer Low-End Processor Board Design, *Martin L. Speer and Nicholas P. Mati*
High-Performance SPU for a Modular Workstation Family, *Jonathan J. Rubinstein*
Custom VLSI Circuits for Series 300 Graphics, *James A. Brokish, David J. Hodge, and Richard E. Warner*
Display Custom IC Design Methodology
Software Compatibility for Series 200 and Series 300 Computers, *Rosemarie Palombo*
Implementing a Worldwide Electronic Mail System, *Luis Hurtado-Sanchez, Amy Tada Mueller, Robert A. Adams, Kristy Ward Swenson, and Rebecca A. Dahlberg*

October 1986
Hewlett-Packard and the Open Systems Interconnection Reference Model, *Gertrude G. Reusser and Donald C. Loughry*
HP AdvanceNet: A Growth-Oriented Computer Networking Architectural Strategy, *Robert J. Carlson, Atul Garg, Arie Scope, Craig Wassenberg, and Lyle A. Weiman*
Network Services and Transport for the HP 3000 Computer, *Kevin J. Faulkner, Charles W. Knouse, and Brian K. Lynn*
A Local Area Network for HP Computers, *Tonia G. Graham and Charles J. de Sostoa*
Network Services for HP Real-Time Computers, *David M. Tribby*
Networking Services for HP 9000 Computers, *J. Christopher Fugitt and Dean R. Thompson*
Connecting NS/9000 and NS/3000
Leaf Node Architecture
X.25 Wide Area Networking for HP Computers, *Pierry Mettetal*
DMI/3000: A Move Toward Integrated Communication, *Nancy L. Navarro, Deepak V. Desai, and Timothy C. Shafer*
Glossary of DMI Terms
Companies Supporting the DMI Standard

November 1986
Molecular-Scale Engineering of Compound Semiconductor Materials, *Douglas M. Collins*
Compound Semiconductor Alloys and Heterojunctions
The Modulation-Doped Heterojunction
Extending Millimeter-Wave Diode Operation to 110 GHz, *Eric R. Ehlers, Sigurd W. Johnsen, and Douglas A. Gray*
26.5-to-40-GHz Waveguide Detector
Diode Integrated Circuits for Millimeter-Wave Applications, *Mark P. Zurakowski, Domingo A. Figueredo, Scott S. Elliott, George A. Patterson, William J. Anklam, and Susan R. Sloan*
Unbiased Subharmonic Mixers for Millimeter-Wave Spectrum Analysis, *Robert J. Matreci*
Predictive Support: Anticipating Computer Hardware Failures, *David B. Wasmuth and Bruce J. Richards*
Systems Design for Worldwide Delivery of Customer Support
Logging Event Data in the Trend Log
AIDA: An Expert Assistant for Dump Readers, *Lynn R. Slater, Jr., Keith A. Harrison, and Craig M. Myles*
What Is a Memory Dump?
A Troubleshooting Aid for Asynchronous Data Communications Links, *Brian T. Button, R. Michael Young, and Diane M. Ahart*
Hierarchies
A Rule-Based System to Diagnose Malfunctioning Computer Peripherals, *George R. Gottschalk and Roy M. Vandoorn*
Multilevel Constraint Based Configuration, *Robert I. Marcus*

December 1986
The HP-UX Operating System on HP Precision Architecture Computers, *Frederick W. Clegg, Gary Shiu-Fan Ho, Steven R. Kusmer, and John R. Sontag*
A UNIX System V Compatible Implementation of 4.2BSD Job Control
Decreasing Real-Time Process Dispatch Latency Through Kernel Preemption
Data Base Management for HP Precision Architecture Computers, *Alan S. Brown, Thomas M. Hirata, Ann M. Koehler, Krishnan Vishwanath, Jenny Ng, Michael J. Pechulis, Mark A. Sikes, David E. Singleton, and Judson E. Veazey*
Data Storage in ALLBASE

PART 2: Subject Index

Subject	Month
A	
Access control	Aug.
Access, data base	Dec.
Active probe	Apr.
Address resolution	Oct.
Addressing model, HP Precision	Aug.
AdvanceNet	Oct.
AIDA	Nov.
AI Workstation	Mar.
Algorithm, averaging	Apr.

Algorithm, configuration ..... Nov.
Algorithm, high-speed plotting ..... Apr.
Aliasing ..... Jan.
June
ALLBASE ..... Dec.
Analog channel, low-frequency waveform recorder ..... Jan.
Analog-to-digital converter, DMM .... Feb.
Analog-to-digital converter, network analyzer ..... Feb.
Analog-to-digital converter, oscilloscope ..... Apr.
Analyzer, LAN protocol ..... July
Analyzer, scalar network ..... Feb.
Application layer, OSI ..... Oct.
Architecture, leaf node ..... Oct.
Architecture, RISC ..... Jan.
Aug.
Area estimation, VLSI design ..... June
ARPA TCP/IP ..... Oct.
Artificial intelligence ..... Mar.
Artificial intelligence applications, computer support ..... Nov.
Assist hardware and instructions .... Aug.
Attached processors ..... Aug.
Authorization control, DBMS ..... Dec.
Automation tool, Scaffold ..... Mar.
Averaging algorithm ..... Apr.

B

Backward chaining rules ..... Nov.
Basic block analysis ..... Jan.
BASIC, data acquisition ROM ..... Mar.
Battery supply, lead-acid ..... July
Beam lead bonding ..... Apr.
Nov.
Beliefs ..... Nov.
Benchmarks ..... Aug.
Berkeley 4.2BSD ..... Dec.
Bills of material ..... June
Blood flow, Doppler measurement June
Bonds, wedge and stitch ..... Apr.
Branching ..... Aug.
Browsing, AI Workstation ..... Mar.
Bss ..... Dec.
B-tree indexes ..... Dec.
Buffer requirements, LAN receiver ... Oct.
Buffering, user control ..... Dec.
Bus adapter ..... Aug.
Bus converter ..... Aug.

C

Cache architecture, Series 300 ..... Sept.
Cache management, HP-UX ..... Dec.
Cache memories ..... Aug.
Cache simulation ..... Sept.
Calibration RAM protection ..... Feb.
Calibrator, scalar network analyzer Feb.
Caller/callee-saves registers ..... Jan.
Cardiac imaging ..... June
Certainty ratios ..... Nov.
Chaining, rules ..... Nov.
Channel (bus) adapter ..... Aug.
Dec.
Clock algorithm ..... Dec.
Coaxial switches, RF ..... Apr.
Code expansion, RISC ..... Jan.
Code generation, RISC ..... Jan.
Color map IC ..... Sept.
Columns ..... Dec.
Combined instructions ..... Aug.
Communications, digital microwave radio ..... Apr.
Compatibility, Series 200 and Series 300 ..... Sept.
Compilers, RISC ..... Jan.
Compound semiconductors, fabrication ..... Nov.
Computer dump reader ..... Nov.
Computer, HP 9000 Model 840 ..... Dec.
Computer, HP 9000 Series 930 ..... Dec.
Computer support ..... Nov.
Computers, portable ..... July
Concurrency control ..... Dec.
Conditional branching ..... Aug.
Configurator, computer systems ..... Nov.
Connect protocol, HP Precision ..... Aug.
Constellation diagram ..... Apr.
Constraint-based configuration ..... Nov.
Continuous-wave Doppler ..... June
Control flow model, HP Precision ... Aug.
Corruption detection, memory ..... Nov.
Cost of a test ..... Nov.
Counter, fast reloading ..... Apr.
Counter, PC Instruments ..... May
Counters, microwave ..... Feb.

D

Daemons, HP-UX ..... Dec.
Data access module ..... Oct.
Data acquisition, handheld computer ..... Mar.
Data acquisition, oscilloscope ..... Apr.
Data base management ..... Dec.
Data communications link troubleshooter ..... Nov.
Data communications, portable computer ..... July
Data entries ..... Dec.
Data items ..... Dec.
Data link layer, OSI ..... Oct.
Data sets ..... Dec.
Data structure macros ..... Mar.
Data types, HP Precision ..... Aug.
DBCore ..... Dec.
DBE Fileset ..... Dec.
DBEnvironment ..... Dec.
DBMS ..... Dec.
Defect tracking system, UNIX ..... Mar.
Delayed triggering ..... Apr.
Demand-pull stock flow ..... June
Detector card, Doppler-shifted ultrasound ..... June
Detectors, microwave ..... Feb.
Device I/O, HP-UX 5.0 ..... July
Diagnostic systems, AI ..... Mar.
Nov.
Digital microwave radio tutorial program ..... Apr.
Digital multimeter ..... Feb.
Digital Multiplexed Interface (DMI) Oct.
Digital processing, Doppler ultrasound ..... June
Digitizing oscilloscopes ..... Apr.
Diode, millimeter-wave ..... Nov.
Diode, modified barrier ..... Nov.
Dispatch table ..... Mar.
Display compatibility, Series 200 and Series 300 ..... Sept.
Display controller IC ..... Sept.
DMA, HP-UX ..... Dec.
DMA module, HP Precision ..... Aug.
DMM ..... Feb.
DMM, PC Instruments ..... May
Dopant redistribution, $TiSi_2$ ..... May
Doppler, Johann Christian ..... June
Doppler theory ..... June
Doppler ultrasound imaging ..... June
Drivers, I/O, HP-UX ..... Dec.
Drivers, PC Instruments ..... May
Dual-path triggering ..... Apr.
Dual-rail data ..... Oct.
Dump reader ..... Nov.
Dynamic coefficients, generation ..... June

E

Editing, AI Workstation ..... Mar.
Electronic mail, implementation .... Sept.
Engineering workstations ..... Sept.
Environments, networking ..... Oct.
Error correction, network analyzer ... Feb.
Error logging, HP-UX ..... Dec.
Error monitoring, predictive ..... Nov.
Estimation, system performance ..... Aug.
Ethernet LANs ..... Oct.
Ethernet protocol analyzer ..... July
Events, failure prediction ..... Nov.
Execution model, HP Precision ..... Aug.
Expert configurator ..... Nov.
Expert dump reader ..... Nov.
Expert troubleshooting systems ..... Nov.
External file transfer ..... Sept.
Eye diagram ..... Apr.

F

Failures, prediction of ..... Nov.
Fast Fourier transform, Doppler detection ..... June
Faults ..... Nov.
FFT, Doppler detection ..... June
File locking ..... Dec.
File system, HP-UX ..... Dec.
Filter effects, microwave radio ..... Apr.
Filter measurements ..... Feb.
Filters, LAN frame ..... July
Filters, wall and Nyquist ..... June
Flight planner/simulator, AI ..... Mar.
Floating-point coprocessor ..... Aug.
Foreign service connection ..... Sept.
Formatting, memory dump ..... Nov.
Forward chaining rules ..... Nov.
Frames, Ethernet/IEEE 802.3 ..... July
Oct.
Framing ..... Oct.

G

GaAs devices, millimeter-wave ..... Nov.
Gallium arsenide sampler ..... Feb.
Gather write ..... Dec.
Gatherer ..... Jan.
Graphics, automated testing ..... May

Graphics, digital microwave radio ... Apr.
Graphics, display subsystem ... Sept.
Graphics, interactive, instrument control ... May
Graphics, managing objects ... May
Graphics, oscilloscope ... Apr.
Graphics software, measurement ... Jan.

H

Hash indexes ... Dec.
HDLC ... Oct.
Heap ... Dec.
Heterojunction devices ... Nov.
Heuristic test selection ... Nov.
Hierarchical I/O system ... Dec.
Hierarchical model, data base ... Dec.
Hierarchies ... Nov.
Hierarchy chart language ... Mar.
Hole, printed-through ... Apr.
Hollow studs, package assembly ... July
HP DeskManager, HP system ... Sept.
HP-HIL, keyboard ... Sept.
HP-IB, command library, MS-DOS ... May
HPIMAGE ... Dec.
HP JIT ... June
HP Precision Architecture ... Jan. Aug. Dec.
HP-RL ... Mar.
HPSQL ... Dec.
HP-UX operating system and DBMS Dec.
HP-UX 5.0 operating system, Series 300 ... July
HPWindows/9000, HP-UX 5.0 ... July
HQMOS, bus interface IC ... May
Hybrid circuit, LCD controller ... July
Hybrids, oscilloscope ... Apr.
Hydrophone, calibration ... June

I

IC advisor, AI ... Mar.
IC, bus interface ... May
ICPL, integrated circuit procedural language ... June
Ideality factor, barrier diodes ... Nov.
ID module, Series 300 ... Sept.
IEEE 802.3 LANs ... Oct.
IEEE 802.3 protocol analyzer ... July
IEEE P1003 ... Dec.
IF bandwidth, counter, optimum ... Feb.
Immediates, HP Precision ... Aug.
Industrial design, PC Instruments ... May
Industrial design, soft front panels ... May
Inference engine ... Nov.
Infinite persistence ... Apr.
In-phase modulation ... Apr.
Input/output system, HP-UX ... July Dec.
Instruction distributions ... Aug.
Instructions, HP Precision ... Aug.
Instrument control, AI ... Mar.
Instruments, personal computers ... May
Integrated Services Digital Network Oct.
Intelligent Peripheral Troubleshooter (IPT) ... Nov.
Intensity measurement, Doppler ultrasound ... June
Interface IC ... May
Interfaces, portable computer ... July
Interfacing, AI Workstation ... Mar.
Interpolator, oscilloscope ... Apr.
Interprocess communication ... Oct.
Interrupt groups hardware ... Aug.
Interrupt servicing, HP-UX ... Dec.
Interruptions, HP Precision Architecture ... Aug.
Interval analysis ... Jan.
Inventory control, JIT ... June
I/O architecture, HP Precision ... Aug.
I/O dependent code ... Aug.
I/O, device, HP-UX 5.0 ... July
I/O, PC Instruments ... May
I/O, portable computers ... July
I/O services, HP-UX ... Dec.
IP (internet protocol) ... Oct.
I•Q Tutor ... Apr.
IQUERY ... Dec.
ISDN ... Oct.
ISO OSI model ... Oct.

J

Jabbering frames ... July
JIT (just-in-time) manufacturing software ... June
Job control, HP-UX ... Dec.

K

Kernel, HP-UX ... Dec.
Keyboard compatibility, Series 200 and Series 300 ... Sept.
Knowledge-assisted design ... June
Knowledge base ... Nov.
Knowledge representation ... Nov.

L

Language cap, PC Instruments ... May
LANIC ... Oct.
LAN protocol analyzer ... July
LANs ... Oct.
LAP-B ... Oct.
LAP-D ... Oct.
LCD controller ... July
Leaf node architecture ... Oct.
LESS machine ... Aug.
Levels, constraint ... Nov.
Limit testing ... Feb.
Linear programming solution ... Feb.
Linkage registers ... Jan.
Link-level access ... Oct.
Liquid-crystal display, portable computer ... July
Lisp ... Mar.
Lisp, ICPL ... June
Local area networks ... Oct.
Localization, HP-UX ... July Dec.
Localization, PAM ... July
Lock modes, DBMS ... Dec.
LO frequencies, counter, optimum ... Feb.
Log, trend ... Nov.
Logarithmic amplifier ... Feb.
Logging, DBMS ... Dec.
Long-pointer addressing ... Aug.
Low-power modes ... July

M

M/A-COM ... Oct.
Managers, I/O ... Dec.
Managing, AI Workstation ... Mar.
Manufacturing software, just-in-time ... June
Material requirements planning, JIT June
MBE, molecular beam epitaxy ... Nov.
Measurement graphics software (MGS) ... Jan.
Mechanical design, PC Instruments May
Mechanical design, portable computer ... July
Media access unit ... July
Medical instruments, Doppler ultrasound imaging ... June
Medical software, testing ... Mar.
Medium attachment unit ... Oct.
Memory dump reader ... Nov.
Memory management, HP-UX ... Dec.
Memory management, portable computer ... July
Memory management, Series 300 ... Sept.
Memory mapped I/O ... Aug. Dec.
Messages, HP-UX ... Dec.
Metallization, IC ... May
Metrics, software quality ... Mar.
MicroScope ... Mar.
Microwave counters ... Feb.
Microwave radio tutorial program ... Apr.
Migration analysis utility (MAU) ... Dec.
Migration, data base ... Dec.
Migration, HP-UX ... Dec.
Millicode ... Jan.
Millimeter-wave devices ... Nov.
MIPS computation ... Aug.
Mixers, millimeter-wave ... Nov.
Model, addressing and protection ... Aug.
Model, communications system ... Apr.
Model, control flow ... Aug.
Model, execution ... Aug.
Model, thick-film resistor ... Apr.
Modems, portable computer ... July
Modular computers ... Sept.
Modules, I/O ... Aug.
MPE XL DBMS ... Dec.
MS-DOS, HP-IB command library ... May
Multilevel constraints ... Nov.
Multimeter, systems ... Feb.
Multipath impairments ... Apr.
Multiple test environments ... Mar.
Multiplexer, oscilloscope probe ... Apr.
Mycon ... Nov.

N

Native language support, HP-UX ... Dec.
Native language support, HP-UX 5.0 July
Natural language understanding system, AI ... Mar.
Network analyzer, scalar ... Feb.
Network file transfer ... Oct.
Network, HP electronic mail ... Sept.
Network layer, OSI ... Oct.
Network model, data base ... Dec.

Network protocol analyzer ..... July
Network Services, HP 1000 ..... Oct.
Network Services, HP 3000 ..... Oct.
Network Services, HP 9000 ..... Oct.
Networking strategy, HP ..... Oct.
Networks, local area ..... Oct.
Networks, wide area ..... Oct.
Nodal management ..... Oct.
Noise degradation, microwave radio Apr.
Noise rejection, DMM ..... Feb.
Nonlinearities, microwave radio ..... Apr.
Nullification ..... Aug.

O

Object-oriented programming ..... Mar.
Object-oriented programming toolset, C ..... Mar.
Observables ..... Nov.
One-server model ..... Oct.
Open systems interconnection ..... Oct.
Operating system, HP-UX ..... Dec.
Operations, HP Precision ..... Aug.
Optimizing compilers ..... Jan.
Optimum IF and LO, counter ..... Feb.
Oscilloscope, PC Instruments ..... May
Oscilloscopes, digitizing ..... Apr.
Oxygen redistribution, $TiSi_2$ ..... May

P

Packet switched networks ..... Oct.
Paging management ..... Aug.
PAM, Personal Applications Manager, portable computer ..... July
PANELS program, PC Instruments ... May
Parallel communications channel, PCIB ..... May
Parent-child relationships ..... Dec.
Patching ..... Jan.
Path reports ..... Oct.
Paths, DBMS ..... Dec.
Paths, protocol ..... Oct.
Patient care software, testing ..... Mar.
PBX-based communication ..... Oct.
PC design, testing ..... Jan.
PCIB ..... May
PC Instruments ..... May
Performance analysis methods ..... Aug.
Performance model, JIT software ..... June
Peripheral processor unit (PPU), portable computer ..... July
Peripheral troubleshooter ..... Nov.
Persistence, variable ..... Apr.
Personal Applications Manager, portable computer ..... July
Phase formation, $TiSi_2$ ..... May
Physical layer, OSI ..... Oct.
Plotting algorithm ..... Apr.
Plotting system, measurement ..... Jan.
Portable computers ..... July
Portable Plus ..... July
Port/HP-UX ..... Dec.
Ports ..... Oct.
Postamplifier, oscilloscope ..... Apr.
Post-deduct transaction ..... June
Potentiometer elimination ..... Jan.
Power measurement, Doppler ultrasound ..... June
Power modes, portable computer ..... July
Power supply, oscilloscope ..... Apr.
Power transformer ..... Feb.
Powerfail recovery ..... Dec.
P-PODS ..... Mar.
Preallocation of disc space ..... Dec.
Preamplifier, oscilloscope ..... Apr.
Precision Architecture, HP ..... Aug.
Predictive support ..... Nov.
Preemption latency, HP-UX ..... Dec.
Presentation layer, OSI ..... Oct.
Privileged groups, HP-UX system ..... July
Probabilities, expert systems ..... Nov.
Probe hybrids ..... Apr.
Probe system, oscilloscope ..... Apr.
Procedure calls ..... Jan.
Process model, UNIX ..... Dec.
Process scheduling, HP-UX ..... Dec.
Process status word ..... Aug.
Process synchronization, HP-UX ..... Dec.
Processing, GaAs ICs ..... Nov.
Processing, IC ..... May
Processor architecture ..... Aug.
Processor board, 10-MHz, 68010 .... Sept.
Processor board, 16.67-MHz, 68020 Sept.
Product design, Series 300 ..... Sept.
Production scheduling and reporting, JIT ..... June
Programming, AI Workstation ..... Mar.
Programming environment, unified, AI ..... Mar.
Programs, protocol analyzer ..... July
Program-to-program communication Oct.
Proper interval ..... Jan.
Protection model, HP Precision ..... Aug.
Protocol analyzer ..... July
Protocols, network ..... Oct.
Prototyping, software ..... June
Pseudoinstructions ..... Jan.
PSNs ..... Oct.
Pulse width modulator chip ..... Apr.
PXP (packet exchange protocol) ..... Oct.

Q

Quadrature modulation ..... Apr.
Quadrature sampler ..... June
Quality metrics, software ..... Mar.
Queries ..... Nov.
Query processing ..... Dec.

R

RAM disc, portable computer ..... July
Random repetitive sampling ..... Apr.
Random values testing ..... Mar.
Rate-based production scheduling ... June
Real-time extensions, HP-UX ..... Dec.
Real-time extensions, HP-UX 5.0 ..... July
Reciprocal counting, firmware ..... May
Recovery, DBMS ..... Dec.
Recovery time ..... Apr.
Reduced instruction set computers ... Jan. Aug.
Register assignment ..... Jan.
Registers, HP Precision ..... Aug.
Relational model, data base ..... Dec.
Relations ..... Dec.
Relationships ..... Dec.
Remote data base access ..... Oct.
Remote debugger ..... Aug.
Remote file access ..... Oct.
Remote process management ..... Oct.
Remote servers ..... Oct.
Response tuning, thick-film hybrid Apr.
RISC ..... Jan. Aug.
Rollback recovery ..... Dec.
Rollforward recovery ..... Dec.
ROM, data acquisition ..... Mar.
ROM disc, portable computer ..... July
ROM IMAGE Development Package, portable computer ..... July
ROMs, plug-in ..... July
RTE migration to HP-UX ..... Dec.
Rule-based programming ..... Mar.
Rule-based systems ..... Nov.
Runt packet filter ..... July

S

Salicide, IC metallization ..... May
Sampler, GaAs ..... Feb.
Sampler, oscilloscope ..... Apr.
Sampling, random repetitive ..... Apr.
Scaffold test package tool/standard ..... Mar.
Scalar network analyzer ..... Feb.
Scanner, imaging, Doppler measurements ..... June
Scatter read ..... Dec.
Schema file ..... Dec.
Schooner ..... Nov.
Schottky barrier diodes ..... Nov.
Screen update rate ..... Apr.
Security, data base ..... Dec.
Security, electronic mail ..... Sept.
Semaphores, HP-UX ..... Dec.
Sequence numbers ..... Jan.
Serial communications channel, PCIB ..... May
Serializability ..... Dec.
Series 300 Computers, design ..... Sept.
Series 300 Computers, HP-UX 5.0 .... July
Servo design, plotting system ..... Jan.
Session layer, OSI ..... Oct.
Shared memory, HP-UX ..... Dec.
Shell, HP-UX ..... Dec.
Short-pointer addressing ..... Aug.
Signals, HP-UX ..... Dec.
Silicon compilation ..... June
Simulation, digital microwave radio Apr.
Simulations, AI ..... Mar.
Simulator, HP Precision ..... Aug.
Single-cycle execution ..... Aug.
Skeletons, data structure ..... Mar.
Socket registry ..... Oct.
Soft front panel ..... May
Software compatibility, Series 200 and Series 300 ..... Sept.
Software development ..... Mar.
Software development, JIT ..... June
Software engineering, AI ..... Mar.
Software graphical design tool ..... Mar.
Software, oscilloscope ..... May
Software quality metrics ..... Mar.
Software testing tool, Triggers ..... Mar.
Space registers ..... Aug.
Special function units ..... Aug.

Special values testing ..... Mar.
Specifications testing ..... Mar.
Spectral moment, calculations ..... June
Spectrum program ..... Jan.
Aug.
Dec.
SRQ response times, HP 9000 Model 310, 320, and 550 Computers ..... July
Stability, DMM ..... Feb.
Stack ..... Dec.
Standard, test package ..... Mar.
Static link ..... Jan.
Stitch bonding ..... Apr.
Stock flow methods ..... June
Stored queries ..... Dec.
Structured Query Language (SQL) .... Dec.
Subsystem formatting ..... Nov.
Supervisor module ..... Aug.
Support, computer ..... Nov.
Support, electronic mail ..... Sept.
SVID ..... Dec.
Sync comparator hybrid ..... Apr.
System monitor, example ..... Mar.
System V, UNIX ..... Dec.

T

T1 carrier ..... Oct.
Table formatting ..... Nov.
Taken branch list ..... Aug.
TCP (transmission control protocol) Oct.
Temperature monitor, darkroom ..... Mar.
Testing, interactive graphics ..... May
Testing, software, automation ..... Mar.
Tests ..... Nov.
The Portable ..... July
Thick-film hybrids, oscilloscope ..... Apr.
Thread ..... Dec.
Time-based scheduling, HP-UX ..... Dec.
Time qualification, trigger ..... Apr.
Titanium silicide ..... May
Tools, software testing ..... Mar.
Toolset, object oriented programming in C ..... Mar.
Topological problem ..... Mar.
Training, electronic mail ..... Sept.
Transaction management, data base Dec.
Transactions, I/O ..... Aug.
Transitive closure ..... Jan.
Translation lookaside buffer ..... Aug.
Transport layer, OSI ..... Oct.
Trap machine ..... July
Trend detection ..... Nov.
Trend log ..... Nov.
Trigger hybrids ..... Apr.
Trigger system, oscilloscope ..... Apr.
Triggers, software testing ..... Mar.
Tuples ..... Dec.
Troubleshooting systems, expert ..... Nov.
Two-server model ..... Oct.

U

Ultrasound, Doppler imaging ..... June
University AI grants program ..... Mar.
UNIX defect tracking system ..... Mar.
UNIX operating system ..... Dec.
Usability testing ..... Jan.
/usr/group ..... Dec.

V

Valvular stenosis, Doppler analysis ..... June
Variable persistence ..... Apr.
Vector diagram ..... Apr.
Vector profiler, algorithm ..... Jan.
Vernier gain stage ..... Jan.
Video DAC IC ..... Sept.
Virtual cache ..... Dec.
Virtual memory addressing ..... Aug.
Virtual memory management, HP-UX Dec.
Virtual terminal protocol ..... Oct.
Virtual terminal tool ..... Mar.
VLSI design language ..... June
VLSI metallization ..... May
VLSI, Series 300 graphics ..... Sept.
Voltage-controlled device measurements ..... Feb.

W

Waveform recorder, low-frequency ... Jan.
Wedge bonding ..... Apr.
Workload data ..... Aug.
Workstations, modular ..... Sept.
Write-ahead log ..... Dec.

X

X.25 extensions, HP-UX 5.0 ..... July
X.25 networking ..... Oct.
X/OPEN ..... Dec.
X-Y recorder ..... Jan.

PART 3: Product Index

ALLBASE Data Base Management System ..... Dec.
ALLBASE/HP-UX ..... Dec.
ALLBASE/XL ..... Dec.
DMI/3000 ..... Oct.
HP DeskManager ..... Sept.
HP-IB HP 150 Command Library ..... May
HP-IB MS-DOS Command Library ..... May
HP JIT (Just-in-time manufacturing software) ..... June
HP-UX ..... July
Dec.
HP Vectra/IBM PC/AT PC Instruments Software ..... May
I•Q Tutor ..... Apr.
LAN/3000 ..... Oct.
Network Services/1000 ..... Oct.
Network Services/3000 ..... Oct.
Network Services/9000 ..... Oct.
PC Instruments ..... May
Portable Plus Computer ..... July
The Portable Computer ..... July
HP-71B Computer ..... Mar.
HP 150 PC Instruments Software ..... May
HP 3000 Computer ..... Nov.
HP 3000 Series 930 Computer ..... Jan.
Aug.
Dec.
3421A Data Acquisition/Control Unit ..... Mar.
3457A Digital Multimeter ..... Feb.
4971S LAN Protocol Analyzer ..... July
5314A Universal Counter ..... May
5350A Microwave Counter ..... Feb.
5351A Microwave Counter ..... Feb.
5352A Microwave Counter ..... Feb.
7090A Measurement Plotting System ..... Jan.
8757A Scalar Network Analyzer ..... Feb.
HP 9000 Series 200 Computer ..... Sept.
HP 9000 Series 300 Computers ..... July
Sept.
HP 9000 Model 310 Computer ..... July
Sept.
HP 9000 Model 320 Computer ..... July
Sept.
HP 9000 Model 550 Computer ..... July
HP 9000 Model 840 Computer ..... Aug.
Dec.
11664D Detector ..... Nov.
11970V/W Harmonic Mixers ..... Nov.
14857A HP-IB HP 150 Command Library ..... May
17090A Measurement Graphics Software ..... Jan.
46020A HP-HIL Keyboard ..... Sept.
46084A ID Module ..... Sept.
54100A/D Digitizing Oscilloscope ..... Apr.
54110D Digitizing Oscilloscope ..... Apr.
61010A Digital I/O Module ..... May
61011A Relay Multiplexer Module ..... May
61012A Dual Voltage DAC Module ..... May
61013A Digital Multimeter Module ..... May
61014A Function Generator Module ..... May
61015A Universal Counter Module ..... May
61016A Digitizing Oscilloscope Module ..... May
61017A Relay Actuator Module ..... May
61060AA HP 150 PC Instruments Software ..... May
61061BA HP Vectra/IBM PC/AT PC Instruments Software May

61062AA/BA HP-IB MS-DOS Command Library ..... May
77020A Phased Array Medical Ultrasound Imaging System June
77200B Scanner ..... June
77410A Doppler Imaging Subsystem ..... June
82479A Data Acquisition Pac ..... Mar.
98203A/B Keyboards ..... Sept.
98204B Video Board ..... Sept.
98546A Display Compatibility Interface ..... Sept.

PART 4: Author Index

Adams, Robert A. ..... Sept.
Adler, Glenn J. ..... July
Ahart, Diane M. ..... Nov.
Amano, Jun ..... May
Anderson, Andrew G. ..... July
Anderson, Keith F. ..... Feb.
Anklam, William J. ..... Nov.

Barbour, Michael J. ..... July
Beaudoin, Mimi ..... May
Beckman, Tom ..... Feb.
Bender, Douglas C. ..... Feb.
Berger, Arnold S. ..... Apr.
Bergmann, Bruce P. ..... Sept.
Berlin, Andrew A. ..... June
Bhargava, Raj K. ..... June
Blair, Steven R. ..... Mar.
Bockman, Francis E. ..... Jan.
Bostick, Diana G. ..... May
Bowen, Michael K. ..... Sept.
Brokish, James A. ..... Sept.
Brown, Alan S. ..... Dec.
Bryg, William R. ..... Aug.
Bugarin, John R. ..... Mar.
Burger, Stephen G. ..... Aug.
Burroughs, Gregory D. ..... Mar.
Button, Brian T. ..... Nov.

Cagan, Martin R. ..... Mar.
Carlson, Robert J. ..... Oct.
Chan, Buck H. ..... May
Chen, James ..... June
Clegg, Frederick W. ..... Dec.
Collins, Douglas M. ..... Nov.
Cordy, Clifford B., Jr. ..... July
Coutant, Deborah S. ..... Jan.

Dahlberg, Rebecca A. ..... Sept.
D'Angelo, Vincent J. ..... Mar.
Daniels, Thomas H. ..... Jan.
Davidson, Andrew W. ..... July
De Sostoa, Charles J. ..... Oct.
Dea, Robert W. ..... Mar.
Dean, Ronald P. ..... Sept.
DeLeon, Tim ..... Oct.
Desai, Deepak V. ..... Oct.
Dierschow, Carl ..... Oct.
Donnelly, James A. ..... Mar.
Duncombe, Alesia ..... July
Duyck, Ella M. ..... July

Eaton, John T. ..... July
Egbert, Jacob H. ..... Feb.
Ehlers, Eric R. ..... Nov.
Elliott, Scott S. ..... Nov.
Ellis, David J. ..... Mar.
Escovitz, William H. ..... Apr.
Evel, Eddie A. ..... Apr.

Faick, John C. ..... Feb.
Faulkner, Kevin J. ..... Oct.
Fearey, Seth G. ..... Mar.
Felps, Jimmie D. ..... Apr.
Fenoglio, John ..... Jan.
Figueredo, Domingo A. ..... Nov.
Frolik, William R. ..... July
Frydendall, David L. ..... July
Fuget, Craig D. ..... Mar.
Fugitt, J. Christopher ..... Oct.

Gardner, Robert D. ..... July
Garg, Atul ..... Oct.
Garrison, Bo ..... Feb.
Genther, Scott A. ..... Apr.
Gibson, Scott R. ..... Feb.
Goodman, Stephen D. ..... Jan.
Gottschalk, George R. ..... Nov.
Graham, Tonia G. ..... Oct.
Gray, Douglas A. ..... Nov.

Halberg, Leslie I. ..... June
Hammond, Carol L. ..... Jan.
Harrington, Daniel B. ..... Jan.
Harrison, Keith A. ..... Nov.
Hempstead, David C. ..... June
Hirata, Thomas M. ..... Dec.
Ho, Gary Shiu-Fan ..... Dec.
Hodge, David J. ..... Sept.
How, Michael ..... June
Huck, Jerome C. ..... Aug.
Hughes, William L. ..... May
Hunt, Barry F. ..... June
Hurtado-Sanchez, Luis ..... Sept.

Jain, Suneel ..... Jan.
James, David V. ..... Aug.
Jensen, Gordon A. ..... July
Johnsen, Sigurd W. ..... Nov.
Jundanian, Rich ..... June

Karp, Sydney M. ..... June
Keith, John C. ..... Sept.
Kelley, Jon W. ..... Jan.
Knoblock, Daryl E. ..... Sept.
Knouse, Charles W. ..... Oct.
Koehler, Ann M. ..... Dec.
Kononenko, George ..... May
Kuchinsky, Allan J. ..... June
Kusmer, Steven R. ..... Dec.

Laczynski, Edward ..... May
Lantz, Carl B. ..... July
Leavitt, Steven C. ..... June
Lee, Ruby Bei-Loh ..... Aug.
Lenk, Robert M. ..... July
Lennert, David C. ..... Dec.
Levine, Allan ..... May
Lewis, Jeffrey A. ..... June
Lombardi, Teri L. ..... June
Loomis, Courtney ..... July
Loughry, Donald C. ..... Oct.
Luehman, Kent W. ..... May
Lukes, Joseph A. ..... Aug.
Lynn, Brian K. ..... Oct.

Magenheimer, Daniel J. ..... Aug.
Maghakian, Emil ..... Jan.
Magnin, Paul A. ..... June
Mahon, Michael J. ..... Aug.
Marcus, Robert I. ..... Nov.
Mariani, Blenda ..... Nov.
Martin, Daniel J. ..... May
Martin, Sally ..... Feb.
Mati, Nicholas P. ..... Sept.
Matreci, Robert J. ..... Nov.
May, Robert B. ..... July
Mettetal, Pierry ..... Oct.
Miller, Robert V. ..... May
Miller, Terrence C. ..... Aug.
Morris, Jerry D. ..... July
Mueller, Amy Tada ..... Sept.
Mueller, Joseph E. ..... Feb.
Muterspaugh, Helen ..... May
Myles, Craig M. ..... Nov.

Navarro, Nancy L. ..... Oct.
Ng, Jenny ..... Dec.
Nishimoto, Alvina Y. ..... June
Noyes, Harold B. ..... July

O'Connell, Raymond G., Jr. ..... June
Odineal, Robert D. ..... Aug.
Oldfield, Danny J. ..... Apr.
Olsen, Douglas W. ..... Feb.

Palombo, Rosemarie ..... Sept.
Pan, Benjamin Y. M. ..... June
Passell, Robert A. ..... June
Patterson, George A. ..... Nov.
Pearson, James W. ..... July
Pechulis, Michael J. ..... Dec.
Pedersen, Christen K. ..... Apr.
Peregrino, Luiz ..... Feb.
Pettit, Ricky L. ..... May
Porter, Arthur W. ..... Apr.

Reames, Stephen P. ..... July
Reusser, Gertrude G. ..... Oct.

Richards, Bruce J. ..... Nov.
Richter, Kenneth A. ..... Feb.
Rodine, Thomas G. ..... Feb.
Rothschild, Charles J., 3rd ..... May
Rowe, Mark S. ..... July
Rowell, Joseph, Jr. ..... Feb.
Rubinstein, Jonathan J. ..... Sept.
Rush, Kenneth ..... Apr.

Sanchez, Jorge ..... Jan.
Sandberg, Gilbert I. ..... Sept.
Sandberg, Kenneth P. ..... Sept.
Schlater, Rodney T. ..... Apr.
Schlesinger, David ..... May
Schneider, Robert J. ..... July
Scope, Arie ..... Oct.
Scott, Barbara J. ..... Mar.
Shafer, Timothy C. ..... Oct.
Sikes, Mark A. ..... Dec.
Singleton, David E. ..... Dec.
Sismilich, Robert C. ..... May
Slater, Lynn R., Jr. ..... Nov.
Sloan, Susan R. ..... Nov.
Smith, Jeffrey H. ..... July
Sontag, John R. ..... Dec.
Speer, Martin L. ..... Sept.
Stahlin, Bonnie Dykes ..... July
Stever, Scott D. ..... Feb.
Swenson, Kristy Ward ..... Sept.

Thiele, Karl E. ..... June
Thompson, Bruce A. ..... Mar.
Thompson, Dean R. ..... Oct.
Toeppen, Derek E. ..... Apr.
Tolley, Ronald G. ..... July
..... Dec.
Tomberlin, Jeffrey ..... July
Tribby, David M. ..... Oct.
Tuttle, Ronald K. ..... Feb.
Tykulsky, Al ..... June

Umphrey, James M. ..... July
Upham, Herb ..... Nov.

Vandoorn, Roy M. ..... Nov.
Van Voorhis, Steven T. ..... Jan.
Veazey, Judson E. ..... Dec.
Vishwanath, Krishnan ..... Dec.

Walker, William T. ..... May
Ward, William T. ..... Mar.
Warner, Richard E. ..... Sept.
Wasmuth, David B. ..... Nov.
Wassenberg, Craig ..... Oct.
Weiman, Lyle A. ..... Oct.
Weller, Dennis J. ..... May
Wolpert, David L. ..... May
Woodhull, Frederic W., II ..... Feb.

Yip, Paul K. ..... June
Young, R. Michael ..... Nov.

Zurakowski, Mark P. ..... Nov.

Authors

December 1986

4 HP-UX Operating System

John R. Sontag

A native of Pittsburgh, Pennsylvania, John Sontag attended Carnegie-Mellon University, receiving his BSEE degree in 1979. After coming to HP's Data Systems Division the same year, he contributed to the development of RTE drivers for HP 1000 Computers and to the HP Micro/1000 Computer. He later worked on HP-UX and is now the HP-UX I/O project manager. John and his wife, who is also an HP engineer, live in Santa Clara, California and have one son. He's active in his church, leading a youth group and a marriage preparation program. During his leisure time he enjoys sailing, skiing, volleyball, and taking his son to the park.

Gary Shiu-Fan Ho

Born in Hong Kong, Gary Ho studied computer science and electrical engineering at the University of California at Berkeley. He earned his BS degree in 1975, his MS degree in 1977, and his PhD degree in 1979 and worked at Bell Laboratories before coming to HP in 1982. He has held several engineering and management positions and is currently section manager for HP-UX for HP Precision Architecture. He's named as inventor on two patents related to multiprocessor virtual memory management and distributed update verification. Gary and his wife live in San Jose, California and have one child.

Steven R. Kusmer

With HP since 1979, Steve Kusmer is the project manager for HP-UX operating systems for HP 9000 Series 800 Computers. Before working on HP-UX, he contributed to the development of hardware and operating systems for HP 1000 Computers. An alumnus of Cornell University, he holds a 1979 BSEE degree. He's a member of the ACM and coauthor of two papers, one a 1984 HP Journal article on RTE system software for HP 1000 Computers. Steve is married and lives in San Francisco. He holds a black belt in Korean karate and likes running and backpacking.

Frederick W. Clegg

With HP since 1975, Fred Clegg is one of the R&D section managers responsible for software development for HP 9000 Series 800 Computers. He has held management and engineering positions for several computer development efforts, including the HP 300 and HP 9000 Series 500 Computers. Born in Atlanta, Georgia, he holds a BS degree in engineering science from Oakland University (1965) and MS and PhD degrees in electrical engineering from Stanford University (1967 and 1970). He was an assistant professor in electrical engineering at Santa Clara University before coming to HP. He's the author or coauthor of 11 papers and other material related to fault-tolerant computing, user interfaces, and the implementation of the UNIX operating system on HP computers. Fred and his wife and daughter live in Cupertino, California. An avid pilot, he moonlights as a certified flight instructor. He's also a skier and amateur radio operator (W6IYO).

Thomas M. Hirata

With HP since 1972, Tom Hirata has contributed to the development of drivers and diagnostics for HP 1000 Computers, and to Image/1000 and DataCap 1000. Born in Poston, Arizona, he graduated from San Jose State University in 1971 with a BA degree in mathematics. He and his wife live in San Jose, California and he enjoys carpentry and tennis.

Mark A. Sikes

With HP since 1985, Mark Sikes is an R&D software engineer at the Information Technology Group who has contributed to the development of HP TurboWindow software for HP Precision Architecture computers. Born in Dallas, Texas, he graduated from Texas A&M University with a BSE degree in computer science in 1982. His other professional experience involved developing operating system software for microcomputers. Mark lives in San Jose, California and is interested in home computing and music. Other activities include hiking, camping, and tennis.

Krishnan Vishwanath

Krishnan (Vish) Vishwanath has been with HP since 1983 and is the technical leader for HPSQL products for HP Precision Architecture. Born in Madras, India, he earned a bachelor's degree in electrical engineering from the Indian Institute of Technology in 1975 and a master's degree in business administration from the Indian Institute of Management in 1977. He continued his education at Case Western Reserve University, completing work for an MS degree in computer science in 1983. A resident of Santa Clara, California, Vish enjoys racquetball, photography, and reading.

David E. Singleton

A Utah native, David Singleton was born in Murray and attended Brigham Young University. His BS degree in computer science was awarded in 1984 and he came to HP the same year. An R&D software engineer, he has contributed to the development of DBCore for HP ALLBASE for HP Precision Architecture computers. Before coming to HP he worked on a circuit simulator for Signetics Corporation. He's interested in distributed data bases, expert systems, and computer graphics. David lives in Cupertino, California and enjoys skiing, sailing, and dancing.

Michael J. Pechulis

A native of Cleveland, Ohio, Mike Pechulis studied mathematics and systems analysis at Miami University, earning a BS degree in each subject in 1979 and 1980. With HP since 1980, he first worked on software integration for the HP 3000 Computer and is now an R&D software engineer. In his current position he works on HP TurboImage software for HP Precision Architecture computers. Mike lives in Cupertino, California and likes hiking and cross-country skiing in California's Sierra Nevada mountains.

Jenny Ng

Jenny Ng has been with HP since 1979 and is currently a software development engineer for HPImage. She has also worked on operating system software for the HP 300 Computer, on HP Word, and on printer support software for the HP 2680A and HP 2688A Laser Printers. Born in Hong Kong, she graduated in 1979 from the University of California at Berkeley with a BS degree in computer science. She's a resident of Sunnyvale, California and lists tennis, aerobics, traveling, and music as leisure-time activities.

Judson E. Veazey

Jay Veazey came to HP in 1985, the same year he graduated with an MSCS degree from the University of Texas at Austin. His BA degree in English was awarded in 1979 and he worked as a computer programmer for the U.S. Geological Survey before coming to HP. An R&D software engineer, he has contributed to the development of HPIMAGE for HP Precision Architecture computers. His responsibilities have included the testing software, utility routines, and the schema processor. His professional interests include data base management systems and expert systems. Born in Dallas, Texas, he's now living in Santa Clara, California. He and his wife have one son and he likes hiking and reading.

Ann M. Koehler

Ann Koehler has been with HP since 1979 and is responsible for the design and implementation of the HPIMAGE schema processor. She was also a product assurance engineer for HP 3000 Computer graphics products, including HP Draw and HP EasyChart, and has developed other operating system and applications software. Born in Minneapolis, Minnesota, she holds a BA degree in computer science from the University of Minnesota (1978). Ann lives in Santa Clara, California, and enjoys bridge, sewing, and embroidery. She says she's also gradually replacing a lifetime collection of records with compact discs.

Alan S. Brown

Alan (A.J.) Brown is a graduate of Oregon State University (BSCS 1982) and has been with HP since 1981. His first project was working on a forecasting methodology for Corvallis Division marketing. More recently he has contributed to the development of externals for HPSQL/V, HP's relational data base for MPE V, and he's currently a project manager for HP Query products. A resident of Sunnyvale, California, A.J. was born in Kansas City, Missouri. His outside interests include polo and English horse riding and showing. He also likes playing tennis and piano.

Reader Forum

The HP Journal encourages technical discussion of the topics presented in recent articles and will publish letters expected to be of interest to our readers.
Letters must be brief and are subject to editing.
Letters should be addressed to:

Editor, Hewlett-Packard Journal, 3200 Hillview Avenue, Palo Alto, CA 94304, U.S.A.

Editor:

In the July 1986 issue of the *HP Journal* you refer to local customs ("New HP-UX Features for HP 9000 Series 300 Workstations," p. 38). You make no mention of international standards relevant to these points. I suggest that your *Journal*, with its world-wide circulation, is an ideal place to do this.

Concerning large decimal numbers, the digits are grouped in threes on either side of the radix, without any separating character. The preferred symbol for the radix is a comma. So your example, written in accord with the International Organization for Standardization (ISO), is 1 432 679,09. For obvious reasons this convention is not always followed when sums of money are involved, in which case the numbers are regularly spaced.

Concerning currency, there is at least one currency that is divided into two smaller units involving two decimal points. The Maltese pound comprises 100 cents of ten mills each. So three Maltese pounds may appear as £M 3.00.0.

Concerning dates, ISO Standard 2014 *Writing of Calendar Dates in All-Numeric Form* requires that dates in the Gregorian calendar be written in the sequence of year, month, day in one of the following ways. Using your example:

19861012	1986-10-12	1986 10 12	861012
	86-10-12	86 10 12	

This standard was approved in 1976 by the U.S.A., Japan, the U.K., and 13 other European countries, among others. Unless we all adopt this standard within about ten years we will have great confusion with dates such as 01-02-03 with three possible interpretations!

For simple numbering of days, independent of calendar conventions, Comité Consultatif International des Radiocommunications Standard 457-1 recommends the use of the Modified Julian Date (MJD), a five-digit decimal day count originating on 17 November 1858. A more convenient reference is the author's fortieth birthday, 27 October 1984 with MJD 46000.

Concerning time scales, in practice all world time scales (and time signals) are derived from Coordinated Universal Time (UTC), which can differ from Greenwich Mean Time (GMT) by up to 0.9 second either way. UTC has seconds of constant length, but not always 60 seconds in each minute. GMT always has 60 seconds in each minute, but the length of the second varies slowly. I suspect that your computer software does not provide for leap seconds so it is effectively, as you say, based on GMT although this is not generally accessible to the public.

Contrary to the information given in your *Journal* the Cook Islands are currently 10 hours behind UTC/GMT and this changes seasonally on 86-10-26 to 9 hours, 30 minutes. Singapore is now always 8 hours ahead of UTC/GMT. The only country with an offset not a multiple of 30 minutes is Nepal (5 hours, 45 minutes ahead).

John P. Chambers
Tadworth, Surrey, United Kingdom

Thank you for the information that you provided.

The intent of the Native Language Support (NLS) section of the July 1986 HP-UX article was to provide an overview of native language support issues and indicate some of the solutions already in place. The UNIX community is just beginning to understand how many limiting assumptions are encountered in UNIX. Character set, local customs, and user messages were selected as three key limitations. There are others. Many people find staggering the total number of changes required to remove these limitations. The section was kept quite short to keep it from becoming overwhelming. However, to keep the information from being too removed from reality, several examples were added, some based on rather obscure facts.

The selected examples reflect the existing Hewlett-Packard solution. Almost all of the information included in HP-UX Native Language Support products is selected to be compatible with the HP 3000 Computer NLS products. The information was derived from international standards as well as inputs from our sales force and customers. With specific reference to the ISO 2014 standard concerning numeric dates, while it was adopted in 1976, the de facto standard in the United States remains month, day, year. It is not clear to me how to effect a change. Witness the failure to convert to the metric system in the U.S.A. However, HP-UX Native Language Support provides the flexibility to accommodate such a change. Using the dumpmsg *and* gencat *commands along with an editor of choice, a user can easily change or correct the format. No recoding or recompilation is required.*

Your point is well taken concerning world time scales. Original AT&T documentation refers to Greenwich Mean Time (GMT). HP-UX has retained this reference. Also retained is the admittedly narrow interval of time that UNIX addresses. Dates before 1970 and after 1999 are not accommodated uniformly. There is discussion currently within the UNIX community concerning dates before and after these limits but to report any conclusions would be premature.

HP-UX maintains historical information about time zone adjustments for the time intervals from 1970 to 1999. The current adjustments (10 hours behind GMT for the Cook Islands, 8 hours ahead for Singapore) and those used as examples (10 hours 38 minutes for the Cook Islands, 7 hours 30 minutes for Singapore) fall within the UNIX time interval and are equally valuable. The dates and times of each change in time zone adjustment are also required.

Leap seconds have been an amusing issue to me. The cost of implementation and performance has made such a change unjustifiable for the moment. While UNIX claims to use GMT, all known hardware uses seconds of uniform interval. From what you say, UNIX could more correctly be said to run Coordinated Universal Time (UTC) and be (I believe) thirteen seconds off.

Ronald G. Tolley
Member of the Technical Staff
Systems Software Operation

Data Base Management for HP Precision Architecture Computers

HP ALLBASE supports both network and relational data access and runs under both the MPE XL and the HP-UX operating systems. Migration of existing data bases to the new architecture has been carefully planned for.

by Alan S. Brown, Thomas M. Hirata, Ann M. Koehler, Krishnan Vishwanath, Jenny Ng, Michael J. Pechulis, Mark A. Sikes, David E. Singleton, and Judson E. Veazey

ONE OF THE MOST VALUABLE ASSETS of a business is the data required to manage its daily operation. This data can include such information as a list of customers, the components required to build a product, sales records, and organization charts—in other words, any information that helps the business operate efficiently and smoothly. A data base management system (DBMS) is a collection of programs and procedures intended to help a business control its data. Logically related data is stored together in a set of files called a data base. The DBMS software provides tools for defining these data bases and regulating access to them.

The benefits of using a DBMS include:

- Centralized control. All data essential to an organization resides in one place, and the responsibility for managing it can be well-defined.
- Data consolidation. Because the same data can be easily shared by many application programs, duplicate copies of the data are not needed. Thus changes to the data can be made once and are available simultaneously to all programs with access to the data base. Since there is only one copy of the data, there is no risk that two programs will use different versions of the data and produce inconsistent results.
- Program independence from physical storage. The DBMS, by providing a standard interface for retrieving and updating data, hides the physical storage representation and the file system dependencies from the application programmer. It allows the programmer to concentrate on developing the functionality of the application rather than designing data file formats and access procedures.
- Flexibility. As new needs arise, data and access routes can be added to the data base to support new functions without affecting currently working programs.
- Data security. When data is stored in standard files, a user can typically be given access to either all of the data or none of it. A DBMS can allow access rights to be regulated down to the data item level. That is, a user may be allowed to read one part of the data base and at the same time be denied access to another part. So it is possible, for example, to allow a personnel clerk to retrieve the name of an employee's manager but not the employee's salary.
- Ad hoc data retrieval. Using a query program supplied with the DBMS, one-time requests for information can be satisfied when the need for the data arises. Special programs do not have to be written to retrieve the information.

Three Data Models

Most data base management systems today model data in one of three ways: hierarchical, network, or relational (see Fig. 1). In the hierarchical model, data has a one-to-many relationship; the data is logically organized as a tree. A "parent" file can have several children, but a "child" file can have only one parent. The network model relaxes the restriction that a child can have only one parent. It supports a many-to-many relationship of data; the data organization is that of a graph. So the network model encompasses the hierarchical model.

In both hierarchical and network models, all data relationships are predefined by the data base designer. These relationships are typically embedded in the data. It is the user's responsibility to identify which path should be followed to retrieve information from the data base. For this reason, the hierarchical and network models are often called navigational data base management systems. The application program navigates through the predefined relationships to manipulate the data.

The relational model, the third form of data base management system, presents the user with a simpler view of the data. Data appears as a collection of tables, which do not contain any predefined relationships. The data relationships are defined by the queries made against the data base. When a query is made, the DBMS, rather than the application program, determines the route used to access the data.

Navigational data base management systems typically offer better performance whenever processing is repetitive and there is a high volume of transactions. A characteristic of such applications is that they use stable data base structures. Performance considerations are more important than arbitrary access to the data. A navigational DBMS allows the programmer to specify exactly how the data is to be accessed. While this requires more planning, it allows the sophisticated user to design for optimum performance.

Relational data base management systems, on the other hand, tend to be easier to use and more flexible. Since the

data base does not contain any fixed data relationships, the scope of a query is not restricted by the structure of the data base. Changes to the data base organization are also easier to make. Furthermore, because the DBMS rather than the programmer selects the access path, complex queries are no more difficult to code than simple ones.

Historically, only one of these data base models has been available from any particular DBMS. With the introduction of HP Precision Architecture, HP offers its customers the choice of either a network or a relational interface from one DBMS. Customers are free to select the best data base model for each application. This new DBMS is called ALLBASE and is supported on both the MPE XL and the HP-UX operating systems.

ALLBASE Overview

ALLBASE has three major software components: HPIMAGE, HPSQL, and DBCore.

HPIMAGE is the network model interface that carries forward the Image tradition that began with HP's highly successful Image/3000 and Image/1000 products. HPIMAGE provides a migration path for current Image users and introduces several new features.

HPSQL is the relational interface based on Structured Query Language, an ANSI industry standard relational interface developed by IBM. This interface is the same as the one supported by the HP SQL/V product recently introduced on the MPE V operating system.

DBCore is an internal set of data base definition and access services shared by the HPIMAGE and HPSQL components of ALLBASE.

In addition to ALLBASE, the data base management software for HP Precision Architecture computers includes a migration package that helps customers move their existing data bases and applications to ALLBASE, and query products that allow users to access their HPIMAGE and HPSQL data bases without writing programs. Fig. 2 gives an overview of the product structure.

DBCore

DBCore is the heart of ALLBASE. It is the internal component used by both HPIMAGE and HPSQL that defines and manipulates data. DBCore performs the basic DBMS functions of data definition, data access, transaction management, concurrency control, logging and recovery, and accounting. In addition, it hides operating system and file system dependencies from the higher levels of software.

From the user's perspective, there are two major categories of DBMS functions: data definition and data manipulation.

Data Definition

Data in ALLBASE is stored in the form of relations (see "Data Storage in ALLBASE," page 46. A relation is a two-dimensional table composed of rows and columns. The rows of a relation are called tuples. A tuple consists of an ordered set of data values. The values that are in the same position in each tuple form a column. All the values in a particular column are the same type of data. The data types supported by DBCore are integer, character, binary, real, packed decimal, and zoned decimal.

The tuples in a relation are often accessed according to the values in a certain set of columns. To avoid exhaustive searches for these values, DBCore allows data structures called indexes to be built on relations. The columns whose values are of interest form the *key* of the index. DBCore supports three types of indexes: b-tree indexes, parent-child relationships, and hash indexes. It also supports linked lists or threads that allow the higher levels of soft-

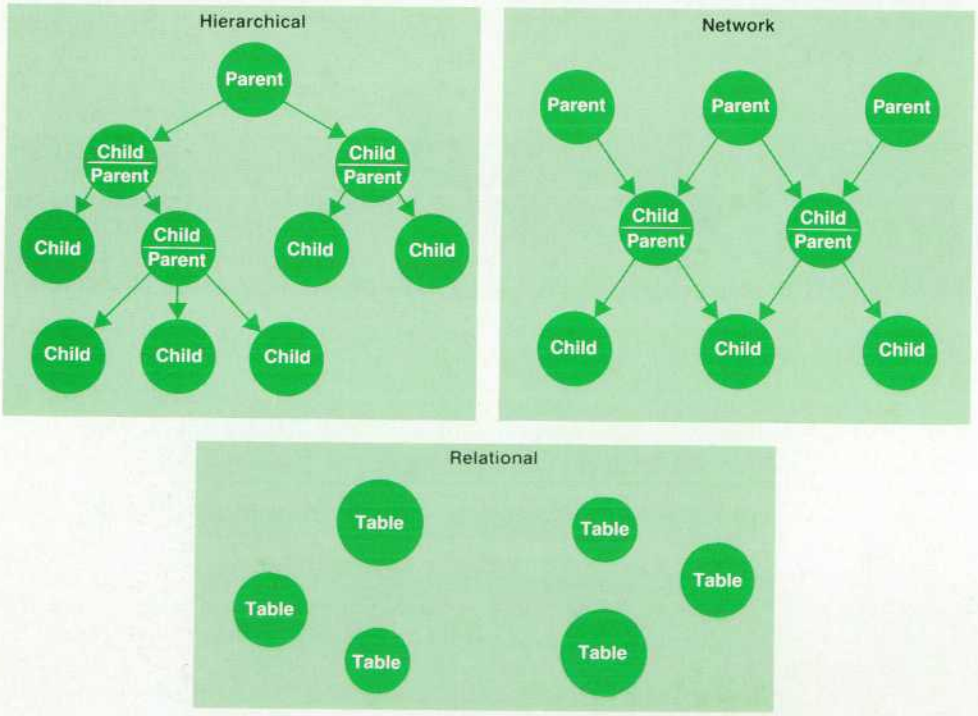


Fig. 1. *The three data base models used by data base management systems.*

ware to define their own access order. It is this variety of access structures that enables DBCore to support both network and relational data base models.

A b-tree index defines a search tree on a relation that divides the tuples in the relation into groups based on the values of the key columns. The b-tree substantially reduces the number of tuples DBCore must examine to find all tuples containing a specific value. When a b-tree is created, it can be given the property of uniqueness; this means that DBCore will not allow more than one tuple with a given key value to be inserted into the relation. If the index is not unique, then any number of tuples with the same key value can be inserted into the relation. B-trees are the most flexible indexes supported by DBCore; they can be created or deleted at any time.

A parent-child relationship (PCR) defines a relationship between the data of two relations. One relation is designated as the parent and one as the child. If a PCR is defined, DBCore will not allow a tuple to be inserted into the child relation unless the parent relation contains a tuple with the same key value. Likewise, a tuple cannot be deleted from a parent relation if there is a tuple in the child relation with the same key value. In addition to supporting this integrity constraint, a PCR provides a set of access methods. First, it can be used just like a b-tree to access the data in either the parent or the child. Second, given a parent tuple, the PCR enables DBCore to retrieve all its child tuples. And conversely, given a child tuple, DBCore can retrieve the related parent tuple.

A hash index is defined on a relation by "hashing" one or more key columns into an address. Hashing is a mathematical function that converts a key into an address. For example, say the first column of a relation contains the names of customers, and this column is the key to the hash index. Each name is put through the hash function and an exact address within the relation is returned. Whenever data concerning a customer is requested the data can be fetched by hashing the customer's name into the address of the tuple containing the data for that customer. Hash indexes, while providing fast random access to data, are basically static structures; a hash index can only be created on an empty relation, and the only way it can be deleted is to drop the relation.

A thread is a linked list of pointers contained in one column of the relation. Each pointer points to another tuple in the relation. The maintenance of this list is the responsibility of the DBCore user, that is, HPIMAGE or HPSQL. DBCore simply provides a fast access method for following the pointers.

Collections of relations, and the indexes on them, form data bases. The relations in a data base have a user-defined, logical relationship to one another. Data bases, in turn, are grouped into data base environments (DBEnvironments). A DBEnvironment is represented by a physical structure. The objects that make up an ALLBASE DBEnvironment are:

- A configuration file (DBECon) containing the DBEnvironment startup parameters.
- A log file that contains a record of all changes made to the DBEnvironment. The DBEnvironment is the ALLBASE unit of backup and recovery.
- The files used to hold the data for the relations and indexes in the data bases belonging to the DBEnvironment. These files are called DBEFiles.

DBEFiles are grouped together into DBEFilesets. A DBEFileset is a mechanism to create a dynamic file system on top of a static file system. When a relation becomes full, new files can be associated with the DBEFileset that contains the relation. Additional data inserted into the relation will then be stored in these new files. DBEFilesets can have up to 32,767 files associated with them.

Data Manipulation

The DBMS commands for adding data to a data base and modifying it are known as the data manipulation language

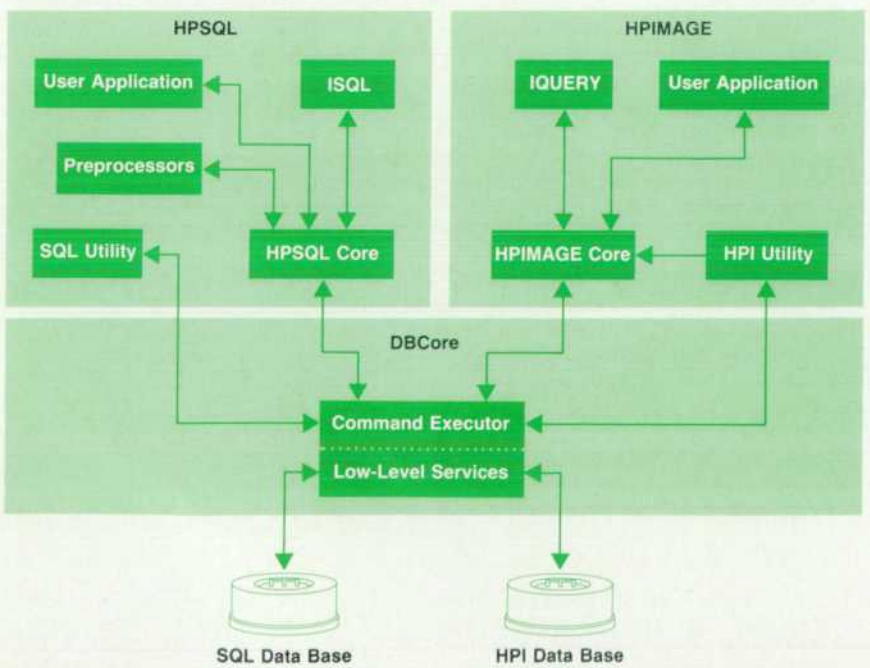


Fig. 2. *HP ALLBASE product structure. HPSQL is the relational user interface. HPIMAGE is the network user interface. DBCore is used by both interfaces.*

(DML). The following scenario describes how the DBCore DML is invoked by the ALLBASE interfaces in a typical data base application.

As users run the HPIMAGE or HPSQL application, it connects to the data base environment and starts a DBCore user session. Within this session, the application proceeds to do its work on behalf of the user. This work can include such things as retrieving data, inserting new tuples, deleting old ones, and updating existing data. All work is done in logical units called transactions. The work is recorded permanently in the data base as each transaction is committed (made permanent). A program commits work for the user by executing an explicit commit work command. At this time, ALLBASE makes all changes done in this transaction visible to other users. When a user exits the application, the program terminates the DBCore session. Other users can run this same program or other programs that access the same data base simultaneously. When the last session is terminated, the data base can be brought down to back up the day's work.

DBCore supports data access methods that take advantage of the indexes defined on relations. DBCore opens scans on relations. The types of scans DBCore supports are: relations scans, b-tree index scans, hash index scans, parent-child scans, and thread scans. Data can be fetched one tuple at a time or in bulk quantities. In a bulk operation the user specifies the number of tuples to retrieve in a single fetch.

In addition to retrieving data, DBCore also supports inserting tuples, deleting tuples, updating data in tuples, and sorting data. Again, these operations can be done one tuple at a time or in bulk. All data manipulation functions are invoked from HPIMAGE or HPSQL by issuing a request to DBCore. The language that defines these manipulations consists, basically, of four data structures: a request block, a predicate tree, a projection list, and a tuple buffer. The request block is initialized for each DBCore request, specifying the type and parameters of the request. The parameters of a request include the number of tuples to manipulate, from one to as many as requested, say five hundred tuples for bulk manipulation. A predicate tree specifies which tuples of a relation are requested. For example, the user may only want to retrieve the tuples of customers who have a credit rating less than 5. A projection list specifies which columns of a tuple are requested. A tuple buffer is the structure to which DBCore writes the output or from which it reads the input of the request.

Data is inserted into relations as tuples. Data is inserted by filling the tuple buffer with the tuples to be inserted and initializing the request block with the information needed to perform the insert. This information includes the identifier of the relation into which the data is to be inserted, the number of of tuples being inserted, and the number, lengths, and data types of the columns in the tuples.

An update works similarly to an insert. If the new data is the same size as the existing data, it is written over the old data. If it is larger, a new tuple containing the updated columns is inserted into the relation and the old tuple is deleted.

Data is deleted by specifying which tuples of a relation to delete. Whenever data is changed, inserted, updated, or deleted, DBCore automatically updates any indexes that are defined on the data in the relation.

Transaction Management

A transaction is a logical unit of work bounded by a begin work statement and a commit work statement. DBCore guarantees that either all of the work in a transaction is performed or that none of it is. The begin and commit work statements allow DBCore to maintain data integrity, transparency, concurrency control, and recoverability. The commit work statement tells DBCore that the user is satisfied with all the work and to commit the work (make it permanent). Changes to the data base are only made permanent when a transaction is committed. Similarly, changes can be seen by other users only after work is committed.

Transactions that are not committed at the time of a system crash are undone at the next startup of the data base.

Concurrency Control

Since the data in a data base is shared, the DBMS must prevent multiple users from altering the same data simultaneously. DBCore must ensure that no transactions are lost, that transactions are not partially committed, and that the changes made to data by a transaction are not seen prematurely by other users. DBCore also guarantees that all concurrent transactions are independent and serializable. To be serializable, the results of concurrent transactions must be equivalent to the results of the transactions if they were run one at a time.

DBCore controls concurrent data base access with transactions and locking. Transactions isolate concurrent users from each other. Each request within a transaction is stamped with a unique transaction ID number and locks the objects it touches. The requests are also marked with a unique time stamp that makes the transactions serializable. The requests are recorded in a log file.

As DBCore touches objects, whether reading or writing, the objects are automatically locked to prevent concurrent users from changing the same data simultaneously. The objects in DBCore that can be locked are: relations, pages within a relation, and tuples within a page. In practice, only the special system relations that store the structure of the data base use tuple locks because of the high volume of access. For performance reasons, only page and relation locks are used on user data. The size of the object being locked is referred to as its granularity. Relations, since they are the largest, have the highest granularity, tuples the lowest.

Objects are locked in different modes. There are five lock modes in DBCore. These lock modes constitute a lock compatibility matrix defining which lock combinations are compatible for a certain object. The exclusive lock mode prevents any access by other users. The share lock mode allows multiple users to read but not alter the same data. In addition to these two classic types of locks (exclusive and share), DBCore supports the notion of intent (sub) locks. It uses these to speed up its compatibility checking across the different granularities of locks. The share subexclusive lock mode allows one user to alter parts of a relation while allowing other users to read the unaltered portions of the relation. The intention share lock mode indicates

that there are share locks at a lower granularity, that is, if there is an intention share lock on a relation, there is a share lock on one or more of the pages of the relation. The intention exclusive lock indicates that there are exclusive locks at lower granularities.

When a lock is requested, DBCore checks to see what locks already exist for that object. If the existing locks on the object are compatible with the requested lock, the lock is granted; otherwise the transaction is suspended until the lock can be granted. DBCore checks to see if suspending the transaction will cause a deadlock; if so, one of the deadlocked transactions is picked by DBCore, usually the younger one that caused the deadlock, and that transaction is aborted, thus resolving the deadlock. A deadlock is defined as two or more user transactions, each waiting for locks held by the other waiting user transactions.

DBCore locks internal control blocks to prevent users from accessing the same control block at the same time. These internal locks are known as latches. Latches are similar to locks, but they are used only on internal data structures or control blocks. Another difference is that latches are never held across calls to DBCore. Since they are held for only a brief time, concurrent access to the data base is not reduced. Latches are acquired and released in a prespecified order to prevent deadlocking on internal objects. When a transaction attempts to latch a control block, the operation is either successful or not. If the operation is successful, the transaction latches the control block and owns the control block. If it is not successful, the transaction is suspended and placed in a queue of transactions waiting for that control block. When the control block is freed, the waiting processes are awakened and they again compete to latch the block. In practice, latches are usually available when transactions need them, so transactions are rarely suspended.

Logging and Recovery

DBCore logs all changes to the data in the DBEnvironment in its log file. If the system crashes, DBCore uses the information in the log file to restore the DBEnvironment to a consistent state. To DBCore, there are two types of system crashes: soft crashes and hard crashes. A soft crash is a system failure in which the machine goes down but the data on stable storage, usually disc, is still intact. A hard crash is a loss of data; a head crash on a disc is an example of a hard crash. After a soft crash the data in the data base is still in the state it was in when the system went down. After a hard crash, all or part of the data has been destroyed; it has been physically lost.

Consistent data is a state in which all transactions of the data base have finished completely. There are no partial changes to data or partially executed transactions. If all transactions are in this state, the data in the data base is consistent. To ensure that data is consistent, DBCore provides the capability of rolling back, or undoing, uncommitted transactions in the case of a soft crash. In the case of a hard crash, DBCore has the ability to roll forward, or redo, all transactions that were committed before the hard crash.

To handle both rollforward and rollback recovery, DBCore has two logging modes: archive mode and nonarchive mode. When DBCore is run in archive mode, all changes to the DBEnvironment are logged and the log space is never reused. When DBCore runs with archive mode turned off, all changes to the data base are logged, but space is periodically recovered and reused as transactions are committed. To perform rollforward recovery, DBCore must be run in archive mode. Rollback recovery is always available.

DBCore uses a write-ahead log to ensure that transactions are not lost and that the transactions are recoverable. Transaction requests are written to the log in the form of log records. Log records contain before and after images of the data and information about the type of the operation performed by DBCore. These log records are written to the log ahead of the data's being written to disc. This ensures that the transactions can be recovered from the log. If there is a system failure while data is being written to disc, the log will already have the log records, from which the transaction can be redone or undone as necessary.

Two transaction status tables are maintained at the beginning of the log file: one for rollback recovery called the checkpoint transaction status table and one for rollforward recovery called the archive transaction status table. During rollback recovery, DBCore reads the checkpoint transaction status table to see which transactions were not committed at system failure time. DBCore reads the log records for the uncommitted transactions, undoing the operations as it goes.

To perform rollforward recovery, an old copy of the DBEnvironment is restored from a backup or archive copy. The current log contains an image of the changes that have been made since the last backup. These changes can then be reapplied to the old data, rolling it forward to a consistent state. The user tells DBCore the time to which to roll forward. DBCore reads the archive transaction status table and starts reading log records at the position indicated by the archive status table. Log records are read one at a time and redone as they are encountered until DBCore has read forward to the specified recovery time. Transactions that do not have an end transaction record at this point will be rolled back to their begin transaction record.

DBCore supplies the fundamental services required by a data base management system, but it lacks the cohesive, user-oriented view of the data needed to be a complete DBMS. The interface components of ALLBASE, HPIMAGE and HPSQL, provide these remaining services.

HPIMAGE

The HPIMAGE component of ALLBASE supplies the user with a network model interface to the data managed by DBCore. Modeled after Image, the successful HP proprietary data base management system found on HP 1000s and MPE V-based HP 3000s, HPIMAGE is a combination of old and new. It supports an interface similar (although not identical) to the one already used by many programs and familiar to customers. At the same time, it has been updated to reflect some of the advances in data base management technology that have been made since Image was first designed 15 years ago.

HPIMAGE consists of two components:

- A utility program called HPIUtil that provides commands to create and delete data bases, back up and recover data base environments, and perform other assorted maintenance tasks.
- A set of intrinsics, or system procedures, for use in application programs. The intrinsics provide a well-defined and structured procedural interface for fast access to the data in data bases.

For ease of discussion, the intrinsics are divided into three categories. The data manipulation intrinsics include all of the procedures that are used to retrieve and modify data. These include HPIGET, HPIPUT, HPIDELETE, HPIUPDATE, HPIFIND, and HPIFINDSET. The intrinsics in the second group, for example HPIBEGIN and HPIEND, allow the user to define transactions. The final category of intrinsics includes those procedures needed to gain access to a data base (HPIOPEN) and to terminate it (HPICLOSE).

User's View of Data

An HPIMAGE data base consists of data items, data entries, and data sets. A data item is the smallest accessible element in an HPIMAGE data base and corresponds to a column in a DBCore relation. A data entry is an ordered set of related items. It is one record of information and corresponds to a DBCore tuple. One or more data entries form a data set, which is the same as a DBCore relation. A data base is a named collection of related data sets. A collection of data bases can be grouped together into a data base environment.

Data Sets

Data sets in HPIMAGE can be defined as one of three types: master, detail, or relation. A master data set can only be defined as a parent, that is, the first level in the network model. A detail set can only be defined as a child on the lowest level. A relation data set can have the properties of a master set, a detail set, or both at the same time, and therefore can be at any level in an HPIMAGE data base. This allows HPIMAGE data bases to be structured in multiple levels (removing the restriction of two levels found in earlier versions of Image). Master sets can also be further defined as either automatic or manual. In a manual master, all data entries must be manipulated explicitly by the user, as with the other data set types. In an automatic master, however, data entries are automatically inserted and deleted for the user by HPIMAGE.

Parent sets, either master or relation data sets, serve as indexes to child sets (detail or relation sets). To represent data relationships, parent and child data sets are combined in a network of data sets that forms the entire data base. This network not only stores data but represents relationships among pieces of data as well. The data can then be retrieved based on their relationships.

Paths

The primary data relationship supported by HPIMAGE is the parent-child relationship known as a path. One item in a parent set can be specified as a key item. A key item in a parent set serves as a unique index into a related child set. Each key item value points to a chain that links all entries in the child set that have a matching item value. The corresponding item in the child set is called a search item. The key and search item relationship is the basis of a path in HPIMAGE. If a path is defined, the user can retrieve all entries in a child set with a given search item value.

Besides providing a data access method, a path also imposes an integrity constraint. An entry can be added to a child only if an entry with a matching key item value exists in each parent of the child. Furthermore, an entry cannot be deleted from a parent set until all child entries with the same search item value have been deleted.

Fig. 3 shows how data items, data entries, and data sets relate to one another, using a sales application as an example.

Data Base Definition

All data relationships are defined in HPIMAGE at the time the data base is created. The structure of an HPIMAGE data base is basically static, and the method of creating a data base reflects this assumption. The user creates a data base by defining its structure and characteristics in a *schema* file. A schema file is much like the data declaration section of a program. The data items, data entries, and data sets that make up the data base are described in a specialized HPIMAGE data definition language. Fig. 4 shows part of the schema file used to define the data base shown in Fig. 3. Once the schema has been written, the user invokes the schema processor from the HPIUtil utility program to build the data base according to the description contained in the schema.

Data Base Security

The schema also contains a description of the data base security. Data base security in HPIMAGE is implemented through the use of passwords and security classes. A security class is a number between 1 and 63, and one password can be assigned to each security class. When a data base is designed, the user specifies read and write class lists for each data item and set. These lists specify which security classes have read access, write access, or no access to all or part of the data base. Whenever a user accesses a

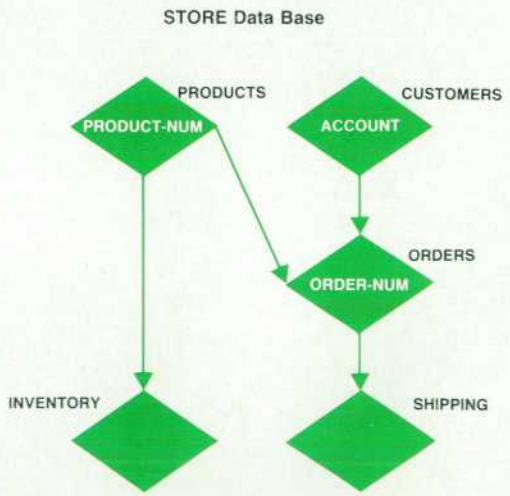


Fig. 3. *An example of an HPIMAGE data base called STORE.*

data base by opening it, a password is specified and subsequent access to data in the data base is either granted or denied depending on the corresponding security class and the read and write class lists.

Data Base Access

Before any data in an HPIMAGE data base can be accessed, HPIOPEN must be called to initiate access to the data base. Opening a data base is like opening a file. It links the data base to the application program and establishes the dynamic information needed for the program to read from and write to the data base. Every time the program is executed, it must call HPIOPEN.

A data base can be opened in one of four access modes. Each mode determines the types of operations that a user can perform on the data base as well as the types of operations other users can perform simultaneously. A data base can be opened for reading or reading and writing, and with shared or exclusive access. Sharing can be restricted to other readers only.

The allowed access option combinations are:

open mode 1: read/write, shared
open mode 3: read/write, exclusive

```
Begin Data Base STORE;

 Passwords:    10  Clerk;
               20  Credit;
               30  Shipping;
               40  Manager;

 Items:        Account,          I4      (10,20,30/40);
               Address,          X26     (10,20,30/40);
               City,             X12     (10,20,30/40);
               Credit-rating,    I2      (20/40);
                     .
                     .
                     .
               Zip,              X5      (10,20,30/40);

 Sets:
   Name:       Customers, Relation       (10,30/20,40);
   Entry:      Account(1),
               Last-Name,
               First-Name,
               Address,
               City,
               State,
               Zip,
               Credit-rating;
   Capacity:   200;
                     .
                     .
                     .
   Name:       Orders, Relation          (30/10,40);
   Entry:      Account (Customers),
               Order-num(1),
               Prod-num,
               Quantity,
   Capacity:   500;
                     .
                     .
                     .
End.
```

Fig. 4. *A portion of the HPIMAGE schema defining the STORE data base of Fig. 3.*

open mode 8: read, shared with other readers only
open mode 9: read, shared.

Using HPIMAGE intrinsics, programs can access data in the data base in several ways. Serial access retrieves successive data entries from the data set, one entry at a time. Direct access retrieves a data entry based on its record location within the data base. Calculated access retrieves a data entry in a parent set based on its key item value. Chained access retrieves successively all the data entries in a child set that share a common search item value. Subset access retrieves data entries from a subset of records within a data set that satisfy a given set of conditions. Data entries can be retrieved in either a forward or a backward direction.

For subset access, the HPIFINDSET intrinsic must be called first to establish a current subset for the data set to be accessed. HPIFINDSET allows a user to identify a subset of entries in a data set that meet a designated set of conditions. These conditions are specified in a predicate. A predicate can span several items within a data set and consists of a set of operators and operands. Generic search is also supported in HPIMAGE, that is, it is possible to retrieve entries based on the value of the first number of characters of an item of type character string. A generic search is indicated by following the character string used for the search by the character @.

Example 1: A user can find all records in the CUSTOMERS data set for which ACCOUNT exceeds 500 and for which item number 8 is not 10 by passing the following predicate to HPIFINDSET:

```
(ACCOUNT >500) and (&8<> 10);
```

Example 2: To find all records in CUSTOMERS for which LAST-NAME starts with the characters LU, the following predicate can be used:

```
LAST-NAME = "LU"@;
```

Implementation on DBCore

The challenge of the HPIMAGE project was to map the well-defined external definition of existing Image products onto the new set of internal services provided by DBCore. Using DBCore offers several advantages. First, DBCore is designed to handle a high level of concurrent access, and therefore, as HP Precision Architecture processors become faster and faster, DBCore and HPIMAGE will be able to support the increased numbers of users and transactions. Second, it was only necessary to implement the DBMS services required by both HPIMAGE and HPSQL once. Traditional Image access methods do not support the relational model of data, whereas DBCore has the functionality required to support both the relational and network models. Third, because both HPIMAGE and HPSQL share a common set of internals, it is possible for the same data to be accessed through either interface. This will be realized in a future release of ALLBASE, which will allow an HPIMAGE data base to be accessed from both the HPIMAGE and HPSQL interfaces.

The following discussion explains how the HPIMAGE data structures and intrinsics are implemented using DBCore.

Data Structures

Because DBCore handles the physical storage of and access to all data, HPIMAGE data structures must be mapped onto the DBCore structures. The key concepts behind the mapping are:

- A data set in an HPIMAGE data base corresponds to a DBCore relation. One or more DBEFiles are created to hold each data set.
- A data entry in the data set corresponds to a tuple in the DBCore relation, and a data item is a column within the relation.
- Paths are represented by DBCore parent-child relationship (PCR) indexes. One PCR is created for each child of a parent. A PCR supports both HPIMAGE chained reads and the parent-child integrity constraint.
- A hash index is defined on each master data set for use in cases where fast random access is required.

HPIMAGE also maintains a set of DBCore relations that contain a description of each data base. These relations are collectively known as the HPIMAGE catalog and are used by the HPIMAGE software to record the structure of a data base and generate the correct calls to DBCore needed to perform the operations requested by the user. The catalog is not accessed by the user.

Opening and Closing a Data Base

The intrinsic HPIOPEN is used to connect a user to a data base. When a user calls HPIOPEN to open a data base for the first time, a DBCore session is started for the user. (A program can open the same data base more than once. This work is done only on the first open.) All further operations on the same DBEnvironment will connect to this DBCore session. After the DBCore session is started, the information about the data base in the HPIMAGE catalog is read into shared memory. Subsequent intrinsic calls to access data in the data base obtain information about the data base from shared memory instead of having to go to the HPIMAGE catalog. The DBCore session is ended when the last open data base in the data base environment is closed. HPICLOSE disconnects a user from a data base.

Transaction Management

DBCore guarantees data base integrity and consistency by requiring data base activity to be performed within a defined DBCore transaction. This is implemented at the user level through HP Image transaction management, which encompasses transaction definition, logging, and locking. In HP Image, a transaction is bracketed by an (HPIBEGIN, HPIEND) pair. All HP Image data manipulation intrinsics must be called from within a transaction. HP Image allows single and multiple data base transactions. A multiple data base transaction is a transaction that spans two or more data bases, as long as the data bases specified in the HPIBEGIN call belong to the same data base environment.

There is a one-to-one mapping between HP Image transactions and DBCore transactions. A DBCore transaction is started when HPIBEGIN is called. The DBCore transaction will be ended when HPIEND is called to commit the transaction. For multiple data base transactions, since all the data bases specified must be in the same data base environment, only one DBCore transaction needs to be started.

Data Manipulation

The insert, delete, and update data operations of HP Image map directly onto their corresponding DBCore functions. The mapping of the various HP Image data retrieval methods is a little more complicated.

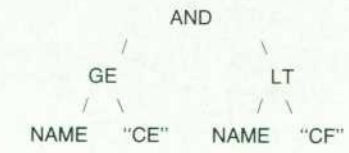
Serial access is implemented by opening a DBCore relation scan on the data set specified by the user. Scan information is stored in a local control block. Each subsequent HPIGET call will fetch one entry at a time in the forward or backward direction, until there are no more entries in the data set.

Direct access is implemented by calling the DBCore fetch function to retrieve the desired entry based on the HP Image record number. Calculated access is implemented by opening a DBCore index scan on the index defined for the parent data set. The entry is then fetched based on the key item value specified by the user.

For chained access, the intrinsic HPIFIND must be called first to establish a current record chain based on a specific search item value. Subsequent HPIGET calls will retrieve entries from that chain. HPIFIND establishes information for the current chain by opening a DBCore index scan on the data set using the search item value as the key.

For subset access, HPIFINDSET establishes a current subset by opening a DBCore index scan based on the predicate specified by the user. Before the HPIMAGE predicate can be passed to DBCore, it has to be translated into a form understandable to DBCore. Building the predicate is done in several steps. First, the HPIMAGE predicate is parsed by HPIFINDSET, which takes a character string and checks whether its syntax follows a given grammar. Its output is the predicate in tree format.

DBCore does not directly support generic searches, so these predicates have to be converted to a form that is understood by DBCore. If the user submits the predicate NAMES = "CE"@ to find all names that begin with the letters CE, HPIMAGE translates this predicate into "find all names greater than or equal to CE and less than CF." From this predicate, the following parse tree is generated:



The parse tree is then converted into DBCore linear format and used to open a DBCore scan. If an index exists on one of the items involved in the predicate, then an index scan is opened. Otherwise, a relation scan is used to search for all records that meet the specified conditions. Subsequent HPIGET calls to access records in this subset will call the DBCore fetch function to retrieve one entry at a time.

Generic search is a new feature not found in earlier versions of Image. It is an example of how the DBCore functionality is being exploited to add the more flexible access methods to HPIMAGE that are typically associated with relational data base management systems. However, for the

user who needs the total flexibility that a relational system can offer, ALLBASE provides HPSQL.

HPSQL

HPSQL, the third component of ALLBASE, gives the user a relational interface to the data managed by DBCore. Based on the ANSI industry standard Structured Query Language, HPSQL supports a nonprocedural command language for accessing data. The user only needs to specify the function to be performed, and HPSQL determines how to perform it. The interface is the same as that of HPSQL/V on MPE V systems, allowing users to develop relational applications on MPE V systems, and later move them to the HPSQL interface of ALLBASE.

Unlike HPIMAGE, all access to HPSQL is through SQL commands. These commands can either be embedded in application programs where language preprocessors supplied with HPSQL translate them into calls to the underlying HPSQL software, or they can be executed from ISQL, the interactive HPSQL subsystem.

The basic HPSQL commands fall into the following categories:

- Data definition
- Data manipulation
- Transaction management
- Authorization control.

HPSQL Data Definition

HPSQL presents the user with a very simple view of data. An HPSQL data base is a collection of tables. Like the DBCore relation on which it is based, a table is composed of rows and columns. To create a table, the user only needs to specify a unique name for it along with the names, types, and lengths of the columns in the table.

Users may also create indexes on a table to reduce the time it takes to retrieve data from the table. An index is specified as an ordered set of columns in a table; these columns form the index key. The key values for each row in the table are stored in a b-tree, so the rows can be located quickly. Index data is never made visible to the user. This allows the user to add and delete indexes without having to modify previously written programs. Unlike HPIMAGE, where the existence of indexes (or paths) determines how the user accesses the data base, indexes in HPSQL are invisible to the user. If an index exists, HPSQL will use it to retrieve data; otherwise it will scan the entire table looking for the data to return. In either case, the user phrases the query in the same way.

HPSQL allows tables and indexes to be added and deleted dynamically, so the user can easily change the data base to reflect changing needs.

Data Manipulation

Data access and update in HPSQL consist of the INSERT, UPDATE, DELETE, and SELECT commands. The INSERT and DELETE operations work at the row level while UPDATE and SELECT are column level operations (i.e., the user specifies which columns are UPDATEd or SELECTed). Furthermore, the SELECT operation supports computed arithmetic, sorting, and aggregate functions (e.g., MIN, MAX, and AVG), and GROUP BY operations.

The UPDATE, DELETE, and SELECT operations can be restrictively applied to certain rows that satisfy a given set of conditions. These conditions are specified in terms of a WHERE clause.

Example:

```
SELECT column1, column2 FROM table1 WHERE column3 > 100;
```

HPSQL supports the notion of null values as the absence of any value. Each column in a table can be specified as potentially NULL or NOT NULL. If a column is potentially NULL, then certain rows in that table may contain no value for that column. Columns in a table can also be updated to contain a null value (i.e., no value).

Transaction Management

All HPSQL data definition and manipulation activities happen within transactions. A user can begin a transaction using the BEGIN WORK directive. (Optionally, if a transaction is not active when an operation is attempted, one will be started.) No changes will actually be written to disc until the user does an explicit COMMIT WORK command. The entire transaction can be annulled by using the ROLLBACK WORK command.

Authorization Control

HPSQL provides a flexible and dynamic security mechanism. The creator of an object, for example, a table or a stored query, automatically becomes its owner. To be able to access some object created by another user, you need explicit authorization from the owner. The owner of a table may grant SELECT, INSERT, DELETE, UPDATE, ALTER, and INDEX authorities, in any combination, to any user. Stored query owners can give other users RUN authority to execute the query.

In addition, there are three special authorities controllable only by the data base administrator (DBA), the person who created the data base environment. To be able to access a data base environment, a user must have CONNECT authority. RESOURCE authority enables the user to create data base objects. Finally, the user needs DBA authority to perform administrative functions like storage space management, backup, and recovery. The DBA authority circumvents all other explicit authorization.

Users can be combined together into groups for authorization purposes. A user group relationship is flexible in that individual users or groups can be added to or removed from other groups. Any authority granted to a group is also implicitly granted to all its member users and groups.

All authority can be rescinded through the REVOKE command in HPSQL. Thus the various authorities in HPSQL are completely dynamic.

HPSQL Objects and System Catalog

HPSQL, like HPIMAGE and DBCore, stores information about its objects in a system catalog. This catalog is composed of a set of tables, one table per object type. Examples of object types are tables, columns, indexes, authorization groups, stored queries, etc. When an object is created, a

record describing it is added to the appropriate table; when it is dropped, the corresponding record is deleted from the table.

Query Processing

One of the major differences between HPIMAGE and HPSQL is that the HPIMAGE user is responsible for identifying the access path to any data that is to be retrieved or modified. With HPSQL, the user only specifies what data is to be accessed and HPSQL determines how to locate it. The query processor in HPSQL determines the best access path. The complete relational functionality of HPSQL is implemented in the query processor. The query processor accepts input query trees from the calling subsystem, and through a series of transformations, converts them into an executable form.

The query processor can be invoked in a variety of ways. An interactive SQL subsystem, ISQL, is provided for the *ad hoc* user. ISQL supports complete SQL functionality with the exception of functions that need programmatic support (for example, host variables). In addition, ISQL has its own set of commands that allow the user to monitor the interactive environment, load and unload data from and into flat files, etc.

The query processor can also be invoked programmatically through a variety of language preprocessors provided with HPSQL. Application programmers can embed SQL commands within their host language programs. Before compilation, an application program needs to go through the appropriate language preprocessor. HPSQL preprocessors generate stored queries for embedded SQL commands and replace those embedded statements by procedure calls to execute the corresponding stored queries. (Stored queries are explained in the next section.)

Before an SQL query can be sent to the query processor, it has to be parsed and linearized. All subsystems that call the query processor have to call the SQL parser first. The parser converts an SQL command into an SQL parse tree. This parse tree is then sent to the SQL linearizer. Linearization consists of generating a query tree without any address pointers. The concept of linearization addresses the issues of passing query trees between processes and writing query trees to disc. After linearization, the resultant linear tree is sent to the query processor where the query is either executed or stored for future use.

Stored Queries

Because it is common in many environments for the same query to be invoked repeatedly, HPSQL allows users to predefine queries and save the executable form in the data base environment. These stored queries are called sections. A section consists of the linearized input tree and the executable tree (also called the run tree) generated by the query processor.

More often than not, a section will depend on the existence of certain other objects and authorizations. If all the dependencies of a section are met then the section is marked as a valid stored query; otherwise it is marked as invalid. For example, the following SQL command creates a section named S1 that adds data to a table:

```
PREPARE S1 from 'INSERT INTO U1.T1 Values (10)';
```

For this section to be valid, the table U1.T1 must exist, and further, the user must have INSERT authority on U1.T1. If either validation criterion is false, the section S1 cannot be completely defined. However, HPSQL will still store the input tree and mark the section as invalid. If the validation criteria are met, the query processor will generate a run tree from the input tree and store it along with a list of its dependencies. The section will remain valid as long as the objects on which it is dependent do not change.

To execute this stored section, the user would issue the command:

```
EXECUTE S1;
```

At run time, the query processor checks the section S1 for validity. If it is valid, the query processor retrieves and executes the run tree. If the section is not valid, then the input tree will be loaded and the query processor will revalidate it. If all validation conditions are met at this time, the run tree will be generated and executed. In addition, the run tree will be restored and the section will be marked as valid.

Query Processor Internals

The query processor operates in two phases. It first prepares the query for execution. This phase is also known as query compilation, query preprocessing, and query definition. Second, it stores the preprocessed query for future execution or executes it immediately.

During the query preprocessing phase, the input linear tree is delinearized. Delinearization of a query tree consists of reestablishing pointers between nodes of a tree.

The delinearized tree then goes through a binding operation, which involves verifying the existence (or nonexistence) of data base objects. For example:

```
SELECT C1,C2 from T1;
```

requires that table T1 be already defined in the data base and that C1 and C2 be valid columns in table T1, whereas the query:

```
CREATE TABLE T2 (C1 integer, C2 char(20));
```

requires that table T2 not be already defined in the data base. Binding also consists of verifying that the user is indeed authorized to perform the function(s) implied by the SQL query. If either condition is not true at this stage, the query preprocessing goes no further, and an appropriate error message is issued.

Query optimization follows the binding phase. HPSQL attempts to optimize all data manipulation commands (INSERT, UPDATE, DELETE, and SELECT).

First, the bound tree is transformed into an optimal form. Projects and filters* are pushed as far down in the tree as possible. Boolean factors are resolved into a conjunctive normal form to the extent possible. View definitions are

*A filter is another name for the WHERE clause. A project is a list of column names to be retrieved.

compressed and aggregation operations are transformed. The query tree is then converted into a set of query blocks. A bound tree thus transformed is called a preoptimized tree.

In the second phase of optimization, the query cost is computed. The cost of a data manipulation query is given by the cost of scanning each table in the query. The optimizer generates a scan plan for each table in the query. This plan indicates the access method to be used for that table and the I/O cost. The I/O cost of a query is estimated in terms of the number of data and index pages likely to be read to perform the query. At present, HPSQL uses two types of DBCore scans. A relation scan will read all pages of the table once. An index scan will use a b-tree index selected by the optimizer to read tuples in the order of key value. The optimizer is guaranteed to generate a relation scan for a table that has no WHERE clause on it. If a WHERE clause is specified, then the optimizer will compute the index scan cost and compare it with the cost of a simple relation scan.

In addition to the scan plans, the optimizer also generates join plans for multiple table retrievals. A join plan determines the scan order for a given pair of tables. Thus, for an n-table SELECT, $n-1$ join plans are generated. The idea behind join optimization is to reduce the number of retrieved tuples as the query proceeds. The output of the cost computation phase is the optimized tree.

After the optimization phase, the query processor is ready to generate the executable tree for data manipulation commands (for non-data-manipulation commands, the bound tree is the executable tree). Along with tree nodes specifying the kind of operation to be performed, the run tree contains pseudocode (assembly-like code) to perform a variety of HPSQL operations. These are:

- Buffer transfer
- Null evaluation
- Arithmetic expression evaluation
- Aggregate computation
- Logical expression evaluation
- Pattern matching.

This code is embedded into the run tree as a binary constant string.

Creation of the run tree marks the end of the query preprocessing phase.

Query Storage/Execution

If the query is being defined (this could happen at preprocessing time, or it could happen in ISQL through the PREPARE command), then at the end of preprocessing, HPSQL causes the query to be stored in the system catalog. A stored query consists of the input tree and the executable tree, which may be a bound tree or a run tree. Temporary queries are held inside the local heaps of the query processor and are never flushed to disc.

At run time (this could happen with the execution of a preprocessed application, through the EXECUTE command in ISQL, or through a direct SQL command in ISQL), the preprocessed query is executed.

Interface with DBCore

HPSQL, like HPIMAGE, depends on DBCore to manage data definition and access, to ensure data integrity, and to control concurrency. The query processor invokes DBCore for each query executed.

ALLBASE on MPE XL and HP-UX

Both commercial and technical customers use applications that need general-purpose data base management systems, so ALLBASE is supported on both the MPE XL and the HP-UX operating systems. Because the user interfaces are uniform, the user need not understand (or be confused by) the intricacies of how the data base management system uses the operating system. Making the interface between a DBMS and the operating system transparent to the user requires care because a DBMS is dependent on the operating system, particularly for accessing files. The importance of the operating system makes it difficult to conceal unless the DBMS is divided into layers as is ALLBASE.

A uniform user interface was achieved rather simply, in spite of the inherent difficulties of making two operating systems transparent, because the ALLBASE products on MPE XL and HP-UX share the same source code. The code in ALLBASE that interfaces with one or the other of the operating systems is conditionally compiled, that is, it is compiled only for the operating system it accesses. But the high-level code, written in either Pascal or C, is nearly the same on both operating systems. Since project teams were developing ALLBASE on both MPE XL and HP-UX simultaneously, each with their own copy of the source code, coordination was required to maintain a complete version of ALLBASE. The HPIMAGE and HPSQL teams used a formal check-in, check-out system to maintain code integrity. Only one team at a time could make changes to any given module. This solution was viable because these components of ALLBASE rely on DBCore to perform most operating system dependent functions. The teams that developed DBCore used a resynchronization method. Each team could make changes to their own copies, but periodically the two teams would merge all changes into a new master copy. These mergers were simplified by the fact that the routines that access the operating system make up only about 10% of all procedures in DBCore.

From a software development point of view, maintaining a single ALLBASE source program had several advantages. First, development of the product was accelerated because the teams developing ALLBASE on MPE XL and those developing it for HP-UX were able to work in parallel. Second, with two teams working on the same code, the quality of the product was enhanced—a bug that might be overlooked by one team is less likely to escape two. And last, a single program is easier to maintain and improve, since changes only need to be made once and they are available in both products.

In software development, using existing code to perform new functions is called leverage. ALLBASE is a highly leveraged product. Not only is it leveraged across operating systems, but the HPSQL language preprocessors are also leveraged; they have very little code that is specific to the language being processed. Most of the code is the same no matter what language the user's program is written in.

In summary, ALLBASE hides both operating systems and presents a uniform interface to the user. The fact that the ALLBASE source code is the same for both operating systems resulted in an accelerated software development process and a product that is more reliable and easier to maintain.

Data Base Migration

One of the main objectives in the HP Precision Architecture program is to provide a high degree of compatibility as well as a smooth migration path between new and existing systems. This path should guarantee that the migration of applications and data bases is easily understood, while at the same time allowing flexibility, such as the capability to maximize performance in specified applications. The migration must be fast, and contingency options must exist.

The HP Precision Architecture data base migration plan addresses all these needs and requirements. It involves a series of smaller migration steps which vary in complexity and performance/functionality improvements. The migration process has been designed to satisfy the needs of the small data base application, consisting of one data base and one program, as well as the large data base application system, consisting of multiple data bases on many disc volumes with multiple applications working together.

The migration plan is separated into two complete migration paths, one for each market. The first, a commercial system migration path, supports migration between existing HP 3000 systems running MPE V with TurboImage and the HP 3000 Series 930 and 950 running MPE XL with both TurboImage and ALLBASE/XL. The second migration path addresses the technical market and provides a migration path between HP 1000 systems running RTE with Image/1000 and HP Precision Architecture systems running HP-UX with ALLBASE. These migration directions are discussed separately in the following paragraphs.

Commercial Data Base Migration

Because commercial data bases tend to be large and customers frequently have many of them, it is assumed that application and data base migration between traditional HP 3000 systems and the HP Precision Architecture systems will be done gradually. The MPE XL migration software takes this assumption into account by allowing users to move applications and data bases individually, in a series of simple steps. It is not necessary to follow all of the steps. The migration software gives customers the flexibity to select the migration path best suited to their particular needs.

Migration to TurboImage on MPE XL is the first step along the migration path. TurboImage on MPE XL is completely compatible with TurboImage on MPE V, so data bases and applications can simply be stored from an HP 3000 MPE V-based system and then restored onto an HP 3000 MPE XL-based system and run in compatibility mode. The advantages of this migration step include its simplicity and the speed of migration. In addition, new applications can be developed in compatibility mode on an MPE XL-based system and then moved back to an MPE V-based system without change.

The second migration step is to move applications to native mode. A performance gain can be realized for applications written in Pascal, Fortran, and COBOL by simply recompiling with the native mode compilers. No source code changes or data base conversions are required. Some languages do not have native mode compilers. Applications written in these languages can remain in compatibility mode or can be rewritten as time permits.

An alternative second step is to move the TurboImage data bases to HPIMAGE. By converting data bases to HPIMAGE using a provided conversion utility, many of the features of HPIMAGE are made available immediately, while the existing TurboImage application can be used in either compatibility mode or native mode. An "onion skin" layer of software on top of HPIMAGE, called TurboWindow, translates each TurboImage call to HPIMAGE and translates the results back to TurboImage format. TurboWindow reduces the migration effort by performing the syntax translations, error mapping, and transaction management on behalf of the application. However, for optimal use of HPIMAGE transaction management, some code modifications will be required.

HPIMAGE features such as transaction consistency, automatic rollback recovery, and dynamic restructuring can be used as soon as the data base is moved to HPIMAGE. HPIMAGE is a native mode subsystem, so data base access will be able to take full advantage of the speed of the HP Precision Architecture.

The last step in the migration, once all applications are in native mode and all data bases are HPIMAGE, is to replace the TurboImage interface with the HPIMAGE interface. At this point, the maximum performance benefits are attained, and the full HPIMAGE feature set is available.

Each migration step is a stable position, so customers can operate indefinitely with a combination of compatibility and native mode applications, and a combination of TurboImage and HPIMAGE data bases accessed from these applications.

MPE XL Data Base Migration Software

The data base migration software consists of four modules and utilities: DBMigrate, the native mode locator/switcher, the compatibility mode locator/switcher, and TurboWindow (see Fig. 5).

The utility DBMigrate is provided to migrate data from TurboImage to HPIMAGE. It unloads a TurboImage data base to tape or disc in HPIMAGE load format, and optionally starts the HPIMAGE load process simultaneously. It can be used to check for conversion problems and requirements, such as disc space required and corrupted data sets, without the I/O time needed to unload. Because the unload/load time is critical in conversion to HPIMAGE, and the quantity of data to unload is enormous for some TurboImage data bases, DBMigrate runs in native mode and uses the MPE XL mapped file feature to access TurboImage data sets. Bypassing the TurboImage intrinsic interface and the MPE file system interface maximizes the performance of DBMigrate, both by reducing the number of software levels involved, and by ensuring that the execution path remains wholly in native mode.

The native and compatibility locator/switcher modules

perform similar functions for each mode of execution. The native switcher mediates between a native mode application and TurboImage in compatibility mode. It handles parameter formatting for compatibility mode and maps compatibility specific information to more meaningful native mode information. Likewise, the compatibility switcher mediates between a compatibility mode application and TurboWindow in native mode. The native and compatibility locators intercept TurboImage intrinsic calls, detect the data base type, and then route data base requests to either the data base management system residing in that mode, or the associated switcher. Thus, both TurboImage and HPIMAGE through TurboWindow can be accessed transparently from the same application from either mode.

TurboWindow performs the translation of calls from TurboImage to HPIMAGE. It performs transaction management on behalf of the application, reformats and aligns parameters, and maps status information from HPIMAGE back into TurboImage form. With only a few exceptions, the full TurboImage interface is supported.

Technical Data Base Migration

ALLBASE migration tools are available for Image/1000 customers running the HP 1000 RTE operating system who want to port their applications to the HP-UX operating system. It is anticipated that HP 1000 customers who migrate to the HP Precision Architecture machines will do so to HP-UX rather than MPE XL. Since a compatibility mode does not exist on HP-UX as it does for MPE XL, it is always necessary to recompile the migrated programs on HP-UX before they are executed. The transfer from one system to another involves moving application programs and data bases. ALLBASE provides a manual and software tools to aid in the transfer. Fig. 6 shows an overview of the technical data base migration process.

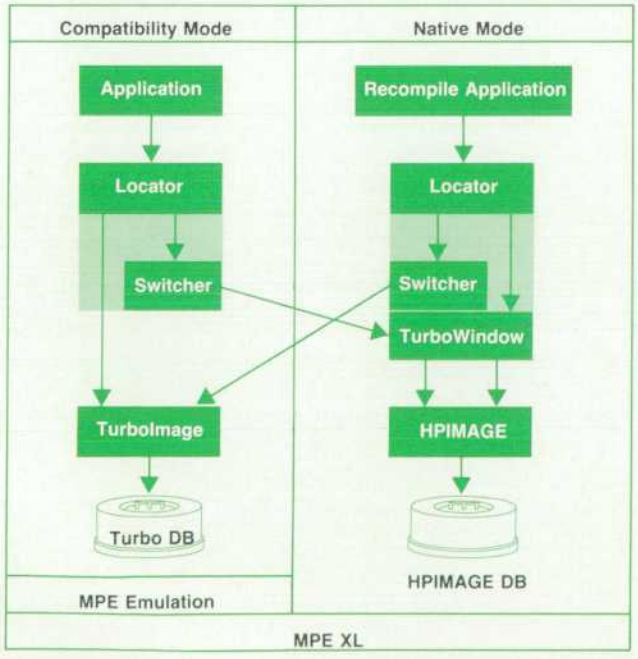


Fig. 5. *Commercial data base migration components and access paths.*

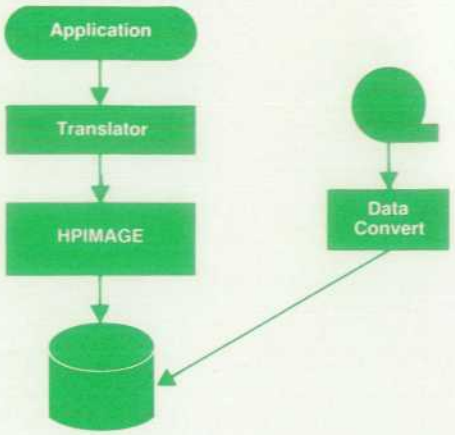


Fig. 6. *Technical data base migration.*

Since the HPIMAGE data base structure differs from Image/1000, Image/1000 data bases must be transformed into HPIMAGE data bases. To accomplish this, the user must edit the Image/1000 schema and convert it to an HPIMAGE schema. If the data base is Image/1000-II, a root file decompiler is available that will do most, if not all, of the conversion. Once the new HPIMAGE data base has been created on the HP-UX system, the data itself can be migrated using a special unload utility, DBMUN, and reloaded onto the new system.

Application programs are brought up on HP-UX in one of two ways: they can have all of their Image/1000 calls manually replaced by HPIMAGE calls and directly access the new HPIMAGE data base, or they can leave the Image/1000 calls unaltered and access the HPIMAGE data base through a run-time call translator that converts Image/1000 calls into HPIMAGE calls. However, the translator is not able to achieve 100% compatibility with Image/1000 functionality. To help with source code conversion, another migration tool, the migration analysis utility (MAU) is provided. MAU scans through a source program, locates and flags Image/1000 calls as well as other HP 1000 system dependencies, provides information about each call such as whether or not it is fully emulated and whether a performance degradation is expected, and summarizes with a statistical profile of the analysis. Where feasible, it is expected that a user will convert the software to HPIMAGE calls, but in the event that a more phased migration is desirable, the translator is available.

Query Products

No data base management offering would be complete without an ad hoc query interface. It is this level of software that gives users the ability to retrieve and report stored information without having to write a program. ALLBASE offers interfaces compatible with current offerings to provide a smooth migration to HP Precision Architecture machines. In addition, new tools that enhance the ALLBASE data base management system have been added.

To solve the issue of a smooth migration, the current version of Query/3000 has been ported to the MPE XL

operating system where it is called Query/V. There have been no changes made to the externals of the product, so users who are using Query/3000 now will have no difficulty using Query/V. Query/V can access both TurboImage and HPIMAGE data bases. This allows customers to move current TurboImage data bases from MPE V to MPE XL and begin using them immediately. If and when data bases are slowly migrated to HPIMAGE, Query will still be able to access them transparently, that is, the person using Query/V will not need to know that some of the data bases being accessed are TurboImage and some are HPIMAGE. The differences will be handled automatically. Once all of a customer's data bases have been migrated to HPIMAGE, Query/V can still be used to perform ad hoc retrieval and reporting without the necessity of using a new interface.

Query/V, then, is simply Query/3000 running on MPE XL. This is accomplished by running Query/3000 in compatibility mode on top of TurboWindow. TurboWindow handles the actual communication between an application and TurboImage and/or HPIMAGE data bases.

Since there have been no enhancements to Query/3000 when moved to MPE XL, it will not support all the features of HPIMAGE such as HPIFINDSET or relation sets. Because of this, a query product is needed that in the long term will support HPIMAGE and will be very robust. To fill this need, there is IQUERY.

IQUERY is very similar to Query/3000. The main difference is that it accesses only HPIMAGE data bases. It runs on the MPE XL and HP-UX operating systems and looks exactly the same on both. For MPE XL, IQUERY runs in native mode, so its performance is optimal. IQUERY will be the long-term solution for programmers and data base administrators to access HPIMAGE data bases. It will be enhanced as necessary to support the full feature set of HPIMAGE.

For the relational side of ALLBASE, there is a different query product called ISQL. ISQL runs on both MPE XL and HP-UX and provides access to HPSQL data bases. This ISQL is exactly the same as ISQL/V, the interface released with HPSQL on MPE V. Again, customers will find no difficulty or surprises when migrating to HP Precision Architecture.

One of the keys to the successful movement of the various query products to the different operating systems lies in the code itself. During development, code sharing among operating systems is planned for at the early stages. This means that whenever possible, machine dependent routines and those nice features of Pascal that only certain compilers support are not used. References in the source code to file names (for INCLUDEd source) are isolated to minimize changes resulting from file system differences. Any operating system calls that are made are also isolated in their own routines. As a result, nearly 90% of the code can be shared among the different operating systems. ISQL, for example, is maintained as one set of source code even though there are slight differences for MPE V, MPE XL, and HP-UX. Conditional compile directives are used to help in some areas.

Data Storage in ALLBASE

Data in ALLBASE is stored in the form of relations. Fig. 1 shows an example. The characteristics of this relation are:

- Its name is CUSTOMERS.
- It has eight columns. All values within any given column will have the same data type.
- When first created, the relation does not contain any data, that is, all the rows in the relation are empty.

Fig. 2 shows this relation with data inserted. Each row of the relation is called a tuple. A tuple is an ordered set of data values—one value for each column. The value of a column may be null (i.e., no value).

Because there are no indexes defined on the CUSTOMERS relation, the only way to retrieve a tuple with a specific value for one of the columns is to read the entire relation sequentially, looking for the correct tuple. If the relation contains thousands of tuples, this search could be too slow (acceptable speed depends on the application and the frequency of the query). To speed up the search, the user could define either a b-tree or hash index on the relation. The key of the index should be the column or columns that contain the values to be searched for. The b-tree example in Fig. 3 assumes that the user wants to retrieve tuples from the relation based on the values in the ACCOUNT column.

To find the tuple with ACCOUNT = 205 using the b-tree index,

CUSTOMERS

ACCOUNT	LAST-NAME	FIRST-NAME	ADDRESS	CITY	STATE	ZIP	CREDIT-RATING

Column

Fig. 1. *An example of a relation called* CUSTOMERS.

CUSTOMERS

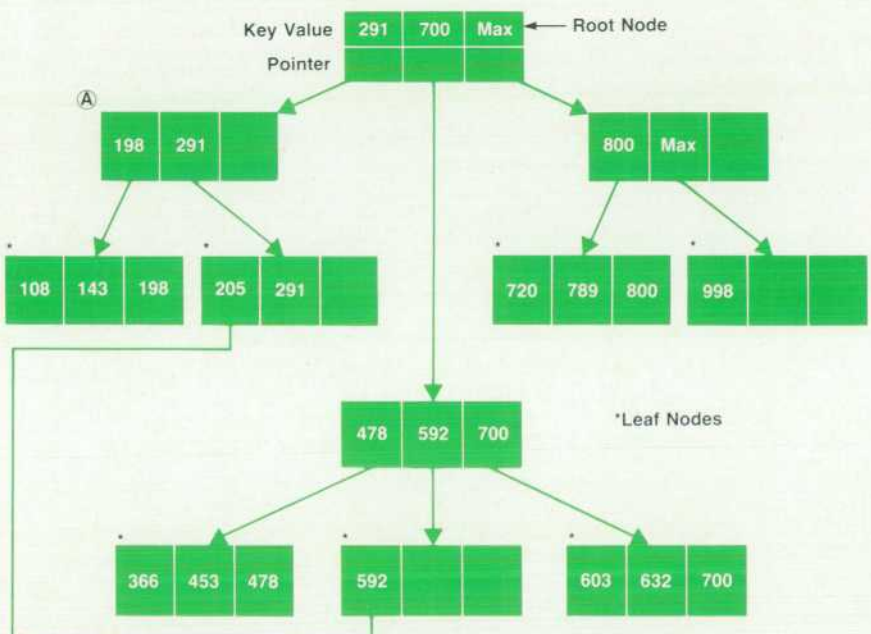
ACCOUNT	LAST-NAME	FIRST-NAME	ADDRESS	CITY	STATE	ZIP	CREDIT-RATING
592	Johnson	Elaine	123 North Street	Atlanta	GA	30348	2
789	Huxley	John	45 Main Street	Hollywood	CA	91601	4
205	Burns	Mary	92 First Ave.	Scranton	PA	18092	1
291	Webster	John	12 Starlight Lane	Toledo	OH	43604	7
800	Haugen	Donald	986 Nicollet Ave.	Minneapolis	MN	55414	10
603	Johnson	William	1078 Bay Street	Andover	MA	01810	5
998	.	.	.	.	.	.	.
148	.	.	.	.	.	.	.
720	.	.	.	.	.	.	.
432							
632							
198							
336							
478							
700							
108							

Tuple

Fig. 2. *The* CUSTOMERS *relation with data.*

the DBCore component of ALLBASE first retrieves the root node of the index. It scans the key values in the root node to determine which pointer to follow. If the specified value is ≤291 then the first pointer is used.* If the value is in the range 291< value ≤700 the second pointer is followed, and if it is >700 then the third pointer is used. Since 205<291, the first pointer is used to retrieve the nonleaf node labeled A. This node is searched in the same way as the root node and new nodes farther down the tree are retrieved until finally a leaf node is found. The leaf node is searched for the key value that matches the specified value

*B-trees are constructed in such a way that it is guaranteed that all key item values ≤291 (or whatever value is contained in the node) will be found along this branch of the tree (if the key value exists in the relation).



CUSTOMERS

ACCOUNT	LAST-NAME	FIRST-NAME	ADDRESS	CITY	STATE	ZIP	CREDIT-RATING
592	Johnson	Elaine	123 North Street	Atlanta	GA	30348	2
789	Huxley	John	45 Main Street	Hollywood	CA	91601	4
205	Burns	Mary	92 First Ave.	Scranton	PA	18092	1
291	Webster	John	12 Starlight Lane	Toledo	OH	43604	7
800	Haugen	Donald	986 Nicollet Ave.	Minneapolis	MN	55414	10
603	Johnson	William	1078 Bay Street	Andover	MA	01810	5
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

Fig. 3. *A b-tree on the* CUSTOMERS *relation. The key column is* ACCOUNT.

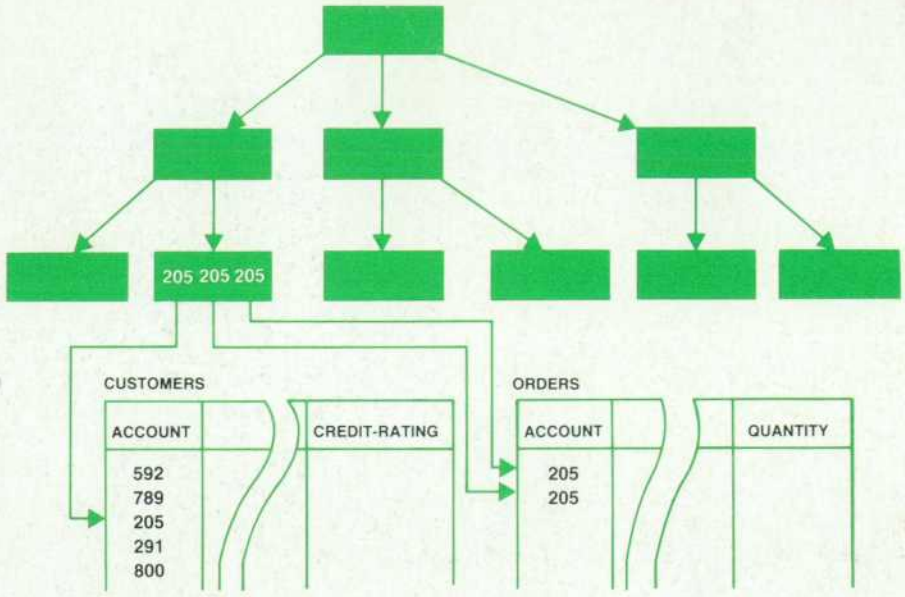


Fig. 4. *An example of a parent-child relationship (PCR) joining the relations* CUSTOMERS *and* ORDERS.

(205). The pointer associated with this key value points to the tuple in the relation that contains this same value. Each tuple in the relation is pointed to by one and only one entry in a leaf page of the index.

For simplicity, the example shows only three values in each node of the tree. In a more typical case, a node would be likely to have 100 or more entries. This keeps the number of levels in the tree small and thus minimizes the number of disc accesses required to retrieve the tuple.

A PCR (parent-child relationship) is very similar in structure to a b-tree. The primary difference between the two index types is that a b-tree contains key values for only one relation while a PCR contains values for two relations. The leaf nodes of a PCR point to tuples in both the parent relation and the child relation.

Fig. 4 shows an example of a PCR joining the relations CUSTOMERS and ORDERS. Features of this PCR are:

- An order cannot be placed for which a customer (ACCOUNT) does not exist.
- The PCR can be used as a b-tree on either the CUSTOMERS or the ORDERS relation to retrieve tuples with particular ACCOUNT values.

Address Correction Requested
Hewlett-Packard Company, 3200 Hillview Avenue, Palo Alto, California 94304

Bulk Rate
U.S. Postage
Paid
Hewlett-Packard
Company

HEWLETT-PACKARD JOURNAL

December 1986 Volume 37 • Number 12

Technical Information from the Laboratories of Hewlett-Packard Company
Hewlett-Packard Company, 3200 Hillview Avenue
Palo Alto, California 94304 U.S.A.
Hewlett-Packard Central Mailing Department
P.O. Box 529, Startbaan 16
1180 AM Amstelveen, The Netherlands
Yokogawa-Hewlett-Packard Ltd., Suginami-Ku Tokyo 168 Japan
Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive, Mississauga, Ontario L4V 1M8 Canada

CHANGE OF ADDRESS: To subscribe, change your address, or delete your name from our mailing list, send your request to Hewlett-Packard Journal, 3200 Hillview Avenue, Palo Alto, CA 94304 U.S.A. Include your old address label, if any. Allow 60 days.

5953-8555