Adding Real Time Capabilities to the UNIX* Operating System

Suzanne M. Daughty

Sol F. Kavy

Steven R. Kusmer

Douglas V. Larson

David C. Lennert

Frank-Peter Schmidt-Lademann

Hewlett-Packard Company

ABSTRACT:

Adapting the Unix operating systems to real-time markets is a lucrative challenge. By adding a little new functionality and a lot of performance tuning, Unix systems can support more demanding real-time applications such as those found on the factory floor, tapping into a multi-billion dollar market demanding a portable software environment such as System V. Most of the needed real-time functionality is already found in System V and 4.2BSD. Performance tuning is needed in the area of response time, especially process dispatch latency, which on typical Unix systems is measured in seconds rather than milliseconds. This paper presents what functionality is needed to adapt Unix systems to real-time markets, how they may acquire the needed performance, and how this combination satisfies real customer needs.

HPUX/RT is a Unix implementation with a BSD4.2 Kernel that implements the SVID and is enhanced with BSD4.2 functionality and HP proprietary services. HPUX/RT was especially tuned to provide deterministic realtime response and enhanced with functionality needed for realtime applications. We will discuss the methods used to achieve realtime performance in the HPUX/RT operating system.

---

* Unix is a registered trademark of AT&T in the United States and other countries.

1. Requirements for Real-Time Systems

The UNIX operating system is found in many marketplaces. It is the operating system for scientific supercomputers and PCs. However, one of the final frontiers for the acceptance of UNIX systems is in the real-time marketplace, and for a good reason: the real-time customer is the most demanding customer there is. The real-time customer's demands fall within three categories:

Performance

Real-time applications are primarily measured by their performance. Therefore, real-time customers will expect to squeeze the last ounce of performance out of a real-time system to meet their needs, and they will sometimes take measurements, their computer vendor never expected. The performance characteristics they measure are typically in terms of response time or throughput. An example of a response time measure is "How long after the receipt of an interrupt from my parallel I/O card can the system run my process which was waiting for that interrupt?" An example of a real-time throughput measure is "How long will it take for me to push my two gigabytes of data from my device to the filesystem?". The real challenge is that both questions will often be asked by the same customer!

Determinism

Customers expect that a real-time system will react in a deterministic manner. For example, it is not enough to have a good response most of the time -- you must provide good response all of the time. Real-time customers often build a computer into a system that has unforgiving constraints, which is usually because the system is controlling or monitoring other devices or machinary. As an example, a real-time computer built into a steel mill whose steel travels at 30 mph will be expected to respond quickly to alarm conditions. If the computer unexpectedly becomes busy for a whole second, the steel in the steel mill will have treveled 44 feet, and could possibly be strewn over the steel mill floor.

Flexibility

In the end, it is the real-time customers who truly

know best how e real-time computer can solve their applications needs. Customers must be provided with tools for writing their own drivers and for measuring system performance. They must be provided with source code, because they choose to understand in-depth how a system performs and they might want to tune it for their application. On the other hand, vendors of real-time computers must be humble, because real-time customers are glad to tell then how to build their systems!

The remainder of this paper presents e definition of a real-time system and then explains the real-time features implemented on the HP 9000 Series 800 computers. The HP 9000 is HP's computer family for engineering and manufacturing, and it runs HP-UX, a superset of AT&T SVID Issue 1. The Series 800 is HP's Precision Architectures computer line under the HP-UX operating system.

2. What is e Real-Time System?

A real-time system is a system that can respond in a deterministic and timely manner to events in the real world. Events in the real world could mean either large amounts of data that must be processed fast enough to prevent losing data (data throughput), or discrete events that must be recognized and responded to within certain time constraints (response time).

The table 1 presents and categorizes some real-time applications. It is important to note, that this list is by no means comprehensive; its purpose is to show the variety and pervasiveness of real-time applications.

3. Adding Real-Time Capabilities to UNIX

Given a definition of real-time and some sample real-time applications, the next question is "How can the UNIX operating system be augmented to meet the requirements of real-time applications?". While using System V as a base, HP-UX answers these questions in two parts: 1) by incorporating functionality from 4.2 BSD and adding new functionality from HP, and 2) by doing performance tuning on the kernel and filesystem. To better understand this approach, it is helpfull to be familiar wit HP's goals for adding real-time capability to the UNIX operating system:

Table 1: General Real-Time Applications and Some Examples

General Real-Time Applications	Examples
process monitoring and control	petroleum refinery paper mill chocolate factory
data aquisition	pipe-line sampling data inputs from a chemical reaction
communications	controlling satellites telephone switching systems
transaction-oriented processing	airlines reservation systems on-line banking stock quotations system
flight simulation and control	autopilot shuttle mission simulator
factory automation	material tracking parts production electronic assembly machine or instrument control
transportation	traffic light system air traffic control
detection systems	radar systems burglar alarm systems
interactive graphics	image processing video games

- Any real-time features implemented must not prevent SVID compatibility.
- Whereever possible, real-time features should be adopted from either System V or 4.2BSD. Only where needed real-time feature does not exist should HP add a new feature.
- Real-time features must be portable.
- Performance tuning must be transparent to user process.
- Real-time response must be comparable to real-time response on the HP 1000 A900 (HP's top-of-the-line real-time A-Series computer.

HP-UX on the Series 800 has met these goals. In addition, HP

is lobbying through standards-setting bodies to encourage their adoption of HP-UX's real-time features as part of an existing or evolving standard such as SVID or IEEE P1003.

3.1 Real-Time Features in HP-UX

This section introduces the real-time features of HP-UX on the Series 800 computers, explains their origin (either System V, 4,2BSD or HP) and also explains how each feature addresses certain concerns about real-time capabilities of UNIX systems.

The following features provide real-time capability to HP-UX:

Added Functionality

- Priority-based preemptive scheduling
- Process memory locking
- Privilege mechanism to control access to real-time priorities and memory locking
- Fine timer resolution and time-scheduling capabilities
- Interprocess communication and synchronisation
- Reliable signals
- Shared memory for high bandwidth communication
- Asynchronous I/O for increased throughput
- Synchronous I/O for increased reliability
- Preallocation of disk space
- Powerfail recovery for increased reliability

Performance Tuning

- Kernel preemption for fast, detrministic response time
- Fast file system I/O
- Miscallaneous performance improvements

3.2 Added Functionality

3.2.1 Priority-Based Preemptive Scheduling Priority-based preemptive scheduling lets the most important process execute first, so that it can respond to events as soon as possible. The most important process executes until it sleeps voluntary or finishes executing, or until a more important process preempts it. Priority based means that a more important process can be assigned a priority higher than other processes, so that the important (high priority) process will execute before other processes. Preemptive means that the high priority process can interrupt or preempt the execution of a lower priority process, instead of waiting for it to be preempted by the opereting system when its time slice is completed or it needs to block.

The scheduling policy of traditional UNIX systems strives for fairness to all users and acceptable response time for terminal users. The kernel dynamicelly adjusts process priorities, favoring interactive processes with light CPU usage at the expense of those using the CPU heavily. Users are given some control of priorities with the nice(2) system call, but the nice value is only one factor in the scheduling formula. As a result, it is difficult or impossible to guarantee that one process has a priority greater than another process. Therefore, each process in a traditional UNIX system effectively has to wait its turn, no matter how important it might be to the real-time application.

HP-UX presents a solution to this problem by adding a new range of priorities, called real-time priorities. Priorities in the real-time range do not fluctuate like priorities in the normal range, and any process with a priority in the real-time range is favored over any process with a priority in the normal range, including those making system calls and even system processes. Important as real-time processes are, interrupt processing is given priority over them. If several real-time processes have the same priority, they are timesliced.

Processes with real-time priorities are favored not only for receiving CPU time, but also are favored for swapping and for file system access. Real-time processes are the last to be swapped out (except for locked processes), and the first to be swapped in. File system requests for real-time processes go to the head of the disk request queue. All of this preferential treatment gives real-time processes very good response, but at the expense of the rest of the system.

Real-time priorities are set by the user either programmaticelly with HP's new rtprio(2) system call, or interactively with the rtprio(1) command. By default, processes are time-shared and continue to be executed according to the normal scheduling policy. Aside from setting a process to a real-time priority, the rtprio(2) system call and rtprio(1) command can be used to read the priority of a real-time process and change the priority of a real-time process to be timeshared.

3.2.2 Process Memory Locking A second important feature in a real-time system is the ability to lock a process in memory so that it can execute without waiting to be paged in or swapped in from disk. In the UNIX and HP-UX operating systems, processes are not normally locked in memory; they are swapped and/or paged in from disk as needed.

HP-UX has adopted a solution to this problem from System V. The plock(2) system call allows a process to lock its executable code and/or its data in memory to avoid unexpected swapping and paging. Also, a process can lock additional data or stack space with the datalock(3C) subroutine, and lock shared memory segments as needed with the shmctl(2) system call.

3.2.3 Controling Access to Real-Time Capabilities Because the priority scheduling and mamory locking featuras of HP-UX are quite powerfull, it is desirable to allow only certain users to access them. If, for example, all users had acces to these capabilities, they could potentially set all of their processes to a high real-time priority and try locking them in memory, which would defeat the purpose of the real-time system. Or a novice user could assign real-time priority of 0 to a process that happens to execute an infinite loop, thus locking up the entire system.

To prevent scenarios such as these. HP-UX created a feature called privilege groups. Privilege groups enable certain users (other than just the superuser) to access the powerfull real-time priority and memory locking features of HP-UX. A privilege group is a group to which the superuser assigns privileges. Existing privileges are real-time priority assignment (RTPRIO), memory locking (MLOCK) and a third not real-time related privilege (CHOWN). The superuser assigns one or more of these privileges to one or more groups with HP's setprivgrp(2) systemcall or setprivgrp(1) command, and assigns certain users to become members of these groups with the 4.2BSD-based setgroups(2) systemcall.

3.2.4 Fine Timer Resolution and Time-Scheduling Another important feature in real-time operating systems is fine timer resolution and time-scheduling capabilities. For high resolution clock based applications, both repetitive and nonrepetitive, it is important to be able to execute a process or subroutine at a precise time. For example, a real-time application might require various sensor readings at 20 millisecond intervals.

Standard features in System V that deal with time, such as alarm(2) which has a resolution of one second, and cron(1) and at(1) which have a resolution of a minute, are not precise enough for many real-tima applications. Therefore, HP-UX has adopted a solution from 4.2BSD, known as interval timers. Each process can enable its own interval timer to interrupt itself once or at repeated intervals, with whatever precision the underlying hardware and operating system can support. The interval is defined in units of

seconds and microseconds to keep the timer interface portable despite the system dependant resolution. For HP-UX on the Series 800, the system clock resolution for scheduling alarms is 10 milliseconds for measuring time 1 microsecond.

3.2.5 Interprocess communication and Synchronization A real-time operating system must provide interprocess communication and synchronization facilities. Interprocess communication and synchronization is important because real-time applications often involve several asynchronous processes that need to exchange information.

Pipes and signals are common interprocess communication facilities in the UNIX operating system. A pipe is essentially an I/O channel through which data is passed with the read(2) and write(2) system calls. An advantage of using a pipe is that it provides synchronization by blocking reader processes when the channel is empty end blocking writer processes when the channel is full. The disedvantage of using pipes are 1) they require the communicating processes to have a common ancestor process that sets up the channel, and 2) they are often slow because the kernel has to copy the data from the writer process to the system buffer cache and then back again to the address space of the reader process. Many UNIX systems including HP-UX support named pipes, which overcome the first problem, but still have the performance penalty of copying data.

A signal is essentially a software interrupt sent to a process by the kernel or by a user process. A process can install a handler for almost any signal, and the handler will execute when the signal is received. Signals can be a good event or alarm machanism because one process can send a signal to inform another process that an event occured, and then the other process can immediately enter its handler to respond to the event. The disadvantages of using signals are 1) they pass little or no data (not even who the sender process is), and 2) they are traditionally unreliable when sent repeatedly or when a process tries to wait for a signal. HP-UX has therefore adopted e reliable signal interface from 4.2BSD, in addition to the System V signal interface. The 4.2BSD signal interface solves the reliability problems of the system V interface, but is more complicated to use. The 4.2BSD signal interface introduces blocking of signals in similar manner as interrupts can be masked on the hardware level.

HP-UX has adopted three IPC (Inter Process Communication) facilities from System V: shared memory, semaphores and messages. These facilities

allow communication and synchronization between arbitrary unrelated processes. An elaborate semaphore facility allows solutions to both simple and complex synchronization problems. A message passing facility allows transfer of arbitrary data structures, along with the ability to prioritize messages. The most important IPC facility for real-time applications is the shared memory facility. Two or more processes can attach the same segment of memory to their data space and read from and write to it. Shared memory can be locked into physical memory.

3.2.6 Asynchronous I/O Asynchronous I/O is I/O that overlaps with process execution or other I/O, typically resulting in increased throughput. Both the UNIX and HP-UX operating systems implement system asynchronous I/O to certain drivers, but HP-UX allows you to communicate with some drivers that do system asynchronous I/O, so you can take advantage of their asynchronous ability.

HP-UX implements system asynchronous facilities for terminals, pipes, named pipes and sockets. The asynchronous I/O facilities that HP-UX provides for terminals are:

The nonblocking I/O facility: Before launching an I/O request, a user process can set a flag to inform the driver that the driver should cause the I/O request to return immediately if the request can not be satisfied without blocking the user process. The request may return partially satisfied.

The SIGIO facility: Before launching an I/O request, a user process can set a flag to enable the driver to send the SIGIO signal to the process when the data has arrived in the drivers input buffer.

The select(2) facility: A user process can call select(2) to check if an I/O request should be issued to one or more devices. The driver sets a bit in a user supplied bit mask for each file descriptor that the user asked about and on which I/O can be performed.

The FIONREAD facility: Before launching a read(2) request, a user can ask the driver to tell it how many characters in the drivers input buffer are available for reading.

These facilities can be used individually or together. For example, suppose you want to read from several terminals and you are not sure, which terminal will send you data or when

to expect this data, if any. You do not want to launch a series of read(2) requests to each terminal, because you might end up missing data from one or more terminals as you try to read from some terminal that will never send you data. Instead, you could enable the SIGIO facility for each terminal so that each can inform you when data has arrived in its input buffer. When SIGIO is sent, you could call select(2) to find out which terminal(s) are ready for reading.

3.2.7 Synchronous I/O A real-time application sometimes prefers to do synchronous I/O operations to make sure that its I/O request actually completed. In synchronous I/O, a process initiates an I/O request and then suspends until the I/O request completes. The file system normally does asynchronous writes, which means that a write(2) returns when the data has been only written to the buffer cache, not to the disk. The data is written from the buffer cache to the disk later, while the process continues to execute. Although this asynchronous disk write increases your process's throughput, the disadvantage is that you cannot be sure that your data has actually been written to the disk. Therefore, HP-UX provides a flag called O_SYNCIO that lets you perform a synchronous disk write. This ensures, that your data actually was written to disk.

3.2.8 Summary of Real-Time Functionality Added to HP-UX Table 2 summarizes the functionality that was added to HP-UX. It presents the system call associated with the particular feature and the origin of the system call (either System V, 4.2BSD or HP).

3.3 Performance Tuning

The performance tuning that HP has implemented in HP-UX on the Series 800 computers is as important as the added real-time functionality. The following features, kernel preemtion, fast file system I/O, and miscellaneous performance improvements comprise the main part of HP's performance tuning efforts.

3.3.1 Kernel Preemption for Faster Response Time An important requirement for real-time systems is a quick and deterministic response time. One of the main concerns about the real-time capability of UNIX systems is that a process can execute in kernel mode for long periods of time (more than 1 second) without allowing a higher priority process to preempt it. Instead, the process keeps executing in kernel mode until it blocks or finishes, while the high priority process must wait. A process executing in user mode gets preempted much more quickly.

Table 2: Real-Time Functionality in HP-UX

Real-Time Function	Associated system call	Origin
Priority based preemptive scheduling	rtprio(2)	HP
Memory locking	plock(2)	System V
Privilege groups	getprivgrp(2) setprivgrp(2)	HP and 4.2BSD
Fine timer and time scheduling	setitimer(2) gettimeofday(2)	4.2BSD
Reliable signals	sigvector(2) sigblock(2) sigpause(2)	4.2BSD
Shared Memory	shmget(2) shmctl(2) shmop(2)	System V
Other IPC facilities	pipe(2) msg{getlctllop} sem{getlctllop}	System V
Asynchronous I/O	ioctl(2) flags select(2)	HP and 4.2BSD
Synchronous I/O	fcntl(2) with O_SYNCIO	HP
Preallocating disk space	prealloc(2)	HP
Powerfail recovery	signal(2) with SIGPWR	HP and System V

HP-UX on the Series 800 solves this problem by implementing a preemptable kernel. At certain safe places in the kernel called preemption points or preemption regions, HP-UX keeps kernel structures at a consistent state, so that a higher priority real-time process can get control of the CPU at that point.

Implementation of Kernel Preemption

Two types of preemption have been implemented: there is the synchronous method which allows preemption at a specific point during kernel execution, and a asynchronous method

which allows preemption anywhere during a region of kernel execution. When a higher priority real-time process becomes runnable, kernel preemption is requested by setting the reqkpreempt flag and generating a hardware supported interrupt. The now pending preemption request is serviced immediately if the running process is in user mode or in a preemptable kernel region, or at the first point when either:

a. the KPREEMPTPOINT() macro is executed which tests the reqkpreempt flag to allow a synchronous preemption,
b. the spl level drops to splpreemptok() which allows the pending preemption interrupt and thereby a asynchronous preemption,
c. user mode is entered (one case where the spl level drops to splpreemptok()), or
d. swtch() is called which always transfers to the highest priority runnable process.

The reqkpreempt flag and the preemption interrupt are both cleared by swtch() whenever it switches to a new (highest priority) process. In total, approximately 180 synchronous preemption points and 20 asynchronous preemption regions were added to the HP-UX kernel.

Limitations

There is one overriding limitation on what kernel preemption can accomplish: Kernel preemption can only preempt an operation which is being executed within a process context. It cannot preempt interrupt processing code and allow a process to execute because UNIX does not support this type of operation. Therefore, all interrupt processing is implicitly considered to be of higher priority than any (realtime) process. Interrupt requests mainly come from the I/O system but additionally 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 specific time offset. The procedure is invoked from a interrupt processing context during clock interrupt processing. To minimize callout queue execution overhead, in HP-UX a seperate system process was created to provide a preemptable process context in which to execute some lengthy callout queue code like gathering statistics and recalculating timesharing priorities.

Measured Improvement

In order 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. Timing measurements are taken by sampling the time at kernel entry and exit and also whenever the kernel changed between preemptable and non preempable state. The time intervals during which the kernel executes in non preemptable state are logged together with the pc stack traces of the start and the end of the interval. The time spent in the interrupt context is also logged.

To determine the improvement made in real-time dispatch time, two sets of measurements were taken. One set with kernel preemption enabled and one set with it disabled under the same workload. The workload consisted of tests which validate the correct working of all kernel functions.

The raw results consist of a stream of times. Each time represents an interval when the kernel was non preemptable. As one way of summarizing the results, a distribution was computed which represents the percentage of total kernel execution time that was spent in non preemptable codepaths of less than x milliseconds. Figure 1 shows this distribution for both preemption on and preemption off.

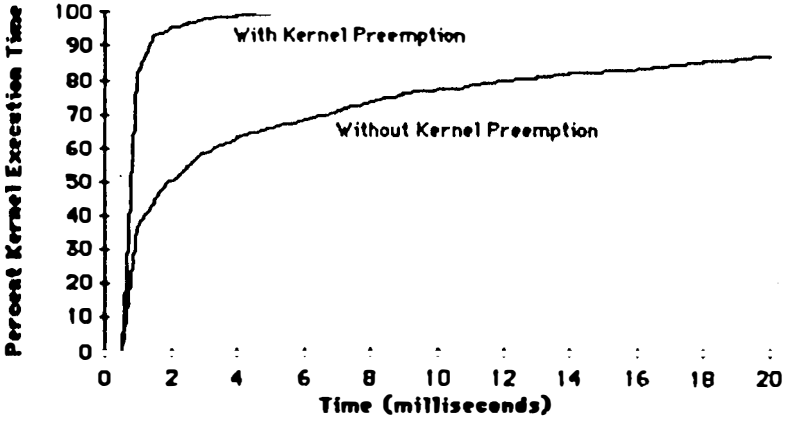

Figure 1

Table 3 shows that HP-UX kernel preemption has provided significant improvements in real-time process dispatch time. In the worst case observed, the improvement was well over 50 fold. In the case where preemption is off the longest non-preemptable code paths are typically large data copies during process creation (fork), process overlay (exec), or user I/O operations. In the cases where preemption is on, the current longest code paths are now in the terminal driver.

Table 3: Non-preemptable Kernel Time			
	Preemption Off	Preemption On	Improvement
90% kernel	40 ms	1.4 ms	x28
99% kernel	129 ms	3.4 ms	x37
max kernel	1127 ms	14.6 ms	x77

These results represent the status of HP's preemption tuning activities as of this writing; work is currently underway to further reduce these times. Future work will entail further improvements to real-time performance via increased kernel semaphoring in order to address more stringent application needs.

3.3.2 Fast File System I/O The traditional UNIX filesystem does not meet the performance and reliability requirements of real-time systems for the following reasons:

- Data blocks are often scattered randomly throughout the disk, resulting in large seek times for sequential reeds.
- The Data blocksize of 512 or 1024 can be inefficient for large read and write requests.
- There is only one copy of the superblock containing all vital data about the file system structure on the disk. If it gets corrupted all data on the filesystem is lost.

The HP-UX file system has adopted its solution from the McKusick or 4.2BSD file system. Two important features in the HP-UX file system are the implementation of cylinder groups, which reduce file seek time and add raliability, and the addition of two block sizes for increased speed without wasting space. Cylinder groups together with allocation algorithems provide locality for related data and spread unrelated data resulting in short seek times. Eech cylinder group has a copy of the vital superblock information for reliability. The HP-UX file system uses a hybrid block size to deal with the time and space tradeoff of big versus small size blocks. There is a blocksize of 4K or 8K and a fragment size of 1/8, 1/4, 1/2 or the same size as the block size. Large file I/O requests are allocated and accessed a block at a time, while smaller requests are alloceted and accesed a fragment at a time.

3.3.3 Miscellaneous Performance Improvements HP-UX on series 800 is tuned for both real-time response and throughput. Benchmarks representing various workloads (for real-time and other enviroments) were run to track and improve the performance of specific paths in the operating

system. Other measures such as time from interrupt to driver were measured with a logic analyzer interface to the hardware. This systematic aproach to performance tuning led to very significant results, with many performance measures improving by a factor of two or more during product development. Also this approach led to justifiable returns, including support for two-hand clock replacement algorithm and conversion of various kernel data structures from linear lists to hashed lists.

4. Conclusion

The functionality additions and performance improvements described in this paper form the foundation by which HP is enabling its version of the UNIX operating system to successfully enter the real-time marketplace. The features decribed are rather simple to implement, and in fact, most of them are already in System V or 4.2BSD. HP is working with the IEEE P1003 committee and the real-time subcommittee to help form a common standard by which any vendor can gain the needed functionality.

References

1. David C. Lennert; "Decreasing Realtime Process Dispatch Latency Through Kernel Preemption"; USENIX Conference Proceedings, Summer 1986, pages 405-413
2. Suzanne M Doughty, Sol F. Kavy, Steve R. Kusmer, Douglas V. Larson; "Unix for Real Time"; UniForum Conference Proceedings, Winter 1987, page 219-230