INTERRUPTS

Interrupts are one solution to signaling asynchronous events to a host computer. This article begins with a discussion of the problem that is being addressed, provides an analytic model to evaluate the design space of event-signaling algorithms, and then illustrates some trade-offs using experiments performed with a 622 Mbit/s ATM adapter for computer workstations. The article concludes with a review of current work in event signaling.

WHY INTERRUPTS?

Operating systems are software systems that manage the hardware resources of a computer system to provide services needed by applications. They evolved from input/output control systems (IOCSs) that were loaded into early computer systems before an application began to run; this was typically done with a deck of punched cards placed immediately ahead of the cards used for the application. It became clear that there was a common set of functions needed by many applications, and this gave rise to early operating systems, which were heavily focused on job service (to maximize the utilization of the expensive machine) and device management.

The main evolutions in operating systems arose from different paradigms for sharing the machine. Early machines in many cases were dedicated to a single use. Later machines were multipurpose, but the per-application IOCS scheme made for sequential execution of jobs, one after another. When IOCS became shared and job management was used to increase utilization of the machine, *spooling* was used to queue work for batch execution. A major advance came from the idea of *multiprogramming,* which took advantage of the fact that the expensive processor was often wasted (idle) as slow input/output devices (such as printers, card punches, and tape machines) were accessed by an application.

Multiprogramming used the idle periods of the processor to perform other computational work until the input/output was completed. A variety of multiprogramming techniques were developed, with fixed and variable numbers of tasks, priorities, and so on. Timesharing is a multiprogramming technique that allows interactive access to the multiprogrammed resources. Access is controlled under a *policy,* such as "fairness." Timesharing systems often periodically schedule job execution in a "round-robin" fashion to preserve a fair allocation of processing resources between jobs. This creates a "virtual time" model, where each job's real processing time (wall-clock time) is dilated in proportion to the amount of competition for processing resources. This scheduling model is typically *preemptive* and is accomplished via use of a hardware alarm timer that generates an *interrupt.* The operating system's interrupt service discipline for this timer event may involve choosing a new job to occupy the processor. The steps of preserving the state of the previous job and loading the saved state of the new job comprise a *context switch.*

MULTIPROCESSING, INTERRUPTS, AND SCHEDULING

The key resource management policy in a multiprocessing system is the scheduling policy, used to decide which of the available processes will occupy the processor. Scheduling can be implemented in two forms, namely, nonpreemptive and preemptive. In the first case, the operating system makes a scheduling decision and the process occupies the processor until it is finished with its current work. In the second case, the operating system may preempt the process, perhaps allocating the processor to a different process in order to implement the scheduling policy. In either of these cases, there is a significant amount of machine state that must be saved and restored for the processor to be allocated to a process. While the machine state to be saved and restored varies with the operating system, the typical process state information that must be saved and restored includes:

- A set of machine registers, including a program counter and a stack pointer
- A set of virtual memory mappings for the process's address space
- A set of pointers and status information used by the operating system to describe the process, such as a priority and an execution privilege

In addition, cache entries must be flushed so that there is no difficulty with future references to memory.

One of the interesting trade-offs that has arisen as a consequence of technology trends is the heavy use of caching techniques and technology to reduce the cost of memory access for computationally intensive programs. Large register sets, characteristic of reduced instruction-set computing (RISC) technology, can be viewed as a compiler-managed cache area. A result of this use of caches is that the process executes more quickly once the cached data are available, but as the amount of preserved state per process rises, the cost of a preemption does as well.

Modern operating systems are typically preemptive in design, as it is believed that the operating system can do a better job of making decisions on a continuous basis than if it has decision points chosen, in effect, by applications processes. When systems are organized so that applications can be preempted, there is typically a hierarchy of scheduling priorities applied so that the highest-priority runnable process is always on the processor. The operating system will then be assigned a set of priorities higher than application priorities so that its operations can complete before application processes are allowed to occupy the machine. The assignment of priorities used for scheduling thus reflects the policy of the operating system designers about which operations should take precedence in the job mix. The preemption is implemented via a high-priority hardware "alarm clock," which generates an interrupt. The clock interrupt routine becomes the highest-priority runnable process at this point, and it operates the scheduling algorithm to determine the next process to occupy the processor.

Clocks, Preemption, and Priorities

As any multiprocessing system can be looked at as a time-division multiplexing (TDM) scheme for processors, timesharing systems such as UNIX and its derivatives can be viewed as statistical TDM schemes. The multiplexing is provided by means of a system clock, which is set to periodically "interrupt" the processor at a known rate. The period of this clock is known as a clock "tick." The events that occur at each tick are roughly as follows:

- An interrupt vector is used to execute a hard clock interrupt routine, resulting in the currently executing process having its state saved and control passing to the clock service code. The clock service code may update some internal operating system state, such as the number of clock ticks the current process has accumulated, before other work proceeds.

- The operating system examines a queue of activities tied to a clock event—the elements of this queue are used, for example, to periodically examine teletype devices for activity. The queue is typically organized as a sorted list so that the elements can be examined in order of their timer expiry. Elements later in the queue have their timer expiries stored as a time offset from the previous element, so that all queue element timers are updated when the head of the list is updated.
- The head of the list's timer expiry is decremented by one tick. Any queue elements that have a timer expiry of zero are executed, and the first queue element with a nonzero timer expiry becomes the new head of the list.
- The operating system selects the next runnable process using its policy—for example, that the highest-priority runnable process should always be running—and restores the saved state of that process to restart execution. It is worth noting that this may well be a different process than was executing when the clock tick occurred. For example, if the previously running process has accumulated a clock tick, its priority may have decreased to the point where another process will be selected for execution. With proper design of the algorithm for choosing the next process to execute (e.g., round-robin within priority bands) effective timesharing can take place.

Unscheduled Preemption—Device Interrupts

Multiprocessing systems are designed under the assumption that there is always an oversupply of useful work to do. Device management policy reflects this assumption by way of event-signaling schemes. What event-signaling means is that the device performs some operation, say, to transfer a packet from user storage to a cellified representation on an ATM network, and this operation must be noted in the control algorithm for the device. This control algorithm may want to signal the device to begin a transfer or pass information to the device for later use, such as a pool of buffer addresses.

Interrupts in UNIX Multiprocessing

As illustrated in Fig. 1, there are events called interrupts that might result in the preemption of the process. These events

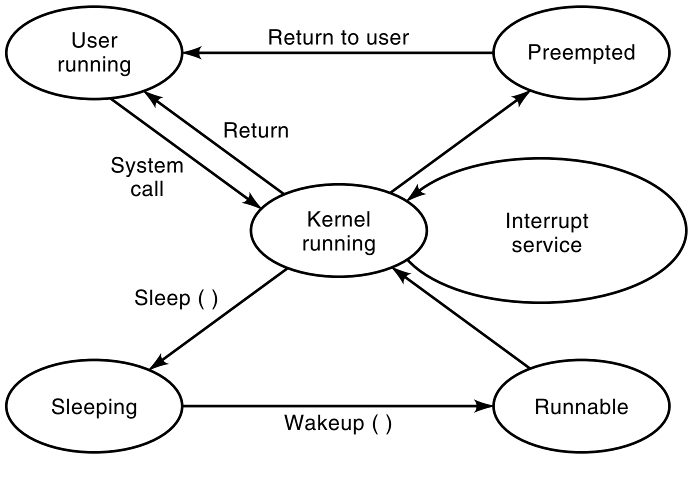

Figure 1. UNIX processes—an abstract state diagram.

are caused by devices that signal that they need service. The service routines are called *device drivers* and consist of (logically) a *top half* and a *bottom half.* The bottom half services are accessed when a device interrupts. The device interrupts by asserting a signal on a control line, and this causes control of the processor to pass to a routine located at an interrupt service vector, which is a small integer used to demultiplex the different devices that might require service. In the lowest levels of UNIX, in fact at the lowest addresses in the system (and often in assembly language, e.g., `locore.s`), reside the mappings between the small integers and addresses of routines to service the interrupts, per device.

Among the usual properties of an interrupt handler is its priority level. When the interrupt handler needs atomic execution, it sets the processor priority level above its own execution level. This has the desirable property that any other device of its type, which may share data structures in a critical section, will not execute. While clever programming can minimize the duration of these "locks" on data structures, they do represent a difficulty for parallel processors, as the locking strategy does not work unless all input/output (I/O) is done through a single processor.

ANALYZING INTERRUPT PERFORMANCE

Consider a system with an interrupt service overhead of C seconds, and k active channels, each with events arriving at an average rate of λ events per second. Independent of interrupt service, each event costs α seconds to service, for example, to transfer the data from the device. The offered traffic is $\lambda * k$, and in a system based on an interrupt-per-event, the total overhead will be $\lambda * k * (C + \alpha)$. Since the maximum number of events serviced per second will be $1/C + \alpha$, the relationship between parameters is that $1 > \lambda * k * (C + \alpha)$. Assuming that C and α are for the most part fixed, we can increase the number of active channels and reduce the arrival rate on each, or we can increase the arrival rate and decrease the number of active channels.

For devices with large data transfers such as disk devices, the data transfer per interrupt event is large and thus the interrupt service overhead is negligible. On the other hand, devices with small units of data transfer can be severely limited by interrupt processing overhead. An example of this situation is a computer used for receiving character data (e.g., one that supports a modem pool) from a large number of devices. One such system was studied during the early 1980s, which was used for receiving `netnews` via the UNIX `uucp` data transfer mechanism; the system was all but unusable for interactive work and traces showed that more than 90% of its time was spent in interrupt service.

AN ALTERNATIVE EVENT-SIGNALING SCHEME: CLOCKED INTERRUPTS

Event-signaling within the network subsystem between the hardware network interface device and the software device driver is typically accomplished via polling or device-generated interrupts. In an implementation of an OC-3c ATM host interface for the IBM RS/6000 family of workstations (1), the traditional forms of this crucial function were replaced with "clocked interrupts." Clocked interrupts, like polling, examine the state of the network interface to observe events that require host operations to be performed. Unlike polling, which requires a thread of execution to continually examine the network interface's state, clocked interrupts perform this examination periodically upon the expiration of a fine-granularity timer. In comparison to interrupts, clocked interrupts are generated indirectly by the timer and not directly by the state change event.

Clocked interrupts may negatively affect the latency of the networking subsystem, but they can improve the bandwidth, which can be handled under a variety of traffic types, as multiple changes of state can be detected by a single clocked interrupt. An analytical model for clocked interrupt performance has been developed (2).

Using the parameters of the previous section, for clocked interrupts delivered at a rate β per second, the capacity limit is $1 > \beta * C + \lambda * k * \alpha$. Since α is very small for small units such as characters, and C is very large, it makes sense to use clocked interrupts, especially when a reasonable value of β can be employed. In the case of modern workstations, C is about a millisecond. Note that as the traffic level rises, more work is done on each clock "tick," so that the data transfer rate $\lambda * k * \alpha$ asymptotically bounds the system performance, rather than the interrupt service rate. Traditional interrupt service schemes can be improved, for example, by aggregating traffic into larger packets (this reduces λ significantly, while typically causing a slight increase in α), by using an interrupt on one channel to prompt scanning of other channels, or masking interrupts and polling some traffic intensity threshold.

For application workloads characterized by high throughput, heavy multiplexing, or "real-time" traffic, clocked interrupts should be more effective than either traditional polling or interrupts. For these intensive work loads, our analysis predicted that clocked interrupts should generate fewer context switches than traditional interrupts and require fewer CPU cycles than polling without significantly increasing the latency observed by the applications. For traditional interrupts with interrupt service routines that detect additional packets enqueued on the adapter, many of the same benefits may accrue. Ramakrishnan (3) has noted a problematic performance overload phenomenon known as receive livelock, which clocked interrupts can help alleviate.

EVALUATING INTERRUPTS: THE HP AFTERBURNER AND UPENN ATM LINK ADAPTER

The OC-12c rate ATM Link Adapter for the HP Bristol Laboratories "Afterburner" was built to test scalability of an ATM host interface architecture (1,2) developed as part of the ATM/SONET infrastructure of the AURORA Gigabit Testbed (4).

The hardware infrastructure for this evaluation consists of HP 9000/700 series workstations equipped with Afterburner generic interface cards and ATM Link Adapters. The remainder of this section briefly describes the architecture and implementation of the Afterburner and ATM Link Adapter.

Afterburner

The Afterburner (5,6), developed by HP Laboratories in Bristol, England, is based on Van Jacobson's *WITLESS* architecture. It provides a high-speed generic packet interface that attaches to the SGC bus of the HP 9000/700 workstations. A large pool of triple ported video RAM (VRAM) is provided by Afterburner. The random access port of the VRAM is visible on the SGC bus, allowing the VRAM to be mapped into the virtual address space of the workstation. The two serial ports are used to provide a bidirectional FIFOed interface to a network specific *Link Adapter.* Several additional first-in, first-out queues (FIFOs) are provided to assist in the management of VRAM buffer tags.

ATM Link Adapter

A Link Adapter provides an interface between the general purpose Afterburner and a specific network technology. The UPenn segmentation and reassembly (SAR) architecture (1) is the basis for the ATM Link Adapter. This architecture performs all per-cell SAR and ATM layer function in a heavily pipelined manner, which can be implemented in a range of hardware technologies. For the ATM Link Adapter the base SAR architecture has been extended to support a larger SAR buffer (up to 2 Mbyte), ATM Adaptation Layer (AAL) 5 including CRC32 (cyclic redundancy check) generation and checking, and demultiplexing based on the full virtual path identifier (VPI), virtual channel identifier (VCI), and message identifier (MID). The performance of the implementation has been improved to 640 Mbit/s by using more advanced electrically programmable logic device (EPLD) technology. Figure 2 shows the host/Afterburner/ATM Link Adapter configuration.

IMPLEMENTATION OF THE CLOCKED INTERRUPT SCHEME ON THE AFTERBURNER ATM LINK ADAPTER

The ATM Link Adapter device driver operates in conjunction with HP Bristol "Single-Copy" TCP/IP (7). The kernel was

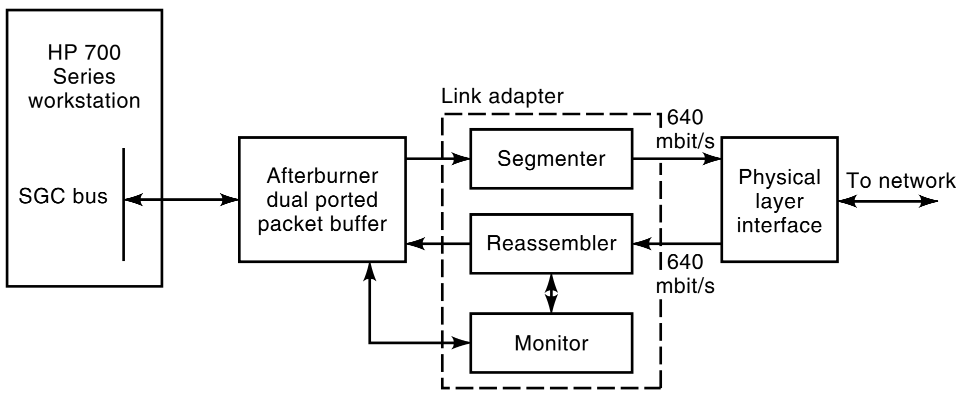

Figure 2. ATM Link Adapter.

Table 1. TCP/IP Throughput (Mbit/s), Afterburner ATM Link Adapter on HP 735s, 32 kbyte Messages

Socket Buffer Size (kbytes)	Traditional Interrupt	Poll	Clock 500 Hz	Clock 1 kHz	Clock 2 kHz	Clock 4 kHz
1	6.75	6.34	2.60	3.92	5.88	6.67
2	12.45	13.24	5.02	7.36	9.81	11.94
4	20.82	22.43	9.28	13.40	18.17	21.57
8	30.80	37.27	16.20	22.96	26.58	35.35
16	51.73	50.03	21.72	42.03	45.64	50.35
32	66.83	64.02	37.95	52.26	61.72	64.27
64	76.25	76.78	57.17	65.27	70.91	73.22
128	124.97	81.57	95.00	110.03	117.24	121.09
256	144.05	82.62	143.76	144.10	143.59	143.81

modified to support a fine-granularity timer, as the standard 100 Hz soft clock rate was inadequate. The operating system was modified to increase the hardware clock interrupt rate, and changing the interrupt service vector to point to a specialized clock service routine rather than the usual `hardclock` interrupt service routine. Clock division is performed inside the software, which calls the `hardclock` interrupt service code at the proper rate. At each vector clock tick, occurring at the clocked interrupt clock rate, the link adapter is examined for packet arrivals. If packets are discovered the interrupt service routine (ISR) for the ATM link adapter is invoked; this ISR provides the packet to the single-copy TCP/IP stack.

Polling requires a continuous thread of execution to examine the state of the I/O device. Because the version of HP-UX used for this evaluation lacks preemptive kernel threads, polling was implemented with a preemptable user process. To minimize the number of system calls, the device status flag was appropriately memory mapped for access by a user process. This allowed a user process to continually examine the state of the device in a preemptable thread of execution, albeit at some cost in overhead. The user process invokes the ISR through an `ioctl()` call; for measurement purposes a small helper daemon was devised, which performed this function, rather than modifying the `netperf` measurement tool, again at a cost in overhead. Preemptive kernel threads would remove both these additional sources of overhead.

Thus, the current implementation includes support for interrupt generation as well as the examination of the card via polling or clocked interrupts. With support for all three types of state change notification, a comparative experimental evaluation of these mechanisms can be performed.

PERFORMANCE

The hardware test configuration consists of two HP 9000 Series 700 workstations connected back-to-back via their Afterburner ATM Link Adapter subsystems.

Measurements and Analysis

The throughput of the resulting network stacks was measured using the `netperf` tool (8).

The results are given in Tables 1–3. Table 1 gives measured throughputs for 32 kbyte messages taken on HP 735s interconnected by adapters, which were in turn interconnected by a 160 Mbps–capable synchronous optical network (SONET)-like "null-modem." Table 2 gives measured throughputs for 32 kbyte messages taken on HP 755s (a faster version of the HP 735) interconnected by adapters, which are in turn connected by a 320 Mbit/s–capable SONET "null-modem." Table 3 repeats these measurements with a CPU intensive artificial work load running on the receiving CPU.

The major observation in comparing event-signaling is that polling does not keep up with the two other schemes above about 32 kbytes. All checksums were enabled for all tests; the measurements were performed on dedicated processors, with no other activity except for necessary system background processes. The tests were run with symmetric configurations; that is, both sender and receiver were using the same signaling mechanism.

It is clear from the figures shown that at high polling rates, the clocked interrupt scheme is able to keep up with the traditional interrupt scheme, which is almost everywhere the best

Table 2. TCP/IP Throughput (Mbps), Afterburner ATM Link Adapter on HP 755s, 32 kbyte Messages

Socket Buffer Size (kbytes)	Traditional Interrupt	Poll	Clock 500 Hz	Clock 1 kHz	Clock 2 kHz	Clock 4 kHz	Clock 2.5 kHz
1	13.17	13.76	3.16	5.88	7.95	11.73	8.85
2	23.40	24.25	6.82	10.59	14.99	19.46	16.90
4	38.07	42.92	11.96	16.29	26.33	38.44	34.39
8	57.04	64.61	23.29	31.60	43.58	56.88	53.46
16	96.02	91.32	35.80	51.05	71.05	87.80	68.77
32	118.15	105.12	59.47	86.43	101.12	111.03	100.28
64	133.52	107.02	77.89	103.14	119.93	126.96	123.40
128	196.51	126.12	123.50	167.28	187.69	196.39	191.63
256	210.66	136.77	210.53	214.77	214.87	213.46	215.15

Table 3. TCP/IP Throughput (Mbit/s), Afterburner ATM Link Adapter on CPU-Loaded HP 755s, 32 kbyte Messages

Socket Buffer Size (kbytes)	Traditional Interrupt	Poll	Clock 500 Hz	Clock 1 kHz	Clock 2 kHz	Clock 4 kHz	Clock 169 kHz
1	11.82	7.43	3.63	4.89	7.76	9.45	1.38
2	21.16	13.37	6.35	9.17	14.40	17.20	2.76
4	33.32	23.53	13.78	25.22	26.03	24.09	5.53
8	47.49	34.57	16.31	31.03	38.73	45.81	8.70
16	60.34	45.31	34.68	49.93	78.89	62.35	21.70
32	72.99	54.76	60.70	85.98	72.56	86.10	22.12
64	83.14	63.36	92.07	79.83	66.11	65.24	54.61
128	92.48	66.78	108.99	90.62	102.90	81.75	76.64
256	95.29	76.26	95.68	106.57	97.08	102.44	166.44

performer, with the exception of polling, which does best for small packet sizes. In a lightly loaded environment, interrupts would appear to be the best solution, except for some anomalous, but repeatable results, which show polling best for small socket buffer sizes.

Performance and Work Load

Since dedicated configurations are not characteristic of real environments, which are often loaded with other work and other network traffic, we created an artificial work load by continuously executing a `factor 99121010311157` command. This has a significant effect on the behavior of the three schemes, as can be seen by measuring the throughput with `netperf` with the artificial work load running on the receiver.

Latency and Event-Signaling

A second important parameter for distributed applications is the round-trip latency induced by the software supporting the adapter. Since the hardware was a constant, we could directly compare the software overheads of the three schemes. This was done with the following test. An artificial network load was created using `netperf` with a socket buffer size of 262,144 bytes and operating it continuously. Against this background load, Internet control message protocol (ICMP) European Commission Host Organization (ECHO) packets of 4 kbytes were sent to the TCP/IP receiver, which was where the event-signaling performance differences would be evident. Sixty tests were done to remove anomalies. Our results showed that traditional interrupts and clocked interrupts at 500 Hz performed similarly, yielding minimum, average, and worst times of 5/12/18 ms, and 4/11/25 ms, respectively. When the systems were not loaded, the performances were 3/3/3 ms and 4/4/6 ms. This suggests that clocked interrupts performed slightly better under heavy load, but slightly worse under unloaded conditions, confirming the analysis given earlier.

SUMMARY AND RECENT WORK

Work per event is the most important factor, by far, in maximizing observed throughput. Thus, systems that employ interrupts should aggregate work, perhaps in the form of larger data objects. An example of this is using interrupt-per-packet rather than interrupt-per-cell, in an ATM context. Buffering is an effective aggregation mechanism and has often been employed to support character-oriented I/O systems. Even newer schemes, such as Mukherjee's (9), use polling at one level of the system (coherence enforcement) to reduce the overall cost. Mogul and Ramakrishnan (10) have developed a hybrid interrupt/polling technique that uses queue length to convert from interrupt-driven signaling to polling; they report good performance with a simple policy.

The experiments described in this article showed the following. First, in the context of high-performance network adapters, clocked interrupts can provide throughput equivalent to the best throughput available from traditional interrupts; both methods provide better performance than polling as implemented here. Second, clocked interrupts provide higher throughput when the processor is loaded by a computationally intensive process; this suggests that clocked interrupts may be a feasible mechanism for heavily loaded systems such as servers, which might also suffer from Ramakrishnan's *receive livelock.* Third, clocked interrupts provide better round-trip delay performance for heavily loaded systems servicing large ICMP ECHO packets.

Taken as a whole, the data suggest that clocked interrupts may be an appropriate mechanism for many of the high performance applications now being proposed, such as Web proxies and other network traffic-intensive servers.

ACKNOWLEDGMENTS

Hewlett-Packard's European Research Laboratories (Bristol, UK) collaborated on the Afterburner ATM Link Adapter and provided the basic single-copy TCP stack. The AURORA research at the University of Pennsylvania was supported by the National Science Foundation and the Advanced Research Projects Agency under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives, by the NSF under agreement CDA-92-14924, by Bell Communications Research under Project DAWN, by an IBM Faculty Development Award, and by the Hewlett-Packard Corporation.

BIBLIOGRAPHY

1. C. Brendan, S. Traw, and J. M. Smith, Hardware/software organization of a high-performance ATM host interface, *IEEE J. Select Areas Commun.,* **11** (2): 240–253, 1993.
2. J. M. Smith and C. B. S. Traw, Giving applications access to Gb/s networking, *IEEE Network,* **7** (4): 44–52, 1993.
3. K. K. Ramakrishnan, Performance considerations in designing network interfaces, *IEEE J. Select. Areas Commun.,* **11** (2): 203–219, 1993.

4. D. D. Clark et al., The AURORA gigabit testbed, *Comput. Netw. ISDN Syst.,* **25** (6): 599–621, 1993.
5. C. Dalton et al., Afterburner: A network-independent card provides architectural support for high-performance protocols, *IEEE Netw.,* **7** (4): 36–43, 1993.
6. D. Banks and M. Prudence, A high-performance network architecture for a PA-RISC workstation, *IEEE J. Select. Areas Commun.,* **11** (2): 191–202, 1993.
7. A. Edwards et al., User-space protocols deliver high performance to applications on a low-cost Gb/s LAN, *Proc. 1994 SIGCOMM Conf.,* London, UK, 1994.
8. Hewlett-Packard Information Networks Division, *Netperf: A network performance benchmark* (Revision 2.0), Feb. 15, 1995.
9. S. Mukherjee and M. D. Hill, The Impact of Data Transfer and Buffering Alternatives on Network Interface Design, *4th HPCA,* 1998.
10. J. Mogul and K. Ramakrishnan, Eliminating Receive Livelock in an Interrupt-Driven Kernel, *Proc. USENIX Conf.,* San Diego, CA, 1996. (More data is available in a technical report version of the paper available from DEC WRL.)

Reading List

K. L. Thompson, UNIX implementation, *Bell Syst. Tech. J.,* **6** (2): 1931–1946, 1978.

C. Brendan, S. Traw, *Applying architectural parallelism in high performance network subsystems,* Ph.D. Thesis, CIS Dept., Univ. Pennsylvania, Jan. 1995.

J. T. van der Veen et al., Performance Modeling of a High Performance ATM Link Adapter, *Proc. 2nd Int. Conf. Comput. Commun. Netw.,* San Diego, CA, 1993.

JONATHAN M. SMITH
JEFFREY D. CHUNG
C. BRENDAN S. TRAW
University of Pennsylvania

INTERSYMBOL SIGNAL INTERFERENCE. See SYMBOL INTERFERENCE.