# Priority Consistency in Protocol Architectures

Clifford W. Mercer and Hideyuki Tokuda
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
{cwm,hxt}@cs.cmu.edu

**Abstract**

The protocol processing software of a multimedia operating system must provide fast response time and predictable delays for time-constrained data streams like digital audio and video streams. This paper describes several different techniques for scheduling the protocol processing of messages. These techniques are analyzed and their (simulated) performance compared using various metrics. One of these metrics is the *priority inversion factor* which provides a way of quantifying priority inversion in the system. Protocol processing time and context switch time are given as parameters in the simulations, and we present guidelines for choosing between the message scheduling techniques based on the ratio of protocol processing time to context switch time for a given system.

## 1 Introduction

A continuous media operating system must be highly preemptable so that *priority inversion* (the situation where a high priority activity is delayed by the execution of a lower priority activity) can be minimized. This provides good response time for high priority activities and better adherence to the priority structure defined by the system designer. Commercial real-time operating systems guarantee fast response time by structuring the kernel in a way that is highly preemptable. For example, if the scheduler is required to search a list of ready processes in order to make a scheduling decision, interrupts must be disabled to preserve the invariants of the list. But if the list is long, the scheduler may delay pending interrupts for a time. To avoid this problem, the search may be broken up into pieces to allow pending interrupts to be serviced at appropriate points during the course of the search. Designing a high degree of preemptability into a simple uniprocessor system is relatively easy, but the problem becomes more difficult in

This research was supported in part by the U.S. Naval Ocean Systems Center under contract number N66001-87-C-0155, by the Office of Naval Research under contract number N00014-84-K-0734, by the Federal Systems Division of IBM Corporation under University Agreement YA-278067, and by the SONY Corporation. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of NOSC, ONR, IBM, SONY, or the U.S. Government.

the context of a distributed environment where network devices must be handled and protocols must be executed.

In this position paper, we describe our on-going study of various structures for protocol processing software where we concentrate on the preemptability properties and efficiency of the techniques. We first describe the problem with current techniques used in network operating systems, and then we identify several techniques which vary in degree of preemptability and in efficiency. We introduce a metric for quantifying the priority inversion in a system, and we use this to compare the various techniques.

# 2 Protocol Processing Techniques

The techniques we consider in our study range from a method where protocol processing is done at a non-preemptable software interrupt level (a priority level higher than any user process but lower than the hardware interrupt level) to a method where prioritized, preemptable threads perform the protocol processing. We consider several techniques between these two extremes and attempt to identify the degree of preemptability that is required to service what we think might be a typical continuous media workload. We start with the software interrupt technique (SOFTINT) used in BSD4.3 in which priority inversion arises as a result of the FIFO queueing of messages as well as the non-preemptable nature of the software interrupt. Then we consider a single-threaded technique (T1F) where the thread executes at the highest priority in the system and the messages are again queued in FIFO order. This gives us the same basic characteristics except that there is some overhead in going to a threaded approach. The single-threaded technique with priority queueing allows us to evaluate the performance of the priority message queueing with a non-preemptable service which is easier to implement. This technique gives rise to priority inversion from the non-preemptability of the service, but the priority inversion due to queueing is eliminated. The problem with these approaches is that if the protocol thread is running at the highest priority in the system, low priority messages will be serviced at the expense of high priority local activity. So we consider a single-threaded technique where the priority of the thread is inherited (based on some mapping of message priority to thread priority) from the priority of the message (T1PI); i.e. the priority of the thread is low when servicing a low priority message. Of course, if a high priority message arrives while the low priority message is being serviced, that high priority message may have to wait for any medium priority threads which preempted the protocol thread (at low priority). This is another case of priority inversion. To solve this problem, we use priority "bumping" to increase the priority of the protocol thread when a high priority message is enqueued (T1PIB).

Finally, we consider two additional techniques for multi-threaded protocol engines which may have any number of threads; aside from the differences in these techniques, the number of threads has important implications for the performance and predictability as well. The two techniques differ in their method of assigning priorities to threads and their method of matching incoming messages to threads. The first technique assigns a fixed priority to each thread and queues incoming messages based on a match between the message priority and the thread priority. In the second technique, the priority changes dynamically depending on the priority of the message being processed. This technique is an extension of the single-threaded priority inheritance technique mentioned above, and the important issue is how to allocate threads to

incoming messages.

In our previous work [2, 9], we determined that the prioritized multi-threaded protocol processing model gives better schedulability (by means of a higher degree of preemptability) than the software interrupt technique employed in the 4.3BSD system. And furthermore, this increase in schedulability costs only about 10% in additional overhead. The effects can be seen in actual applications [8] as well. However, this earlier work did not shed any light on the relative importance of associating priorities with messages vs. using preemptable workers for the protocol processing. The more recent work [4] gives a better indication of the performance implications of these techniques for a variety of different types of computation including distributed continuous media streams, local continuous media computations, background traffic on the network, and low priority activities on the local host.

We evaluated the techniques by simulating a task set where tasks send audio streams across the network. The tasks are:

- 1 16-bit audio task at high priority (20 ms period),
- 1 8-bit audio task at medium priority (40 ms period),
- 1 high priority local computation (20 ms period) and
- $n$ low priority tasks (100 ms period).

The value $n$ is a background load parameter, a larger $n$ indicates more background load. Figure 1 shows the task set. The 16-bit audio stream is generated by $\tau_1$ and received by $\tau_2$. The 8-bit audio stream is sent from $\tau_3$ to $\tau_4$. There is a local computation, $\tau_5$, and the background messages go from $\tau_6$ to $\tau_7$. These two tasks are replicated $n$ times, and this replication determines the size of the background load. It is also important to note that the periodic background activity is bursty; measuring the performance of the techniques under strenuous conditions is our objective. The scheduling of the activities on the machine labeled PCPU4 is our main interest.

An important consideration in comparing these techniques is the ratio of protocol processing time to context switch overhead. Intuitively, a large ratio would indicate that preemption should be used to reduce the effects of priority inversion, and a small ratio would mean that preemption would be detrimental to efficient completion of the protocol processing. Our convention for describing the ratio is to use the one-way protocol processing time and the thread context switch time. In our simulation studies, we found that for a protocol processing time to context switch time ratio of 5-to-1 or 10-to-1, preempting protocol processing to avoid priority inversion makes sense. With a ratio of 2-to-1 or 1-to-1, however, the simulation showed that it is not worth preempting the protocol processing.

The table shows the protocol processing time and context switch time for two operating systems we are concerned with. The numbers are based on RPC protocol performance where we measure the round-trip response time. From this, we conclude that for systems such as ARTS [7] where the ratio is about 20-to-1 (using a one-way protocol processing time of about 4500 $\mu$s), the multi-threaded model is appropriate. Systems such as the $x$-kernel [1], with a ratio of about 50-to-1 could also benefit from this approach. For protocol processing engines implemented in hardware [5], however, the ratio may be more like 1-to-1, and so non-preemptable processing is appropriate in this case.

| Operating System | Time | |
|---|---|---|
| ARTS | | |
| context switch time | 260 | $\mu$s |
| protocol processing time (RPC) | 9000 | $\mu$s |
| *x*-kernel | | |
| context switch time | 38 | $\mu$s |
| protocol processing time (RPC) | 4000 | $\mu$s |

Table 1: Operating System Timing Measurements

# 3 Measuring the "Degree of Preemptability"

In evaluating the performance of the techniques in our simulation studies, we use several traditional metrics, and we introduced a new measure as well. We measure the mean and variance for the response times of the simulated activities, and we count the missed deadlines for the time-constrained, periodic continuous media activities.

We have introduced [4] a new metric which we call the *priority inversion factor* to measure the degree of preemptability in a system. Until this work, there was no way to describe the level of preemptability in a system except by specifying bounds on interrupt response time which is a very limited way to measure this effect. Interrupt response time, for example, does not give any indication of priority inversion in lower priority activities where hardware interrupts are not involved.

This priority inversion factor is a utilization-based method of measuring the preemptability. Utilization is the fraction of time during some interval that the processor was actually in use. During the time that the processor is not idle, it will be running some activity which is hopefully the highest priority activity available in the system. It is possible, however, that the running activity is not the highest priority (ready) activity, i.e. the system is suffering from priority inversion. To find the priority inversion factor, we consider only the time during which the processor is not idle, and we define the priority inversion factor to be the fraction of this time that the *wrong* activity is running. That is, the priority inversion factor is the time during which some priority inversion occurs divided by the time during which the processor is active. The product of the priority inversion factor and the total utilization is called the *priority inverted utilization.*

In order to illustrate this measure, we consider two systems which we simulated: one uses a single-threaded protocol processing engine with FIFO queueing and the other uses a multi-threaded engine with priority queueing. Furthermore, the multi-threaded engine has one thread for each message priority level in the system, and context switch time is taken to be zero, just to see what happens when preempting is free. We put the periodic task set described above on each system and measured the utilization and priority inverted utilization. Figure 2 shows the measured utilizations from the single-threaded case (T1F). The total utilization is given by the solid line; the priority inverted utilization is given by the broken line. In this case the priority inverted utilization is quite large compared with total utilization.

In Figure 3, we show the utilization for the W4P case. The total utilization is again given by

the solid line, but the priority inverted utilization is zero. This is because each message priority has its own thread and the context switch time (a non-preemptable critical region in the system) is zero.

The idea of quantifying the predictability of a task set comes from real-time scheduling theory. The *schedulable bound* [3] provides a numerical measure of the schedulability of a task set. If a task set contains independent, periodic tasks with fixed computation times and if the utilization of a task set is less that the schedulable bound, then no deadlines will be missed; if the utilization is greater than the schedulable bound, no guarantees can be made. Furthermore, if these tasks are allowed to contain shared critical regions, the schedulable bound is effectively reduced, admitting fewer task sets [6]. And the more priority inversion is associated with this resource sharing, the more the schedulable bound is reduced. Thus, priority inversion adversely affects the schedulable bound, and quantifying the priority inversion for comparison among competing techniques is therefore a reasonable way to evaluate the predictability of the system.

In our simulation study, the priority inversion factor observed for the T1PI, T2P, and T4P techniques ranged from .20 for the protocol processing time to context switch time ratio of 20-1 to .06 for a ratio of 1-1. For the T1F and T1P techniques, the priority inversion factor increases linearly (with the size of the background spike) for a ratio of 20-1, but the factor is constant at about .06 for the case where the ratio is 1-1.

# 4 Conclusion

The simulation results we have obtained [4] indicate that the method for structuring the protocol processing software should be chosen based on the ratio of protocol processing time to context switch time. This choice affects the predictability of continuous media streams on the network as well as the performance of local computations on each host in the system. And the timing characteristics of current operating systems indicate that these systems can benefit from the use of multi-threaded protocol engines. The priority inversion factor provides a way to measure the schedulability of a particular technique and allows us to evaluate the relative performance of the techniques.

# References

[1] N. C. Hutchinson and L. L. Peterson. The *x*-Kernel: An Architecture for Implementing Network Protocols. *IEEE Transactions on Software Engineering*, January 1991.

[2] Y. Ishikawa, H. Tokuda, and C. W. Mercer. Priority Inversion in Network Protocol Module. *Proceedings of 1989 National Conference of the Japan Society for Software Science and Technology*, October 1989.

[3] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment. *JACM*, 20(1):46–61, 1973.

[4] C. W. Mercer and H. Tokuda. An Evaluation of Priority Consistency in Protocol Architectures. In *Proceedings of the IEEE Conference on Local Area Networks*, October 1991.

[5] Protocol Engines, Inc., Santa Barbara, CA. *XTP Protocol Definition, Revision 3.5*, September 1990. PEI 90-120.

[6] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers*, 39(9), September 1990.

[7] H. Tokuda and C. W. Mercer. ARTS: A Distributed Real-Time Kernel. *ACM Operating Systems Review*, 23(3), July 1989.

[8] H. Tokuda, C. W. Mercer, and S. E. Breach. The Impact of Priority Inversion on Continuous Media Applications. In *Proceedings of the International Workshop on Network Operating System Support for Digital Audio and Video*, November 1990. Available in International Computer Science Institute Techical Report TR-90-062.

[9] H. Tokuda, C. W. Mercer, Y. Ishikawa, and T. E. Marchok. Priority Inversions in Real-Time Communication. In *Proceedings of 10th IEEE Real-Time Systems Symposium*, December 1989.

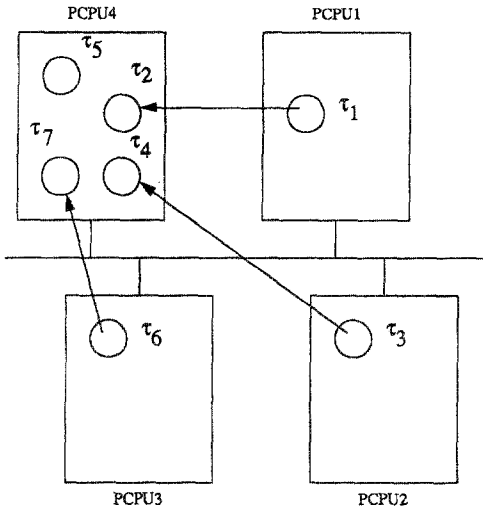


Figure 1: Simulated Task Set

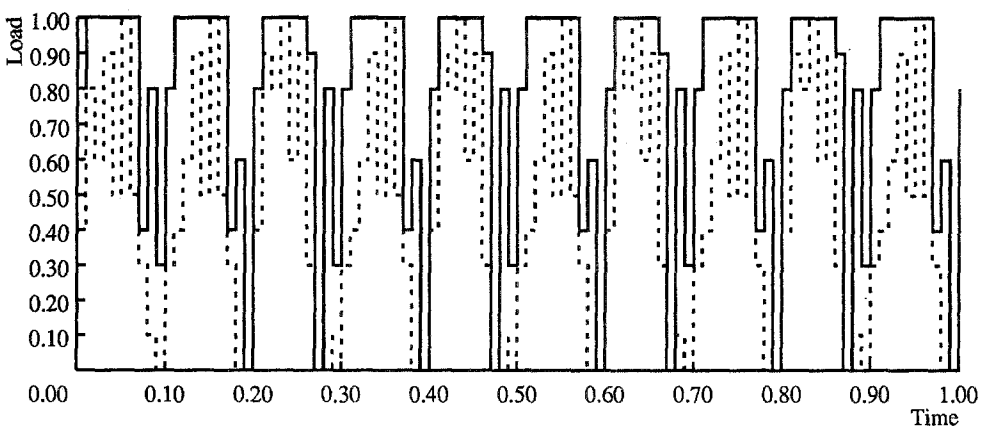


Figure 2: Utilization (T1F, 2000-0)

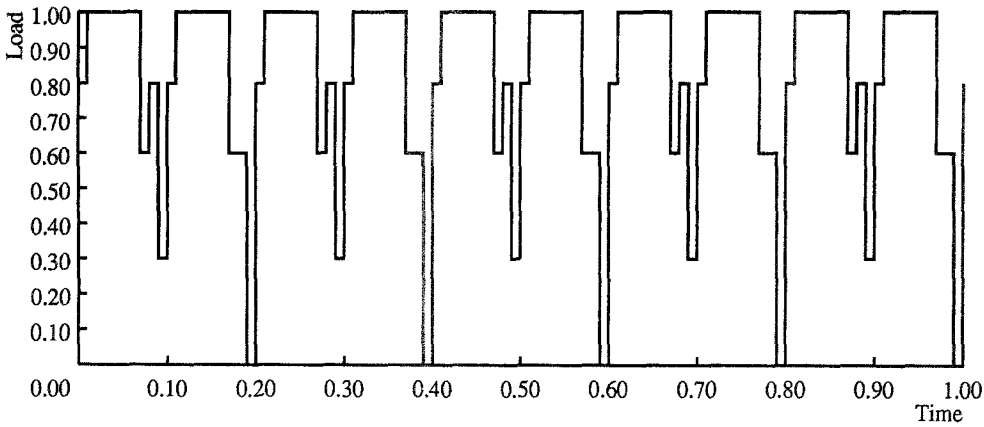


Figure 3: Utilization (T4P, 2000-0)