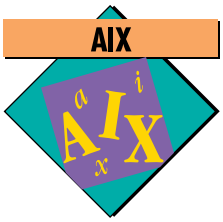
Process Priorities and the AIX Scheduler



By Allan Pellnat and Donald Totten

This article provides application developers and system designers insights into using process prioritization to achieve user satisfaction when both response time-oriented and throughput-oriented applications must run together.

The AIX process priority schema and scheduler are effective in delivering a fair share of the CPU resource to each user. Mixing response time-oriented transaction applications with throughput-oriented batch operations on the same system is the classic case for process prioritization. Fine-tuning key application process priorities makes the AIX Scheduler's performance even better and keeps end users happy. To do this, developers need detailed knowledge of how processes fit together to provide an end-user application.

The Environment

Directory One™ is the automated Directory Assistance (DA) service of Nortel™ Directory & Operator Services division of Northern Telecom®. It has established feature and performance benchmarks for the telephone industry worldwide. Telephone companies require their automated DA systems to ensure rapid response time for operator searches of very large databases under heavy system loads every day around the clock.

Directory One currently runs on IBM RISC System/6000 (RS/6000™) processors using AIX Version 3.2.5. Plans are underway to migrate to AIX Version 4.1 and Symmetric Multiprocessing (SMP) processors. The Oracle® database is used for the listings.

The Problem

Databases of telephone numbers contain millions of listings. Because as much as 1% of these databases are updated daily, the processes for updating and backing up the databases must ensure transaction throughput without degrading DA operator search response time.

The challenge is to deliver rapid search response time during the CPU intensive update and backup periods without significantly reducing the update and backup throughput. The quick and easy answer seemed to be boosting the process priority of key search application processes above the update and backup application priorities. But that proved to be wrong.

The Approach

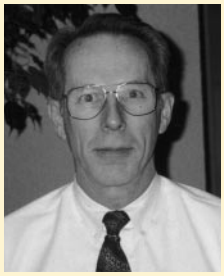
The test consisted of RS/6000 processors handling simulated DA calls and searches, while a batch update process added and deleted listings to the database. Search response time was measured by trapping and time stamping search requests and responses with a protocol analyzer on the Fiber-optic Data Distribution Interface (FDDI) ring. Update throughput was calculated from log file start and end times, and the known number of add and delete transactions.

Figure 1 shows the process prioritization test bed.

Response Time Measurements

We needed a data presentation method that would clearly show us the effect of changing process priorities. We chose a graph of the cumulative distribution function for the probability of search response time. Plotting the results of each test series against a known baseline makes the relative differences readily apparent.

Baseline measurements of search response time were taken for a "no load" case consisting of 600 different searches allowing the AIX Scheduler to "nice" the process priority values up from the starting value of 60. (*Note:* A process whose priority value has been raised above the 60 starting point has lower precedence in gaining access to the CPU.)



Allan Pellnat

Donald Totten

Trace A in Figure 2 shows these results. The graph shows the probability of search response time being less than a given value. Under no load conditions—those with minimal queuing—the average response time will occur approximately at P = .67. The average response time for this baseline test was 40 ms.

Figure 2 shows search response time probabilities for three cases with AIX doing dynamic prioritization of processes. Trace A is the no load baseline condition where search response averages 40 ms. Trace B shows that a search-and-call load that utilizes 60% of the CPU results in an average search response of 289 ms under the AIX dynamic prioritization regime.

The critical case is Trace C, the 60% call-and-search load concurrent with a batch database update application. Average search response time increased to greater than one full second, which was totally unacceptable.

Roles of the Processes

The batch update process added 1,800 listings, then deleted the same 1,800 listings. Under no load conditions, the batch update completed the 3,600 transactions in 38 minutes. Concurrent with the 60% CPU load of search-and-call control, batch update did the 3,600 transactions in 61 minutes. This is an acceptable throughput rate of one transaction per second.

The solution to the problem lies in understanding the role that each of the six separate processes plays in delivering the whole call control and search application.

In some cases, a single instance of a process handles all transactions for all calls or searches. In other cases there may be multiple instances of a process, in which each instance handles one or more calls or searches concurrently. Although a typical call lasts for less than 30 seconds in normal operations, hundreds of calls can be active in various states at any point in time. On average, each call generates two searches. Two servers sequentially handle searches of a particular database. A single, dedicated process per communication link handles all communication between the outside world and the call control processes.

Earlier work had established the per-call and per-search average service times for each of the six unique processes that together form the primary application of the system. The distribution of work per call among the six processes is characterized in Figure 3.

In a typical business day, peak call-and-search loads that occur in mid-morning and mid-afternoon are approximately equal. Total CPU utilization of the RS/6000 processors may exceed 70% for an hour or more at these peak times. The six critical search-and-call control processes account for the lion's share of this CPU utilization.

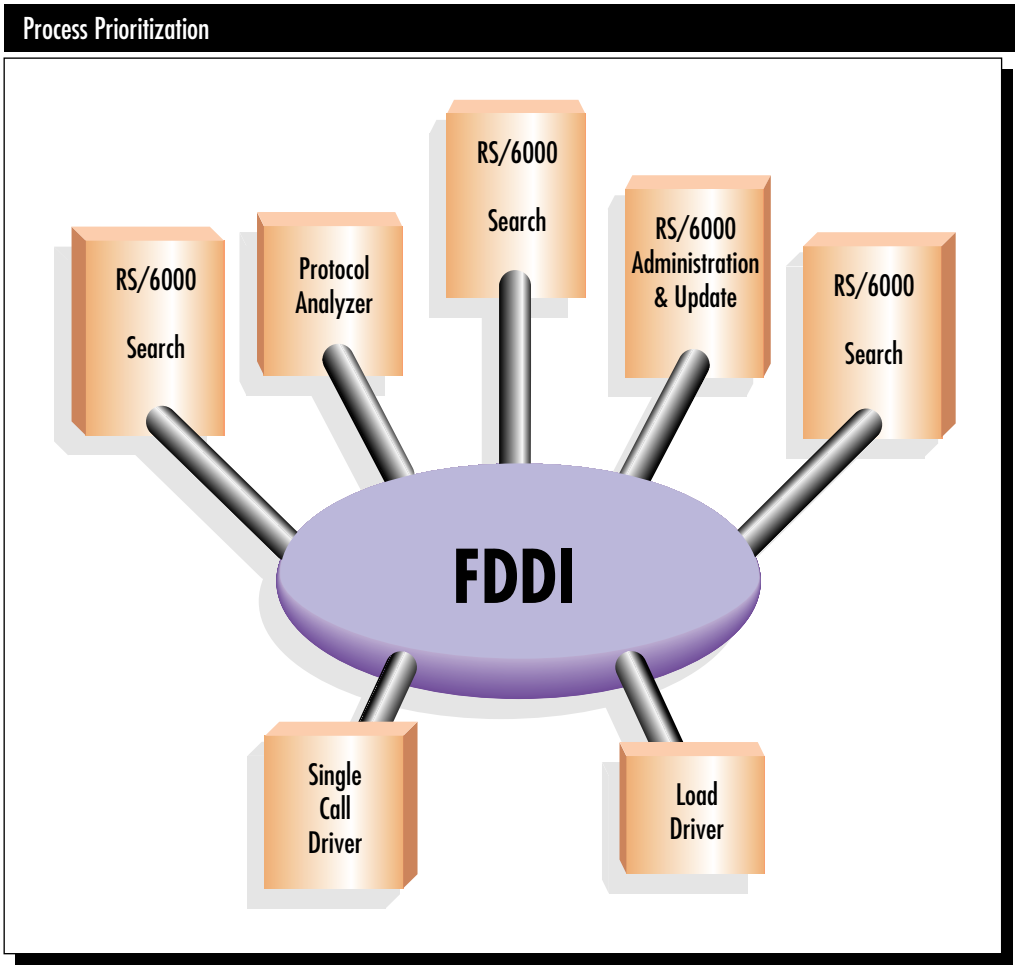


Figure 1. Process prioritization test bed

Moving all six processes to a higher fixed priority creates a problem: Process B, a single process that consumes the largest amount of CPU time on a per call basis, starves out the other five processes. Every time a CPU context switch takes place, there is a high probability that Process B will be next in the run queue because it is used so many times during a call.

It was evident that simply prioritizing the six key processes over the other workloads on the system would probably cause more harm than good. The solution to the problem was to understand how the six separate processes work together to deliver the application.

Process A—the search—accounts for 48% of the per call CPU demand. Because any given search is served by one of several instances of Process A, the average utilization of any of those processes is only a fraction of the total CPU time for a call. Similarly, Process C can have ten or more instances, which results in a small amount of actual usage on a per call basis for any single instance.

However, Process B is only a single instance called to perform some processing seven or more times during every call. The AIX Scheduler dynamically prioritizes these processes based on short-term CPU utilization of each process. Process B, the "hog" of the family, gets "niced" out of the way by the AIX Scheduler. When its priority is fixed, it tends to starve out the other processes, especially the key element, Process A.

The Solution

The solution is to set each process to a fixed priority based on its contribution to the key measured element, the response of Process A—database search. Figure 4 shows the processes again with their percentages and fixed priority settings.

The search process itself gets top priority. Search requests and responses pass through Processes B and C, so they are next in order of preference. The remaining processes are not directly in the search path. Getting them into and out of the CPU quickly assures that they are not holding up the critical search processes. They are prioritized equally at a level above other jobs running in the system, but below the critical search-related processes.

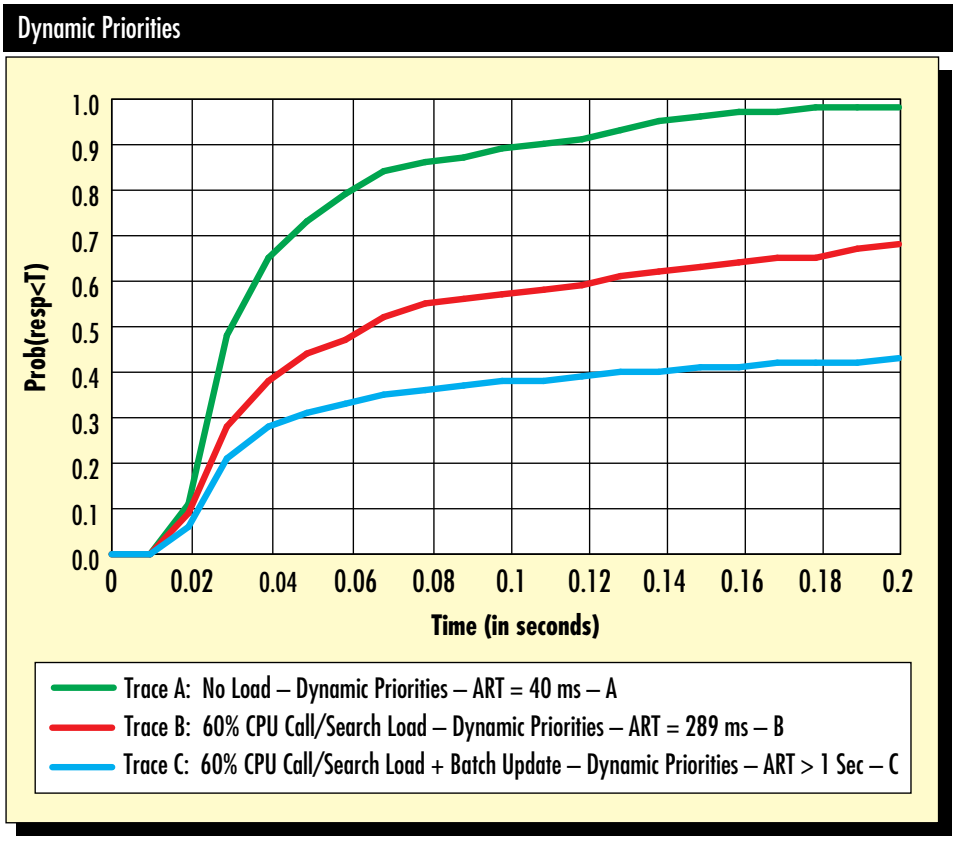


Figure 2. Search response time: no load and 60% CPU load + batch update = dynamic priorities

Process	Per Call Work Proportion	Work Distribution
A	48%	Runs twice during a call
B	26%	Runs seven or more times during a call
C	16%	Runs four or more times during a call
D	4%	Runs four or more times during a call
E	4%	Runs once at the very end of a call
F	2%	Runs one to three times during a call

Figure 3. Work distribution per call

Process	Per Call Work Proportion	Work Distribution
A	48%	Fixed Priority 45
B	26%	Fixed Priority 46
C	16%	Fixed Priority 47
D	4%	Fixed Priority 48
E	4%	Fixed Priority 48
F	2%	Fixed Priority 48

Figure 4. Fixed priority settings per call

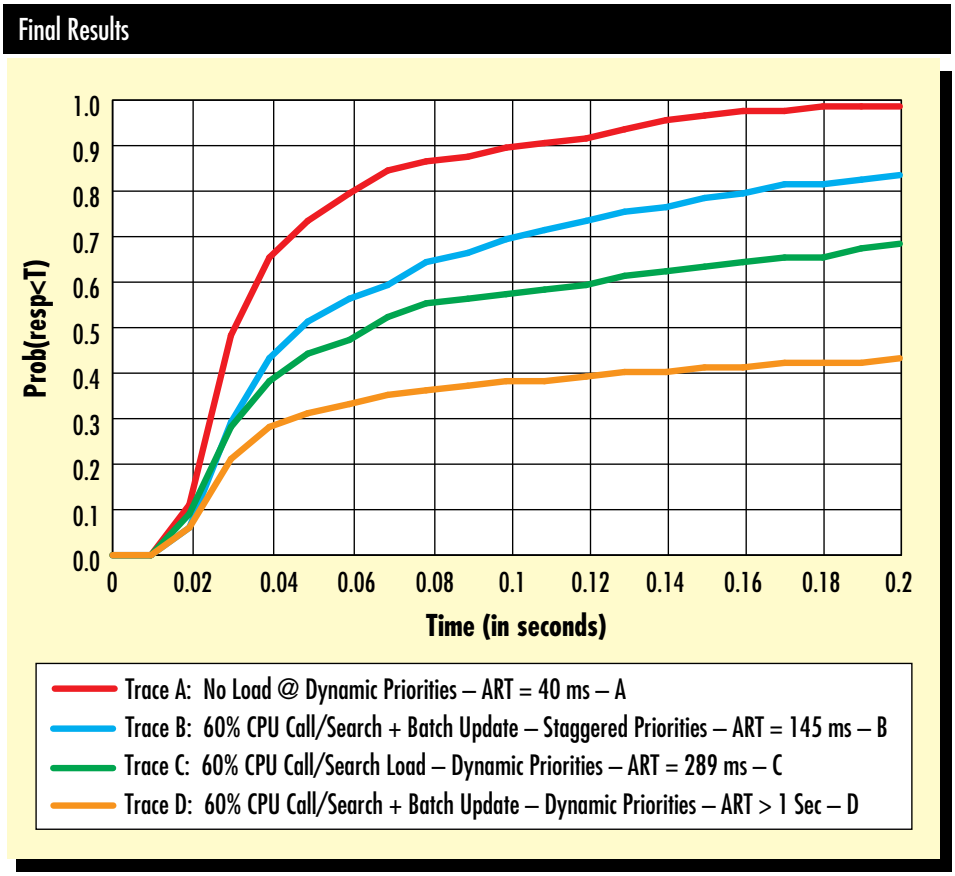


Figure 5. The 60% CPU load and batch update—staggered fixed priorities

Figure 5 illustrates the dramatic effect of this scheme on search response time probability. Together, the full 60% call-and-search load and the batch update process now yield an average search response time of 145 ms—twice as good as the call-and-search load alone under dynamic priorities.

Although this slows down the batch update throughput, it is still within the acceptable range. Under dynamic priorities, the 3,600 batch transactions required nearly an hour to complete. Under the staggered fixed priorities, the same operations took 70 minutes—a 16.7% change. This represents a worst-case scenario, however, for the concurrent batch update. Under normal operating conditions, batch database updates are run in the early hours of the morning when the call-and-search volume consumes considerably less than 60% of the CPU.

Conclusion

Identifying a key application to receive priority access to the shared CPU resource can backfire unless you understand the inner workings of that application. Applying process prioritization to a block of application processes without analyzing the interaction of the processes may lead to poorer performance rather than better. AIX does a very good job of scheduling fair-share CPU usage among diverse processes and applications. When application developers and system designers cooperate in using AIX performance tuning tools, end users reap the benefits.

Allan Pellnat, Northern Telecom, Directory and Operator Services, 97 Humboldt Street, Rochester, NY 14609. Internet: agp@sun111.cci.com. Mr. Pellnat is an advisory engineer specializing in defining and evaluating overall system performance for the Directory One product.

Donald Totten, Northern Telecom, Directory and Operator Services, 97 Humboldt Street, Rochester, NY 14609. Internet: djt@nt.com. Mr. Totten is an advisory engineer responsible for specification, validation, and verification of product performance. He has a BS in Mathematics and a BS in Computer Science from SUNY Brockport.