Sample Final Exam Question

The following question is in the style of the questions that will appear on the final exam. It depends on material discussed in class and the readings, but requires application of the concepts discussed, rather than merely requiring you to remember and understand the materials covered. A reasonable answer to the question is on the second page of this document.

Question:

Waiting time in the ready queue is one metric for CPU scheduler performance. This question deals with finding metrics for the memory scheduler performance. In a landmark paper, Peter Denning asserted that the cost of any memory management strategy can be measured by the sum (over all processes) of the time-integral, for each process, of the number of page frames assigned to that process (or more simply the total number of page-seconds consumed by all of the processes in the system). If, for instance, a process has an image size of N pages and we leave it in memory for T seconds, the time-space cost of this decision is $N*T$ page-seconds.

(a) Justify this time-space product as a reasonable measure for the cost or performance of a memory management strategy.

(b) If we were to use pure demand-paging (without any pre-loading) rather than swapping, what (approximately) would be the time-space cost of allowing the same process to run for the same T seconds (if we assume that the we started with no pages and that the process faulted for its N pages uniformly over the T second period)?

(c) Is it likely that the demand paged process would require the same number of pages n as the swapped process N? Why or why not?

(d) The amount of time that a process is in memory is not merely the time it takes the process to run (T) but also includes the time it takes to swap the process in and out of memory. If we can swap N pages in or out in $T_s(N)$ seconds, the time-space product for swapping becomes $N*(T+2T_s(N))$. Suppose that we can fault-in or replace-out N pages in $T_p(N)$ seconds. Which is likely to take longer: swapping in N pages, or page-faulting for N pages? Why?

(e) What mathematical relationship must hold (between T, T_s, T_p, N and n) in order for demand paging to out-perform swapping? What does this mean about the programs, operating system, or hardware?

Answer:

Beyond requiring you to understand what swapping and demand paging are, this question does not actually require you to remember much else about them. Rather, this is a test of your ability to do some simple mathematical reasoning about them

(a) If each a process uses fewer pages of memory, more processes can be fit in memory at one time, meaning that the scheduler is more likely to have ready tasks to run, and the CPU will be kept busy, and system throughput will be maximized. Thus, if the memory manager can run more processes in fewer pages, it is doing a good job.

*(b) At the beginning there would be no pages in memory, and by the end there would be N pages. If the fault rate is uniform, the area under that curve is T*N/2.*

(c) It is most likely that (n < N). Most programs exhibit reasonable code and data locality, so that during a small period of time they reference only a fraction of their pages (perhaps because they are executing one loop, calling only a few subroutines, and manipulating only a small amount of data).

(d) Tp(N) is likely to be greater than Ts(N) for a few reasons:

A process is usually swapped out to one contiguous region on disk, which means that the I/O involves little or no head motion. A process that is demand paging might have pages all over the disk.

There is start-up and completion over-head associated with each I/O operation. In swapping, there is likely to be only one large request, whereas in demand paging, there is likely to be one request per page, each with its own startup and completion overheads.

Beyond the I/O time, there is also an overhead associated with taking each page-fault.

(e) The cost of swapping is (from above) N(T+2Ts(N)). The corresponding cost of demand paging is*

$$n*(T+2Tp(n))$$

Demand paging will win if

$$n*(T+2Tp(n)) < N*(T+2Ts(N))$$

or (perhaps more clearly) if

$$n/N < (T+2Ts(N))/(T+2Tp(n))$$

This means that either n is much smaller than N (the programs exhibit good locality) or that Tp is not much greater than Ts (the performance penalty for page faulting and scattered I/O is low).