Operating System Questions

Matthew Johnson, Trinity Hall

May 18, 2002

1 2001 Paper 1 Question 11

1.1

With static priority scheduling, an integer priority is assigned to each process currently awaiting the CPU time. They are arranging in decreasing order of priority and the scheduler will step down priority levels in order, executing the highest priority non-blocked process. If there is a tie, then the scheduler may assign a higher priority to the job that was started first, or away from the one which had the CPU most recently.

1.2

There are several types of non-priority based scheduling algorithms, which use factors such as the time the jobs were started, the estimated time the process will take to complete and so on. FCFS scheduling merely executes jobs in the order that they were presented to the computer, as embodied in the name First-Come First-Served. There are 2 algorithms that look at the job length, Shortest Job First and Shortest Remaining Time First. The first one of these only looks at the initial time the job will take, and suspends long jobs in favor of short ones, and the second updates this value as it goes along, so a long job that is almost finished won't be pre-empted by a shorter one. Finally there is the Round Robin-type scheduling which merely visits each job in turn for a small time slice.

1.3

The major problem with static priority scheduling, is that low priority jobs have no guarantee of every being scheduled, particularly when they are largely IO bound, and hence need little CPU time, when they need it. This can be fixed by having dynamic priorities. This is accomplished by calculating the priority based on the initial, assigned, priority, but modifying by how much time it has already had. This means that a job will always filter up to the top eventually, because the higher priority jobs will have their priority decreased as they keep using CPU time.

1.4

Many scheduling algorithms use the above method of dynamic priorities and try to favor principally I/O bound jobs. This is because an I/O bound job will need little processor time, at infrequent intervals, and so can be relied upon to voluntarily release the processor quite quickly. CPU bound jobs will tend to hog the processor, since their I/O delays are far between.

1.5

<insert 'nix/NT case study here>

2 1999 Paper 1 Question 4

The scheduling algorithm can be inferred to be pre-emptive. Round robin scheduling can be appropriate for multi user systems, because you can guarantee that every job will be scheduled.. and a user won't be left hanging

Process Response / Completion Table

Process	Create	Required	1st CPU	Complete
P_1	0	9	0	15
P_2	1	4	3	9
P_3	7	2	5	7

Using small quantum is good because processes will get a share of CPU time quite quickly, and if they don't need much will tend to complete faster, however, there is more scheduler overhead time wasted.

3 1999 Paper 1 Question 11

Processes are typically provided with its own private address space which is indirectly mapped to physical memory. This means that the operating system can assign it anywhere in memory, and calls within the program will still work, since it does not need to know the absolute address it is loaded at. This also allows the operating system to more easily relocate running processes, unlike a system which sets the correct addresses into the code when it is loaded. Finally, it can be used to facilitate memory protection, and restrict the process from accessing outside of its designated area.

This system solves a lot of problems, and makes the operating system's handling of processes much more flexible, however it does introduce another level of indirection and will therefore slow down all accesses into memory.

3.1

<insert pretty picture>

3.2

Physical memory allocation is easier with paging than with segmentation, since pages are fixed sizes that often correspond to frame sizes, and make it easier to have a 1-1 mapping. Segments, on the other hand, vary in size, and need more complex indirection.

3.3

Segmented addressing has the advantage of being more closely related to the image of memory that the programmer sees when programming, and allows more flexibility of memory allocation within programs.

4

4.1

Page sizes are always given in powers of 2, because internal addresses will be specified by an n-bit binary number, and the range of that is 2^n - so the most efficient way to store the addresses uses the full range.

4.2

process	First Fit	Best Fit	Worst Fit
212	in 500	in 300	in 600
417	in 600	in 500	in 500
112	in 500*	in 200	in 600*
426	no fit	in 600	no fit

* after the 212 process

4.3

It's easier to share reentrant modules using segmentation than pure paging because segmentation provides a view of memory according to the programmer's perspective. Reentrant module can be loaded as a logical unit. Any information can be shared if it is defined to be a segment. Several segment can be shared. Another difference is that when sharing page, the page number of the shared frame should be the same in the different process, but in segmentation sharing, the segment number need not to be the same.

5

5.1

In external (to the processor) hardware devices, you can often shift functionality into hardware on the chip. This is good for several reasons, because the code can effectively have full permissions to access the device in question, but not be running in effective kernel mode for other restricted areas. It will also tend to speed the system up, since the processing is not done by the operating system. The disadvantage is that changing the code is much more difficult, and the hardware tends to be more expensive.

5.2

Blocking IO is best used in batch-type programming, where you are reading fixed blocks of data from a source where there will be little or no delay, or where you know you need all of the data. More interactive systems might use non-blocking IO, and only act on the data when there is some available. Local sources such as files might be read with blocking IO, whereas remote connections, where long delays could be expected, and input from inconsistent sources such as the console would typically use non-blocking methods

Blocking I/O is useful, because it is easier to use and support.