Operating systems – supervision 2

James Wood

March 3, 2015

1. (a) i. Cache memory stores a selection of data the OS expects to need soon. This is much smaller than main memory, and quicker to access.
   ii. Main memory, ideally, should store all of the intermediate state of the processes currently running. To avoid using other storage methods (such as the hard disk), this should be large.
   iii. Registers store information being used immediately by the CPU. They store a very small amount of information, and have very quick access time.

   (b) When main memory has no available space, the OS will swap some memory to disk. Thus, there will be virtual memory for which the physical address is not in main memory.

2. If an application blocks to read keyboard input, it will be suspended, and resumed only when a keypress occurs. It will not be able to do any processing or take any other input until a key has been pressed.

   If the application does a non-blocking read, the operating system will read the state of the keyboard and send this information back to the application.

   If the application does an asynchronous read, it will continue running. When a keyboard event occurs, the operating system will send the application a signal, and it will respond to that signal. In many languages, this is how events are implemented.

3. A non-premptive scheduler is usually simpler than a premptive scheduler, since a premptive scheduler has to do premption, which requires each process to have extra information stored about it and extra calculations done.

A premptive scheduler is capable of choosing which process to run based on how long it is likely to take, so can schedule in such a way that each process gets a similar amount of processor time. This makes it fairer than a non-premptive scheduler could be.

Premptive scheduling introduces the overhead of premption, and thus is less performant than non-premptive scheduling. However, a premptive scheduler will make the system seem more responsive.

To gather any data to use for premption, the hardware must include a timer for CPU bursts.

4. (a) When choosing a process to run, a static priority scheduler will choose a process of high priority before any process of lower priority. In the event of a tie, the scheduler has to resort to some priorityless scheduling method.

   (b) FCFS can be considered as a priority scheduling algorithm in which the priority of a process is always lower than any that came before it. Similarly, SJF has priority based on the prempted length of process, and SRTF has priority based on prempted time remaining (which changes as the process is run). Finally, in RR, the priority is based on how long it has been since the process last ran.

   (c) Static priority scheduling can cause jobs of low priority to never be run. This problem can be overcome by augmenting the priority with some dynamic measure favouring jobs that have been ready for a long time.

   (d) I/O-intensive jobs are favoured because they take a long time to complete, but use very little CPU time while completing. Thus, it is good to have them scheduled to be first, since they don't get in the way of more CPU-intensive jobs.

5. (a) ?

   (b) All jobs are guaranteed to be allowed to run within a reasonable time, so many users can be having jobs run at the same time.

(c)

start	end	process running	required time remaining
0	3	P1	6
3	6	P2	1
6	9	P1	3
9	10	P2	0
10	12	P3	0
12	15	P1	0

process	time from creation to finish
P1	15
P2	9
P3	5

(d) Having a small quantum reduces the worst case time a job will have to wait to be run, but causes a need for more swaps, which take time. This causes an increase in average waiting time.

6. (a) The sceduler maintains two priority queues: the ready queue and the wait queue. The ready queue stores processes that currently have processing to do, and the wait queue stores processes blocked due to IO requests. Consider this example, where the front of a queue is shown at the top:

ready queue	wait queue
P83	P12
P28	P42
P76	

If a new process, P94, is created, the scheduler works out its priority and places it in the queue accordingly. The queues may look like this:

ready queue	wait queue
P83	P12
P94	P42
P28	
P76	

Eventually, P83 finishes, gets put back in the ready queue, or makes an IO call that puts it in the wait queue. Then, P94 gets run. If it ever makes a blocking IO request, it will be put in the wait queue, and a process from the ready queue (like P28) will be run. Once the request has completed, it will be put back into the ready queue. P94 may get put back into the ready queue while being processed. Once it has finished, it will be released by the CPU.

(b) Each process has a PCB, which stores the state of the process (including program counter) as well as information needed by the

scheduler, such as an ID and history about the process's performance.

(c) SJF and SRTF need data upon which to base a prediction of burst length for each process. The simplest prediction schemes need only the length of the last CPU burst for that process, but if more history is stored, a better prediction can be made. The burst length can be obtained using a timer.

(d) A non-premptive scheduler has the advantage of having less overhead associated with recording of burst lengths and calculating predictions, and the disadvantage that it can't distinguish between long and short jobs, which tends to make it less able to be fair.

(e) In the programmed IO case, the CPU will have to arrange the reading from and writing to memory, which it won't do for DMA devices. After this, the scheduler checks the wait queue for processes waiting for the given interrupt, and moves any of them into the ready queue. Given that they have been using blocking IO, the scheduler will probably give them high priority.

(f) If there were many interrupts being raised, the bus handling signals may become full, causing signals to have to wait. Also, the CPU will have to modify the wait queue many times, possibly causing processes that were on the wait queue to be run soon. This creates a need for more swapping.

7. (a) When a program is written, it is not generally possible to know where in physical memory its storage should be.

(b) To solve this problem at compile time means that addresses are substituted in to the program based on the state of memory when the program is being compiled. This typically means that the program has some reserved area of memory to use that will not be used by any other processes. This is the simplest solution, but generally not viable.

To solve this problem at link time means that addresses are substituted when an instance of the program is loaded as a process. This is a safer solution, because physical addresses chosen can be based on what other processes are using. However, it adds a start-up cost to each process.

To solve this problem at run time means that the source program contains virtual addresses, which are translated into physical ad-

dresses when memory needs to be accessed (usually by dedicated hardware). This allows physical addresses to change whilst the process is running, which includes allowing virtual memory to be larger than main memory (by use of the hard disk or similar). However, without dedicated hardware, it can make lookups slow.

8. (a) Premptive schedulers require something to time burst length, so do require hardware support.

   (b) It is possible that there is some state of the process being switched in not stored in the TLB, particularly if that process has been idle for a long time. In this case, a flip-flop in the TLB will not suffice, since we also need to deal with main memory.

   (c) If a device doesn't allow its state to be read without blocking, non-blocking IO with it is not possible.

   (d) It depends on the required properties and metric of performance of the scheduling algorithm. SRTF will usually be a better choice than SJF if it is possible.

   (e) RR does not suffer from the convoy effect because it gives every ready proccess the same share of CPU time. A long process does not take up particularly much of this time in each cycle.

   (f) Paging virtual memory facilitates having physical memory other than the main memory, so paged memory will be larger than segmented memory.

   (g) DMA allows faster reads and writes to memory for devices.

   (h) System calls are still needed so that applications can access devices, write to files &c, but are usually hidden from application programmers by the OS's libraries.

   (i) Yes, I tried it.

   (j) The temporary files used by Unix commands are likely to be stored in the buffer cache, rather than on the hard disk.