Operating Systems 2000 p1q11 SMH

Main thing we could do would be to use language-level techniques. E.g. we could require that all applications were written in a "safe" language like java, and only load such applications. This would involve severely constraining the interface to the user (i.e. no "poke" command!), but might just work.

Paper 1 Question 11 (20 Marks)

Describe with the aid of a diagram the life-cycle of a process. You should describe each of the states that it can be in, and the reasons it moves between these states. [4 marks]

What information does the operating system keep in the process control block? [4 marks]

What information do the shortest job first (SJF) and shortest remaining time first (SRTF) algorithms require about each job or process? How can this information be obtained? [2 marks]

Give one advantage and one disadvantage of non-preemptive scheduling. [2 marks]

What steps does the operating system take when an interrupt occurs? Consider both the programmed I/O and DMA cases, and the interaction with the CPU scheduler. [4 marks]

What problems could occur if a system experienced a very high interrupt load? What if the device[s] in question were DMA-capable? [4 marks]

Answers to Paper 1 Question 11

Process Life-cycle

Looking for a five-state (or more) picture like that in notes. The states involved will be something like:

1. *New*: state for newly created processes. Moves from here to *Ready*.
2. *Ready*: state for any runnable process (i.e. one which could use CPU if granted it). Typically many processes in this state. Moves into *Running* when dispatched by scheduler.
3. *Running*: state for processes currently executing; only one of these per CPU. Moves to state *Ready* if preempted, *Exited* if exits, or *Blocked* if waits on I/O or some other event.
4. *Blocked*: state for all those processes currently awaiting an event of some sort. Typically many processes here. Moves into *Ready* once the requisite event occurs.

5. *Exited*: state for all those processes who have [willingly or unwillingly] called exit.

Could also have various swapped states in there.

Contents of PCB

1 mark each for any four of the following

Lots of things in the PCB including:

1. Process state.
2. Memory management information.
3. Program counter.
4. CPU registers.
5. CPU scheduling information.
6. Accounting information.
7. I/O Status information.

I expect that 1, 2, 3/4 and 5 will be the most common answers, but the others or anything else plausible will be ok.

SJF and SRTF

1 marks for each part

They require the length of time each job/task needs to execute (or, more generally, the length of its next CPU "burst").

This cannot usually be obtained easily or precisely, but rather must be estimated. One possibility is to use an exponential average e.g. $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$ where τ_n is the n^{th} estimate, t_n is the n^{th} sample and $0 \leq \alpha \leq 1$.

Non-preemptive Scheduling

One mark for an advantage, one for a disadvantage.

Main advantage is that it is simple. Also can be slightly more efficient since only ever schedule and context switch when one has to [e.g. when a process has just blocked].

Main disadvantage is lack of protection; cannot prevent denial of service. Similarly get larger variance in waiting times.

Action on Interrupt

1 mark for I/O handling in each of programmed I/O and DMA cases; 2 marks for interaction with scheduler.

When an interrupt occurs, the CPU transfers control to a/the interrupt handler. This is low-level software which may need to e.g. demultiplex the interrupt, save some state, etc.

Next an interrupt-specific device driver is invoked which will:

1. for programmed I/O device:
    - transfer data
    - clear interrupt (sometimes a side effect of transfer)
2. for DMA device:
    - acknowledge transfer
3. request another transfer if any more I/O requests pending on device
4. unblock any waiting processes.
5. enter scheduler or return

In summary, on an interrupt OS will (a) try to keep I/O device[s] busy and (b) potentially unblock (and dispatch) a process/processes.

Problems with Interrupt Load / DMA

2 marks for each the below, or viable substitutes.

Two main problems can occur. The most notable is that if the interrupt load is high enough, one doesn't get any time to schedule processes at all ("interrupt livelock"). Secondly, if a large amount of DMA traffic is going on, contention on the bus might cause degradation in processor performance.