Operating Systems (SMH) (Part IA)
Paper 1 Question 12 (20 Marks)

System calls

A system call is the means by which an application (user-space process) can invoke services within the operating system.

They are typically implemented by using *software interrupts* or *traps*: hardware supported instructions which transfer control to a well-defined entry point in the operating system, switching the processor mode to "privileged" in the process.

Scheduling

4 marks each for preemptive and non-preemptive scheduling; in each case, expecting one point for each of the four issues mentioned.

Preemptive scheduling is where a process can be interrupted or *preempted* in its execution at any time by the operating system (or, more precisely, an interrupt). It is relatively complex, but has scope to implement fairness by ensuring no process "hogs" the CPU. In terms of performance there is some additional overhead in *context switches* which aren't, strictly speaking, necessary. The hardware support required is a programmable or periodic timer which can generate interrupts.

Non-preemptive scheduling refers to schemes in which a process only relinquishes the processor voluntarily (e.g. via `yield`) or implicitly (e.g. when blocking on I/O). This is somewhat less complex to implement since there are fewer times when scheduling decisions need to be made, and less worry about concurrency in user-space. Fairness cannot in general be guaranteed since a rogue process can fail to yield the processor for an arbitrarily long time; however performance is usually good since there are no unnecessary context switches. No additional hardware support is required beyond the ability to save and restore process context.

XP versus Unix

2 marks for the notions of process; 3 marks each for the descriptions of scheduling in each case. I don't require a huge amount of detail about the scheduling algorithms.

Processes in XP are container objects with a related address space, security token, and other resources (e.g. object handles). They also contain one or more *threads*, the unit of scheduling.

In UNIX, the notions of process and thread are conflated: every process has implicitly precisely one thread of control.

Scheduling in XP is done on a thread level: there are 16 dynamic priorities (0-15) and 16 static or "real-time" priorities (16-31). Scheduling is by strict priority, with round robin being used within a given priority. Dynamic priorities are implemented by giving a "boost" to processes after unblocking – this decays over time until it returns to the base priority.

In Unix, processes have priorities between 0 and 127, with the lower numerical value being higher priority. Again round robin is used between processes with the same priority. Processes have their priorities *reduced* (i.e. a positive integer is added) based on how long they have run for in the recent past. Over time, if a process does not run, its priority will return to its base value.