SOLUTIONS

1(a) Briefly outline the historical motivation behind the development of the modern operating system. What is the major factor which influences the design of operating systems today?

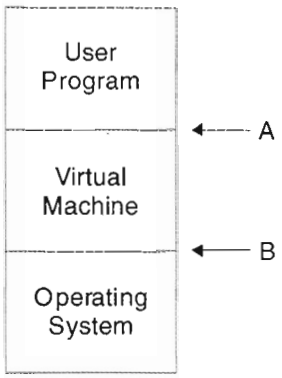
Historically, the major motivating factor behind the development of operating systems was the large capital cost of early computers. Not only was the capital cost of early computers high but also their running costs. In this situation it was financially important to maximise the utilisation of such a costly resource and early developments in operating systems were all geared to this end.

Issues such as keeping the CPU running with maximum utilisation were addressed by advances such as: monitors, batching, spooling and finally multi-tasking.

Apart from a few niche markets, operating systems today lay most emphasis on usability. Hardware is comparatively cheap, hence utilisation is no longer a concern. The ability of people without high levels of training to use computers is now the paramount issue.

(6)

1(b) What is a *virtual machine*.



The user's program runs and makes systems calls (to the VM layer) across the interface 'A' The virtual machine – which is a software component implements systems calls at 'A' using the systems call available to it across interface 'B'.

(4)

Explain the mechanism that allows a virtual machine to run, for example, a Windows program on a computer with a Unix operating system.

Since the virtual machine 'translates' systems calls, there is no requirement that the systems calls across 'A' and 'B' be the same. Consequently, a Windows program, say, can issue Win 32 API calls at 'A' which are implemented on the underlying operating system by the virtual machine layer.

(2)

1(c) What is Posix? Explain its significance. Briefly describe the four main components of the Posix standard.

Posix is an IEEE standard for the applications programming interface (API) which is vendor-independent. (Posix is not an operating system, just an API specification, although it is largely synonymous with Unix. The interface is written in the C language.)

The significance of Posix is that it ensures portability of applications from one make of computer to another. If a program is written to be Posix-compliant, it will run identically on any machine which conforms with the Posix standard.

(4)

The main four components of Posix are:

i) Base Definitions - Just what the title says!

ii) System Interfaces- Describes the system call prototypes, their possible argument values, actions and return values. Various sub-sections cover things like basic functions, inter-process communication, real-time extensions, threads etc.

iii) Shells & Utilities - Covers the command interpreter and various utility functions.

iv) Rationale - Sets-out the design decisions underpinning Posix.

(4)

2(a) What is a *privileged instruction* and why is it necessary? Give three examples of privileged instructions.

A privileged instruction is one which can only be executed by the kernel of the operating system. If a user program attempts to execute a privileged instruction, an error condition results.

Privileged instructions are those which fundamentally affect the operation of the computer or can result in conflicts between users. Such 'sensitive' instructions must not be made available to users since their unrestricted use could cause major operational problems.

Examples of privileged instructions include: Disabling/enabling interrupts, I/O operations, stop (halt) instruction, etc.

(7)

Explain how a user's program can access privileged instructions.

Privileged instructions can only be executed indirectly by the user via systems calls. In this way, the operating system can block and/or manage any undesirable side-effects.

(2)

2(b) Describe the hardware mechanism by which a CPU supports the forcing of a context switch in pre-emptive multi-tasking operating systems. What important condition is necessary to prevent users from allowing their processes to run without any time limit?

To support, pre-emptive, multi-tasking, the CPU incorporates a countdown 'watchdog' timer. At the very end of a context switch, the OS scheduler loads this timer with some value which specifies the process's timeslice. The timer counts down on an independent clock. If the process uses all of its timeslice, the timer will countdown to zero which will trigger an interrupt which will in turn, invoke the scheduler to perform a context switch.

Loading the countdown timer must be a privileged instruction otherwise a user could repeatedly reload the timer and extend their timeslice without limit.

(5)

2(c) Apart from what happens when a normal process terminates, explain what extra happens when:

i) A child process terminates

When a child process terminates a return code will be made available to the parent process. In addition, the child will be removed from the parent's list of child processes.

(3)

ii) The parent of a child process terminates before the child terminates.

If the parent terminates before the child, the child (and any sub-children) will also be terminated. The parent process will maintain a list of child processes and this will be cleared before the parent finally terminates.

(3)

3(a) In UNIX, what are signals? For what purpose are they used?

Signals are a low-level inter-process communication mechanism, originally intended for the kernel to communicate to user processes, although signals have subsequently been extended to allow communication between user processes.

Since signals are typically single integer values, they are used typically for signalling events.

(4)

What is the major drawback of base-level Posix signals (as opposed to *real-time* Posix signals)?

The major drawback of standard signals is that they are not queued. There is no guarantee that they will be delivered. (Real-time signals, on the other hand, are queued).

(2)

3(b) For inter-process communication, what is the fundamental difference between a *pipe* and a *mailbox*?

For a pipe to be used for inter-process communication both processes have to co-exist at the same time since the buffer which holds the information is non-persistent. For a mailbox, on the other hand, there is no requirement for the two processes to exist at the same time, whereas a pipe can be arranged to block on the reader consuming the information, there is no guarantee that any information posted to a mailbox, will ever be read.

(4)

3(c) In the context of scheduling algorithms, what is the *CPU burst duration* and why is it a useful concept? Outline the major objection to using burst duration.

A CPU burst duration is the time for which a process executes before it performs an I/O operation and hence causes a context switch. Burst duration and its distribution is a useful concept for quantitatively modelling the performance of scheduling algorithms.

The major objection to burst duration is that it can be a gross generalisation, of process behaviour although it probably fits many 'commercial' programs adequately.

(4)

3(d) In a pre-emptive multi-tasking system, a user program will spend a significant time running in privileged (or system) mode. Outline the potential problem that may arise if a context switch occurs while a user program is running privileged mode. Suggest two possible solutions.

A context switch 'arriving' when a program is executing a system call can potentially cause a problem since the kernel may be left in an indeterminate state. Two possible solutions are:

i) To disable all interrupts on entry to a system call and re-enable them just before exit. This is not an attractive option since critical interrupts may be blocked.

ii) Make the kernel either partially or fully pre-emptible. The former approach involves defining 'safe' pre-emption points when interrupts can be enabled. A fully pre-emptible kernel is also technically feasible although complicated; this can be implemented by 'locking' key kernel data structures.

(6)

4(a) In a paged memory system, what are the factors which influence the choice of page size. What is the most common page size used in modern operating systems?

Paged memory systems suffer from internal fragmentation with an average of half a page lost per process. This would tend to suggest as small a page size as possible. Small pages, however, mean a large number of pages and so the size of the page table increases. Thus the optimal choice of page size is a compromise between these two factors.

Common page sizes lie in the range ½ - 8kB; the most popular page size is 4kB.

(6)

4(b) Paged memory systems suffer from *internal* fragmentation loss. Explain how segmented memory systems solve this problem.

Segmented memory systems allocate blocks of varying size to processes and in this way, the internal fragmentation loss associated with paged memory is avoided.

(4)

What are the two main drawbacks of segmented memory systems? Why are segmented memory systems not used very widely?

Although segmented memory removes internal fragmentation, this is replaced by external fragmentation with all the associated problems of managing this situation. Additionally, segmented memory systems are more complicated.

Despite several theoretical advantages over paged memory, segmented memory is not widely used since the improved memory utilisation is not economically attractive, given the cost of memory.

(4)

4(c) Describe how the security attributes (access rights) of a file are determined? Explain the major advantage of this approach.

The access attributes of files are typically set on a per group basis. Users are organised into groups, and the group is given access (or otherwise). In practice, this means one bit has been allocated for each permission. If access was granted on a per-user basis, for a typical multi-user system a large number of bits would have to be allocated to control access to each file.

(6)