EEE412 / 6032 - WORKED SOLUTIONS – 2010/11

Question 1

(a) Describe the key elements of a device driver.

Key to the uses of a device driver is the definition by the OS of a set of generic, manufacturer-independent systems calls which typify the operation of a particular class of peripherals, for example, for a graphics adaptor, a set of functions such as draw a rectangle of the specified dimensions. The device driver then implements these generic systems calls and passes device-dependent instructions to the physical device.

[4 marks]

Why is a device driver the preferred method of interfacing with peripherals?

Historically, the code to access a peripheral was embedded in the kernel which meant that adding a peripheral required a rebuild of the OS. It also prevented 3rd party suppliers from providing peripherals. Using a device driver makes it possible to install and change peripherals easily and provides a great deal of flexibility.

[3 marks]

Are there any major drawbacks to the installable device driver architecture?

Since a device driver accesses hardware directly, it has to run in systems mode. This means that it is a potential security breach if the driver contains malicious code.

[3 marks]

(b) What are the three main types of virtual machines? Briefly describe the area of application of each type.

i) System virtual machines: These run an entire operating system on a host operating system (e.g. Linux on Windows). Such a VM can be used for developing/running applications intended for the guest operating system.

ii) Application virtual machines: Rather than making available a complete guest OS, application VMs provide the system calls for running an application intended for the guest on another machine. WINE on Linux is a good example that allows Windows programs to be run on Linux independently of the Windows OS.

iii) Java virtual machines: These provide an OS-independent environment to run Java programs. The VM here provides a means to interpret a byte-code intermediate format of a Java program.

[6 marks]

How does software emulation differ from a virtual machine?

Both VMs and software emulators can run programs intended for other machines but the key difference is at the level of machine instruction compatibility. VMs run programs written for the host CPU, albeit through via different API. Emulators, on the other hand, run programs where there has to be some translation of machine-code at runtime because the program was written for another type of CPU. This on-the-fly translation to equivalent instructions is time consuming.

[4 marks]

Question 2

(a) Why is disabling all interrupts not an appropriate way of implementing synchronisation of user processes?

Because that would give users the ability to disable interrupts and hence disrupt the OS.

[4 marks]

(b) Why are interrupts not a suitable mechanism to implement synchronisation on multiprocessor systems?

Because interrupts are not synchronised across different processors, therefore mutual exclusion cannot be correctly assured.

[4 marks]

(c) Show how two processes, P_1 and P_2, containing the following code fragments can deadlock:

P_1 = … `fopen(f1, "w"); fopen(f2, "w");` …

P_2 = … `fopen(f2, "w"); fopen(f1, "w");` …

where the `fopen()` function opens a file handle for file `f1` or `f2`; the "`w`" parameter indicates the file is being opened for write access.

If P_1 executes the first `fopen()` call for `f1` and then context switches to P_2 which then tries to open f1, the two processes will deadlock since holding write access for a file is mutually exclusive. Once P_1 has acquired f1, P_2 will be unable to open it. Similarly, P1 will be unable to open f2, and so the two processes will deadlock.

[6 marks]

Will deadlock always occur when running P_1 and P_2 together?

No. In fact, deadlock will be a rather rare event since it will only happen when there happens to be a context switch from P_1 to P_2 immediately after P_1 has executed the first `fopen()` and just before P_2 seeks to execute its second `fopen()`.

[2 marks]

How can the system call be modified to prevent deadlock occurring?

Deadlock only occurs if `fopen()` blocks when it is unable to acquire the requested file. Hence a non-blocking `fopen()` will prevent deadlock although there would need to be some programmatic test for a non-existent file pointer later in the code.

[4 marks]

Question 3

(a) In the context of memory management, what is a *hole* and what are the consequences of holes? How can holes be eradicated?

A hole is an isolated block of memory in an otherwise contiguous block. This can lead to significant memory wastage as the memory allocation tends to fragment into a large number of unusably small holes. Holes can be eradicated by periodically compacting the memory but this is a costly operation.

[5 marks]

(b) In a virtual memory system, what is the principal disadvantage of storing the page table in main memory? Explain how this disadvantage can be greatly reduced by adding suitable hardware support.

The principal disadvantage of storing the page table in main memory is that accessing a memory location requires two memory accesses: one to get the physical page address from the page table and the second to access the memory location itself. The problem can be ameliorated by caching the most recently used, say 32-4096, entries in the page table in a so-called Translation Lookaside Buffer (TLB) which is searched in the first instance for the page entry. The Principle of Locality means that we would normally expect a high hit rate on this cache.

[5 marks]

(c) In a paged memory system, pages are typically selected for replacement on the grounds of being the *least recently used* (LRU). Suggest a crude but adequate approximation to the LRU algorithm which uses only one extra bit in each the page table entry but which is much cheaper than the full LRU implementation above. Comment on the accuracy and efficiency of your approximation.

We can produce a crude but adequate approximation with a single bit field in the page table by arranging that this 'reference bit' is set every time the page is accessed. Periodically, we can clear all the bits in the page table which means that when we need to replace a page, all the page table entries with reference bits set to zero are candidates for replacement. We can then choose one at random. The accuracy of this approximation will depend on the period between reference bit clears counter-balanced by the computational cost of performing this operation.

[6 marks]

As an alternative approximation to the least recently used algorithm, one possibility would be to write the frame number of every memory access to a first-in-first-out buffer (FIFO) and select for replacement the page at the head of the FIFO. Explain why this is not a good idea.

Because this measures the oldest access, not necessarily the least recent! It may incorrectly select frequently used pages which have been in memory for a long time rather than pages which have not been used for a long time.

[4 marks]

Question 4

With reference to a graphical user interface (GUI), what is an *event* and how is it communicated to an user application?

[4 marks]

An event is an asynchronous signal by which a window component signals to an application that it requires some action to be carried out. The OS packages the event in a message which is then polled by the user application in a message loop.

What is an *event handler* and how is it invoked?

An event handler is a function which is executed in response to an event. An application polls for messages (containing the events) and typically, an application has a look-up table which maps events to their handlers. On receipt of a given event, the application uses this table to call the correct handler.

[4 marks]

What happens if a window receives an event for which no handler is defined?

The application will pass the event to its parent. (It is principally for this reason that windows are passed their parent window on creation.) If the parent process has no appropriate handler, the event is passed further up the chain until it is eventually passed to the OS. If this has no handler, the event is ignored.

[4 marks]

In a GUI program, a user places the mouse cursor over an 'OK' button and presses the left mouse button down. The users keeps the mouse button depressed, moves the mouse cursor off the 'OK' button and releases the left mouse button. Nothing happens! The action associated with the 'OK' button is not executed. Why?

The application has only been programmed to respond to the left-button-up event! An event will exist for left-button-down but unless a handler is defined, it will be ignored. (Normally, button clicks are dealt with using a button-up event since this is less prone to inadvertent operation than button up events.

[4 marks]

Why is an object oriented language particularly suited to programming GUI systems?

GUIs can be programmed in non-OO languages but a great deal of overhead code has to be written to handle message loops, *etc*. Such code is common to all GUI programs and so can be conveniently encapsulated in OO classes. The class hierarchies also mean that message loop code can be inherited by derived classes, leading to a more compact, uniform implementation.

[4 marks]