1.2.1 Systems Software

Revision sheet

1 Operating Systems

The Operating System (OS) provides an interface between the users input and the physical hardware of the computer. Operating Systems also have 'utility programs' which can aid the performance of the computer.

1.1 Features Of An OS

They can schedule jobs on the CPU; allocate resources; control the hardware; protect the work of each user (where there are more than one user per computer); manage memory; provide disk storage for each user; run multiple jobs for each user.

1.2 Purpose Of An OS

File system management; input and output control; memory management; process control.

1.3 Types Of OS

There are a number of different types of operating systems, each are best utilised in different situations.

1.3.1 Distributed

This is a collection of independently networked node. Each node has its own hardware and there will be system management software which connects all the systems together and coordinates them. Usually there is a supercomputer at the heart of the network. Examples include World Community Grid and Seti @ Home.

1.3.2 Embedded

These have a dedicated function and are part of a larger device. They generally have low power consumption, cost little and use a limited amount of hardware resources - this means they have to be interfaced with in a bespoke manner. Examples include operating system to run fridge, washing machine, microwave - all of which will usually be programmed in assembly language.

1.3.3 Multi-Tasking

This is the standard modern OS. It allows many tasks to run at once.

1.3.4 Multi-User

This allows many different users to use the systems resources concurrently. Processor time is shared, and very little processing is done at the terminals. Examples include supermarket tills.

1.3.5 Real-Time

These run under pre-defined time deadlines so they can be classed as time critical. Examples of uses include aeroplane control systems.

1.3.6 Network

These allow users to share resources such as printers and data but each of the terminals have their own processing capabilities. They also have security features and can grant different permissions to different groups of people.

2 Memory Management

Memory is a very limited resource on most modern computers. It has to be managed so that processes that need to run at the same time can, whilst protecting each processes memory from other processes.

2.1 Types Of Memory

There are two main types of memory.

2.1.1 Primary Memory

Also known as *main memory.* It is physically made up of RAM (*Random Access Memory*, a typical PC would have between 4 and 32GB) and is volatile. Software which is being processed in the CPU (*executing software*) is stored here. The CPU also contains volatile memory in the form of registers and its cache.

2.1.2 Secondary Memory

Also known as *secondary storage*. It is non-volatile and typically made up of a SSD or HDD, a typical PC would have between 500GB and 4TB. This is used to hold data and program s which are not currently being used, but will need to be accessed at some point.

2.2 Virtual Memory

This uses the secondary storage as an extension of the primary memory. This is considerably slower as the secondary storage has a much slower access time than primary memory.

2.2.1 Disk Thrashing

This can occur when the hard drive is accessed very frequently - for example when primary memory is very small so virtual memory has to be used extensively. Disk Thrashing is not good for the hard disk and can lead to failures.

2.3 Dividing Up Memory

There are two methods which can be used to divide memory into smaller chunks.

2.3.1 Paging

This uses physical divisions of the memory and divides the program into equal sized parts. The small chunks of program can be stored anywhere in memory, this doesn't affect the performance of the program.

2.3.2 Segmentation

This uses logical divisions and divides the program into different sized parts, based on bits of program. The chunks of program work best if they are stored contiguously in memory.

3 Interrupts

These are useful in computers because rather than continually polling the inputs for changes, as soon as an event (for example, a key is pressed on the keyboard) happens, the CPU interrupts what is doing and actions the event.

3.1 Interrupt Service Routine

The ISR gets called whenever an interrupt is detected. It handles the interrupt there an then, before handing control back over to the CPU. This means, for the example of pressing a key on the keyboard, the key which has been pressed is retrieved then stored in the keyboard buffer; the IRS would then hand over to the CPU which can later make use of this input.

3.2 Servicing the Interrupt

When there are no interrupts, the CPU continually cycles through the fetch-decode-execute (FDE) cycle. An interrupt causes the process to suspend what it is doing, so that it can run the ISR. Once the ISR has finished executing, the processor must return to the instruction it was executing, and continue from where it left off. The processors execution will be affected by the interrupt in the following way.

1. Processor receives the interrupt
2. Processor completes FDE cycle for current instruction
3. Contents of processors registers are saved to memory
4. Origin of the interrupt is found, so appropriate ISR is called
5. All lower-priority interrupts are put on hold, so the ISR can finish
6. PC is updated with the address of the first instruction of the ISR
7. ISR completes execution
8. Processor's registers are reloaded with what was saved to memory
9. Lower priority interrupts that were put on hold are re-established
10. PC is set to point to the address of the next instruction that needs to be executed in the program that the processor was running when it received the interrupt.

3.3 Priorities

Interrupts have different priorities. Through this, the computer is able to determine what the most important interrupts to process are. The interrupts with the highest priority are generally to do with power supply to the computer, and the lowest are to do with input or output devices.

3.4 Problems

There are a number of problems with interrupts: will the CPU respond quick enough and what happens when there are too many interrupts at the same time.

4 Scheduling

This is the process by which the many different processes which need access to the resources of the computer are given access. It is used to make the most efficient use of the processor; be fair to all applications; provide a reasonable response time and to prevent processes from failing to run/ process starvation/ deadlock.

4.1 Types Of Scheduling

There are a number of different types of scheduling. They are all fundamentally the same - processes are stored on the 'ready queue' until the processor is ready for them.

4.1.1 Round Robin

This allocates a fixed amount of time per process and cycles through them until all the processes are complete. It is used on network operating systems, to cycle through users.

4.1.2 First Come First Served

This is the simplest as it puts the processes in a queue when they arrive at the CPU and executes them in the order they arrived in - first in, first out. This has a problem though, low priority processes may be completed before high priority processes.

4.1.3 Multi-Level Feedback Queues

This categorises processes into multiple different queues. Each queue has its own priority for processing and within each queue, there is a priority ordering.

4.1.4 Shortest Job First

The scheduler prioritises the smaller jobs to get them done first, meaning bigger jobs might not get done.

4.1.5 Shortest Time Remaining

This is similar to shortest job first; except, if a job arrives into the ready queue which has less time remaining than that currently executing, the new job will interrupt the current job and be completed rather than joining the queue.

4.2 States

Jobs are programs which are loaded into memory which can potentially be executed by the CPU. They are usually in one of three states.
Running - its code is running in the CPU.
Ready - Ready to run, but not active in the CPU.
Blocked/ Waiting - Waiting for a resource to become available (eg. file or user input).

5 Device Drivers

Drivers are computer programs that are usually provided by the manufacture of the device and allow the operating system to interact with hardware. Device drivers are specific to the computer's architecture. This means different devices must use different drivers. Drivers are also specific to the operating system installed on the device.

6 BIOS

The *Basic Input Output System* is a piece of software which is the first thing to run on the computer when it turns on. The program counter points to the location of the BIOS upon each start-up of the computer as the BIOS runs tests before the operating system is loaded into memory. These tests include: POST (power on self test, ensures all hardware is connected correctly and functional); check if the CPU clock, memory and processor is operational; testing for external memory devices connected to the computer. It is only after these tests have been completed that the operating system can be loaded into RAM form the hard disk.

7 Virtual Machines

This is a computer inside a computer.

7.1 Types of Virtual Machine

There are two types (and key uses) of virtual machines.

7.1.1 Process VM

This is a programming environment that allows a program written for one type of machine to be run on the other types of machine without any any changes being necessary. For example, the Java Virtual Machine (JVM) allows Java bytecode produced on one type of OS to be executed on another type of OS without any syntax changes.

7.1.2 System VM

This is a software application run by the host OS, which contains an emulation of a second OS within it. This permits the installation and execution of software applications on the VM as if they were being run on a separate computer.

7.2 Intermediate Code

This is code that is halfway between machine code and object code. It is independent of processor architecture therefore can be used across different machines and operating systems. An example of this is using the JVM to execute Java. Running intermediate code through a virtual machine can be considerably slower than running low-level code on the device it was designed for.