# **Operating Systems**

- An Operating System is software that manages a computer.

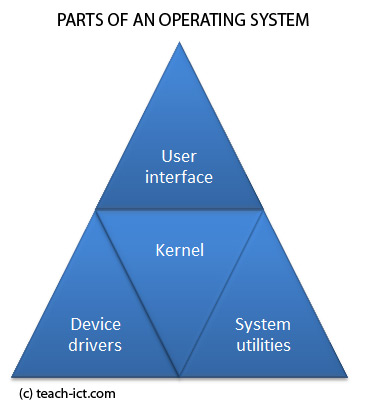
- Modern OSs have several purposes:

- To manage the hardware of a system.

- To manage programs installed and being run.

- To manage the security of the system.

- To provide an interface between the user and the computer.

**Parts of an Operating System**

- User Interface.

- Kernel.

- Device Drivers.

- System Utilities.

**Kernel**

- The Kernel is the very core of the OS. It helps manage the system resources including:

- Loading/Unloading applications.

- Scheduling tasks to run efficiently in the CPU.

- Memory management.

- File storage to/from secondary storage.

- File management.

- Data security.

**Device Driver**

- Required for every piece of hardware that communicates with the OS both internal and external.

- Device Driver enables the OS to control and communicate with the device.

- There could be hundreds of device drivers pre-installed with the OS and the right ones for a given set-up are loaded on boot up.

- The exact details of which device driver is needed by the OS is kept in a file:

- “registry” in windows.

- “config files” in Linux.

**User Interface**

- UI allows the user to interact with the computer.

- Types of UI include:

1. CLI (Command Line Interface)
   1. Users interact by typing commands into a screen.
2. GUI (Graphical User Interface)
   1. Uses graphical components and a mouse pointer in a user friendly manner.
3. Menu Based
   1. Limited range of options could be used in a restaurant/airport.
4. Natural Language
   1. Voice recognition.

**System Utilities**

- The part of the OS that provides all the basic facilities that run in the background without UI.

- Print spool services.

- Cryptographic password management.

- File management services.

- You may also install:

- Anti-virus software.

- Disk defragmentation software.

- File compression software.

**Different Types of OS**

Operating Systems can be classed in different ways including:

-Single or multi-use.

- Multi-tasking.

- Embedded.

- Real-time.

- Distributed.

**Multi Tasking**

- When you use a computer you will often be running multiple programs at once. This is organising by a multi-tasking OS.

- Processors may have to deal with more processes than they have cores. Multi-tasking allows for a single core to deal with multiple processes.

- This is possible because the speed of the processor is significantly greater than the speed of other parts of the computer.

- By rapidly switching between processes, a CPU gives the impression that it is running multiple programs at once.

- The part of the OS that controls this is called the **scheduler**.

**Multi User**

- A computer at home where different users can login is not necessarily a true multi user computer.

- A true multi user computer OS must allow more than one person to share a computer’s resources at the same time.

- Multi user OSs are common on **mainframe/super computers** as there may be many accessing them simultaneously.

- A multi User OS carries out the following tasks:

- Each user logs onto the system and is presented with a workspace.

- Allocated resources for the job they want to run.

- Keeps logs of how much time and resources are being used.

- Works out the most efficient use of the computer processing cycle.

- Maintain security and privacy between different users.

**Distributed OS**

- Sometimes we want to combine the power of a group of computers to work together on a single task, this is what a distributed OS is for.

- Distributed computing is a form of parallel processing.

- Used for resource intensive work such as scientific problems and rendering.

**Embedded OS**

- An embedded system will most likely have one job and no need for multi tasking such as simple appliances and ATMs.

- Often specifically designed for the device on which they run with efficiency as a priority to operate on low end CPUs with little RAM.

- Designed for a specific task on a specific system hardware.

**Real Time OS**

- Designed to carry out actions within a guaranteed time (latency) even when left running for long periods.

- They must be reliable and have precise and constant timings. Response times normally a fraction of a second.

- Safety systems will often run on RTOS such as autopilot.

- Also suitable for:

- Factory production.

- Car engine management.

- Rovers.

**Batch OS**

- An OS with the task of doing the same job over and over again with different input data presented for each iteration.

- Common on a mainframe computer purchased with a massive repetitive data process in mind.

- Batch process differs from RTOS in that it requires separate programs for input, output and processing. This means all the input is handled first then processing then outputs.

- Real Time on the other hand requires continual input-process-output activity, as soon as an input arrives, it is processed as output.

**Memory Management**

- Computer memory is a limited resource.

- One of the key jobs of an OS is the management of memory.

- Memory stores programs and data in use by the system. Memory management allows multiple processes to be stored in memory safely and efficiently.

- Memory management protects programs and data from eachother.

- Each program will be using its own data. Without memory management one program could change the data of another and a maliciously coded program could access the data of another program.

- The memory management part of an OS restricts each program to accessing and amending its own area of data.

- Memory management allows programs to use more memory than a system has - **virtual memory**.

- Different types of memory in the system must be used properly to ensure efficiency.

- Memory management can be split into two broad tasks.

- Each program must have enough memory with which to execute.

- It can neither run into memory space of other programs or be run into.

**How is memory divided?**

Sometimes there isn’t a space in memory large enough to fit data in as a whole piece so it needs to be split up. There are two main ways to do this:

**Segmentation**

The allocation of sections or segments of memory to allow the process to run, These segments are not a fixed size they are as large as necessary to hold the process.

The memory segment given to the process is further divided into several sections:

**Code Segment -** where the program instructions reside.

**Data Segment** **-** memory space allocated for a program’s variables.

**Stack Segment -** For functions and subroutines.

- Whenever an instruction is interrupted by a subroutine, the program notes the address where it left off.

- The address is kept in the stack whilst the sub runs.

- When the sub finishes, the program returns to the address stored in stack.

- More subs - larger stack could have stack overflow.

**Pagination**

- Each page is a fixed size division of memory (typically a number of KB)

- The OS uses a page table to keep track of where the pages are stored. This means all the pages of a process don’t have to be stored continuously.

- Mostly used for **virtual memory.**

**Comparison**

|  |  |
| --- | --- |
| **Similarities** | **Differences** |
| - Both are ways of dividing memory but have different purposes.  - Bot hare assigned by the memory manager as and when they are needed.  - Both are managed by the memory manager within the OS.  - Neither physically divide memory, they are just different ways for the OS to manage memory efficiently. | - Pagination only used for virtual memory. Segmentation always used.  - Pagination is used to swap data in and out of VM.  - Pages have fixed size; segments vary.  - Pages are not subdivided whereas segments are subdivided by code, data, stack and free elements.  - Pages are continuous. |

**Virtual Memory**

- RAM is significantly more expensive than secondary storage.

- A system will have more secondary storage than RAM.

- When a system is running low on RAM it can use part of the secondary storage as **virtual memory.**

**Disk Thrashing**

- When the OS believes a page is not likely to be needed in the near future, it is more from RAM to VM. When the page is needed again it is moved back.

- Slower than than having everything in RAM so ideally it would be used less often. If RAM is full, the OS can end up moving pages between RAM and VM too much significantly slowing performance.

- This is known as **disk thrashing.**

**Memory Problems**

**Memory Leaks**:

- If a memory location is in use, the memory manager will mark it as being unavailable.

- A well written program will flush and declare free the space in memory it was occupying.

- A badly written program might not do this.

- The system will run slower and slower and might run out of memory.

**Stack Overflow:**

- If a poorly written program has an excessively deep or infinite recursion the stack can run out of free memory - called a stack overflow.

**Systems Software**

**Scheduling**

- Multi-tasking Operating Systems need to make sure that multiple processes can run alongside one another, apparently simultaneously.

- For this to be possible, OS need to carry out scheduling and this is the job of a scheduler.

- A scheduler is a program that manages the amount of time each process has in a CPU.

**Round Robin**

- Each process is given a fixed amount of time. If it hasn’t finished in that time it goes to the back of the queue.

Advantages:

- Simple to implement.

- Good if all processes are approximately the same size and priority.

Disadvantages:

- Does not take into account the priority of a process.

- Does not consider the size of processes which makes it an inefficient way to allocate resources.

**First Come First Serve**

- Processes are finished in the order they start with each process being finished before the next starts

Advantages:

- Simple to implement.

- Once a process starts it will finish in minimum time.

Disadvantages:

- Prevents other processes from running until the one it is working on is finished.

- Does not consider the priority or size of processes.

**Shortest Job First**

-Shortest job first orders its jobs by the amount of FDE cycles it will take and executes them starting with the smallest first.

- If a new process comes along, the scheduler works out the processing time and slots it into the queue.

- If it is shorter than the current process, the current process will be interrupted and the shorter one will run.

Advantages:

- Ensures maximum number of jobs are completed.

- Ensures that short jobs aren’t kept waiting whilst a long job takes resources.

Disadvantages:

- Scheduler estimates of number of cycles could be wrong.

- Long jobs may never be completed if short jobs keep getting added.

- Does not take priority into account.

**Shortest Time Remaining**

- Do I really have to explain this one.

**Multi-level Feedback Queues**

- Takes into account the priority of a task by maintaining queues with different priority levels.

- When a process arrives it is placed into the relevant queue.

- The high priority queue is completed first followed by the other queues.

- The algorithm can change a tasks priority if:

- It receives new info.

- A job has been waiting too long.

Advantages:

- Makes use of process priority.

- Ensures high priority tasks run on time.

Disadvantages:

- Complex to implement.

- Not efficient if all processes have similar priority.

- Low priority tasks may take a while to complete.

**Polling**

- The CPU stops at regular intervals to check with every connected device to see if CPU action is required.

- Very inefficient seeing as this takes substantial processing time.

- Easy to implement.

**Interrupts**

- When a device or event sends a signal to a processor to get its attention.

- Interrupts will have a priority indicating how urgently it requires attention.

- When an interrupt is raised the OS runs the relative Interrupt Service Routine.

**Hardware Interrupts**

- It is efficient to assign an interrupt line to each device. This allows the CPU to continually process other things

- If a device requires attention it sends an interrupt to the CPU.

**Software Interrupts**

- Software interrupts can be generated by the processor itself or by a special instruction within the instruction set.

- A software interrupt sometimes called a ‘trap’ or ‘exception’ is generated by the processes executing a specific instruction.

- Requires more hardware as you need “interrupt” wires for each device.

- The CPU needs to be able to jump out of what it is doing, handle the input then get back to what it was doing which is complicated.

- Multiple interrupts could be received simultaneously.

- This can lead to **thrashing** where the CPU is spending all its time servicing interrupts rather than doing the jobs in hand.

**Problems With Interrupts:**

**Responding Quickly Enough (Latency)**

- For very precise tasks software needs to be written that can respond to interrupts very quickly.

**Too Many Interrupts (Stack Overflow)**

- The stack can fill up because too many interrupts are waiting to run. This will cause a stack overflow.

- Sometimes a polling system is the way to go for low demand situations.

**Interrupt Service Routine (ISR)**

- When a peripheral or software requires attention an interrupt is raised to tell the CPU.

- Each interrupt has a priority level.

- The OS has ISRs that determine what happens when a particular interrupt is carried out.

- The OS has ISRs that determine what happens when a particular interrupt is carried out.

- At the end of each iteration of the FDE cycle the processor checks to see if there are any interrupts.

- If there are any interrupts of higher priority than current task then:

- Content of Program Counter and other registers are copied into an area of memory called the tack.

- The relevant interrupt service routine is located by changing the PC to the value where the ISP starts in memory.

- When the ISR is complete the previous values of the PC and the other registers can be restored from memory.

- This process is sometimes called a context switch.

- A **context switch** is a process to change from one task to another while ensuring the tasks do not conflict.

- Interrupts can interrupt interrupts.

**Virtual Machines**

- Software written for one computer system may not work on another one.

- You can write a program that has the same functionality as a physical computer. We call such program **virtual machines.**

- A common use of VMs is to run Operating Systems within another Operating System.

- This helps with compatibility and running old apps.

- Easy to back up and reverse to older states.

- Infected VMs are contained.

**Virtualisation:** The process of creating virtual machines.

- Some programs may run more slowly in a VM as it requires overhead for host machine in terms of CPU usage and memory.

- Could be used to interpret intermediary code (I.e JVM).

**BIOS**

- Basic Input Output System.

- Small program stored in non-volatile memory on the motherboard.

- In the past this was a ROM chip but now flash memory is used.

- Controls start-up sequence of computer.

- First thing that happens when on button is pressed is the power supply fires up and once it is producing a steady output a reset command is sent to CPU.

- The CPU reset command has been hard-coded by its designers to execute an instruction at a specific location within the BIOS chip.

- This location contains a **jump** command that points it to the starting address of the BIOS start-up program.

- This BIOS is separate from the OS. It has to work before the OS is loaded.

- The BIOS does some basic checks. These checks are called POST for short (Power On Self Test). They Include:

- System Memory is okay.

- System clock/timer is working.

- Processor is okay.

- Keyboard is present.

- Screen display memory is working.

- BIOS is not corrupted.

- Booting happens very quickly but should there be a problem a beeper or speaker can alert the user.

- These **beep codes** inform you of the problem.

- After POST has been processed the BIOS will initialise the video card and the screen will display something. Normally the BIOS maker’s startup message.

- Then it looks for peripherals and starts them.

- Now hardware is working the BIOS will look for an OS to start up.

- BIOS will have a priority order for looking for bootable drive.

- Once a bootable drive has been found the instructions in the **boot sector** will be loaded.

- The boot sector is the first region of storage that the computer will read on that device.

- The boost sector contains the **master boot record (MBR)**

- The MBR is a data table indicating how the memory is partitioned.

- The MBR points the BIOS to the **primary partition** which contains the OS.

- The CPU will execute the code in the boot sector of the primary sector. This is called the **boot loader.**

- The boot loader is a program and part of the OS.

- The boot loader will first load the kernel into memory then other parts of the OS.

- After the OS is loaded a script called the **Boot File or Boot Script** is run to prepare the computer for personal use. Ie desktop and personalised appearance.

**UEFI and GPT**

- Classical BIOS is limited- 16 bit 1MB of space.

- Has been replaced by UEFI - Unified Firmware Interface.

- MBR has been replaced by GPT

**Safe Mode**

- If there is a problem with OS set up then by pressing a key the boot loader will load the OS in safe mode.

- Stripped down version of OS that contains minimal set of device drivers and screen set to low resolution so even a very simple monitor can display it.