3.4 System software

3.4.1 Purposes of an operating system

The operating system (OS) must provide and manage hardware resources as well as provide an interface between the user and the machine as well as between applications software and the machine.

The OS must also provide other services such as data security.

An operating system (OS) is a set of programs that manage computer resources and run in the background on a computer system, giving an environment in which application software can be executed.

Responsibilities of the OS include:

- Hiding the complexities of hardware from the user.
- Managing between the hardware's resources which include the processors, memory, data storage and I/O devices.
- Handling "interrupts" generated by the I/O controllers
- Sharing of I/O between many programs using the CPU.

The 2 types of software that usually reside on an OS are:

- **System Software –** programs that manage the operation of a computer
- **Application Software –** programs that help the user perform a particular task.

3.4.1 Purposes of an operating system

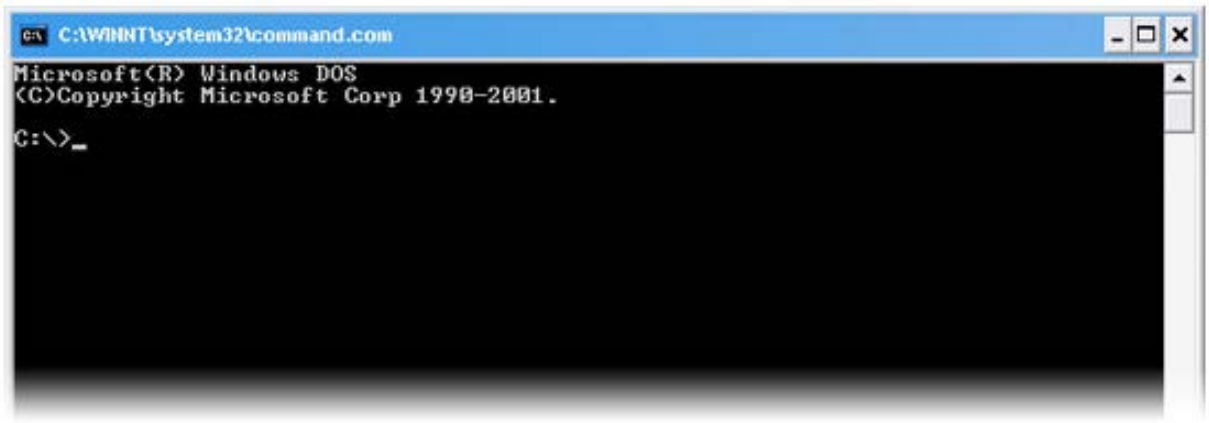
User Interfaces

The user interface is the interaction between the User and the Machine, letting the user send commands with the expected results. Two types of interfaces are:

- Command Line Interface (CLI)
- Graphical User Interface (GUI)

Command Line Interface (CLI)

A command-line interface allows the user to interact with the computer by typing in **commands**. The computer displays a prompt, the user keys in the command and presses the enter button.



Features of a command-line interface

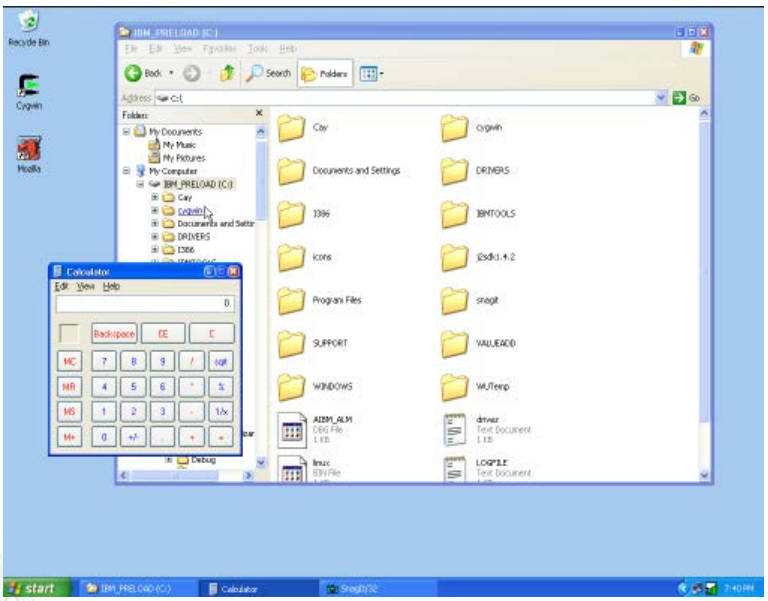
- Commands must be typed correctly and in the right order or the command will not work.
- Experienced users who know the commands can work out very quickly without having to find their way around the menus.
- An advantage of command driven programs is that they do not need the memory and processing power of the latest computer and will often run on lower spec machines.
- A command-line interface can run many programs, for example a batch file could launch half a dozen programs to do its task.
- An inexperienced user can sometimes find a command driven program difficult to use because of the number of commands that have to be learnt.

3.4.1 Purposes of an operating system

An example of command line interface is **MS-DOS.** The MS-DOS command to display all files on the C drive would be "**dir c:\"**

Graphical User Interface (GUI)

Graphical user interface is sometimes shortened to **GUI.** The user chooses an option usually by pointing a mouse at an icon representing that option.



Features of GUIs include:

- Much easier to use for beginners.
- They enable you to easily exchange information between software using cut and paste
- They use a lot of memory and processing power. It can be slower to use than a command-line interface.
- They can be irritating to experienced users when simple tasks require a number of operations.
- Examples of GUIs are **Windows XP, Windows 8, Apple OSX, and Ubuntu.**

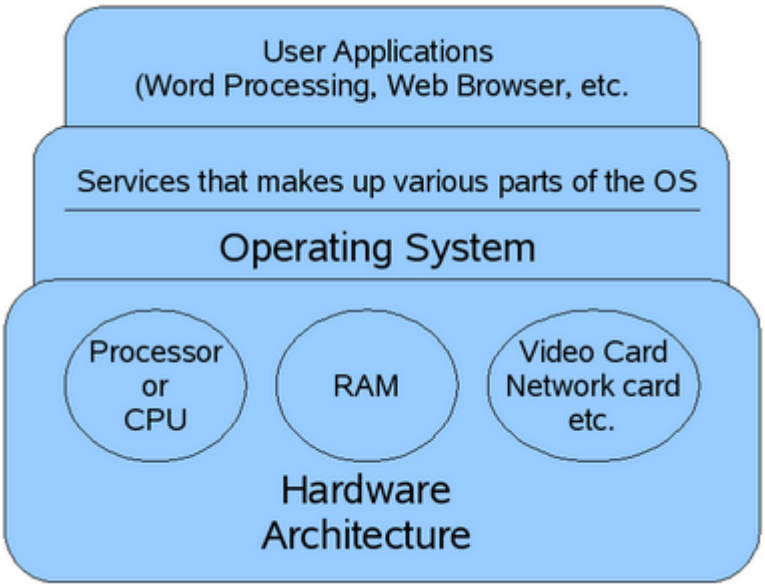
3.4.1 Purposes of an operating system

Topic	Command line (CLI)	GUI
Ease	Due to a higher degree of memorization and familiarity needed for operation and navigation, new users find operating a command line interface more difficult than a GUI.	Because a GUI is much more visually intuitive, new users almost always pick up this interface faster than a CLI.
Control	Users have more control over both the file and operating systems in a command line interface. For example, users can copy a specific file from one location to another with a one-line command.	Although a GUI offers ample access to the file and operating systems, advanced tasks may still need to utilize the command line.
Multitasking	Although many command line environments are capable of multitasking, they do not offer the same ease and ability to view multiple things at once on one screen.	GUI users have windows that enable a user to view, control, manipulate, and toggle through multiple programs and folders at same time.
Speed	Command line users only need to utilize their keyboards to navigate a the interface. Additionally, they often only need to execute a few lines to perform a task.	Using both a mouse and keyboard to navigate and control your operating or file system is going to be much slower than someone who is working in a command line.
Resources	A computer that is only using the command line takes a lot less of the computer's system resources than a GUI.	A GUI requires more system resources because of the elements that require loading, such as icons and fonts. In addition, video, mouse, and other drivers need to be loaded; taking additional system resources.
Scripting	A command line interface enables a user to script a sequence of commands to perform a task or execute a program.	Although A GUI enables a user to create shortcuts, tasks, or other similar actions, it doesn't even come close in comparison to what is available through a command line.
Remote access	When accessing another computer or device over a network, a user can only manipulate the device or its files with a command line interface.	Although remote graphical access is possible. Not all computers and network equipment has this ability.
Diversity	After you've learned how to navigate and use a command line, it's not going to change as much as a new GUI. Although new commands may be introduced, the original commands always remain the same.	Each GUI has a different design and structure when it comes to performing different tasks. Even different iterations of the same GUI, such as Windows, can have hundreds of different changes between each version.
Strain	The command line allows the user to keep their hands on the keyboard, almost never touching the mouse. Moving back and forth between a keyboard and mouse can cause additional strain and may help contribute to Carpal Tunnel Syndrome.	Although shortcut keys can help reduce the amount of times you have move from the keyboard to the mouse, you will still be moving much more between devices in a GUI.

3.4.1 Purposes of an operating system

The operating system is designed to hide the complexity of the PC's hardware. In order to achieve this, it offers different features.

- Provides an interface between computer and the user.
- Input and output devices can be controlled.
- The system's resources are managed by the OS and can also be manages by the user using the OS.
- Reads and writes from or to a disk, also providing an interface for the user to manage the files.
- Controls user access rights, by restricting different actions or interfaces

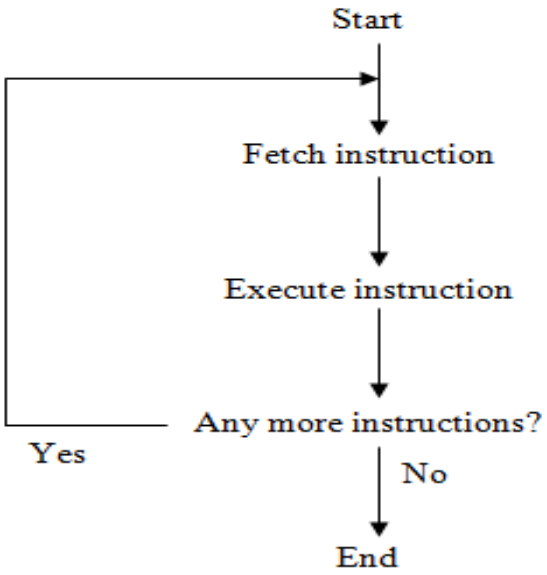


This shows that the OS hides the hardware's complexity by providing a user friendly interface and controlling the system's resources so the user has a better experience.

3.4.1 Purposes of an operating system

Interrupts

The simplest way of carrying out instructions is shown below,

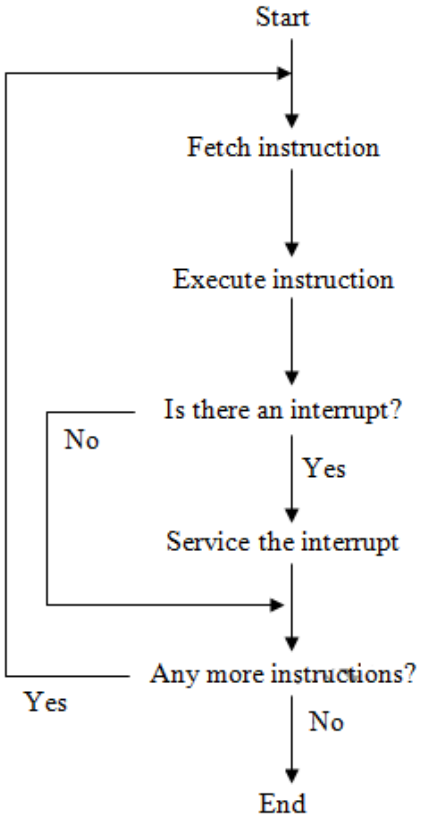


This is satisfactory as long as nothing goes wrong. Unfortunately, things do go wrong and sometimes the normal order of operations needs to be changed. For example, a user has used up all the time allocated to his use of the processor. This change in order is instigated by messages to the processor, **called interrupts**. There are a number of different types of interrupts. **The nature of each of these types of interrupt is:**

- **I/O interrupt**
  - Generated by an I/O device that a job is complete or an error has occurred. Example may be **printer is out of paper** or **printer is not connected**.
- **Timer interrupt**
  - Generated by an internal clock indicating that the processor must attend to time critical activities (see scheduling later).
- **Hardware interrupt**
  - For example, power failure which indicates that the OS must close down as safely as possible
- **Program interrupt**
  - Generated due to an error in a program such as violation of memory use (trying to use part of the memory reserved by the OS for other use) or an attempt to execute an invalid instruction (division by zero)

3.4.1 Purposes of an operating system

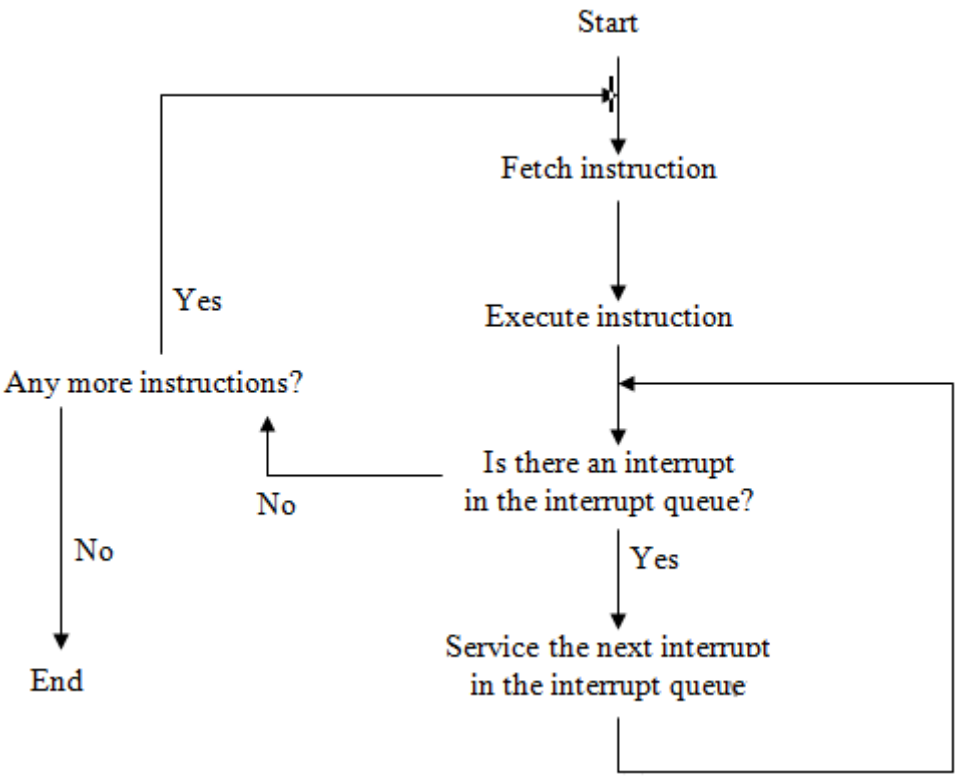
If the OS is to manage interrupts, the sequence shown above needs to be modified as shown below.



This diagram shows that after the execution of an instruction, the OS must see if an interrupt has occurred. If one has occurred, the OS must service the interrupt **in the case that it is more important than the task already being carried out** (see priorities later). This involves obeying a new set of instructions. The real problem is **'how can the OS arrange for the interrupted program to resume from exactly where it left off?'** In order to do this, the contents of all the registers in the processor must be saved so that the OS can use them to service the interrupt.

Another problem the OS has to deal with happened if an interrupt occurs while another interrupt is already being dealt with. There are several ways of dealing with this, but the simplest way is to place the interrupts in a queue and only allow the processor to return to the originally interrupted program **when the queue is empty**. The order of processing is shown in the diagram below.

3.4.1 Purposes of an operating system



The queue of interrupts is the normal first in first out (FIFO) queue and holds indications to the next interrupts that have to be dealt with.

3.4.1 Purposes of an operating system

Scheduling

The earliest operating systems were used to control single-user computer systems. In those days, the operating system would read in one job, find the data and devices the job needed, let the job run to completion, and then read in another job. In contrast, computers today with modern OS systems are capable of **multiprogramming** or executing many programs simultaneously. With multiprogramming, when a job cannot use the processor, the system can suspend, or interrupt the job freeing the processor to work on another job.

OS makes multiprogramming possible by capturing and saving all the relevant information about the interrupted program before allowing another program to execute. When the interrupted program is ready to begin executing again, it can resume execution just where it left off. Multiprogramming allows the OS to run thousands of programs simultaneously for users who might be working on different projects at different physical locations around the world.

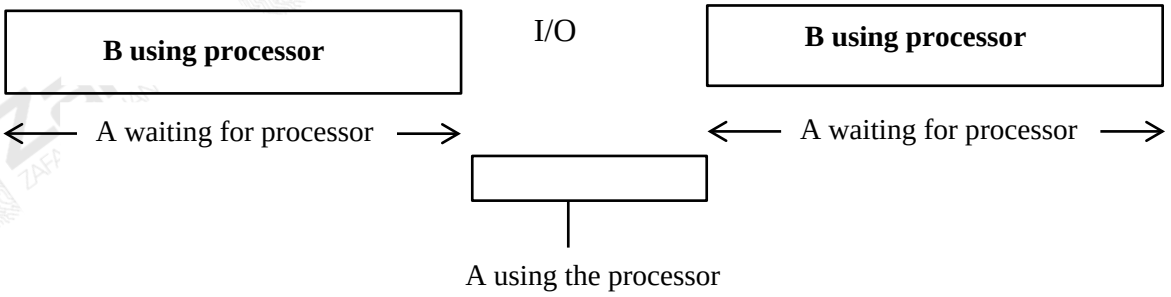
So, one of the tasks of the OS is to arrange the jobs that need to be done into an appropriate order. The order may be chosen to ensure that maximum use is made of the processor; another order may make one job more important than the other. In the latter case, the OS makes use of priorities.

Suppose the processor is required by **program A**, which is printing wage slips for the employees of a large company, and by **program B**, which is analyzing the annual, world-wide sales of the company which has a turnover of many millions of dollars.

Program A makes little use of the processor and is said to be **"I/O bound"**.

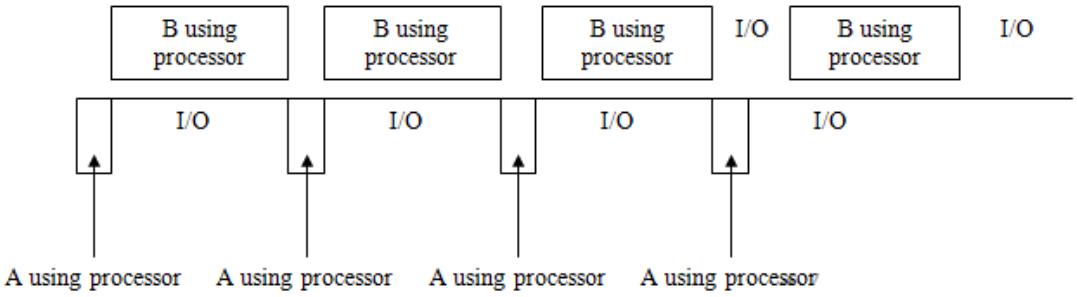
Program B makes a great deal of the processor and is said to be **"Processor Bound"**. B

If **program B** has priority over **program A** for use of the processor, it could be a long time before program A can print any wage slips. This is shown in the diagram below.



3.4.1 Purposes of an operating system

The following figure shows what happens if **program A** is given priority over **program B** for use of the processor. This shows that the I/O bound program can still run in a reasonable time and much better throughput is achieved.



The objectives of scheduling are to:

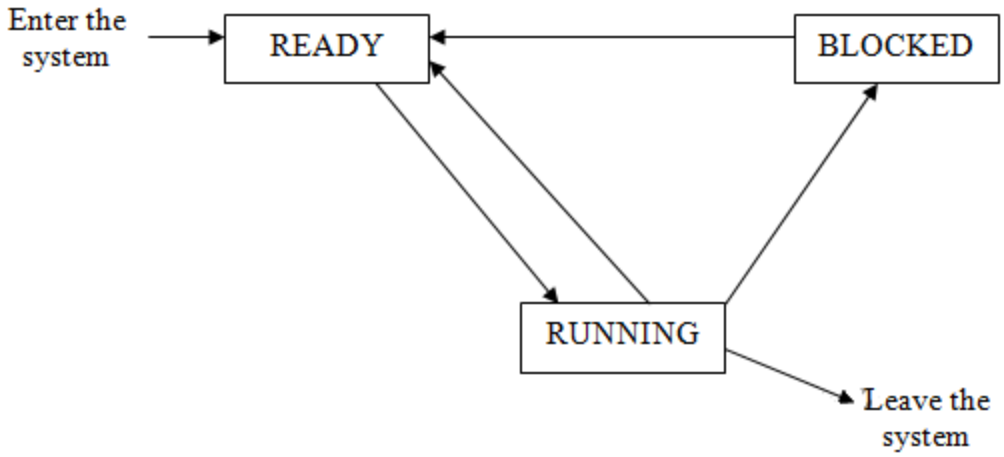
- Maximize the use of the whole computer system
- Be fair to all the users
- Provide a reasonable response time to all the users, whether they are on-line users or a batch processing user
- Prevent the system failing if it has become overloaded
- Make sure that the system is consistent by always giving similar response times to similar activities from day to day.

To achieve these objectives, some criteria are needed in order to determine the order in which jobs are executed. The following is a list of criteria which may be used to determine a schedule which will achieve the above objectives.

- **Priority.** Give some jobs a greater priority than others when deciding which job should be given access to the processor.
- **I/O or Processor Bound.** If a processor bound job is given the main access to the processor, it could prevent the I/O devices being serviced efficiently.
- **Type of job.** Batch processing, on-line and real-time jobs all require different response times.
- **Resource requirements.** The amount of time needed to complete the job, the memory required, I/O and processor time.
- **Resources used so far.** The amount of processor time used so far, how much I/O used so far.
- **Waiting time.** The time the job has been waiting to use the system.

3.4.1 Purposes of an operating system

In order to understand how scheduling is accomplished, it is important to realize that any job may be in one, **and only one,** of three states. A job may be **ready** to start, **running** on the system or **blocked** because it is waiting for a peripheral, for example, the figure below shows how jobs may be moved from one state to another. Note that a job can only enter the **running** state from the **ready** state. The **ready** and **blocked** states are queue that may hold several jobs. On a standard single processor computer, only **one** job can be in the **running** state. Also, all jobs entering the system normally enter from the **running** state and (normally) only leave the system from the **running** state.



When entering the system, a job is placed in the ready queue by a part of the OS called the **High level Scheduler (HLS).** The HLS makes sure that the system is not overloaded.

Sometimes it is necessary to swap jobs between the main memory and backing store (see memory management later). This task is done by the **Medium Level Scheduler (MLS)**.

Moving jobs in and out of the ready state is done by the **Low Level Scheduler (LLS)**. The LLS decides the order in which jobs are to be placed in the running state. There are many policies that may be used to do scheduling, but they can all be placed in one of two classes. **These are pre-emptive and non-pre-emptive policies.**

A pre-emptive scheme allows the LLS to remove a job from the running state so that another job can be placed in the running state.

A non-pre-emptive scheme allows each job to run until it no longer requires the processor. This may be because it has finished or it needs an I/O device.

3.4.1 Purposes of an operating system

Some common scheduling policies are:

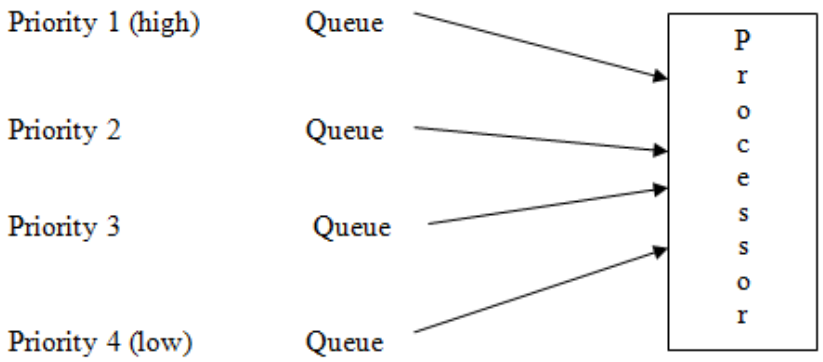
- **First Come First Served (FCFS)**
- **Shortest Job First (SJF)**
- **Round Robin (RR)**
- **Shortest Remaining Time (SRT)**
- **Multi-level Feedback Queues (MFQ)**

And there are many more.

- **FCFS**
  - Simply means that the first job to enter the queue is the first to enter the **running** state. This policy favors long jobs.
- **SJF**
  - Simply means sort jobs in the ready queue in ascending order of time needed by each job. New jobs are added to the queue in such a way as to preserve this order.
- **RR**
  - This gives each job a maximum length of processor time (called a time slice) after which the job is put at the back of the ready queue and the job at the front of the queue is given use of the processor. If a job is completed before the maximum time is up, it leaves the system
- **SRT**
  - The ready queue is sorted on the amount of expected time still required by a job. This scheme favors short jobs even more than **SJF**. Also there is a danger of long jobs being prevented from running.
- **MFQ**
  - Involves several queues of different priorities with jobs migrating downwards.

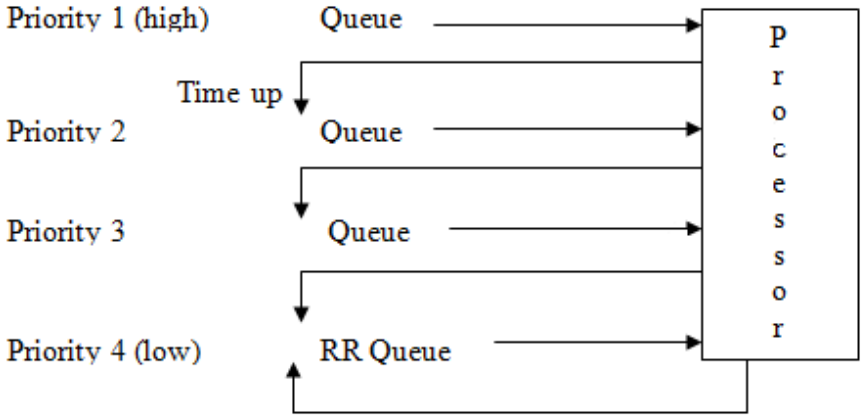
For **SJF**, the short jobs are processed first depending on the time required; they will be sorted in this order once. While **SRT** will mean that once an interrupt is raised, the queue will be reshuffled again with the shortest job first. In this way, **SRT** favors short jobs more than **SJF**.

3.4.1 Purposes of an operating system



There are other ways of allocating priorities. Safety critical jobs will be given very high priority, on-line and real time applications will also have to have high priorities. For example, a computer monitoring the temperature and pressure in a chemical process whilst analyzing results of readings taken over a period of time must give the high priority to the control program. If the temperature or pressure goes out of a pre-defined range, the control program must take over immediately. Similarly, if a bank's computer is printing bank statements overnight and someone wishes to use an ATM, the ATM job must take priority. This scheme is shown below. It shows that queues are needed for jobs with the same priority.

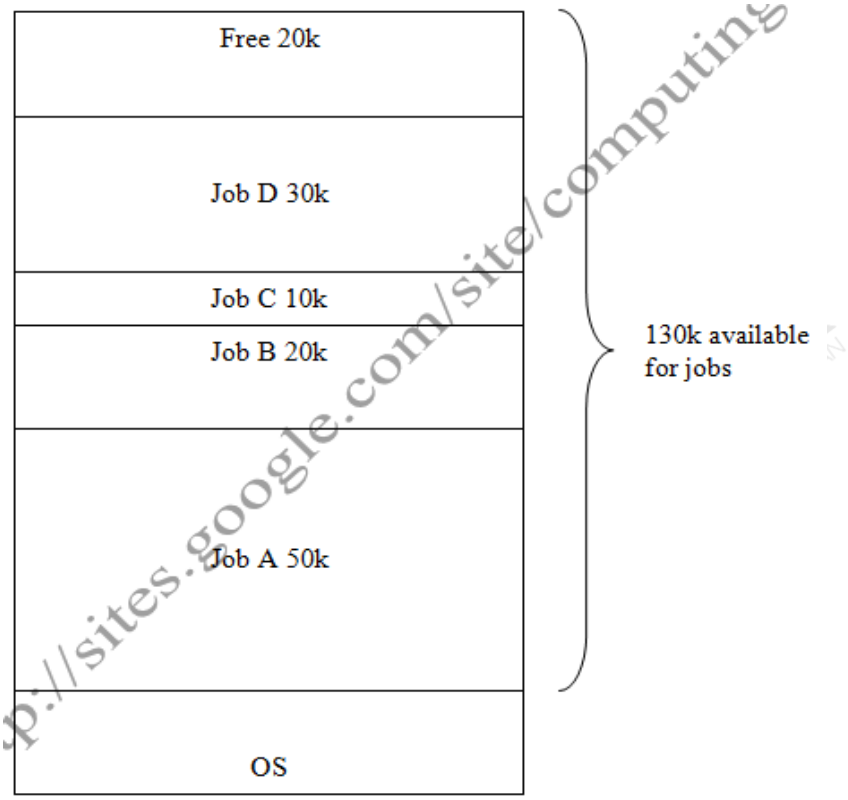
In this scheme, any job can only be given use of the processor if all the jobs at higher levels have been completed. Also, if a job enters a queue that has a higher priority than the queue from which the running program has come, the running program is placed back in the queue from which it came and the job that has entered the higher priority queue is placed in the running state.



3.4.1 Purposes of an operating system

Memory Management

This section can become very complex. In an examination, the questions will be limited to basic definitions and explanations. Calculations of the addresses and other details will not be required; however, they are included here for the completeness of the topic for those students who wish to understand in more detail.



In order for a job to be able to use the processor, the job must be stored in the computer's main memory. If there are several jobs to be stored, they, and their data, must be protected from the actions of other jobs.

Suppose jobs A, B, C, and D require 50k, 20k, 10k and 30 k of memory respectively and the computer has a total of 130 k available for jobs. (Remember the OS will require some memory.) The figure below shows one possible arrangement of the jobs.

When jobs are loaded into memory, they may not always occupy the same locations. Supposing, instead of jobs A,B,C, and D being needed and loaded in that order, it is required to load jobs A,B,D, and E in that order. Now job D occupies different locations in memory to those shown above. So again there is a problem of using different addresses.

The OS has a task of both loading the jobs and adjusting the addresses. The part of the OS which carries out these tasks is a program called the **loader**. The calculation of addresses can be done by recalculating each address used in the instructions once the address of the first instruction is known. Alternatively, relative addressing can be used. That is, addresses are specified relative to the first instruction.

3.4.1 Purposes of an operating system

Another problem to be solved is **when to move jobs.** Possible solutions are:

- Whenever a job terminates.
- When a new job is too large for any existing space.
- At regular intervals.
- When the user decides.

This system is known as **variable partitioning with compaction.** Imagine that each job needs a space to fit into, this space is the **partition**. Each of the jobs requires a different size of space, hence "variable partitions". These variable partitions are normally called **segments** and the method of dividing memory up is called **segmentation.** We also say that sometimes it is necessary to move jobs around so that they fill the "holes" left by jobs that leave, this is called "**compaction".**

An alternative method to divide both the memory and the jobs into fixed size units called "pages". As an example, suppose jobs A, B, C, D and E consist of 6,4,1,3 and 2 pages respectively. Also suppose that the available memory for jobs consists of 12 pages and jobs A, B and C have been loaded into memory as shown below.

Job A
Page 6
Page 5
Page 4
Page 3
Page 2
Page 1

Job B
Page 4
Page 3
Page 2
Page 1

Job C
Page 1

Memory
Free
C1
B4
B3
B2
B1
A6
A5
A4
A3
A2
A1

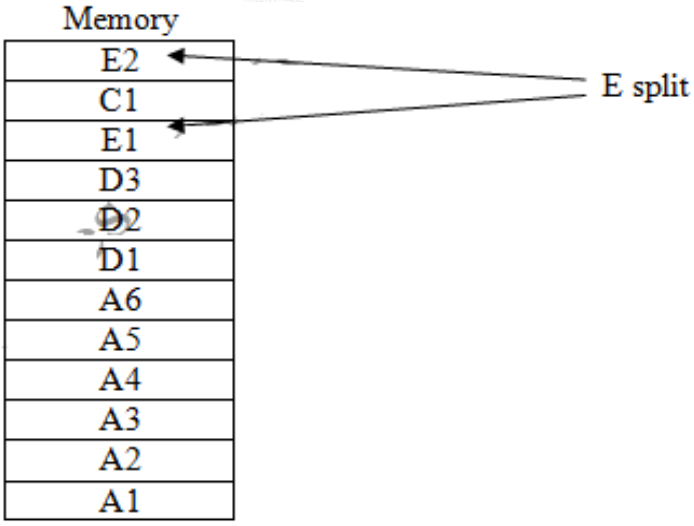
3.4.1 Purposes of an operating system

Now suppose job B terminates, releasing 4 pages, and jobs D and E are ready to be loaded. Clearly we have a similar problem to that caused by segmentation. The 'hole' consists of 4 pages into which job D (3 pages) will fit, leaving a single page plus the original one page free of memory. Job E consists of two pages, so there is enough memory for it but the pages are not contiguous, in other words, they are not joined together and we have the situation shown below.

Job E
Page 2
Page 1

Memory
Free
C1
Free
D3
D2
D1
A6
A5
A4
A3
A2
A1

The big difference between partitioning and paging is that jobs do not have to occupy contiguous pages. Thus the solution is shown.



The problem with paging is again address allocation. This can be overcome by keeping a table that shows which memory pages are used for the job pages. Then, if each address used in a job consists

of a page number and the distance required is from the start to the end of the page, a suitable conversion is possible.

Suppose, in job A, an instruction refers to a location that is on page 5 and is 46 locations from the start of page 5. This may be represented by:

5	46

Now suppose we have the following table

Job Page	Memory Page
A1	4
A2	5
A3	6
A4	7
A5	8
A6	9

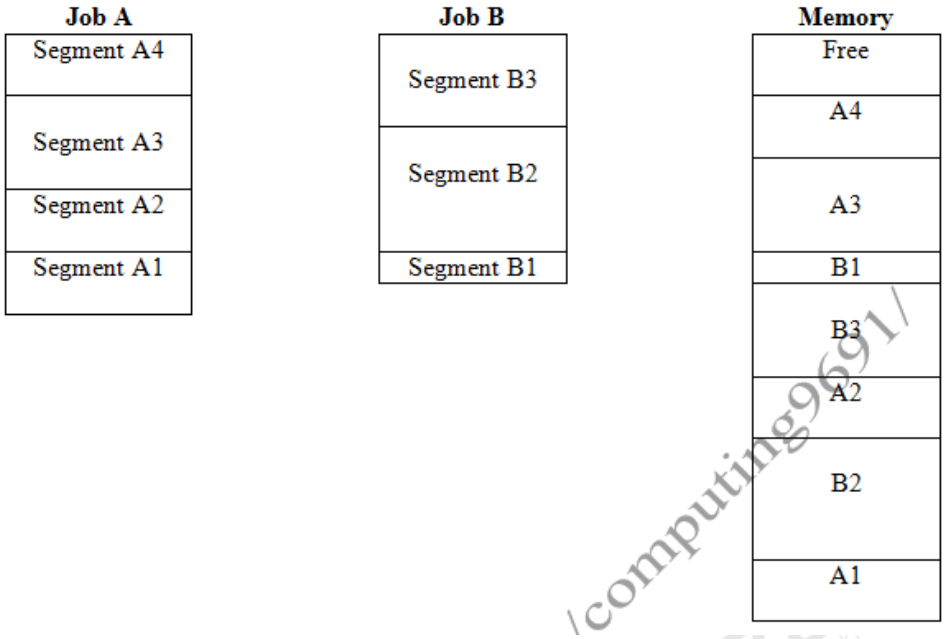
We see that page A5 is stored in page 8 of memory, thus

5	46	Becomes	8	46

Paging uses fixed length block of memory. An alternative is to use variable length blocks. This method is called **segmentation.** In segmentation, programmers divide jobs into segments, possibly of different sizes. Usually, the segments would consist of data, or sub-routines or groups of related sub-routines.

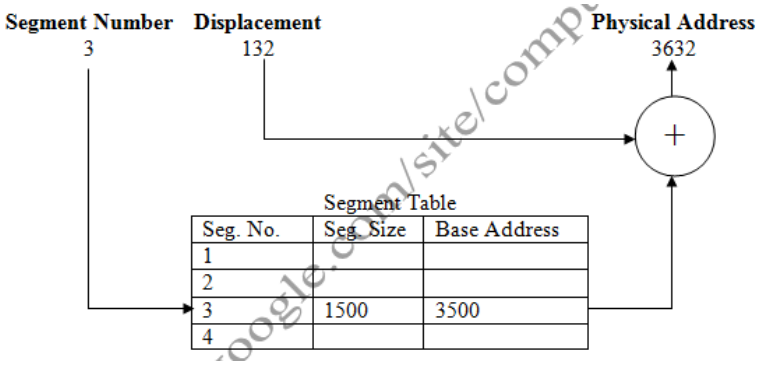
Since segments may be of different length, address calculation has to be carefully checked. The segment table must not only contain the start position of each segment but also the size of each segment. This is needed to ensure that an address does not go out of range. The figure below shows how 2 jobs may be stored in memory. In this case, the programmer slit Job A into 4 segments and Job B into 3 segments. These 2 jobs, when loaded into memory, took up the positions shown in the diagram.

3.4.1 Purposes of an operating system



Now suppose that an instruction specifies an address as segment 3, displacement (from start of segment) 132. The OS will look up, in the process segment table, the basic address (in memory) of segment 3. The OS checks that the displacement is not greater than the segment size. **If it is, an error is reported**. Otherwise the displacement is added to the base address to produce the actual address in memory to be used. The algorithm for this process is:

1. Get segment number.
2. Get displacement.
3. Use segment number to find length of segment from segment table.
4. If displacement is greater than segment size,
    a. Produce an error message
    b. Stop
5. Use segment number to find base address of segment from the segment table.
6. Add displacement to base address to produce physical address.



3.4.1 Purposes of an operating system

Paging and segmentation lead to another important technique called “virtual memory”. We have seen that jobs can be loaded into memory when they are needed using a paging technique. When a program is running, only those pages that contain the code that is needed are loaded. For example, suppose a word processor has been written with page 1 containing the instructions to allow users to enter text and to alter the text. Also suppose that page 2 contains the code for formatting characters, page 3 contains the code for formatting paragraphs and page 4 contains the code for cutting, copying and pasting. To run this word processor, only page 1 needs to be loaded initially. If the user wanted to format some characters so that they are in bold, then page 2 will have to be loaded. When other facilities are needed, the appropriate page can be loaded. When other facilities are needed, the appropriate page can be loaded. If the user now wants to format some more characters, the OS does not need to load page 2 again because it is already loaded.

Now, what happens if there is insufficient space for the new page to be loaded? As only the page containing active instructions need to be loaded, the new page can overwrite a page that is not currently in use. For example, suppose the user wishes to use paragraph formatting; then the OS can load page 3 into the memory currently occupied by page 2. Clearly, this means that programs can be written and used that are larger than the available memory.

There must be some system that decides which pages to overwrite. There are many systems such as **overwrite the page that has not been used for the longest period of time. Replace the page that has not recently been used** or the **first in first out method**. All of these create extra work for the OS.

To further complicate matters, not every page can be overwritten. Some pages contain a job’s data that will change during the running of a program. To keep track of this, the OS keeps a flag for each page that can be initially set to zero. If the content of the page changes, the flag can be set to 1. Now, before overwriting a page, the OS can see if that page has been changed. If it is, then the OS will save the page before loading a new page in its place. The OS now has to load and save pages. If the memory is very full, this loading and saving can use up a great deal of time and can mean that most of the processor’s time is involved in swapping pages. This situation is called **disk threshing.**

Systems can use both multi-programming and virtual memory. Also, virtual memory can use segmentation as well as paging although this technique may become very complex.