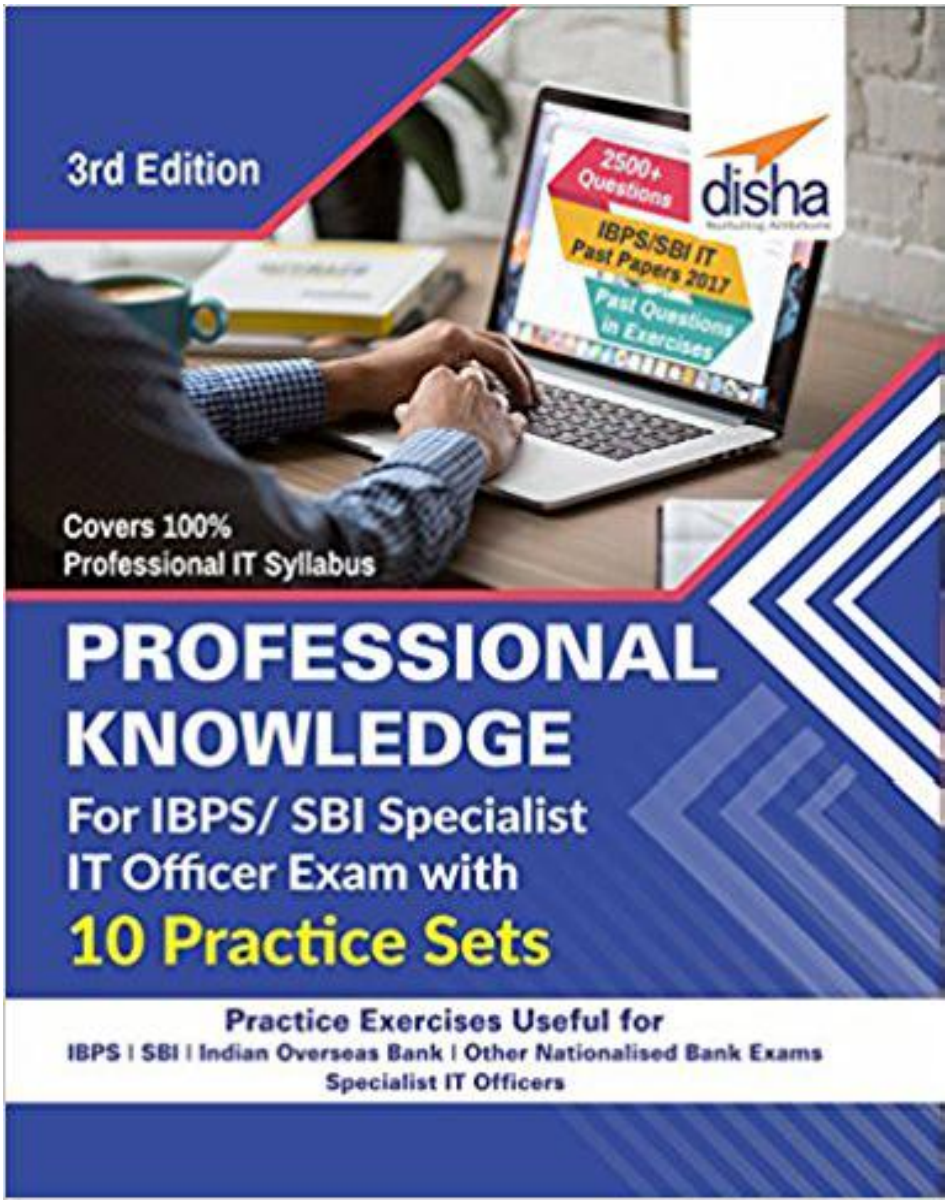
Operating System

This chapter "Operating System" **is taken from our book:**



ISBN : 9789386320971

CHAPTER 6

OPERATING SYSTEM

OPERATING SYSTEM

A program that acts as an intermediary between a user of a computer and the computer hardware is called an operating system. The operating system is an essential component of the system software in a computer system. Application programs usually require an operating system to function.

Operating systems can be found on almost any device that contains a computer—from cellular phones and video game consoles to supercomputers and web servers. For example: UNIX, MS-DOS,WINDOWS, 98/2000/xp/7.

FUNCTIONS OF AN OPERATING SYSTEM

The basic functions of an operating system are:

I. **Booting the computer :** The process of starting or restarting the computer is known as booting. A cold boot is when you turn on a computer that has been turned off completely. A warm boot is the process of using the operating system to restart the computer.

II. **Performs basic computer tasks :** The operating system performs basic computer tasks, such as managing the various peripheral devices such as the mouse, keyboard and printers. For example, most operating systems now are plug and play which means a device such as a printer will automatically be detected and configured without any user intervention.

III. **Provides a user interface :** A user interacts with software through the user interface. The two main types of user interfaces are: command line and a graphical user interface (GUI). With a command line interface, the user interacts with the operating system by typing commands to perform specific tasks. An example of a command line interface is DOS (disk operating system). With a graphical user interface, the user interacts with the operating system by using a mouse to access windows, icons, and menus. An example of a graphical user interface is Windows Vista or Windows 7.

IV. **Handles system resources :** The operating system also handles system resources such as the computer's memory and sharing of the central processing unit (CPU) time by various applications or peripheral devices. Programs and input methods are constantly competing for the attention of the CPU and demand memory, storage and input/output bandwidth. The operating system ensures that each application gets the necessary resources it needs in order to maximise the functionality of the overall system.

V. **Provides file management :** The operating system also handles the organisation and tracking of files and directories (folders) saved or retrieved from a computer disk. The file management system allows the user to perform such tasks as creating files and directories, renaming files, coping and moving files, and deleting files. The operating system keeps track of where files are located on the hard drive through the type of file system. The type two main types of file system are File Allocation table (FAT) or New Technology File system (NTFS).

File Allocation table (FAT) : It uses the file allocation table which records, which clusters are used and unused and where files are located within the clusters.

NTFS : It is a file system introduced by Microsoft and it has a number of advantages over the previous file system, named FAT32 (File Allocation Table).NTFS also allows permissions (such as read, write, and execute) to be set for individual directories and files.

The three most common operating systems for personal computers are Microsoft Windows, Mac OS X, and Linux.

TYPES OF OPERATING SYSTEM

There are different types of operating system to support the computer system. Each type of operating system offers distinct facilities that are appropriate to the computer system in which it is used.

The operating systems are of mainly following types:

Single-user, single task Operating System : This operating system is designed to manage the computer so that one user can effectively do one thing at a time. The Palm OS for Palm handheld computers is a good example of a modern single-user, single-task operating system.

Single-user, multi-tasking Operating System : This operating system mostly used by people which are using desktop and laptop computers today. Microsoft's Windows and Apple's MacOS platforms are both examples of operating systems that will let a single user have several programs in operation at the same time. For example, it's entirely possible for a Windows user to be writing a note in a word processor while downloading a file from the Internet while printing the text of an e-mail message.

Multi-user Operating System : A multi-user operating system allows many different users to take advantage of the computer's resources simultaneously. The operating system must make sure that the requirements of the various users are balanced, and that each of the programs they are using has sufficient and separate resources so that a problem with one user doesn't affect the entire community of users. Unix, VMS and mainframe operating systems, such as MVS, are examples of multi-user operating systems.

Real Time operating System : Real time operating system controls the environment as they have a data processing system in which

the time interval required to process and respond to inputs is very small. The time taken by the system to respond to an input and display the result of the required inputted information is termed as response time.A key characteristic of an RTOS is the level of its consistency concerning the amount of time it takes to accept and complete an application's task; the variability is jitter.A hard real-time operating system has less jitter than a soft real-time operating system. The chief design goal is not high throughput, but rather a guarantee of a soft or hard performance category. An RTOS that can usually or generally meet a deadline is a soft real-time OS, but if it can meet a deadline deterministically it is a hard real-time OS.

Time-sharing operating system : Time sharing is a type of operating system that enables many people, located at various terminals, to use a particular computer system at the same time. Because of the above feature timesharing operating system is called multitasking operating system. In other words it is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.The main purpose of Multiprogrammed batch systems, is to maximize processor use, whereas in Time-Sharing Systems the primary objective is to minimize response time.

Distributed operating System : Distributed systems use a number of central processors to serve multiple real time application and different users. Data processing jobs are distributed among the processors accordingly to which one can perform each job most efficiently.

Network operating System : This operating system runs on a server.This server is responsible for managing data, users, groups, security, applications, and other networking functions. The network operating system allows shared file and printer access among multiple computers in a network. This can also be referred as a local area network (LAN), or a private network. Examples of network operating systems include Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.

Stand-Alone Operating System : A stand-alone operating system is a complete operating system that works on a desktop computer, notebook computer, or mobile computing device. Some stand-alone operating systems are called client operating systems because they also work in conjunction with a server operating system. Client operating systems can operate with or without a network. Examples of currently used stand-alone operating systems are Microsoft Windows, Mac OS, and Linux.

(i) **Microsoft Windows :** Microsoft developed Windows operating system that has a graphical user interface. Some of the popular Windows Operating System are Windows 3.1, Windows 95, Windows 98, Windows 2000, Windows NT, Windows ME, Windows XP, Windows Vista and Windows 7. Windows 8 is the latest Windows operating systems from Microsoft. It is a graphical user interface (GUI) operating system which is very easy to learn and operate. Windows 7 provides many ways to manage the files stored on your computer. You can open, rename, print, delete, move and search for files.

(ii) **Mac OS :** It is a series of graphical user interface-based operating systems developed by Apple Inc. for their Macintosh line of computer systems. The Macintosh user experience is credited with popularizing the graphical user interface.

(iii) **Linux :** It is a UNIX-based operating system that is available for free on the World Wide Web. Many companies, such as Red Hat, Corel and Mandrake, create easy-to-use versions of Linux that you can purchase. Red Hat Linux is a popular version that comes with the GNOME desktop environment. GNOME displays pictures on the screen to help you perform tasks.

Linux is an open source code operating system. It can be copied, modified and redistributed with few restrictions. This flexibility is one of the reasons why Linux is so popular among users.

Embedded Operating System : An embedded system is a computer that is part of a different kind of machine. Examples include computers in cars, traffic lights, digital televisions, ATMs, airplane controls, point of sale (POS) terminals, digital cameras, GPS navigation systems, elevators, digital media receivers and smart meters, among many other possibilities.

In contrast to an operating system for a general-purpose computer, an embedded operating system is typically quite limited in terms of function – depending on the device in question, the system may only run a single application. However, that single application is crucial to the device's operation, so an embedded OS must be reliable and able to run with constraints on memory, size and processing power.

Thousands of connected embedded devices have been built on Windows Embedded platforms, from portable ultrasound machines to GPS devices and from ATMs to devices that power large construction machinery. With comprehensive features, easy-to-use and familiar Microsoft development tools, free evaluation kits and access to a large network of community support, working with Windows Embedded Products helps yield faster time to market for your devices and decreased development costs.

Mobile Operating System : A mobile operating system, also called a mobile OS, is an operating system that is specifically designed to run on mobile devices such as mobile phones, smartphones, PDAs, tablet computers and other handheld devices. The mobile operating system is the software platform on top of which other programs, called application programs, can run on mobile device.

Examples of mobile operating systems include Apple iOS, Windows Phone, and Google Android.

Operating systems for mobile devices generally aren't as fully featured as those made for desktop or laptop computers, and they aren't able to run all of the same software. However, you can still do a lot of things with them, like watch movies, browse the Web, manage your calendar, and play games.

The BIOS (basic input/output system) gets the computer system started after you turn it on and manages the data flow between the operating system and attached devices such as the hard disk, video adapter, keyboard, mouse, and printer.

An **assembler** takes basic computer instructions and converts them into a pattern of bits that the computer's processor can use to perform its basic operations.

A **device driver** controls a particular type of device that is attached to your computer, such as a keyboard or a mouse. The driver program converts the more general input/output instructions of the operating system to messages that the device type can understand.

Buffering is the pre-loading of data into a reserved area of memory (the buffer). In streaming audio or video from the Internet, buffering refers to downloading a certain amount of data before starting to play the music or movie. Having an advance supply of audio

samples or video frames in memory at all times prevents disruption if there are momentary delays in transmission while the material is being played. Even a live broadcast would have a few seconds of delay built in.

Spooling is the overlapping of low-speed operations with normal processing. Spooling originated with mainframes in order to optimise slow operations such as reading cards and printing. Card input was read onto disk and printer output was stored on disk. In that way, the business data processing was performed at high speed, receiving input from disk and sending output to disk. Subsequently, spooling is used to buffer data for the printer as well as remote batch terminals.

GUI (GRAPHICAL USER INTERFACE)

GUI is a program interface that takes advantage of the computer's graphics capabilities to make the program easier to use.

BASIC COMPONENTS OF A GUI

Graphical user interfaces, such as Microsoft Windows and the one used by the Apple Macintosh, feature the following basic components:

Pointer : A symbol that appears on the display screen and that you move to select objects and commands. Usually, the pointer appears as a small angled arrow. Text -processing applications, however, use an I-beam pointer that is shaped like a capital I.

Pointing device : A device, such as a mouse or trackball, that enables you to select objects on the display screen.

Icons : Small pictures that represent commands, files, or windows. By moving the pointer to the icon and pressing a mouse button, you can execute a command or convert the icon into a window. You can also move the icons around the display screen as if they were real objects on your desk.

Desktop : The area on the display screen where icons are grouped is often referred to as the desktop because the icons are intended to represent real objects on a real desktop.

Windows : You can divide the screen into different areas. In each window, you can run a different program or display a different file. You can move windows around the display screen, and change their shape and size at will.

Menus : Most graphical user interfaces let you execute commands by selecting a choice from a menu.

MS DOS (MICROSOFT DISK OPERATING SYSTEM)

Short for Microsoft Disk operating system, MS-DOS is a non-graphical command line operating system derived from 86-DOS that was created for IBM compatible computers. MS-DOS originally written by Tim Paterson and introduced by Microsoft in August 1981 and was last updated in 1994 when MS-DOS 6.22 was released. MS-DOS allows the user to navigate, open, and otherwise manipulate files on their computer from a command line instead of a GUI like Window.

DOS has a character user interface (CUI) i.e : Communication between a computer and the user can be done by using characters. In Dos, one has to key in the commands on the prompt. Prompt is a place where commands are issued. It may look like C:\> or C:\windows\>

PROCESS

A program in the execution is called a Process. Process is not the same as program. A process is more than a program code. A process is an 'active' entity as opposed to program which is considered to be a 'passive' entity. Attributes held by process include hardware state, memory, CPU etc.

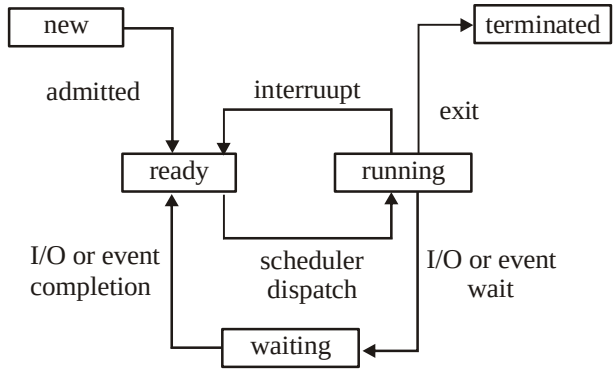
Process memory is divided into four sections for efficient working :

- The text section is made up of the compiled program code, read in from non-volatile storage when the program is launched.
- The data section is made up the global and static variables, allocated and initialized prior to executing the main.
- The heap is used for the dynamic memory allocation, and is managed via calls to new, delete, malloc, free, etc.
- The stack is used for local variables. Space on the stack is reserved for local variables when they are declared.

PROCESS STATE

Processes can be any of the following states :

- New - The process is in the stage of being created.
- Ready - The process has all the resources available that it needs to run, but the CPU is not currently working on this process's instructions.
- Running - The CPU is working on this process's instructions.
- Waiting - The process cannot run at the moment, because it is waiting for some resource to become available or for some event to occur.
- Terminated - The process has completed.



PROCESS CONTROL BLOCK

There is a Process Control Block for each process, enclosing all the information about the process. It is a data structure, which contains the following :

- Process State - It can be running, waiting etc.
- Process ID and parent process ID.
- CPU registers and Program Counter. Program Counter holds the address of the next instruction to be executed for that process.
- CPU Scheduling information - Such as priority information and pointers to scheduling queues.
- Memory Management information - Eg. page tables or segment tables.

- Accounting information - user and kernel CPU time consumed, account numbers, limits, etc.
- I/O Status information - Devices allocated, open file tables, etc.

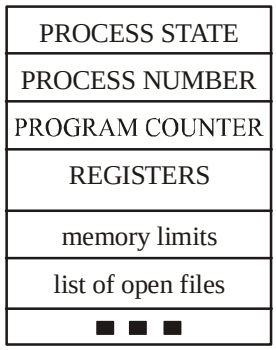
PROCESS SCHEDULING

The act of determining which process in the ready state should be moved to the running state is known as Process Scheduling.

The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs. For achieving this, the scheduler must apply appropriate rules for swapping processes IN and OUT of CPU.

Schedulers fell into one of the two general categories :

- Non pre-emptive scheduling. When the currently executing process gives up the CPU voluntarily.
- Pre-emptive scheduling. When the operating system decides to favour another process, pre-empting the currently executing process.



SCHEDULING QUEUES

- All processes when enters into the system are stored in the job queue.
- Processes in the Ready state are placed in the ready queue.
- Processes waiting for a device to become available are placed in device queues. There are unique device queues for each I/O device available.

Types of Schedulers

There are three types of schedulers available :

1. **Long Term Scheduler :** Long term scheduler runs less frequently. Long Term Schedulers decide which program must get into the job queue. From the job queue, the Job Processor, selects processes and loads them into the memory for execution. Primary aim of the Job Scheduler is to maintain a good degree of Multiprogramming. An optimal degree of Multiprogramming means the average rate of process creation is equal to the average departure rate of processes from the execution memory.
2. **Short Term Scheduler :** This is also known as CPU Scheduler and runs very frequently. The primary aim of this scheduler is to enhance CPU performance and increase process execution rate.
3. **Medium Term Scheduler :** During extra load, this scheduler picks out big processes from the ready queue for some time, to allow smaller processes to execute, thereby reducing the number of processes in the ready queue.

PROCESS CREATION

Through appropriate system calls, such as fork or spawn, processes may create other processes. The process which creates other process, is termed the parent of the other process, while the created sub-process is termed its child.

Each process is given an integer identifier, termed as process identifier, or PID. The parent PID (PPID) is also stored for each process.

CPU SCHEDULING

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair.

SCHEDULING CRITERIA

There are many different criterias to check when considering the "best" scheduling algorithm :

- **CPU utilization :** To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time(Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)
- **Throughput :** It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.
- **Turnaround time :** It is the amount of time taken to execute a particular process, i.e. The interval from time of submission of the process to the time of completion of the process(Wall clock time).
- **Waiting time :** The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.
- **Load average :** It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.
- **Response time :** Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution(final response).

In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

SCHEDULING ALGORITHMS

We'll discuss four major scheduling algorithms here which are following :

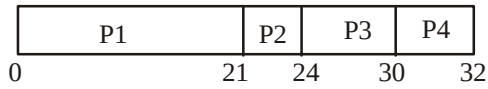
1. First Come First Serve(FCFS) Scheduling
2. Shortest-Job-First(SJF) Scheduling
3. Priority Scheduling
4. Round Robin(RR) Scheduling
5. Multilevel Queue Scheduling

First Come First Serve(FCFS) Scheduling

- Jobs are executed on first come, first serve basis.
- Easy to understand and implement.
- Poor in performance as average wait time is high.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The average waiting time will be = (0 +21 +24 +30) /4 = 18.75 ms



This is the GANITT chart for the above processes

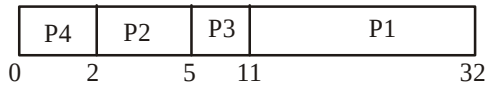
Shortest-Job-First(SJF) Scheduling

- Best approach to minimize waiting time.
- Actual time taken by the process is already known to processor.
- Impossible to implement.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

In Shortest Job First Scheduling. The Shortest process is executed First.

Hence the GANTT chart will be following :

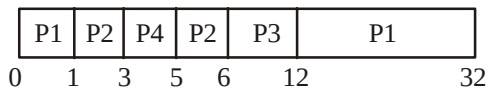


Now, the average waiting time will be = (0+2+5+11)/4 = 4.5 ms

In Preemptive Shortest Job First Scheduling, jobs are put into ready queue as they arrive, but as a process with short burst time arrives, the existing process is preemptied.

PROCESS	BURST TIME	ARRIVAL TIME
P1	21	0
P2	3	1
P3	6	2
P4	2	3

The GANTT chart for Preemptive Shortest Job First Scheduling will be.



The average waiting time will be. ((5-3) + (12-1))/4 = 4.25 ms

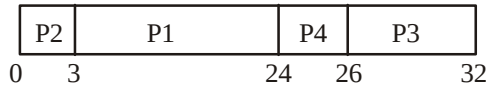
The average waiting time for preemptive shortest job first sceduling is less than both, non-preemptive SJF scheduling and FCFS scheduling.

Priority Scheduling

- Priority is assigned for each process.
- Process with highest priority is executed first and so on.
- Processes with same priority are executed in FCFS manner.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

PROCESS	BURST TIME	PRIORITY
P1	21	2
P2	3	1
P3	6	4
P4	2	3

The GANTT chart for following process based on Priority scheduling will be.



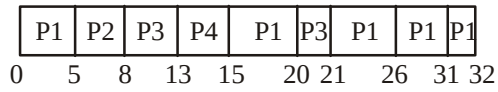
The average waiting time will be. (0 +3 +24 + 26)/4 = 13.25 ms

Round Robin(RR) Scheduling

- A fixed time is allotted to each process, called quantum, for execution.
- Once a process is executed for given time period that process is preemptied and other process executes for given time period.
- Context switching is used to save states of preemptied processes.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The GANTT chart for round robin scheduling will be.



The average waiting time will be, 11 ms.

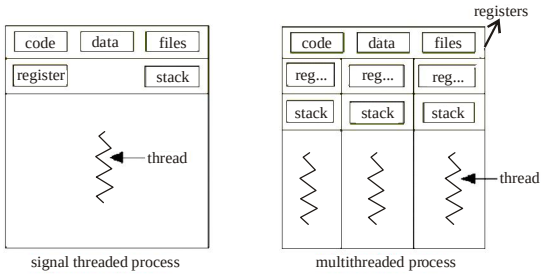
Multilevel Queue Scheduling

- Multiple queues are maintained for processes.
- Each queue can have its own scheduling algorithms.
- Priorities are assigned to each queue.

THREADS

Thread is an execution unit which consists of its own program counter, a stack, and a set of registers. Threads are also known as Lightweight processes. Threads are popular way to improve application through parallelism. The CPU switches rapidly back and forth among the threads giving illusion that the threads are running in parallel.

As each thread has its own independent resource for process execution, multpile processes can be executed parallely by increasing number of threads.



TYPES OF THREAD

There are two types of threads :

- User Threads
- Kernel Threads

User threads, are above the kernel and without kernel support. These are the threads that application programmers use in their programs.

Kernel threads are supported within the kernel of the OS itself. All modern OSs support kernel level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.

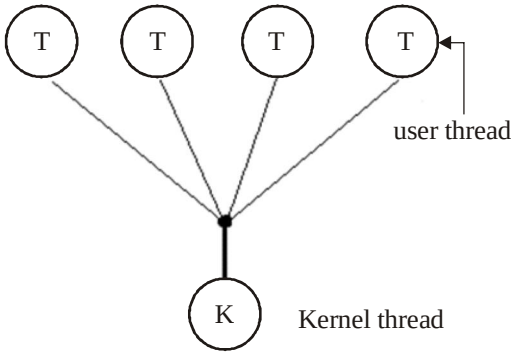
MULTITHREADING MODELS

The user threads must be mapped to kernel threads, by one of the following strategies.

- Many-To-One Model
- One-To-One Model
- Many-To-Many Model

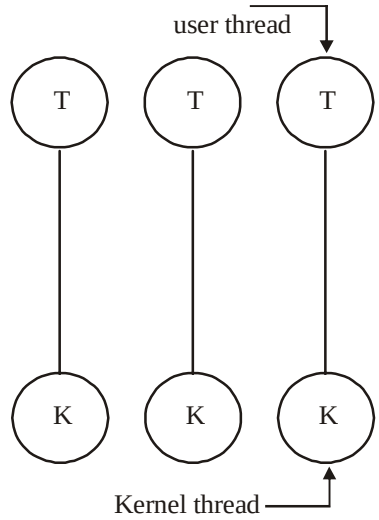
MANY-TO-ONE MODEL

- In the many-to-one model, many user-level threads are all mapped onto a single kernel thread.
- Thread management is handled by the thread library in user space, which is efficient in nature.



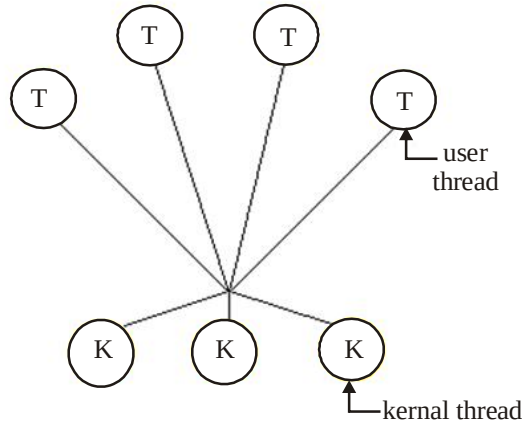
ONE-TO-ONE MODEL

- The one-to-one model creates a separate kernel thread to handle each and every user thread.
- Most implementations of this model place a limit on how many threads can be created.
- Linux and Windows from 95 to XP implement the one-to-one model for threads.



MANY-TO-MANY MODEL

- The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads, combining the best features of the one-to-one and many-to-one models.
- Users can create any number of the threads.
- Blocking the kernel system calls does not block the entire process.
- Processes can be split across multiple processors.



BENEFITS OF MULTITHREADING

1. Responsiveness
2. Resource sharing, hence allowing better utilization of resources.
3. Economy. Creating and managing threads becomes easier.
4. Scalability. One thread runs on one CPU. In Multithreaded processes, threads can be distributed over a series of processors to scale.
5. Context Switching is smooth. Context switching refers to the procedure followed by CPU to change from one task to another.

MULTITHREADING ISSUES

1. **Thread Cancellation :** Thread cancellation means terminating a thread before it has finished working. There can be two approaches for this, one is Asynchronous cancellation, which terminates the target thread immediately. The other is Deferred cancellation allows the target thread to periodically check if it should be cancelled.
2. **Signal Handling :** Signals are used in UNIX systems to notify a process that a particular event has occurred. Now in when a Multithreaded process receives a signal, to which thread it must be delivered. It can be delivered to all, or a single thread.
3. **fork() System Call :** fork() is a system call executed in the kernel through which a process creates a copy of itself. Now the problem in Multithreaded process is, if one thread forks, will the entire process be copied or not
4. Security Issues because of extensive sharing of resources between multiple threads.

There are many other issues that you might face in a multithreaded process, but there are appropriate solutions available for them. Pointing out some issues here was just to study both sides of the coin.

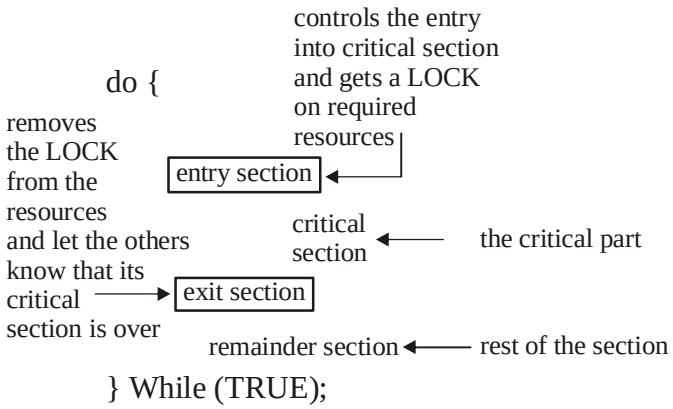
PROCESS SYNCHRONIZATION

Process Synchronization means sharing system resources by processes in a such a way that, Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes.

Process Synchronization was introduced to handle problems that arose while multiple process executions. Some of the problems are discussed below.

CRITICAL SECTION PROBLEM

A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes.



SOLUTION TO CRITICAL SECTION PROBLEM

A solution to the critical section problem must satisfy the following three conditions :

1. **Mutual Exclusion :** Out of a group of cooperating processes, only one process can be in its critical section at a given point of time.
2. **Progress :** If no process is in its critical section, and if one or more threads want to execute their critical section then any one of these threads must be allowed to get into its critical section.
3. **Bounded Waiting :** After a process makes a request for getting into its critical section, there is a limit for how many other processes can get into their critical section, before this process's request is granted. So after the limit is reached, system must grant the process permission to get into its critical section.

SYNCHRONIZATION HARDWARE

Many systems provide hardware support for critical section code. The critical section problem could be solved easily in a single-processor environment if we could disallow interrupts to occur while a shared variable or resource is being modified.

In this manner, we could be sure that the current sequence of instructions would be allowed to execute in order without pre-emption. Unfortunately, this solution is not feasible in a multiprocessor environment.

Disabling interrupt on a multiprocessor environment can be time consuming as the message is passed to all the processors.

This message transmission lag, delays entry of threads into critical section and the system efficiency decreases.

MUTEX LOCKS

As the synchronization hardware solution is not easy to implement fro everyone, a strict software approach called Mutex Locks was introduced. In this approach, in the entry section of code, a LOCK is acquired over the critical resources modified and used inside critical section, and in the exit section that LOCK is released.

As the resource is locked while a process executes its critical section hence no other process can access it.

SEMAPHORES

In 1965, Dijkstra proposed a new and very significant technique for managing concurrent processes by using the value of a simple integer variable to synchronize the progress of interacting processes. This integer variable is called semaphore. So it is basically a synchronizing tool and is accessed only through two low standard atomic operations, wait and signal designated by P() and V() respectively.

The classical definition of wait and signal are :

- Wait : decrement the value of its argument S as soon as it would become non-negative.
- Signal : increment the value of its argument, S as an individual operation.

PROPERTIES OF SEMAPHORES

1. Works with many processes
2. Can have many different critical sections with different semaphores

3. Each critical section has unique access semaphores
4. Can permit multiple processes into the critical section at once, if desirable.

TYPES OF SEMAPHORES

Semaphores are mainly of two types:

1. **Binary Semaphore :** It is a special form of semaphore used for implementing mutual exclusion, hence it is often called Mutex.
   (a) Binary semaphores have 2 methods associated with it. (up, down / lock, unlock)
   (b) Binary semaphores can take only 2 values (0/1). It is initialized to one. They are used to acquire locks.
   (c) When a resource is available, the process in charge set the semaphore to 1 else 0.
2. **Counting Semaphores :** These are used to implement bounded concurrency. Counting Semaphore may have value to be greater than one, typically used to allocate resources from a pool of identical resources

LIMITATIONS OF SEMAPHORES

1. Priority Inversion is a big limitation os semaphores.
2. Their use is not enforced, but is by convention only.
3. With improper use, a process may block indefinitely. Such a situation is called Deadlock.

CLASSICAL PROBLEM OF SYNCHRONIZATION

Following are some of the classical problem faced while process synchronaization in systems where cooperating processes are present.

(a) Bounded Buffer Problem

- This problem is generalised in terms of the Producer-Consumer problem.
- Solution to this problem is, creating two counting semaphores "full" and "empty" to keep track of the current number of full and empty buffers respectively.

(b) The Readers Writers Problem

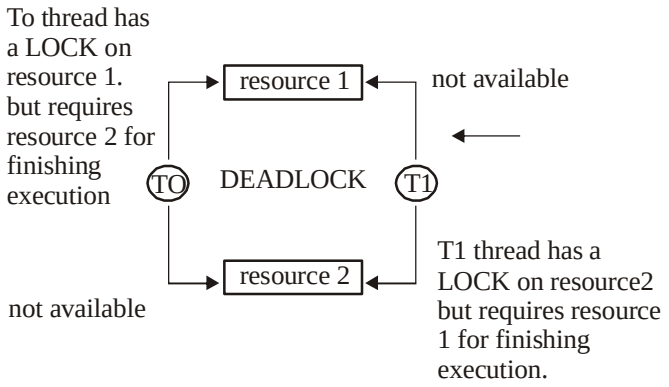
- In this problem there are some processes(called readers) that only read the shared data, and never change it, and there are other processes(called writers) who may change the data in addition to reading or instead of reading it.
- There are various type of the readers-writers problem, most centred on relative priorities of readers and writers

(c) Dining Philosophers Problem

- The dining philosopher's problem involves the allocation of limited resources from a group of processes in a deadlock-free and starvation-free manner.
- There are five philosophers sitting around a table, in which there are five chopsticks kept beside them and a bowl of rice in the centre, When a philosopher wants to eat, he uses two chopsticks - one from their left and one from their right. When a philosopher wants to think, he keeps down both chopsticks at their original place.

DEADLOCK

Deadlocks are a set of blocked processes each holding a resource and waiting to acquire a resource held by another process.



HOW TO AVOID DEADLOCKS

Deadlocks can be avoided by avoiding at least one of the four conditions, because all this four conditions are required simultaneously to cause deadlock.

1. **Mutual Exclusion :** Resources shared such as read-only files do not lead to deadlocks but resources, such as printers and tape drives, requires exclusive access by a single process.
2. **Hold and Wait :** In this condition processes must be prevented from holding one or more resources while simultaneously waiting for one or more others.
3. **No Preemption :** Preemption of process resource allocations can avoid the condition of deadlocks, where ever possible.
4. **Circular Wait :** Circular wait can be avoided if we number all resources, and require that processes request resources only in strictly increasing(or decreasing) order.

HANDLING DEADLOCK

The above points focus on preventing deadlocks. But what to do once a deadlock has occured. Following three strategies can be used to remove deadlock after its occurrence.

1. **Preemption :** We can take a resource from one process and give it to other. This will resolve the deadlock situation, but sometimes it does causes problems.
2. **Rollback :** In situations where deadlock is a real possibility, the system can periodically make a record of the state of each process and when deadlock occurs, roll everything back to the last checkpoint, and restart, but allocating resources differently so that deadlock does not occur.
3. **Kill one or more processes :** This is the simplest way, but it works.

LIVELOCK

There is a variant of deadlock called livelock. This is a situation in which two or more processes continuously change their state in response to changes in the other process(es) without doing any useful work. This is similar to deadlock in that no progress is made but differs in that neither process is blocked or waiting for anything.

A human example of livelock would be two people who meet face-to-face in a corridor and each moves aside to let the other pass, but they end up swaying from side to side without making any progress because they always move the same way at the same time.

MEMORY MANAGEMENT

Main Memory refers to a physical memory that is the internal memory to the computer. The word main is used to distinguish it from external mass storage devices such as disk drives. Main memory is also known as RAM. The computer is able to change only data that is in main memory. Therefore, every program we execute and every file we access must be copied from a storage device into main memory.

All the programs are loaded in the main memeory for execution. Sometimes complete program is loaded into the memory, but some times a certain part or routine of the program is loaded into the main memory only when it is called by the program, this mechanism is called Dynamic Loading, this enhance the performance.

Also, at times one program is dependent on some other program. In such a case, rather than loading all the dependent programs, CPU links the dependent programs to the main executing program when its required. This mechanism is known as Dynamic Linking.

SWAPPING

A process needs to be in memory for execution. But sometimes there is not enough main memory to hold all the currently active processes in a timesharing system. So, excess process are kept on disk and brought in to run dynamically. Swapping is the process of bringing in each process in main memory, running it for a while and then putting it back to the disk.

CONTIGUOUS MEMORY ALLOCATION

In contiguous memory allocation each process is contained in a single contiguous block of memory. Memory is divided into several fixed size partitions. Each partition contains exactly one process. When a partition is free, a process is selected from the input queue and loaded into it. The free blocks of memory are known as holes. The set of holes is searched to determine which hole is best to allocate.

MEMORY PROTECTION

Memory protection is a phenomenon by which we control memory access rights on a computer. The main aim of it is to prevent a process from accessing memory that has not been allocated to it. Hence prevents a bug within a process from affecting other processes, or the operating system itself, and instead results in a segmentation fault or storage violation exception being sent to the disturbing process, generally killing of process.

MEMORY ALLOCATION

Memory allocation is a process by which computer programs are assigned memory or space. It is of three types :

1. **First Fit :** The first hole that is big enough is allocated to program.
2. **Best Fit :** The smallest hole that is big enough is allocated to program.
3. **Worst Fit :** The largest hole that is big enough is allocated to program.

FRAGMENTATION

Fragmentation occurs in a dynamic memory allocation system when most of the free blocks are too small to satisfy any request. It is generally termed as inability to use the available memory.
In such situation processes are loaded and removed from the memory. As a result of this, free holes exists to satisfy a request but is non contiguous i.e. the memory is fragmented into large no. Of small holes. This phenomenon is known as External Fragmentation.
Also, at times the physical memory is broken into fixed size blocks and memory is allocated in unit of block sizes. The memory allocated to a space may be slightly larger than the requested memory. The difference between allocated and required memory is known as Internal fragmentation i.e. the memory that is internal to a partition but is of no use.

PAGING

A solution to fragmentation problem is Paging. Paging is a memory management mechanism that allows the physical address space of a process to be non-contagious. Here physical memory is divided into blocks of equal size called Pages. The pages belonging to a certain process are loaded into available memory frames.

PAGE TABLE

A Page Table is the data structure used by a virtual memory system in a computer operating system to store the mapping between virtual address and physical addresses.

Virtual address is also known as Logical address and is generated by the CPU. While Physical address is the address that actually exists on memory.

SEGMENTATION

Segmentation is another memory management scheme that supports the user-view of memory. Segmentation allows breaking of the virtual address space of a single process into segments that may be placed in non-contiguous areas of physical memory.

SEGMENTATION WITH PAGING

Both paging and segmentation have their advantages and disadvantages, it is better to combine these two schemes to improve on each. The combined scheme is known as 'Page the Elements'. Each segment in this scheme is divided into pages and each segment is maintained in a page table. So the logical address is divided into following 3 parts :

- Segment numbers(S)
- Page number (P)
- The displacement or offset number (D)

VIRTUAL MEMORY

Virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory. This technique is useful as large virtual memory is provided for user programs when a very small physical memory is there.

In real scenarios, most processes never need all their pages at once, for following reasons :

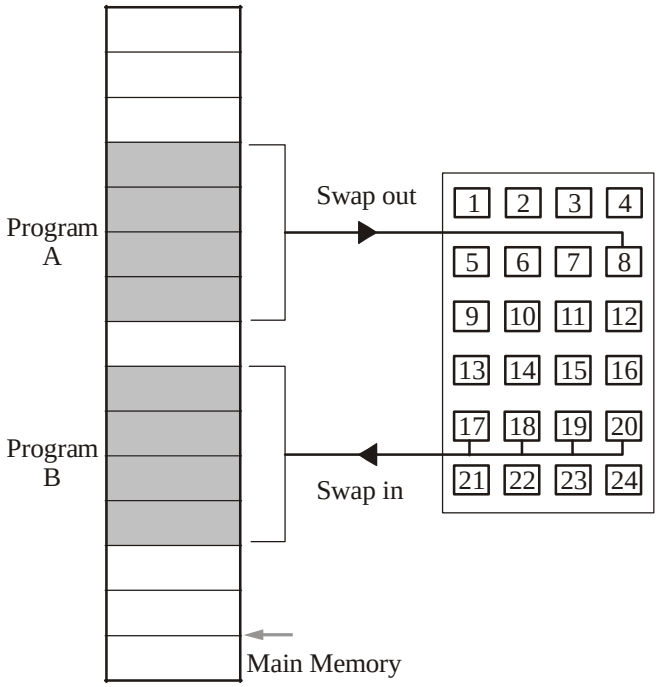
- Error handling code is not needed unless that specific error occurs, some of which are quite rare.
- Arrays are often over-sized for worst-case scenarios, and only a small fraction of the arrays are actually used in practice.
- Certain features of certain programs are rarely used.

BENEFITS OF HAVING VIRTUAL MEMORY :

1. Large programs can be written, as virtual space available is huge compared to physical memory.
2. Less I/O required, leads to faster and easy swapping of processes.
3. More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.

DEMAND PAGING

The basic idea behind demand paging is that when a process is swapped in, its pages are not swapped in all at once. Rather they are swapped in only when the process needs them(On demand). This is termed as lazy swapper, although a pager is a more accurate term.



Initially only those pages are loaded which will be required the process immediately.

The pages that are not moved into the memory, are marked as invalid in the page table. For an invalid entry the rest of the table is empty. In case of pages that are loaded in the memory, they are marked as valid along with the information about where to find the swapped out page.

When the process requires any of the page that is not loaded into the memory, a page fault trap is triggered and following steps are followed,

1. The memory address which is requested by the process is first checked, to verify the request made by the process.
2. If its found to be invalid, the process is terminated.
3. In case the request by the process is valid, a free frame is located, possibly from a free-frame list, where the required page will be moved.
4. A new operation is scheduled to move the necessary page from disk to the specified memory location. (This will usually block the process on an I/O wait, allowing some other process to use the CPU in the meantime.)
5. When the I/O operation is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to valid.
6. The instruction that caused the page fault must now be restarted from the beginning.

There are cases when no pages are loaded into the memory initially, pages are only loaded when demanded by the process by generating page faults. This is called Pure Demand Paging.

The only major issue with Demand Paging is, after a new page is loaded, the process starts execution from the beginning. Its is not a big issue for small programs, but for larger programs it affects performance drastically.

PAGE REPLACEMENT

As studied in Demand Paging, only certain pages of a process are loaded initially into the memory. This allows us to get more number of processes into the memory at the same time. but what happens when a process requests for more pages and no free memory is available to bring them in. Following steps can be taken to deal with this problem :

1. Put the process in the wait queue, until any other process finishes its execution thereby freeing frames.
2. Or, remove some other process completely from the memory to free frames.
3. Or, find some pages that are not being used right now, move them to the disk to get free frames. This technique is called Page replacement and is most commonly used. We have some great algorithms to carry on page replacement efficiently.

BASIC PAGE REPLACEMENT

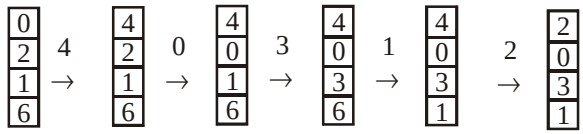
- Find the location of the page requested by ongoing process on the disk.
- Find a free frame. If there is a free frame, use it. If there is no free frame, use a page-replacement algorithm to select any existing frame to be replaced, such frame is known as victim frame.
- Write the victim frame to disk. Change all related page tables to indicate that this page is no longer in memory.
- Move the required page and store it in the frame. Adjust all related page and frame tables to indicate the change.
- Restart the process that was waiting for this page.

FIFO PAGE REPLACEMENT

- A very simple way of Page replacement is FIFO (First in First Out)
- As new pages are requested and are swapped in, they are added to tail of a queue and the page which is at the head becomes the victim.
- Its not an effective way of page replacement but can be used for small systems.
- In main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

Reference String: 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses: xxxxxxxx



Fault Rate = 9/12 = 0.75

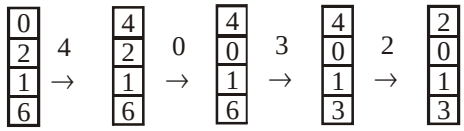
LRU PAGE REPLACEMENT

Below is a video, which will explain LRU Page replacement algorithm in details with an example.

- In main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

  Reference String: 0,2,1,6,4,0,1,0,3,1,2,1

  Misses : x xxxxxxxx

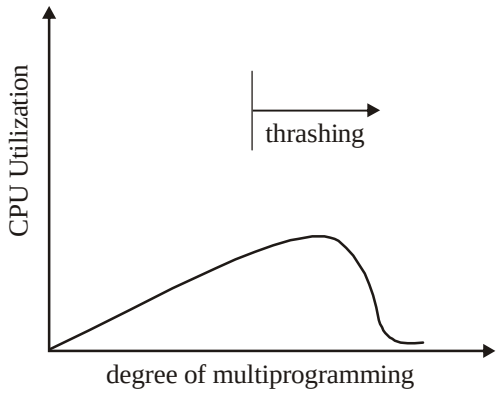


Fault Rate = 9/12 = 0.75

THRASHING

A process that is spending more time paging than executing is said to be thrashing. In other words it means, that the process doesn't have enough frames to hold all the pages for its execution, so it is swapping pages in and out very frequently to keep executing. Sometimes, the pages which will be required in the near future have to be swapped out.

Initially when the CPU utilization is low, the process scheduling mechanism, to increase the level of multiprogramming loads multiple processes into the memory at the same time, allocating a limited amount of frames to each process. As the memory fills up, process starts to spend a lot of time for the required pages to be swapped in, again leading to low CPU utilization because most of the proccesses are waiting for pages. Hence the scheduler loads more processes to increase CPU utilization, as this continues at a point of time the complete system comes to a stop.



To prevent thrashing we must provide processes with as many frames as they really need "right now".

FILE SYSTEM

A file can be "free formed", indexed or structured collection of related bytes having meaning only to the one who created it. Or in other words an entry in a directory is the file. The file may have attributes like name, creator, date, type, permissions etc.

FILE STRUCTURE

A file has various kinds of structure. Some of them can be :

- Simple Record Structure with lines of fixed or variable lengths.
- Complex Structures like formatted document or reloadable load files.
- No Definite Structure like sequence of words and bytes etc.

ATTRIBUTES OF A FILE

Following are some of the attributes of a file :

- **Name .** It is the only information which is in human-readable form.
- **Identifier.** The file is identified by a unique tag(number) within file system.
- **Type.** It is needed for systems that support different types of files.
- **Location.** Pointer to file location on device.
- **Size.** The current size of the file.
- **Protection.** This controls and assigns the power of reading, writing, executing.
- **Time, date, and user identification.** This is the data for protection, security, and usage monitoring.

FILE ACCESS METHODS

The way that files are accessed and read into memory is determined by Access methods. Usually a single access method is supported by systems while there are OS's that support multiple access methods.

Sequential Access

- Data is accessed one record right after another is an order.
- Read command cause a pointer to be moved ahead by one.
- Write command allocate space for the record and move the pointer to the new End Of File.
- Such a method is reasonable for tape.

Direct Access

- This method is useful for disks.
- The file is viewed as a numbered sequence of blocks or records.
- There are no restrictions on which blocks are read/written, it can be dobe in any order.
- User now says "read n" rather than "read next".
- "n" is a number relative to the beginning of file, not relative to an absolute physical disk location.

Indexed Sequential Access

- It is built on top of Sequential access.
- It uses an Index to control the pointer while accessing files.

DIRECTORY

Information about files is maintained by Directories. A directory can contain multiple files. It can even have directories inside of them. In Windows we also call these directories as folders.

Following is the information maintained in a directory :

- Name : The name visible to user.
- Type : Type of the directory.
- Location : Device and location on the device where the file header is located.
- Size : Number of bytes/words/blocks in the file.
- Position : Current next-read/next-write pointers.
- Protection : Access control on read/write/execute/delete.
- Usage : Time of creation, access, modification etc.
- Mounting : When the root of one file system is "grafted" into the existing tree of another file system its called Mounting.

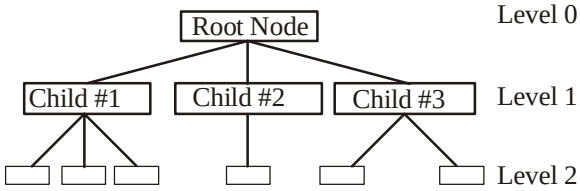
KEY CONCEPTS OF OPERATING SYSTEMS AND SOFTWARE

We will now review some key ideas behind the whole idea of operating systems. Although these ideas may seem simple, but it is important to remember that them.

HIERARCHIES AND BLACK BOXES

A hierarchy is a way of organizing information using levels of detail. The phrase high-level implies few details, whereas low-level implies a lot of detail, down in the heart of things. A hierarchy usually has the form of a tree, which branches from the highest level to the lowest, since each high-level object is composed of several lower-level objects. The key to making large computer programs and to solving difficult problems is to create a hierarchical structure, in which large high-level problems are gradually broken up into manageable low-level problems. Each level works by using a series of 'black boxes' (e.g. subroutines) whose inner details are not directly visible.

This important concept in computing is used repeatedly to organize complex problems.



RESOURCES AND SHARING

A computer is not just a box which adds numbers together. It has resources like the keyboard and the screen, the disk drives and the memory. In a multi-tasking system there may be several programs which need to receive input or write output simultaneously and thus the operating system may have to share these resources between several running programs. If the system has two or more keyboards (or terminals) connected to it, then the OS can allocate these to different programs. If only a single terminal is connected then competing programs must wait for the resources to become free.

Most multi-tasking systems have only a single central processing unit (CPU) but this is the most precious resource a computer has. A multi-tasking operating system must therefore share CPU time between programs. That is, it must work for a time on one program, then work sometime on the next program, and so on. If the first program is left unfinished, it must then return to work more on that, in a systematic way. The way an OS decides to share its time between different tasks is called scheduling.

COMMUNICATION, PROTOCOLS, DATA TYPES

The exchange of information is an essential part of computing. Suppose computer A sends a message to computer B reporting on the names of all the users and how long they have been working. To do this it sends a stream of bits across a network. When computer B receives a stream of bits, it doesn't automatically know what they mean. It must decide if the bits represent numbers or characters, integers or decimal point numbers, or a mixture of all of them. These different types of data are all stored as binary information - the only difference between them is the way one chooses to interpret them.

The resolution to this problem is to define a protocol. This is a convention or agreement between the operating systems of two machines on what messages may contain. The agreement may say, for instance, that the first thirty-two bits are four integers which give the address of the machine which sent the message. The next thirty-two bits are a special number telling the OS which protocol to use in order to interpret the data. The OS can then look up this protocol and discover that the rest of the data are arranged according to a pattern of

<name><time><name><time>...

where the name is a string of bytes, terminated by a zero, and the time is a four byte digit containing the time in hours. Computer B now knows enough to be able to extract the information from the stream of bits.

It is important to understand that all computers have to agree on the way in which the data are sent in advance. If the wrong protocol is diagnosed, then a string of characters could easily be converted into a floating point (or decimal) number - but the result would have been nonsense. Similarly, if computer A had sent the information incorrectly, computer B might not be able to read the data and a protocol error would arise.

More generally, a protocol is an agreed sequence of behaviour which must be followed.

For example, when passing parameters to functions in a computer program, there are rules about how the parameter should be declared and in which order they are sent. This is a simple example of a protocol. Protocols are an important part of communication and data classification and they appear in many forms.

SYSTEM OVERHEAD

An operating system is itself a computer program which must be executed. It therefore requires its own share of a computer's resources. This is especially true on multitasking systems, such as UNIX, where the OS is running all the time alongside users' programs. Since user programs have to wait for the OS to perform certain services, such as allocating resources, they are slowed down by the OS. The time spent by the OS servicing user requests

is called the system overhead. On a multi-user system one would like this overhead to be kept to a minimum, since programs which make many requests to the OS slow themselves down, as well as all other programs which are queuing up for resources.

CACHING

Caching is a technique used to speed up communication with slow devices. Usually the CPU can read data much faster from memory than it can from a disk or network connection, so it would like to keep an up-to-date copy of frequently used information in memory. The memory area used to do this is called a cache. You can think of the whole of the primary memory as being a cache for the secondary memory (disk).

Sometimes caching is used more generally to mean `keeping a local copy of data for convenience'.

THE CPU

The CPU, or central processing unit is the brain of every computer. This is the part which does the work of executing machine instructions. Traditionally, it is just one microprocessor with lots of pins to connect it to memory and devices - usually identifiable by being the largest chip. On modern machines, there may be several CPUs which can work in parallel. Also VLSI or Very Large Scale Integration technology has made it possible to put very many separate processors and memory into a single package, so the physical distinction between the CPU and its support chips is getting blurred. Nevertheless, the CPU is still logically separate from the memory and devices.

The CPU is driven by a `clock' or pulse generator. Each instruction is completed in a certain number of `clock cycles'. Traditionally CPUs are based on CISC (Complex Instruction Set Computing) architecture, where a single instruction takes one or more clock cycles to complete. A new trend is to build RISC (Reduced Instruction Set Computing) processors which aim to be more efficient for a subset of instructions by using redundancy. These have simpler instructions but can execute much more quickly, sometimes with several instructions per clock cycle.

MEMORY

The primary memory is the most important resource a computer has. Since CPUs are only made with instructions for reading and writing to memory, no programs would be able to run without it. There are two types of memory: RAM - Random Access Memory, or read/write memory, which loses its contents when the machine is switched off, and ROM - Read Only Memory, which never loses its contents unless destroyed. ROM is normally used for storing those most fundamental parts of the operating system which are required the instant a computer is switched on, before it knows about disks etc.

DEVICES

The concept of a device consists of two parts. There is the hardware unit which is connected to the machine, and there is the logical part which is a name given by the OS to a legal entry point for talking to a hardware-device. When a user writes to a logical device, the OS invokes a device driver which performs the physical operations of controlling the hardware. For example, when writing to a disk, the OS must control the movement of the read-write heads. When writing to a printer, the OS places the information in a queue and services the request when the printer becomes free.

Some common logical devices are: system disks, keyboard, screen, printer and audio devices.

Disks and tapes are often called secondary memory or secondary storage.

INTERRUPTS, TRAPS, EXCEPTIONS

Interrupts can be categorized into two groups which are asynchronous interrupts (aka interrupt, hardware interrupt) and synchronous interrupts (aka exception). The former may arrive anytime, typically IO interrupts, the latter may only arrive after the execution of an instruction, for example when the cpu try to divide a number by 0 or a page fault. So that's the difference between interrupts and exception.

A **trap** is a kind of exceptions, whose main purpose is for debugging (eg. notify the debugger that an instruction has been reached).

Interrupts are hardware signals which are sent to the CPU by the devices it is connected to. These signals literally interrupt the CPU from what it is doing and demand that it spend a few clock cycles servicing a request. For example, interrupts may come from the keyboard because a user pressed a key. Then the CPU must stop what it is doing and read the keyboard, place the key value into a buffer for later reading, and return to what it was doing. Other ‘events’ generate interrupts: the system clock sends interrupts at periodic intervals, disk devices generate interrupts when they have finished an I/O task and interrupts can be used to allow computers to monitor sensors and detectors. User programs can also generate `software interrupts' in order to handle special situations like a `division by zero' error. These are often called traps or exceptions on some systems.

Interrupts are graded by levels. Low level interrupts have a low priority, whereas high level interrupts have a high priority. A high level interrupt can interrupt a low level interrupt, so that the CPU must be able to recover from several `layers' of interruption and end up doing what it was originally doing. This is accomplished by means of a stack. Moreover, programs can often choose whether or not they wish to be interrupted by setting an interrupt mask which masks out the interrupts it does not want to know about. Masking interrupts can be dangerous, since data can be lost. All systems therefore have non-maskable interrupts for the most crucial operations.

Resource Management

In order to keep track of how the system resources are being used, an OS must keep tables or lists telling it what is free and what is not. For example, data cannot be stored neatly on a disk for all times. As files are deleted and some are modified, holes appear and the data becomes scattered randomly over the disk surface.

SPOOLING

Spooling is a way of processing data serially. Print jobs are spooled to the printer, because they must be printed in the right order (it would not help the user if the lines of his/her file were mixed together with parts of someone else's file). During a spooling operation, only one job is performed at a time and other jobs wait in a queue to be processed.

- Spooling is a form of batch processing.
- Spooling comes from the need to copy data onto a spool of tape for storage.

SYSTEM CALLS

An important task of an operating system is to provide black-box functions for the most frequently needed operations, so that users do not have to waste their time programming very low level code which is irrelevant to their purpose. These ready-made functions comprise frequently used code and are called system calls.

For example, controlling devices requires very careful and complex programming. Users should not have to write code to position the head of the disk drive at the right place just to save a file to the disk. This is a very basic operation which everyone requires and thus it becomes the responsibility of the OS. Another example is mathematical functions or graphics primitives.

System calls can be thought of as a very simple protocol - an agreed way of asking the OS to perform a service. Some typical OS calls are: read, write (to screen, disk, printer etc), stat (get the status of a file: its size and type) and malloc (request for memory allocation).

On older microcomputers, where high level languages were uncommon, system calls were often available only through assembler or machine code. On modern systems and integrated systems like UNIX, they are available as functions in a high level language like C.

BASIC COMMAND LANGUAGE

Commands like
dir ; list files (DOS)
ls ; list files (UNIX)
cd ; change directory
copy file prn ; copy file to printer
myprog ; execute program `myprog'

constitute a basic command language. Every computer must have such a language. In microcomputer operating systems the command language is often built into the system code, whereas on larger systems (UNIX) the commands are just executable programs like the last example above.

The command language deals typically with file management, process management and text editing.

FILESYSTEM

In creating a system to store files we must answer some basic questions.

- Should the file system distinguish between types of files e.g. executable files, text files, scripts, etc. If so, how? One way is to use file extensions, or a naming convention to identify files, likemyprog.exe, SCRIPT.BAT, file.txt. The problem with this is that the names can be abused by users. If one tries to execute a file which is not meant to be executed, the result would be nonsense and might even be dangerous to the point of crashing the system. One way around this problem is to introduce a protocol or standard format for executable files, so that when the OS opens a file for execution it first checks to see whether the file obeys the protocol. This method is used for binary files in UNIX, for instance.
- Protection. If several users will be storing files together on the same disk, should each user's files be exclusive to him or her?
- A hierarchical file system is a good starting point for organizing files, but it can be too restrictive. Sometimes it is useful to have a file appear in several places at one time. This can be accomplished with links. A link is not a copy of a file, but a pointer to where a file really is. By making links to other places in a hierarchical file system, its flexibility is increased considerably.

MULTIPLE WINDOWS AND SCREENS

Multitasking cannot be fully exploited if the users has only one output terminal monitor or screen. Each interactive program needs its own screen and keyboard. There are three solutions to this problem:

1. Several physical screens can be attached to the computer. This is expensive and probably wasteful.
2. Toggling between `logical screens'. By pressing a key on the keyboard the user can switch between two different images, which are separately maintained in memory.
3. Windows system.

The technology for the last of these solutions has only been available for a few years. While it is clearly the best of the three, it requires a considerable amount of memory and CPU power to implement. The problem of overlapping windows requires there to be a manager which controls the sharing of space on the screen. All of the graphics must be drawn and redrawn continuously. The operating system must provide primitives for doing this.

SINGLE-TASK OS

Before tackling the complexities of multi-tasking, it is useful to think about the operation of a single-task OS without all the clutter that multi-tasking entails. In a multi-task OS the features we shall discuss below have to be reproduced many times and then augmented by extra control structures.

MEMORY MAP AND REGISTERS

The key elements of a single-task computer are:

At the hardware level a computer consists of a CPU, memory and a number of peripheral devices. The CPU contains registers or `internal variables' which control its operation. The CPU can store information only in the memory it can address and in the registers of other microprocessors it is connected to. The CPU reads machine code instructions, one at a time, from the memory and executes them without stopping.

Here is a brief summary of the types of register a CPU has. Some microprocessors have several of each type.

Register	Purpose
Accumulator	Holds the data currently being worked on.
Program counter	Holds the address of the next instruction to be executed
Index (addressing) registers	Used to specify the address of data to be loaded into or saved from the accumulator, or operated on in some way.
Stack pointer	Points to the top of the CPUs own hardware controlled stack.
Status register	Contains status information after each instruction which can be tested to detect errors etc.

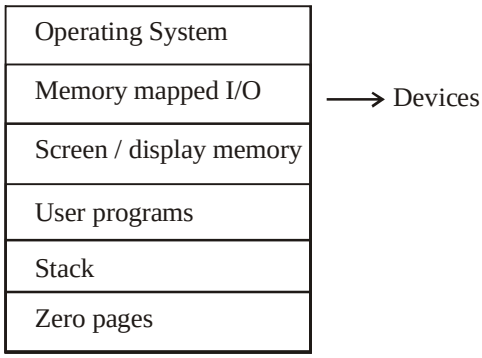
The memory, as seen by the CPU, is a large string of bytes starting with address O and increasing up to the maximum address.

Physically it is made up, like a jigsaw puzzle, of many memory chips and control chips. mapped into it. Normally, because of the hardware design of the CPU, not all of the memory is available to the user of the machine. Some of it is required for the operation of the CPU.

The areas of memory are :

- **Zero page:** The first `page' of the memory is often reserved for a special purpose. It is often faster to write to the zero page because you don't have to code the leading zero for the address - special instructions for the zero page can leave the `zero' implicit.
- **Stack:** Every CPU needs a stack for executing subroutines.
- **User programs:** Space the user programs can `grow into'.
- **Screen memory:** What you see on the screen of a computer is the image of an area of memory, converted into colours and positions by a hardware video-controller. The screen memory is the area of memory needed to define the colour of every `point' or `unit' on the screen. Depending on what kind of visual system a computer uses, this might be one byte per character and it might be four bytes per pixel
- **Memory mapped I/O:** Hardware devices like disks and video controllers contain smaller microprocessors of their own. The CPU gives them instructions by placing numbers into their registers. To make this process simpler, these device registers (only a few bytes per device, perhaps) are `wired' into the main memory map, so that writing to the device is the same as writing to the rest of the memory.
- **Operating system:** The operating system itself is a large program which often takes up a large part of the available memory.

Note that this figure is very simplified. It does not show, for instance, special memory which might be located inside the devices or CPU. Such memory is often used for caching. Also it does not show how the various components are connected together by means of a high speed data bus.



A simple schematic memory map of a microcomputer. The order of the different segments of memory can vary depending on the system.

STACK

A stack is a so-called last-in first-out (LIFO) data structure. That is to say - the last thing to be placed on top of a stack, when making it, is the first item which gets removed when un-making it. Stacks are used by the CPU to store the current position within a program before jumping to subroutines, so that they remember where to return to after the subroutine is finished. Because of the nature of the stack, the CPU can simply deposit the address of the next instruction to be executed (after the subroutine is finished) on top of the stack. When the subroutine is finished, the CPU pulls the first address it finds off the top of the stack and jumps to that location.

Notice that the stack mechanism will continue to work even if the subroutine itself calls another subroutine, since the second subroutine causes another stack frame to be saved on the top of the stack. When that is finished, it returns to the first subroutine and then to the original program in the correct order.

On many older microcomputers and in many operating systems the stack is allocated with a fixed size in advance. If too many levels of nested subroutines are called, the stack can overflow. Consider the following example code for a stack.

INPUT/OUTPUT

Input arrives at the computer at unpredictable intervals. The system must be able to detect its arrival and respond to it.

INTERRUPTS

Interrupts are hardware triggered signals which cause the CPU to stop what it is doing and jump to a special subroutine. Interrupts normally arrive from hardware devices, such as when the user presses a key on the keyboard, or the disk device has fetched some data from the disk. They can also be generated in software by errors like division by zero or illegal memory address.

When the CPU receives an interrupt, it saves the contents of its registers on the hardware stack and jumps to a special routine which will determine the cause of the interrupt and respond to it appropriately. Interrupts occur at different levels. Low level interrupts can be interrupted by high level interrupts. Interrupt handling routines have to work quickly, or the computer will be drowned in the business of servicing interrupts. For certain critical operations, low level interrupts can be ignored by setting a mask.

There is no logical difference between what happens during the execution of an interrupt routine and a subroutine. The difference is that interrupt routines are triggered by events, whereas software subroutines follow a prearranged plan.

An important area is the interrupt vector. This is a region of memory reserved by the hardware for servicing of interrupts. Each interrupt has a number from zero to the maximum number of interrupts supported on the CPU; for each interrupt, the interrupt vector must be programmed with the address of a routine which is to be executed when the interrupt occurs. i.e. when an interrupt occurs, the system examines the address in the interrupt vector for that interrupt and jumps to that location. The routine exits when it meets an RTI (return from interrupt) instruction.

BUFFERS

The CPU and the devices attached to it do not work at the same speed. Buffers are therefore needed to store incoming or outgoing information temporarily, while it is waiting to be picked up by the other party. A buffer is simply an area of memory which works as a waiting area. It is a first-in first-out (FIFO) data structure or queue.

SYNCHRONOUS AND ASYNCHRONOUS I/O

To start an I/O operation, the CPU writes appropriate values into the registers of the device controller. The device controller acts on the values it finds in its registers. For example, if the operation is to read from a disk, the device controller fetches data from the disk and places it in its local buffer. It then signals the CPU by generating an interrupt.

While the CPU is waiting for the I/O to complete it may do one of two things. It can do nothing or idle until the device returns with the data (synchronous I/O), or it can continue doing something else until the completion interrupt arrives (asynchronous I/O). The second of these possibilities is clearly much more efficient.

DMA - DIRECT MEMORY ACCESS

Very high speed devices could place heavy demands on the CPU for I/O servicing if they relied on the CPU to copy data word by word. The DMA controller is a device which copies blocks of data at a time from one place to the other, without the intervention of the CPU. To use it, its registers must be loaded with the information about what it should copy and where it should copy to. Once this is done, it generates an interrupt to signal the completion of the task. The advantage of the DMA is that it transfers large amounts of data before generating an interrupt. Without it, the CPU would have to copy the data one register-full at a time, using up hundreds or even thousands of interrupts and possibly bringing a halt to the machine!

MULTI-TASING AND MULTI-USER OS

To make a multi-tasking OS we need loosely to reproduce all of the features discussed in the last chapter for each task or process which runs. It is not necessary for each task to have its own set of devices. The basic hardware resources of the system are shared between the tasks. The operating system must therefore have a `manager' which shares resources at all times. This manager is called the `kernel' and it constitutes the main difference between single and multitasking operating systems.

COMPETITION FOR RESOURCES

Users - authentication

If a system supports several users, then each user must have his or her own place on the system disk, where files can be stored. Since each user's files may be private, the file system should record the owner of each file. For this to be possible, all users must have a user identity or login name and must supply a password which prevents others from impersonating them. Passwords are stored in a cryptographic (coded) form. When a user logs in, the OS encrypts the typed password and compares it to the stored version. Stored passwords are never decrypted for comparison.

Privileges and security

On a multi-user system it is important that one user should not be able to interfere with another user's activities, either purposefully or accidentally. Certain commands and system calls are therefore not available to normal users directly. The super-user is a privileged user (normally the system operator) who has permission to do anything, but normal users have restrictions placed on them in the interest of system safety.

For example: normal users should never be able to halt the system; nor should they be able to control the devices connected to the computer, or write directly into memory without making a formal request of the OS. One of the tasks of the OS is to prevent collisions between users.

Tasks - two-mode operation

It is crucial for the security of the system that different tasks, working side by side, should not be allowed to interfere with one another (although this occasionally happens in microcomputer operating systems, like the Macintosh, which allow several programs to be resident in memory simultaneously). Protection mechanisms are needed to deal with this problem. The way this is normally done is to make the operating system all-powerful and allow no user to access the system resources without going via the OS.

To prevent users from tricking the OS, multiuser systems are based on hardware which supports two-mode operation: privileged mode for executing OS instructions and user mode for working on user programs. When running in user mode a task has no special privileges and must ask the OS for resources through system calls. When I/O or resource management is performed, the OS takes over and switches to privileged mode. The OS switches between these modes personally, so provided it starts off in control of the system, it will alway remain in control.

- At boot-time, the system starts in privileged mode.
- During user execution, it is switched to user mode.

When interrupts occur, the OS takes over and it is switched back to privileged mode.

Other names for privileged mode are monitor mode or supervisor mode.

I/O and Memory protection

To prevent users from gaining control of devices, by tricking the OS, a mechanism is required to prevent them from writing to an arbitrary address in the memory. For example, if the user could modify the OS program, then it would clearly be possible to gain control of the entire system in privileged mode. All a user would have to do would be to change the addresses in the interrupt vector to point to a routine of their own making. This routine would then be executed when an interrupt was received in privileged mode.

The solution to this problem is to let the OS define a segment of memory for each user process and to check, when running in user mode, every address that the user program refers to. If the user attempts to read or write outside this allowed segment, a segmentation fault is generated and control returns to the OS. This checking is normally hard-wired into the hardware of the computer so that it cannot be switched off. No checking is required in privileged mode.

Time sharing

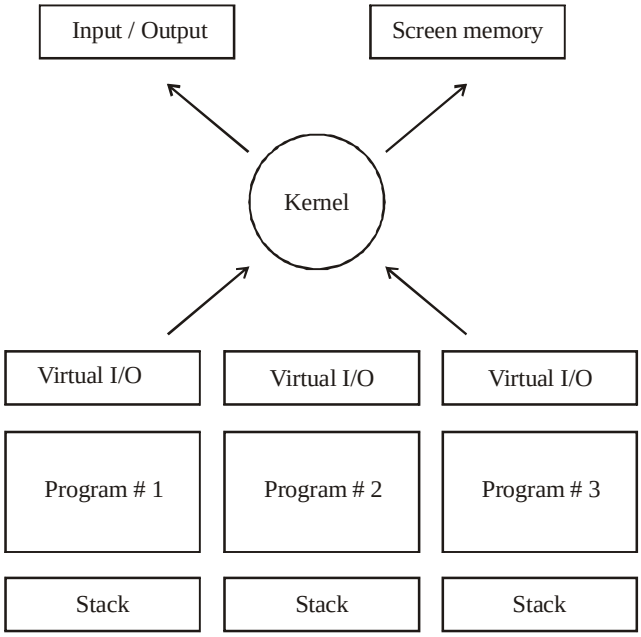
There is always the problem in a multi-tasking system that a user program will go into an infinite loop, so that control never returns to the OS and the whole system stops. We have to make sure that

the OS always remains in control by some method. Here are two possibilities:

- The operating system fetches each instruction from the user program and executes it personally, never giving it directly to the CPU. The OS software switches between different processes by fetching the instructions it decides to execute. This is a kind of software emulation. This method works, but it is extremely inefficient because the OS and the user program are always running together. The full speed of the CPU is not realized. This method is often used to make simulators and debuggers.
- A more common method is to switch off the OS while the user program is executing and switch off the user process while the OS is executing. The switching is achieved by hardware rather than software, as follows. When handing control to a user program, the OS uses a hardware timer to ensure that control will return after a certain time. The OS loads a fixed time interval into the timer's control registers and gives control to the user process. The timer then counts down to zero and when it reaches zero it generates a non-maskable interrupt, whereupon control returns to the OS.

MEMORY MAP

We can represent a multi-tasking system schematically as in figure. Clearly the memory map of a computer does not look like this figure. It looks like the figures in the previous chapter, so the OS has to simulate this behaviour using software. The point of this diagram is only that it shows the elements required by each process executing on the system.



Schematic diagram of a multitasking system.

Each program must have a memory area to work in and a stack to keep track of subroutine calls and local variables.

Each program must have its own input/output sources. These cannot be the actual resources of the system: instead, each program has a virtual I/O stream. The operating system arranges things so that the virtual I/O looks, to the user program, as though it is just normal I/O. In reality, the OS controls all the I/O itself and arranges the sharing of resources transparently. The virtual output stream for a program might be a window on the real screen, for instance. The virtual printer is really a print-queue. The keyboard is only `connected' to one task at a time, but the OS can share this too. For example, in a window environment, this happens when a user clicks in a particular window.

KERNEL AND SHELLS - LAYERS OF SOFTWARE

So far we have talked about the OS almost as though it were a living thing. In a multitasking, multi-user OS like UNIX this is not a bad approximation to the truth! In what follows we make use of UNIX terminology and all of the examples we shall cover later will refer to versions of the UNIX operating system.

The part of the OS which handles all of the details of sharing and device handling is called the kernel or core. The kernel is not something which can be used directly, although its services can be accessed through system calls. What is needed is a user interface or command line interface (CLI) which allows users to log onto the machine and manipulate files, compile programs and execute them using simple commands. Since this is a layer of software which wraps the kernel in more acceptable clothes, it is called a shell around the kernel.

It is only by making layers of software, in a hierachy that very complex programs can be written and maintained. The idea of layers and hierarchies returns again and again.

MULTIPROCESSORS - PARALLELISM

The idea of constructing computers with more than one CPU has become more popular recently. On a system with several CPUs it is not just a virtual fact that several tasks can be performed simultaneously - it is a reality. This introduces a number of complications in OS design. For example - how can we stop two independent processors from altering some memory location which they both share simultaneously (so that neither of them can detect the collision)? This is a problem in process synchronization. The solution to this problem is much simpler in a single CPU system since no two things ever happen truly simultaneously.

We shall consider this in more detail in later chapters. For now it is useful to keep in mind that multiprocessors are an important element of modern OS design.