Annai Women's College

(Affiliated to Bharathidasan University, tiruchirapalli 620 024)

Arts & Science

TNPL Road Punnamchatram, Karur – 639 1356

By

Asst.Prof. S. Sasikala.,MCA., M.Phil.,B.Ed.,

Department of Computer Applications

OPERATING SYSTEM

Unit I

Evolution of operating systems- Functions – Different views of OS – Batch processing, Multiprocessing, Time sharing OS – I / O programming concepts – Interrupt Structure & processing

Unit II

Memory Management – Single Contiguous Allocation- Partitioned Allocation – Relocatable Partitions allocations – Paged and Demand paged Memory Management – Segmented Memory Management – Segmented and Demand paged Memory Management – overlay Techniques – Swapping

Unit III

Processor Management – Job Scheduling – Process Scheduling – Functions and Policies – Evolution of Round Robin Multiprogramming Performance – Process Synchronisation – Wait and Signal mechanisms – Semaphores P & V Operations – Deadlock – Banker's Algorithm.

Unit IV

Device Management – Techniques for Device Management – I/O Traffic Controller, I/O Scheduler, I/O Device Handlers – Spooling.

Unit V

File Management: Simple File System, General Model of a File System, Physical and Logical File System. Case Studies: MSDOS, UNIX.

Text Book: Operating Systems – E. Madnick & John J.Donavan, Tata McGraw Hill Publishing Co., Limited. **Reference Book**: System Programming and Operating Systems – D.M. Dhamdhere, Tata McGraw Hill Publishing Co., Limited. *****

Operating system

UNIT-I

An operating system can be defined as:

- An operating system is a program that acts as an interface or intermediary between the user of a computer and the computer hardware.
- An operating system exploits the hardware resources of one or more processors to provide a set of services to system users and also manages secondary memory and Input/Output devices on the behalf of its users.
- An operating system is a set of program modules which provides a friendly interface between the user and the computer resources such as processors, memory, Input/Output devices and information.

OBJECTIVES AND FUNCTIONS

- **Convenience**

  The primary goal of an operating system is convenience for the user. If an application program is a set of machine instructions then it is completely responsible for controlling the computer hardware. It is a complicated task. To simplify this task, a set of system programs are provided, called utilities and they implement frequently used functions which assist in program creation, management of files and control of Input/Output devices.

- **Efficiency**

  The secondary goal of an operating system is efficient operation of the system. Operating system is responsible for managing the resources. That is the movement, storage and processing of data.A portion of operating system is in main memory. This includes the Kernel or nucleus, which contains the most frequently used functions in the operating system. The remainder of main memory contains other user programs and data. Operating system determine how much processor time is to be devoted to the execution of a program. That is the efficient utilization of the resources.

- **Ability to Evolve**

  Operating system should be constructed in such a way as to permit the effective development, testing, and introduction of new system functions.

Operating system will evolve over time for a number of reasons:

- Hardware upgrades plus new types of hardware. For example, view several applications at the same time through windows.
- New services, that is new measurement and control tools may be added.
- Fixes, that is faults will be discovered and fixes.

BASIC ELEMENTS

Processor

Processor controls the operation of computer and performs its data processing functions like arithmetic, logic and others.

Main Memory

Main memory is also called as **volatile memory, primary memory, real memory or temporary memory**, because it stores data and programs temporarily during the processing time only.

Input/Output Modules

Input/Output modules move data between the computer and its external environment like secondary memory, communications equipment and terminals etc.

System Inter Connection

System inter connection provide some structure and mechanisms that provide for communication among processors, main memory and Input/Output modules.

EVOLUTION OF OPERATING SYSTEMS

Serial Processing

Users access the computer in series. From the late 1940's to mid 1950's, the programmer interacted directly with computer hardware i.e., no operating system. These machines were run with a console consisting of display lights, toggle switches, some form of input device and a printer. Programs in machine code are loaded with the input device like card reader. If an error occur the program was halted and the error condition was indicated by lights. Programmers examine the registers and main memory to determine error. If the program is success, then output will appear on the printer.

Main problem here is the setup time. That is single program needs to load source program into memory, saving the compiled (object) program and then loading and linking together.

Simple Batch Systems

To speed up processing, jobs with similar needs are batched together and run as a group. Thus, the programmers will leave their programs with the operator. The operator will sort programs into batches with similar requirements.

The problems with Batch Systems are:

- Lack of interaction between the user and job.
- CPU is often idle, because the speeds of the mechanical I/O devices are slower than CPU.

For overcoming this problem use the Spooling Technique. Spool is a buffer that holds output for a device, such as printer, that can not accept interleaved data streams. That is when the job requests the printer to output a line, that line is copied into a system buffer and is written to the disk. When the job is completed, the output is printed. Spooling technique can keep both the CPU and the I/O devices working at much higher rates.

Multiprogrammed Batch Systems

Jobs must be run sequentially, on a first-come, first-served basis. However when several jobs are on a direct-access device like disk, job scheduling is possible. The main aspect of job scheduling is multiprogramming. Single user cannot keep the CPU or I/O devices busy at all times. Thus multiprogramming increases CPU utilization.

In when one job needs to wait, the CPU is switched to another job, and so on. Eventually, the first job finishes waiting and gets the CPU back.

The memory layout for multiprogramming system is shown below:

Time-Sharing Systems

Time-sharing systems are not available in 1960s. Time-sharing or multitasking is a logical extension of multiprogramming. That is processors time is shared among multiple users simultaneously is called time-sharing. The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is in Multiprogrammed batch systems its objective is maximize processor use, whereas in Time-Sharing Systems its objective is minimize response time.
Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receives an immediate response. For example, in a transaction processing, processor execute each user program in a short burst or quantum of computation. That is if n users are present, each user can get time quantum. When the user submits the command, the response time is seconds at most.
Operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time. Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.

For example IBM's OS/360.

Time-sharing operating systems are even more complex than multiprogrammed operating systems. As in multiprogramming, several jobs must be kept simultaneously in memory.

Personal-Computer Systems (PCs)

A computer system is dedicated to a single user is called personal computer, appeared in the 1970s. Micro computers are considerably smaller and less expensive than mainframe computers. The goals of the operating system have changed with time; instead of maximizing CPU and peripheral utilization, the systems developed for maximizing user convenience and responsiveness.

For e.g., MS-DOS, Microsoft Windows and Apple Macintosh.

Hardware costs for microcomputers are sufficiently low. Decrease the cost of computer hardware (such as processors and other devices) will increase our needs to understand the concepts of operating system. Malicious programs destroy data on systems. These programs may be self-replicating and may spread rapidly via worm or virus mechanisms to disrupt entire companies or even worldwide networks.

MULTICS operating system was developed from 1965 to 1970 at the Massachusetts Institute of Technology (MIT) as a computing utility. Many of the ideas in MULTICS were subsequently used at Bell Laboratories in the design of UNIX OS.

Parallel Systems

Most systems to date are single-processor systems; that is they have only one main CPU. Multiprocessor systems have more than one processor.

The advantages of parallel system are as follows:

throughput (Number of jobs to finish in a time period)
Save money by sharing peripherals, cabinets and power supplies
Increase reliability
Fault-tolerant (Failure of one processor will not halt the system).
Symmetric multiprocessing model

Each processor runs an identical job (copy) of the operating system, and these copies communicate. Encore's version of UNIX operating system is a symmetric model.
E.g., If two processors are connected by a bus. One is primary and the other is the backup. At fixed check points in the execution of the system, the state information of each job is copied from the primary machine to the backup. If a failure is detected, the backup copy is activated, and is restarted from the most recent checkpoint. But it is expensive.

Asymmetric multiprocessing model

Each processor is assigned a specific task. A master processor controls the system. Sun's operating system SunOS version 4 is a asymmetric model. Personal computers contain a microprocessor in the keyboard to convert the key strokes into codes to be sent to the CPU.

Distributed Systems

Distributed systems distribute computation among several processors. In contrast to tightly coupled systems (i.e., parallel systems), the processors do not share memory or a clock. Instead, each processor has its own local memory.

The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as loosely coupled systems or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, computers and so on.

The advantages of distributed systems are as follows:

Resource Sharing: With resource sharing facility user at one site may be able to use the resources available at another.
Communication Speedup: Speedup the exchange of data with one another via electronic mail.
Reliability: If one site fails in a distributed system, the remaining sites can potentially continue operating.
Real-time Systems
Real-time systems are used when there are rigid time requirements on the operation of a processor or the flow of data and real-time systems can be used as a control device in a dedicated application. Real-time operating system has well-defined, fixed time constraints otherwise system will fail.

E.g., Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, and home-applicance controllers.

There are two types of real-time systems:

Hard real-time systems

Hard real-time systems gurantees that critical tasks complete on time. In hard real-time systems secondary storage is limited or missing with data stored in ROM. In these systems virtual memory is almost never found.
Soft real-time systems

Soft real time systems are less restrictive. Critical real-time task gets priority over other tasks and retains the priority until it completes. Soft real-time systems have limited utility than hard real-time systems. E.g., Multimedia, virtual reality, Advanced Scientific Projects like undersea exploration and planetary rovers.
Different views of operating system:
User View and System View
Operating System is designed both by taking user view and system view into consideration.
Below is what the users and system thinks about Operating System.
User View
The goal of the Operating System is to maximize the work and minimize the effort of the user.
Most of the systems are designed to be operated by single user, however in some systems multiple users can share resources, memory. In these cases Operating System is designed to handle available resources among multiple users and CPU efficiently.
Operating System must be designed by taking both usability and efficient resource utilization into view.

In embedded systems(Automated systems) user view is not present.
Operating System gives an effect to the user as if the processor is dealing only with the current task, but in background processor is dealing with several processes.
System View
From the system point of view Operating System is a program involved with the hardware.
Operating System is allocator, which allocate memory, resources among various processes.It controls the sharing of resources among programs.

Functions of Operating System

There are Many Functions those are Performed by the Operating System But the Main Goal of Operating System is to Provide the Interface between the user and the hardware Means Provides the Interface for Working on the System by the user. The various Functions

Operating System as a Resource Manager

Operating System Also Known as the Resource Manager Means Operating System will Manages all the Resources those are Attached to the System means all the Resource like Memory and Processor and all the Input output Devices those are Attached to the System are Known as the Resources of the Computer System and the Operating system will Manage all the Resources of the System. The Operating System will identify at which Time the CPU will perform which Operation and in which Time the Memory is used by which Programs. And which Input Device will respond to which Request of the user means When the Input and Output Devices are used by the which Programs. So this will manage all the Resources those are attached to the Computer System.

Storage Management

Operating System also Controls the all the Storage Operations means how the data or files will be Stored into the computers and how the Files will be Accessed by the users etc. All the Operations those are Responsible for Storing and Accessing the Files is determined by the Operating System Operating System also Allows us Creation of Files, Creation of Directories and Reading and Writing the data of Files and Directories and also Copy the contents of the Files and the Directories from One Place to Another Place.

1) **Process Management : The Operating System also Treats the Process Management means all the Processes those are given by the user or the Process those are System 's own Process are Handled by the Operating System** . The Operating System will Create the Priorities foe the user and also Start or Stops the Execution of the Process and Also Makes the Child Process after dividing the Large Processes into the Small Processes.

2) **Memory Management:** Operating System also Manages the Memory of the Computer System means Provide the Memory to the Process and Also Deallocate the Memory

from the Process. And also defines that if a Process gets completed then this will deallocate the Memory from the Processes.

3) **Extended Machine :** Operating System also behaves like an Extended Machine means Operating system also Provides us Sharing of Files between Multiple Users, also Provides Some Graphical Environments and also Provides Various Languages for Communications and also Provides Many Complex Operations like using Many Hardware's and Software's.

4) **Mastermind:** Operating System also performs Many Functions and for those Reasons we can say that Operating System is a Mastermind. It provides Booting without an Operating System and Provides Facility to increase the Logical Memory of the Computer System by using the Physical Memory of the Computer System and also provides various Types of Formats Like NTFS and FAT File Systems.

Operating System also controls the Errors those have been Occurred into the Program and Also Provides Recovery of the System when the System gets Damaged Means When due to Some Hardware Failure , if System Doesn't Works properly then this Recover the System and also Correct the System and also Provides us the Backup Facility. And Operating System also breaks the large program into the Smaller Programs those are also called as the threads. And execute those threads one by one.

Batch Process

Batch processing: The grouping together of several processing jobs to be executed one after another by a computer, without any user interaction. This is achieved by placing a list of the commands to start the required jobs into a BATCH FILE that can be executed as if it were a single program: hence batch processing is most often used in operating systems that have a COMMAND LINE user interface. Indeed, batch processing was the normal mode of working in the early days of mainframe computers, but modern personal computer applications typically require frequent user interaction, making them unsuitable for batch execution.

A batch process performs a list of commands in sequence. It be run by a computer's operating system using a script or batch file, or may be executed within a program using a macro or internal scripting tool. For example, an accountant may create a script to open several financial programs at once, saving him the hassle of opening each program individually. This type of batch process would be executed by the operating system, such as Windows or the Mac OS. A Photoshop user, on the other hand, might use a batch process to modify several images at one time. For example, she might record an action within Photoshop that resizes and crops an image. Once the action has been recorded, she can batch process a folder of images, which will perform the action on all the images in the folder.

A batch process performs a list of commands in sequence. It be run by a computer's operating system using a script or batch file, or may be executed within a program using a macro or internal scripting tool. For example, an accountant may create a script to open several financial programs at once, saving him the hassle of opening each program individually. This type of batch process would be executed by the operating system, such as Windows or the Mac OS. A Photoshop user, on the other hand, might use a batch process to modify several images at one time. For example, she might record an action

within Photoshop that resizes and crops an image. Once the action has been recorded, she can batch process a folder of images, which will perform the action on all the images in the folder.
Batch processing can save time and energy by automating repetitive tasks. While it may take awhile to write the script or record the repetitive actions, doing it once is certainly better than having to do it many times.

Batch processing can save time and energy by automating repetitive tasks. While it may take awhile to write the script or record the repetitive actions, doing it once is certainly better than having to do it many times.
processing, but there are plenty of others. When you select several documents from the same application and print them all in one step (if the application allows you to do that), you are "batch printing," which is a form of batch processing. Or let's say that you want to send a whole group of files to someone else via your modem-if your *communications software* permits batch processing, you can choose all the files you want to send, and have the software send them off in a batch while you go to the kitchen for a snack. Batch processing is a good feature to have in most applications.

Multiprocessing

Multiprocessing is the use of two or more central processing units (CPUs) within a single computer system.[1][2] The term also refers to the ability of a system to support more than one processor or the ability to allocate tasks between them.[3] There are many variations on this basic theme, and the definition of multiprocessing can vary with context, mostly as a function of how CPUs are defined (multiple cores on one die, multiple dies in one package, multiple packages in one system unit, etc.).
According to some on-line dictionaries, a **multiprocessor** is a computer system having two or more processing units (multiple processors) each sharing main memory and peripherals, in order to simultaneously process programs.[4][5] A 2009 textbook defined multiprocessor system similarly, but noting that the processors may share "some or all of the system's memory and I/O facilities"; it also gave **tightly coupled system** as a synonymous term.[6]
At the operating system level, *multiprocessing* is sometimes used to refer to the execution of multiple concurrent processes in a system, with each process running on a separate CPU or core, as opposed to a single process at any one instant.[7][8] When used with this definition, multiprocessing is sometimes contrasted with multitasking, which may use just a single processor but switch it in time slices between tasks (i.e. a time-sharing system). Multiprocessing however means true parallel execution of multiple processes using more than one processor.[8] Multiprocessing doesn't necessarily mean that a single process or task uses more than one processor simultaneously; the term parallel processing is generally used to denote that scenario.[7] Other authors prefer to refer to the operating system techniques as multiprogramming and reserve the term *multiprocessing* for the hardware aspect of having more than one processor.[2][9] The remainder of this article discusses multiprocessing only in this hardware sense.
In Flynn's taxonomy, multiprocessors as defined above are MIMD machines.[10][11] As they are normally construed to be tightly coupled (share memory), multiprocessors are not the entire class of MIMD machines, which also contains message passing multicomputer systems.[10]
n a **multiprocessing** system, all CPUs may be equal, or some may be reserved for special purposes. A combination of hardware and operating system software design considerations

determine the symmetry (or lack thereof) in a given system. For example, hardware or software considerations may require that only one particular CPU respond to all hardware interrupts, whereas all other work in the system may be distributed equally among CPUs; or execution of kernel-mode code may be restricted to only one particular CPU, whereas user-mode code may be executed in any combination of processors. Multiprocessing systems are often easier to design if such restrictions are imposed, but they tend to be less efficient than systems in which all CPUs are utilized.

Systems that treat all CPUs equally are called symmetric multiprocessing (SMP) systems. In systems where all CPUs are not equal, system resources may be divided in a number of ways, including asymmetric multiprocessing (ASMP), non-uniform memory access (NUMA) multiprocessing, and clustered multiprocessing.

Instruction and data streams

In multiprocessing, the processors can be used to execute a single sequence of instructions in multiple contexts (single-instruction, multiple-data or SIMD, often used in vector processing), multiple sequences of instructions in a single context (multiple-instruction, single-data or MISD, used for redundancy in fail-safe systems and sometimes applied to describe pipelined processors or hyper-threading), or multiple sequences of instructions in multiple contexts (multiple-instruction, multiple-data or MIMD).

Processor coupling

Tightly coupled multiprocessor system

Tightly coupled multiprocessor **systems** contain multiple CPUs that are connected at the bus level. These CPUs may have access to a central shared memory (SMP or UMA), or may participate in a memory hierarchy with both local and shared memory (SM)(NUMA). The IBM p690 Regatta is an example of a high end SMP system. Intel Xeon processors dominated the multiprocessor market for business PCs and were the only major x86 option until the release of AMD's Opteron range of processors in 2004. Both ranges of processors had their own onboard cache but provided access to shared memory; the Xeon processors via a common pipe and the Opteron processors via independent pathways to the system RAM.

Chip multiprocessors, also known as multi-core computing, involves more than one processor placed on a single chip and can be thought of the most extreme form of tightly coupled multiprocessing. Mainframe systems with multiple processors are often tightly coupled.

Loosely coupled multiprocessor system

Main article: shared nothing architecture

Loosely coupled multiprocessor systems (often referred to as clusters) are based on multiple standalone single or dual processor commodity computers interconnected via a high speed communication system (Gigabit Ethernet is common). A Linux Beowulf cluster is an example of a loosely coupled system.

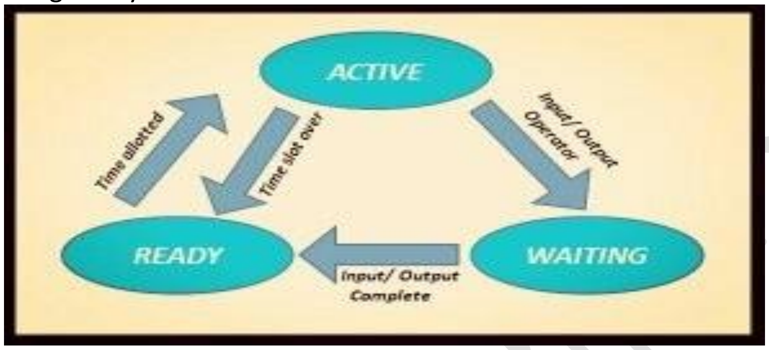
Tightly coupled systems perform better and are physically smaller than loosely coupled systems, but have historically required greater initial investments and may depreciate rapidly; nodes in a loosely coupled system are usually inexpensive commodity computers and can be recycled as independent machines upon retirement from the cluster.

Power consumption is also a consideration. Tightly coupled systems tend to be much more energy efficient than clusters. This is because considerable economy can be realized by designing components to work together from the beginning in tightly coupled systems, whereas loosely coupled systems use components that were not necessarily intended specifically for use in such systems.

Loosely coupled systems have the ability to run different operating systems or OS versions on different systems.

Time sharing operating system

A time sharing operating system is that in which each task is given some time to execute and all tasks are given time so that all processes run seamlessly without any problem. Suppose there are many users attached to a single system then each user has given time of CPU. No user can feel to have trouble in using the system.



Usage of time sharing operating system

Advantages of time sharing operating systems:

In time sharing systems all the tasks are given specific time and task switching time is very less so applications don't get interrupted by it. Many applications can run at the same time. You can also use time sharing in batch systems if appropriate which increases performance.

Time sharing systems is better way to run a business having lot of tasks to be done and no task get interrupted by the system. Each task and each user get its time. The tasks which are near to end get more attention so that new tasks can get time.

Threads also work on time sharing. Have you heard about multi-threading or multi-tasking? It is using time sharing to switch jobs/tasks. Suppose you are using MS word or MS excel. Now in these applications many small threads or tasks are running like spelling checking and grammatical checking in MS word. So time sharing operating systems have to give time to these application individual tasks and other applications also, so that all system behave correctly.

Disadvantages of time sharing operating systems:

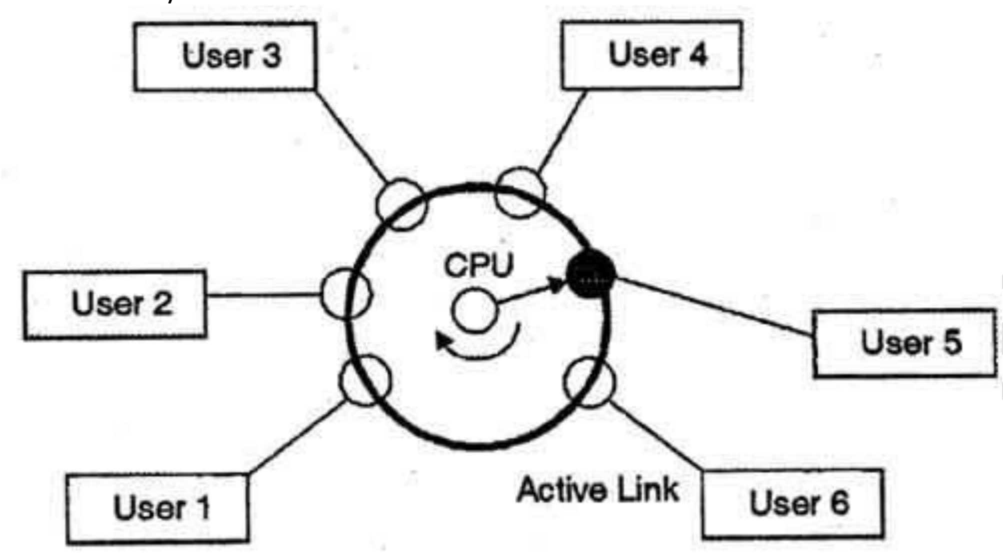
The big disadvantages of time sharing systems is that it consumes much resources so it need special operating systems. Switching between tasks becomes sometimes sophisticated as there are lot of users and applications running which may hang up the system. So the time sharing systems should have high specifications of hardware.

Examples of time sharing is:

The Multics & Unix operating systems are time sharing Operating Systems

Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of

multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.
The word 'Time-sharing' is not being used as 'Multi-tasking' which means doing multiple tasks simultaneously.



In above figure the user 5 is active but user 1, user 2, user 3, and user 4 are in waiting state whereas user 6 is in ready status.

I/O Programming concepts

One of the most important techniques in I/O programming is one that you should avoid: forcing the operating system to wait for your device. Almost everyone has had the experience of seeing Microsoft Windows "freeze up". Sometimes the freeze is due to a crash, but other times the system is simply waiting for a device to respond.
There are two basic programming techniques for dealing with waiting for a device: *synchronous*and *asynchronous*. Synchronous programming waits for the device and should be avoided. Asynchronous programming uses other techniques (such as waiting for interrupt requests). For more information about synchronous and asynchronous programming, see the following topics:
Synchronous I/O Programming
Asynchronous I/O Programming
Microsoft Vista has a new policy for dealing with problems with synchronous programming. For more information about this new policy, see Restricting Waits in Windows Vista for more information.
In earlier device driver programming, a driver would need to repeatedly request information from a driver until the answer was provided. This technique is called polling and should almost never be used. The best way to handle the problem of polling is to use hardware interrupts. For more information about hardware interrupts, see Servicing Interrupts. For more information on polling and why you should not use it, see Avoid Device Polling.

Interrupt structure and processing

An interrupt is a signal from a device attached to a computer or from a program within the computer that requires the operating system to stop and figure out what to do next. Almost all personal (or larger) computers today are *interrupt-driven* - that is, they start down the list of computer instructions in one program (perhaps an application such as a word processor) and keep running the instructions until either (A) they can't go any further or (B) an interrupt signal is sensed. After the interrupt signal is sensed, the computer either resumes running the current program or begins running another program.
Basically, a single computer can perform only one computer instruction at a time. But, because it can be interrupted, it can take turns in which programs or sets of instructions that it performs. This is known as multitasking. It allows the user to do a number of different things at the same time. The computer simply takes turns managing the programs that the user starts. Of course, the computer operates at speeds that make it seem as though all of the user's tasks are being performed at the same time. (The computer's operating system is good at using little pauses in operations and user think time to work on other programs.)
An operating system usually has some code that is called an *interrupt handler*. The interrupt handler prioritizes the interrupts and saves them in a queue if more than one is waiting to be handled. The operating system has another little program, sometimes called a scheduler, that figures out which program to give control to next.
In general, there are hardware interrupts and software interrupts. A hardware interrupt occurs, for example, when an I/O operation is completed such as reading some data into the computer from a tape drive. A software interrupt occurs when an application program terminates or requests certain services from the operating system. In a personal computer, a hardware interrupt request (IRQ) has a value that associates it with a particular device.

UNIT-II

Memory management

From Wikipedia, the free encyclopedia
In operating systems, **memory management** is the function responsible for managing the computer's primary memory.[1]:pp-105–208
The memory management function keeps track of the status of each memory location, either *allocated* or *free*. It determines how memory is allocated among competing processes, deciding which gets memory, when they receive it, and how much they are allowed. When memory is allocated it determines which memory locations will be assigned. It tracks when memory is freed or *unallocated* and updates the status.

Single contiguous allocation

Single allocation is the simplest memory management technique. All the computer's memory, usually with the exception of a small portion reserved for the operating system, is available to the single application. MS-DOS is an example of a system which allocates memory in this way. An embedded system running a single application might also use this technique.
A system using single contiguous allocation may still multitask by swapping the contents of memory to switch among users. Early versions of the Music operating system used this technique.

Partitioned allocation

Partitioned allocation divides primary memory into multiple *memory partitions*, usually contiguous areas of memory. Each partition might contain all the information for a specific jobor task. Memory management consists of allocating a partition to a job when it starts and unallocating it when the job ends.
Partitioned allocation usually requires some hardware support to prevent the jobs from interfering with one another or with the operating system. The IBM System/360 used a *lock-and-key* technique. Other systems used *base and bounds* registers which contained the limits of the partition and flagged invalid accesses. The UNIVAC 1108 *Storage Limits Register* had separate base/bound sets for instructions and data. The system took advantage of memory interleaving to place what were called the *i bank* and *d bank* in separate memory modules.[2]:3-3
Partitions may be either *static*, that is defined at Initial Program Load (IPL) or *boot time* or by the computer operator, or *dynamic*, that is automatically created for a specific job. IBM System/360 Operating System *Multiprogramming with a Fixed Number of Tasks* (MFT) is an example of static partitioning, and *Multiprogramming with a Variable Number of Tasks*(MVT) is an example of dynamic. MVT and successors use the term *region* to distinguish dynamic partitions from static ones in other systems.[3]:73
Partitions may be *relocatable* using hardware *typed memory*, like the Burroughs Corporation B5500, or base and bounds registers like the PDP-10 or GE-635. Relocatable partitions are able to be *compacted* to provide larger chunks of contiguous physical memory. Compaction moves "in-use" areas of memory to eliminate "holes" or unused areas of memory caused by process termination in order to create larger contiguous free areas.[4]:94
Some systems allow partitions to be *swapped out* to secondary storage to free additional memory. Early versions of IBM's *Time Sharing Option* (TSO) swapped users in and out of a single time-sharing partition.[5]

Relocatable Partitioned Allocation

Relocatable Partitioned Allocation :
The fragmentation problem is removed by relocated partitioned scheme. The blocks (jobs) already in the main memory can be relocated to make a hole (region) large enough for incoming information the relocation of the blocks already stored in the main memory accomplished by a technics called "compaction". The blocks currently in the main memory combined into a single block placed at one end of the memory this creates a single available region of maximum possible size at other end. The blocks can be relocated efficiently with a special hard ware facility for this purpose relocation register issued whose contents are

automatically added to every address. This register is used to reference memory after each compaction incoming blocks assigned to the available region of the memory when a new block is assigned in the available region compaction is again carried out.

Advantages :

1) It removes fragmentation problem.

Dis-advantages :

1) When a job ends the system may have to relocate all other jobs in order to re-compact.
2) There is still a small amount of memory is wasted.

Paged Memory Allocation : Paging is another solution for the fragmentation problem. In this each jobs address space divided into equal pieces called Pages. The memory is also divided into pieces of same size called Page Frames with the help of suitable hard ware mapping facility, any page can be placed into any Page Frame. The pages remain logically continuous but the corresponding Page Frames are not necessary continuous.

There is a seperate register for each Job called Page Map Table these registers may be special hard ware registers or reserved section of the micro memory. If the page size is too large, it becomes a relocatable partitioned memory. If the page size is too small many page registers (PMT'S) are required which increases the cost of the computer systems. The paged memory allocation is shown in figure. In the next example address space of job1 is divided into two pages. Job2 is divided into three pages and Job3 consist of only one page we are assuming each page size is 1000 B there is a page map table for each Job which consists of page. Number and the location of that page in memory , there is mapping that takes place with each address i.e.., each address in the Job 's address space can be transferred into an address in the physical memory.

In the above example a Job may not be a multiple of 1000 B long. Then a portion of last page of that Job will be wasted. This is called Internal fragmentation.

Advantages :

1. It solves the fragmentation problem without physically moving the pages in memory.
2. This allows a higher degree of multiprogramming.
3. The compaction in relocatable partitioned allocation is elimated.

Dis-Advantages :

1. Page address mapping hardware increase the cost of the computer system.
2. Extra core or extra registers needed for page map tables.
3. There is a possibility of internal fragmentation (or) page breakage may occur.
4. Some memory will still unused if the number of available page frames are not sufficient for the Job's page.

Paged memory management

Main article: *Virtual memory*

Paged allocation divides the computer's primary memory into fixed-size units called page frames, and the program's virtual *address space* into *pages* of the same size. The hardware memory management unit maps pages to frames. The physical memory can be allocated on a page basis while the address space appears contiguous.

Usually, with paged memory management, each job runs in its own address space. However, there are some single address space operating systems that run all processes within a single address space, such as IBM i, which runs all processes within a large address space, and IBM OS/VS2 SVS, which ran all jobs in a single 16MiB virtual address space.

Paged memory can be *demand-paged* when the system can move pages as required between primary and secondary memory.

Demand Paged memory management

In computer operating systems, **demand paging** (as opposed to anticipatory paging) is a method of virtual memory management. In a system that uses demand paging, the operating system copies a disk page into physical memory only if an attempt is made to access it and that page is not already in memory (*i.e.*, if a page fault occurs). It follows that a process begins execution with none of its pages in physical memory, and many page faults will occur until most of a process's working set of pages is located in physical memory. This is an example of a lazy loading technique.

Contents

Basic concept

Demand paging follows that pages should only be brought into memory if the executing process demands them. This is often referred to as lazy evaluation as only those pages demanded by the process are swapped from secondary storage to main memory. Contrast this to pure swapping, where all memory for a process is swapped from secondary storage to main memory during the process startup.

Commonly, to achieve this process a page table implementation is used. The page table maps logical memory to physical memory. The page table uses a bitwise operator to mark if a page is valid or invalid. A valid page is one that currently resides in main memory. An invalid page is one that currently resides in secondary memory. When a process tries to access a page, the following steps are generally followed:

Attempt to access page.

If page is valid (in memory) then continue processing instruction as normal.

If page is invalid then a **page-fault trap** occurs.

Check if the memory reference is a valid reference to a location on secondary memory. If not, the process is terminated (**illegal memory access**). Otherwise, we have to**page in** the required page.

Schedule disk operation to read the desired page into main memory.

Restart the instruction that was interrupted by the operating system trap.
Advantages
Demand paging, as opposed to loading all pages immediately:
Only loads pages that are demanded by the executing process.
As there is more space in main memory, more processes can be loaded, reducing the context switching time, which utilizes large amounts of resources.
Less loading latency occurs at program startup, as less information is accessed from secondary storage and less information is brought into main memory.
As main memory is expensive compared to secondary memory, this technique helps significantly reduce the bill of material (BOM) cost in smart phones for example. Symbian OS had this feature.
Disadvantages
Individual programs face extra latency when they access a page for the first time.
Low-cost, low-power embedded systems may not have a memory management unit that supports page replacement.
Memory management with page replacement algorithms becomes slightly more complex.
Possible security risks, including vulnerability to timing attacks; see Percival 2005 Cache Missing for Fun and Profit (specifically the virtual memory attack in section 2).
Thrashing which may occur due to repeated page faults.

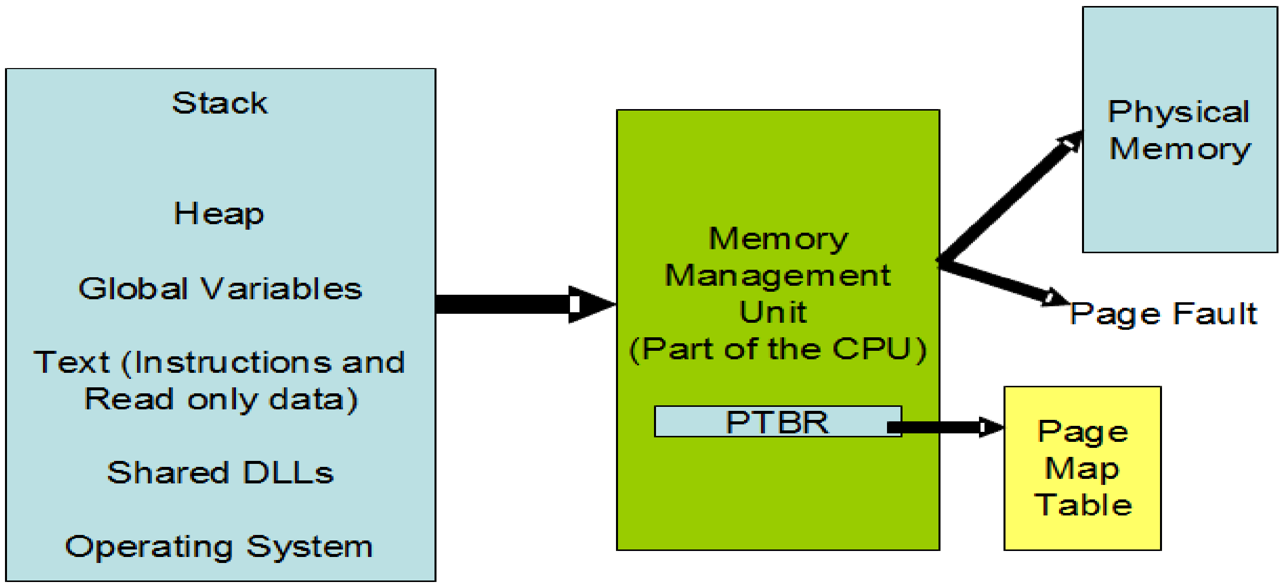
Segmented memory management

Segmented memory is the only memory management technique that does not provide the user's program with a 'linear and contiguous address space."[1]:p.165 *Segments* are areas of memory that usually correspond to a logical grouping of information such as a code procedure or a data array. Segments require hardware support in the form of a *segment table* which usually contains the physical address of the segment in memory, its size, and other data such as access protection bits and status (swapped in, swapped out, etc.)
Segmentation allows better access protection than other schemes because memory references are relative to a specific segment and the hardware will not permit the application to reference memory not defined for that segment.
It is possible to implement segmentation with or without paging. Without paging support the segment is the physical unit swapped in and out of memory if required. With paging support the pages are usually the unit of swapping and segmentation only adds an additional level of security.
Addresses in a segmented system usually consist of the segment id and an offset relative to the segment base address, defined to be offset zero.
The Intel IA-32 (x86) architecture allows a process to have up to 16,383 segments of up to 4GiB each. IA-32 segments are subdivisions of the computer's *linear address space*, the virtual address space provided by the paging hardware.[6]
The Multics operating system is probably the best known system implementing segmented memory. Multics segments are subdivisions of the computer's *physical memory* of up to 256 pages, each page being 1K 36-bit words in size, resulting in a maximum segment size of 1MiB (with 9-bit bytes, as used in Multics). A process could have up to 4046 segments.[7]

Segmented and Demand paged memory management

Recall that the logical memory address space of each process is divided into regions according to the type of data contained. (See The Virtual Memory Address Space) Some of these regions or sections of memory might be shared between different processes (text or shared libraries) or threads (global data). It may also be desired for quick reference to include the operating system in The Virtual Memory Address Space.

Since both the Windows PMT and the Linux PMT, divide the logical address into multiple tables, it fits to divide the tables per segments. The Page Tables (middle set of tables) can be shared between processes or threads. This allows a simple mechanism to share memory between processes and threads.



Overlay techniques

In a general computing sense, **overlaying** means "the process of transferring a block of program code or other data into internal memory, replacing what is already stored".[1] Overlaying is a programming method that allows programs to be larger than the computer's main memory.[2] An embedded system would normally use overlays because of the limitation of physical memory, which is internal memory for a system-on-chip, and the lack of virtual memory facilities.

Contents

[hide]

Usage

Constructing an overlay program involves manually dividing a program into self-contained object code blocks called **overlays** laid out in a tree structure. *Sibling* segments, those at the same depth level, share the same memory, called *overlay region* or *destination region*. An overlay manager, either part of the operating system or part of the overlay program, loads the required overlay from external memory into its destination region when it is needed. Often linkers provide support for overlays.[3]

Example

The following example shows the control statements that instruct the OS/360 Linkage Editor to link an overlay program, indented to show structure (segment names are arbitrary):

```
 INCLUDE SYSLIB(MOD1)
 INCLUDE SYSLIB(MOD2)
 OVERLAY A
   INCLUDE SYSLIB(MOD3)
     OVERLAY AA
       INCLUDE SYSLIB(MOD4)
       INCLUDE SYSLIB(MOD5)
     OVERLAY AB
        INCLUDE SYSLIB(MOD6)
 OVERLAY B
    INCLUDE SYSLIB(MOD7)
                       +--------------+
                       | Root Segment |
                       | MOD1, MOD2   |
                       +--------------+
                               |
                    +----------+----------+
                    |                     |
         +-------------+       +-------------+
         |  Overlay A  |       |  Overlay B  |
         |  MOD3       |       |  MOD7       |
         +-------------+       +-------------+
                    |
       +--------+--------+
       |                 |
+-------------+   +-------------+
| Overlay AA  |   | Overlay AB  |
| MOD4, MOD5  |   | MOD6        |
+-------------+   +-------------+
```

These statements define a tree consisting of the permanently resident segment, called the *root*, and two overlays A and B which will be loaded following the end of MOD2. Overlay A itself consists of two overlay segments, AA, and AB. At execution time overlays A and B will both

utilize the same memory locations; AA and AB will both utilize the same locations following the end of MOD3.
All the segments between the root and a given overlay segment are called a *path*.

Applications

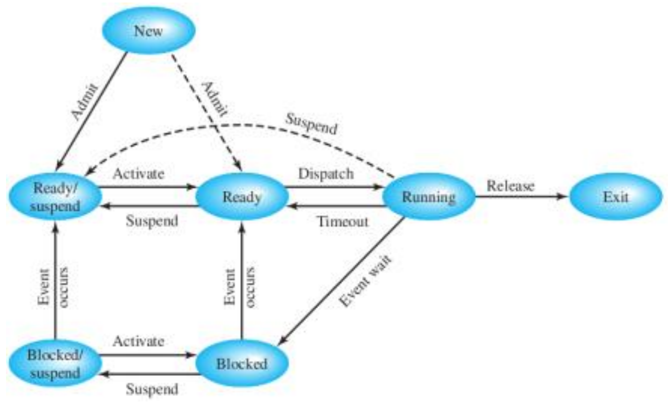
As of 2015, most business applications are intended to run on platforms with virtual memory. A developer on such a platform can design a program as if the memory constraint does not exist unless the program's working set exceeds the available physical memory. Most importantly, the architect can focus on the problem being solved without the added design difficulty of forcing the processing into steps constrained by the overlay size. Thus, the designer can use higher-level programming languages that do not allow the programmer much control over size (e.g. Java, C++, Smalltalk).
Still, overlays remain useful in embedded systems.[4] Some low-cost processors used in embedded systems do not provide a memory management unit (MMU). In addition many embedded systems are real-time systems and overlays provide more determinate response-time than paging. For example, the Space Shuttle *Primary Avionics System Software (PASS)* uses programmed overlays.[5]
Even on platforms with virtual memory, software components such as codecs may be decoupled to the point where they can be loaded in and out as needed.

Historical use

In the home computer era overlays were popular because the operating system and many of the computer systems it ran on lacked virtual memory and had very little RAM by current standards — the original IBM PC had between 16K and 64K depending on configuration. Overlays were a popular technique in Commodore BASIC to load graphics screens. In order to detect when an overlay was already loaded, a flag variable could be used.[6]
"Several DOS linkers in the 1980s supported [overlays] in a form nearly identical to that used 25 years earlier on mainframe computers."[4] Binary files containing memory overlays had a de facto standard extension, **.OVL**. This file type was used among others by WordStar, dBase, and the *Enable* DOS office automation software package from Enable Software, Inc.. The GFA BASIC compiler was able to produce .OVL files.

Swapping

Swapping is a simple memory/process management technique used by the operating system(os) to increase the utilization of the processor by moving some blocked process from the main memory to the secondary memory(hard disk);thus forming a queue of temporarily suspended process and the execution continues with the newly arrived process.After performing the swapping process,the operating system has two options in selecting a process for execution :
**Operating System can admit newly created process* (OR) **operating system can activate suspended process from the swap memory*.



Conclusion : If you have ever used any Linux based operating system then at the time of installation …Did you see an options/warning for the need of swap memory space?? If you have enough primary memory(RAM) e.g greater than 2GB then you may need not any swapping memory space for desktop users(I am using Ubuntu 10.04 LTS and total RAM is 4GB so I am not feeling any trouble without swap memory space) and some times using swap memory may slow down your computer performance

Unit-III

Process management

Process management is an integral part of any modern-day operating system (OS). The OS must allocate resources to processes, enable processes to share and exchange information, protect the resources of each process from other processes and enable synchronization among processes. To meet these requirements, the OS must maintain a data structure for each process, which describes the state and resource ownership of that process, and which enables the OS to exert control over each process.

Job Scheduling

Definition
Job scheduling is the process of allocating system resources to many different tasks by an operating system (OS). The system handles prioritized job queues that are awaiting CPU time and it should determine which job to be taken from which queue and the amount of time to be allocated for the job. This type of scheduling makes sure that all jobs are carried out fairly and on time.
Most OSs like Unix, Windows, etc., include standard job-scheduling abilities. A number of programs including database management systems (DBMS), backup, enterprise resource planning (ERP) and business process management (BPM) feature specific job-scheduling capabilities as well.

[LAST CHANCE] Manage Complex ERP Environments Webinar

Techopedia explains Job Scheduling
Job scheduling is performed using job schedulers. Job schedulers are programs that enable scheduling and, at times, track computer "batch" jobs, or units of work like the operation of a payroll program. Job schedulers have the ability to start and control jobs automatically by running prepared job-control-language statements or by means of similar communication with a human operator. Generally, the present-day job schedulers include a graphical user interface (GUI) along with a single point of control.
Organizations wishing to automate unrelated IT workload could also use more sophisticated attributes from a job scheduler, for example:
Real-time scheduling in accordance with external, unforeseen events
Automated restart and recovery in case of failures
Notifying the operations personnel
Generating reports of incidents
Audit trails meant for regulation compliance purposes
In-house developers can write these advanced capabilities; however, these are usually offered by providers who are experts in systems-management software.
In scheduling, many different schemes are used to determine which specific job to run. Some parameters that may be considered are as follows:
Job priority
Availability of computing resource
License key if the job is utilizing a licensed software
Execution time assigned to the user
Number of parallel jobs permitted for a user
Projected execution time
Elapsed execution time
Presence of peripheral devices
Number of cases of prescribed events

Process Scheduling

Definition
The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.
Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.
Process Scheduling Queues
The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.
The Operating System maintains the following important process scheduling queues −
Job queue − This queue keeps all the processes in the system.

Ready queue − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
Device queues − The processes which are blocked due to unavailability of an I/O device constitute this queue.

The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.
Two-State Process Model
Two-state process model refers to running and non-running states which are described below −

S.N.	**State & Description**
1	**Running** When a new process is created, it enters into the system as in the running state.
2	**Not Running** Processes that are not running are kept in queue, waiting for their turn to execute. Each entry in the queue is a pointer to a particular process. Queue is implemented by using linked list. Use of dispatcher is as follows. When a process is interrupted, that process is transferred in the waiting queue. If the process has completed or aborted, the process is discarded. In either case, the dispatcher then selects a process from the queue to execute.

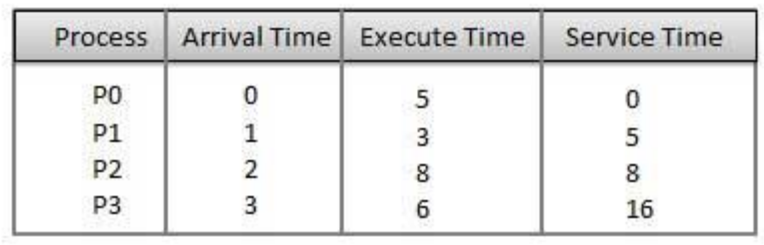
Schedulers
Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run.
Schedulers are of three types −
Long-Term Scheduler
Short-Term Scheduler
Medium-Term Scheduler
Long Term Scheduler
It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.
The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.
On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.
Short Term Scheduler
It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.
Medium Term Scheduler
Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.
A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.
Comparison among Scheduler

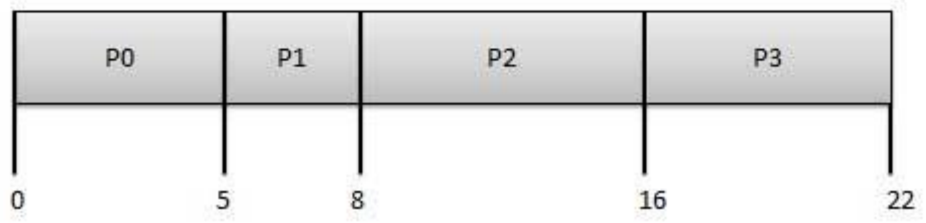
S.N.	**Long-Term Scheduler**	**Short-Term Scheduler**	**Medium-Term Scheduler**
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

Base and limit register value
Currently used register
Changed State
I/O State information
Accounting information
A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithms which we are going to discuss in this chapter −
First-Come, First-Served (FCFS) Scheduling
Shortest-Job-Next (SJN) Scheduling
Priority Scheduling
Shortest Remaining Time
Round Robin(RR) Scheduling
Multiple-Level Queues Scheduling
These algorithms are either **non-preemptive or preemptive**. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.
First Come First Serve (FCFS)

Jobs are executed on first come, first serve basis.
It is a non-preemptive, pre-emptive scheduling algorithm.
Easy to understand and implement.
Its implementation is based on FIFO queue.
Poor in performance as average wait time is high.



Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Wait time of each process is as follows −

Process Wait Time : Service Time - Arrival Time

P0	0 - 0 = 0
P1	5 - 1 = 4
P2	8 - 2 = 6
P3	16 - 3 = 13

Average Wait Time: (0+4+6+13) / 4 = 5.7

Functions and Policies

The **operating system**, or **OS**, of a computer is the first software that gets installed on the hard disk, and it remains there even when the computer is turned off. The OS is also the first software that gets loaded into the computer's memory when it is turned on. Once the OS is up and running, it performs five critical tasks.

System Management

Without an OS, your computer would not even start up. The first task of the OS is to manage the starting up of your computer, also known as booting up. When this happens, the OS makes sure all the various elements of your computer are working properly.

Once the OS is up and running, you're ready to start using your computer. Perhaps you're writing an essay for school, so you open up a word processing application. You do some research online, so you open up a web browser. And, while you are working, you want to listen to some music, so you launch your music player.

So, you're running multiple applications at the same time; this is known as multi-tasking. We take this for granted on today's computers, but in the early days of computing, a computer system only carried out a single task at a time.
While the OS is multi-tasking, it is constantly managing system resources. For example, applications require memory to run, and there's only so much memory installed on a computer system.
So, let's say you want to include an image into your essay, and you start using photo editing software to work on a high-quality photograph. This may take a fair bit of memory. You don't want the photo editing to be terribly slow, but you also don't want your music to stop playing. So, the OS tries to balance the memory needs of all the applications that are running. System management also includes routine maintenance tasks, such as file management, defragmenting disks to optimize hard drive storage, and keeping track of power supply.

Communication Services

The OS establishes an Internet connection so you can surf the Web or send e-mails. We take being online almost for granted, but there are a lot of protocols at work behind the scenes to make sure you stay connected. The OS makes sure you don't have to worry about managing these protocols by yourself.
Every time you visit a website, download a song, or send an e-mail, your computer interacts with a computer network that stretches across the globe. Your OS manages your connections, such as Ethernet and Wi-Fi connections, and ensures all communications with the network occur seamlessly.

Security

There are numerous security threats to your computer, in particular various types of malware, which is short for malicious software. This includes computer viruses, which can interfere with the normal operations of your computer. Viruses can be very harmful and result in loss of data or system crashes.
The OS of a computer has a number of built-in tools to protect against security threats, including the use of virus scanning utilities and setting up a firewall to block suspicious network activity.
One of the most common ways to get a computer virus is by e-mail. If you have received an e-mail message from someone you don't know with an unknown file attachment, be careful about opening up that file since it may just contain a virus or other malicious software.
While the OS has a number of built-in security tools, you may need additional software to set up the best protection, in particular virus scanning software. These types of utilities expand the functionality of the OS.
Another basic security feature is to control access to your computer by setting up a password. Without the password, someone else will not be able to get access to the software applications and files on your computer.

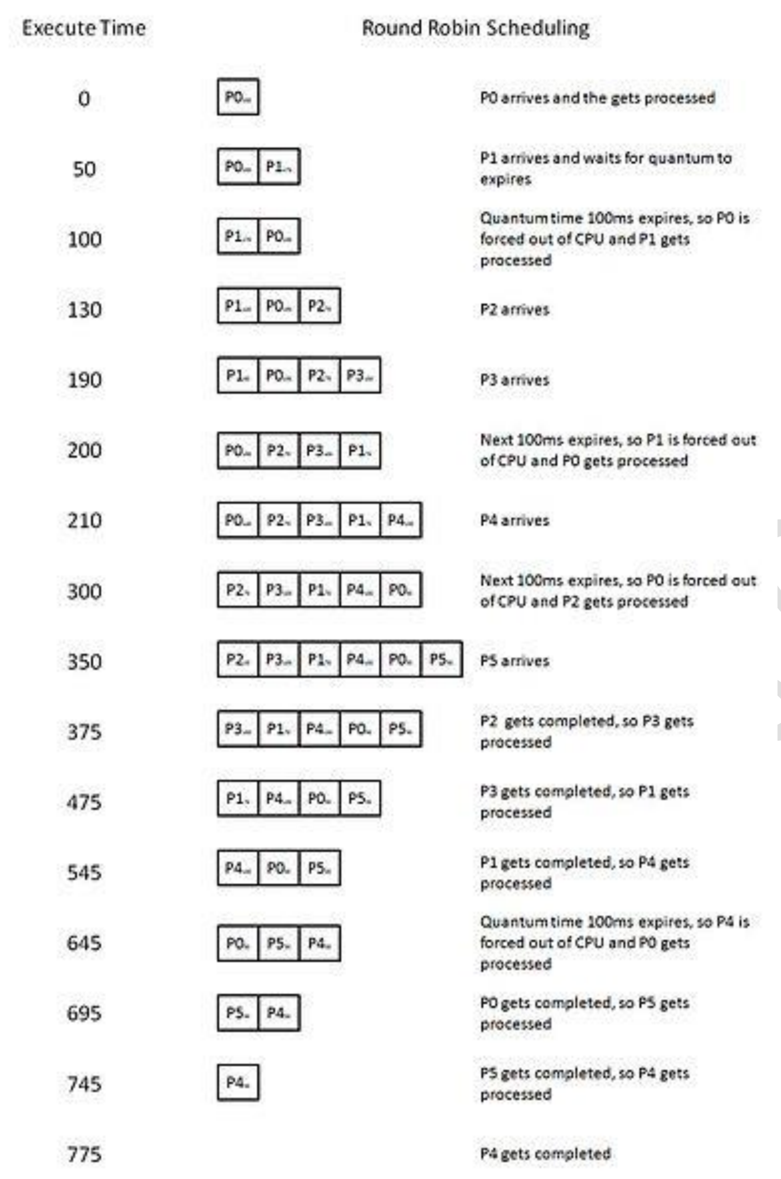
Round-robin scheduling

round-robin (RR) is one of the algorithms employed by process and network schedulers in computing.[1][2] As the term is generally used, time slices (also known as time quanta)[3] are assigned to each process in equal portions and in circular order, handling all processes without priority (also known as cyclic executive). Round-robin scheduling is simple, easy to implement, and starvation-free. Round-robin scheduling can also be applied to other

scheduling problems, such as data packet scheduling in computer networks. It is an operating system concept.
The name of the algorithm comes from the round-robin principle known from other fields, where each person takes an equal share of something in turn.
Contents
[hide]

Process scheduling[edit]
Main article: Process scheduler
To schedule processes fairly, a round-robin scheduler generally employs time-sharing, giving each job a time slot or *quantum*[4] (its allowance of CPU time), and interrupting the job if it is not completed by then. The job is resumed next time a time slot is assigned to that process. If the process terminates or changes its state to waiting during its attributed time quantum, the scheduler selects the first process in the ready queue to execute. In the absence of time-sharing, or if the quanta were large relative to the sizes of the jobs, a process that produced large jobs would be favoured over other processes.
Round-robin algorithm is a pre-emptive algorithm as the scheduler forces the process out of the CPU once the time quota expires.
For example, if the time slot is 100 milliseconds, and *job1* takes a total time of 250 ms to complete, the round-robin scheduler will suspend the job after 100 ms and give other jobs their time on the CPU. Once the other jobs have had their equal share (100 ms each), *job1* will get another allocation of CPU time and the cycle will repeat. This process continues until the job finishes and needs no more time on the CPU.
Job1 = Total time to complete 250 ms (quantum 100 ms).
First allocation = 100 ms.
Second allocation = 100 ms.
Third allocation = 100 ms
Total CPU time of *job1* = 250 ms
Consider the following table with the arrival time and execute time of the process with the quantum time of 100ms to understand the round-robin scheduling:

Process name	Arrival time	Execute time
P0	0	250
P1	50	170
P2	130	75
P3	190	100
P4	210	130
P5	350	50



Execute Time	Round Robin Scheduling	
0	P0	P0 arrives and the gets processed
50	P0 P1	P1 arrives and waits for quantum to expires
100	P1 P0	Quantum time 100ms expires, so P0 is forced out of CPU and P1 gets processed
130	P1 P0 P2	P2 arrives
190	P1 P0 P2 P3	P3 arrives
200	P0 P2 P3 P1	Next 100ms expires, so P1 is forced out of CPU and P0 gets processed
210	P0 P2 P3 P1 P4	P4 arrives
300	P2 P3 P1 P4 P0	Next 100ms expires, so P0 is forced out of CPU and P2 gets processed
350	P2 P3 P1 P4 P0 P5	P5 arrives
375	P3 P1 P4 P0 P5	P2 gets completed, so P3 gets processed
475	P1 P4 P0 P5	P3 gets completed, so P1 gets processed
545	P4 P0 P5	P1 gets completed, so P4 gets processed
645	P0 P5 P4	Quantum time 100ms expires, so P4 is forced out of CPU and P0 gets processed
695	P5 P4	P0 gets completed, so P5 gets processed
745	P4	P5 gets completed, so P4 gets processed
775		P4 gets completed

Another approach is to divide all processes into an equal number of timing quanta such that the quantum size is proportional to the size of the process. Hence, all processes end at the same time.

Network packet scheduling[edit]

Main article: Network scheduler

In best-effort packet switching and other statistical multiplexing, round-robin scheduling can be used as an alternative to first-come first-served queuing.

A multiplexer, switch, or router that provides round-robin scheduling has a separate queue for every data flow, where a data flow may be identified by its source and destination address. The algorithm lets every active data flow that has data packets in the queue to take turns in transferring packets on a shared channel in a periodically repeated order. The scheduling is work-conserving, meaning that if one flow is out of packets, the next data flow will take its place. Hence, the scheduling tries to prevent link resources from going unused.

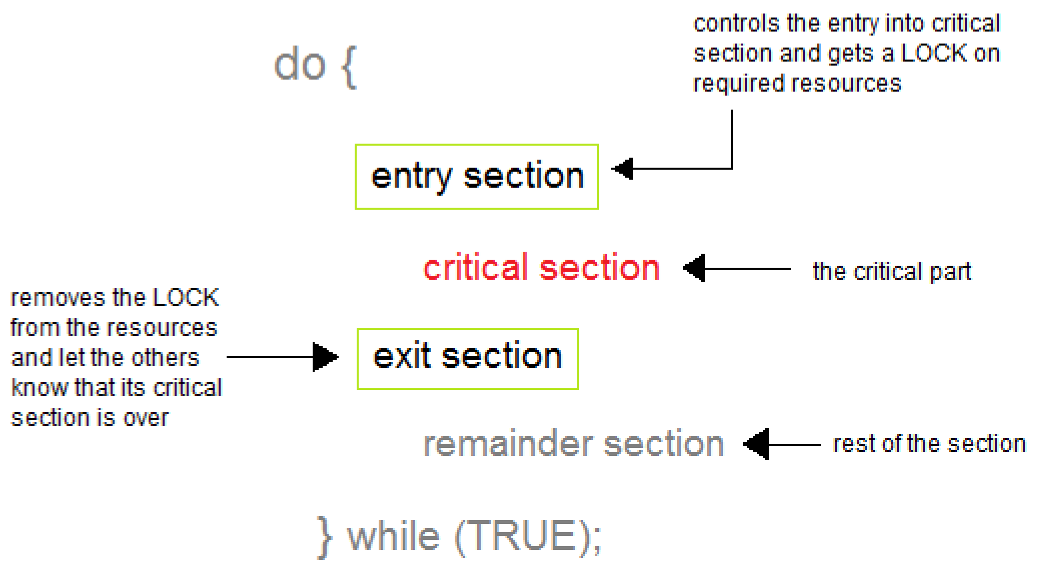
Round-robin scheduling results in max-min fairness if the data packets are equally sized, since the data flow that has waited the longest time is given scheduling priority. It may not be desirable if the size of the data packets varies widely from one job to another. A user that produces large packets would be favored over other users. In that case fair queuingwould be preferable.
If guaranteed or differentiated quality of service is offered, and not only best-effort communication, deficit round-robin (DRR) scheduling, weighted round-robin (WRR) scheduling, or weighted fair queuing (WFQ) may be considered.
In multiple-access networks, where several terminals are connected to a shared physical medium, round-robin scheduling may be provided by token passing channel accessschemes such as token ring, or by polling or resource reservation from a central control station.
In a centralized wireless packet radio network, where many stations share one frequency channel, a scheduling algorithm in a central base station may reserve time slots for the mobile stations in a round-robin fashion and provide fairness. However, if link adaptation is used, it will take a much longer time to transmit a certain amount of data to "expensive" users than to others since the channel conditions differ. It would be more efficient to wait with the transmission until the channel conditions are improved, or at least to give scheduling priority to less expensive users. Round-robin scheduling does not utilize this. Higher throughput and system spectrum efficiency may be achieved by channel-dependent scheduling, for example a proportionally fair algorithm, or maximum throughput scheduling. Note that the latter is characterized by undesirable scheduling starvation. This type of scheduling is one of the very basic algorithms for Operating Systems in computers which can be implemented through circular queue data structure.

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.



do {
controls the entry into critical section and gets a LOCK on required resources
entry section
critical section
the critical part
removes the LOCK from the resources and let the others know that its critical section is over
exit section
remainder section
rest of the section
} while (TRUE);

Solution to Critical Section Problem

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

Mutual Exclusion

Out of a group of cooperating processes, only one process can be in its critical section at a given point of time.

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.

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.

Wait and Signal mechanisms

n computer science, a **semaphore** is a variable or abstract data type used to control access to a common resource by multiple processes in a concurrent system such as amultiprogramming operating system.
A trivial semaphore is a plain variable that is changed (for example, incremented or decremented, or toggled) depending on programmer-defined conditions. The variable is then used as a condition to control access to some system resource.
A useful way to think of a semaphore as used in the real-world systems is as a record of how many units of a particular resource are available, coupled with operations to adjust that record *safely* (i.e. to avoid race conditions) as units are required or become free, and, if necessary, wait until a unit of the resource becomes available. Semaphores are a useful tool in the prevention of race conditions; however, their use is by no means a guarantee that a program is free from these problems. Semaphores which allow an arbitrary resource count are called **counting semaphores**, while semaphores which are restricted to the values 0 and 1 (or locked/unlocked, unavailable/available) are called **binary semaphores** and are used to implement locks.
The semaphore concept was invented by Dutch computer scientist Edsger Dijkstra in 1962 or 1963,[1] and has found widespread use in a variety of operating systems. It has also been used as the control mechanism for I/O controllers, for example in the Electrologica X8 computer.
Contents
[hide]

Library analogy[edit]

Suppose a library has 10 identical study rooms, to be used by one student at a time. Students must request a room from the front desk if they wish to use a study room. If no rooms are free, students wait at the desk until someone relinquishes a room. When a student has finished using a room, the student must return to the desk and indicate that one room has become free.
In the simplest implementation, the clerk at the front desk knows only the number of free rooms available, which they only know correctly if all of the students actually use their room while they've signed up for them and return them when they're done. When a student requests a room, the clerk decreases this number. When a student releases a room, the clerk increases this number. The room can be used for as long as desired, and so it is not possible to book rooms ahead of time.

In this scenario the front desk count-holder represents a counting semaphore, the rooms are the resource, and the students represent processes/threads. The value of the semaphore in this scenario is initially 10, with all rooms empty. When a student requests a room, they are granted access, and the value of the semaphore is changed to 9. After the next student comes, it drops to 8, then 7 and so on. If someone requests a room and the resulting value of the semaphore would be negative,[2] they are forced to wait until a room is freed (when the count is increased from 0). If one of the rooms was released, but there are several students waiting, then any method can be used to select the one who will occupy the room (like FIFO or flipping a coin). And of course, a student needs to inform the clerk about releasing their room only after really leaving it, otherwise, there can be an awkward situation when such student is in the process of leaving the room (they are packing their textbooks, etc.) and another student enters the room before they leave it.

Important observations[**edit**]

When used to control access to a pool of resources, a semaphore tracks only *how many* resources are free; it does not keep track of *which* of the resources are free. Some other mechanism (possibly involving more semaphores) may be required to select a particular free resource.

The paradigm is especially powerful because the semaphore count may serve as a useful trigger for a number of different actions. The librarian above may turn the lights off in the study hall when there are no students remaining, or may place a sign that says the rooms are very busy when most of the rooms are occupied.

The success of the protocol requires applications follow it correctly. Fairness and safety are likely to be compromised (which practically means a program may behave slowly, act erratically, hang or crash) if even a single process acts incorrectly. This includes:

requesting a resource and forgetting to release it;

releasing a resource that was never requested;

holding a resource for a long time without needing it;

using a resource without requesting it first (or after releasing it).

Even if all processes follow these rules, *multi-resource deadlock* may still occur when there are different resources managed by different semaphores and when processes need to use more than one resource at a time, as illustrated by the dining philosophers problem.

Semantics and implementation[edit]

Counting semaphores are equipped with two operations, historically denoted as P and V (see § Operation names for alternative names). Operation V increments the semaphore*S*, and operation P decrements it.

The value of the semaphore *S* is the number of units of the resource that are currently available. The P operation wastes time or sleeps until a resource protected by the semaphore becomes available, at which time the resource is immediately claimed. The V operation is the inverse: it makes a resource available again after the process has finished using it. One important property of semaphore *S* is that its value cannot be changed except by using the V and P operations.

A simple way to understand wait (P) and signal (V) operations is:

wait: If the value of semaphore variable is not negative, decrement it by 1. If the semaphore variable is now negative, the process executing wait is blocked (i.e., added to the semaphore's queue) until the value is greater or equal to 1. Otherwise, the process continues execution, having used a unit of the resource.

signal: Increments the value of semaphore variable by 1. After the increment, if the pre-increment value was negative (meaning there are processes waiting for a resource), it transfers a blocked process from the semaphore's waiting queue to the ready queue.
Many operating systems provide efficient semaphore primitives that unblock a waiting process when the semaphore is incremented. This means that processes do not waste time checking the semaphore value unnecessarily.
The counting semaphore concept can be extended with the ability to claim or return more than one "unit" from the semaphore, a technique implemented in Unix. The modified V and P operations are as follows, using square brackets to indicate atomic operations, i.e., operations which appear indivisible from the perspective of other processes:

```
function V(semaphore S, integer I):
   [S ← S + I]

function P(semaphore S, integer I):
   repeat:
      [if S ≥ I:
      S ← S − I
      break]
```

However, the remainder of this section refers to semaphores with unary V and P operations, unless otherwise specified.
To avoid starvation, a semaphore has an associated queue of processes (usually with FIFO semantics). If a process performs a P operation on a semaphore that has the value zero, the process is added to the semaphore's queue and its execution is suspended. When another process increments the semaphore by performing a V operation, and there are processes on the queue, one of them is removed from the queue and resumes execution. When processes have different priorities the queue may be ordered by priority, so that the highest priority process is taken from the queue first.
If the implementation does not ensure atomicity of the increment, decrement and comparison operations, then there is a risk of increments or decrements being forgotten, or of the semaphore value becoming negative. Atomicity may be achieved by using a machine instruction that is able to read, modify and write the semaphore in a single operation. In the absence of such a hardware instruction, an atomic operation may be synthesized through the use of a software mutual exclusion algorithm. On uniprocessor systems, atomic operations can be ensured by temporarily suspending preemption or disabling hardware interrupts. This approach does not work on multiprocessor systems where it is possible for two programs sharing a semaphore to run on different processors at the same time. To solve this problem in a multiprocessor system a locking variable can be used to control access to the semaphore. The locking variable is manipulated using a test-and-set-lock command

Semaphores P & V Operations

• An efficient synchronisation mechanism
• POSIX 1003.1.b, an IEEE standard.
• POSIX— for portable OS interfaces in Unix.
• P and V semaphores — represents the by integers in place of binary or unsigned integers 2008

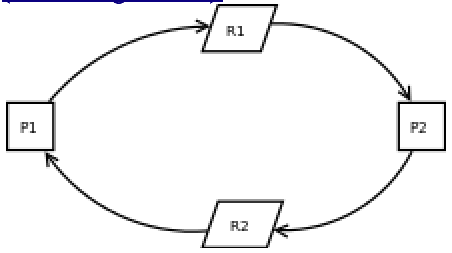
Chapter-7 L9: "Embedded Systems - Architecture, Programming
and Design" , Raj Kamal, Publs.: McGraw-Hill, Inc.
4
P and V semaphore Variables
• The semaphore, apart from initialization, i accessed only through two standard atomic operations─ P and V
• P (for wait operation)─ derived from a
Dutch word ‘Proberen’, which means 'to
test'.
• V (for signal passing operation)─ derived
from the word 'Verhogen' which means 'to
increment'. ing
and Design" , Raj Kamal, Publs.: McGraw-Hill, Inc.
5
 P semaphore function signals that the task
requires a resource and if not available
waits for it.
 V semaphore function signals which the
task passes to the OS that the resource is
now free for the other users.
P g
and Design" , Raj Kamal, Publs.: McGraw-Hill, Inc.
6
1. /* Decrease the semaphore variable*/
sem_1 = sem_1 -1;
2. /* If sem_1 is less than 0, send a message to
OS by calling a function waitCallToOS.
Control of the process transfers to OS,
because less than 0 means that some other
process has already executed P function on
sem_1. Whenever there is return for the OS,
it will be to step 1. */
if (sem_1 < 0){waitCallToOS (sem_1);}

P g
and Design" , Raj Kamal, Publs.: McGraw-Hill, Inc.
6
1. /* Decrease the semaphore variable*/
sem_1 = sem_1 -1;
2. /* If sem_1 is less than 0, send a message to
OS by calling a function waitCallToOS.
Control of the process transfers to OS,
because less than 0 means that some other
process has already executed P function on
sem_1. Whenever there is return for the OS,
it will be to step 1. */
if (sem_1 < 0){waitCallToOS (sem_1);}

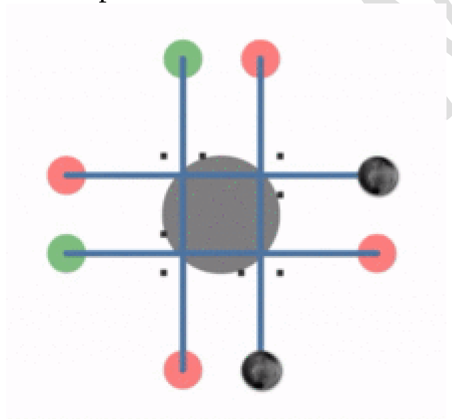
P 7
3. /* Increase the semaphore variable*/
sem_2 = sem_2 + 1;
4. /* If sem_2 is less or equal to 0, send a
message to OS by calling a function
signalCallToOS. Control of the process
transfers to OS, because < or = 0 means that
some other process is already executed P
function on sem_2. Whenever there is
return for the OS, it will be to step 3. */
if (sem_2 < = 0){signalCallToOS (sem_2);}V

Deadlock

This article is about the computer science concept. For other uses, see *Deadlock (disambiguation)*.



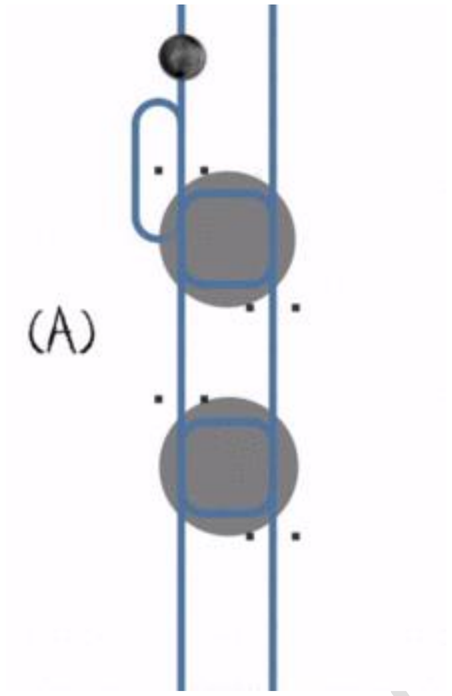
Both processes need resources to continue execution. P1 requires additional resource R1 and is in possession of resource R2, P2requires additional resource R2 and is in possession of R1; neither process can continue.



Four processes (blue lines) compete for one resource (grey circle), following a right-before-left policy. A deadlock occurs when all processes lock the resource simultaneously (black lines). The deadlock can be resolved by breaking the symmetry.

In concurrent computing, a **deadlock** is a state in which each member of a group is waiting for some other member to take action, such as sending a message or more commonly releasing a lock.[1] Deadlock is a common problem in multiprocessing systems, parallel computing, and distributed systems, where software and hardware locks are used to handle shared resources and implement process synchronization.[2]

In an operating system, a deadlock occurs when a process or thread enters a waiting state because a requested system resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process. If a process is unable to change its state indefinitely because the resources requested by it are being used by another waiting process, then the system is said to be in a deadlock.[3]
In a communications system, deadlocks occur mainly due to lost or corrupt signals rather than resource contention.[4]



Two processes competing for two resources in opposite order.
(A) A single process goes through.
(B) The later process has to wait.
(C) A deadlock occurs when the first process locks the first resource at the same time as the second process locks the second resource.
(D) The deadlock can be resolved by cancelling and restarting the first process.

Contents
 [hide]

Necessary conditions[edit]

A deadlock situation on a resource can arise if and only if all of the following conditions hold simultaneously in a system:[5]
Mutual exclusion: The resources involved must be unshareable; otherwise, the processes would not be prevented from using the resource when necessary. Only one process can use the resource at any given instant of time.[6]
Hold and wait or resource holding: a process is currently holding at least one resource and requesting additional resources which are being held by other processes.
No *preemption*: a resource can be released only voluntarily by the process holding it.
Circular wait: each process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource. In general, there is a set of waiting processes, P = {P_1, P_2, …, P_N}, such that P_1 is waiting for a resource held by P_2, P_2 is waiting for a resource held by P_3 and so on until P_N is waiting for a resource held by P_1.[3][7]
These four conditions are known as the Coffman conditions from their first description in a 1971 article by Edward G. Coffman, Jr.[7]

Deadlock handling[edit]

Most current operating systems cannot prevent deadlocks.[8] When a deadlock occurs, different operating systems respond to them in different non-standard manners. Most approaches work by preventing one of the four Coffman conditions from occurring, especially the fourth one.[9] Major approaches are as follows.

Ignoring deadlock[**edit**]

In this approach, it is assumed that a deadlock will never occur. This is also an application of the Ostrich algorithm.[9][10] This approach was initially used by MINIX and UNIX.[7] This is used when the time intervals between occurrences of deadlocks are large and the data loss incurred each time is tolerable.

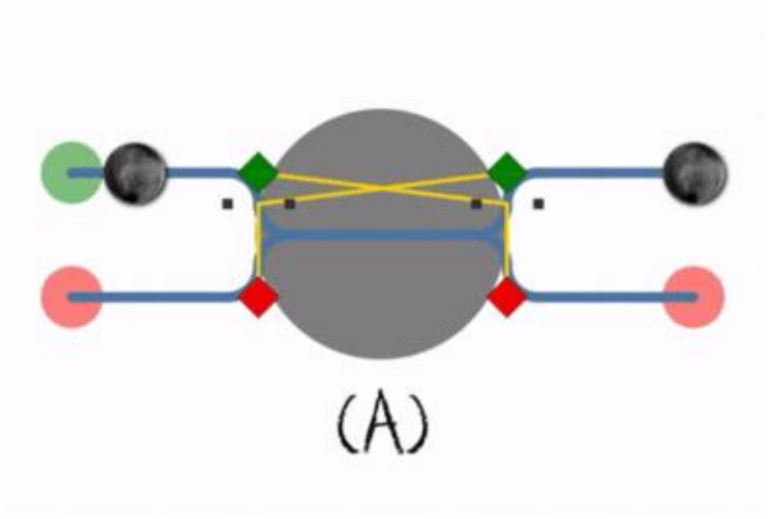
Detection[**edit**]

Under the deadlock detection, deadlocks are allowed to occur. Then the state of the system is examined to detect that a deadlock has occurred and subsequently it is corrected. An algorithm is employed that tracks resource allocation and process states, it rolls back and restarts one or more of the processes in order to remove the detected deadlock. Detecting a deadlock that has already occurred is easily possible since the resources that each process has locked and/or currently requested are known to the resource scheduler of the operating system.[10]
After a deadlock is detected, it can be corrected by using one of the following methods:[*citation needed*]

Process termination: one or more processes involved in the deadlock may be aborted. One could choose to abort all competing processes involved in the deadlock. This ensures that deadlock is resolved with certainty and speed.[*citation needed*] But the expense is high as partial computations will be lost. Or, one could choose to abort one process at a time until the deadlock is resolved. This approach has high overhead because after each abort an algorithm must determine whether the system is still in deadlock.[*citation needed*] Several factors must be considered while choosing a candidate for termination, such as priority and age of the process.[*citation needed*]
Resource preemption: resources allocated to various processes may be successively preempted and allocated to other processes until the deadlock is broken.[11]

Prevention[**edit**]

Main article: *Deadlock prevention algorithms*



(A) Two processes competing for one resource, following a first-come, first-served policy. (B) Deadlock occurs when both processes lock the resource simultaneously. (C) The deadlock can be resolvedby breaking the symmetry of the locks. (D) The deadlock can beprevented by breaking the symmetry of the locking mechanism.

Deadlock prevention works by preventing one of the four Coffman conditions from occurring. Removing the mutual exclusion condition means that no process will have exclusive access to a resource. This proves impossible for resources that cannot be spooled. But even with spooled resources, deadlock could still occur. Algorithms that avoid mutual exclusion are called non-blocking synchronization algorithms.

The hold and wait or resource holding conditions may be removed by requiring processes to request all the resources they will need before starting up (or before embarking upon a particular set of operations). This advance knowledge is frequently difficult to satisfy and, in any case, is an inefficient use of resources. Another way is to require processes to request resources only when it has none. Thus, first they must release all their currently held resources before requesting all the resources they will need from scratch. This too is often impractical. It is so because resources may be allocated and remain unused for long periods. Also, a process requiring a popular resource may have to wait indefinitely, as such a resource may always be allocated to some process, resulting in resource starvation.[12] (These algorithms, such as serializing tokens, are known as the all-or-none algorithms.)

The no *preemption* condition may also be difficult or impossible to avoid as a process has to be able to have a resource for a certain amount of time, or the processing outcome may be inconsistent or thrashing may occur. However, inability to enforce preemption may interfere with a priority algorithm. Preemption of a "locked out" resource generally implies a rollback, and is to be avoided, since it is very costly in overhead. Algorithms that allow preemption include lock-free and wait-free algorithms and optimistic concurrency control. If a process holding some resources and requests for some another resource(s) that cannot be immediately allocated to it, the condition may be removed by releasing all the currently being held resources of that process.

The final condition is the circular wait condition. Approaches that avoid circular waits include disabling interrupts during critical sections and using a hierarchy to determine apartial ordering of resources. If no obvious hierarchy exists, even the memory address of resources has been used to determine ordering and resources are requested in the increasing order of the enumeration.[3] Dijkstra's solution can also be used.

Livelock[edit]
A livelock is similar to a deadlock, except that the states of the processes involved in the livelock constantly change with regard to one another, none progressing.
The term was defined formally at some time during the 1970s. An early sighting in the published literature is in Babich's 1979 article on program correctness.[13] Livelock is a special case of resource starvation; the general definition only states that a specific process is not progressing.[14]
Livelock is a risk with some algorithms that detect and recover from deadlock. If more than one process takes action, the deadlock detection algorithm can be repeatedly triggered. This can be avoided by ensuring that only one process (chosen arbitrarily or by priority) takes action.[15]
Distributed deadlock[edit]
Distributed deadlocks can occur in distributed systems when distributed transactions or concurrency control is being used. Distributed deadlocks can be detected either by constructing a global wait-for graph from local wait-for graphs at a deadlock detector or by a distributed algorithm like edge chasing.
Phantom deadlocks are deadlocks that are falsely detected in a distributed system due to system internal delays but don't actually exist. For example, if a process releases a resource R1 and issues a request for R2, and the first message is lost or delayed, a coordinator (detector of deadlocks) could falsely conclude a deadlock (if the request for R2while having R1 would cause a deadlock).

Banker's algorithm

The **Banker's algorithm**, sometimes referred to as the **detection algorithm**, is a resource allocation and deadlock avoidance algorithm developed by Edsger Dijkstra that tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources, and then makes an "s-state" check to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to continue.
The algorithm was developed in the design process for the THE operating system and originally described (in Dutch) in EWD108.[1] When a new process enters a system, it must declare the maximum number of instances of each resource type that it may ever claim; clearly, that number may not exceed the total number of resources in the system. Also, when a process gets all its requested resources it must return them in a finite amount of time.
Contents
[hide]

Resources[edit]
For the Banker's algorithm to work, it needs to know three things:

How much of each resource each process could possibly request[MAX]
How much of each resource each process is currently holding[ALLOCATED]
How much of each resource the system currently has available[AVAILABLE]
Resources may be allocated to a process only if it satisfies the following conditions:
request ≤ available, else process waits until resources are available.
Some of the resources that are tracked in real systems
are memory, semaphores and interface access.
The Banker's Algorithm derives its name from the fact that this algorithm could be used in a banking system to ensure that the bank does not run out of resources, because the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers[*citation needed*]. By using the Banker's algorithm, the bank ensures that when customers request money the bank never leaves a safe state. If the customer's request does not cause the bank to leave a safe state, the cash will be allocated, otherwise the customer must wait until some other customer deposits enough.
Basic data structures to be maintained to implement the Banker's Algorithm:
Let n be the number of processes in the system and m be the number of resource types. Then we need the following data structures:
Available: A vector of length m indicates the number of available resources of each type. If Available[j] = k, there are k instances of resource type R_j available.
Max: An n×m matrix defines the maximum demand of each process. If Max[i,j] = k, then P_i may request at most k instances of resource type R_j.
Allocation: An n×m matrix defines the number of resources of each type currently allocated to each process. If Allocation[i,j] = k, then process P_i is currently allocated k instances of resource type R_j.
Need: An n×m matrix indicates the remaining resource need of each process. If Need[i,j] = k, then P_i may need k more instances of resource type R_j to complete the task.
Note: Need[i,j] = Max[i,j] - Allocation[i,j].
Example**[edit]**
Total system resources are:

A	B	C	D
6	5	7	6

Available system resources are:

A	B	C	D
3	1	1	2

Processes (currently allocated resources):

	A	B	C	D
P1	1	2	2	1
P2	1	0	3	3
P3	1	2	1	0

Processes (maximum resources):

	A	B	C	D
P1	3	3	2	2
P2	1	2	3	4
P3	1	3	5	0

Need = maximum resources - currently allocated resources
Processes (possibly needed resources):

	A	B	C	D
P1	2	1	0	1
P2	0	2	0	1
P3	0	1	4	0

Safe and Unsafe States[edit]

A state (as in the above example) is considered safe if it is possible for all processes to finish executing (terminate). Since the system cannot know when a process will terminate, or how many resources it will have requested by then, the system assumes that all processes will eventually attempt to acquire their stated maximum resources and terminate soon afterward. This is a reasonable assumption in most cases since the system is not particularly concerned with how long each process runs (at least not from a deadlock avoidance perspective). Also, if a process terminates without acquiring its maximum resource it only makes it easier on the system. A safe state is considered to be the decision maker if it's going to process ready queue.

Given that assumption, the algorithm determines if a state is **safe** by trying to find a hypothetical set of requests by the processes that would allow each to acquire its maximum resources and then terminate (returning its resources to the system). Any state where no such set exists is an **unsafe** state.

Unit IV

Device Management

Hardware devices typically provide the ability to **input** data into the computer or **output** data from the computer. To simplify the ability to support a variety of hardware devices, standardized application programming interfaces (API) are used.

- Application programs use the System Call API to request one of a finite set of preset I/O requests from the Operating System.
- The Operating System uses algorithms for processing the request that are device independent.
- The Operating System uses another API to request data from the device driver.
- The device driver is third party software that knows how to interact with the specific device to perform the I/O.
- Sometimes we have a layering of device drivers where one device driver will call on another device driver to facilitate the I/O. An example of this is when devices are connected to a USB port. The driver for the device will make use of the USB device driver to facilitate passing data to and from the device.
-
- **Techniques for Device Management**

All modern operating systems have a subsystem called the device manager. The device manager is responsible for detecting and managing devices, performing power management, and exposing devices to userspace. Since the device manager is a crucial part of any operating system, it's important to make sure it's well designed.

Contents

[hide]

Device Drivers

Device drivers allow user applications to communicate with a system's devices. They provide a high-level abstraction of the hardware to user applications while handling the low-level device-specific I/O and interrupts. Device drivers can be implemented as loadable kernel modules (for a Monolithic Kernel) or user-mode servers (for Microkernels).

Device Detection

The main role of the device manager is detecting devices on the system. Usually, devices are organized in a tree structure, with devices enumerating their children. Device detection should begin with a "root bus driver". On x86 systems, the root bus driver would use ACPI. The root bus driver sits at the root of the device tree. It detects the buses present on the system as well as devices directly connected to the motherboard. Each bus is then recursively enumerated, with its children continuing to enumerate their children until the bottom of the device tree is reached.

Each device that is detected should contain a list of resources for the device to use. Examples of resources are I/O, memory, IRQs, DMA channels, and configuration space. Devices are assigned resources by their parent devices. Devices should just use the resources they're given, which provides support for having the same device driver work on different machines where the resource assignments may be different, but the programming interface is otherwise the same.

Drivers are loaded for each device that's found. When a device is detected, the device manager finds the device's driver. If not loaded already, the device manager loads the driver. It then calls the driver to initialize that device.

How the device manager matches a device to a device driver is an important choice. The way devices are identified is very bus specific. On PCI, a device is identified through a combination of its vendor and device IDs. USB has the same scheme as PCI, using a vendor and product ID. ACPI uses PNP IDs to identify devices in the ACPI namespace. With this information, it's possible to build a database using matching IDs to drivers. This information is best stored in a separate file.

IPC

The device manager needs to implement some form of IPC between it and device drivers. IPC will be used by the device manager to send I/O requests to device drivers, and by drivers to respond to these requests. It is usually implemented with messages that contain data about the request, such as the I/O function code, buffer pointer, device offset, and buffer length. To respond to these I/O requests, every device driver needs dispatch functions used to handle each I/O function code. Each device needs a queue of these IPC messages for it to handle. On Windows NT, this IPC is done with I/O Request Packets.

Asynchronous I/O

There are two main types of I/O: synchronous I/O and asynchronous I/O. Synchronous I/O sends an I/O request and then puts the current thread to sleep until the I/O completes. Asynchronous I/O just sends the I/O request and then returns. I/O completion is reported asynchronously using a callback. Asynchronous I/O improves the efficiency of the system by allowing allowing for the program execution to continue while I/O is performed. It also allows for multiple I/O requests to be started and then handled in the order they complete, not the order they execute. However, this comes at the cost of making programming more complex than using synchronous I/O.

Internally, an operating system should use asynchronous I/O for all of its I/O requests. I/O requests are sent to drivers, and then the function that sent them immediately returns. Eventually, the I/O request will be handled. Once it completes, it returns through the driver stack and finally notifies the application of I/O completion. It can do this using callbacks, signals, or completion queues.

Synchronous I/O can simply be implemented as a special case of asychronous I/O. Just like with asynchronous I/O, an I/O request is sent to the driver, but instead of returning, the thread goes to sleep. Once the I/O completion event is queued, the thread will wake up and execute the callback before returning.

Power Management

The device manager also performs power management. Power management is a feature of hardware that allows for the power consumption of the system and devices to be controlled. Each device managed by the device manager should provide functions to set their power state. For power management support, all systems require a power management driver that controls the system power. On x86, this is done through ACPI. Each device also needs to support power management.

The device manager needs to respond to power management events. Power management events can come from two sources: the user or the system. User-generated power management events are created by user mode applications. They are system-wide events for shutting down, rebooting, hibernating, or putting the system to sleep. When the device manager receives a system-wide power management event, it sets the power state of the system.

System-generated power management events are events that come from the system hardware. Examples of system-generated power management events are plugging/unplugging an AC adapter or closing/opening the lid of a laptop. The device manager takes the appropriate action in response to the event.

Userspace Exposure

Once the kernel interfaces for device drivers are complete, one also needs to figure out how to expose devices to userspace. Most Unix-based operating systems expose devices through the filesystem tree. When devices are placed in the filesystem tree, there is a directory (usually /dev) containing special files that represent devices. The advantage of placing devices in the filesystem tree is that devices can be treated as files, meaning they can be read from or written to. Windows NT does not expose devices through the filesystem tree. Instead, there is an internal namespace of objects, through which devices can be found and accessed similarly to files.

No matter how devices are exposed, the functions that are provided for devices must be decided on as well. Both Unix-based operating systems and Windows NT treat devices like files, meaning their functions are open(), close(), read(), and write(). However, it was soon realized that this API would not be adequate for device functions that don't fit into these functions, like setting the graphics mode of a video card. For this purpose, a new syscall called ioctl() was developed, that allows a device to have special functions. However, this is by no means the only way to call device functions.

Existing Driver Interfaces

An operating system doesn't need to implement its own driver interface. A few driver interfaces have already been programmed with the intent of being integrated into operating systems. These driver interfaces can be implemented instead of a native driver interface, on top of a native driver interface, or along with a native driver interface.

Uniform Driver Interface

Main article: Uniform Driver Interface

Project UDI is a driver interface intended to be binary portable or source portable when running on different CPU architectures. It is not very widespread (however, neither are EDI or CDI); for example, due to philosophical concerns, Linux did not embrace UDI. However, several members of the community are striving to popularize UDI again since it would be of a huge benefit to hobby operating systems. You are strongly encouraged to participate by implenting a UDI environment and writing drivers.

I/O Traffic Controller

WARCO's ITC-2 traffic controller is modular in design and can be configured for all types of intersections. Using TCP/IP or 3G modem it can be directly connected to many different control and monitoring systems.

ITC-2: POWERFUL – FAST – RELIABLE

The controller can be delivered in three different sizes depending on the number of signal groups required.

For normal intersections the 3U rack with 16 signal groups and 32 detectors will be suitable. For larger intersections, or for covering more than one intersection, there are versions with 32 and 64 signal groups and 80 detectors.

ITC-2 is designed for any climate and is installed worldwide, from the cold Nordic to the hot Middle East and Africa regions. The larger cabinet is equipped with a swing frame allowing easy access to the backside of the unit.

The controller has been developed according to European and national standard

- **KEY BENEFITS**
-

- Linux operations system with web-interface
- The built-in operator panel allows full control and access to all parameters. The police panel can switch signals on/off and to amber flash as well as full stage control. The RS232 terminal interface provides further access to the internal software for control and debugging. The parameters are password protected with two security levels.
- The detection system is based on inductive loop-detectors with 8 loops per card. Alternatively video detection Autoscope Rackvision or Atlas cards can be installed.
- Lamp group cards with triac outputs for 230 or 42 VAC with full monitoring of voltage and currents on all outputs. Each card controls two signal groups.
- Optional I/O-cards for control of relay input or output.
- CPU-board with ARM processor for control and supervision of functions. The CPU has a real time clock with battery backup.
- ITC-2 is available also in a 42V version.

The ITC-2 controller's standard software provides a large number of parameter-controlled functions.

- There are 16 traffic plans and 16 traffic situations available with standard parameters for programming of local and central co-ordination. Cable-free linking is possible with a GPS clock.
- One controller can control up to four independent intersections in four separate rings. Each ring can have eight primary stages and an unlimited number of secondary stages. The logic is signal group controlled with a full conflict matrix between all groups.
- Traffic counting with internal detectors with seven-day backup.User defined counting interval.
- Fulfils Scandinavian LHOVRA specification.
- Built-in bus priority functions.
- Built-in advanced programmable control logics enabling the user to create new functions.
- For control and supervision ITC-2 has interfaces to Omnia/Utopia/Spot, Omnivue and EC-Trak UTC systems. The controller can send SMS or e-mails in case of faults.

- Internal web-interface.

I/O scheduling

Input/output (**I/O**) **scheduling** is the method that computer operating systems use to decide in which order the block I/O operations will be submitted to storage volumes. I/O scheduling is sometimes called disk scheduling.

Contents

[hide]

I/O scheduling[edit]

I/O scheduling usually has to work with hard disk drives that have long access times for requests placed far away from the current position of the disk head (this operation is called a seek). To minimize the effect this has on system performance, most I/O schedulers implement a variant of the elevator algorithm that reorders the incoming randomly ordered requests so the associated data would be accessed with minimal arm/head movement.

I/O schedulers can have many purposes depending on the goals; common purposes include the following:

- To minimize time wasted by hard disk seeks
- To prioritize a certain processes' I/O requests
- To give a share of the disk bandwidth to each running process
- To guarantee that certain requests will be issued before a particular deadline

Scheduling disciplines[edit]

Common scheduling disciplines include the following:

- Random scheduling (RSS)
- First In, First Out (FIFO), also known as First Come First Served (FCFS)
- Last In, First Out (LIFO)
- Shortest seek first, also known as Shortest Seek / Service Time First (SSTF)
- Elevator algorithm, also known as SCAN (including its variants, C-SCAN, LOOK, and C-LOOK)
- N-Step-SCAN SCAN of N records at a time
- FSCAN, N-Step-SCAN where N equals queue size at start of the SCAN cycle
- Completely Fair Queuing (CFQ) on Linux
- Anticipatory scheduling
- Noop scheduler
- Deadline scheduler
- mClock scheduler[2]

- Budget Fair Queueing (BFQ) scheduler.[3][4]
- Kyber[5][6]

Device Handler

- *This article, the first of a two-part series for experienced BASIC programmers, explalns how the Atari operating system handles the various devices attached to your computer and gives you a step-by-step approach to adding a new device handler or modifying an existing one. The BASIC programs work with any Atari 8-bit computer, disk or cassette.*
- The ability to change your computer's operating system is a very powerful technique. Atari 8-bit owners are in for a treat, because there are several ways to customize your system through software. To illustrate, two programs are included with this article. The first creates a device handler that does nothing. (Believe it or not, that can be useful.) The second modifies the printer handler to print non-printing characters.
- The printer handler was written for the Epson RX-80 and the C.Itoh Prowriter, but it should work with any printer capable of dot-graphics printing. Each program illustrates a different aspect of creating device handlers.
- What good is adding a device handler to the operating system instead of having your program perform the same function? To answer this, let's look at how the operating system interacts with the outside world.
- **CIO ROUTINE**
- Among the Atari operating system's best features is the way it handles input and output (I/O). All I/O operations are generally performed in the same way, regardless of which peripheral device is accessed---disk drive, keyboard, screen editor or printer. We can simplify I/O because it's normally handled through the Central Input/Output routine (CIO).
- In BASIC, I/O is done so naturally that you hardly notice the complexity of what is actually happening. One example is the use of CIO to send data to the printer:
- First, open the device through a CIO control block with the command OPEN #3,8,O,"P:". This does three things: 1) it tells the operating system to OPEN CIO control block #3 (IOCB3) for I/O and to prepare for I/O to the printer; 2) the 8 tells the operating system that data will flow from the computer to the printer; and 3) the "P:" tells the computer to send data to the printer. Therefore the OPEN command is for initialization.
- Summing up, a device handler simply tells the computer how to talk to a device. The computer needs to know the direction of data flow, which path (or channel) it will use, where to find data, where to put it and how much of it to grab.

- **DEVICE HANDLERS**
- If CIO is the computer's I/O interfnce to the user, then the device handlers are the computer's interface to peripheral devices.
- For example, how does the computer send data to the printer? Since the printer isn't a disk drive or keyboard, your Atari obviously needs a special routine, which is part of the device handler.
- Each device handler has six routines: OPEN, CLOSE, GET, PUT, STATUS, and SPECIAL (See Figure 1). In BASIC these machine language routines are controlled through I/O commands. For example, when your program issues the command OPEN #3,8,0,"P:" it is actually using the OPEN part of the printer handler.
- Every handler contains six machine language routines and a table containing the address of each routine minus one.
- Why the minus one? CIO accesses a function by pushing its address onto the stack, then executing an RTS (ReTurn from Subroutine). The RTS instruction directs the program to the address on the stack plus one. To arrive at the correct address, we must compensate by subtracting one from our target address.
- **HATABS**
- CIO finds the address of the appropriate handler table in the Handler Address Table, HATABS. This table is a 38-byte block of memory occupying locations 734--831 ($031A--$033F).
- Each device handler has its own three-byte entry in the Handler Address Table. The first byte is an ASCII character representing the name of the device (K for keyboard, D for disk drive, etc.). The next two bytes hold the address of that device's handler table.
- When you issue an OPEN #3,8,O,"P:" command, for example, CIO looks through HATABS for a "P:". Then it uses the next two bytes to find the address of the printer's handler table. Once found, CIO searches the printer's handler table for the address of the printer handler's OPEN routine. Finally, CIO executes the printer handler's OPEN routine (See Figure 2).
- Again, CIO finds the handler in two steps:
- - 1) Get the address of the appropriate handler table from HATABS.
  - 2) Get the address of the handler routines from the handler table.
- **WRITING YOUR OWN HANDLER**
- Now that we know how CIO locates the handler table, here's how to make your own handler:
- 1) Write the program for the handler. (The handler must have the functions listed in Figure 1.)

- 2) Set up a Handler Table with the address of each function (minus one) in the order given in Figure 1. The special function is a jump instruction beginning with a decimal 76, followed by the two-byte address of the special routine.
- 3) Make an entry in the Handler Address Table for the device.
- That's all. Now let's focus on the two examples and how they implement these steps.
-

  NULL HANDLER
- Type in Listing 1,NULLHAND.BAS, check it with TYPO II and SAVE a copy before you RUN it. When RUN, NULLHAND.BAS installs the Null Handler, N:, on Page Six. Once installed, you can test the handler by typing:
- LIST "N:"
- This command LISTs the program to the N: device. If your Atari responds with a "READY" prompt, the handler is properly installed. If you get an error message, however, something went wrong. Check NULLHAND.BAS for typing errors, and try again.
- One use of the null handler is to check a disk for a scrambled file (Error 164). One way to check is to use DOS to copy the file to the screen. This, however, is quite time- consuming. Instead, copy the file to the N: device. The Null Handler will do the job in a jiffy.
- To copy a file to the Null Handler, select choice C from the DOS menu and type: D:filename, N:
- If the file is read completely without error, then it's intact. All the files on a disk can be checked by typing "D: *.*,N"

Magnetic recording tape wound onto a spool may have contributed to the origin of the term

In computing, **spooling** is a specialized form of multi-programming for the purpose of copying data between different devices. In contemporary systems[a] it is usually used for mediating between a computer application and a slow peripheral, such as a printer. Spooling allows programs to "hand off" work to be done by the peripheral and then proceed to other tasks, or do not begin until input has been transcribed. A dedicated program, the **spooler**, maintains an orderly sequence of jobs for the peripheral and feeds it data at its own rate. Conversely, for slow *input* peripherals, such as a card reader, a spooler can maintain a sequence of computational jobs waiting for data, starting each job when all of the relevant input is available; see batch processing. The **spool** itself refers to the sequence of jobs, or the storage area where they are held. In many cases the spooler is able to drive devices at their full rated speed with minimal impact on other processing.

Spooling is a combination of buffering and queueing.

Contents

[hide]

Print spooling[edit]

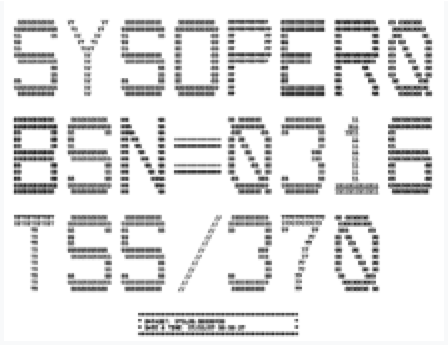
Nowadays, the most common use of spooling is printing: documents formatted for printing are stored in a queue at the speed of the computer, then retrieved and printed at the speed of the printer. Multiple processes can write documents to the spool without waiting, and can then perform other tasks, while the "spooler" process operates the printer.[1]

For example, when a large organization prepares payroll checks, the computation takes only a few minutes or even seconds, but the printing process might take hours. If the payroll program printed checks directly, it would be unable to proceed to other computations until all the checks were printed. Similarly, before spooling was added to PC operating systems, word processors were unable to do anything else, including interact with the user, while printing.

Spooler or print management software often includes a variety of related features, such as allowing priorities to be assigned to print jobs, notifying users when their documents have been printed, distributing print jobs among several printers, selecting appropriate paper for each document, etc.

A print server applies spooling techniques to allow many computers to share the same printer or group of printers.

Banner page[edit]



```
SYSOPER0
BSN=0316
TSS/370
```

Sample banner page generated by TSS/370

Print spoolers can often be configured to add a **banner page** (also called a *burst page*, *job sheet*, or *printer separator*) to the front of each document. These separate documents from each other, identify each document (e.g. with its title) and often also state who printed it (e.g. by username or job name). Banner pages are valuable in office environments where many people share a small number of printers. Depending on the configuration, banner pages might be generated on each client computer, or on a centralized print server, or by the printer itself.

On printers using fanfold continuous forms a leading banner page would often be printed twice, so that one copy would always be face-up when the jobs were separated. The page might include lines printed over the fold, which would be visible along the edge of a stack of printed output, allowing the operator to easily separate the jobs. Some systems would also print a banner page at the end of each job, assuring users that they had collected all of their printout.

Other applications[edit]

Spooling is also used to mediate access to punched card readers and punches, magnetic tape drives, and other slow, sequential I/O devices. It allows the application to run at the speed of the CPU while operating peripheral devices at their full rates speed.

A batch processing system uses spooling to maintain a queue of ready-to-run tasks, which can be started as soon as the system has the resources to process them.

Some store and forward messaging systems, such as uucp, used "spool" to refer to their inbound and outbound message queues, and this terminology is still found in the documentation for email and Usenet software, even though messages are often delivered immediately nowadays.

History[edit]

Peripherals device have always been much slower than core processing units. This was an especially severe problem for early mainframes. For example, a job which read punched cards or generated printed output directly was forced to run at the speed of the slow mechanical devices. The first spooling programs, such as IBM's "SPOOL System" (7070-IO-076) copied data from punched cards to magnetic tape, and from tape back to punched cards and printers. Hard disks, which are even faster and support random access, started to replace this use of magnetic tape in the middle 1960s, and by the 1970s had eliminated this use of tape.

Because the unit record equipment on IBM mainframes of the early 1960s was so slow, it was common to use a small offline machine such as a 1401 instead of spooling.

The term "spool" probably originates with the Simultaneous Peripheral Operations On-Line[2] (SPOOL) software. Its derivation is uncertain. *Simultaneous peripheral operations on-line* may be a backronym.[3] Another explanation is that it refers to "spools" or reels of magnetic tape.

List of spooling systems[edit]

- IBM SPOOL System, 7070-IO-076
- Integrated facility of various operating systems, e.g., GCOS, OS/360
- Houston Automatic Spooling Priority (HASP), prominent in the 1960s[4]
- Job Entry Subsystem 2/3, a follower of HASP[5]
- Priority Output Writers, Execution Processors and Input Readers (POWER)[6][7]
- GRASP
- The Spooler, IBM DOS/360, DOS/VS, and DOS/VSE spooler, 1975–1980s
- The Berkeley printing system (lpr/lpd)

- CUPS
- VM/370 RSCS (Remote Spooling Communications Subsystem)

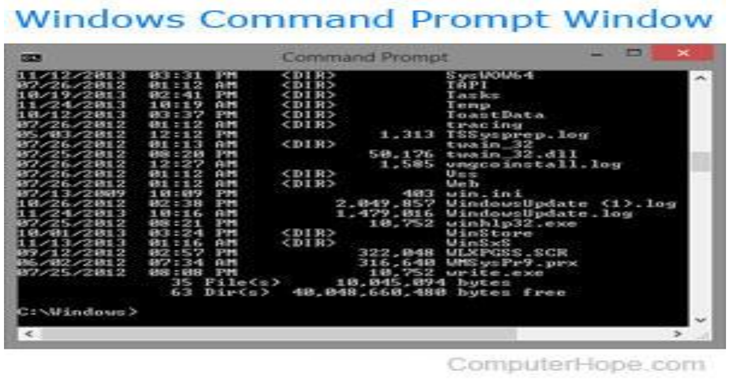
    -

-

MS-DOS

Updated: 04/26/2017 by Computer Hope

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 Windows.



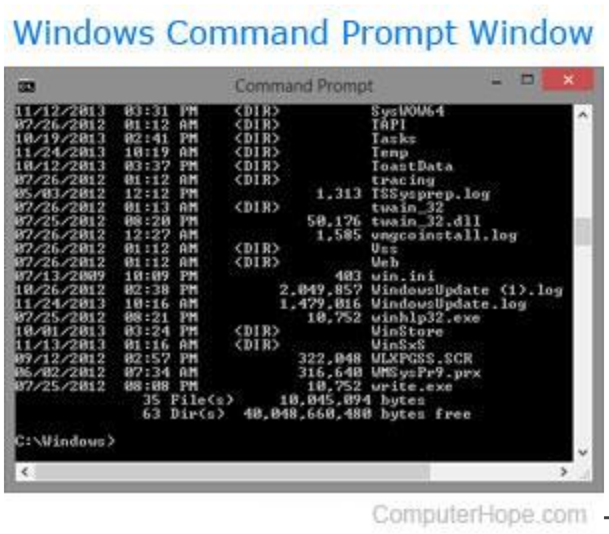
```
11/12/2013  03:31 PM    <DIR>          SysWOW64
07/26/2012  01:12 AM    <DIR>          TAPI
10/19/2013  02:41 PM    <DIR>          Tasks
11/24/2013  10:19 AM    <DIR>          Temp
10/12/2013  03:37 PM    <DIR>          ToastData
07/26/2012  01:12 AM    <DIR>          tracing
05/03/2012  12:12 PM             1,313 TSSysprep.log
07/26/2012  01:13 AM    <DIR>          twain_32
07/25/2012  08:20 PM            50,176 twain_32.dll
07/26/2012  12:27 AM             1,585 vmgcoinstall.log
07/26/2012  01:12 AM    <DIR>          Vss
07/26/2012  01:12 AM    <DIR>          Web
07/13/2009  10:09 PM               403 win.ini
10/26/2012  02:38 PM         2,049,857 WindowsUpdate (1).log
11/24/2013  10:16 AM         1,479,016 WindowsUpdate.log
07/25/2012  08:21 PM            10,752 winhlp32.exe
10/01/2013  03:24 PM    <DIR>          WinStore
11/13/2013  01:16 AM    <DIR>          WinSxS
09/12/2012  02:57 PM           322,048 WLXPGSS.SCR
06/02/2012  07:34 AM           316,640 WMSysPr9.prx
07/25/2012  08:08 PM            10,752 write.exe
              35 File(s)     10,045,094 bytes
              63 Dir(s)  40,048,660,480 bytes free

C:\Windows>
```

Today, MS-DOS is no longer used; however, the command shell, more commonly known as the **Windows command line**is still used by many users. The picture to the right, is an example of what an MS-DOS window more appropriately referred to as the Windows command line looks like running under Microsoft Windows.

Most computer users are only familiar with how to navigate Microsoft Windows using the mouse. Unlike Windows, MS-DOS is a command-line and is navigated by using MS-DOS commands. For example, if you wanted to see all the files in a folder in Windows you would double-click the folder to open the folder in Windows Explorer. In MS-DOS, to view that same folder you would navigate to the folder using the cd commandand then list the files in that folder using the dir command.

How to use the Windows command line (DOS)

Updated: 06/21/2017 by Computer Hope



```
11/12/2013  03:31 PM    <DIR>          SysWOW64
07/26/2012  01:12 AM    <DIR>          TAPI
10/19/2013  02:41 PM    <DIR>          Tasks
11/24/2013  10:19 AM    <DIR>          Temp
10/12/2013  03:37 PM    <DIR>          ToastData
07/26/2012  01:12 AM    <DIR>          tracing
05/03/2012  12:12 PM             1,313 TSSysprep.log
07/26/2012  01:13 AM    <DIR>          twain_32
07/25/2012  08:20 PM            50,176 twain_32.dll
07/26/2012  12:27 AM             1,505 vmgcoinstall.log
07/26/2012  01:12 AM    <DIR>          Vss
07/26/2012  01:12 AM    <DIR>          Web
07/13/2009  10:09 PM               403 win.ini
10/26/2012  02:38 PM         2,049,857 WindowsUpdate (1).log
11/24/2013  10:16 AM         1,479,016 WindowsUpdate.log
07/25/2012  08:21 PM            10,752 winhlp32.exe
10/01/2013  03:24 PM    <DIR>          WinStore
11/13/2013  01:16 AM    <DIR>          WinSxS
09/12/2012  02:57 PM           322,048 WLXPGSS.SCR
06/02/2012  07:34 AM           316,640 WMSysPr9.prx
07/25/2012  08:08 PM            10,752 write.exe
              35 File(s)     10,045,094 bytes
              63 Dir(s)  40,048,660,480 bytes free

C:\Windows>
```

This document covers the basic in navigating and using the Microsoft Windows command line. On this page, you'll learn how to move around in the command line, find files, manipulate files, and other important commands. Keep in mind that there are over 100 different commands that have been used in MS-DOS and the Windows command line. If you are interested in learning about the command line in more detail, see our DOS and command prompt overview, which gives a description and example for every command.

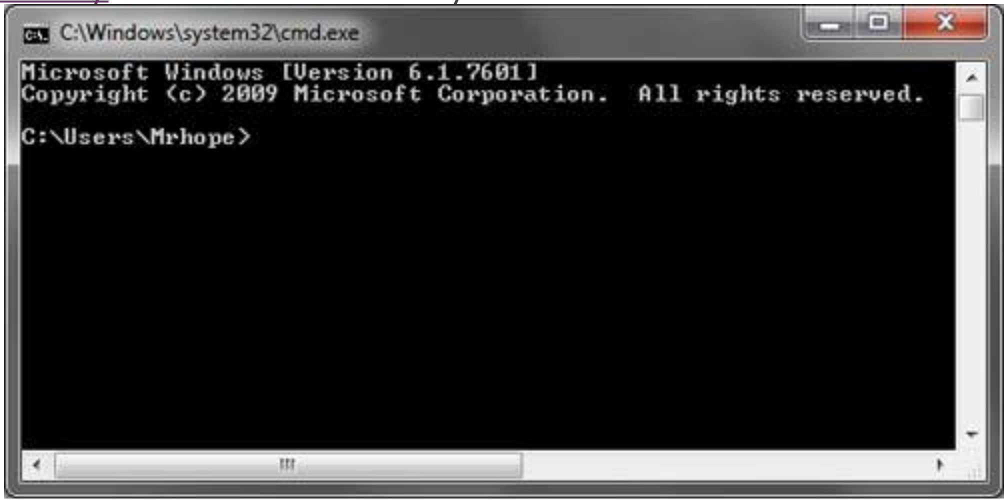
Get into the Windows command line

Open a Windows command line window by following the steps below. If you need additional information or alternative methods for all versions of Windows, see our how to get into DOS and Windows command line page.

1. Click Start
2. In the Search or Run line, type **cmd** (short for command), and press Enter.

Understanding the prompt

After following the above steps, the Windows command line should be shown (similar to the example below). Typically, Windows starts you at your user directory. In the example below, the user is Mrhope, so our prompt is C:\Users\Mrhope>. This prompt tells us we are in the C: drive (the default drive letter of the hard drive) and currently in the Mrhope directory, which is a subdirectory of the Users directory.



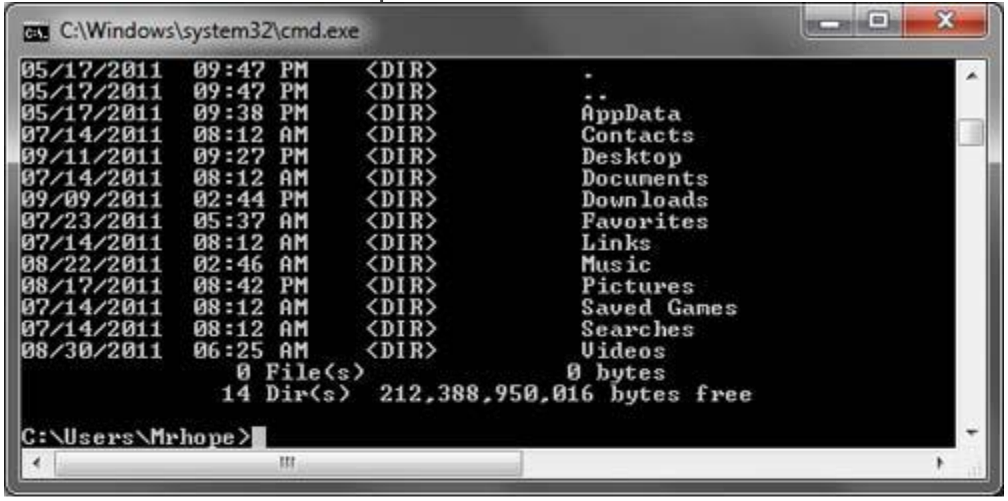
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Mrhope>
```

Key tips

- MS-DOS and the Windows command line are *not* case sensitive.
- The files and directories shown in Windows are also found in the command line.
- When working with a file or directory with a space, surround it in quotes. For example, the directory **My Documents** would be "My Documents" when typed.
- File names can have a long file name of 255 characters and a 3 character file extension.
- When a file or directory is deleted in the command line, it is not moved into the Recycle Bin.
- If you need help with any of command type /? after the command. For example, dir /? would give the options available for the dir command.

Listing the files

Let's learn your first command. Type **dir** at the prompt to list files in the current directory. You should get an output similar to the example image below. Without using any dir options this is how dir output appears. As can be seen, you are given lots of useful information including the creation date and time, directories (<DIR>), and the name of the directory or file. In the example below, there are 0 files listed and 14 directories as indicated by the status at the bottom of the output.



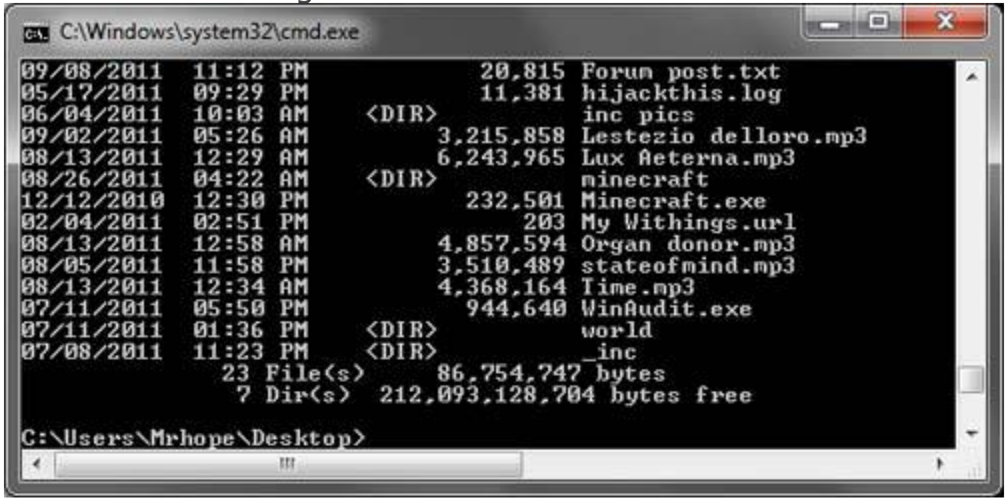
```
05/17/2011  09:47 PM    <DIR>          .
05/17/2011  09:47 PM    <DIR>          ..
05/17/2011  09:38 PM    <DIR>          AppData
07/14/2011  08:12 AM    <DIR>          Contacts
09/11/2011  09:27 PM    <DIR>          Desktop
07/14/2011  08:12 AM    <DIR>          Documents
09/09/2011  02:44 PM    <DIR>          Downloads
07/23/2011  05:37 AM    <DIR>          Favorites
07/14/2011  08:12 AM    <DIR>          Links
08/22/2011  02:46 AM    <DIR>          Music
08/17/2011  08:42 PM    <DIR>          Pictures
07/14/2011  08:12 AM    <DIR>          Saved Games
07/14/2011  08:12 AM    <DIR>          Searches
08/30/2011  06:25 AM    <DIR>          Videos
               0 File(s)              0 bytes
              14 Dir(s)  212,388,950,016 bytes free

C:\Users\Mrhope>
```

Every command in the command line has options, which are additional switches and commands that can be added after the command. For example, with the dir command you can type **dir /p** to list the files and directories in the current directory one page at a time. This switch is useful to see all the files and directories in a directory that has dozens or hundreds of files. Each of the command options and switches is listed in our DOS command overview. We offer guides for individual commands, as well. For example, if you want to see all the options for the **dir** command, refer to our dir command overview for a complete option listing.

The **dir** command can also be used to search for specific files and directories by using wildcards. For example, if you only wanted to list files or directories that begin with the letter "A" you could type **dir a*** to list only the AppData directory, in this above example. See the wildcard definition for other examples and help with using wildcards.

Moving into a directory

Now that we've seen a list of directories (shown below) in the current directory move into one of those directories. To move into a directory, we use the cd command, so to move into the Desktop type **cd desktop** and press enter. Once you've moved into a new directory the prompt should change, so in our example, the prompt is now C:\Users\Mrhope\Desktop>. Now in this desktop directory, see what files are found in this directory by typing the dir command again.



```
09/08/2011  11:12 PM            20,815 Forum post.txt
05/17/2011  09:29 PM            11,381 hijackthis.log
06/04/2011  10:03 AM    <DIR>          inc pics
09/02/2011  05:26 AM         3,215,858 Lestezio delloro.mp3
08/13/2011  12:29 AM         6,243,965 Lux Aeterna.mp3
08/26/2011  04:22 AM    <DIR>          minecraft
12/12/2010  12:30 PM           232,501 Minecraft.exe
02/04/2011  02:51 PM               203 My Withings.url
08/13/2011  12:58 AM         4,857,594 Organ donor.mp3
08/05/2011  11:58 PM         3,510,489 stateofmind.mp3
08/13/2011  12:34 AM         4,368,164 Time.mp3
07/11/2011  05:50 PM           944,640 WinAudit.exe
07/11/2011  01:36 PM    <DIR>          world
07/08/2011  11:23 PM    <DIR>          _inc
              23 File(s)     86,754,747 bytes
               7 Dir(s)  212,093,128,704 bytes free

C:\Users\Mrhope\Desktop>
```

Understand the files

In the Desktop directory, as shown in the above example, there are 23 files and 7 directories, representing different file types. In Windows, you are familiar with files having icons that help represent the file type. In the command line, the same thing is accomplished by the file extensions. For example, "forum posts.txt" is a text file because it has a .txt file extension. Time.mp3 is an MP3 music file and minecraft.exe is an executable file.

- Listing of file extensions and additional help with file extensions.

For most users, you'll only be concerned with executable files, which as mentioned above, is a file that ends with .exe and are also files that end with .com and .bat. When the name of these files are typed into the command line, the program runs, which is the same as double-clicking a file

in Windows. For example, if we wanted to run minecraft.exe typing "minecraft" at the prompt runs that program.
If you want to view the contents of a file, most versions of the command line use the edit command. For example, if we wanted to look at the log file hijackthis.log we would type **edit hijackthis.log** at the prompt. For 64-bit versions of Windows that do not support this command you can use the start command, for example, type **start notepad hijackthis.log** to open the file in Notepad. Further information about opening and editing a file from the command line can also be found on the link below.

- How to open and view the contents of a file on a computer.

Moving back a directory

You learned earlier the cd command can move into a directory. This command also allows you to go back a directory by typing **cd..** at the prompt. When this command is typed you'll be moved out of the Desktop directory and back into the user directory. If you wanted to move back to the root directory typing **cd\** takes you to the C:\> prompt. If you know the name of the directory you want to move into, you can also type cd\ and the directory name. For example, to move into C:\Windows>
type **cd\windows** at the prompt.

Creating a directory

Now with your basic understanding of navigating the command line let's start creating new directories. To create a directory in the current directory use the mkdir command. For example, create a directory called "test" by typing **mkdir test** at the prompt. If created successfully you should be returned to the prompt with no error message. After the directory has been created, move into that directory with the cd command.

Switching drives

In some circumstances, you may want to copy or list files on another drive. To switch drives in the Windows command line, type the letter of the drive followed by a colon. For example, if your CD-ROM drive was the D drive you would type **d:** and press enter. If the drive exists the prompt will change to that drive letter.

- How do you copy files from one drive to another drive?
- Additional information and examples of drive letters.

Creating a new file

You can create a new file from the command line using the edit command, copy con command, or using the start command to open a file.

- Complete steps on how to create a file in MS-DOS.

Creating a new batch file

In the new test directory let's create your first file. In most circumstances, you never need to create any file at the command line, but it is still good to understand how files are created. In this example, we are creating a batch file. A batch file is a file that ends with .bat and is a file that can help automate frequently used commands in the command line. We are calling this batch file "example", so type **edit example.bat** at the prompt. As mentioned in the document on creating a file, if the edit command does not work with your version of Windows, use the start command to open the batch file in Notepad. To perform this action, you type **start notepad example.bat** into the prompt.

Both of the above commands open a new blank example.bat window. In the file, type the below three lines, which clear the screen with the cls command and then run the dir command.

```
@echo off
cls
dir
```

After these three lines have been typed into the file save and exit the file. If you are in the edit command click File (or press Alt+F) and then Save. After the file has been saved and you are back into the command prompt, typing dir should display the example.bat in the test directory.

Now run the batch file to get a better understanding of what a batch file does. To run the batch file type **example** at the prompt, which executes the batch file and clears the screen and then runs the dir command.

- Full information and additional examples on batch files.

Moving and copying a file

Now that we've created a file let's move it into an alternate directory. To help make things easier, create another directory for the files. So, type **mkdir dir2** to create a new directory in the test directory called dir2. After the new directory has been created, use the move command to move the example.bat file into that directory. To do this type **move example.bat dir2** at the prompt, if done successfully you should get a message indicated the file was moved. You could also substitute the move command for the copy command to copy the file instead of moving it.

Rename a file

After the file has been moved into the dir2 directory, move into that directory with the cd command to rename the file. In the dir2 directory use the rename command to rename the example file into an alternate name. Type **rename example.bat first.bat** at the prompt to rename the file to first.bat. Now when using the dir command you should see the first.bat as the only file.

Deleting a file

Now that we've had our fun with our new file, delete the file with the del command. Type **del first.bat** to delete the first.bat file. If successful, you are returned to the prompt with no errors and the dir command shows no files in the current directory.

Renaming a directory

Go back one directory to get back into the test directory by using the **cd..** command mentioned earlier. Now rename our dir2 directory to something else using the same rename command we used earlier. At the prompt, type **rename dir2 hope** to rename the directory to hope. After this command has been completed, type dir and you should now see one directory called hope.

Removing a directory

While still in the test directory, remove the hope directory by using the rmdir command. At the prompt, type **rmdir hope** to remove the hope directory.

How to list available commands

After getting a good understanding of using the command line from the steps shown above you can move on to other available commands by typing **help** at the command line. Typing "help" gives you a listing of available commands with a brief description of each of the commands.

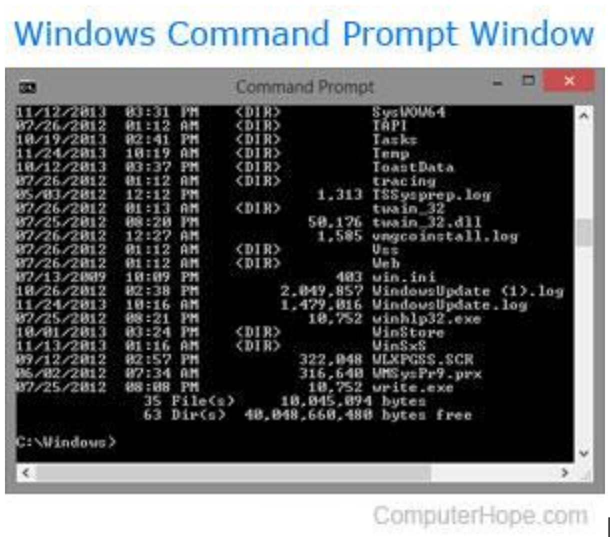
Closing or exiting the command line window

After you are done with the Windows command line, you can type **exit** to close the window.

In conclusion

You should now have a good understanding how to navigate the command line, create directories and files, rename directories and files, and delete. As mentioned earlier, there are hundreds of other commands that can be used at the command line. If you want to expand your knowledge even more, we recommend looking at the options available for each of the above commands and go through our commands overview. You can also use our search to find any command by the name of the command or by the action it performs.

DOS and Windows command line Top 10 commands



```
11/12/2013  03:31 PM    <DIR>          SysWOW64
07/26/2012  01:12 AM    <DIR>          TAPI
10/19/2013  02:41 PM    <DIR>          Tasks
11/24/2013  10:19 AM    <DIR>          Temp
10/12/2013  03:37 PM    <DIR>          ToastData
07/26/2012  01:12 AM    <DIR>          tracing
05/03/2012  12:12 PM             1,313 TSSysprep.log
07/26/2012  01:13 AM    <DIR>          twain_32
07/25/2012  08:20 PM            50,176 twain_32.dll
07/26/2012  12:27 AM             1,585 vmgcoinstall.log
07/26/2012  01:12 AM    <DIR>          Vss
07/26/2012  01:12 AM    <DIR>          Web
07/13/2009  10:09 PM               403 win.ini
10/26/2012  02:38 PM         2,049,857 WindowsUpdate (1).log
11/24/2013  10:16 AM         1,479,016 WindowsUpdate.log
07/25/2012  08:21 PM            10,752 winhlp32.exe
10/01/2013  03:24 PM    <DIR>          WinStore
11/13/2013  01:16 AM    <DIR>          WinSxS
09/12/2012  02:57 PM           322,048 WLXPGSS.SCR
06/02/2012  07:34 AM           316,640 WMSysPr9.prx
07/25/2012  08:08 PM            10,752 write.exe
              35 File(s)     10,045,094 bytes
              63 Dir(s)  40,048,660,480 bytes free

C:\Windows>
```

Below is a listing of the top 10 MS-DOS commands most commonly used and that you will most likely be using during a normal DOS session.

1. cd
2. dir
3. copy
4. del
5. edit
6. move
7. ren (rename)
8. deltree
9. cls
10. format

Top 10 command pages

Below is a listing of the top 10 MS-DOS command pages by the amount of times they have been accessed on the Computer Hope server.

1. xcopy
2. copy
3. dir
4. net
5. format
6. del
7. attrib
8. cd
9. ping
10. set

UNIT-V

UNIX

(Pronounced yoo-niks) UNIX is a popular multi-user, multitaskingoperating system (OS) developed at Bell Labs in the early 1970s. Created by just a handful of programmers, UNIX was designed to be a small, flexible system used exclusively by programmers.

Due to its portability, flexibility, and power, UNIX has become a leading operating system for workstations. Historically, it has been less popular in the personal computer market.

UNIX History

UNIX was one of the first operating systems to be written in a high-level programming language, namely C. This meant that it could be installed on virtually any computer for which a C compiler existed. This natural portability combined with its low price made it a popular choice among universities. It was inexpensive because antitrust regulations prohibited Bell Labs from marketing it as a full-scale product.

The Challenges of Cloud Integration

Bell Labs distributed the operating system in its source language form, so anyone who obtained a copy could modify and customize it for his own purposes. By the end of the 1970s, dozens of different versions of UNIX were running at various sites. After its breakup in 1982, AT&T began to market UNIX in earnest. It also began the long and difficult process of defining a standard version of UNIX.

The UNIX Standard, Trademark

Today, the trademarked "UNIX" and the "Single UNIX Specification" interface are owned by The Open Group. An operating system that is certified by The Open Group to use the UNIX trademark conforms to the Single UNIX Specification. The latest version of the certification standard is UNIX V7, aligned with the Single UNIX Specification Version 4, 2013 Edition.

According to The Open Group's Web site, "As the owner of the UNIX trademark, The Open Group has separated the UNIX trademark from any actual code stream itself, thus allowing multiple implementations. Since the introduction of the Single UNIX Specification, there has been a single, open, consensus specification that defines the requirements for a conformant UNIX system. There is also a mark, or brand, that is used to identify those products that have been certified as conforming to the Single UNIX Specification, initially UNIX 93, followed subsequently by UNIX 95, UNIX 98 and now UNIX 03. Both the specification and the UNIX trade mark are managed and held in trust for the industry by The Open Group."

Basic UNIX Commands

Examples of the basic UNIX commands include the following:

- ls (Lists files)
- ls -l (Lists files in long format)
- cd *name* (Change directory)
- cd .. (Go to directory above current)
- cp *filename1 filename2* (Copies a file)
- chmod *options filename* (Change the read, write, and execute permissions on your files)
- mkdir *name* (Creates a directory)

What is Unix ?

The Unix operating system is a set of programs that act as a link between the computer and the user.

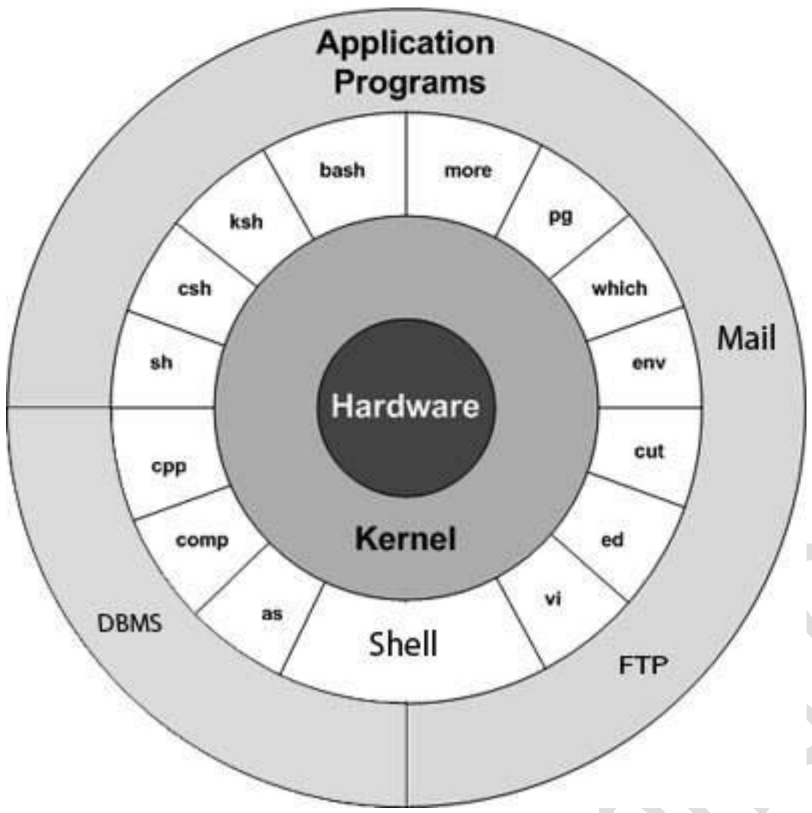
The computer programs that allocate the system resources and coordinate all the details of the computer's internals is called the **operating system** or the **kernel**.

Users communicate with the kernel through a program known as the **shell**. The shell is a command line interpreter; it translates commands entered by the user and converts them into a language that is understood by the kernel.

- Unix was originally developed in 1969 by a group of AT&T employees Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna at Bell Labs.
- There are various Unix variants available in the market. Solaris Unix, AIX, HP Unix and BSD are a few examples. Linux is also a flavor of Unix which is freely available.
- Several people can use a Unix computer at the same time; hence Unix is called a multiuser system.
- A user can also run multiple programs at the same time; hence Unix is a multitasking environment.

Unix Architecture

Here is a basic block diagram of a Unix system –



The main concept that unites all the versions of Unix is the following four basics −
Kernel − The kernel is the heart of the operating system. It interacts with the hardware and most of the tasks like memory management, task scheduling and file management.
Shell − The shell is the utility that processes your requests. When you type in a command at your terminal, the shell interprets the command and calls the program that you want. The shell uses standard syntax for all commands. C Shell, Bourne Shell and Korn Shell are the most famous shells which are available with most of the Unix variants.
Commands and Utilities − There are various commands and utilities which you can make use of in your day to day activities. **cp**, **mv**, **cat**and **grep**, etc. are few examples of commands and utilities. There are over 250 standard commands plus numerous others provided through 3rd party software. All the commands come along with various options.
Files and Directories − All the data of Unix is organized into files. All files are then organized into directories. These directories are further organized into a tree-like structure called the **filesystem**.
System Bootup
If you have a computer which has the Unix operating system installed in it, then you simply need to turn on the system to make it live.
As soon as you turn on the system, it starts booting up and finally it prompts you to log into the system, which is an activity to log into the system and use it for your day-to-day activities.
Login Unix
When you first connect to a Unix system, you usually see a prompt such as the following −

```
login:
```

To log in
Have your userid (user identification) and password ready. Contact your system administrator if you don't have these yet.

Type your userid at the login prompt, then press **ENTER**. Your userid is **case-sensitive**, so be sure you type it exactly as your system administrator has instructed.
Type your password at the password prompt, then press **ENTER**. Your password is also case-sensitive.
If you provide the correct userid and password, then you will be allowed to enter into the system. Read the information and messages that comes up on the screen, which is as follows.

```
login : amrood
amrood's password:
Last login: Sun Jun 14 09:32:32 2009 from 62.61.164.73
$
```

You will be provided with a command prompt (sometime called the **$** prompt) where you type all your commands. For example, to check calendar, you need to type the **cal** command as follows −

```
$ cal
    June 2009
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30

$
```

Change Password
All Unix systems require passwords to help ensure that your files and data remain your own and that the system itself is secure from hackers and crackers. Following are the steps to change your password −
Step 1 − To start, type password at the command prompt as shown below.
Step 2 − Enter your old password, the one you're currently using.
Step 3 − Type in your new password. Always keep your password complex enough so that nobody can guess it. But make sure, you remember it.
Step 4 − You must verify the password by typing it again.

```
$ passwd
Changing password for amrood
(current) Unix password:******
New UNIX password:*******
Retype new UNIX password:*******
passwd: all authentication tokens updated  successfully

$
```

Note − We have added asterisk (*) here just to show the location where you need to enter the current and new passwords otherwise at your system. It does not show you any character when you type.
Listing Directories and Files
All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the filesystem.
You can use the **ls** command to list out all the files or directories available in a directory. Following is the example of using **ls** command with **-l** option.

```
$ ls -l
total 19621
drwxrwxr-x  2 amrood amrood      4096 Dec 25 09:59 uml
-rw-rw-r--  1 amrood amrood      5341 Dec 25 08:38 uml.jpg
```

```
drwxr-xr-x  2 amrood amrood      4096 Feb 15  2006 univ
drwxr-xr-x  2 root   root        4096 Dec  9  2007 urlspedia
-rw-r--r--  1 root   root      276480 Dec  9  2007 urlspedia.tar
drwxr-xr-x  8 root   root        4096 Nov 25  2007 usr
-rwxr-xr-x  1 root   root        3192 Nov 25  2007 webthumb.php
-rw-rw-r--  1 amrood amrood     20480 Nov 25  2007 webthumb.tar
-rw-rw-r--  1 amrood amrood      5654 Aug  9  2007 yourfile.mid
-rw-rw-r--  1 amrood amrood    166255 Aug  9  2007 yourfile.swf

$
```

Here entries starting with **d.....** represent directories. For example, uml, univ and urlspedia are directories and rest of the entries are files.

Who Are You?

While you're logged into the system, you might be willing to know : **Who am I**?

The easiest way to find out "who you are" is to enter the **whoami** command −

```
$ whoami
 amrood

$
```

Try it on your system. This command lists the account name associated with the current login. You can try **who am i** command as well to get information about yourself.

Who is Logged in?

Sometime you might be interested to know who is logged in to the computer at the same time.

There are three commands available to get you this information, based on how much you wish to know about the other users: **users**, **who**, and **w**.

```
$ users
 amrood bablu qadir

$ who
amrood ttyp0 Oct 8 14:10 (limbo)
bablu  ttyp2 Oct 4 09:08 (calliope)
qadir  ttyp4 Oct 8 12:09 (dent)

$
```

Try the **w** command on your system to check the output. This lists down information associated with the users logged in the system.

Logging Out

When you finish your session, you need to log out of the system. This is to ensure that nobody else accesses your files.

To log out

Just type the **logout** command at the command prompt, and the system will clean up everything and break the connection.

System Shutdown

The most consistent way to shut down a Unix system properly via the command line is to use one of the following commands −

S.No.	Command & Description

1	**halt** Brings the system down immediately
2	**init 0** Powers off the system using predefined scripts to synchronize and clean up the system prior to shutting down
3	**init 6** Reboots the system by shutting it down completely and then restarting it
4	**poweroff** Shuts down the system by powering off
5	**reboot** Reboots the system
6	**shutdown** Shuts down the system

In Unix, there are three basic types of files −
Ordinary Files − An ordinary file is a file on the system that contains data, text, or program instructions. In this tutorial, you look at working with ordinary files.
Directories − Directories store both special and ordinary files. For users familiar with Windows or Mac OS, Unix directories are equivalent to folders.
Special Files − Some special files provide access to hardware such as hard drives, CD-ROM drives, modems, and Ethernet adapters. Other special files are similar to aliases or shortcuts and enable you to access a single file using different names.
Hidden Files
An invisible file is one, the first character of which is the dot or the period character (.). Unix programs (including the shell) use most of these files to store configuration information.
Some common examples of the hidden files include the files −
.profile − The Bourne shell (sh) initialization script
.kshrc − The Korn shell (ksh) initialization script
.cshrc − The C shell (csh) initialization script
.rhosts − The remote shell configuration file
Single dot (.) − This represents the current directory.
Double dot (..) − This represents the parent directory.