**Chapter-11**

**Operating Systems(OS)**

# **Introduction**

The operating system acts as an interface between the hardware and the programs requesting I/O. It is the most fundamental of all system software programs.

Responsibilities of the OS include:

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

There are two types of software:

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

## **Need of Operating System**

* OS as a platform for Application programs: Operating system provides a platform, on top of which, other programs, called application programs can run. These application programs help the users to perform a specific task easily. It acts as an interface between the computer and the user. It’s designed in such a manner that it operates controls and executes various applications on the computer.
* Managing Input-Output unit: Operating System also allows the computer to manage its own resources such as memory, monitor, keyboard, printer etc. Management of these resources is required for an effective utilization. The operating system controls the various system input-output resources and allocates them to the users or programs as per their requirement.
* Consistent user interface: Operating System provides the user an easy-to-work user interface, so the user does not have to learn a different UI every time and can focus on the content and be productive as quickly as possible. Operating System provides templates, UI components to make the working of a computer, easy for the user.
* Multitasking: Operating System manages memory and allow multiple programs to run in their own space and even communicate with each other through shared memory. Multitasking gives users a good experience as they can perform several tasks on a computer at a time.

## **Functions of an Operating System**

An operating system has variety of functions to perform. Some of the prominent functions of an operating system can be broadly outlined as:

* [Processor Management](https://www.geeksforgeeks.org/gate-notes-operating-system-process-management-introduction/):

This deals with management of the Central Processing Unit (CPU). The operating system takes care of the allotment of CPU time to different processes. When a process finishes its CPU processing after executing for the allotted time period, this is called scheduling. There is various type of scheduling techniques that are used by the operating systems:

* 1. [Shortest Job First (SJF](https://www.geeksforgeeks.org/operating-system-shortest-job-first-scheduling-predicted-burst-time/)): Process, which need the shortest CPU time are scheduled first.
  2. [Round Robin Scheduling](https://www.geeksforgeeks.org/program-round-robin-scheduling-set-1/): Each process is assigned a fixed CPU execution time in cyclic way.
  3. [Priority Based scheduling (Non Pre-emptive)](https://www.geeksforgeeks.org/operating-system-priority-scheduling-different-arrival-time-set-2/): In this scheduling, processes are scheduled according to their priorities, i.e., highest priority process is schedule first. If priorities of two processes match, then schedule according to arrival time
* Device Management:

The Operating System communicates with hardware and the attached devices and maintains a balance between them and the CPU. This is all the more important because the CPU processing speed is much higher than that of I/O devices. In order to optimize the CPU time, the operating system employs two techniques – Buffering and Spooling.

* Buffering:

In this technique, input and output data is temporarily stored in Input Buffer and Output Buffer. Once the signal for input or output is sent to or from the CPU respectively, the operating system through the device controller moves the data from the input device to the input buffer and for the output device to the output buffer. In case of input, if the buffer is full, the operating system sends a signal to the program, which processes the data stored in the buffer. When the buffer becomes empty, the program informs the operating system, which reloads the buffer, and the input operation continues.

* [Spooling (Simultaneous Peripheral Operation on Line)](https://www.geeksforgeeks.org/what-exactly-spooling-is-all-about/):

This is a device management technique used for processing of different tasks on the same input/output device. When there are various users on a network sharing the same resource then it can be a possibility, that more than one user might give it a command at the same point of time. So, the operating system temporarily stores the data of every user on the hard disk of the computer to which the resource is attached. The individual user need not wait for the execution process to be completed. Instead, the operating system sends the data from the hard disk to the resource one by one.  
Example: printer

* [Memory management](https://www.geeksforgeeks.org/operating-systems-gq/memory-management-gq/):

In a computer, both the CPU and the I/O devices interact with the memory. When a program needs to be executed, it is loaded onto the main memory till the execution is completed. Thereafter that memory space is freed and is available for other programs. The common memory management techniques used by the operating system are Partitioning and Virtual Memory.

* Partitioning:  
  The total memory is divided into various partitions of same size or different sizes. This helps to accommodate number of programs in the memory. The partition can be fixed i.e. remains same for all the programs in the memory or variable i.e. memory is allocated when a program is loaded on to the memory. The later approach causes less wastage of memory but in due course of time, it may become fragmented.
* [Virtual Memory](https://www.geeksforgeeks.org/virtual-memory-operating-systems/):  
  This is a technique used by the operating systems which allow the user can load the programs which are larger than the main memory of the computer. In this technique the program is executed even if the complete program can not be loaded inside the main memory leading to efficient memory utilization.
* File Management:

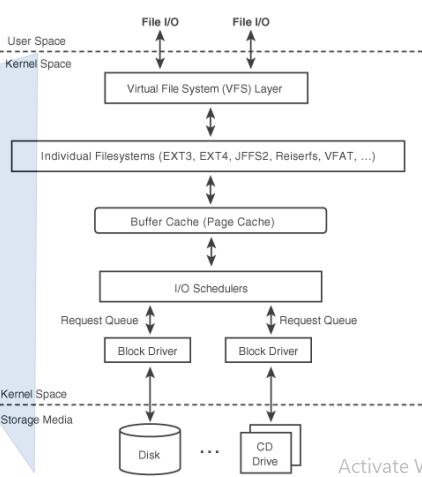
The operating System manages the files, folders and directory systems on a computer. Any data on a computer is stored in the form of files and the operating system keeps information about all of them using File Allocation Table (FAT). The FAT stores general information about files like filename, type (text or binary), size, starting address and access mode (sequential/indexed sequential/direct/relative). The file manager of the operating system helps to create, edit, copy, allocate memory to the files and also updates the FAT. The operating system also takes care that files are opened with proper access rights to read or edit them.

# **OS Organization**

* Layered
* Monolithic
* Micro kernel
* Hybrid Kernel (Distributed)

## **Layered**

* The operating system is split into various layers in the layered operating system and each of the layers have different functionalities. This type of operating system was created as an improvement over the early monolithic systems.
* Layering provides a distinct advantage in an operating system. All the layers can be defined separately and interact with each other as required. In addition, it is easier to create, maintain and update the system if it is done in the form of layers. Change in one-layer specification does not affect the rest of the layers.
* Each of the layers in the operating system can only interact with the layers that are above and below it. The lowest layer handles the hardware and the uppermost layer deals with the user applications.



**Figure 1: Layered Architecture**

## **Monolithic**

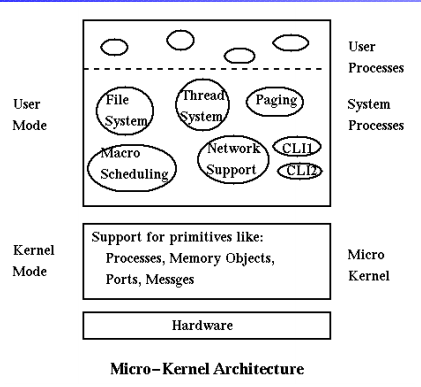
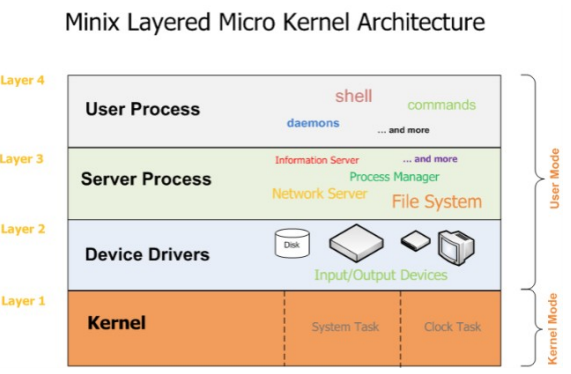
A monolithic kernel is an operating system architecture where the entire operating system is working in kernel space. The monolithic model differs from other operating system architectures (such as the microkernel architecture) in that it alone defines a high-level virtual interface over computer hardware. A set of primitives or system calls implement all operating system services such as process management, concurrency, and memory management. Device drivers can be added to the kernel as modules.

**Figure2: Monolithic Architecture**

## **Micro Kernel**

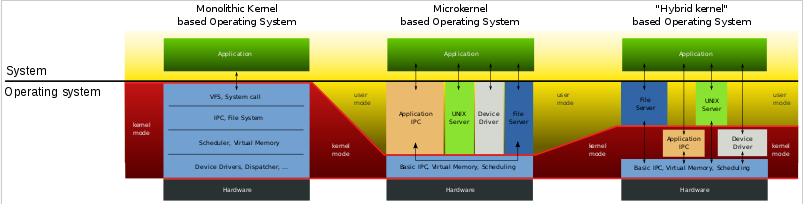
Microkernel is one of the classification of the kernel. Being a kernel, it manages all system resources. However, in a microkernel, the user services and kernel services are implemented in different address space. The user services are kept in user address space, and kernel services are kept under kernel address space, thus also reduces the size of kernel and size of operating system as well.

It provides minimal services of process and memory management. The communication between client program/application and services running in user address space is established through message passing, reducing the speed of execution microkernel. The Operating System remains unaffected as user services and kernel services are isolated so if any user service fails it does not affect kernel service. Thus it adds to one of the advantages in a microkernel. It is easily extendable i.e. if any new services are to be added they are added to user address space and hence requires no modification in kernel space. It is also portable, secure and reliable.



**Figure 3: Micro Kernel Architecture**

## **Hybrid Kernel**

A hybrid kernel runs some services in the kernel space to reduce the performance overhead of a traditional microkernel, while still running kernel code as servers in the user space. For instance, a hybrid kernel design may keep the virtual files system and bus controllers inside the kernel and the file system drivers and storage drivers as user mode programs outside the kernel. Such a design keeps the performance and design principles of a monolithic kernel.   
  
The Microsoft NT kernel is a well-known example of a hybrid kernel that powers Windows NT, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008 and Windows 7. It is referred to as a monolithic kernel as emulation subsystems run in the user mode server processes. One of the most important thing about it is its structure, which is a collection of modules that communicate via well-known interfaces, with a small microkernel limited to core functions such as first-level interrupt handling, thread scheduling and synchronization primitives. This allows for the possibility of using either direct procedure calls or inter-process communication to communicate between modules, and hence for the potential location of modules in different address spaces.

**Figure 4: Hybrid Architecture**

# **Distributed Operating System**

**Distributed Operating System** is a model where distributed applications are running on multiple computers linked by communications. A distributed operating system is an extension of the network operating system that supports higher levels of communication and integration of the machines on the network.

This system looks to its users like an ordinary centralized operating system but runs on multiple, independent central processing units (CPUs).

These systems are referred as *loosely coupled systems*where each processor has its own local memory and processors communicate with one another through various communication lines, such as high speed buses or telephone lines. By loosely coupled systems, we mean that such computers possess no hardware connections at the CPU - memory bus level, but are connected by external interfaces that run under the control of software.

The Distributed Os involves a collection of autonomous computer systems, capable of communicating and cooperating with each other through a LAN / WAN. A Distributed Os provides a virtual machine abstraction to its users and wide sharing of resources like as computational capacity, I/O and files etc.

The structure shown in fig contains a set of individual computer systems and workstations connected via communication systems, but by this structure we cannot say it is a distributed system because it is the software, not the hardware, that determines whether a system is distributed or not.

The users of a true distributed system should not know, on which machine their programs are running and where their files are stored. LOCUS and MICROS are the best examples of distributed operating systems.

Using LOCUS operating system it was possible to access local and distant files in uniform manner. This feature enabled a user to log on any node of the network and to utilize the resources in a network without the reference of his/her location. MICROS provided sharing of resources in an automatic manner. The jobs were assigned to different nodes of the whole system to balance the load on different nodes.

**Below given are some of the examples of distributed operating systems:**

l. IRIX operating system; is the implementation of UNIX System V, Release 3 for Silicon Graphics multiprocessor workstations.

2.  DYNIX operating system running on Sequent Symmetry multiprocessor computers.

3.  AIX operating system for IBM RS/6000 computers.

4.  Solaris operating system for SUN multiprocessor workstations.

5.  Mach/OS is a multithreading and multitasking UNIX compatible operating system;

6.  OSF/1 operating system developed by Open Foundation Software: UNIX compatible.

**Distributed systems provide the following advantages:**

1 Sharing of resources.

2 Reliability.

3 Communication.

4 Computation speedup

Distributed systems are potentially more reliable than a central system because if a system has only one instance of some critical component, such as a CPU, disk, or network interface, and that component fails, the system will go down. When there are multiple instances, the system may be able to continue in spite of occasional failures. In addition to hardware failures, one can also consider software failures. Distributed systems allow both hardware and software errors to be dealt with.

A distributed system is a set of computers that communicate and collaborate each other using software and hardware interconnecting components. Multiprocessors (MIMD computers using shared memory architecture), multicomputers connected through static or dynamic interconnection networks (MIMD computers using message passing architecture) and workstations connected through local area network are examples of such distributed systems.

A distributed system is managed by a distributed operating system. A distributed operating system manages the system shared resources used by multiple processes, the process scheduling activity (how processes are allocating on available processors), the communication and synchronization between running processes and so on. The software for parallel computers could be also tightly coupled or loosely coupled. The loosely coupled software allows computers and users of a distributed system to be independent each other but having a limited possibility to cooperate. An example of such a system is a group of computers connected through a local network. Every computer has its own memory, hard disk. There are some shared resources such files and [printer](http://ecomputernotes.com/fundamental/input-output-and-memory/what-is-a-printer-and-what-are-the-different-types-of-printers)s. If the interconnection network broke down, individual computers could be used but without some features like printing to a non-local printer.

# **Basic OS Architecture**

A general purpose OS is composed of:

– Process manager

* + multiplexes the CPU time between the multiple execution units (processes)

– memory manager

* + controls, manages and multiplexes the access to physical and virtual memory

– Inter-process communication

* + Implements and handles mechanism for processes to communicate

– I/O manager

* + Manages communication with peripheral (keyboard/screen, disk, network)

– User interface

* + command line interpreter
  + GUI

– File System manager

* + manages and organizes data available on disks (file systems)

– Function calls

* + Programming functions that allow applications to use OS services (memory, disk, I/O)

# **Virtual Machine**

A virtual machine (VM) is a software program or operating system that not only exhibits the behaviour of a separate computer, but is also capable of performing tasks such as running applications and programs like a separate computer. A virtual machine, usually known as a guest is created within another computing environment referred as a "host." Multiple virtual machines can exist within a single host at one time.

A virtual machine is also known as a guest.

Virtual machines are becoming more common with the evolution of virtualization technology. Virtual machines are often created to perform certain tasks that are different than tasks performed in a host environment.

Virtual machines are implemented by software emulation methods or hardware virtualization techniques. Depending on their use and level of correspondence to any physical computer, virtual machines can be divided into two categories:

1. System Virtual Machines: A system platform that supports the sharing of the host computer's physical resources between multiple virtual machines, each running with its own copy of the operating system. The virtualization technique is provided by a software layer known as a hypervisor, which can run either on bare hardware or on top of an operating system.
2. Process Virtual Machine: Designed to provide a platform-independent programming environment that masks the information of the underlying hardware or operating system and allows program execution to take place in the same way on any given platform.

Some of the advantages of a virtual machine include:

* Allows multiple operating system environments on a single physical computer without any intervention
* Virtual machines are widely available and are easy to manage and maintain.
* Offers application provisioning and disaster recovery options

Some of the drawbacks of virtual machines include:

* They are not as efficient as a physical computer because the hardware resources are distributed in an indirect way.
* Multiple VMs running on a single physical machine can deliver unstable performance

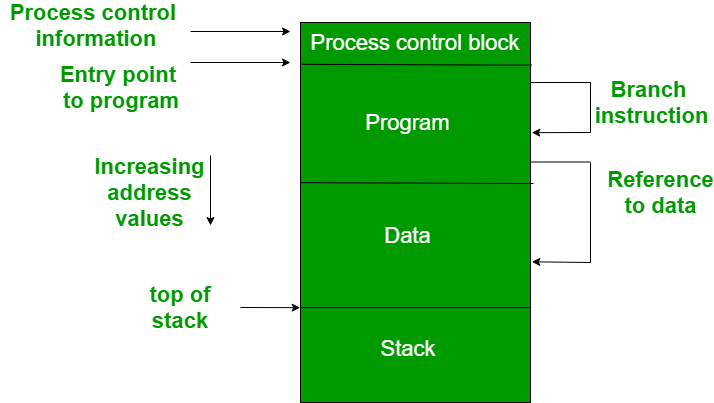
# **Requirements of memory management system**

Memory management keeps track of the status of each memory location, whether it is allocated or free. It allocates the memory dynamically to the programs at their request and frees it for reuse when it is no longer needed. Memory management meant to satisfy some requirements that we should keep in mind.

These Requirements of memory management are:

Relocation – The available memory is generally shared among a number of processes in a multiprogramming system, so it is not possible to know in advance which other programs will be resident in main memory at the time of execution of his program. Swapping the active processes in and out of the main memory enables the operating system to have a larger pool of ready-to-execute process.

When a program gets swapped out to a disk memory, then it is not always possible that when it is swapped back into main memory then it occupies the previous memory location, since the location may still be occupied by another process. We may need to relocate the process to a different area of memory. Thus there is a possibility that program may be moved in main memory due to swapping.

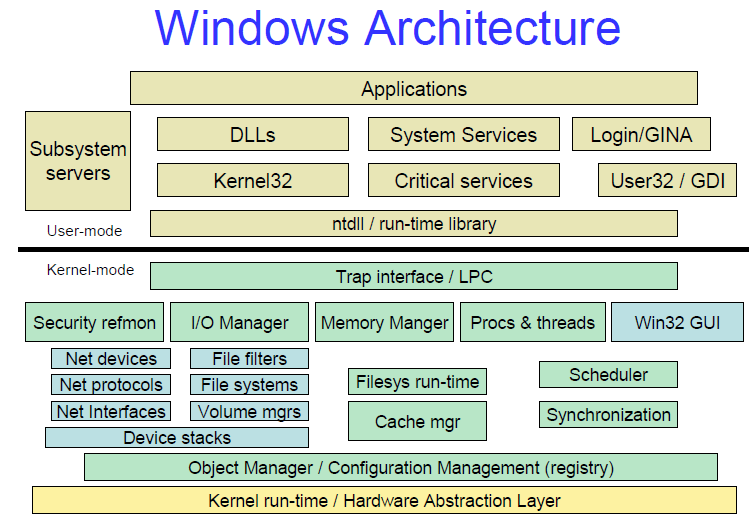


**Figure: Memory layout**

1. The figure depicts a process image. The process image is occupying a continuous region of main memory. The operating system will need to know many things including the location of process control information, the execution stack, and the code entry. Within a program, there are memory references in various instructions and these are called logical addresses.  
   After loading of the program into main memory, the processor and the operating system must be able to translate logical addresses into physical addresses. Branch instructions contain the address of the next instruction to be executed. Data reference instructions contain the address of byte or word of data referenced.
2. **Protection –** There is always a danger when we have multiple programs at the same time as one program may write to the address space of another program. So every process must be protected against unwanted interference when other process tries to write in a process whether accidental or incidental. Between relocation and protection requirement a trade-off occurs as the satisfaction of relocation requirement increases the difficulty of satisfying the protection requirement.  
   Prediction of the location of a program in main memory is not possible, that’s why it is impossible to check the absolute address at compile time to assure protection. Most of the programming language allows the dynamic calculation of address at run time. The memory protection requirement must be satisfied by the processor rather than the operating system because the operating system can hardly control a process when it occupies the processor. Thus it is possible to check the validity of memory references.
3. **Sharing –** A protection mechanism must have to allow several processes to access the same portion of main memory. Allowing each processes access to the same copy of the program rather than have their own separate copy has an advantage.  
   For example, multiple processes may use the same system file and it is natural to load one copy of the file in main memory and let it shared by those processes. It is the task of Memory management to allow controlled access to the shared areas of memory without compromising the protection. Mechanisms are used to support relocation supported sharing capabilities.
4. **Logical organization –** Main memory is organized as linear or it can be a one-dimensional address space which consists of a sequence of bytes or words. Most of the programs can be organized into modules, some of those are unmodifiable (read-only, execute only) and some of those contain data that can be modified. To effectively deal with a user program, the operating system and computer hardware must support a basic module to provide the required protection and sharing. It has the following advantages:  
   * Modules are written and compiled independently and all the references from one module to another module are resolved by `the system at run time.
   * Different modules are provided with different degrees of protection.
   * There are mechanisms by which modules can be shared among processes. Sharing can be provided on a module level that lets the user specify the sharing that is desired.
5. **Physical organization –** The structure of computer memory has two levels referred to as main memory and secondary memory. Main memory is relatively very fast and costly as compared to the secondary memory. Main memory is volatile. Thus secondary memory is provided for storage of data on a long-term basis while the main memory holds currently used programs. The major system concern between main memory and secondary memory is the flow of information and it is impractical for programmers to understand this for two reasons:  
   * The programmer may engage in a practice known as overlaying when the main memory available for a program and its data may be insufficient. It allows different modules to be assigned to the same region of memory. One disadvantage is that it is time-consuming for the programmer.
   * In a multiprogramming environment, the programmer does not know how much space will be available at the time of coding and where that space will be located inside the memory.

# **Basics of Windows Architecture**

Let's dig deeper into the internal structure and the role each key operating system component plays. The figure below is more detailed diagram of the core Windows system

architecture and components. Note that it still does not show all components (networking in particular).

**Figure 5: simplified windows Architecture**

First notice the line dividing the user-mode and kernel-mode parts of the Windows operating system. The boxes above the line represent user-mode processes, and the components below the line are kernel-mode operating system services.

User and kernel modes are two processor access modes, where a kernel mode refers to a mode of execution privilege that grants access to system memory and all CPU instructions.

User mode is a less privileged processor mode than kernel mode. It uses well-defined operating system application program interfaces (APIs) to request system services. **A User mode** process:

* Have no direct access to hardware or kernel memory (Only kernel mode processes can access kernel resources as a way of protection).
* Is limited to an assigned address space.
* Can be paged out of physical memory into virtual RAM on a hard disk.
* Process at a lower priority than kernel mode components (OS components). Which means that the OS does not slow down or have to wait while an application finishes processing.
* Cannot access another user process address space (Unless opened a handle to the process, which means passing through security access check).

Before we continue let's just have a quick definition for virtual memory. Virtual memory*,*also known as virtual RAM, allows hard disk space to be used as if it were additional memory. In this manner, the user mode processes have access to more memory than is actually available to them.

Top right of the figure shows theenvironment subsystems**. Environment subsystems** provide exposed, documented interface between applications and Windows native APIs (undocumented APIs). The environment subsystem translates environment-specific instructions from an application into instructions that the Executive Services (First layer of the OS kernel) can carry out. Each subsystem can provide access to different subsets of the native services in Windows.

Environment subsystems include POSIX, OS/2 and Windows subsystems. The Windows subsystem major components consist of the environment subsystem process (Csrss.exe which you can see tuning in the task manager), the kernel-mode device driver (Win32k.sys), subsystem DLLs (such as Kernel32.dll, Advapi32.dll, User32.dll, and Gdi32.dll) and Graphics device drivers.

In the user mode and just above the line that divides the user and kernel modes is the Ntdll.dll. **Ntdll.dll** is a special system support library primarily for the use of subsystem DLLs.

Now let's move on to the kernel mode. Kernel mode is the privileged mode of operation in which the code has direct access to all hardware and all memory, including the address spaces of all user mode processes**. Kernel mode** components:

* Can access hardware directly.
* Can access all of the memory on the computer.
* Are not moved to the virtual memory page file on the hard disk.
* Process at a higher priority than user mode processes.

The kernel mode in Windows is comprised of the Windows Executive, which includes the Executive Services, the kernel, and the hardware abstraction layer (HAL).

**The Windows executive** is the upper layer of Ntoskrnl.exe. (The kernel is the lower layer.)The executive provides core OS services. The executive contains major components such as various modules that manage I/O, objects, security, processes, inter-process communications (IPC), virtual memory, and window and graphics management. It also includes device drivers (defined in previous post) functions.

**The kernel** consists of a set of functions in Ntoskrnl.exe provides the most basic operating system services, such as thread scheduling, first-level interrupt handling, and deferred procedure calls. The kernel resides between the Executive Services and HAL layers.

The other major job of the kernel is to abstract or isolate the executive and device drivers from variations between the hardware architectures supported by Windows.

One of the crucial elements of the Windows design is its portability across a variety of hardware platforms. **The hardware abstraction layer (HAL)** is a key part of making this portability possible. The HAL is a loadable kernel-mode module (Hal.dll) enables the same operating system to run on different platforms with different processors.

Also part of the kernel is the **device drivers**. Device drivers in Windows don't manipulate hardware directly, but rather they call functions in the HAL to interface with the hardware.

# **Windows OS Features**

**Extensibility**: Code must be able to grow and change as market requirements change.

• **Portability**: The system must be able to run on multiple hardware architectures and must be able to move with relative ease to new ones as market demands dictate.

• **Dependability**: Protection against internal malfunction and external tampering.

• Applications should not be able to harm the OS or other running applications.

• **Compatibility**: User interface and APIs should be compatible with older versions of

Windows as well as older operating systems such as MS-DOS.

• It should also interoperate well with UNIX, OS/2, and NetWare.

• **Performance**: Within the constraints of the other design goals, the system should be as fast and responsive as possible on each hardware platform.

# **Processes and Threads Basics**

An application consists of one or more processes. A process, in the simplest terms, is an executing program. One or more threads run in the context of the process. A thread is the basic unit to which the operating system allocates processor time. A thread can execute any part of the process code, including parts currently being executed by another thread.

A job object allows groups of processes to be managed as a unit. Job objects are namable, securable, sharable objects that control attributes of the processes associated with them. Operations performed on the job object affect all processes associated with the job object.

A thread pool is a collection of worker threads that efficiently execute asynchronous callbacks on behalf of the application. The thread pool is primarily used to reduce the number of application threads and provide management of the worker threads.

A fiber is a unit of execution that must be manually scheduled by the application. Fibers run in the context of the threads that schedule them.

User-mode scheduling (UMS) is a lightweight mechanism that applications can use to schedule their own threads. UMS threads differ from fibers in that each UMS thread has its own thread context instead of sharing the thread context of a single thread.

For more Details and for more information, see the following topics:

* [Multitasking](https://docs.microsoft.com/en-us/windows/desktop/procthread/multitasking)
* [Scheduling](https://docs.microsoft.com/en-us/windows/desktop/procthread/scheduling)
* [Multiple Threads](https://docs.microsoft.com/en-us/windows/desktop/procthread/multiple-threads)
* [Child Processes](https://docs.microsoft.com/en-us/windows/desktop/procthread/child-processes)
* [Thread Pools](https://docs.microsoft.com/en-us/windows/desktop/procthread/thread-pools)
* [Job Objects](https://docs.microsoft.com/en-us/windows/desktop/procthread/job-objects)
* [User-Mode Scheduling](https://docs.microsoft.com/en-us/windows/desktop/procthread/user-mode-scheduling)
* [Fibers](https://docs.microsoft.com/en-us/windows/desktop/procthread/fibers)

# **Process Table and Process Control Block (PCB)**

While creating a process the operating system performs several operations. To identify these process, it must identify each process, hence it assigns a process identification number (PID) to each process. As the operating system supports multi-programming, it needs to keep track of all the processes. For this task, the process control block (PCB) is used to track the process’s execution status. Each block of memory contains information about the process state, program counter, stack pointer, status of opened files, scheduling algorithms, etc. All these information is required and must be saved when the process is switched from one state to another. When the process made transitions from one state to another, the operating system must update information in the process’s PCB.

A process control block (PCB) contains information about the process, i.e. registers, quantum, priority, etc. The process table is an array of PCB’s, that means logically contains a PCB for all of the current processes in the system.

Pointer – It is a stack pointer which is required to be saved when the process is switched from one state to another to retain the current position of the process.

Process state – It stores the respective state of the process.

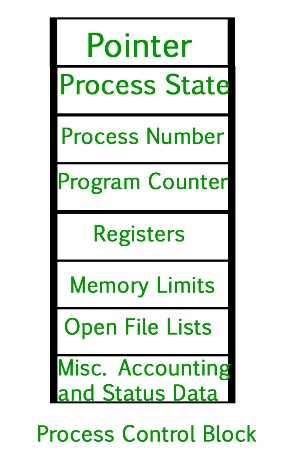
Process number – Every process is assigned with a unique id known as process ID or PID which stores the process identifier.

Program counter – It stores the counter which contains the address of the next instruction that is to be executed for the process.

Register – These are the CPU registers which includes: accumulator, base, registers and general purpose registers.

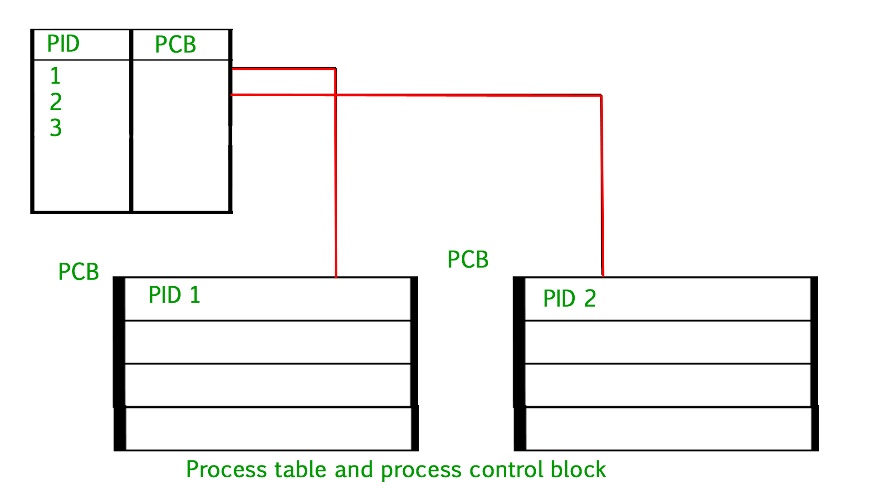
Memory limits – This field contains the information about memory management system used by operating system. This may include the page tables, segment tables etc.

Open files list – This information includes the list of files opened for a process.



Miscellaneous accounting and status data – This field includes information about the amount of CPU used, time constraints, jobs or process number, etc.

The process control block stores the register content also known as execution content of the processor when it was blocked from running. This execution content architecture enables the operating system to restore a process’s execution context when the process returns to the running state. When the process made transitions from one state to another, the operating system update its information in the process’s PCB. The operating system maintains pointers to each process’s PCB in a process table so that it can access the PCB quickly.



# **Kernel-Mode Vs User-Mode**

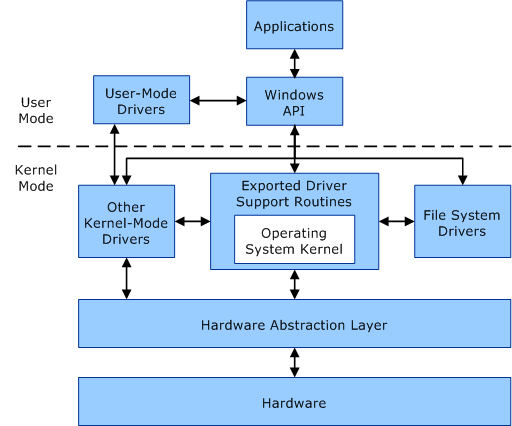
A processor in a computer running Windows has two different modes: user mode and kernel mode. The processor switches between the two modes depending on what type of code is running on the processor. Applications run in user mode, and core operating system components run in kernel mode. While many drivers run in kernel mode, some drivers may run in user mode.

When you start a user-mode application, Windows creates a process for the application. The process provides the application with a private virtual address space and a private handle table. Because an application's virtual address space is private, one application cannot alter data that belongs to another application. Each application runs in isolation, and if an application crashes, the crash is limited to that one application. Other applications and the operating system are not affected by the crash.

In addition to being private, the virtual address space of a user-mode application is limited. A processor running in user mode cannot access virtual addresses that are reserved for the operating system. Limiting the virtual address space of a user-mode application prevents the application from altering, and possibly damaging, critical operating system data.

All code that runs in kernel mode shares a single virtual address space. This means that a kernel-mode driver is not isolated from other drivers and the operating system itself. If a kernel-mode driver accidentally writes to the wrong virtual address, data that belongs to the operating system or another driver could be compromised. If a kernel-mode driver crashes, the entire operating system crashes.

This diagram illustrates communication between user-mode and kernel-mode components.



**Figure 6: user-mode and kernel-mode components**

# **Windows Services**

Microsoft Windows services, formerly known as NT services, enable you to create long-running executable applications that run in their own Windows sessions. These services can be automatically started when the computer boots, can be paused and restarted, and do not show any user interface. These features make services ideal for use on a server or whenever you need long-running functionality that does not interfere with other users who are working on the same computer. You can also run services in the security context of a specific user account that is different from the logged-on user or the default computer account. For more information about services and Windows sessions, see the Windows SDK documentation.

You can easily create services by creating an application that is installed as a service. For example, suppose you want to monitor performance counter data and react to threshold values. You could write a Windows Service application that listens to the performance counter data, deploy the application, and begin collecting and analysing data.

You create your service as a Microsoft Visual Studio project, defining code within it that controls what commands can be sent to the service and what actions should be taken when those commands are received. Commands that can be sent to a service include starting, pausing, resuming, and stopping the service; you can also execute custom commands.

After you create and build the application, you can install it by running the command-line utility InstallUtil.exe and passing the path to the service's executable file. You can then use the **Services Control Manager** to start, stop, pause, resume, and configure your service. You can also accomplish many of these same tasks in the **Services** node in **Server Explorer** or by using the ServiceController class.

For more info

<https://docs.microsoft.com/en-us/dotnet/framework/windows-services/introduction-to-windows-service-applications>

# **Windows API**

The Windows API is Microsoft's core set of application programming interfaces (APIs) available in the Microsoft Windows operating systems. It was formerly called WinAPI or Win32 API. Ex.

**CreateProcess** (&ProcHandle, Access, SectionHandle, DebugPort, ExceptionPort, …)

**CreateThread** (&ThreadHandle, ProcHandle, Access, ThreadContext, bCreateSuspended, …)

**AllocateVirtualMemory** (ProcHandle, Addr, Size, Type, Protection, …)

**MapViewOfSection** (SectionHandle, ProcHandle, Addr, Size, Protection, …)

**ReadVirtualMemory** (ProcHandle, Addr, Size, …)

**DuplicateObject** (srcProcHandle, srcObjHandle, dstProcHandle, dstHandle, Access, Attributes, Options)

For more details

<https://docs.microsoft.com/en-us/windows/desktop/apiindex/windows-api-list>

# **Basics of System Calls**

In computing, a **system call** is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. A system call is a way for programs to **interact with the operating system**. A computer program makes a system call when it makes a request to the operating system’s kernel. System call **provides** the services of the operating system to the user programs via Application Program Interface(API). It provides an interface between a process and operating system to allow user-level processes to request services of the operating system. System calls are the only entry points into the kernel system. All programs needing resources must use system calls.

**Services Provided by System Calls:**

1. Process creation and management
2. Main memory management
3. File Access, Directory and File system management
4. Device handling(I/O)
5. Protection
6. Networking, etc.

**Types of System Calls:** There are 5 different categories of system calls –

* 1. **Process control:** end, abort, create, terminate, allocate and free memory.
  2. **File management:** create, open, close, delete, read file etc.
  3. Device management
  4. Information maintenance
  5. Communication

**Examples of Windows and Unix System Calls –**

|  |  |  |
| --- | --- | --- |
|  | **WINDOWS** | **UNIX** |
| Process Control | CreateProcess() ExitProcess() WaitForSingleObject() | fork() exit() wait() |
| File Manipulation | CreateFile() ReadFile() WriteFile() CloseHandle() | open() read() write() close() |
| Device Manipulation | SetConsoleMode() ReadConsole() WriteConsole() | ioctl() read() write() |
| Information Maintenance | GetCurrentProcessID() SetTimer() Sleep() | getpid() alarm() sleep() |
| Communication | CreatePipe() CreateFileMapping() MapViewOfFile() | pipe() shmget() mmap() |
| Protection | SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup() | chmod() umask() chown() |

# **MinWin**

* **MinWin is first step at creating architectural partitions**
  + Can be built, booted and tested separately from the rest of the system
  + Higher layers can evolve independently
  + *An engineering process improvement, not a microkernel NT!*
* **MinWin was defined as set of components required to boot and access network**
  + Kernel, file system driver, TCP/IP stack, device drivers, services
  + No servicing, WMI, graphics, audio or shell, etc.
* **MinWin footprint:** 
  + 150 binaries, 25MB on disk, 40MB in-memory

# **Virtual Memory**

Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program generated addresses are translated automatically to the corresponding machine addresses.

The size of virtual storage is limited by the addressing scheme of the computer system and amount of secondary memory is available not by the actual number of the main storage locations.

It is a technique that is implemented using both hardware and software. It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory.

1. All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time. This means that a process can be swapped in and out of main memory such that it occupies different places in main memory at different times during the course of execution.
2. A process may be broken into number of pieces and these pieces need not be continuously located in the main memory during execution. The combination of dynamic run-time address translation and use of page or segment table permits this.

If these characteristics are present then, it is not necessary that all the pages or segments are present in the main memory during execution. This means that the required pages need to be loaded into memory whenever required. Virtual memory is implemented using Demand Paging or Demand Segmentation.

**Demand Paging:**

The process of loading the page into memory on demand (whenever page fault occurs) is known as demand paging.

The process includes the following steps:

[virtual_mem](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/VirtualDiagram-1.png)

**Figure 7: address Translation**

1. If CPU try to refer a page that is currently not available in the main memory, it generates an interrupt indicating memory access fault.
2. The OS puts the interrupted process in a blocking state. For the execution to proceed the OS must bring the required page into the memory.
3. The OS will search for the required page in the logical address space.
4. The required page will be brought from logical address space to physical address space. The page replacement algorithms are used for the decision making of replacing the page in physical address space.
5. The page table will have updated accordingly.
6. The signal will be sent to the CPU to continue the program execution and it will place the process back into ready state.

Hence whenever a page fault occurs these steps are followed by the operating system and the required page is brought into memory.

**Advantages:**

* More processes may be maintained in the main memory: Because we are going to load only some of the pages of any particular process, there is room for more processes. This leads to more efficient utilization of the processor because it is more likely that at least one of the more numerous processes will be in the ready state at any particular time.
* A process may be larger than all of main memory: One of the most fundamental restrictions in programming is lifted. A process larger than the main memory can be executed because of demand paging. The OS itself loads pages of a process in main memory as required.
* It allows greater multiprogramming levels by using less of the available (primary) memory for each process.

**Page Fault Service Time:**

The time taken to service the page fault is called as page fault service time. The page fault service time includes the time taken to perform all the above six steps.

Let Main memory access time is: m

Page fault service time is: s

Page fault rate is : p

Then, Effective memory access time = (p\*s) + (1-p)\*m

**Swapping:**

Swapping a process out means removing all of its pages from memory, or marking them so that they will be removed by the normal page replacement process. Suspending a process ensures that it is not runnable while it is swapped out. At some later time, the system swaps back the process from the secondary storage to main memory. When a process is busy swapping pages in and out then this situation is called thrashing.

[swaping](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/VirtualDiagram-2.png)

**Figure 8: Page swapping**

**Thrashing:**

[virtual_mem_2](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/VirtualDiagram-3.png)

**Figure 9: Thrashing**

At any given time, only few pages of any process are in main memory and therefore more processes can be maintained in memory. Furthermore, time is saved because unused pages are not swapped in and out of memory. However, the OS must be clever about how it manages this scheme. In the steady state practically, all of main memory will be occupied with process’s pages, so that the processor and OS has direct access to as many processes as possible. Thus when the OS brings one page in, it must throw another out. If it throws out a page just before it is used, then it will just have to get that page again almost immediately. Too much of this leads to a condition called Thrashing. The system spends most of its time swapping pages rather than executing instructions. So a good page replacement algorithm is required.  
  
In the given diagram, initial degree of multi programming upto some extent of point(lamda), the CPU utilization is very high and the system resources are utilized 100%. But if we further increase the degree of multi programming the CPU utilization will drastically fall down and the system will spend more time only in the page replacement and the time taken to complete the execution of the process will increase. This situation in the system is called as thrashing.  
  
**Causes of Thrashing :**

1. **High degree of multiprogramming:**

If the number of processes keeps on increasing in the memory than number of frames allocated to each process will be decreased. So, less number of frames will be available to each process. Due to this, page fault will occur more frequently and more CPU time will be wasted in just swapping in and out of pages and the utilization will keep on decreasing.

For example:  
Let free frames = 400  
**Case 1**: Number of process = 100  
Then, each process will get 4 frames.

**Case 2**: Number of process = 400  
Each process will get 1 frame.  
Case 2 is a condition of thrashing, as the number of processes are increased, frames per process are decreased. Hence CPU time will be consumed in just swapping pages.

1. **Lacks of Frames**: If a process has less number of frames then less pages of that process will be able to reside in memory and hence more frequent swapping in and out will be required. This may lead to thrashing. Hence sufficient amount of frames must be allocated to each process in order to prevent thrashing.

**Recovery of Thrashing:**

* Do not allow the system to go into thrashing by instructing the long term scheduler not to bring the processes into memory after the threshold.
* If the system is already in thrashing, then instruct the midterm scheduler to suspend some of the processes so that we can recover the system from thrashing.

# **Segmentation:**

A process is divided into Segments. The chunks that a program is divided into which are not necessarily all of the same sizes are called segments. Segmentation gives user’s view of the process which paging does not give. Here the user’s view is mapped to physical memory.

There are types of segmentation:

Virtual memory segmentation – Each process is divided into a number of segments, not all of which are resident at any one point in time.

Simple segmentation – Each process is divided into a number of segments, all of which are loaded into memory at run time, though not necessarily contiguously.

There is no simple relationship between logical addresses and physical addresses in segmentation. A table stores the information about all such segments and is called Segment Table.

Segment Table – It maps two-dimensional Logical address into one-dimensional Physical address. It’s each table entry has:

Base Address: It contains the starting physical address where the segments reside in memory.

Limit: It specifies the length of the segment.

Address generated by the CPU is divided into:

* **Segment number (s):** Number of bits required to represent the segment.
* **Segment offset (d):** Number of bits required to represent the size of the segment.

**Advantages of Segmentation –**

* No Internal fragmentation.
* Segment Table consumes less space in comparison to Page table in paging.

**Disadvantage of Segmentation –**

* As processes are loaded and removed from the memory, the free memory space is broken into little pieces, causing External fragmentation.

# **Working Set**

The working set of a process is the set of pages in the virtual address space of the process that are currently resident in physical memory. The working set contains only page-able memory allocations; non-page-able memory allocations such as Address Windowing Extensions (AWE) or large page allocations are not included in the working set.

When a process references page-able memory that is not currently in its working set, a page fault occurs. The system page fault handler attempts to resolve the page fault and, if it succeeds, the page is added to the working set. (Accessing AWE or large page allocations never causes a page fault, because these allocations are not page-able.)

A *hard page fault* must be resolved by reading page contents from the page's *backing store*, which is either the system paging file or a memory-mapped file created by the process. A *soft page fault* can be resolved without accessing the backing store. A soft page fault occurs when:

* The page is in the working set of some other process, so it is already resident in memory.
* The page is in transition, because it either has been removed from the working sets of all processes that were using the page and has not yet been repurposed, or it is already resident as a result of a memory manager pre-fetch operation.
* A process references an allocated virtual page for the first time (sometimes called a *demand-zero fault*).

Pages can be removed from a process working set as a result of the following actions:

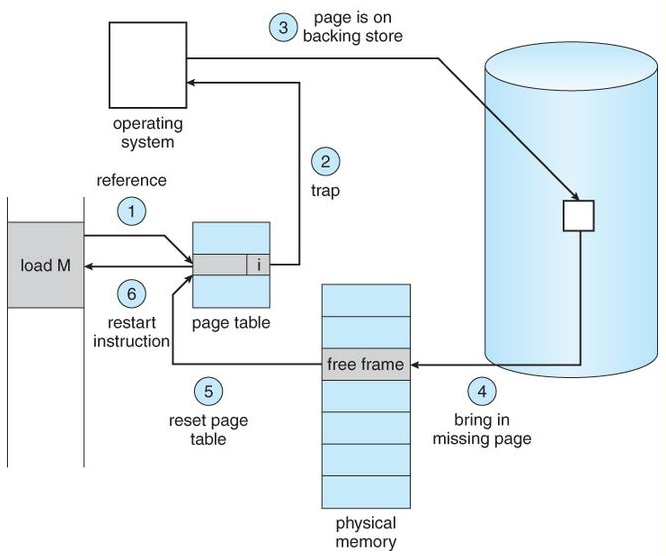
* The process reduces or empties the working set by calling the [**SetProcessWorkingSetSize**](https://msdn.microsoft.com/en-us/library/ms686234(v=VS.85).aspx), [**SetProcessWorkingSetSizeEx**](https://msdn.microsoft.com/en-us/library/ms686237(v=VS.85).aspx) or [**EmptyWorkingSet**](https://msdn.microsoft.com/library/ms682606(v=VS.85).aspx) function.
* The process calls the **[VirtualUnlock](https://msdn.microsoft.com/en-us/library/Aa366910(v=VS.85).aspx)** function on a memory range that is not locked.
* The process unmaps a mapped view of a file using the **[UnmapViewOfFile](https://msdn.microsoft.com/en-us/library/Aa366882(v=VS.85).aspx)** function.
* The memory manager trims pages from the working set to create more available memory.
* The memory manager must remove a page from the working set to make room for a new page (for example, because the working set is at its maximum size).

If several processes share a page, removing the page from the working set of one process does not affect other processes. After a page is removed from the working sets of all processes that were using it, the page becomes a *transition page*. Transition pages remain cached in RAM until the page is either referenced again by some process or repurposed (for example, filled with zeros and given to another process). If a transition page has been modified since it was last written to disk (that is, if the page is "dirty"), then the page must be written to its backing store before it can be repurposed. The system may start writing dirty transition pages to their backing store as soon as such pages become available.

Each process has a minimum and maximum working set size that affect the virtual memory paging behavior of the process. To obtain the current size of the working set of a specified process, use the GetProcessMemoryInfo function. To obtain or change the minimum and maximum working set sizes, use the GetProcessWorkingSetSizeEx and SetProcessWorkingSetSizeEx functions. The process status application programming interface (PSAPI) provides a number of functions that return detailed information about the working set of a process. For details, see Working Set Information.

# Page fault

A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM. So when page fault occurs then following sequence of events happens:



**Figure 10: Page fault**

* The computer hardware traps to the kernel and program counter (PC) is saved on the stack. Current instruction state information is saved in CPU registers.
* An assembly program is started to save the general registers and other volatile information to keep the OS from destroying it.
* Operating system finds that a page fault has occurred and tries to find out which virtual page is needed. Sometimes hardware register contains this required information. If not, the operating system must retrieve PC, fetch instruction and find out what it was doing when the fault occurred.
* Once virtual address caused page fault is known, system checks to see if address is valid and checks if there is no protection access problem.
* If the virtual address is valid, the system checks to see if a page frame is free. If no frames are free, the page replacement algorithm is run to remove a page.
* If frame selected is dirty, page is scheduled for transfer to disk, context switch takes place, fault process is suspended and another process is made to run until disk transfer is completed.
* As soon as page frame is clean, operating system looks up disk address where needed page is, schedules disk operation to bring it in.
* When disk interrupt indicates page has arrived, page tables are updated to reflect its position, and frame marked as being in normal state.
* Faulting instruction is backed up to state it had when it began and PC is reset. Faulting is scheduled, operating system returns to routine that called it.
* Assembly Routine reloads register and other state information, returns to user space to continue execution.

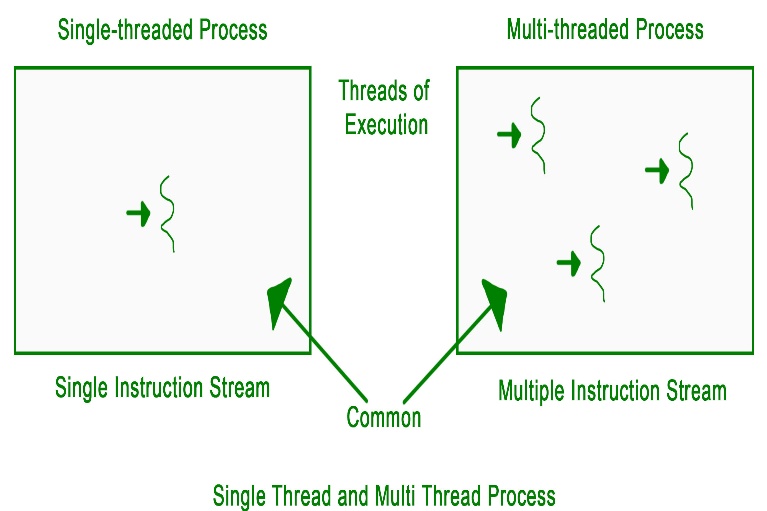
# **Multithreading**

A **thread** is a path which is followed during a program’s execution. Majority of programs written now a days run as a single thread. Let’s say, for example a program is not capable of reading keystrokes while making drawings. These tasks cannot be executed by the program at the same time. This problem can be solved through multitasking so that two or more tasks can be executed simultaneously.

Multitasking is of two types: Processor based and thread based. Processor based multitasking is totally managed by the OS, however multitasking through multithreading can be controlled by the programmer to some extent.

The concept of **multi-threading** needs proper understanding of these two terms – **a process and a thread**. A process is a program being executed. A process can be further divided into independent units known as threads.

A thread is like a small light-weight process within a process. Or we can say a collection of threads is what is known as a process.



**Figure 11: Single thread vs Multithread**

**Applications:**  
Threading is used widely in almost every field. Most widely it is seen over the internet now days where we are using transaction processing of every type like recharges, online transfer, banking etc. Threading is a segment which divide the code into small parts that are of very light weight and has less burden on CPU memory so that it can be easily worked out and can achieve goal in desired field. The concept of threading is designed due to the problem of fast and regular changes in technology and less the work in different areas due to less application. Then as says “need is the generation of creation or innovation” hence by following this approach human mind develop the concept of thread to enhance the capability of programming.

## **Benefits of Multithreading**

Prerequisite – Operating-System-Thread The benefits of multi-threaded programming can be broken down into four major categories:

Responsiveness:

Multithreading in an interactive application may allow a program to continue running even if a part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user. In a non-multi-threaded environment, a server listens to the port for some request and when the request comes, it processes the request and then resume listening to another request. The time taken while processing of request makes other users wait unnecessarily. Instead a better approach would be to pass the request to a worker thread and continue listening to port.

For example, a multi-threaded web browser allows user interaction in one thread while an video is being loaded in another thread. So instead of waiting for the whole web-page to load the user can continue viewing some portion of the web-page.

Resource Sharing:

Processes may share resources only through techniques such as-

* Message Passing
* Shared Memory

Such techniques must be explicitly organized by programmer. However, threads share the memory and the resources of the process to which they belong by default.

The benefit of sharing code and data is that it allows an application to have several threads of activity within same address space.

Economy:

Allocating memory and resources for process creation is a costly job in terms of time and space.

Since, threads share memory with the process it belongs, it is more economical to create and context switch threads. Generally, much more time is consumed in creating and managing processes than in threads.

In Solaris, for example, creating process is 30 times slower than creating threads and context switching is 5 times slower.

Scalability:

The benefits of multi-programming greatly increase in case of multiprocessor architecture, where threads may be running parallel on multiple processors. If there is only one thread, then it is not possible to divide the processes into smaller tasks that different processors can perform.

Single threaded process can run only on one processor regardless of how many processors are available.

Multi-threading on a multiple CPU machine increases parallelism.

## **Difference between multitasking, multithreading and multiprocessing**

1. **Multiprogramming –** A computer running more than one program at a time (like running Excel and Firefox simultaneously).
2. **Multiprocessing –** A computer using more than one CPU at a time.
3. **Multitasking –** Tasks sharing a common resource (like 1 CPU).
4. **Multithreading** is an extension of multitasking.

### **1. Multi programming:**

In a modern computing system, there are usually several concurrent application processes which want to execute. Now it is the responsibility of the Operating System to manage all the processes effectively and efficiently.

One of the most important aspects of an Operating System is to multi program.  
In a computer system, there are multiple processes waiting to be executed, i.e. they are waiting when the CPU will be allocated to them and they begin their execution. These processes are also known as jobs. Now the main memory is too small to accommodate all of these processes or jobs into it. Thus, these processes are initially kept in an area called job pool. This job pool consists of all those processes awaiting allocation of main memory and CPU.  
CPU selects one job out of all these waiting jobs, brings it from the job pool to main memory and starts executing it. The processor executes one job until it is interrupted by some external factor or it goes for an I/O task.

**Non-multi programmed system’s working:**

In a non multi programmed system, as soon as one job leaves the CPU and goes for some other task (say I/O), the CPU becomes idle. The CPU keeps waiting and waiting until this job (which was executing earlier) comes back and resumes its execution with the CPU. So CPU remains free for all this while.

Now it has a drawback that the CPU remains idle for a very long period of time. Also, other jobs which are waiting to be executed might not get a chance to execute because the CPU is still allocated to the earlier job.

This poses a very serious problem that even though other jobs are ready to execute, CPU is not allocated to them as the CPU is allocated to a job which is not even utilizing it (as it is busy in I/O tasks).

It cannot happen that one job is using the CPU for say 1 hour while the others have been waiting in the queue for 5 hours. To avoid situations like this and come up with efficient utilization of CPU, the concept of multi programming came up.

The main idea of multi programming is to maximize the CPU time.

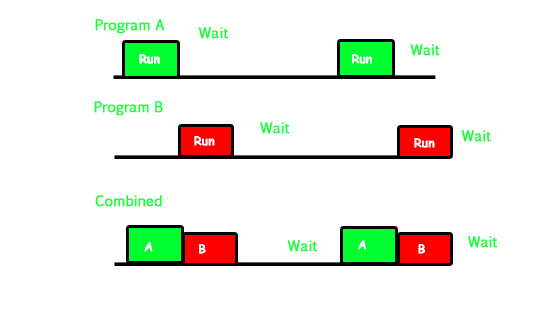
Multi programmed system’s working:

In a multi-programmed system, as soon as one job goes for an I/O task, the Operating System interrupts that job, chooses another job from the job pool (waiting queue), gives CPU to this new job and starts its execution. The previous job keeps doing its I/O operation while this new job does CPU bound tasks. Now say the second job also goes for an I/O task, the CPU chooses a third job and starts executing it. As soon as a job completes its I/O operation and comes back for CPU tasks, the CPU is allocated to it.

In this way, no CPU time is wasted by the system waiting for the I/O task to be completed.

Therefore, the ultimate goal of multi programming is to keep the CPU busy as long as there are processes ready to execute. This way, multiple programs can be executed on a single processor by executing a part of a program at one time, a part of another program after this, then a part of another program and so on, hence executing multiple programs. Hence, the CPU never remains idle.

In the image below, program A runs for some time and then goes to waiting state. In the meantime, program B begins its execution. So the CPU does not waste its resources and gives program B an opportunity to run.



**Figure 12: Multi programmed system’s working**

### 2. Multiprocessing:

In a uni-processor system, only one process executes at a time. Multiprocessing is the use of two or more CPUs (processors) within a single Computer system. The term also refers to the ability of a system to support more than one processor within a single computer system. Now since there are multiple processors available, multiple processes can be executed at a time. These multi processors share the computer bus, sometimes the clock, memory and peripheral devices also.

Multi-processing system’s working:

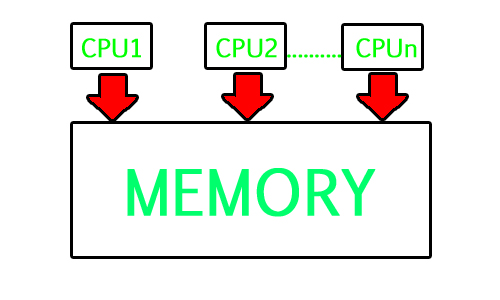
With the help of multiprocessing, many processes can be executed simultaneously. Say processes P1, P2, P3 and P4 are waiting for execution. Now in a single processor system, firstly one process will execute, then the other, then the other and so on.

But with multiprocessing, each process can be assigned to a different processor for its execution. If it’s a dual-core processor (2 processors), two processes can be executed simultaneously and thus will be two times faster, similarly a quad core processor will be four times as fast as a single processor.

Why use multi-processing:

The main advantage of multiprocessor system is to get more work done in a shorter period of time. These types of systems are used when very high speed is required to process a large volume of data. Multi-processing systems can save money in comparison to single processor systems because the processors can share peripherals and power supplies.

It also provides increased reliability in the sense that if one processor fails, the work does not halt, it only slows down. e.g. if we have 10 processors and 1 fails, then the work does not halt, rather the remaining 9 processors can share the work of the 10th processor. Thus the whole system runs only 10 percent slower, rather than failing altogether.



**Figure 13: Multi-processing system**

Multiprocessing refers to the hardware (i.e., the CPU units) rather than the software (i.e., running processes). If the underlying hardware provides more than one processor, then that is multiprocessing. It is the ability of the system to leverage multiple processors’ computing power.

Difference between Multi programming and Multi processing:

* A System can be both multi programmed by having multiple programs running at the same time and multiprocessing by having more than one physical processor. The difference between multiprocessing and multi programming is that Multiprocessing is basically executing multiple processes at the same time on multiple processors, whereas multi programming is keeping several programs in main memory and executing them concurrently using a single CPU only.
* Multiprocessing occurs by means of parallel processing whereas Multi programming occurs by switching from one process to other (phenomenon called as context switching).

### 3. Multitasking:

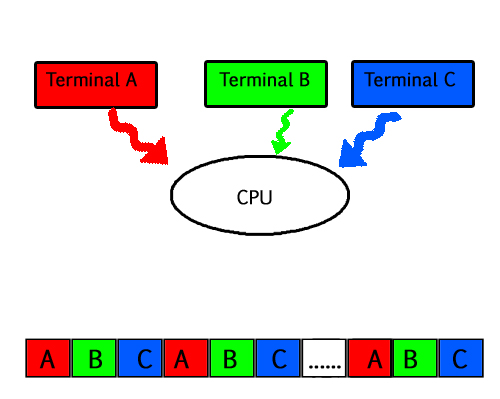
As the name itself suggests, multi-tasking refers to execution of multiple tasks (say processes, programs, threads etc.) at a time. In the modern operating systems, we are able to play MP3 music, edit documents in Microsoft Word, surf the Google Chrome all simultaneously, this is accomplished by means of multi-tasking.

Multitasking is a logical extension of multi programming. The major way in which multitasking differs from multi programming is that multi programming works solely on the concept of context switching whereas multitasking is based on time sharing alongside the concept of context switching.

Multi-tasking system’s working:

* In a time sharing system, each process is assigned some specific quantum of time for which a process is meant to execute. Say there are 4 processes P1, P2, P3, P4 ready to execute. So each of them are assigned some time quantum for which they will execute e.g time quantum of 5 nanoseconds (5 ns). As one process begins execution (say P2), it executes for that quantum of time (5 ns). After 5 ns the CPU starts the execution of the other process (say P3) for the specified quantum of time.
* Thus the CPU makes the processes to share time slices between them and execute accordingly. As soon as time quantum of one process expires, another process begins its execution.
* Here also basically a context switch is occurring but it is occurring so fast that the user is able to interact with each program separately while it is running. This way, the user is given the illusion that multiple processes/ tasks are executing simultaneously. But actually only one process/ task is executing at a particular instant of time. In multitasking, time sharing is best manifested because each running process takes only a fair quantum of the CPU time.

In a more general sense, multitasking refers to having multiple programs, processes, tasks, threads running at the same time. This term is used in modern operating systems when multiple tasks share a common processing resource (e.g., CPU and Memory).



**Figure 14:** **Multi-tasking system**

* As depicted in the above image, at any time the CPU is executing only one task while other tasks are waiting for their turn. The illusion of parallelism is achieved when the CPU is reassigned to another task. i.e. all the three tasks A, B and C are appearing to occur simultaneously because of time sharing.
* So for multitasking to take place, firstly there should be multiprogramming i.e. presence of multiple programs ready for execution. And secondly the concept of time sharing.

### 4. Multi-threading:

A thread is a basic unit of CPU utilization. Multi-threading is an execution model that allows a single process to have multiple code segments (i.e., threads) running concurrently within the “context” of that process.

e.g. VLC media player, where one thread is used for opening the VLC media player, one thread for playing a particular song and another thread for adding new songs to the playlist.

Multi-threading is the ability of a process to manage its use by more than one user at a time and to manage multiple requests by the same user without having to have multiple copies of the program.

Multi-threading system’s working –

Example 1 –

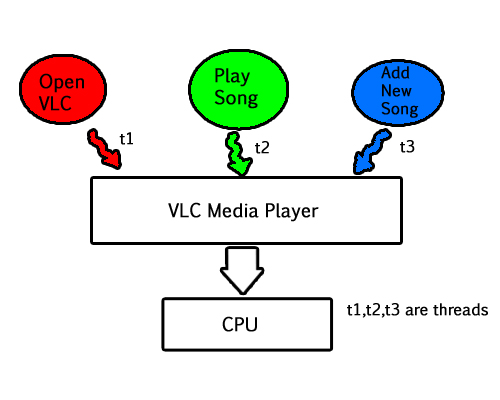
Say there is a web server which processes client requests. Now if it executes as a single threaded process, then it will not be able to process multiple requests at a time. Firstly one client will make its request and finish its execution and only then, the server will be able to process another client request. This is really costly, time consuming and tiring task. To avoid this, multi-threading can be made use of.

Now, whenever a new client request comes in, the web server simply creates a new thread for processing this request and resumes its execution to hear more client requests. So the web server has the task of listening to new client requests and creating threads for each individual request. Each newly created thread processes one client request, thus reducing the burden on web server.

Example 2 –

We can think of threads as child processes that share the parent process resources but execute independently. Now take the case of a GUI. Say we are performing a calculation on the GUI (which is taking very long time to finish). Now we cannot interact with the rest of the GUI until this command finishes its execution. To be able to interact with the rest of the GUI, this command of calculation should be assigned to a separate thread. So at this point of time, 2 threads will be executing i.e. one for calculation, and one for the rest of the GUI. Hence here in a single process, we used multiple threads for multiple functionality.

The image below completely describes the VLC player example:



**Figure 15: Multi-threading system**

Advantages of Multi-threading:

Benefits of Multi-threading include increased responsiveness. Since there are multiple threads in a program, so if one thread is taking too long to execute or if it gets blocked, the rest of the threads keep executing without any problem. Thus the whole program remains responsive to the user by means of remaining threads.

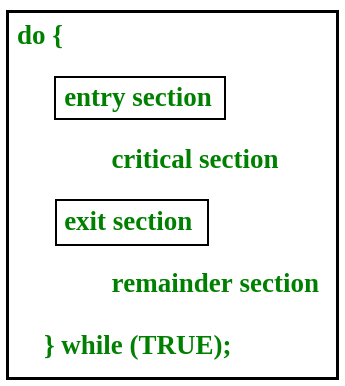
Another advantage of multi-threading is that it is less costly. Creating brand new processes and allocating resources is a time consuming task, but since threads share resources of the parent process, creating threads and switching between them is comparatively easy. Hence multi-threading is the need of modern Operating Systems.

## **Process Synchronization**

On the basis of synchronization, processes are categorized as one of the following two types:

* **Independent Process**: Execution of one process does not affects the execution of other processes.
* **Cooperative Process**: Execution of one process affects the execution of other processes.

Process synchronization problem arises in the case of Cooperative process also because resources are shared in Cooperative processes.  
   
**Race Condition**  
Several processes access and process the manipulations over the same data concurrently, then the outcome depends on the particular order in which the access takes place.  
   
**Critical Section Problem**

Critical section is a code segment that can be accessed by only one process at a time. Critical section contains shared variables which need to be synchronized to maintain consistency of data variables.  
[](https://www.geeksforgeeks.org/wp-content/uploads/gq/2015/06/critical-section-problem.png)

In the entry section, the process requests for entry in the **Critical Section.**

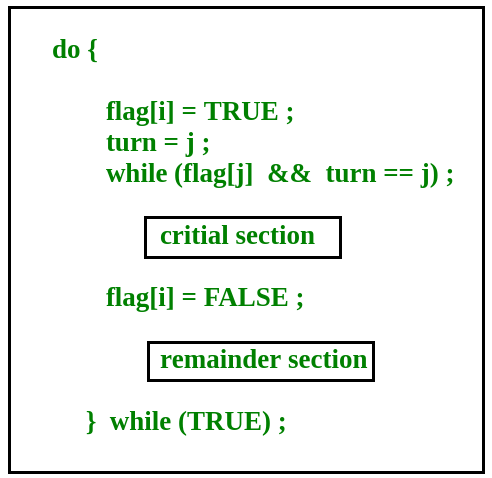
Any solution to the critical section problem must satisfy three requirements:

* **Mutual Exclusion**: If a process is executing in its critical section, then no other process is allowed to execute in the critical section.
* **Progress**: If no process is executing in the critical section and other processes are waiting outside the critical sectios, then only those processes that are not executing in their remainder section can participate in deciding which will enter in the critical section next, and the selection cannot be postponed indefinitely.
* **Bounded Waiting**: A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

**Peterson’s Solution**  
Peterson’s Solution is a classical software based solution to the critical section problem.

In Peterson’s solution, we have two shared variables:

* Boolean flag[i]: Initialized to FALSE, initially no one is interested in entering the critical section
* int turn: The process whose turn is to enter the critical section.

[](https://www.geeksforgeeks.org/wp-content/uploads/gq/2015/06/peterson.png)  
   
Peterson’s Solution preserves all three conditions:

* Mutual Exclusion is assured as only one process can access the critical section at any time.
* Progress is also assured, as a process outside the critical section does not blocks other processes from entering the critical section.
* Bounded Waiting is preserved as every process gets a fair chance.

Disadvantages of Peterson’s Solution

* + It involves Busy waiting
  + It is limited to 2 processes.

## **Deadlock Introduction**

A process in operating systems uses different resources and uses resources in following way.  
1) Requests a resource  
2) Use the resource  
2) Releases the resource

**Deadlock**is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.  
Consider an example when two trains are coming toward each other on same track and there is only one track, none of the trains can move once they are in front of each other. Similar situation occurs in operating systems when there are two or more processes hold some resources and wait for resources held by other(s). For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.

[[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/gq/2015/06/deadlock.png)](https://media.geeksforgeeks.org/wp-content/cdn-uploads/gq/2015/06/deadlock.png)

**Figure 16: Deadlock**

Deadlock can arise if following four conditions hold simultaneously (Necessary Conditions)   
Mutual Exclusion: One or more than one resource are non-sharable (Only one process can use at a time)  
Hold and Wait: A process is holding at least one resource and waiting for resources.  
No Preemption: A resource cannot be taken from a process unless the process releases the resource.  
Circular Wait: A set of processes are waiting for each other in circular form.

Methods for handling deadlock  
There are three ways to handle deadlock  
1) Deadlock prevention or avoidance: The idea is to not let the system into deadlock state.  
One can zoom into each category individually, Prevention is done by negating one of above mentioned necessary conditions for deadlock.  
Avoidance is kind of futuristic in nature. By using strategy of “Avoidance”, we have to make an assumption. We need to ensure that all information about resources which process WILL need are known to us prior to execution of the process. We use Banker’s algorithm (Which is in-turn a gift from Dijkstra) in order to avoid deadlock.

2) Deadlock detection and recovery: Let deadlock occur, then do preemption to handle it once occurred.

3) Ignore the problem all together: If deadlock is very rare, then let it happen and reboot the system. This is the approach that both Windows and UNIX take.

# **Process Scheduler**

There are three types of process scheduler.

1.Long Term or job scheduler It brings the new process to the ‘Ready State’. It controls Degree of Multi-programming, i.e., number of process present in ready state at any point of time. It is important that the long-term scheduler make a careful selection of both IO and CPU bound process.

2.Short term or CPU scheduler: It is responsible for selecting one process from ready state for scheduling it on the running state. Note: Short-term scheduler only selects the process to schedule it doesn’t load the process on running. Dispatcher is responsible for loading the process selected by Short-term scheduler on the CPU (Ready to Running State) Context switching is done by dispatcher only. A dispatcher does the following:

* Switching context.
* Switching to user mode.
* Jumping to the proper location in the newly loaded program.

3. Medium-term scheduler It is responsible for suspending and resuming the process. It mainly does swapping (moving processes from main memory to disk and vice versa). Swapping may be necessary to improve the process mix or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up.