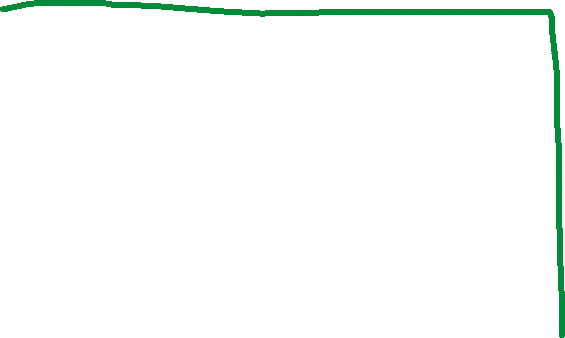
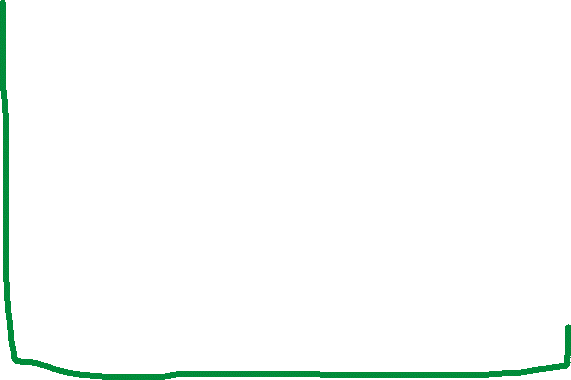
Operating System is a System software which controls hardware as well as System software and application software. Banking System Airline Reservation Web Browser Compiler Editor Shell(Command Interpreter) Operating System Machine Language(ISA) Microarchitecture(Functional Units) Physical Devices(IC,wires,CRT etc)



(A Computer System Consists of Hardware, System Programs, and application programs)

1,2,3 refers to Hardware

4,5 refers System Programs

6 refers to application software

Goals of an OS

1.Convenient for user-Provides an easy-to-use interface along with

2.Efficient use of Resources

The main goal of the Operating System is to thus make the computer environment more convenient to use and the secondary goal is to use the resources in the most efficient manner.

**What is an** **Operating System?**   
A system Software that manages hardware and other software of a computer. This software includes both System and application software.

System software can be thought of as software that is closer to hardware where as application software is built on top of system software. Example of system software

Language Converter: Operating System, Device Drivers, Assembler, Compiler, Interpreter

Application Software: Bowser, Office Package, Banking Application

**Important functions of an operating System:**

1. **Security –**   
   The operating system uses password protection to protect user data and similar other techniques. it also prevents unauthorized access to programs and user data.
2. **Control over system performance –**   
   Monitors overall system health to help improve performance. records the response time between service requests and system response to having a complete view of the system health. This can help improve performance by providing important information needed to troubleshoot problems.
3. **Job accounting –**   
   Operating system Keeps track of time and resources used by various tasks and users, this information can be used to track resource usage for a particular user or group of users.
4. **Error detecting aids –**   
   The operating system constantly monitors the system to detect errors and avoid the malfunctioning of a computer system.
5. **Coordination between other software and users –**   
   Operating systems also coordinate and assign interpreters, compilers, assemblers, and other software to the various users of the computer systems.
6. **Memory Management –**   
   The operating system manages the Primary Memory or Main Memory. Main memory is made up of a large array of bytes or words where each byte or word is assigned a certain address. Main memory is fast storage and it can be accessed directly by the CPU. For a program to be executed, it should be first loaded in the main memory. An Operating System performs the following activities for memory management:

It keeps track of primary memory, i.e., which bytes of memory are used by which user program. The memory addresses that have already been allocated and the memory addresses of the memory that has not yet been used. In multiprogramming, the OS decides the order in which processes are granted access to memory, and for how long. It Allocates the memory to a process when the process requests it and deallocates the memory when the process has terminated or is performing an I/O operation. 

1. **Processor Management –**   
   In a multi-programming environment, the OS decides the order in which processes have access to the processor, and how much processing time each process has. This function of OS is called process scheduling. An Operating System performs the following activities for processor management.

Keeps track of the status of processes. The program which performs this task is known as a traffic controller. Allocates the CPU that is a processor to a process. De-allocates processor when a process is no more required. 

1. **Device Management –**   
   An OS manages device communication via their respective drivers. It performs the following activities for device management. Keeps track of all devices connected to the system. designates a program responsible for every device known as the Input/Output controller. Decides which process gets access to a certain device and for how long. Allocates devices in an effective and efficient way. Deallocates devices when they are no longer required.
2. **File Management –**   
   A file system is organized into directories for efficient or easy navigation and usage. These directories may contain other directories and other files. An Operating System carries out the following file management activities. It keeps track of where information is stored, user access settings and status of every file, and more… These facilities are collectively known as the file system.

Moreover, Operating System also provides certain services to the computer system in one form or the other.   
The Operating System provides certain services to the users which can be listed in the following manner:

1. **Program Execution**: The Operating System is responsible for the execution of all types of programs whether it be user programs or system programs. The Operating System utilizes various resources available for the efficient running of all types of functionalities.
2. **Handling Input/Output Operations**: The Operating System is responsible for handling all sorts of inputs, i.e, from the keyboard, mouse, desktop, etc. The Operating System does all interfacing in the most appropriate manner regarding all kinds of Inputs and Outputs.   
   For example, there is a difference in the nature of all types of peripheral devices such as mice or keyboards, the Operating System is responsible for handling data between them.
3. **Manipulation of File System**: The Operating System is responsible for making decisions regarding the storage of all types of data or files, i.e, floppy disk/hard disk/pen drive, etc. The Operating System decides how the data should be manipulated and stored.
4. **Error Detection and Handling**: The Operating System is responsible for the detection of any type of error or bugs that can occur while any task. The well-secured OS sometimes also acts as a countermeasure for preventing any sort of breach to the Computer System from any external source and probably handling them.
5. **Resource Allocation:** The Operating System ensures the proper use of all the resources available by deciding which resource to be used by whom for how much time. All the decisions are taken by the Operating System.
6. **Accounting:** The Operating System tracks an account of all the functionalities taking place in the computer system at a time. All the details such as the types of errors that occurred are recorded by the Operating System.
7. **Information and Resource Protection:** The Operating System is responsible for using all the information and resources available on the machine in the most protected way. The Operating System must foil an attempt from any external resource to hamper any sort of data or information.

All these services are ensured by the Operating System for the convenience of the users to make the programming task easier. All different kinds of Operating systems more or less provide the same services.

**Operating System Evolution**

Operating system is divided into four generations, which are explained as follows −

**First Generation (1940s)**

Programmer interacted directly with the computer hardware

No operating System

Machines were run from a console display lights,toggle Switches

Program in machine code were loaded via input device card reader

Error condition was refelected by the lights,else successful completion can give output which can be printed on a printer

Two main problems

Fixed Alloted Time per programmer:A programmer may finish before time or may end up requiring more time than the allotted time for usage of computer

SetUp Time:There are many tasks like compiling,linking,loading involved in execution of a jon.If an error occur,the user had to go to beginning of setup.

**Second Generation (1950s)** The users of a batch operating system do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together and run as a group. The programmers leave their programs with the operator and the operator then sorts the programs with similar requirements into batches.

The problems with Batch Systems are as follows −

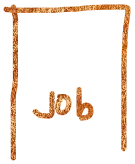
* Lack of interaction between the user and the job.
* CPU is often idle, because the speed of the mechanical I/O devices is slower than the CPU.

Difficult to provide the desired priority

The central idea behind a batch OS is monitor which performed a scheduling function,for which a program was containing special instructions from JCL(Job Control Language).The cPU time alternates between user job and monitor.

The processing of job was serial,one job at a time.

****

****

**(1960s)**

**The systems of 60s were also batch processing ,but they were multiprogramed batch OS. It means more than one job could be loaded into RAM.**

The development of the operating system was developed to serve multiple users at once in the third generation. Here the interactive users can communicate through an online terminal to a computer, so the operating system becomes multi-user and multiprogramming.

**TimeSharing -Supports more than one interactive users who operate from a terminal connected to a central computer.Users can communicate with the program while it is running.**.

1970s

Multimode programming systems,that supports batch,timesharing,realtime applications

TCP/IP and LAN enabled communication among computers.

Security become a challenge

**1980s**

**Decade of Personal Computers and workstations**

**User interaction becomes easy for GUI**

**Distributed Computing become widespread under Client Server model.**

**1990s**

**Microsoft Corporation becomes dominant**

**Windows 3.1 becomes popular as users navigated among multiple concurrent applications for its user friendly interface and rich functionalities**

**Beginning of OPEN-SOURCE movement**

**2000s and beyond Use of Middleware**

**Massive parallelism**

**Standardization of user and application of interfaces, for example POSIX**

**Popularity of distributed OS**

Types of OS

Batch operating system

The users of a batch operating system do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together and run as a group. The programmers leave their programs with the operator and the operator then sorts the programs with similar requirements into batches.

The problems with Batch Systems are as follows −

* Lack of interaction between the user and the job.
* CPU is often idle, because the speed of the mechanical I/O devices is slower than the CPU.
* Difficult to provide the desired priority.

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 main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is that in case of Multiprogrammed batch systems, the objective is to maximize processor use, whereas in Time-Sharing Systems, the objective is to minimize response time.

Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receive an immediate response. For example, in a transaction processing, the processor executes each user program in a short burst or quantum of computation. That is, if **n** users are present, then each user can get a time quantum. When the user submits the command, the response time is in few seconds at most.

The 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.

Advantages of Timesharing operating systems are as follows −

* Provides the advantage of quick response.
* Avoids duplication of software.
* Reduces CPU idle time.

Disadvantages of Time-sharing operating systems are as follows −

* Problem of reliability.
* Question of security and integrity of user programs and data.
* Problem of data communication.

Distributed operating System

Distributed systems use multiple central processors to serve multiple real-time applications and multiple users. Data processing jobs are distributed among the processors accordingly.

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 −

* With resource sharing facility, a user at one site may be able to use the resources available at another.
* Speedup the exchange of data with one another via electronic mail.
* If one site fails in a distributed system, the remaining sites can potentially continue operating.
* Better service to the customers.
* Reduction of the load on the host computer.
* Reduction of delays in data processing.

Network operating System

A Network Operating System runs on a server and provides the server the capability to manage data, users, groups, security, applications, and other networking functions. The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks.

Examples of network operating systems include Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.

The advantages of network operating systems are as follows −

* Centralized servers are highly stable.
* Security is server managed.
* Upgrades to new technologies and hardware can be easily integrated into the system.
* Remote access to servers is possible from different locations and types of systems.

The disadvantages of network operating systems are as follows −

* High cost of buying and running a server.
* Dependency on a central location for most operations.
* Regular maintenance and updates are required.

Real Time operating System

A real-time system is defined as a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment. The time taken by the system to respond to an input and display of required updated information is termed as the **response time**. So in this method, the response time is very less as compared to online processing.

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. A real-time operating system must have well-defined, fixed time constraints, otherwise the system will fail. For example, Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

There are two types of real-time operating systems.

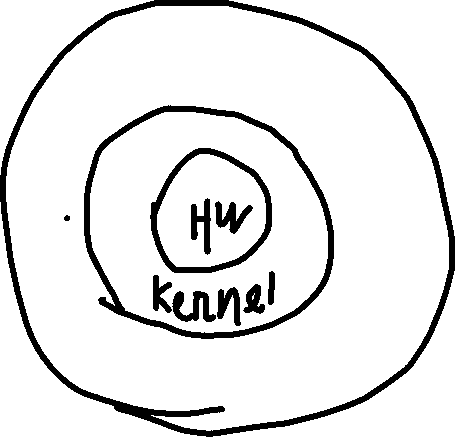
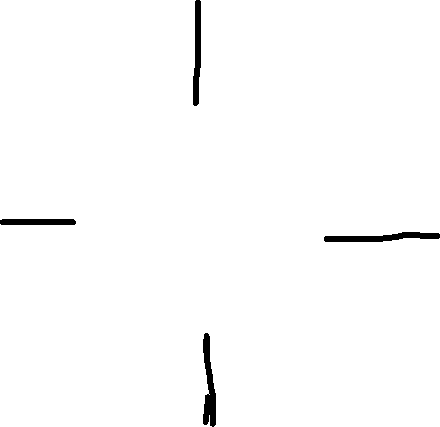
Hard real-time systems

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

Soft real-time systems

Soft real-time systems are less restrictive. A 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. For example, multimedia, virtual reality, Advanced Scientific Projects like undersea exploration and planetary rovers, etc.

Shell Vs Kernel



Observe shell is not a part of OS, rather it is a special software which is implemented as text-based interfaces that enable the user to issue commands from a keyboard or as a GUI to point and click, drag and drop icons to request services from the the OS

Ex- Windows XP provides a GUI as well as a command prompt window that accepts typed command.

The software that contains the core components of the OS is referred to as the kernel. Kernel tends to fall under 3 categories:

Micro Kernel :A micro kernel groups most essential features to reduce the size of kernel.For example,CPU management,memory management and IPC may constitute micro kernel and rest of the tasks(Device Management,file system management and system server calls can be handled at user level.

Monolithic: These kernels group all activities in a kernel.Ex Linux

Hybrid: Windows Kernel belongs to this category ,it has the flexibility to choose what to run in both user and kernel mode.

Kernel in general can have following components.

The process Manager

Process creation, termination

Process Scheduling and dispatching

Process Switching

Process Synchronization

Memory manager

Allocation of address space to to processes

Swapping

Page and Segment management

I/O Manager

Buffer Management

Allocation of IO channels and devices to process

IPC (Interprocess Communication) Manager

Communication

File system Manager

All the above code runs in kernel mode/Supervisor mode, which is protected from user tampering, we cannot modify clock interrupt handler.

Compilers and editors run in user mode, if we do not like we can change it. At a given time CPU may be executing in User mode or Kernel mode

Why two modes of execution????

Kernel executes in Which mode?



Your C prog runs in \_\_\_\_ mode

Program Status word (PSW): There is a bit in PSW which is known as mode bit .PSW is a register in CPU.

Mode bit=0 kernel Mode

=1,User mode

Normally the user program runs in user mode, but whenever wants service from OS, CPU has to switch to kernel mode to run a OS routine. For this user can take help of System Call (A call to OS System to avail service from kernel)

The interface between a process and an operating system is provided by system calls. In general, system calls are available as assembly language instructions. They are also included in the manuals used by the assembly level programmers.

System calls are usually made when a process in user mode requires access to a resource. Then it requests the kernel to provide the resource via a system call.

**Types of System Calls**

There are mainly five types of system calls. These are explained in detail as follows −

**Process Control**

These system calls deal with processes such as process creation, process termination etc.

**File Management**

These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.

**Device Management**

These system calls are responsible for device manipulation such as reading from device buffers, writing into device buffers etc.

**Information Maintenance**

These system calls handle information and its transfer between the operating system and the user program.

**Communication**

These system calls are useful for interprocess communication. They also deal with creating and deleting a communication connection.

Some of the examples of all the above types of system calls in Windows and Unix are given as follows −

| **Types of System Calls** | **Windows** | **Linux** |
| --- | --- | --- |
| Process Control | CreateProcess() ExitProcess() WaitForSingleObject() | fork() exit() wait() |
| File Management | CreateFile() ReadFile() WriteFile() CloseHandle() | open() read() write() close() |
| Device Management | SetConsoleMode() ReadConsole() WriteConsole() | ioctl() read() write() |
| Information Maintenance | GetCurrentProcessID() SetTimer() Sleep() | getpid() alarm() sleep() |
| Communication | CreatePipe() CreateFileMapping() MapViewOfFile() | pipe() shmget() mmap() |

There are many different system calls as shown above. Details of some of those system calls are as follows −

**wait()**

In some systems, a process may wait for another process to complete its execution. This happens when a parent process creates a child process and the execution of the parent process is suspended until the child process executes. The suspending of the parent process occurs with a wait() system call. When the child process completes execution, the control is returned back to the parent process.

**exec()**

This system call runs an executable file in the context of an already running process. It replaces the previous executable file. This is known as an overlay. The original process identifier remains since a new process is not created but data, heap, stack etc. of the process are replaced by the new process.

**fork()**

Processes use the fork() system call to create processes that are a copy of themselves. This is one of the major methods of process creation in operating systems. When a parent process creates a child process and the execution of the parent process is suspended until the child process executes. When the child process completes execution, the control is returned back to the parent process.

**exit()**

The exit() system call is used by a program to terminate its execution. In a multithreaded environment, this means that the thread execution is complete. The operating system reclaims resources that were used by the process after the exit() system call.

**kill()**

The kill() system call is used by the operating system to send a termination signal to a process that urges the process to exit.However, kill system call does not necessary mean killing the process and can have various meanings.

Program

A program is a piece of code which may be a single line or millions of lines. A computer program is usually written by a computer programmer in a programming language. For example, here is a simple program written in C programming language −

#include <stdio.h>

int main() {

printf("Hello, World! \n");

return 0;

}

A computer program is a collection of instructions that performs a specific task when executed by a computer. When we compare a program with a process, we can conclude that a process is a dynamic instance of a computer program.

A part of a computer program that performs a well-defined task is known as an **algorithm**. A collection of computer programs, libraries and related data are referred to as a **software**.

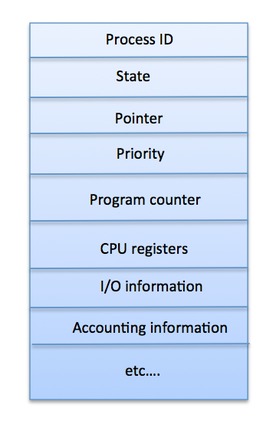
## **Process**

A process is basically a program in execution. The execution of a process must progress in a sequential fashion.

A process is defined as an entity which represents the basic unit of work to be implemented in the system.

To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data. The following image shows a simplified layout of a process inside main memory −

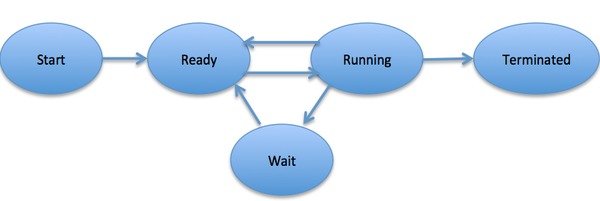


Process Life Cycle

When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized.

In general, a process can have one of the following five states at a time.

|  |  |
| --- | --- |
| **S.N.** | **State & Description** |
| 1 | **Start**  This is the initial state when a process is first started/created. |
| 2 | **Ready**  The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to some other process. |
| 3 | **Running**  Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions. |
| 4 | **Waiting**  Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available. |
| 5 | **Terminated or Exit**  Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory. |

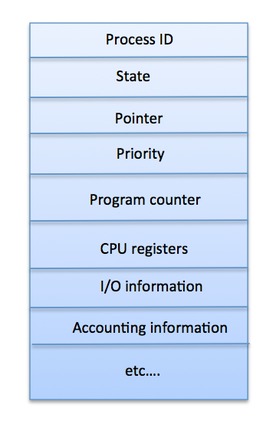


## Process Control Block (PCB)

A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process as listed below in the table −

|  |  |
| --- | --- |
| **S.N.** | **Information & Description** |
| 1 | **Process State**  The current state of the process i.e., whether it is ready, running, waiting, or whatever. |
| 2 | **Process privileges**  This is required to allow/disallow access to system resources. |
| 3 | **Process ID**  Unique identification for each of the process in the operating system. |
| 4 | **Pointer**  A pointer to parent process. |
| 5 | **Program Counter**  Program Counter is a pointer to the address of the next instruction to be executed for this process. |
| 6 | **CPU registers**  Various CPU registers where process need to be stored for execution for running state. |
| 7 | **CPU Scheduling Information**  Process priority and other scheduling information which is required to schedule the process. |
| 8 | **Memory management information**  This includes the information of page table, memory limits, Segment table depending on memory used by the operating system. |
| 9 | **Accounting information**  This includes the amount of CPU used for process execution, time limits, execution ID etc. |
| 10 | **IO status information**  This includes a list of I/O devices allocated to the process. |

The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified diagram of a PCB −



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 its execution is over, 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.

Operations on Process

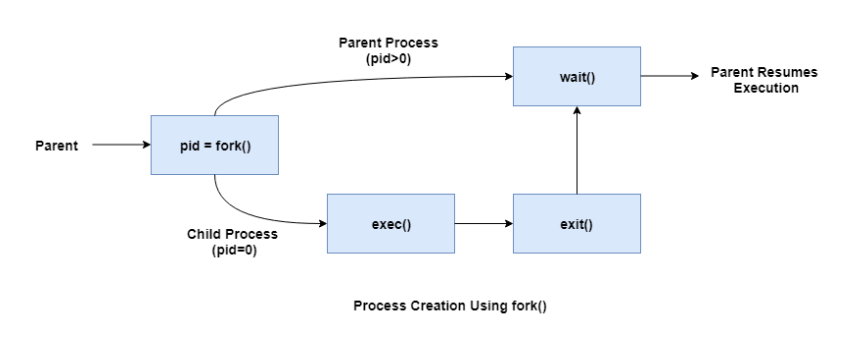
**Process Creation**

Processes need to be created in the system for different operations. This can be done by the following events −

* User request for process creation
* System initialization
* Execution of a process creation system call by a running process
* Batch job initialization

A process may be created by another process using fork(). The creating process is called the parent process and the created process is the child process. A child process can have only one parent but a parent process may have many children. Both the parent and child processes have the same memory image, open files, and environment strings. However, they have distinct address spaces.

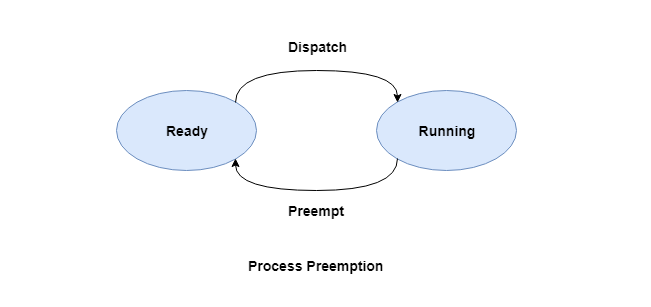
A diagram that demonstrates process creation using fork() is as follows −



**Process Preemption**

An interrupt mechanism is used in preemption that suspends the process executing currently and the next process to execute is determined by the short-term scheduler. Preemption makes sure that all processes get some CPU time for execution.

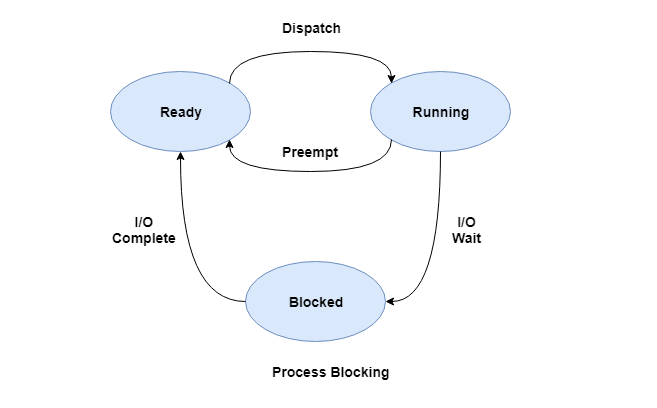
A diagram that demonstrates process preemption is as follows −



**Process Blocking**

The process is blocked if it is waiting for some event to occur. This event may be I/O as the I/O events are executed in the main memory and don't require the processor. After the event is complete, the process again goes to the ready state.

A diagram that demonstrates process blocking is as follows −



**Process Termination**

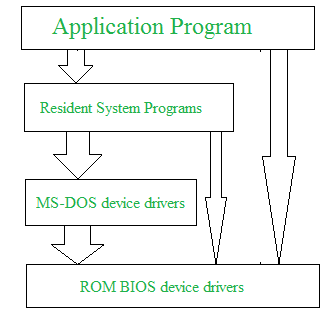
After the process has completed the execution of its last instruction, it is terminated. The resources held by a process are released after it is terminated.

A child process can be terminated by its parent process if its task is no longer relevant. The child process sends its status information to the parent process before it terminates. Also, when a parent process is terminated, its child processes are terminated as well as the child processes cannot run if the parent processes are terminated.

Operating System Structure

Operating system can be implemented with the help of various structures. The structure of the OS depends mainly on how the various common components of the operating system are interconnected and melded into the kernel. Depending on this we have following structures of the operating system:

**Simple structure:**   
Such operating systems do not have well defined structure and are small, simple and limited systems. The interfaces and levels of functionality are not well separated. MS-DOS is an example of such operating system. In MS-DOS application programs are able to access the basic I/O routines. These types of operating system cause the entire system to crash if one of the user programs fails.   
Diagram of the structure of MS-DOS is shown below.



**Advantages of Simple structure:**

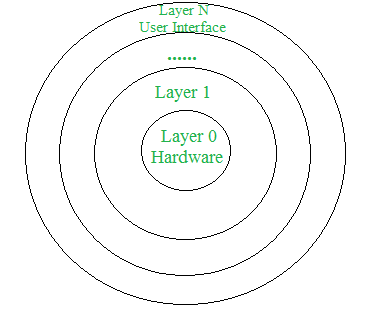
* It delivers better application performance because of the few interfaces between the application program and the hardware.
* Easy for kernel developers to develop such an operating system.

**Disadvantages of Simple structure:**

* The structure is very complicated as no clear boundaries exists between modules.
* It does not enforce data hiding in the operating system.

**Layered structure:**   
An OS can be broken into pieces and retain much more control on system. In this structure the OS is broken into number of layers (levels). The bottom layer (layer 0) is the hardware and the topmost layer (layer N) is the user interface. These layers are so designed that each layer uses the functions of the lower-level layers only. This simplifies the debugging process as if lower-level layers are debugged and an error occurs during debugging then the error must be on that layer only as the lower-level layers have already been debugged.

The main disadvantage of this structure is that at each layer, the data needs to be modified and passed on which adds overhead to the system. Moreover, careful planning of the layers is necessary as a layer can use only lower-level layers. UNIX is an example of this structure.



**Advantages of Layered structure:**

* Layering makes it easier to enhance the operating system as implementation of a layer can be changed easily without affecting the other layers.
* It is very easy to perform debugging and system verification.

**Disadvantages of Layered structure:**

* In this structure the application performance is degraded as compared to simple structure.
* It requires careful planning for designing the layers as higher layers use the functionalities of only the lower layers.

**Micro-kernel:**   
This structure designs the operating system by removing all non-essential components from the kernel and implementing them as system and user programs. This result in a smaller kernel called the micro-kernel.   
Advantages of this structure are that all new services need to be added to user space and does not require the kernel to be modified. Thus, it is more secure and reliable as if a service fails then rest of the operating system remains untouched. Mac OS is an example of this type of OS.

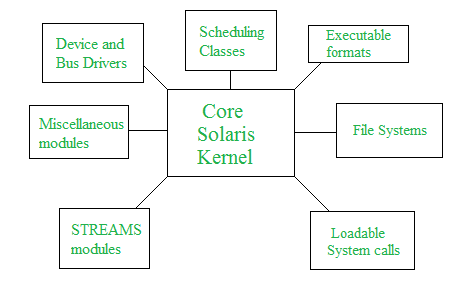
**Advantages of Micro-kernel structure:**

* It makes the operating system portable to various platforms.
* As microkernels are small so these can be tested effectively.

**Disadvantages of Micro-kernel structure:**

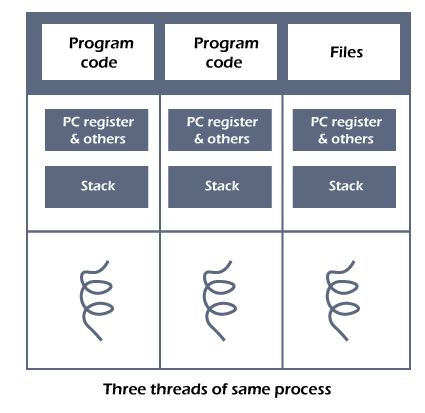
* Increased level of inter module communication degrades system performance.

**Modular structure or approach:**   
It is considered as the best approach for an OS. It involves designing of a modular kernel. The kernel has only set of core components and other services are added as dynamically loadable modules to the kernel either during run time or boot time. It resembles layered structure due to the fact that each kernel has defined and protected interfaces but it is more flexible than the layered structure as a module can call any other module.   
For example, Solaris OS is organized as shown in the figure.



Threads in Operating System

A thread is a single sequential flow of execution of tasks of a process so it is also known as thread of execution or thread of control. There is a way of thread execution inside the process of any operating system. Apart from this, there can be more than one thread inside a process. Each thread of the same process makes use of a separate program counter and a stack of activation records and control blocks. Thread is often referred to as a lightweight process.



The process can be split down into so many threads. **For example**, in a browser, many tabs can be viewed as threads. MS Word uses many threads - formatting text from one thread, processing input from another thread, etc.

Types of Threads

In the [operating system](https://www.javatpoint.com/os-tutorial)

, there are two types of threads.

1. Kernel level thread.
2. User-level thread.

User-level thread

The [operating system](https://www.javatpoint.com/operating-system)

does not recognize the user-level thread. User threads can be easily implemented and it is implemented by the user. If a user performs a user-level thread blocking operation, the whole process is blocked. The kernel level thread does not know nothing about the user level thread. The kernel-level thread manages user-level threads as if they are single-threaded processes? Examples: [Java](https://www.javatpoint.com/java-tutorial)

thread, POSIX threads, etc.

30.1M

1

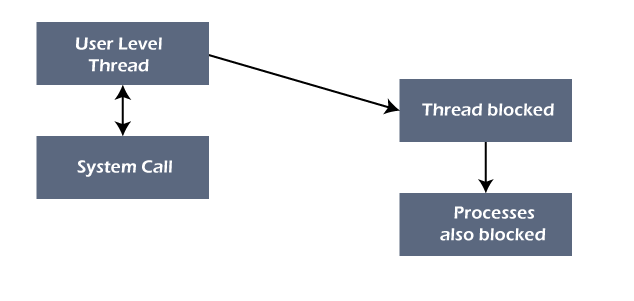
C++ vs Java

**Advantages of User-level threads**

1. The user threads can be easily implemented than the kernel thread.
2. User-level threads can be applied to such types of operating systems that do not support threads at the kernel-level.
3. It is faster and efficient.
4. Context switch time is shorter than the kernel-level threads.
5. It does not require modifications of the operating system.
6. User-level threads representation is very simple. The register, PC, stack, and mini thread control blocks are stored in the address space of the user-level process.
7. It is simple to create, switch, and synchronize threads without the intervention of the process.

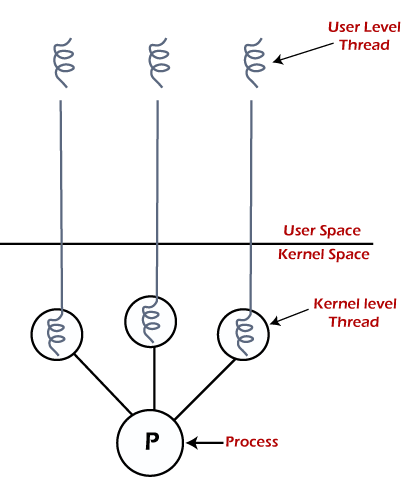
**Disadvantages of User-level threads**

1. User-level threads lack coordination between the thread and the kernel.
2. If a thread causes a page fault, the entire process is blocked.



Kernel level thread

The kernel thread recognizes the operating system. There is a thread control block and process control block in the system for each thread and process in the kernel-level thread. The kernel-level thread is implemented by the operating system. The kernel knows about all the threads and manages them. The kernel-level thread offers a system call to create and manage the threads from user-space. The implementation of kernel threads is more difficult than the user thread. Context switch time is longer in the kernel thread. If a kernel thread performs a blocking operation, the Banky thread execution can continue. Example: Window Solaris.



**Advantages of Kernel-level threads**

1. The kernel-level thread is fully aware of all threads.
2. The scheduler may decide to spend more CPU time in the process of threads being large numerical.
3. The kernel-level thread is good for those applications that block the frequency.

**Disadvantages of Kernel-level threads**

1. The kernel thread manages and schedules all threads.
2. The implementation of kernel threads is difficult than the user thread.
3. The kernel-level thread is slower than user-level threads.

Benefits of Threads

* **Enhanced throughput of the system:** When the process is split into many threads, and each thread is treated as a job, the number of jobs done in the unit time increases. That is why the throughput of the system also increases.
* **Effective Utilization of Multiprocessor system:** When you have more than one thread in one process, you can schedule more than one thread in more than one processor.
* **Faster context switch:** The context switching period between threads is less than the process context switching. The process context switch means more overhead for the CPU.
* **Responsiveness:** When the process is split into several threads, and when a thread completes its execution, that process can be responded to as soon as possible.
* **Communication:** Multiple-thread communication is simple because the threads share the same address space, while in process, we adopt just a few exclusive communication strategies for communication between two processes.
* **Resource sharing:** Resources can be shared between all threads within a process, such as code, data, and files. Note: The stack and register cannot be shared between threads. There is a stack and register for each thread.

Multithreading Models in Operating system

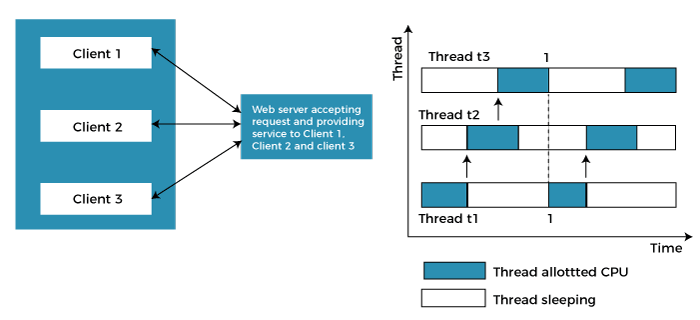
In this article, we will understand the multithreading model in the Operating system.

Multithreading Model:

Multithreading allows the application to divide its task into individual threads. In multi-threads, the same process or task can be done by the number of threads, or we can say that there is more than one thread to perform the task in multithreading. With the use of multithreading, multitasking can be achieved.



The main drawback of single threading systems is that only one task can be performed at a time, so to overcome the drawback of this single threading, there is multithreading that allows multiple tasks to be performed.



In the above example, client1, client2, and client3 are accessing the web server without any waiting. In multithreading, several tasks can run at the same time.

In an [operating system](https://www.javatpoint.com/os-tutorial)

threads are divided into the user-level thread and the Kernel-level thread. User-level threads handled independent form above the kernel and thereby managed without any kernel support. On the opposite hand, the operating system directly manages the kernel-level threads. Nevertheless, there must be a form of relationship between user-level and kernel-level threads.

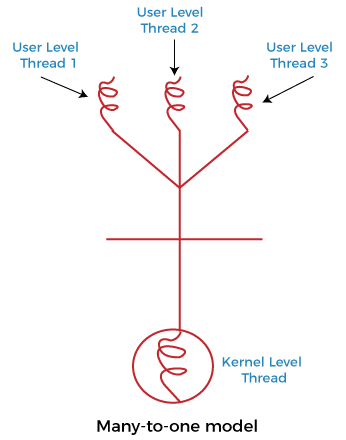
**There exists three established multithreading models classifying these relationships are:**

* Many to one multithreading model
* One to one multithreading model
* Many to Many multithreading models

Many to one multithreading model:

The many to one model maps many user levels threads to one kernel thread. This type of relationship facilitates an effective context-switching environment, easily implemented even on the simple kernel with no thread support.

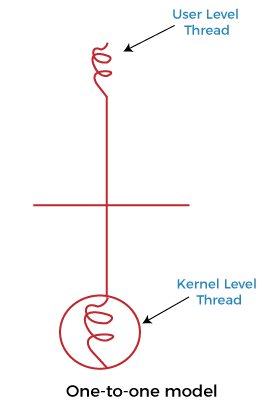
The disadvantage of this model is that since there is only one kernel-level thread schedule at any given time, this model cannot take advantage of the hardware acceleration offered by multithreaded processes or multi-processor systems. In this, all the thread management is done in the user space. If blocking comes, this model blocks the whole system.



In the above figure, the many to one model associates all user-level threads to single kernel-level threads.

One to one multithreading model

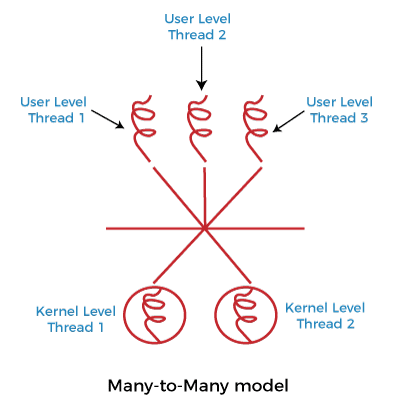
The one-to-one model maps a single user-level thread to a single kernel-level thread. This type of relationship facilitates the running of multiple threads in parallel. However, this benefit comes with its drawback. The generation of every new user thread must include creating a corresponding kernel thread causing an overhead, which can hinder the performance of the parent process. Windows series and Linux operating systems try to tackle this problem by limiting the growth of the thread count.



In the above figure, one model associates that one user-level thread to a single kernel-level thread.

Many to Many Models multithreading model

In this type of model, there are several user-level threads and several kernel-level threads. The number of kernel threads created depends upon a particular application. The developer can create as many threads at both levels but may not be the same. The many to many model is a compromise between the other two models. In this model, if any thread makes a blocking system call, the kernel can schedule another thread for execution. Also, with the introduction of multiple threads, complexity is not present as in the previous models. Though this model allows the creation of multiple kernel threads, true concurrency cannot be achieved by this model. This is because the kernel can schedule only one process at a time.



Many to many versions of the multithreading model associate several user-level threads to the same or much less variety of kernel-level threads in the above figure.