Difference Between fork() and vfork()

Both **fork()** and **vfork()** are the **system calls** that creates a new process that is identical to the process that invoked fork() or vfork().

Using **fork()** allows the execution of parent and child process simultaneously. The other way, **vfork()** suspends the execution of parent process until child process completes its execution.

The primary difference between the fork() and vfork() system call is that the child process created using fork has separate address space as that of the parent process. On the other hand, child process created using vfork has to share the address space of its parent process.

Comparison Chart

Basis for Comparison	**fork()**	**vfork()**
Basic	Child process and parent process has separate address spaces.	Child process and parent process shares the same address space.
Execution	Parent and child process execute simultaneously.	Parent process remains suspended till child process completes its execution.
Modification	If the child process alters any page in the address space, it is invisible to the parent process as the address space are separate.	If child process alters any page in the address space, it is visible to the parent process as they share the same address space.
Copy-on-write	fork() uses copy-on-write as an alternative where the parent and child shares same pages until any one of them modifies the shared page.	vfork() does not use copy-on-write.

Definition of fork()

The **fork()** is a system call use to create a **new process**. The new process created by the fork() call is the child process, of the process that invoked the fork() system call. The code of child process is identical to the code of its parent process. After the creation of child process, both process i.e. parent and child process start their execution from the next statement after fork() and both the processes get executed **simultaneously**.

The parent process and child process do have **separate address space**. Hence, when any of the processes modifies any statement or variable in the code. It would not be reflected in other process codes. Let’s suppose if child process modifies the code it would not affect the parent process.

Some child process after their creation immediately calls **exec()**. The exec() system call **replaces the process** with the program specified in its parameter. Then the separate address space of child process is of no use. The one alternative here is copy-on-write.

The **copy-on-write** let the parent and child process to share same address space. If the any of the processes writes on the pages in address space the copy of address space is created to let both the process work independently.

Definition of vfork()

The modified version of fork() is vfork(). The **vfork()** system call is also used to create a new process. Similar to the fork(), here also the new process created is the child process, of the process that invoked vfork(). The child process code is also identical to the parent process code. Here,the child process **suspends the execution** of parent process till it completes its execution as both the process share the same address space to use.

As the child and parent process shares the **same address space**. If any of the processes modifies the code, it is visible to the other process sharing the same pages. Let us suppose if the parent process alters the code; it will reflect in the code of child process.

As using vfork() does not create separate address spaces for child and parent processes. Hence, it must be **implemented** where the child process calls **exec()** immediately after its creation. So, there will be no wastage of address space, and it is the **efficient** way to create a process. vfork does not use **copy-on-write**.

Key Differences Between fork() and vfork()

1. The primary difference between fork and vfork is that the child process created by the **fork** has a **separate memory space** from the parent process. However, the child process created by the **vfork** system call shares the **same address space** of its parent process.
2. The child process created using fork **execute simultaneously** with the parent process. On the other hand, child process created using vfork **suspend** the execution of parent process till its execution is completed.
3. As the memory space of parent and child process is separate modification done by any of the processes does not affect other's pages. However, as the parent and child process shares the same memory address modification done by any process reflects in the address space.
4. The system call fork() uses **copy-on-write** as an alternative, which let child and parent process share the same address space until any one of them modifies the pages. On the other hand, the vfork does not use copy-on-wri

Conclusion:

The vfork() system call has to be implemented when child process call exec() immediately after its creation using fork(). As separate address space for child and parent process will be of no use here.

Difference Between Program and Process

A program and a process are related terms. The major difference between program and process is that program is a group of instructions to carry out a specified task whereas the process is a program in execution. While a process is an active entity, a program is considered to be a passive one.

There exist a many-to-one relationship between process and program, which means one program can invoke multiple processes or in other words multiple processes can be a part of the same program.

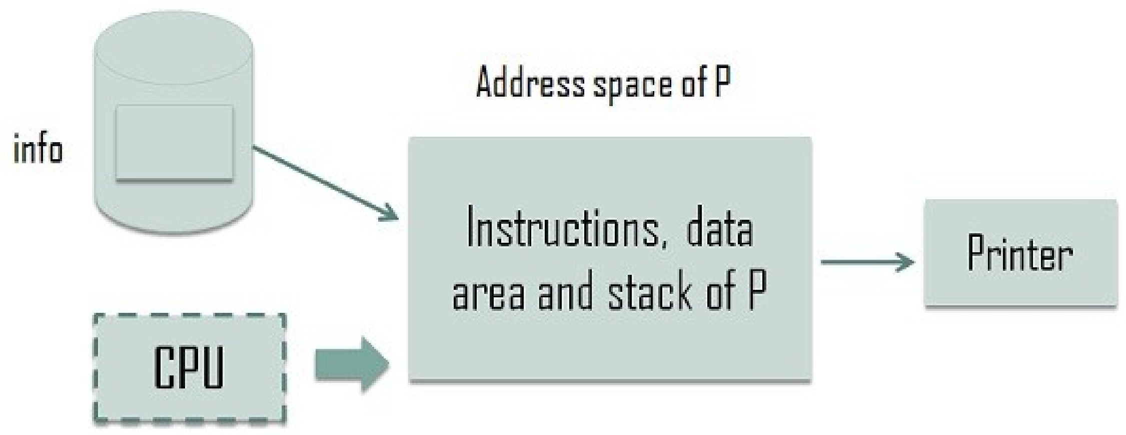
Basis for comparison	Program	Process
Basic	Program is a set of instruction.	When a program is executed, it is known as process.
Nature	Passive	Active
Lifespan	Longer	Limited
Required resources	Program is stored on disk in some file and does not require any other resources.	Process holds resources such as CPU, memory address, disk, I/O etc.

Definition of Program

A **Program**, in simple words, can be considered as a system activity. In batch processing system these are called executing jobs while in a real-time operating system it is called tasks or programs. A user can run multiple programs where the operating system facilitates its own internal programmed activities such as memory management using some techniques.

A program is a **passive entity,** for example, a file accommodating a group of instructions to be executed (executable file). It is so called because it doesn't perform any action by itself, it has to be executed to realize the actions specified in it.

The address space of a program is composed of the instruction, data and stack. Assume P is the program we are writing, to realize execution of P, the operating system allocates memory to accommodate P's address space.



It schedules P for execution and also sets up an arrangement through which p can access file info. The CPU is shown in the dashed box because it is not always executing instructions of P; in fact, OS shares the CPU between the execution of P and executions of other programs.

Definition of Process

A **Process** is an execution of a program. It is considered as an **active entity** and realizes the actions specified in a program. Multiple processes can be related to the same program. It handles the operating system activities through **PCB (Process control Block)** which includes program counter, stack, state etc. Program counter stores the next sequence of instruction that is to be executed later.

It needs resources like processing, memory and I/O resources to accomplish management tasks. During the execution of a program, it could engage processor or I/O operation that makes a process different from a program.

Let us understand this from an example; we are writing a C program. While writing and storing a program in a file, it is just a script and does not perform any action, but when it is executed it turns into process hence process is dynamic in nature. Sharing of resources among multiple processes is employed by current machines, but in actual a single processor is distributed among several processes.

Key Differences Between Program and Process

1. A program is a definite group of **ordered operations** that are to be performed. On the other hand, an **instance** of a program being executed is a process.
2. The nature of the program is passive as it does nothing until it gets executed whereas a process is dynamic or active in nature as it is an instance of executing program and perform the specific action.
3. A program has a **longer** lifespan because it is stored in the memory until it is not manually deleted while a process has a shorter and **limited** lifespan because it gets terminated after the completion of the task.
4. The resource requirement is much higher in case of a process; it could need processing, memory, I/O resources for the successful execution. In contrast, a program just requires memory for storage.

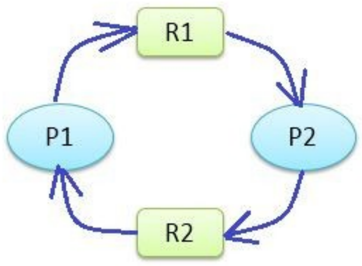
Conclusion

Program and process are relevant but are dissimilar. A program is just a script stored on disk or seem to as the previous stage of the process. On the contrary, the process is an event of a program in execution

Definition of Deadlock

Deadlock is a situation where the several processes in CPU compete for the finite number of resources available within the CPU. Here, each process holds a resource and wait to acquire a resource that is held by some other process. All the processes wait for resources in a circular fashion. In the image below, you can see that Process P1 has acquired resource R2 that is requested by process P2 and Process P1 is requesting for resource R1 which is again held by R2. So process

P1 and P2 form a deadlock.

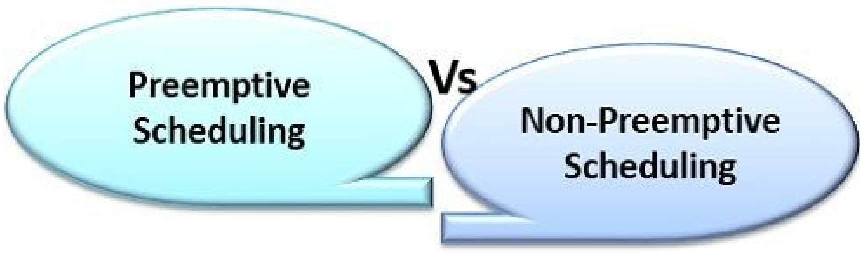


Deadlock is a common problem in multiprocessing operating systems, distributed systems, and also in parallel computing systems. There are four conditions which must occur simultaneously to raise the condition of deadlock, which are Mutual exclusion, Hold and waits, No preemption, and Circular wait.

- **Mutual exclusion:** Only one process at a time can use a resource if other process requests the same resource, it has to wait till the process using resource releases it.
- **Hold and Wait:** A process must be holding a resource and waiting to acquire another resource that is held by some other process.
- **No Preemption:** The process holding the resources can not be preempted. The process holding the resource must release the resource voluntarily when it has completed its task.
- **Circular wait:** The process must wait for resources in a circular fashion. Suppose we have three processes {P0, P1, P2}. The P0 must wait for the resource held by P1; P1 must wait to acquire the resource held by process P2, and P2 must wait to acquire the process held by P0.

Although there are some applications that can detect the programs that may get deadlocked. But the operating system is never responsible for preventing the deadlocks. It is the responsibility of programmers to design deadlock free programs. It can be done by avoiding the above conditions which are necessary for deadlock occurrence

Difference Between Preemptive and Non-Preemptive Scheduling in OS



It is the responsibility of CPU scheduler to allot a process to CPU whenever the CPU is in the idle state. The CPU scheduler selects a process from ready queue and allocates the process to CPU. The scheduling which takes place when a process switches from running state to ready state or from waiting state to ready state is called **Preemptive Scheduling**. On the hands, the scheduling which takes place when a process terminates or switches from running to waiting for state this kind of CPU scheduling is called **Non-Preemptive Scheduling**. The basic difference between preemptive and non-preemptive scheduling lies in their name itself. That is a Preemptive scheduling can be preempted; the processes can be scheduled. In Non-preemptive scheduling, the processes can not be scheduled.

Let us discuss the differences between the both Preemptive and Non-Preemptive Scheduling in brief with the help of comparison chart shown below.

Basis for Comparison	Preemptive Scheduling	Non Preemptive Scheduling
Basic	The resources are allocated to a process for a limited time.	Once resources are allocated to a process, the process holds it till it completes its burst time or switches to waiting state.
Interrupt	Process can be interrupted in between.	Process can not be interrupted till it terminates or switches to waiting state.
Starvation	If a high priority process frequently arrives in the ready queue, low priority process may starve.	If a process with long burst time is running CPU, then another process with less CPU burst time may starve.
Overhead	Preemptive scheduling has overheads of scheduling the processes.	Non-preemptive scheduling does not have overheads.
Flexibility	Preemptive scheduling is flexible.	Non-preemptive scheduling is rigid.
Cost	Preemptive scheduling is cost associated.	Non-preemptive scheduling is not cost associative.

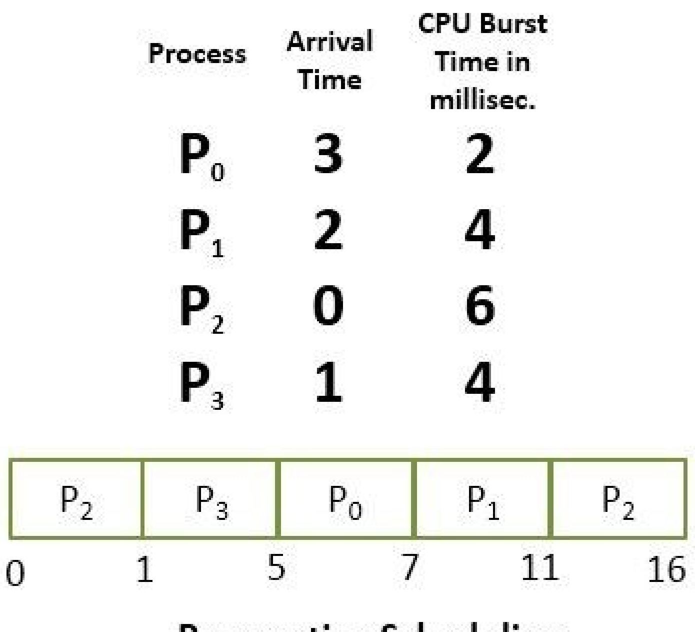
Definition of Preemptive Scheduling

Preemptive scheduling is one which can be done in the circumstances when a process switches from **running state** to **ready state** or from **waiting state** to **ready state**. Here, the resources (CPU cycles) are allocated to the process for the **limited** amount of time and then is taken away, and the process is placed back in the ready queue again if it still has CPU burst time remaining. The process stays in ready queue till it gets next chance to execute.

If a process with high priority arrives in the ready queue, it does not have to wait for the current process to complete its burst time. Instead, the current process is interrupted in the middle of execution and is placed in the ready queue till the process with high priority is utilizing the CPU cycles. In this way, each process in the ready queue gets some time to run CPU. It makes the preemptive scheduling flexible but, increases the overhead of switching the process from running state to ready state and vise-verse.

Algorithms that work on preemptive scheduling are Round Robin. Shortest Job First (SJF) and Priority scheduling may or may not come under preemptive scheduling.

Let us take an example of Preemptive Scheduling, look in the picture below. We have four processes P0, P1, P2, P3. Out of which, P2 arrives at time 0. So the CPU is allocated to the process P2 as there is no other process in the queue. Meanwhile, P2 was executing, P3 arrives at time 1, now the remaining time for process P2 (5 milliseconds) which is larger than the time required by P3 (4 milli-sec). So CPU is allocated to processor P3.



Process	Arrival Time	CPU Burst Time in millisec.
P_0	3	2
P_1	2	4
P_2	0	6
P_3	1	4

Preemptive Scheduling

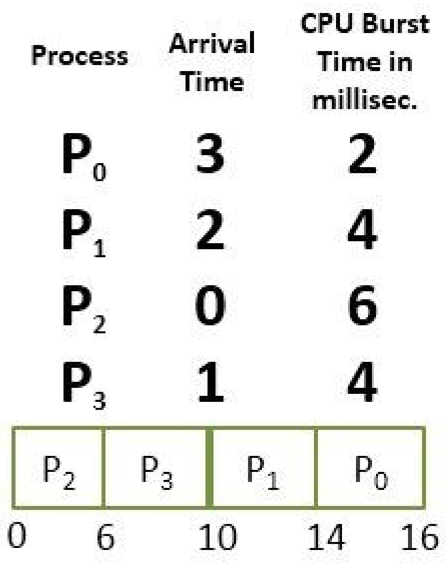
Meanwhile, P3 was executing, process P1 arrives at time 2. Now the remaining time for P3 (3 milliseconds) is less than the time required by processes P1 (4 milliseconds) and P2 (5 milliseconds). So P3 is allowed to continue. While P3 is continuing process P0 arrives at time 3, now the remaining time for P3 (2 milliseconds) is equal to the time require by P0 (2 milliseconds). So P3 continues and after P3 terminates the CPU is allocated to P0 as it has less burst time than other. After P0 terminates, the CPU is allocated to P1 and then to P2.

Definition of Non-Preemptive Scheduling

Non-preemptive Scheduling is one which can be applied in the circumstances when a process **terminates**, or a process switches from **running** to **ready state**. In Non-Preemptive Scheduling, once the resources (CPU) is allocated to a process, the process holds the CPU till it gets terminated or it reaches a waiting state.

Like preemptive scheduling, non-preemptive scheduling does not interrupt a process running CPU in middle of the execution. Instead, it waits for the process to complete its CPU burst time and then it can allocate the CPU to another process.

In Non-preemptive scheduling, if a process with long CPU burst time is executing then the other process will have to wait for a long time which increases the average waiting time of the processes in the ready queue. However, the non-preemptive scheduling does not have any overhead of switching the processes from ready queue to CPU but it makes the scheduling rigid as the process in execution is not even preempted for a process with higher priority.



Process	Arrival Time	CPU Burst Time in millisec.
P_0	3	2
P_1	2	4
P_2	0	6
P_3	1	4

Non-Preemptive Scheduling

Let us solve the above scheduling example in non-preemptive fashion. As initially the process P2 arrives at time 0, so CPU is allocated to the process P2 it takes 6 milliseconds to execute. In between all the processes i.e. P0, P1, P3 arrives into ready queue. But all waits till process P2 completes its CPU burst time. Then process that arrives after P2 i.e. P3 is then allocated the CPU till it finishes it’s burst time. Similarly, then P1 executes, and CPU is later given to process P0.

Key Differences Between Preemptive and Non-Preemptive Scheduling

1. The basic difference between preemptive and non-preemptive scheduling is that in preemptive scheduling the CPU is allocated to the processes for the **limited** time. While in Non-preemptive scheduling, the CPU is allocated to the process till it **terminates** or switches to **waiting state**.
2. The executing process in preemptive scheduling is **interrupted** in the middle of execution whereas, the executing process in non-preemptive scheduling is **not interrupted** in the middle of execution.

3. Preemptive Scheduling has the **overhead** of switching the process from ready state to running state, vise-verse, and maintaining the ready queue. On the other hands, non-preemptive scheduling has **no overhead** of switching the process from running state to ready state.
4. In preemptive scheduling, if a process with high priority frequently arrives in the ready queue then the process with low priority have to wait for a long, and it may have to starve. On the other hands, in the non-preemptive scheduling, if CPU is allocated to the process with larger burst time then the processes with small burst time may have to starve.
5. Preemptive scheduling is quite **flexible** because the critical processes are allowed to access CPU as they arrive into the ready queue, no matter what process is executing currently. Non-preemptive scheduling is **rigid** as even if a critical process enters the ready queue the process running CPU is not disturbed.
6. The Preemptive Scheduling is cost associative as it has to maintain the integrity of shared data which is not the case with Non-preemptive Scheduling.

Conclusion:

It is not that preemptive scheduling is better than non-preemptive scheduling or vise-verse. All depends on how a scheduling minimizes the average waiting time of the processes and maximizes CPU utilization.

Difference between Process and Thread

Leave a Comment

The Process and Thread are the essentially associated. The process is an execution of a program whereas thread is an execution of a program driven by the environment of a process.

Another major point which differentiates process and thread is that processes are isolated with each other whereas threads share memory or resources with each other.

Basis for comparison	Process	Thread
Basic	Program in execution.	Lightweight process or part of it.
Memory sharing	Completely isolated and do not share memory.	Shares memory with each other.
Resource consumption	More	Less
Efficiency	Less efficient as compared to the process in the context of communication.	Enhances efficiency in the context of communication.
Time required for creation	More	Less
Context switching time	Takes more time.	Consumes less time.
Uncertain termination	Results in loss of process.	A thread can be reclaimed.
Time required for termination	More	Less

Definition of Process

The process is the execution of a program and performs the relevant actions specified in a program, or it is an execution unit where a program runs. The operating system creates, schedules and terminates the processes for the use of the CPU. The other processes created by the main process are known as child process.

A process operations are controlled with the help of PCB(Process control Block) can be considered as the brain of the process, which contains all the crucial information regarding to a process such as a process id, priority, state, PWS and contents CPU register.

PCB is also a kernel-based data structure which uses the three kinds of functions which are scheduling, dispatching and context save.

- **Scheduling –** It is a method of selecting the sequence of the process in simple words chooses the process which has to be executed first in the CPU.
- **Dispatching –** It sets up an environment for the process to be executed.
- **Context save –** This function saves the information regarding to a process when it gets resumed or blocked.

There are certain states included in a process lifecycle such as ready, running, blocked and terminated. Process States are used for keeping the track of the process activity at an instant.

From the programmer’s point of view, processes are the medium to achieve the concurrent execution of a program. The chief process of a concurrent program creates a child process. The main process and child process need to interact with each to achieve a common goal.

Interleaving operations of processes enhance the computation speed when i/o operation in one process overlaps with a computational activity in another process.

Properties of a Process:

- Creation of each process includes system calls for each process separately.
- A process is an isolated execution entity and does not share data and information.
- Processes use IPC (Inter-process communication) mechanism for communication which significantly increases the number of system calls.
- Process management consumes more system calls.
- Each process has its own stack and heap memory, instruction, data and memory map.

Definition of Thread

The thread is a program execution that uses process resources for accomplishing the task. All threads within a single program are logically contained within a process. The kernel allocates a stack and a thread control block (TCB) to each thread. The operating system saves only the stack pointer and CPU state at the time of switching between the threads of the same process.

Threads are implemented in three different ways; these are kernel-level threads, user-level threads, hybrid threads. Threads can have three states running, ready and blocked; it only includes computational state not resource allocation and communication state which reduces the switching overhead. It enhances the concurrency (parallelism) hence speed also increases.

Multithreading also comes with demerits, Multiple threads doesn’t create complexity, but the interaction between them does.

A thread must have priority property when there are multiple threads are active. The time it gets for execution relative to other active threads in the same process is specified by the priority of the thread.

Properties of a Thread:

- Only one system call can create more than one thread (Lightweight process).
- Threads share data and information.
- Threads shares instruction, global and heap regions but has its own individual stack and registers.
- Thread management consumes no or fewer system calls as the communication between threads can be achieved using shared memory.
- The isolation property of the process increases its overhead in terms of resource consumption.

Key Differences Between Process and Thread

1. All threads of a program are logically contained within a process.
2. A process is heavy weighted, but a thread is light weighted.
3. A program is an isolated execution unit whereas thread is not isolated and shares memory.
4. A thread cannot have an individual existence; it is attached to a process. On the other hand, a process can exist individually.
5. At the time of expiration of a thread, its associated stack could be recovered as every thread has its own stack. In contrast, if a process dies, all threads die including the process.

Conclusion

Processes are used to achieve execution of programs in concurrent and sequential manner. While a thread is a program execution unit which uses the environment of the process when many threads use the environment of the same process they need to share its code, data and resources. The operating system uses this fact to reduce the overhead and improve computation.

Difference Between Interrupt and Polling in OS

We have many external devices attached to the CPU like a mouse, keyboard, scanner, printer, etc. These devices also need CPU attention. Suppose, a CPU is busy in displaying a PDF and you click the window media player icon on the desktop. Though the CPU does not have any idea when an event like this would occur, but it has to respond to such inputs from the I/O devices. Interrupt and Polling are the two ways to handle the events generated by the devices that can happen at any moment while CPU is busy in executing another process.

Polling and Interrupt let CPU stop what it is currently doing and respond to the more important task. Polling and Interrupt are different from each other in many aspects. But the basic point that distinguishes Polling and Interrupt is that in **polling** CPU keeps on checking I/O devices at regular interval whether it needs CPU service whereas, in **interrupt**, the I/O device interrupts the CPU and tell CPU that it need CPU service. I have discussed some differences between Interrupt and Polling in the comparison chart below, please have a look.

Basis for Comparison	Interrupt	Polling
Basic	Device notify CPU that it needs CPU attention.	CPU constantly checks device status whether it needs CPU's attention.
Mechanism	An interrupt is a hardware mechanism.	Polling is a Protocol.
Servicing	Interrupt handler services the Device.	CPU services the device.
Indication	Interrupt-request line indicates that device needs servicing.	Comand-ready bit indicates the device needs servicing.
CPU	CPU is disturbed only when a device needs servicing, which saves CPU cycles.	CPU has to wait and check whether a device needs servicing which wastes lots of CPU cycles.
Occurrence	An interrupt can occur at any time.	CPU polls the devices at regular interval.
Efficiency	Interrupt becomes inefficient when devices keep on interrupting the CPU repeatedly.	Polling becomes inefficient when CPU rarely finds a device ready for service.
Example	Let the bell ring then open the door to check who has come.	Constantly keep on opening the door to check whether anybody has come.

Definition of Interrupt

An interrupt is a **hardware mechanism** that enables CPU to detect that a device needs its attention. The CPU has a wire **interrupt-request line** which is checked by CPU after execution of every single instruction. When CPU senses an interrupt signal on the interrupt-request line, CPU stops its currently executing task and respond to the interrupt send by I/O device by passing the control to **interrupt handler**. The interrupt handler resolves the interrupt by servicing the device.

Although CPU is not aware when an interrupt would occur as it can occur at any moment, but it has to respond to the interrupt whenever it occurs.

When the interrupt handler finishes executing the interrupt, then the CPU **resumes** the execution of the task that it has stopped for responding the interrupt. **Software**, **hardware**, **user**, **some error in the program**, etc. can also generate an interrupt. Interrupts handling nature of CPU leads to **multitasking**, i.e. a user can perform a number of different tasks at the same time.

If more than one interrupts are sent to the CPU, the interrupt handler helps in managing the interrupts that are waiting to be processed. As interrupt handler gets **triggered** by the reception of an interrupt, it **prioritizes** the interrupts waiting to be processed by the CPU and arranges them in a **queue** to get serviced.

Definition of Polling

As we have seen in interrupts, the input from I/O device can arrive at any moment requesting the CPU to process it. Polling is a **protocol** that notifies CPU that a device needs its attention. Unlike in interrupt, where device tells CPU that it needs CPU processing, in polling CPU keeps **asking** the I/O device whether it needs CPU processing.

The CPU **continuously** test each and every device attached to it for detecting whether any device needs CPU attention. Every **device** has a **command-ready** bit which indicates the status of that device i.e. whether it has some command to be executed by CPU or not. If command bit is set **1**, then it has some command to be executed else if the bit is **0**, then it has no commands. **CPU** has a **busy bit** that indicates the status of CPU whether it is busy or not. If the busy bit is set **1**, then it is busy in executing the command of some device, else it is **0**.

Algorithm for polling

- When a device has some command to be executed by CPU it continuously checks the busy bit of CPU until it becomes clear (0).
- As the busy bit becomes clear, the device set write-bit in its command register and writes a byte in data-out register.
- Now the device sets (1) the command-ready bit.
- When CPU checks the devices command-ready bit and finds it set (1), it sets (1) its busy bit.
- The CPU then reads the command register of the device and executes the command of the device.
- After command execution, CPU clears(0) the command-ready bit, error bit of the device to indicate successful execution of the command of the device and further it clears (0) its busy bit also to indicate that the CPU is free to execute the command of some other device.

Key Differences Between Interrupt and Polling in OS

1. In interrupt, the device notifies the CPU that it needs servicing whereas, in polling CPU repeatedly checks whether a device needs servicing.
2. Interrupt is a **hardware mechanism** as CPU has a wire, **interrupt-request line** which signal that interrupt has occurred. On the other hands, Polling is a **protocol** that keeps checking the **control bits** to notify whether a device has something to execute.
3. **Interrupt handler** handles the interrupts generated by the devices. On the other hands, in polling, **CPU** services the device when they require.
4. Interrupts are signalled by the **interrupt-request line**. However, **Command-ready** bit indicate that the device needs servicing.
5. In interrupts, CPU is only disturbed when any device interrupts it. On the other hand, in polling, CPU waste lots of CPU cycles by repeatedly checking the command-ready bit of every device.
6. An interrupt can occur at **any instant of time** whereas, CPU keeps polling the device at the **regular intervals**.
7. Polling becomes inefficient when CPU keeps on polling the device and rarely finds any device ready for servicing. On the other hands, interrupts become inefficient when the devices keep on interrupting the CPU processing repeatedly.

Conclusion:

Both Polling and Interrupts are efficient in attending the I/O devices. But they can become inefficient at the certain condition as discussed above.

Difference Between RAM and ROM Memory

RAM and ROM both are the internal memories of the computer. Where **RAM** is a **temporary** memory, **ROM** is a **permanent** memory of the computer. There are many differences between RAM and ROM, but the basic difference is that RAM is a **read-write** memory and the ROM is a **read only** memory. I have discussed some differences between RAM and ROM with the help of comparison chart shown below.

Comparison Chart

Basis for Comparison	RAM	ROM
Basic	It is a read-write memory.	It is read only memory.
Use	Used to store the data that has to be currently processed by CPU temporarily.	It stores the instructions required during bootstrap of the computer.
Volatility	It is a volatile memory.	It is a nonvolatile memory.
Stands for	Random Access Memory.	Read Only Memory.
Modification	Data in ROM can be modified.	Data in ROM can not be modified.

Basis for Comparison	RAM	ROM
Capacity	RAM sizes from 64 MB to 4GB.	ROM is comparatively smaller than RAM.
Cost	RAM is a costlier memory.	ROM is comparatively cheaper than RAM.
Type	Types of RAM are static RAM and dynamic RAM.	Types of ROM are PROM, EPROM, EEPROM.

Definition of RAM

RAM is a **Random access memory**; it means the CPU can **directly** access any address location of RAM memory. RAM is a quickly accessible memory of the computer. It stores the data **temporarily**.

RAM is a **volatile** memory. RAM stores the data till the power is switched on. Once the power of the CPU is switched off the whole data in RAM gets erased. The data which has to be **currently processed** must be in RAM. The storage capacity of the RAM ranges from 64 MB to 4 GB.

RAM is the **fastest** and **costliest** memory of the computer. It is a **read-write** memory of the computer. The processor can read the instructions from RAM and write the result to the RAM. The data in RAM can be **modified**.

There are two kinds of RAM, **Static RAM** and **Dynamic RAM**. **Static RAM** is one which requires the constant flow of the power to retain the data inside it. It is **faster** and **more expensive** than DRAM. It is used as a **cache memory** for the computer. **Dynamic RAM** needs to be refreshed to retain the data it holds. It is **slower** and **cheaper** than static RAM.

Definition of ROM

ROM is a **Read Only Memory**. The data in ROM can only be read by CPU but, it can not be modified. The CPU can **not directly** access the ROM memory, the data has to be first transferred to the RAM, and then the CPU can access that data from the RAM.

ROM stores the instruction that computer requires during **Bootstraping** (a process of booting up of the computer). The content in ROM can not be modified. ROM is a **non-volatile** memory, the data inside the ROM retains even if the power of the CPU is switched off.

The **capacity** of ROM is comparatively **smaller** than RAM, it **slower** and **cheaper** than RAM. There many kinds of ROM that are as follow:

PROM: Programmable ROM, it can be modified only once by the user.

EPROM: Erasable and Programable ROM, the content of this ROM can be erased using ultraviolet rays and the ROm can be reprogrammed.

EEPROM: Electrically Erasable and Programmable ROM, it can be erased electrically and reprogrammed about ten thousand times.

Key Differences Between RAM and ROM Memory

1. The key difference between RAM and ROM is that RAM is basically a **read-write** memory whereas, ROM is a **read only** memory.
2. RAM temporarily stores the data that have to be processed by CPU currently. On the other hands, ROM stores the instructions that are required during Bootstrap.
3. RAM is a **volatile** memory. However, ROM is a **nonvolatile** memory.
4. RAM stands for **Random Access Memory** whereas, ROM stands for **Read Only Memory**.
5. On the one hand, where the data in RAM can be **modified easily**, the data in ROM can be **hardly or never be modified**.
6. The RAM can range from 64 MB to 4 GB whereas, the **ROM** is always comparatively **smaller** than RAM.
7. RAM is **costlier** than ROM.
8. RAM can be classified into **static** and **Dynamic RAM**. On the other hands, ROM can be classified into **PROM, EPROM and EEPROM**.

Conclusion:

RAM and ROM both are the necessary memory for the computer. ROM is a necessary for a computer to boot up. RAM is important for CPU processing.

Difference Between Virtual and Cache Memory in OS

Memory is a hardware device that is used to store the information either temporary or permanently. In this article, I have discussed the differences between virtual and cache memory. A **Cache memory** is a high-speed memory which is used to reduce the access time for data. On the other hands, **Virtual memory** is not exactly a physical memory it is a technique which extends the capacity of the main memory beyond its limit.

The major difference between virtual memory and the cache memory is that a **virtual memory** allows a user to execute programs that are larger than the main memory whereas, **cache memory** allows the quicker access to the data which has been recently used. We will discuss some more differences with the help of comparison chart shown below.

Comparison Chart

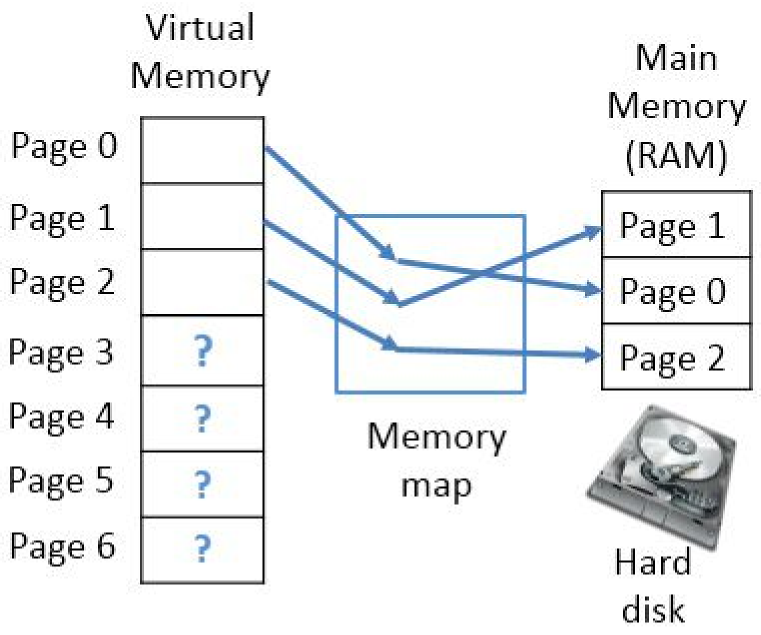
Basis for Comparison	Virtual Memory	Cached Memory
Basic	Virtual memory extends the capacity of main memory for the user.	Cache memory fastens the data accessing speed of CPU.

Basis for Comparison	Virtual Memory	Cached Memory
Nature	Virtual memory is technique.	Cache memory is a storage unit.
Function	Virtual memory allows execution of the program that is larger than the main memory.	Cache memory stores the copies of original data that has been recently used.
Memory management	Virtual memory is managed by the operating system.	Cache memory is fully managed by the hardware.
Size	Virtual memory is far larger than cached memory.	Cache memory has bounded size.
Mapping	Virtual memory requires mapping structures to map virtual address to physical address.	No mapping structures are required as such in a cache memory.

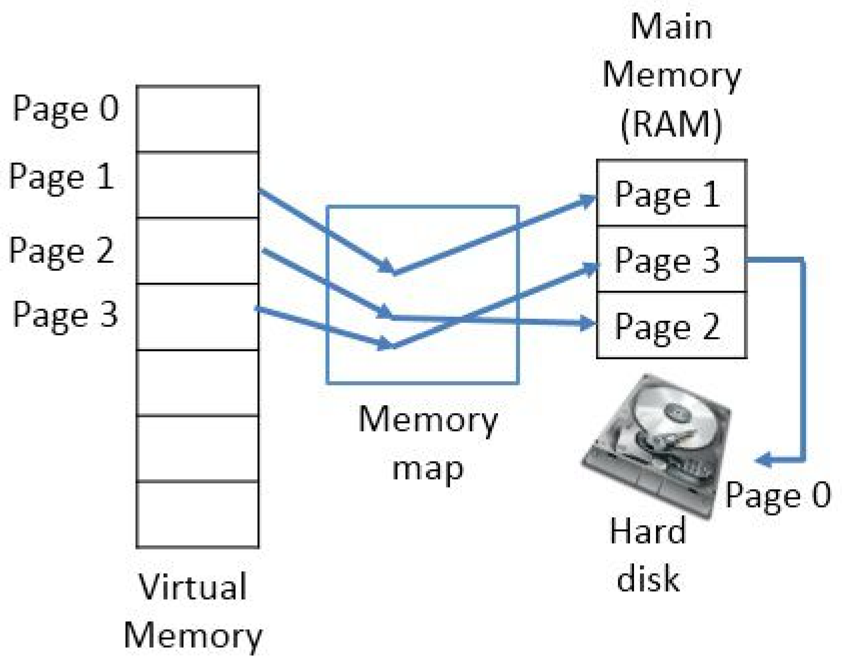
Definition of Virtual Memory

Virtual memory is not exactly a physical memory of a computer instead it's a **technique** that allows the execution of a **large program** that may **not** be **completely placed in the main memory**. It enables the programmer to execute the programs larger than the main memory.

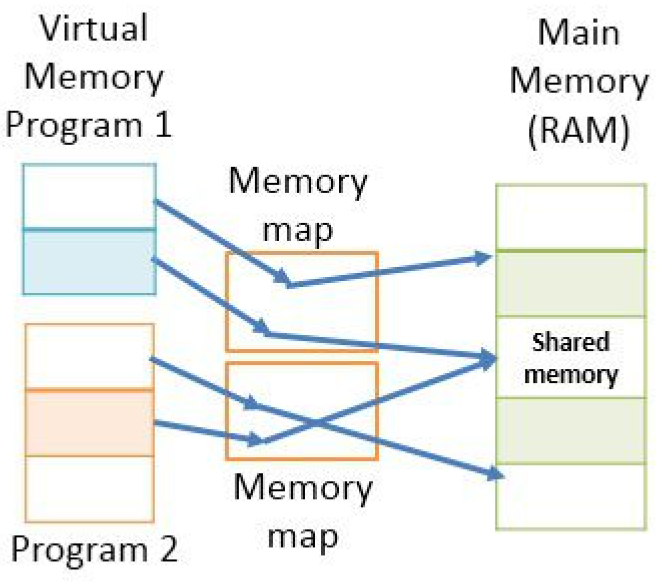
Now let us understand how does the virtual memory works? The program has its virtual memory address which is divided into a number of **pages**. The main memory is also divided into a number of **pages**. Now, as we can see the virtual address of a program is larger than the available main memory. So memory map is used to map the virtual address to the main memory.



Page 0, 1, 2, gets mapped in main memory and the main memory fills up. Now, when page 3 of virtual memory asks for space in main memory, the oldest placed page i.e. page 0 get transferred to the hard disk and evacuates the place for page 3, in main memory and the process goes on. In case the page 0 is again needed, then again the oldest placed page at that time is transferred to hard disk making the place for page 0.



If the two programs need same data, placed in main memory, the memory mapping unit allows both the programs to share the same address space in main memory where the **shared data** is stored. This makes **sharing** of files easy.



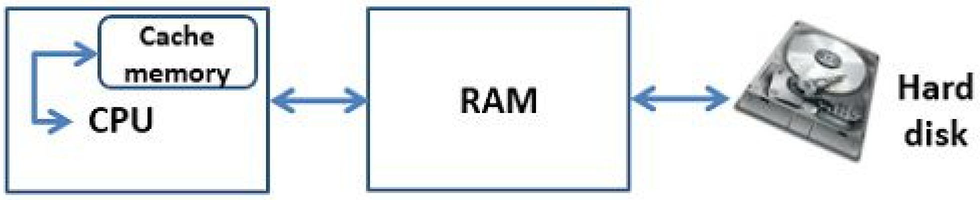
The benefits of virtual memory are:

- The programs are no longer constrained by the limit of main memory.
- Virtual memory increases the degree of multiprogramming.
- Increases CPU utilization.
- The less I/O unit will require to load or to swap programs in the memory.

But there is a **drawback** of virtual memory, placing more pages of a program in hard disk will **slow** down the **performance** as accessing the data from hard disk takes more time in comparison to accessing data from main memory.

Definition of Cache Memory

Unlike virtual memory, **Cache** is a **storage device** implemented on the **processor** itself. It carries the copies of original data that has been accessed recently. The original data may be placed in the main memory or a secondary memory. The cache memory **fastens** the accessing speed of data, but how? Let's understand.



Key Differences Between Virtual and Cache Memory

1. Virtual memory **extends** the capacity of main memory virtually for the user. However, the cache memory makes the accessing of data **faster** for CPU.
2. Cache is a memory **storage unit** whereas as the Virtual memory is a **technique**.
3. Virtual memory enables the executions of the program that **larger** than the main memory.On the other hands, cache memory stores the **copies** of original data that were used recently.
4. Virtual memory management is done by the **operating system**. On the other hands, cache memory management is done by the **hardware**.
5. Virtual memory is far **larger** than the cached memory in size.
6. Virtual memory technique requires the **mapping structures** to map virtual address to physical address whereas, cache memory **does not** require any mapping structures.

Conclusion:

The Virtual memory is a technique to expand the capacity of main memory virtually for the users. The cache memory is a storage unit that stores the recently accessed data which enables the CPU to access it faster.

Difference Between Paging and Segmentation in OS

The memory management in the operating system is an essential functionality, which allows the allocation of memory to the processes for execution and deallocates the memory when the process is no longer needed. In this article, we will discuss two memory management schemes paging and segmentation. If we talk about the basic differences between the paging and segmentation it is, a **page** is a fixed-sized block whereas, a **segment** is a variable-sized block. We will discuss some more differences between Paging and Segmentation with the help of comparison chart shown below.

Comparison Chart

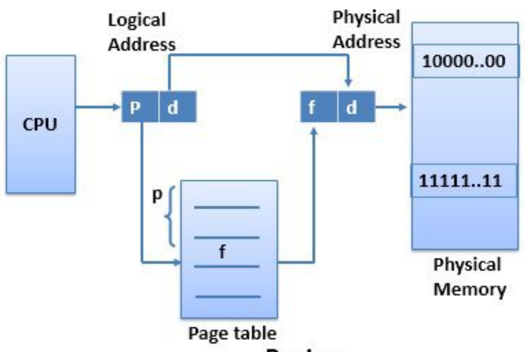
Basis for Comparison	Paging	Segmentation
Basic	A page is of fixed block size.	A segment is of variable size.
Fragmentation	Paging may lead to internal fragmentation.	Segmentation may lead to external fragmentation.
Address	The user specified address is divided by CPU into a page number and offset.	The user specifies each address by two quantities a segment number and the offset (Segment limit).
Size	The hardware decides the page size.	The segment size is specified by the user.
Table	Paging involves a page table that contains base address of each page.	Segmentation involves the segment table that contains segment number and offset (segment length).

Definition of Paging

Paging is a **memory management scheme**. Paging allows a process to be stored in a memory in a **non-contiguous** manner. Storing process in a non-contiguous manner solves the problem of **external fragmentation**.

For implementing paging the **physical and logical memory spaces** are divided into the same fixed-sized blocks. These fixed-sized blocks of physical memory are called **frames**, and the fixed-sized blocks of logical memory are called **pages**.

When a process needs to be executed the process pages from logical memory space are loaded into the frames of physical memory address space. Now the address generated by **CPU** for accessing the frame is divided into two parts i.e. **page number** and **page offset**.



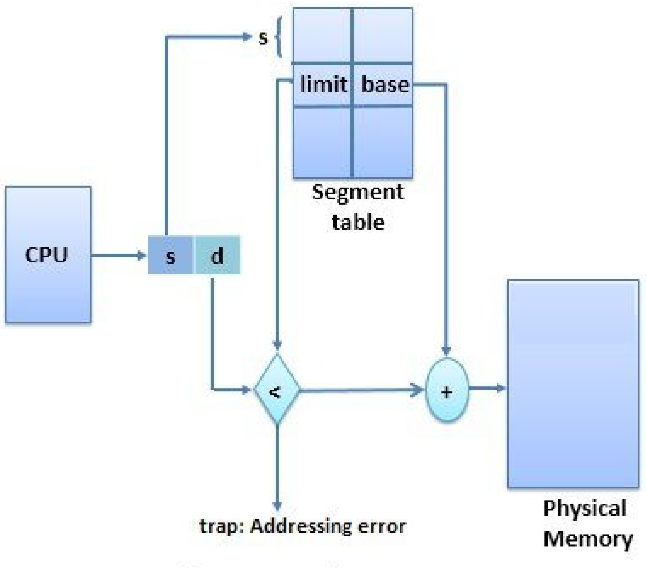
Paging

The **page table** uses page number as an index; each process has its separate page table that maps logical address to the physical address. The page table contains base address of the page stored in the frame of physical memory space. The base address defined by page table is combined with the page offset to define the frame number in physical memory where the page is stored.

Definition of Segmentation

Like Paging, **Segmentation** is also a **memory management scheme**. It supports the user's view of the memory. The process is divided into the **variable size segments** and loaded to the logical memory address space.

The logical address space is the collection of variable size segments. Each segment has its **name** and **length**. For the execution, the segments from logical memory space are loaded to the physical memory space.



Segmentation

The address specified by the user contain two quantities the **segment name** and the **Offset**. The segments are numbered and referred by the **segment number** instead of segment name. This segment number is used as an index in the **segment table**, and **offset** value decides the length or **limit** of the segment. The segment number and the offset together combinely generates the address of the segment in the physical memory space.

Key Differences Between Paging and Segmentation

1. The basic difference between paging and segmentation is that a page is always of **fixed block size** whereas, a segment is of **variable size**.

2. Paging may lead to **internal fragmentation** as the page is of fixed block size, but it may happen that the process does not acquire the entire block size which will generate the internal fragment in memory. The segmentation may lead to **external fragmentation** as the memory is filled with the variable sized blocks.
3. In paging the user only provides a **single integer** as the address which is divided by the hardware into a **page number and Offset**. On the other hands, in segmentation the user specifies the address in two quantities i.e. **segment number and offset**.
4. The size of the page is decided or specified by the **hardware**. On the other hands, the size of the segment is specified by the **user**.
5. In paging, the **page table** maps the **logical address to the physical address**, and it contains base address of each page stored in the frames of physical memory space. However, in segmentation, the **segment table** maps the **logical address to the physical address**, and it contains segment number and offset (segment limit).

Conclusion:

Paging and segmentation both are the **memory management schemes**. Paging allows the memory to be divided into **fixed sized block** whereas the segmentation, divides the memory space into **segments of the variable block size**. Where the paging leads to **internal fragmentation** the segmentation leads to **external fragmentation**.

Difference Between Logical and Physical Address in Operating System

Address uniquely identifies a location in the memory. We have two types of addresses that are logical address and physical address. The logical address is a virtual address and can be viewed by the user. The user can’t view the physical address directly. The logical address is used like a reference, to access the physical address. The fundamental difference between logical and physical address is that **logical address** is generated by CPU during a program execution whereas, the **physical address** refers to a location in the memory unit.

Comparison Chart

Basis for Comparison	Logical Address	Physical Address
Basic	It is the virtual address generated by CPU	The physical address is a location in a memory unit.
Address Space	Set of all logical addresses generated by CPU in reference to a program is referred as Logical Address Space.	Set of all physical addresses mapped to the corresponding logical addresses is referred as Physical Address.
Visibility	The user can view the logical address of a program.	The user can never view physical address of program
Access	The user uses the logical address to access the physical address.	The user can not directly access physical address.

Basis for Comparison	Logical Address	Physical Address
Generation	The Logical Address is generated by the CPU	Physical Address is Computed by MMU

Definition of Logical Address

Address generated by **CPU** while a program is running is referred as **Logical Address**. The logical address is virtual as it does not exist physically. Hence, it is also called as **Virtual Address**. This address is used as a reference to access the physical memory location. The set of all logical addresses generated by a programs perspective is called **Logical Address Space**.

The logical address is mapped to its corresponding physical address by a hardware device called **Memory-Management Unit**. The address-binding methods used by MMU generates **identical** logical and physical address during **compile time** and **load time**. However, while **run-time** the address-binding methods generate **different** logical and physical address.

Definition of Physical Address

Physical Address identifies a physical location in a memory. MMU (**Memory-Management Unit)** computes the physical address for the corresponding logical address. MMU also uses logical address computing physical address. The user never deals with the physical address. Instead, the physical address is accessed by its corresponding logical address by the user. The user program generates the logical address and thinks that the program is running in this logical address. But the program needs physical memory for its execution. Hence, the logical address must be mapped to the physical address before they are used.

The logical address is mapped to the physical address using a hardware called **Memory-Management Unit**. The set of all physical addresses corresponding to the logical addresses in a Logical address space is called **Physical Address Space**.

Key Differences Between Logical and Physical Address in OS

1. The basic difference between Logical and physical address is that Logical address is generated by CPU in perspective of a program. On the other hand, the physical address is a location that exists in the memory unit.
2. The set of all logical addresses generated by CPU for a program is called Logical Address Space. However, the set of all physical address mapped to corresponding logical addresses is referred as Physical Address Space.
3. The logical address is also called virtual address as the logical address does not exist physically in the memory unit. The physical address is a location in the memory unit that can be accessed physically.
4. Identical logical address and physical address are generated by Compile-time and Load time address binding methods.
5. The logical and physical address generated while run-time address binding method differs from each other.
6. The logical address is generated by the CPU while program is running whereas, the physical addres is computed by the MMU (Memory Management Unit).

Conclusion:

The logical address is a reference used to access physical address. The user can access physical address in the memory unit using this logical address.

Difference Between Contiguous and Noncontiguous Memory Allocation

Memory is a large array of bytes, where each byte has its own address. The memory allocation can be classified into two methods contiguous memory allocation and non-contiguous memory allocation. The major difference between Contiguous and Noncontiguous memory allocation is that the **contiguous memory allocation** assigns the consecutive blocks of memory to a process requesting for memory whereas, the **noncontiguous memory allocation** assigns the separate memory blocks at the different location in memory space in a nonconsecutive manner to a process requesting for memory. We will discuss some more differences between contiguous and non-contiguous memory allocation with the help of comparison chart shown below.

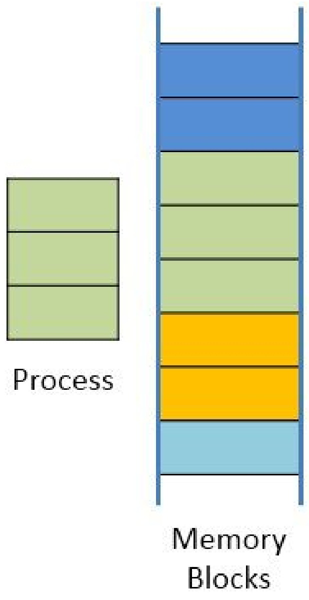
Comparison Chart

Basis the Comparison	Contiguous Memory Allocation	Noncontiguous Memory Allocation
Basic	Allocates consecutive blocks of memory to a process.	Allocates separate blocks of memory to a process.
Overheads	Contiguous memory allocation does not have the overhead of address translation while execution of a process.	Noncontiguous memory allocation has overhead of address translation while execution of a process.
Execution rate	A process executes fatser in contiguous memory allocation	A process executes quite slower comparatively in noncontiguous memory allocation.
Solution	The memory space must be divided into the fixed-sized partition and each partition is allocated to a single process only.	Divide the process into several blocks and place them in different parts of the memory according to the availability of memory space available.
Table	A table is maintained by operating system which maintains the list of available and occupied partition in the memory space	A table has to be maintained for each process that carries the base addresses of each block which has been acquired by a process in memory.

Definition of Contiguous Memory Allocation

The operating system and the user's processes both must be accommodated in the main memory. Hence the main memory is **divided into two** partitions: at one partition the operating system resides and at other the user processes reside. In usual conditions, the several user processes must reside in the memory at the same time, and therefore, it is important to consider the allocation of memory to the processes.

The Contiguous memory allocation is one of the methods of memory allocation. In contiguous memory allocation, when a process requests for the memory, a **single contiguous section of memory blocks** is assigned to the process according to its requirement.



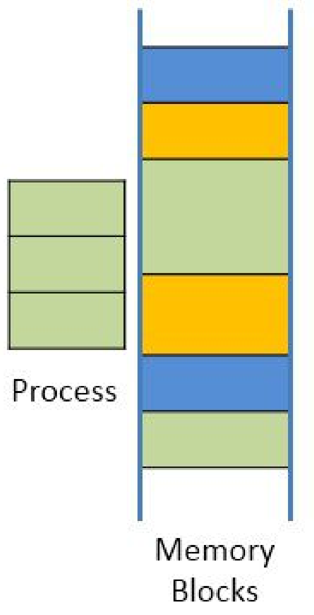
Contiguous Memory Allocation

The contiguous memory allocation can be achieved by dividing the memory into the fixed-sized **partition** and allocate each partition to a single process only. But this will cause the degree of multiprogramming, bounding to the number of fixed partition done in the memory. The contiguous memory allocation also leads to the **internal fragmentation**. Like, if a fixed sized memory block allocated to a process is slightly larger than its requirement then the left over memory space in the block is called internal fragmentation. When the process residing in the partition terminates the partition becomes available for the another process.

In the variable partitioning scheme, the operating system maintains a **table** which indicates, which partition of the memory is free and which occupied by the processes. The contiguous memory allocation fastens the execution of a process by reducing the overheads of address translation.

Definition Non-Contiguous Memory Allocation

The Non-contiguous memory allocation allows a process to **acquire the several memory blocks at the different location in the memory** according to its requirement. The noncontiguous memory allocation also **reduces** the **memory wastage** caused due to internal and external fragmentation. As it utilizes the memory holes, created during internal and external fragmentation.



Noncontiguous Memory Allocation

Paging and segmentation are the two ways which allow a process's physical address space to be non-contiguous. In non-contiguous memory allocation, the process is divided into **blocks** (pages or segments) which are placed into the different area of memory space according to the availability of the memory.

The noncontiguous memory allocation has an advantage of reducing memory wastage but, but it **increases** the **overheads** of address translation. As the parts of the process are placed in a different location in memory, it **slows the execution** of the memory because time is consumed in address translation.

Here, the operating system needs to maintain the **table** for each **process** which contains the base address of the each block which is acquired by the process in memory space.

Key Differences Between Contiguous and Noncontiguous Memory Allocation

1. The basic difference between contiguous and noncontiguous memory allocation is that contiguous allocation allocates **one single contiguous block of memory** to the process whereas, the noncontiguous allocation divides the **process into several blocks** and place them in the **different address space of the memory** i.e. in a noncontiguous manner.

2. In contiguous memory allocation, the process is stored in contiguous memory space; so there is **no overhead of address translation** during execution. But in noncontiguous memory allocation, there is **an overhead of address translation** while the process execution, as the process blocks are spread in the memory space.
3. Process stored in contiguous memory executes **faster** in comparison to process stored in noncontiguous memory space.
4. The solution for contiguous memory allocation is to **divide the memory space into the fixed-sized partition** and allocate a partition to a single process only. On the other hands, in noncontigous memory allocation, **a process is divided into several blocks** and each block is placed at **different places in memory** according to the availability of the memory.
5. In contiguous memory allocation, operating system has to maintain a **table** which indicates which partition is available for the process and which is occupied by the process. In noncontiguous memory allocation, a **table** is maintained for **each process** which indicates the base address of each block of the process placed in the memory space.

Conclusion:

Contiguous memory allocation does not create any overheads and fastens the execution speed of the process but **increases memory wastage**. In turn noncontiguous memory allocation creates overheads of address translation, reduces execution speed of a process but, **increases memory utilization**. So there are pros and cons of both allocation methods.

Difference Between Logical and Physical Address in Operating System

Address uniquely identifies a location in the memory. We have two types of addresses that are logical address and physical address. The logical address is a virtual address and can be viewed by the user. The user can't view the physical address directly. The logical address is used like a reference, to access the physical address. The fundamental difference between logical and physical address is that **logical address** is generated by CPU during a program execution whereas, the **physical address** refers to a location in the memory unit.

Comparison Chart

Basis for Comparison	Logical Address	Physical Address
Basic	It is the virtual address generated by CPU	The physical address is a location in a memory unit.
Address Space	Set of all logical addresses generated by CPU in reference to a program is referred as Logical Address Space.	Set of all physical addresses mapped to the corresponding logical addresses is referred as Physical Address.
Visibility	The user can view the logical address	The user can never view physical address

Basis for Comparison	Logical Address	Physical Address
	of a program.	of program
Access	The user uses the logical address to access the physical address.	The user can not directly access physical address.
Generation	The Logical Address is generated by the CPU	Physical Address is Computed by MMU

Definition of Logical Address

Address generated by **CPU** while a program is running is referred as **Logical Address**. The logical address is virtual as it does not exist physically. Hence, it is also called as **Virtual Address**. This address is used as a reference to access the physical memory location. The set of all logical addresses generated by a programs perspective is called **Logical Address Space**.

The logical address is mapped to its corresponding physical address by a hardware device called **Memory-Management Unit**. The address-binding methods used by MMU generates **identical** logical and physical address during **compile time** and **load time**. However, while **run-time** the address-binding methods generate **different** logical and physical address.

Definition of Physical Address

Physical Address identifies a physical location in a memory. MMU (**Memory-Management Unit)** computes the physical address for the corresponding logical address. MMU also uses logical address computing physical address. The user never deals with the physical address. Instead, the physical address is accessed by its corresponding logical address by the user. The user program generates the logical address and thinks that the program is running in this logical address. But the program needs physical memory for its execution. Hence, the logical address must be mapped to the physical address before they are used.

The logical address is mapped to the physical address using a hardware called **Memory-Management Unit**. The set of all physical addresses corresponding to the logical addresses in a Logical address space is called **Physical Address Space**.

Key Differences Between Logical and Physical Address in OS

1. The basic difference between Logical and physical address is that Logical address is generated by CPU in perspective of a program. On the other hand, the physical address is a location that exists in the memory unit.
2. The set of all logical addresses generated by CPU for a program is called Logical Address Space. However, the set of all physical address mapped to corresponding logical addresses is referred as Physical Address Space.
3. The logical address is also called virtual address as the logical address does not exist physically in the memory unit. The physical address is a location in the memory unit that can be accessed physically.

4. Identical logical address and physical address are generated by Compile-time and Load time address binding methods.
5. The logical and physical address generated while run-time address binding method differs from each other.
6. The logical address is generated by the CPU while program is running whereas, the physical addres is computed by the MMU (Memory Management Unit).

Conclusion:

The logical address is a reference used to access physical address. The user can access physical address in the memory unit using this logical address.

Difference Between Paging and Swapping in OS

Paging and Swapping are two **memory management strategies.** For execution, each process is required to be placed in main memory. Swapping and Paging both places the process in main memory for execution. **Swapping** could be added to any CPU scheduling algorithm where processes are swapped from main memory to back store and swapped backed to main memory. **Paging** allows the physical address space of a process to be **noncontiguous**. Let us discuss the differences between paging and swapping with the help of comparison chart shown below.

Comparison Chart

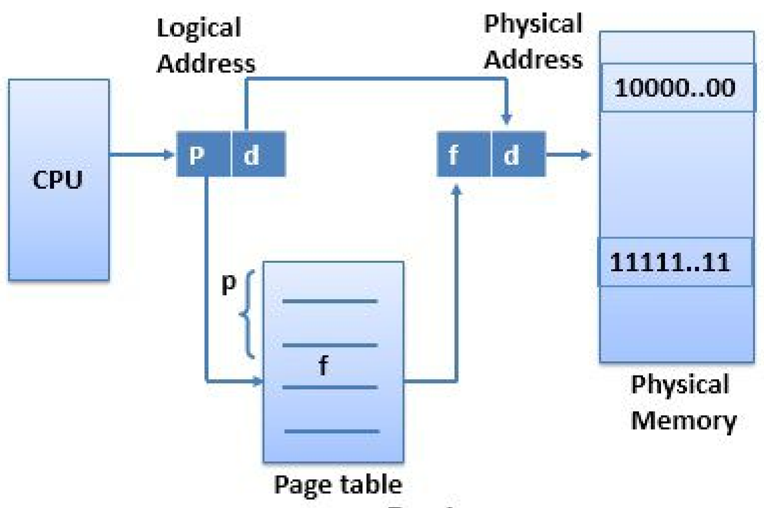
Basis of Comparison	Paging	Swapping
Basic	Paging allows the memory address space of a process to be noncontiguous.	Swapping allows multiple programs to run parallelly in the operating system.
Flexibility	Paging is more flexible as only pages of a process are moved.	Swapping is less flexible as it moves entire process back and forth between main memory and back store.
Multiprogramming	Paging allows more processes to reside in main memory	Compared to paging swapping allows less processes to reside in main memory.

Definition of Paging

Paging is a memory management scheme, which allots a **noncontiguous address space** to a process. Now, when a process's physical address can be non-contiguous the problem of **external fragmentation** would not arise.

Paging is implemented by breaking the **main memory** into fixed-sized blocks that are called **frames**. The **logical memory of a process** is broken into the same fixed-sized blocks called **pages**. The page size and frame size is defined by the hardware. As we know, the process is to be placed in main memory for execution. So, when a process is to be executed, the pages of the process from the source i.e. back store are loaded into any available frames in main memory.

Now let us discuss how paging is implemented. CPU generates the logical address for a process which consists of two parts that are **page number** and the **page offset**. The page number is used as an **index** in the **page table**.



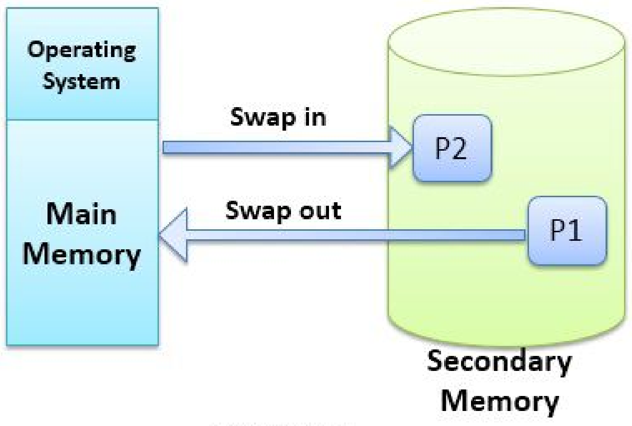
Paging

The page table contains the **base address** of each page that loaded in main memory. This base address is combined with page offset to generate the address of the page in main memory.

Every operating system has its own way of storing page table. Most of the operating system has a separate page table for each process.

Definition of Swapping

For execution, each process must be placed in the main memory. When we need to execute a process, and the main memory is entirely full, then the **memory manager swaps** a process from main memory to backing store by evacuating the place for the other processes to execute. The memory manager swaps the processes so frequently that there is always a process in main memory ready for execution.



Swapping

Due to **address binding** methods, the process that is swapped out of main memory occupies same address space when it is swapped back to the main memory if the binding is done at the assembly or load time. If the binding is done at execution time, the process can occupy any available address space in main memory as addresses are computed at the execution time.

Although the performance is affected by swapping, it helps in running **multiple processes in parallel**.

Key Differences Between Paging and Swapping in OS

1. The basic difference between paging and swapping is that paging avoids **external fragmentation** by allowing the physical address space of a process to be noncontiguous whereas, swapping allows **multiprogramming**.
2. Paging would transfer pages of a process back and forth between main memory, and secondary memory hence paging is flexible. However swapping swaps entire process back and forth between the main and secondary memory and hence swapping is less flexible.
3. Paging can allow more processes to be in main memory than the swapping.

Conclusion:

Paging avoids external fragmentation as utilizes the non-contiguous address spaces in the main memory. Swapping could be added to the CPU scheduling algorithm where process frequently needs to be in and out of main memory.

Difference Between Register and Memory

Register and memory, hold the data that can be **directly** accessed by the **processor** which also increases the processing speed of CPU. The processing speed of CPU can also be increased by increasing the number of bits of the register or increasing the number of the physical register in the CPU. Same is the case with memory, more the amount of memory faster is the CPU. Memory is genereally referred to the primary memory of the computer.

Despite these similarities, the register and memory share few differences with each other. The basic difference between the register and memory is that the **register** holds the data that CPU is currently processing whereas, the **memory** holds program instruction and data that the program requires for execution.

Comparison Chart

Basis for Comparison	Register	Memory
Basic	Registers hold the operands or instruction that CPU is currently processing.	Memory holds the instructions and the data that the currently executing program in CPU requires.
Capacity	Register holds the small amount of data around 32-bits to 64-bits.	Memory of the computer can range from some GB to TB.
Access	CPU can operate on register contents at the rate of more than one operation in one clock cycle.	CPU accesses memory at the slower rate than register.
Type	Accumulator register, Program counter, Instruction register, Address register, etc.	RAM.

Definition of Register

Registers are the **smallest** data holding elements that are **built into** the processor itself. Registers are the memory locations that are **directly** accessible by the processor. The registers hold the instruction or operands that is currently being accessed by the CPU.

Registers are the **high-speed** accessible storage elements. The processor accesses the registers within **one CPU clock cycle**. In fact, the processor can decode the instructions and perform operations on the register contents at the **rate of more than one operation per CPU clock cycle.** So we can say that processor can access registers faster than the main memory.

The register is measured in bits like a processor may have 16-bit, 32-bit, or 64-bit registers. The number of register bits specifies the speed and power of CPU. For example, a CPU which has 32-bit register can access the 32-bit instructions at a time. The CPU which has 64-bit register can execute 64-bit instructions. Hence, more the number of bits of register more is the speed and power of CPU.

The computer registers are categorized as follow:

DR: Data Register is a 16-bit register which holds the **operands** to be operated by the processor.

AR: Address Register is a 12-bit register that holds the **address of a memory location**.

AC: Accumulator is also a 16-bit register which holds the **result computed** by the processor.

IR: Instruction Register is a 16-bit register that holds the **instruction code** that has to currently executed.

PC: Program Counter is a 12-bit register that holds the **address of instruction** that is to be executed by the processor.

TR: Temporary Register is a 16-bit register that holds the **temporary intermediate result** computed by the processor.

INPR: Input Register is an 8-bit register that holds the **input character** received from an **input device** and delivered it to the **Accumulator**.

OUTR: Output Register is an 8-bit register that holds the **output character** received from **Accumulator** and deliver it to the **output device**.

Definition of Memory

Memory is a hardware device used to store computer programs, instructions and data. The memory that is internal to the processor is a **primary memory (RAM)**, and the memory that is external to the processor is a **secondary memory (Hard Drive)**. Memory can also be categorized on the basis of **volatile** and **non-volatile** memory.

Basically, the **computer memory** refers to the **primary memory** of the computer whereas, the **secondary memory** is referred as **storage** of the computer. Primary memory is the memory that can be **directly** accessed by the processor due to which there is no delay in accessing data, and thus the processor computes faster.

Primary memory or RAM is a **volatile** memory which means the data in the primary memory exist when the systems power is on, and the data vanishes as the system is switched off. The primary memory contains the data that will be required by the currently executing program in CPU. If the data required by the processor is not in primary memory, then the data is transferred from secondary storage to primary memory, and then it is fetched by the processor.

Once you **save** the data on the computer, then it is transferred to **secondary storage** till then it remains in the primary memory. Today the primary memory or RAM can range from **1 GB to 16 GB**. On the other hands, the secondary storage today ranges from some **Giga Bytes (GB) to TeraBytes (TB)**.

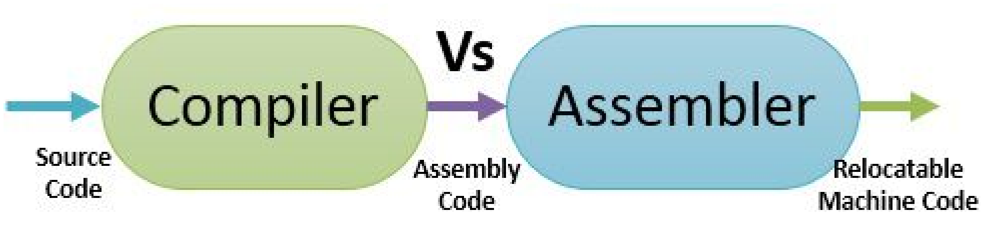
Key Differences Between Register and Memory

1. The primary difference between register and memory is that register **holds the data that the CPU is currently processing** whereas, the memory **holds the data the that will be required for processing**.
2. The Register ranges from **32-bits register to 64-bits register** whereas, the memory capacity ranges from some **GB** to some **TB**.
3. The processor accesses register **faster** than the memory.
4. Computers registers are **accumulator register, program counter, instruction register, address register**, etc. On the other hands, memory is referred as the main memory of the computer which is RAM.

Conclusion:

Normally the register resides at the top of the memory hierarchy. It is the smallest and fastly accessible storage element. On the other hands, the memory generally referred to as the main memory which is larger than register and its CPU access is slower than register but it is accessed faster than the secondary storage.

Difference Between Compiler and Assembler



Compiler and Assembler plays an important role in the execution of a program. Some compilers directly generate the executable code instead of assembly code. The compiler takes the preprocessed source code and translates it into the assembly code. The assembler takes the assembly code from the compiler and translates it to the relocatable machine code. In this article, I have discussed the differences between compiler and assembler with the help of comparison chart shown below, just have a look.

Comparison Chart

Basis for Comparison	Compiler	Assembler
Basic	Generates the assembly language code or directly the executable code.	Generates the relocatable machine code.
Input	Preprocessed source code.	Assembly language code.
Phases/ Passes	The compilation phases are lexical analyzer, syntax analyzer, semantic analyzer, intermediate code generation, code optimization, code generation.	Assembler makes two passes over the given input.
Output	The assembly code generated by the compiler is a mnemonic version of machine code.	The relocatable machine code generated by an assembler is represented by binary code.

Definition of Compiler

The **compiler** is a computer program that reads the program written in a source language, translates it into equivalent **assembly language** and forwards the assembly language code to the **Assembler**.

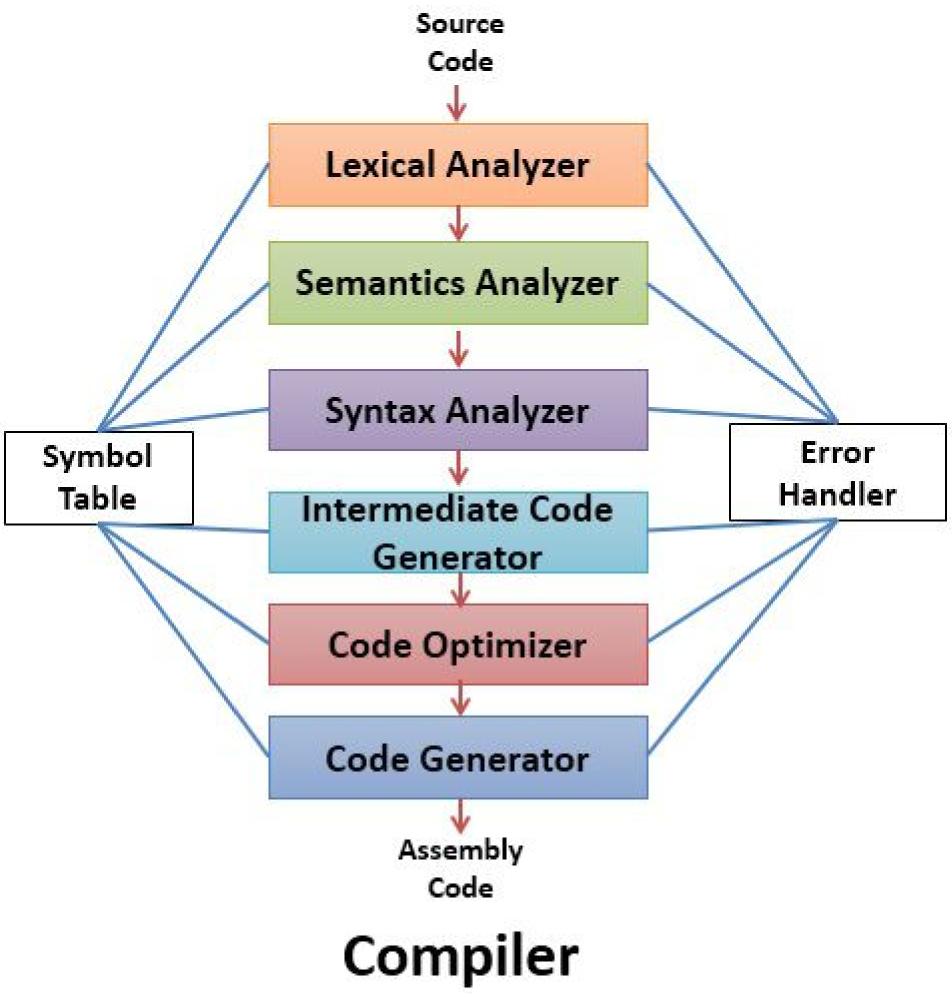
While the translation of the source code to assembly code the compiler also reports the **error** in the source code to its user.

Compilers are also classified as **single-pass, multi-pass, load-and-go, debugging and optimization.** Classification is done on the basis of what function does a compiler perform and how it has been constructed. Despite these complexities, the basic task of compiler remains same.

The compilation is performed in two parts, **analysis part** and **synthesis part**. The **analysis part** breaks the source code into constituent pieces and forms the intermediate representation of the source code. The **synthesis part** forms the target code from the intermediate representation.

The compilation is performed in the following phases:

Lexical analyzer, syntax analyzer, semantic analyzer, intermediate code generator, code optimizer, code generator, Symbol table and error Handler.



The **lexical analyzer** reads the characters of source code and groups them into the **streams of tokens**. Each token represents the logical sequence of characters like **keyword, identifiers, operators**. The sequence of character that forms a token is called **lexeme**.

- The **syntax analyzer** parses the token obtained from lexical analyzer and groups tokens into a **hierarchical structure**.
- The **semantic analyzer** checks the source code for any **semantic error**.
- **Intermediate code generator** generates the **intermediate representation** of the source code
- The **code Optimizer** optimizes the intermediate code into faster running machine code.
- The **code generator** finally generates the target code which is a **relocatable machine code or an assembly code**.
- The **symbol table** is a data structure which contains the record for each identifier in the source code.
- **Error handler** detects the error in each phase and handles those errors.

Definition of Assembler

Some compilers perform the task of assembler and directly generate a relocatable machine code instead of assembly code, which is further directly passed to linker /loader. The **assembler** takes as input the assembly code generated by the compiler and translates it into **relocatable machine code**.

Assembly Code
Assembler
Relocatable Machine Code

Let us see how machine code is different from assembly code. **Assembly code** is the **mnemonic** version of machine code. It means it assembly code uses names for representing operations and it even gives names to the memory addresses. On the other hands, the **machine code** uses **binary codes** for representation of operations and memory addresses.

Even the simplest form of assembler performs **two passes** over the input. The **first pass** detects all the **identifiers** in the assembly code that denotes storage location and store them in the **symbol table** (other than compilers symbol table). The **storage location is assigned** to the identifier that is encountered in the first pass.

In the **second pass**, the input is scanned again, and this time the **operation code** are **translated** into a **sequence of bits** representing that operation in the machine code. The second pass also translates **identifiers** into the **addresses** defined in the symbol table. Thus the second pass generates the **relocatable machine code**.

Key Differences Between Compiler and Assembler

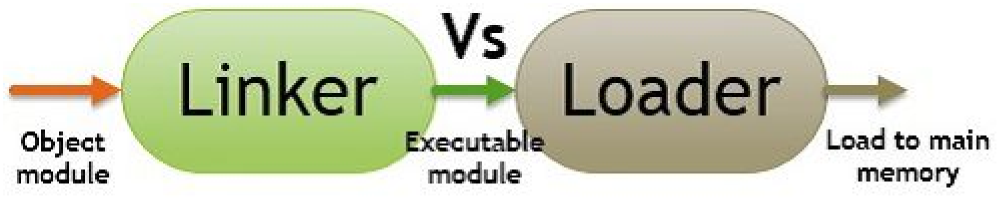
1. The key difference between compiler and assembler is that the **compiler** generates assembly code and some compilers can also directly generate executable code whereas, the **assembler** generates relocatable machine code.

2. The compiler takes as input the **preprocessed code** generated by preprocessor. On the other hands, the assembler takes **assembly code** as input.
3. The compilation takes place in two phases that are **analysis phase** and **synthesis phase**. In analysis phase, the input goes through **lexical analyzer, syntax analyzer, semantic analyzer** whereas, the synthesis analysis takes place via **intermediate code generator, code optimizer, code generator**. On the other hands, assembler passes the input through **two phases**. The first phase detects the identifiers and allots addresses to them in the second phase the assembly code is translated to binary code.
4. The assembly code generated by the compiler is a **mnemonic version** of machine code. However, the relocatable machine code generated by assembler is a **binary relocatable code**.

Conclusion:

The assembler may not be required as some compilers directly generate executable code. If assembler is used it requires linker to link all the built-in libraries to the library functions used in the source code.

Difference Between Linker and Loader



Linker and Loader are the utility programs that plays a major role in the execution of a program. The Source code of a program passes through compiler, assembler, linker, loader in the respective order, before execution. On the one hand, where the **linker** intakes the object codes generated by the assembler and combine them to generate the executable module. On the other hands, the **loader** loads this executable module to the main memory for execution. Let us discuss the difference between Linker and loader with the help of a comparison chart.

Comparison Chart

Basis for Comparison	Linker	Loader
Basic	It generates the executable module of a source program.	It loads the executable module to the main memory.
Input	It takes as input, the object code generated by an assembler.	It takes executable module generated by a linker.

Basis for Comparison	Linker	Loader
Function	It combines all the object modules of a source code to generate an executable module.	It allocates the addresses to an executable module in main memory for execution.
Type/Approach	Linkage Editor, Dynamic linker.	Absolute loading, Relocatable loading and Dynamic Run-time loading.

Definition of Linker

The Assembler generates the object code of a source program and hands it over to the linker. The linker takes this object code and generates the **executable code** for the program, and hand it over to the Loader.

The high-level language, programs have some **built-in libraries** and **header files**. The source program may contain some library functions whose definition are stored in the built-in libraries. The linker links these function to the built-in libraries. In case the built-in libraries are not found it informs to the compiler, and the compiler then generates the error.

Sometimes the large programs are divided into the subprograms which are called **modules**. Now when these modules are compiled and assembled, the object modules of the source program are generated. The linker has the responsibility of combining/linking all the object modules to generate a single executable file of the source program. We have two types of linkers.

Linkage Editor: It is a linker that generates the relocatable, executable module.

Dynamic Linker: It defers/postpones the linkage of some external modules until the load module/executable module is generated. Here, linking is done during load time or run time.

Definition of Loader

As the program that has to be executed currently must reside in the main memory of the computer. It is the responsibility of the **loader,** a program in an operating system, to load the executable file/module of a program, generated by the linker, to the main memory for execution. It

allocates the memory space to the executable module in main memory.

There are three kinds of loading approaches:

- Absolute loading
- Relocatable loading
- Dynamic run-time loading

Absolute loading: This approach loads the executable file of a program into a **same main memory location** each time. But it has some **disadvantages** like a programmer must be aware of the assignment strategy for loading the modules to main memory. In case, the program is to be modified involving some insertion and deletion in the program, then all the addresses of the program have to be altered.

Relocatable loading: In this approach, the compiler or assembler does **not produce actual main memory address**. It produces the relative addresses.

Dynamic Run-Time loading: In this approach, the absolute address for a program is generated when an instruction of an executable module is actually executed. It is very flexible, the loadable module/executable module can be loaded into **any region of main memory**. The executing program can be interrupted in between and can be swapped out to the disk and back to main

memory this time at a different main memory address.

Key Differences Between Linker and Loader

1. The key difference between linker and loader is that the linker generates the **executable** file of a program whereas, the loader loads the executable file obtained from the linker into **main memory for execution**.
2. The linker intakes the **object module** of a program generated by the assembler. However, the loader intakes the **executable module** generated by the linker.
3. The linker combines all object module of a program to generate **executable modules** it also links the **library function** in the object module to **built-in libraries** of the high-level programming language. On the other hands, loader **allocates space to an executable** module in main memory.
4. The linker can be classified as **linkage editor,** and **dynamic linker** whereas loader can be classified as **absolute loader, relocatable loader** and **dynamic run-time loader**.

Conclusion:

The linker takes the object modules of a program from the assembler and links them together to generate an executable module of a program. The executable module is then loaded by the loader into the main memory for execution.

Difference Between Compiler and Interpreter

A compiler is a translator which transforms source language (high-level language) into object language (machine language). In contrast with a compiler, an interpreter is a program which imitates the execution of programs written in a source language. Another difference between Compiler and interpreter is that Compiler converts the whole program in one go on the other hand Interpreter converts the program by taking a single line at a time..

Evidently, the perceivability of humans and an electronic device like a computer is different. Humans can understand anything through the natural languages, but a computer doesn’t. The computer needs a translator to convert the languages written in the human readable form to the computer readable form.

Compiler and interpreter are the types of language translator. What is Language translator? This question might be arising in your mind.

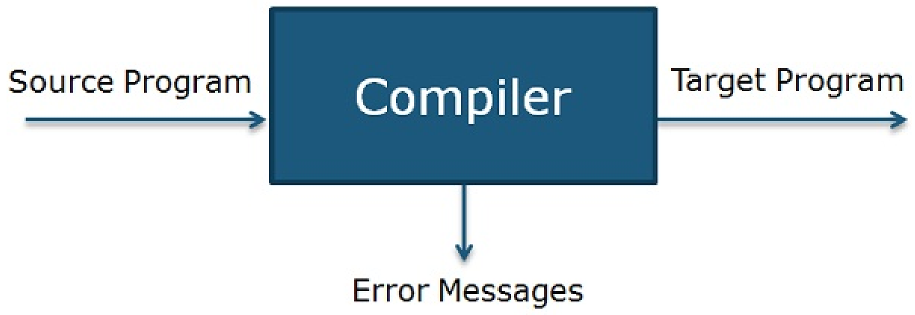
A language translator is a software which translates the programs from a source language that are in human readable form into an equivalent program in an object language. The source language is usually a high-level programming language, and the object language is typically the machine language of an actual computer.

Comparison Chart

Basis for comparison	Compiler	Interpreter
Input	It takes an entire program at a time.	It takes a single line of code or instruction at a time.
Output	It generates intermediate object code.	It does not generate any intermediate object code.
Working mechanism	The compilation is done before execution.	Compilation and execution take place simultaneously.
Speed	Comparatively faster	Slower
Memory	Memory requirement is more due to the creation of object code.	It requires less memory as it does not create intermediate object code.
Errors	Display all errors after compilation, all at the same time.	Displays error of each line one by one.
Error detection	Difficult	Easier comparatively
Pertaining Programming languages	C, C++, C#, Scala, typescript uses compiler.	Java, PHP, Perl, Python, Ruby uses an interpreter.

Definition of Compiler

A compiler is a program that reads a program written in the high-level language and converts it into the machine or low-level language and reports the errors present in the program. It converts the entire source code in one go or could take multiple passes to do so, but at last, the user gets the compiled code which is ready to execute.



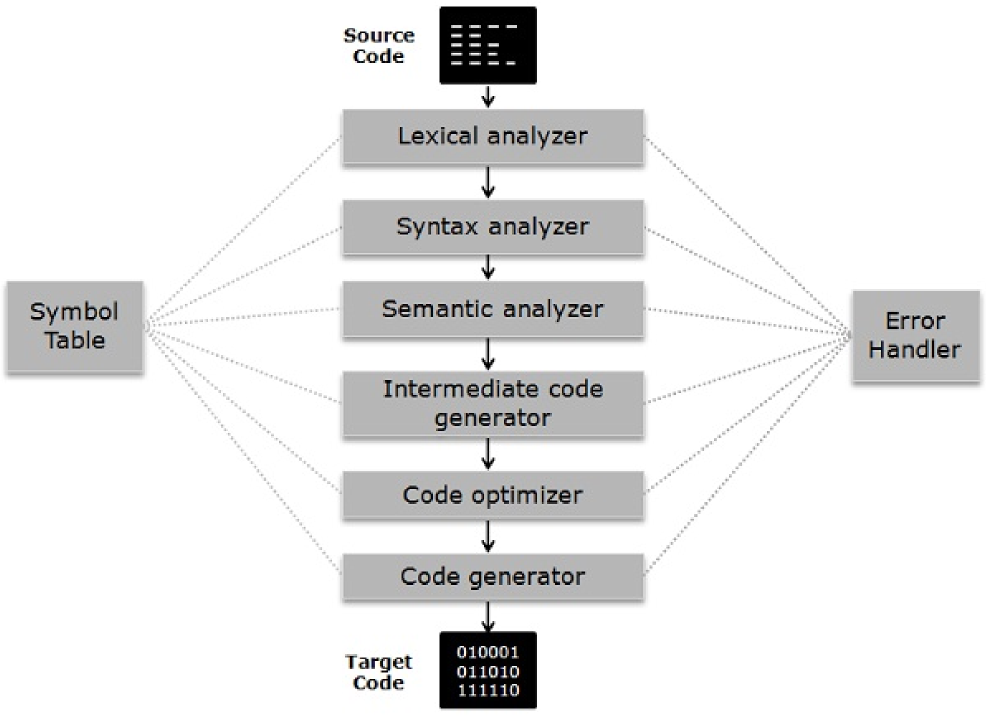
Compiler operates on phases; various stages can be grouped into two parts that are:

- **Analysis Phase** of the compiler is also known as the front end in which program is divided into fundamental constituent parts and checks grammar, semantic and syntax of the code after which intermediate code is generated. Analysis phase includes lexical analyzer, syntax analyzer and semantic analyzer.
- **Synthesis phase** of the compiler is also known as the back end in which intermediate code is optimized, and target code is generated. Synthesis phase includes code optimizer and code generator.

PHASES OF COMPILER

Now let's understand the working of each stage in detail.

1. **Lexical Analyzer**: It scans the code as a stream of characters, groups the sequence of characters into lexemes and outputs a sequence of tokens with reference to the programming language.
2. **Syntax Analyzer**: In this phase, the tokens that are generated in the previous stage are checked against the grammar of programming language, whether the expressions are syntactically correct or not. It makes parse trees for doing so.
3. **Semantic Analyzer**: It checks whether the expressions and statements generated in the previous phase follow the rule of programming language or not and it creates annotated parse trees.
4. **Intermediate code generator**: It generates equivalent intermediate code of the source code. There are many representations of intermediate code, but TAC (Three Address Code) is the used most widely.
5. C**ode Optimizer**: It improves time and space requirement of the program. For doing so, it eliminates the redundant code present in the program.
6. **Code generator**: This is the final phase of the compiler in which target code for a particular machine is generated. It performs operations like memory management, Register assignment, and machine specific optimization.



The **symbol table** is somewhat a data structure which manages the identifiers along with the relevant type of data it is storing. **Error Handler** detect, report, correct the errors encountering in between the different phases of a compiler.

Definition of Interpreter

The interpreter is a different way of implementing a programming language and does the same work as a compiler. Interpreter performs **lexing**, **parsing** and **type checking** similar to a compiler. But interpreter processes syntax tree directly to evaluate expressions and execute statement instead of generating code from the syntax tree.

An interpreter may require processing same syntax tree more than once that is the reason why interpretation is comparatively slower than executing the compiled program.

Compilation and interpretation probably combined to implement a programming language. In which a compiler generates intermediate-level code then the code is interpreted rather than compiled to machine code.

Using an interpreter is very useful during program development, where the most important part is to be able to test a program modification rapidly rather than run the program efficiently.

Key Differences Between Compiler and Interpreter

Let’s look at major differences between Compiler and Interpreter.

1. The compiler takes a program as a whole and translates it, but interpreter translates a program statement by statement.
2. Intermediate code or target code is generated in case of a compiler. As against interpreter doesn’t create intermediate code.
3. A compiler is comparatively faster than Interpreter as the compiler take the whole program at one go whereas interpreters compile each line of code after the other.
4. The compiler requires more memory than interpreter because of the generation of object code.
5. Compiler presents all errors concurrently, and it’s difficult to detect the errors in contrast interpreter display errors of each statement one by one, and it’s easier to detect errors.
6. In compiler when an error occurs in the program, it stops its translation and after removing error whole program is translated again. On the contrary, when an error takes place in the interpreter, it prevents its translation and after removing the error, translation resumes.
7. In a compiler, the process requires two steps in which firstly source code is translated to target program then executed. While in Interpreter It’s a one step process in which Source code is compiled and executed at the same time.
8. The compiler is used in programming languages like C, C++, C#, Scala, etc. On the other Interpreter is employed in languages like Java, PHP, Ruby, Python, etc.

Conclusion

Compiler and interpreter both are intended to do the same work but differ in operating procedure, Compiler takes source code in an aggregated way whereas Interpreter takes constituent parts of source code, i.e., statement by statement.

Although both compiler and interpreter have certain advantages and disadvantages like Interpreted languages are considered as cross-platform, i.e., the code is portable. It also doesn't need to compile instruction previously unlike compiler which is time-saving. Compiled languages are faster regarding compilation process.

Difference Between Primary and Secondary Memory

Comparison Chart

Basis for Comparison	Primary Memory	Secondary Memory
Basic	Primary memory is directly accessible by Processor/CPU.	Secondary memory is not directly accessible by CPU.
Altered Name	Main memory.	Auxiliary memory.
Data	Instructions or data to be currently executed are copied to main memory.	Data to be permanently stored is kept in secondary memory.
Volatility	Primary memory is usually volatile.	Secondary memory is non-volatile.
Formation	Primary memories are made of semiconductors.	Secondary memories are made of magnetic and optical material.
Access Speed	Accessing data from primary memory is faster.	Accessing data from secondary memory is slower.
Access	Primary memory is accessed by the data bus.	Secondary memory is accessed by input-output channels.
Size	The computer has a small primary memory.	The computer has a larger secondary memory.
Expense	Primary memory is costlier than secondary memory.	Secondary memory is cheaper than primary memory
Memory	Primary memory is an internal memory.	Secondary memory is an external memory.

Key Differences Between Primary and Secondary Memory

1. The key difference between primary and secondary memory is that primary memory can be **directly accessed by the CPU** whereas, the **CPU can not directly access** the secondary memory.
2. The primary memory of the computer is also known as the **main memory** of the computer. However, secondary memory is known as **auxiliary memory**.

3. The data that is to be **currently processed** is in primary memory whereas, the data that has to be **permanently stored** is kept in secondary memory.
4. Primary memory is a **volatile** memory whereas, the secondary memory is a **non-volatile** memory.
5. Primary memories are **semiconductor memories** whereas; the secondary memories are the **magnetic and optical memories**.
6. Data accessing speed of the primary memory is **faster** than secondary memory.
7. Primary memory is accessed by the **data bus**. On the other hand, secondary memory is accessed using **input-output channels**.
8. Primary memory's capacity is always **smaller** than secondary memory's capacity.
9. Primary memory is **costlier** than secondary memory.
10. Primary memory is an **internal memory** whereas, secondary memory is an **external memory**.

Difference Between Loosely Coupled and Tightly Coupled Multiprocessor System

Comparison Chart

Basis for Comparison	Loosely Coupled Multiprocessor System	Tightly Coupled Multiprocessor System
Basic	Each processor has its own memory module.	Processors have shared memory modules.
Efficient	Efficient when tasks running on different processors, has minimal interaction.	Efficient for high-speed or real-time processing.
Memory conflict	It generally, do not encounter memory conflict.	It experiences more memory conflicts.
Interconnections	Message transfer system (MTS).	Interconnection networks PMIN, IOPIN, ISIN.
Data rate	Low.	High.
Expensive	Less expensive.	More expensive.

Key Differences Between Loosely Coupled and Tightly Coupled Multiprocessor System

1. The key difference between loosely coupled and tightly coupled system is that **loosely coupled system** has **distributed memory**, whereas, the **tightly coupled system** has **shared memory**.
2. Loosely coupled is **efficient** when the tasks running on different processors has **minimal interaction** between them. On the other hands, the tightly coupled system can take a **higher**

degree of interaction between processes and is efficient for **high-speed** and **real-time processing**.
3. The loosely coupled system generally do **not encounter memory conflict** which is mostly experienced by tightly couples system.
4. The interconnection network in a loosely coupled system is **Message Transfer system (MTS)** whereas, in a tightly coupled system the interconnection networks are **processor-memory interconnection network (PMIN), I/O-processor interconnection network (IOPIN)** and **the interrupt-signal interconnection network (ISIN)**.
5. The **data rate** of the loosely coupled system is **low** whereas, the **data rate** of the tightly coupled system is **high**.
6. The loosely coupled system is **less expensive** but **larger in size** whereas, the tightly coupled system is **more expensive** but **compact in size**.

Difference Between Multiprocessing and Multithreading

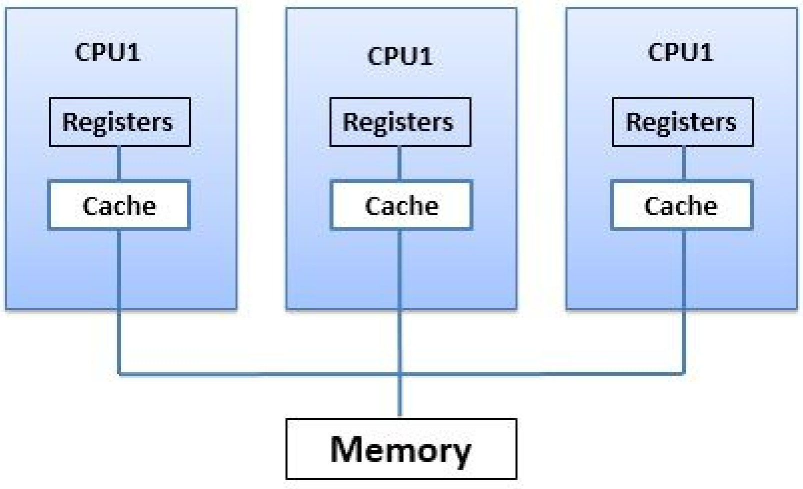
Multiprocessing and Multithreading both adds performance to the system. **Multiprocessing** is adding more number of or CPUs/processors to the system which increases the computing speed of the system. **Multithreading** is allowing a process to create more threads which increase the responsiveness of the system. I have figured some more differences between multiprocessing and multithreading which I have discussed with the help of comparison chart shown below.

Comparison Chart

Basis for Comparison	Multiprocessing	Multithreading
Basic	Multiprocessing adds CPUs to increase computing power.	Multithreading creates multiple threads of a single process to increase computing power.
Execution	Multiple processes are executed concurrently.	Multiple threads of a single process are executed concurrently.
Creation	Creation of a process is time-consuming and resource intensive.	Creation of a thread is economical in both sense time and resource.
Classification	Multiprocessing can be symmetric or asymmetric.	Multithreading is not classified.

Definition of Multiprocessing

A multiprocessing system is one which has more than two processors. The CPUs are added to the system to increase the computing speed of the system. Each CPU has its own set of registers and main memory. Just because CPUs are separate, it may happen that one CPU must not have anything to process and may sit idle and the other may be overloaded with the processes. In such cases, the processes and the resources are shared dynamically among the processors.



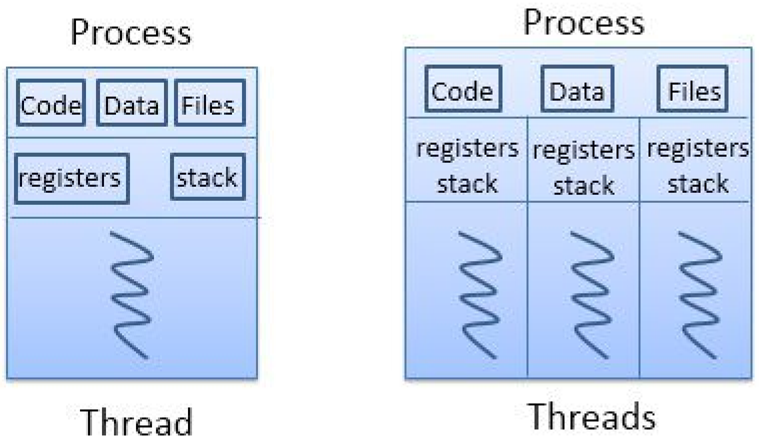
Multiprocessing

Multiprocessing can be classified as **symmetric multiprocessing** and **asymmetric multiprocessing**. In symmetric multiprocessing, all processors are free to run any process in a system. In Asymmetric multiprocessing, there is a master-slave relationship among the processors. The master processor is responsible for allotting the process to slave processors.

If the processor has **integrated memory controller** then adding processor would increase the amount of addressable memory in the system. Multiprocessing can change the memory access model from **uniform memory access** to **nonuniform memory access**. The uniform memory access amounts the same time for accessing any RAM from any Processor. On the other hands, non-uniform memory access amounts longer time to access some part of memory than the other parts.

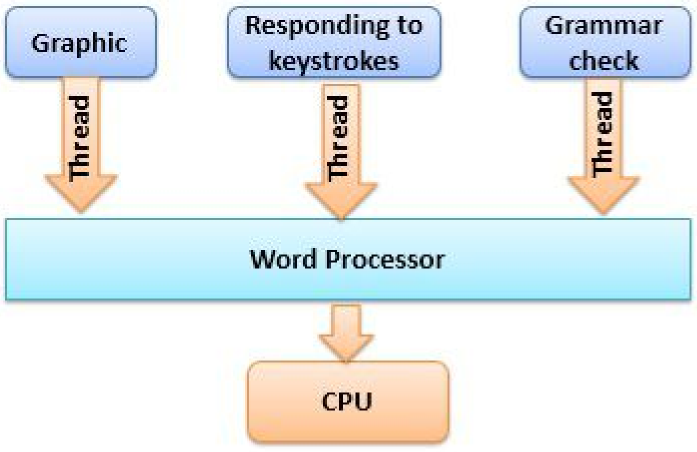
Definition of Multithreading

Multithreading is the execution of multiple threads of a single process concurrently within the context of that process. Now let us first discuss what is a thread? A **thread** of a process means a code segment of a process, which has its own thread ID, program counter, registers and stack and can execute independently. But threads belonging to the same process has to share the belongings of that process like code, data, and system resources. Creating separate processes for each service request consumes time and exhaust system resources. Instead of incurring this overhead, it is more efficient to create threads of a process.



Multithreading

To understand the multithreading concept let us take an **example** of a word processor. A word processor, displays graphic, responds to keystrokes, and at the same time, it continues spelling and grammar checking. You do not have to open different word processors to do this concurrently. It does get happen in a single word processor with the help of multiple threads.



Multithreading

Now let us take into consideration the benefits of multithreading. Multithreading increases the **responsiveness** as if one thread of a process is blocked or performing the lengthy operation, the process still continues. The second benefit of multithreading is **resource sharing** as several threads of a process share same code and data within the same address space.

Creating a thread is **economical** as it shares the code and data of the process to which they belong. So the system does not have to allocate resources separately for each thread. Multithreading can be **increased** on multiprocessing operating system. As multithreading on multiple CPUs increases **parallelism**.

Key Differences Between Multiprocessing and Multithreading

1. The key difference between multiprocessing and multithreading is that multiprocessing allows a system to have more than two CPUs added to the system whereas multithreading lets a process generate multiple threads to increase the computing speed of a system.
2. Multiprocessing system executes **multiple processes** simultaneously whereas, the multithreading system let execute **multiple threads** of a process simultaneously.
3. Creating a process can **consume time** and **even exhaust** the system resources. However creating threads is **economical** as threads belonging to the same process share the belongings of that process.
4. Multiprocessing can be classified into **symmetric multiprocessing** and **asymmetric multiprocessing** whereas, multithreading is not classified further.

Conclusion:

The benefits of multithreading can be gradually increased in multiprocessing environment as multithreading on a multiprocessing system increases parallelism

Difference Between Preemptive and Non-Preemptive Scheduling in OS

It is the responsibility of CPU scheduler to allot a process to CPU whenever the CPU is in the idle state. The CPU scheduler selects a process from ready queue and allocates the process to CPU. The scheduling which takes place when a process switches from running state to ready state or from waiting state to ready state is called **Preemptive Scheduling**. On the hands, the scheduling which takes place when a process terminates or switches from running to waiting for state this kind of CPU scheduling is called **Non-Preemptive Scheduling**. The basic difference between preemptive and non-preemptive scheduling lies in their name itself. That is a Preemptive scheduling can be preempted; the processes can be scheduled. In Non-preemptive scheduling, the processes can not be scheduled.

Comparison Chart

Basis for Comparison	**Preemptive Scheduling**	**Non Preemptive Scheduling**
Basic	The resources are allocated to a process for a limited time.	Once resources are allocated to a process, the process holds it till it completes its burst time or switches to waiting state.
Interrupt	Process can be interrupted in between.	Process can not be interrupted till it terminates or switches to waiting state.
Starvation	If a high priority process frequently arrives in the ready queue, low priority process may starve.	If a process with long burst time is running CPU, then another process with less CPU burst time may starve.
Overhead	Preemptive scheduling has overheads of scheduling the processes.	Non-preemptive scheduling does not have overheads.
Flexibility	Preemptive scheduling is flexible.	Non-preemptive scheduling is rigid.
Cost	Preemptive scheduling is cost associated.	Non-preemptive scheduling is not cost associative.

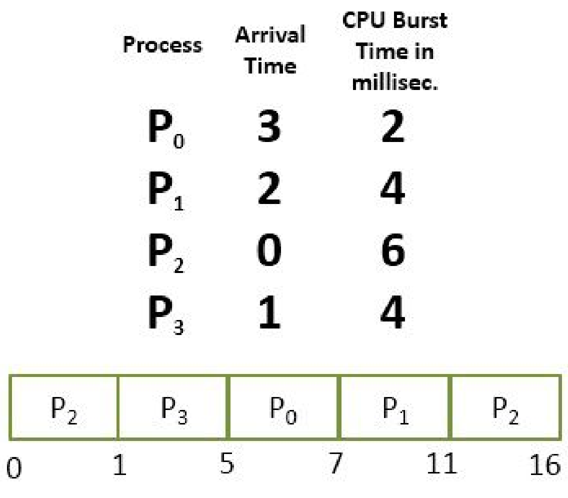
Definition of Preemptive Scheduling

Preemptive scheduling is one which can be done in the circumstances when a process switches from **running state** to **ready state** or from **waiting state** to **ready state**. Here, the resources (CPU cycles) are allocated to the process for the **limited** amount of time and then is taken away, and the process is placed back in the ready queue again if it still has CPU burst time remaining. The process stays in ready queue till it gets next chance to execute.

If a process with high priority arrives in the ready queue, it does not have to wait for the current process to complete its burst time. Instead, the current process is interrupted in the middle of execution and is placed in the ready queue till the process with high priority is utilizing the CPU cycles. In this way, each process in the ready queue gets some time to run CPU. It makes the preemptive scheduling flexible but, increases the overhead of switching the process from running state to ready state and vise-verse.

Algorithms that work on preemptive scheduling are Round Robin. Shortest Job First (SJF) and Priority scheduling may or may not come under preemptive scheduling.

Let us take an example of Preemptive Scheduling, look in the picture below. We have four processes P0, P1, P2, P3. Out of which, P2 arrives at time 0. So the CPU is allocated to the process P2 as there is no other process in the queue. Meanwhile, P2 was executing, P3 arrives at time 1, now the remaining time for process P2 (5 milliseconds) which is larger than the time required by P3 (4 milli-sec). So CPU is allocated to processor P3.



Process	Arrival Time	CPU Burst Time in millisec.
P_0	3	2
P_1	2	4
P_2	0	6
P_3	1	4

Preemptive Scheduling

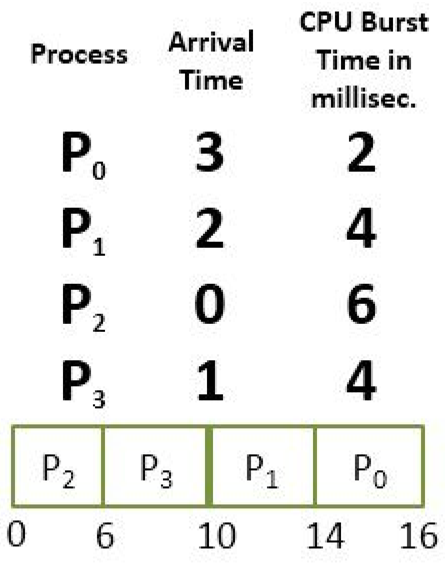
Meanwhile, P3 was executing, process P1 arrives at time 2. Now the remaining time for P3 (3 milliseconds) is less than the time required by processes P1 (4 milliseconds) and P2 (5 milliseconds). So P3 is allowed to continue. While P3 is continuing process P0 arrives at time 3, now the remaining time for P3 (2 milliseconds) is equal to the time require by P0 (2 milliseconds). So P3 continues and after P3 terminates the CPU is allocated to P0 as it has less burst time than other. After P0 terminates, the CPU is allocated to P1 and then to P2.

Definition of Non-Preemptive Scheduling

Non-preemptive Scheduling is one which can be applied in the circumstances when a process **terminates**, or a process switches from **running** to **ready state**. In Non-Preemptive Scheduling, once the resources (CPU) is allocated to a process, the process holds the CPU till it gets terminated or it reaches a waiting state.

Like preemptive scheduling, non-preemptive scheduling does not interrupt a process running CPU in middle of the execution. Instead, it waits for the process to complete its CPU burst time and then it can allocate the CPU to another process.

In Non-preemptive scheduling, if a process with long CPU burst time is executing then the other process will have to wait for a long time which increases the average waiting time of the processes in the ready queue. However, the non-preemptive scheduling does not have any overhead of switching the processes from ready queue to CPU but it makes the scheduling rigid as the process in execution is not even preempted for a process with higher priority.



Process	Arrival Time	CPU Burst Time in millisec.
P_0	3	2
P_1	2	4
P_2	0	6
P_3	1	4

Non-Preemptive Scheduling

Let us solve the above scheduling example in non-preemptive fashion. As initially the process P2 arrives at time 0, so CPU is allocated to the process P2 it takes 6 milliseconds to execute. In between all the processes i.e. P0, P1, P3 arrives into ready queue. But all waits till process P2 completes its CPU burst time. Then process that arrives after P2 i.e. P3 is then allocated the CPU till it finishes it’s burst time. Similarly, then P1 executes, and CPU is later given to process P0.

Key Differences Between Preemptive and Non-Preemptive Scheduling

1. The basic difference between preemptive and non-preemptive scheduling is that in preemptive scheduling the CPU is allocated to the processes for the **limited** time. While in Non-preemptive scheduling, the CPU is allocated to the process till it **terminates** or switches to **waiting state**.
2. The executing process in preemptive scheduling is **interrupted** in the middle of execution whereas, the executing process in non-preemptive scheduling is **not interrupted** in the middle of execution.
3. Preemptive Scheduling has the **overhead** of switching the process from ready state to running state, vise-verse, and maintaining the ready queue. On the other hands, non-preemptive scheduling has **no overhead** of switching the process from running state to ready state.
4. In preemptive scheduling, if a process with high priority frequently arrives in the ready queue then the process with low priority have to wait for a long, and it may have to starve. On the other hands, in the non-preemptive scheduling, if CPU is allocated to the process with larger burst time then the processes with small burst time may have to starve.
5. Preemptive scheduling is quite **flexible** because the critical processes are allowed to access CPU as they arrive into the ready queue, no matter what process is executing currently. Non-preemptive scheduling is **rigid** as even if a critical process enters the ready queue the process running CPU is not disturbed.

6. The Preemptive Scheduling is cost associative as it has to maintain the integrity of shared data which is not the case with Non-preemptive Scheduling.

Conclusion:

It is not that preemptive scheduling is better than non-preemptive scheduling or vise-verse. All depends on how a scheduling minimizes the average waiting time of the processes and maximizes CPU utilization.