States of a Process in Operating Systems

In an operating system, a process is a program that is being executed. During its execution, a process goes through different states. Understanding these states helps us see how the operating system manages processes, ensuring that the computer runs efficiently. Please refer Process in Operating System to understand more details about processes.

Each process goes through several stages throughout its life cycle. In this article, We discuss different states of process in detail.

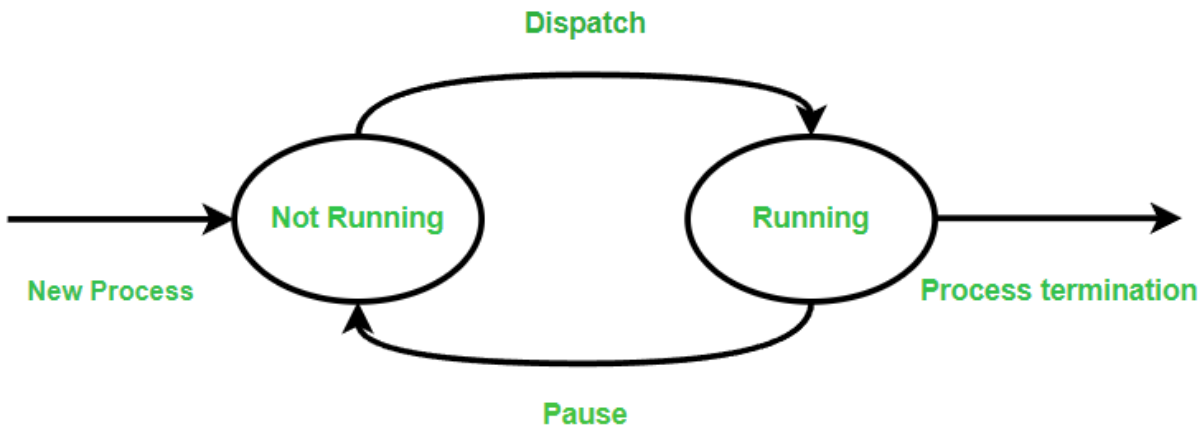
Process Lifecycle

When you run a program (which becomes a process), it goes through different phases before it completion. These phases, or states, can vary depending on the operating system, but the most common process lifecycle includes **two**, **five**, or **seven** states. Here's a simple explanation of these states:

The Two-State Model

The simplest way to think about a process's lifecycle is with just **two states**:

1. **Running**: This means the process is actively using the CPU to do its work.
2. **Not Running**: This means the process is not currently using the CPU. It could be waiting for something, like user input or data, or it might just be paused.



Two State Process Model

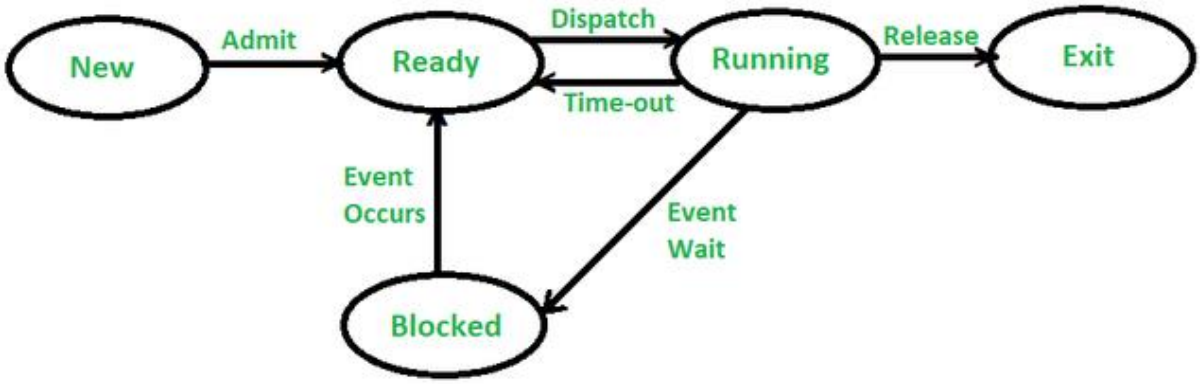
When a new process is created, it starts in the **not running** state. Initially, this process is kept in a program called the **dispatcher**.

Here's what happens step by step:

1. **Not Running State**: When the process is first created, it is not using the CPU.
2. **Dispatcher Role**: The dispatcher checks if the CPU is free (available for use).
3. **Moving to Running State**: If the CPU is free, the dispatcher lets the process use the CPU, and it moves into the **running** state.
4. **CPU Scheduler Role**: When the CPU is available, the **CPU scheduler** decides which process gets to run next. It picks the process based on a set of rules called the **scheduling scheme**, which varies from one operating system to another.

The Five-State Model

The **five-state process lifecycle** is an expanded version of the two-state model. The two-state model works well when all processes in the **not running** state are ready to run. However, in some operating systems, a process may not be able to run because it is waiting for something, like input or data from an external device. To handle this situation better, the **not running state** is divided into two separate states:



Five state Process Model

Here's a simple explanation of the five-state process model:

- **New:** This state represents a newly created process that hasn't started running yet. It has not been loaded into the main memory, but its process control block (PCB) has been created, which holds important information about the process.
- **Ready:** A process in this state is ready to run as soon as the CPU becomes available. It is waiting for the operating system to give it a chance to execute.

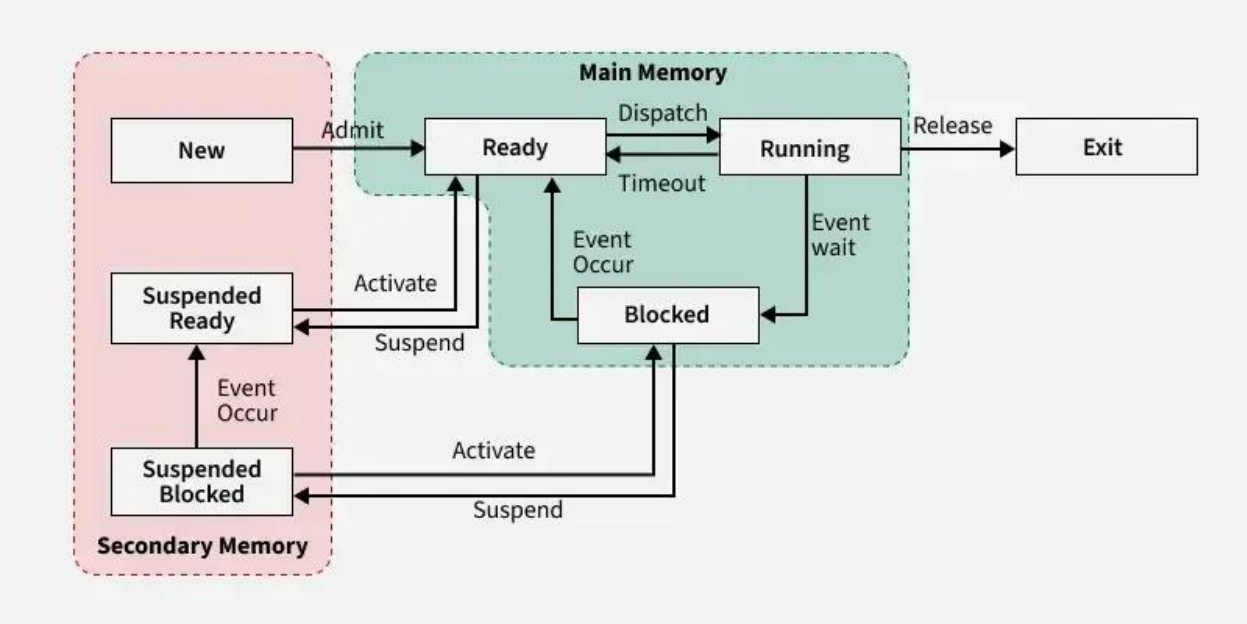
- **Running**: This state means the process is currently being executed by the CPU. Since we're assuming there is only one CPU, at any time, only one process can be in this state.
- **Blocked/Waiting:** This state means the process cannot continue executing right now. It is waiting for some event to happen, like the completion of an input/output operation (for example, reading data from a disk).
- **Exit/Terminate:** A process in this state has finished its execution or has been stopped by the user for some reason. At this point, it is released by the operating system and removed from memory.

The Seven-State Model

The states of a process are as follows:

- **New State:** In this step, the process is about to be created but not yet created. It is the program that is present in secondary memory that will be picked up by the OS to create the process.
- **Ready State: New -> Ready** to run. After the creation of a process, the process enters the ready state i.e. the process is loaded into the main memory. The process here is ready to run and is waiting to get the CPU time for its execution. Processes that are ready for execution by the CPU are maintained in a queue called a ready queue for ready processes.
- **Run State:** The process is chosen from the ready queue by the OS for execution and the instructions within the process are executed by any one of the available processors.
- **Blocked or Wait State:** Whenever the process requests access to I/O needs input from the user or needs access to a critical region(the lock for which is already acquired) it enters the blocked or waits state. The process continues to wait in the main memory and does not require CPU. Once the I/O operation is completed the process goes to the ready state.
- **Terminated or Completed State:** Process is killed as well as PCB is deleted. The resources allocated to the process will be released or deallocated.
- **Suspend Ready:** Process that was initially in the ready state but was swapped out of main memory(refer to Virtual Memory topic) and placed onto external storage by the scheduler is said to be in suspend ready state. The process will transition back to a ready state whenever the process is again brought onto the main memory.

- **Suspend Wait or Suspend Blocked:** Similar to suspend ready but uses the process which was performing I/O operation and lack of main memory caused them to move to secondary memory. When work is finished it may go to suspend ready.



- **CPU and I/O Bound Processes:** If the process is intensive in terms of CPU operations, then it is called CPU bound process. Similarly, If the process is intensive in terms of I/O operations then it is called I/O bound process.

How Does a Process Move From One State to Other State?

A process can move between different states in an operating system based on its execution status and resource availability. Here are some examples of how a process can move between different states:

- **New to Ready:** When a process is created, it is in a new state. It moves to the ready state when the operating system has allocated resources to it and it is ready to be executed.
- **Ready to Running:** When the CPU becomes available, the operating system selects a process from the ready queue depending on various scheduling algorithms and moves it to the running state.
- **Running to Blocked:** When a process needs to wait for an event to occur (I/O operation or system call), it moves to the blocked state. For example, if a process needs to wait for user input, it moves to the blocked state until the user provides the input.

- **Running to Ready:** When a running process is preempted by the operating system, it moves to the ready state. For example, if a higher-priority process becomes ready, the operating system may preempt the running process and move it to the ready state.
- **Blocked to Ready:** When the event a blocked process was waiting for occurs, the process moves to the ready state. For example, if a process was waiting for user input and the input is provided, it moves to the ready state.
- **Running to Terminated:** When a process completes its execution or is terminated by the operating system, it moves to the terminated state.

Types of Schedulers

- **Long-Term Scheduler:** Decides how many processes should be made to stay in the ready state. This decides the degree of multiprogramming. Once a decision is taken it lasts for a long time which also indicates that it runs infrequently. Hence it is called a long-term scheduler.
- **Short-Term Scheduler:** Short-term scheduler will decide which process is to be executed next and then it will call the dispatcher. A dispatcher is a software that moves the process from ready to run and vice versa. In other words, it is context switching. It runs frequently. Short-term scheduler is also called CPU scheduler.
- **Medium Scheduler:** Suspension decision is taken by the medium-term scheduler. The medium-term scheduler is used for swapping which is moving the process from main memory to secondary and vice versa. The swapping is done to reduce degree of multiprogramming.

Multiprogramming

We have many processes ready to run. There are two types of multiprogramming:

- **Preemption:** Process is forcefully removed from CPU. Pre-emption is also called time sharing or multitasking.
- **Non-Preemption:** Processes are not removed until they complete the execution. Once control is given to the CPU for a process execution, till the CPU releases the control by itself, control cannot be taken back forcibly from the CPU.

Degree of Multiprogramming

The number of processes that can reside in the ready state at maximum decides the degree of multiprogramming, e.g., if the degree of programming = 100, this means 100 processes can reside in the ready state at maximum.

Operation on The Process

- **Creation:** The process will be ready once it has been created, enter the ready queue (main memory), and be prepared for execution.
- **Planning:** The operating system picks one process to begin executing from among the numerous processes that are currently in the ready queue. Scheduling is the process of choosing the next process to run.
- **Application:** The processor begins running the process as soon as it is scheduled to run. During execution, a process may become blocked or wait, at which point the processor switches to executing the other processes.
- **Killing or Deletion:** The OS will terminate the process once its purpose has been fulfilled. The process's context will be over there.
- **Blocking:** When a process is waiting for an event or resource, it is blocked. The operating system will place it in a blocked state, and it will not be able to execute until the event or resource becomes available.
- **Resumption:** When the event or resource that caused a process to block becomes available, the process is removed from the blocked state and added back to the ready queue.
- **Context Switching:** When the operating system switches from executing one process to another, it must save the current process's context and load the context of the next process to execute. This is known as context switching.
- **Inter-Process Communication:** Processes may need to communicate with each other to share data or coordinate actions. The operating system provides mechanisms for inter-process communication, such as shared memory, message passing, and synchronization primitives.
- **Process Synchronization:** Multiple processes may need to access a shared resource or critical section of code simultaneously. The operating system provides synchronization mechanisms to ensure that only one process can access the resource or critical section at a time.
- **Process States:** Processes may be in one of several states, including ready, running, waiting, and terminated. The operating system manages the process states and transitions between them.

Features of The Process State

- A process can move from the running state to the waiting state if it needs to wait for a resource to become available.
- A process can move from the waiting state to the ready state when the resource it was waiting for becomes available.
- A process can move from the ready state to the running state when it is selected by the operating system for execution.
- The scheduling algorithm used by the operating system determines which process is selected to execute from the ready state.
- The operating system may also move a process from the running state to the ready state to allow other processes to execute.
- A process can move from the running state to the terminated state when it completes its execution.
- A process can move from the waiting state directly to the terminated state if it is aborted or killed by the operating system or another process.
- A process can go through ready, running and waiting state any number of times in its lifecycle but new and terminated happens only once.
- The process state includes information about the program counter, CPU registers, memory allocation, and other resources used by the process.
- The operating system maintains a process control block (PCB) for each process, which contains information about the process state, priority, scheduling information, and other process-related data.
- The process state diagram is used to represent the transitions between different states of a process and is an essential concept in process management in operating systems.

Process Control Block

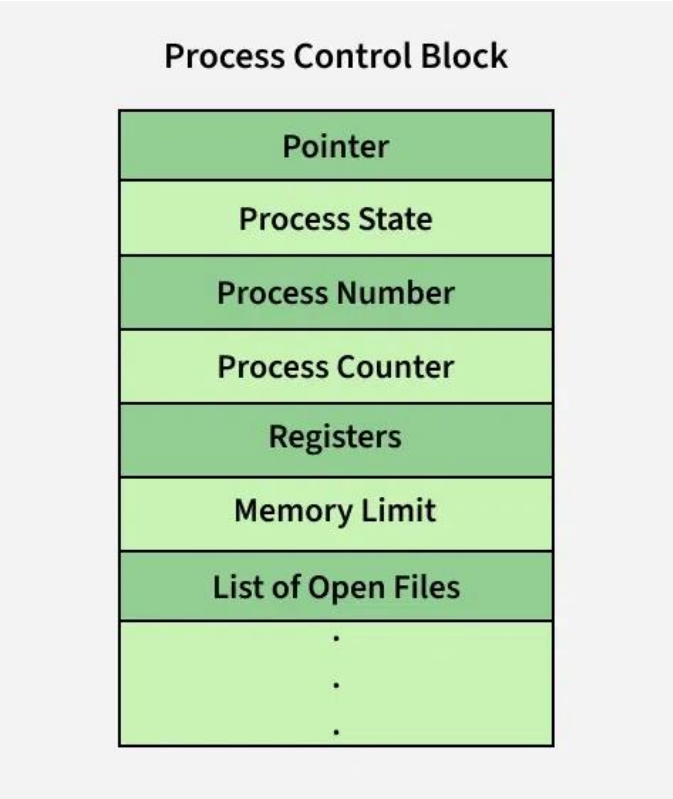
While creating a process, the operating system performs several operations. To identify the processes, it assigns a process identification number (PID) to each process. As the operating system supports multi-programming, it needs to keep track of all the processes. For this task, the process control block (PCB) is used to track the process's execution status. Each block of memory contains information about the process state, program counter, stack pointer, status of opened files, scheduling algorithms, etc.

All this information is required and must be saved when the process is switched from one state to another. When the process makes a transition from one state to another, the operating

system must update information in the process's PCB. A **Process Control Block (PCB)** contains information about the process, i.e. registers, quantum, priority, etc. The **Process Table** is an array of PCBs, which logically contains a PCB for all of the current processes in the system.

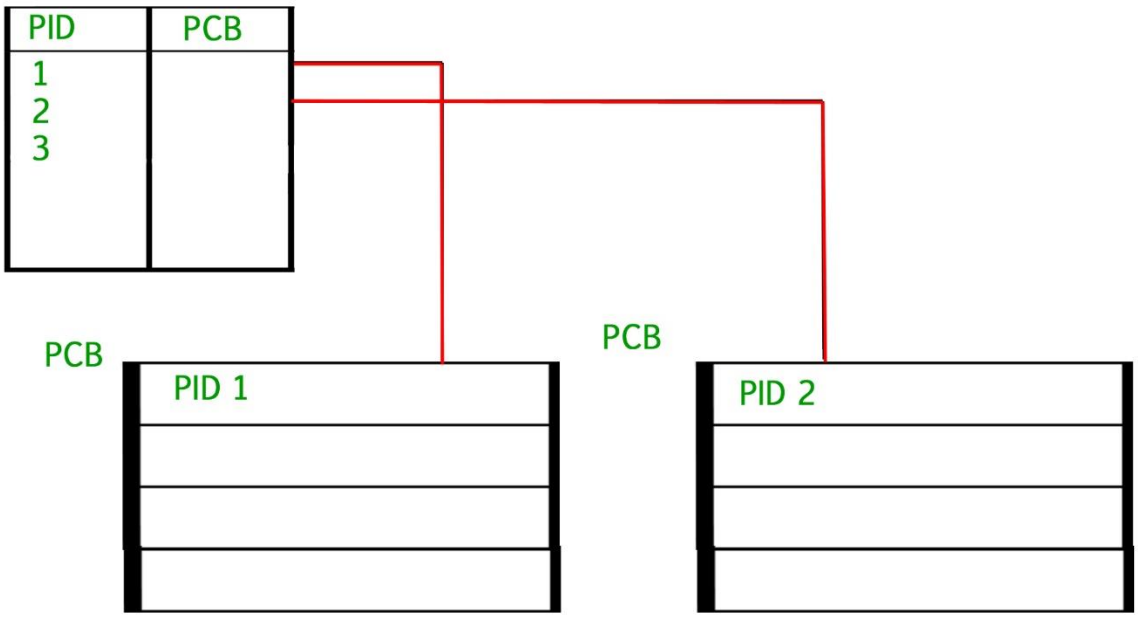
Structure of the Process Control Block

A Process Control Block (PCB) is a data structure used by the operating system to manage information about a process. The process control keeps track of many important pieces of information needed to manage processes efficiently. The diagram helps explain some of these key data items.



- **Pointer:** It is a stack pointer that is required to be saved when the process is switched from one state to another to retain the current position of the process.
- **Process state:** It stores the respective state of the process.
- **Process number:** Every process is assigned a unique id known as process ID or PID which stores the process identifier.
- **Program counter:** Program Counter stores the counter, which contains the address of the next instruction that is to be executed for the process.

- **Register:** Registers in the PCB, it is a data structure. When a processes is running and it's time slice expires, the current value of process specific registers would be stored in the PCB and the process would be swapped out. When the process is scheduled to be run, the register values is read from the PCB and written to the CPU registers. This is the main purpose of the registers in the PCB.
- **Memory limits:** This field contains the information about memory management system used by the operating system. This may include page tables, segment tables, etc.
- **List of Open files:** This information includes the list of files opened for a process.



Process table and process control block

Additional Points to Consider for Process Control Block (PCB)

- **Interrupt Handling:** The PCB also contains information about the interrupts that a process may have generated and how they were handled by the operating system.
- **Context Switching:** The process of switching from one process to another is called context switching. The PCB plays a crucial role in context switching by saving the state of the current process and restoring the state of the next process.
- **Real-Time Systems:** Real-time operating systems may require additional information in the PCB, such as deadlines and priorities, to ensure that time-critical processes are executed in a timely manner.

- **Virtual Memory Management:** The PCB may contain information about a process virtual memory management, such as page tables and page fault handling.
- **Fault Tolerance:** Some operating systems may use multiple copies of the PCB to provide fault tolerance in case of hardware failures or software errors.

Location of The Process Control Block

The Process Control Block (PCB) is stored in a special part of memory that normal users can't access. This is because it holds important information about the process. Some operating systems place the PCB at the start of the kernel stack for the process, as this is a safe and secure spot.

Advantages

Advantages of Process Table

- **Keeps Track of Processes:** It helps the operating system know which processes are running, waiting, or completed.
- **Helps in Scheduling:** The process table provides information needed to decide which process should run next.
- **Easy Process Management:** It organizes all the details about processes in one place, making it simple for the OS to manage them.

Advantages of Process Control Block (PCB)

- **Stores Process Details:** PCB keeps all the important information about a process, like its state, ID, and resources it uses.
- **Helps Resume Processes:** When a process is paused, PCB saves its current state so it can continue later without losing data.
- **Ensures Smooth Execution:** By storing all the necessary details, PCB helps the operating system run processes efficiently and without interruptions.

Disadvantages

Disadvantages of Process Table

- **Takes Up Memory** : The process table needs space to store information about all processes, which can use a lot of memory in systems with many processes.
- **Slower Operations** : When there are too many processes, searching or updating the table can take more time, slowing down the system.

- **Extra Work for the System** : The operating system has to constantly update the process table, which adds extra work and can reduce overall system performance.

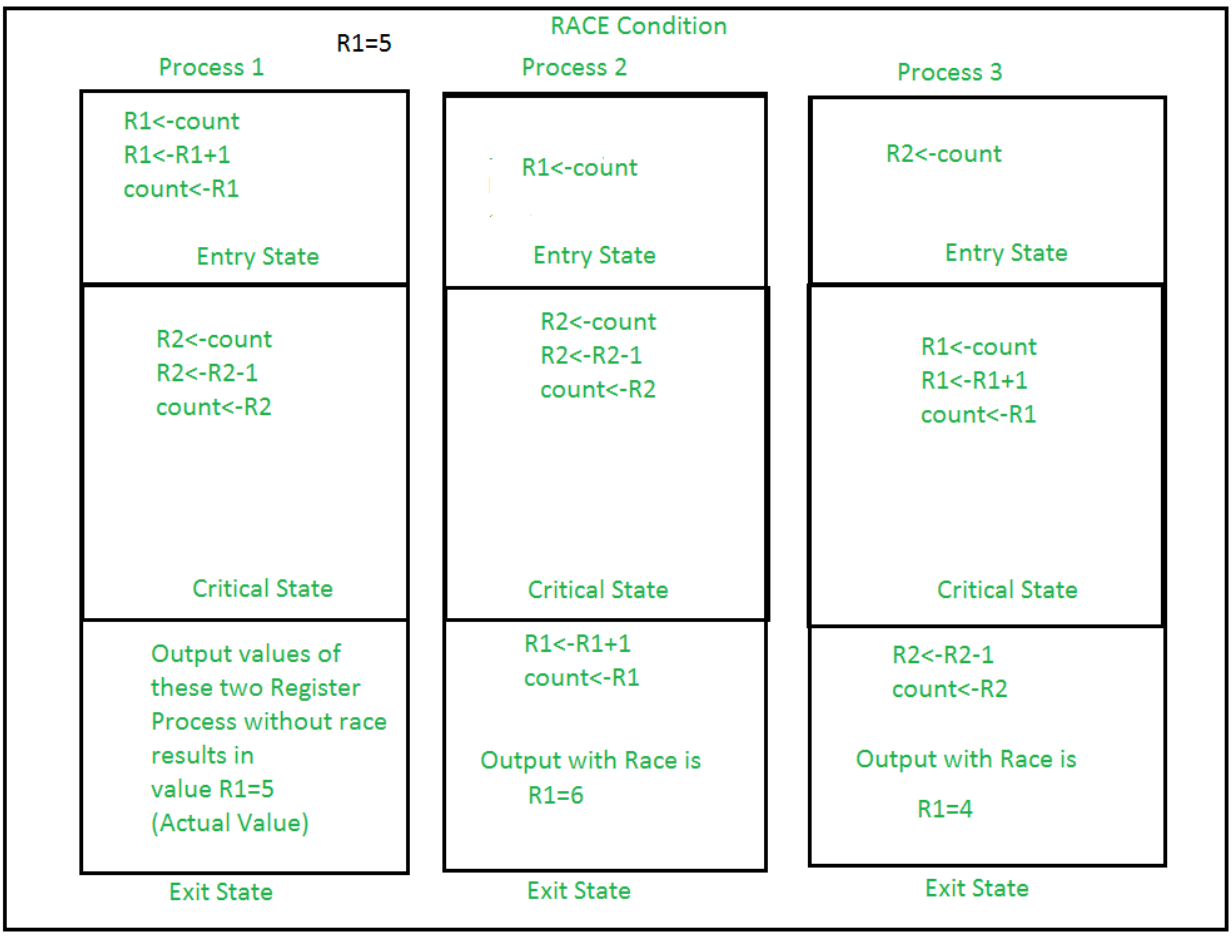
Disadvantages of Process Control Block (PCB)

- **Uses More Memory** : Each process needs its own PCB, so having many processes can consume a lot of memory.
- **Slows Context Switching** : During context switching , the system has to update the PCB of the old process and load the PCB of the new one, which takes time and affects performance.
- **Security Risks** : If the PCB is not well-protected, someone could access or modify it, causing security problems for processes.

Cooperating Process Systems

In an operating system, everything is around the process. How the process goes through several different states. So in this article, we are going to discuss one type of process called as Cooperating Process. In the operating system there are two types of processes:

- **Independent Process:** Independent Processes are those processes whose task is not dependent on any other processes.
- **Cooperating Process:** Cooperating Processes are those processes that depend on other processes or processes. They work together to achieve a common task in an operating system. These processes interact with each other by sharing the resources such as CPU, memory, and I/O devices to complete the task.



So now let's discuss the concept of cooperating processes and how they are used in operating systems.

- **Inter-Process Communication (IPC):** Cooperating processes interact with each other via Inter-Process Communication (IPC). As they are interacting to each other and sharing some resources with another so running task get the synchronization and possibilities of deadlock decreases. To implement the IPC there are many options such as pipes, message queues, semaphores, and shared memory.
- **Concurrent execution:** These cooperating processes executes simultaneously which can be done by operating system scheduler which helps to select the process from ready queue to go to the running state. Because of concurrent execution of several processes the completion time decreases.
- **Resource sharing:** In order to do the work, cooperating processes cooperate by sharing resources including CPU, memory, and I/O hardware. If several processes are sharing

resources as if they have their turn, synchronization increases as well as the response time of process increase.

- **Deadlocks:** As cooperating processes shares their resources, there might be a deadlock condition. Deadlock means if p1 process holds the resource A and wait for B and p2 process hold the B and wait for A. In this condition deadlock occur in cooperating process. To avoid deadlocks, operating systems typically use algorithms such as the Banker's algorithm to manage and allocate resources to processes.
- **Process scheduling:** Cooperating processes runs simultaneously but after context switch, which process should be next on CPU to executes, this is done by the scheduler. Scheduler do it by using several scheduling algorithms such as Round-Robin, FCFS, SJF, Priority etc.

Difference Between Process and Thread

- The table below represents the difference between process and thread.

Process	Thread
Process means a program in execution.	Thread means a segment of a process.
A process takes more time to terminate.	A thread takes less time to terminate.
It takes more time for creation.	It takes less time for creation.
It also takes more time for context switching.	It takes less time for context switching.
A process is less efficient in terms of communication.	Thread is more efficient in terms of communication.
Multiprogramming holds the concepts of multi-process.	We don't need multi programs in action for multiple threads because a single process consists of multiple threads.

Process	Thread
Every process runs in its own memory.	Threads share memory.
A process is heavyweight compared to a thread.	A Thread is lightweight as each thread in a process shares code, data, and resources.
Process switching uses an interface in an operating system.	Thread switching may not require calling involvement of operating system.
If one process is blocked, then it will not affect the execution of other processes.	If a user-level thread is blocked, then all other user-level threads are blocked.
A process has its own Process Control Block, Stack, and Address Space.	Thread has Parents' PCB, its own Thread Control Block, and Stack and common Address space.
Changes to the parent process do not affect child processes.	Since all threads of the same process share address space and other resources so any changes to the main thread may affect the behavior of the other threads of the process.
A system call is involved in it.	No system call is involved, it is created using APIs.
A process does not share data with each other.	Threads share data with each other.

Hierarchy of Processes

In operating systems, processes are typically organized into a **hierarchical structure** that allows the operating system to manage and track the execution of programs effectively. The hierarchy

of processes reflects how different processes are related to each other, either as parent-child relationships or as part of a larger structure like threads and jobs. Here's an overview of the hierarchy of processes in operating systems:

1. Root (or Init Process)

- At the top of the hierarchy is the **root process** (in Unix/Linux) or **init process** (in many Unix-like systems). This is the first process that is created when the operating system starts.
- Its **PID** (Process ID) is typically **1**.
- The init process is responsible for starting other system processes and initializing various subsystems of the OS. It is a parent to many other system processes.

2. Parent Processes

- Any process that creates another process is called a **parent process**.
- Parent processes often create child processes to perform specific tasks. For example, a parent process might spawn a child process to execute a command or run an application.

3. Child Processes

- **Child processes** are created by a parent process, and they inherit some properties from their parent (such as user privileges).
- A child process can also create its own child processes, and this chain can continue.
- In many operating systems, when a child process finishes executing, it sends an exit status to its parent.

4. Siblings

- Processes that share the same parent process are called **siblings**.
- These processes are at the same level in the hierarchy and can communicate with each other if required.

5. Grandparent and Grandchild Processes

- A process that is the parent of the parent process is called a **grandparent process**.
- Similarly, a **grandchild process** is created by a child process, making it two levels below the parent process in the hierarchy.

6. Zombie Processes

- A **zombie process** is a process that has completed execution but still has an entry in the process table because the parent has not yet read its exit status.
- These processes are still part of the hierarchy until they are completely removed.

7. Orphan Processes

- An **orphan process** is a process whose parent has terminated, leaving it without a controlling process.
- In Unix-like systems, orphan processes are usually adopted by the **init process** (PID 1), which ensures that these processes are properly managed and terminated.