

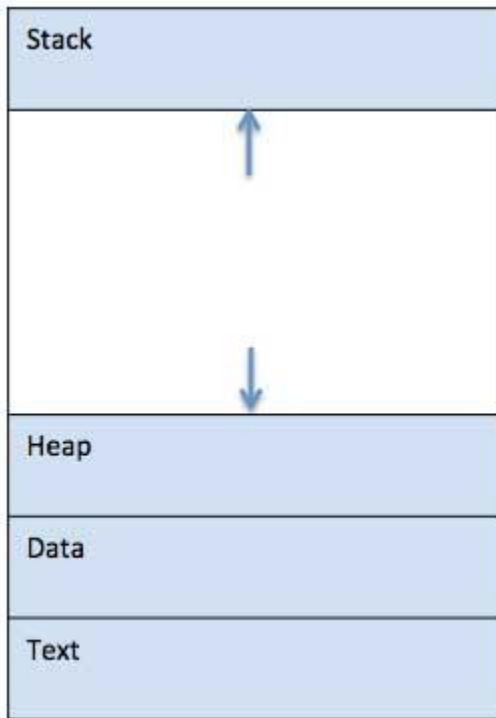
Process

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

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

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

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



S.N.	Component & Description
1	Stack The process Stack contains the <u>temporary data</u> such as <u>method/function parameters</u> , <u>return address</u> and <u>local variables</u> .
2	Heap This is <u>dynamically allocated memory</u> to a process during its <u>run time</u> .
3	Text This includes the <u>current activity</u> represented by the <u>value of Program Counter</u> and the contents of the processor's registers.
4	Data

This section contains the global and static variables.

Program

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

```
#include <stdio.h>

int main() {
    printf("Hello, World! \n");
    return 0;
}
```

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

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

Process Life Cycle

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

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

New State

When a program in secondary memory is started for execution, the process is said to be in a new state.

Ready State

After being loaded into the main memory and ready for execution, a process transitions from a new to a ready state. The process will now be in the ready state, waiting for the processor to execute it. Many processes may be in the ready stage in a multiprogramming environment.

Run State

After being allotted the CPU for execution, a process passes from the ready state to the run state.

Terminate State

When a process's execution is finished, it goes from the run state to the terminate state. The operating system deletes the process control box (or PCB) after it enters the terminate state.

Block or Wait State

If a process requires an Input/Output operation or a blocked resource during execution, it changes from run to block or the wait state. The process advances to the ready state after the I/O operation is completed or the resource becomes available.

Suspend Ready State

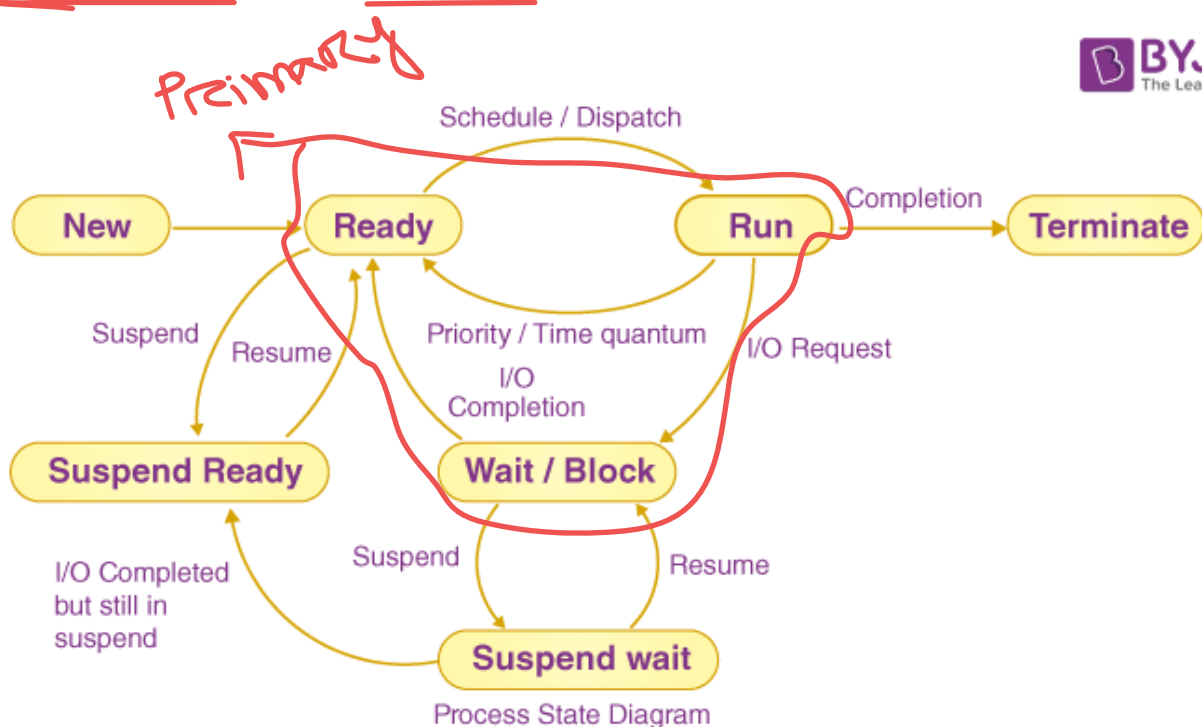
If a process with a higher priority needs to be executed while the main memory is full, the process goes from ready to suspend ready state. Moving a lower-priority process from the ready state to the suspend ready state freees up space in the ready state for a higher-priority process.

Until the main memory becomes available, the process stays in the suspend-ready state. The process is brought to its ready state when the main memory becomes accessible.

Suspend Wait State

If a process with a higher priority needs to be executed while the main memory is full, the process goes from the wait state to the suspend wait state. Moving a lower-priority process from the wait state to the suspend wait state freees up space in the ready state for a higher-priority process.

The process gets moved to the suspend-ready state once the resource becomes accessible. The process is shifted to the ready state once the main memory is available.




Process Control Block (PCB)

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

S.N.	Information & Description
1	Process State The <u>current state</u> of the process i.e., whether it is <u>ready, running, waiting, or whatever</u> .
2	Process <u>privileges</u>

	This is required <u>to allow/disallow access</u> to system resources.
3	Process ID <u>Unique identification</u> for each of the process in the operating system.
4	Pointer A pointer <u>to parent process</u> .
5	Program Counter Program Counter is a <u>pointer</u> to the <u>address of the next instruction</u> to be executed for this process.
6	CPU registers Various CPU registers where process need to be stored for execution for <u>running state</u> .
7	CPU Scheduling Information <u>Process priority and other scheduling information</u> which is required to schedule the process.
8	Memory management information This includes the information of <u>page table, memory limits, Segment table</u> depending on memory used by the operating system.
9	Accounting information This includes the <u>amount of CPU</u> used for <u>process execution, time limits, execution ID</u> etc.
10	IO status information This includes a list of <u>I/O devices</u> allocated to the process.

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



Process ID
State
Pointer
Priority
Program counter
CPU registers
I/O information
Accounting information
etc....

The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.