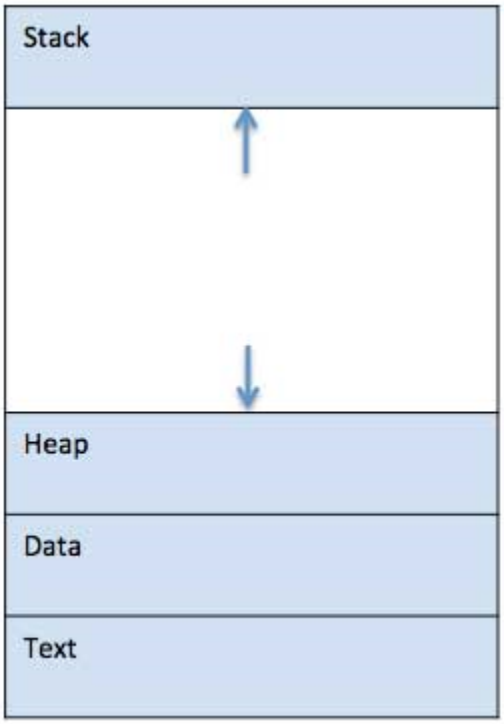
Process

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 temporary data such as method/function parameters, return address and local variables.
2	**Heap** This is dynamically allocated memory to a process during its run time.
3	**Text**

	This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
4	**Data** This section contains the global and static variables.

From <https://www.tutorialspoint.com/operating_system/os_processes.htm>

What is Process Control Block (PCB)?
Computer EngineeringMCAOperating System

Process Control Block is a data structure that contains information of the process related to it. The process control block is

also known as a task control block, entry of the process table, etc.

It is very important for process management as the data structuring for processes is done in terms of the PCB. It also defines the current state of the operating system.

Structure of the Process Control Block

The process control stores many data items that are needed for efficient process management. Some of these data items are explained with the help of the given diagram –

Process Control Block (PCB)

The following are the data items –

Process State

This specifies the process state i.e. new, ready, running, waiting or terminated.

Process Number

This shows the number of the particular process.

Program Counter

This contains the address of the next instruction that needs to be executed in the process.

Registers

This specifies the registers that are used by the process. They may include accumulators, index registers, stack pointers, general purpose registers etc.

List of Open Files

These are the different files that are associated with the process

From <https://www.tutorialspoint.com/what-is-process-control-block-pcb>

1. Process creation: This operation involves creating a new process by the operating system. When a user starts a new program, the operating system creates a new process for it.

2. Process termination: This operation involves terminating a process when it is no longer needed or has completed its execution.
3. Process suspension and resumption: These operations involve temporarily stopping a process's execution, allowing other processes to run, and later resuming the suspended process from where it left off.
4. Process scheduling: This operation involves assigning system resources, such as CPU time, to different processes, based on various scheduling algorithms.
5. Inter-process communication: This operation involves allowing processes to communicate with each other, such as by sending messages or sharing data.
6. Process synchronization: This operation involves coordinating the execution of multiple processes to avoid conflicts and ensure that they cooperate correctly.

7. Process management: This operation involves managing various aspects of a process, such as its priority, memory usage, and access to system resources.

From <https://chat.openai.com/>

In a computer system, scheduling is the process of allocating system resources, such as CPU time, to different processes. There are several types of scheduling algorithms that an operating system can use to schedule processes. Here are some of the most common types of scheduling:

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-Next (SJN) Scheduling
- Priority Scheduling
- Shortest Remaining Time
- Round Robin(RR) Scheduling

- Multiple-Level Queues Scheduling

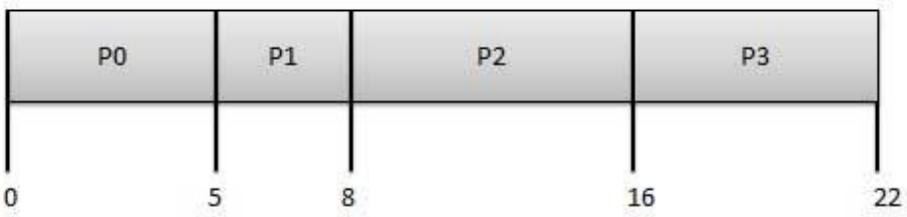
  hese algorithms are either **non-preemptive or preemptive**. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

  First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.

- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Wait time of each process is as follows –

Process	Wait Time : Service Time - Arrival Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	8 - 2 = 6
P3	16 - 3 = 13

Average Wait Time: (0+4+6+13) / 4 = 5.75

Shortest Job Next (SJN)

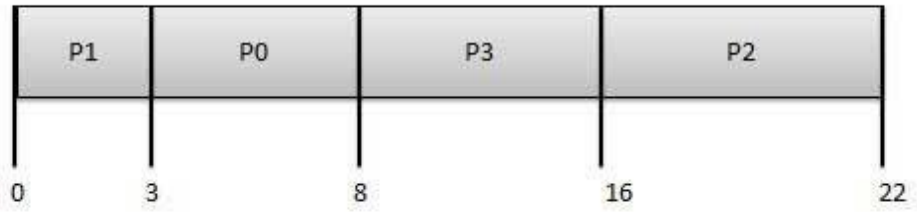
- This is also known as **shortest job first**, or SJF
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.

Given: Table of processes, and their Arrival time, Execution time

Process	Arrival Time	Execution Time	Service Time
P0	0	5	0

P1	1	3	5
P2	2	8	14
P3	3	6	8

Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	16
P3	3	6	8



Waiting time of each process is as follows –

Process	Waiting Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	14 - 2 = 12
P3	8 - 3 = 5

Average Wait Time: (0 + 4 + 12 + 5)/4 = 21 / 4 = 5.25

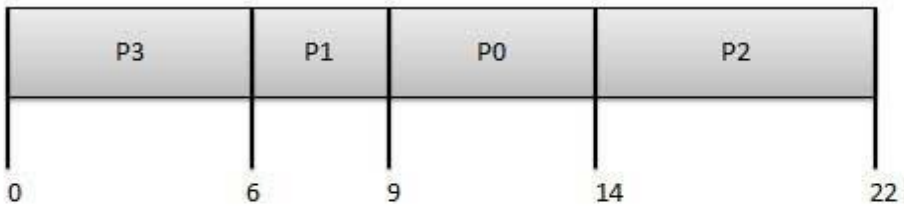
Priority Based Scheduling

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Given: Table of processes, and their Arrival time, Execution time, and priority. Here we are considering 1 is the lowest priority.

Process	Arrival Time	Execution Time	Priority	Service Time
P0	0	5	1	0
P1	1	3	2	11
P2	2	8	1	14
P3	3	6	3	5

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0



Waiting time of each process is as follows –

Process	Waiting Time

P0	0 - 0 = 0
P1	11 - 1 = 10
P2	14 - 2 = 12
P3	5 - 3 = 2

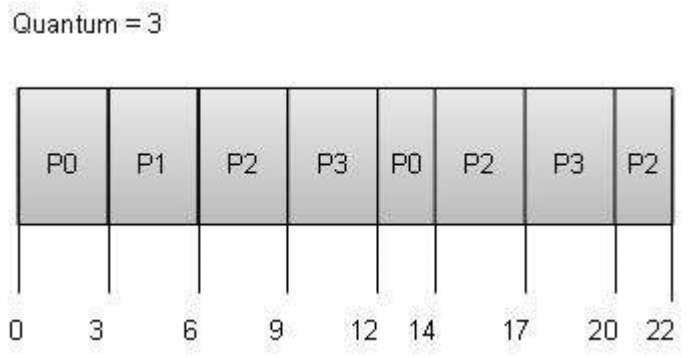
Average Wait Time: (0 + 10 + 12 + 2)/4 = 24 / 4 = 6

Shortest Remaining Time

- Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive systems where required CPU time is not known.
- It is often used in batch environments where short jobs need to give preference.

Round Robin Scheduling

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.



Wait time of each process is as follows –

Process	Wait Time : Service Time - Arrival Time
P0	(0 - 0) + (12 - 3) = 9

P1	(3 - 1) = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

Average Wait Time: (9+2+12+11) / 4 = 8.5

Multiple-Level Queues Scheduling

Multiple-level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.

- Multiple queues are maintained for processes with common characteristics.
- Each queue can have its own scheduling algorithms.
- Priorities are assigned to each queue.

For example, CPU-bound jobs can be scheduled in one queue and all I/O-

bound jobs in another queue. The Process Scheduler then alternately selects jobs from each queue and assigns them to the CPU based on the algorithm assigned to the queue.

From <https://www.tutorialspoint.com/operating_system/os_process_scheduling_algorithms.htm>

From <https://chat.openai.com/>

Cooperating processes are those that can affect or are affected by other processes running on the system. Cooperating processes may share data with each other.

Reasons for needing cooperating processes

There may be many reasons for the requirement of cooperating processes. Some of these are given as follows –

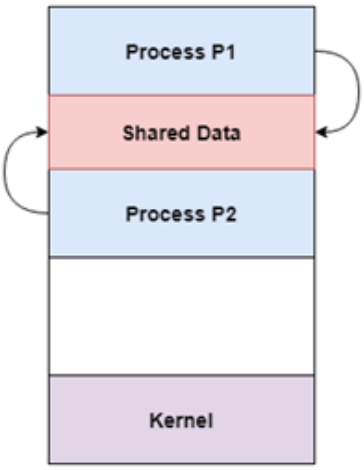
- **Modularity**
  Modularity involves dividing complicated tasks into smaller subtasks. These subtasks can completed by different cooperating processes. This leads to faster and more efficient completion of the required tasks.
- **Information Sharing**
  Sharing of information between multiple processes can be accomplished using cooperating processes. This may include access to the same files. A mechanism is required so that the processes can access the files in parallel to each other.

- **Convenience**
  There are many tasks that a user needs to do such as compiling, printing, editing etc. It is convenient if these tasks can be managed by cooperating processes.
- **Computation Speedup**
  Subtasks of a single task can be performed parallely using cooperating processes. This increases the computation speedup as the task can be executed faster. However, this is only possible if the system has multiple processing elements.

Methods of Cooperation

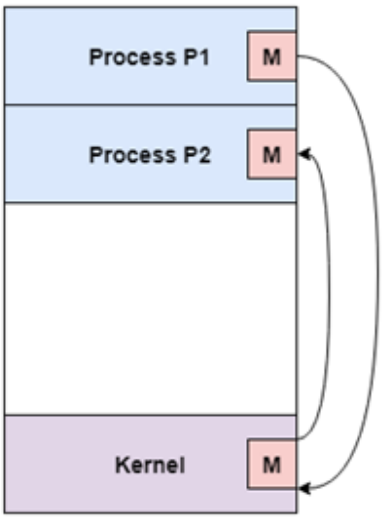
Cooperating processes can coordinate with each other using shared data or messages. Details about these are given as follows –

- **Cooperation by Sharing**
  The cooperating processes can cooperate with each other using shared data such as memory, variables, files, databases etc. Critical section is used to provide data integrity and writing is mutually exclusive to prevent inconsistent data. A diagram that demonstrates cooperation by sharing is given as follows –



In the above diagram, Process P1 and P2 can cooperate with each other using shared data such as memory, variables, files, databases etc.

- **Cooperation by Communication**
  The cooperating processes can cooperate with each other using messages. This may lead to deadlock if each process is waiting for a message from the other to perform a operation. Starvation is also possible if a process never receives a message.
  A diagram that demonstrates cooperation by communication is given as follows –

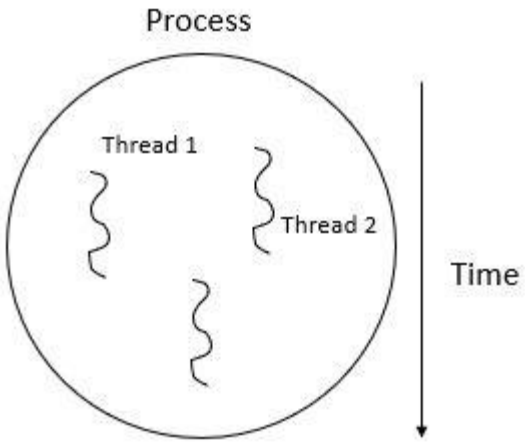


In the above diagram, Process P1 and P2 can cooperate with each other using messages to communicate.

From <https://www.tutorialspoint.com/cooperating-process>

A thread is a lightweight of process and is a basic unit of CPU utilization which consists of a program counter, a stack, and a set of registers.

Given below is the structure of thread in a process –



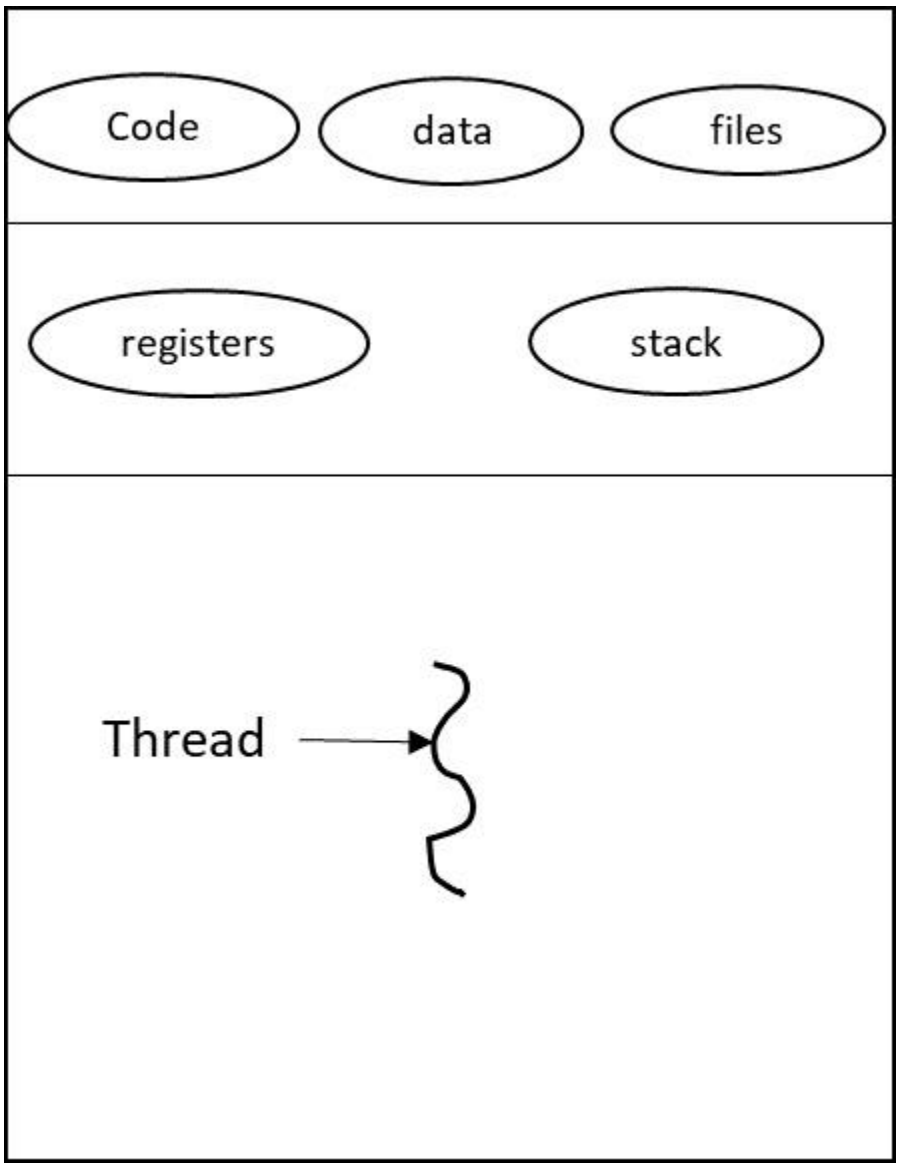
A process has a single thread of control where one program can counter and one sequence of instructions is carried out at any given time. Dividing an application or

a program into multiple sequential threads that run in quasi-parallel, the programming model becomes simpler.

Thread has the ability to share an address space and all of its data among themselves. This ability is essential for some specific applications.

Threads are lighter weight than processes, but they are faster to create and destroy than processes.

Let us see the **single thread model** which is as follows –



Now, let us see about **classical thread model** which is as follows –

A process contains a number of resources like address space, open files, accounting information, etc. In addition to these resources, a process is also having a

thread of control. For example, program counter, register contents, stack.

The idea of threads is to allow multiple threads of control to execute within one process. This is often called multithreading and threads are also known as lightweight processes.

Since threads in the same process share state and stack, switching between them is less expensive than switching between separate processes.

Individual threads within the same process are not completely independent but they are cooperating and all are from the same process.

The shared resources make it easier between threads to use each other’s resources. A new thread in the same process is created by a library routine like

thread_create. Similarly, thread_exit terminates a thread.

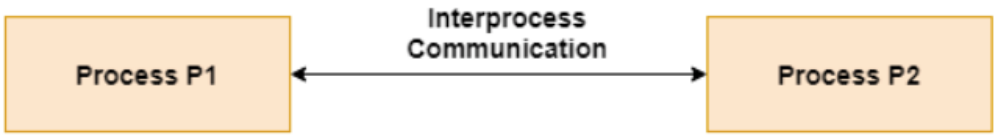
From <https://www.tutorialspoint.com/what-is-the-concept-of-thread>

USER LEVEL THREAD	KERNEL LEVEL THREAD
User thread are implemented by users.	kernel threads are implemented by OS.
OS doesn't recognized user level threads.	Kernel threads are recognized by OS.
Implementation of User threads is easy.	Implementation of Kernel thread is complicated.
Context switch time is less.	Context switch time is more.
Context switch requires no hardware support.	Hardware support is needed.
If one user level thread perform blocking operation then entire process will be blocked.	If one kernel thread perform blocking operation then another thread can continue execution.
Example : Java thread, POSIX threads.	Example : Window Solaris.

Interprocess communication is the mechanism provided by the operating system that allows processes to communicate with each other. This communication could involve a process letting another process know that some

event has occurred or the transferring of data from one process to another.

A diagram that illustrates interprocess communication is as follows –



From <https://www.tutorialspoint.com/what-is-interprocess-communication>

No, function callbacks and inter-process communication (IPC) are not the same thing, although they are related concepts in computer science.

1. A function callback is a programming technique where a function is passed as an argument to another function.
2. The receiving function can then call the passed function at a later time,

potentially with different arguments or in a different context.

3. Function callbacks are often used in event-driven programming or when implementing higher-order functions in functional programming. In this context, a function callback is simply a way of passing behavior as data.

1. Inter-process communication, on the other hand, is a mechanism for processes or threads to communicate with each other, typically in a concurrent or distributed system.
2. IPC can take many forms, such as message passing, shared memory, or remote procedure calls. The purpose of IPC is to enable processes or threads to coordinate their actions, exchange data, and synchronize their execution.

From <https://chat.openai.com/>

Actions taken by a kernel to context-switch between processes are -

- The OS must save the PC and user stack pointer of the currently executing process, in response to a clock interrupt and transfers control to the kernel clock interrupt handler
- Saving the rest of the registers, as well as other machine state, such as the state of the floating point registers, in the process PCB is done by the clock interrupt handler.
- The scheduler to determine the next process to execute is invoked the OS.
- Then the state of the next process from its PCB is retrieved by OS and restores the registers. The restore operation takes the processor back to the state in which the previous process was previously interrupted,

executing in user code with user-mode privileges.

Many architecture-specific operations, including flushing data and instruction caches also must be performed by Context switches.

From <https://www.tutorialspoint.com/actions-taken-by-a-kernel-to-context-switch-between-processes>

a) Pthread_creat()
b) fork()

- Burst Time (BT): This is how long a task needs to run on the computer to finish what it needs to do. For example, if a task is to print a document, the burst time would be how long it takes to actually print the document.

- Arrival Time (AT): This is when a task arrives and is ready to be worked on. For example, if a task to print a document arrives at 9:00am, that's the arrival time.
- Completion Time (CT): This is when a task finishes running on the computer. For example, if a task to print a document takes 5 minutes to run, and it arrives at 9:00am, the completion time would be 9:05am.
- Turnaround Time (TAT): This is how long it takes for a task to arrive, run, and then finish. It's the difference between the completion time and the arrival time. For example, if a task to print a document arrives at 9:00am and finishes at 9:05am, the turnaround time would be 5 minutes.
- Waiting Time (WT): This is how long a task waits in the queue before it can start running on the computer. For example, if a task arrives at 9:00am, but there are other tasks already running, it

> may have to wait a while before it can start running. The waiting time is how long it waited.

From <https://chat.openai.com/>