UNIT -2 PROCESS MANAGEMENT

Process: A process or task is an instance of a program in execution. The execution of a process must programs in a sequential manner. At any time at most one instruction is executed. The process includes the current activity as represented by the value of the program counter and the content of the processors registers. Also it includes the process stack which contain temporary data (such as method parameters return address and local variables) & a data section which contain global variables.

How Process Look like in Memory?

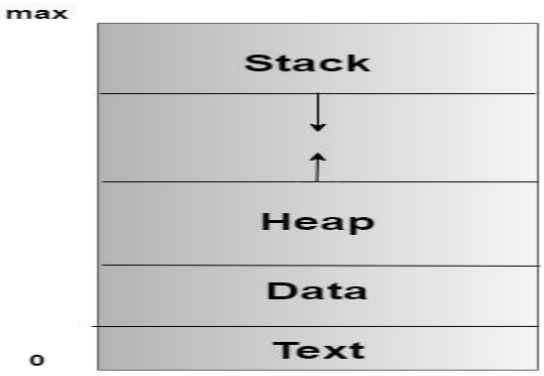
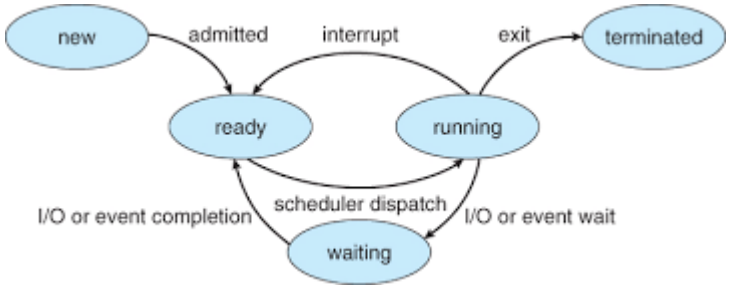


Figure: Process in the Memory

- The **Text section** is made up of the compiled program code, read in from non-volatile storage when the program is launched.
- The **Data section** is made up of the global and static variables, allocated and initialized prior to executing the main.
- The **Heap** is used for the dynamic memory allocation and is managed via calls to new, delete, malloc, free, etc.
- The **Stack** is used for local variables. Space on the stack is reserved for local variables when they are declared.

PROCESS STATES



The process executes when it changes the state. The state of a process is defined by the current activity of the process.

Each process may be in any one of the following states −

- **New** − The process is being created.
- **Running** − In this state the instructions are being executed.
- **Waiting** − The process is in waiting state until an event occurs like I/O operation completion or receiving a signal.
- **Ready** − The process is waiting to be assigned to a processor.
- **Terminated** − the process has finished execution.

It is important to know that only one process can be running on any processor at any instant. Many processes may be ready and waiting.

PROCESS CONTROL BLOCK (PCB)

Each process is represented in the OS by a process control block. It is also by a process control block. It is also known as task control block.

A process control block contains many pieces of information associated with a specific process. It includes the following information.

• **Process state:** The state may be new, ready, running, waiting or terminated state.

• **Program counter:** it indicates the address of the next instruction to be executed for this purpose.

• **CPU registers:** The registers vary in number & type depending on the computer architecture. It includes accumulators, index registers, stack pointer & general purpose registers, plus any condition- code information must be saved when an interrupt occurs to allow the process to be continued correctly after- ward.

• **CPU scheduling information:** This information includes process priority pointers to scheduling queues & any other scheduling parameters.

• **Memory management information:** This information may include such information as the value of the bar & limit registers, the page tables or the segment tables, depending upon the memory system used by the operating system.

• **Accounting information:** This information includes the amount of CPU and real time used, time limits, account number, job or process numbers and so on.

• **I/O Status Information:** This information includes the list of I/O devices allocated to this process, a list of open files and so on. The PCB simply serves as the repository for any information that may vary from process to process.

Process scheduling:

Scheduling is a fundamental function of OS. When a computer is multi-programmed, it has multiple processes completing for the CPU at the same time. If only one CPU is available, then a choice has to be made regarding which process to execute next. This decision making process is known as scheduling and the part of the OS that makes this choice is call in making this choice is called scheduling algorithm.

SCHEDULERS:

A process migrates between the various scheduling queues throughout its life-time purposes. The OS must select for scheduling processes from these queues in some fashion. This selection process is carried out by the appropriate scheduler. In a batch system, more processes are submitted and then executed immediately. So these processes are spooled to a mass storage device like disk, where they are kept for later execution. Types of schedulers: There are 3 types of schedulers mainly used:

1. Long term scheduler:

Long term scheduler selects process from the disk & loads them into memory for execution. It controls the degree of multi-programming i.e. no. of processes in memory. It executes less frequently than other schedulers. If the degree of multiprogramming is stable than the average rate of process creation is equal to the average departure rate of processes leaving the system. So, the long term scheduler is needed to be invoked only when a process leaves the system. Due to longer intervals between executions it can afford to take more time to decide which process should be selected for execution. Most processes in the CPU are either I/O bound or CPU bound. An I/O bound process (an interactive 'C' program is one that spends most of its time in I/O operation than it spends in doing I/O operation. A CPU bound process is one that spends more of its time in doing computations than I/O operations

(complex sorting program). It is important that the long term scheduler should select a good mix of I/O bound & CPU bound processes.

2. **Short - term scheduler:**

The short term scheduler selects among the process that are ready to execute & allocates the CPU to one of them. The primary distinction between these two schedulers is the frequency of their execution. The short-term scheduler must select a new process for the CPU quite frequently. It must execute at least one in 100ms. Due to the short duration of time between executions, it must be very fast.

3. **Medium - term scheduler:**

Some operating systems introduce an additional intermediate level of scheduling known as medium - term scheduler. The main idea behind this scheduler is that sometimes it is advantageous to remove processes from memory & thus reduce the degree of multiprogramming. At some later time, the process can be reintroduced into memory & its execution can be continued from where it had left off. This is called as swapping. The process is swapped out & swapped in later by medium term scheduler. Swapping is necessary to improve the process miss or due to some change in memory requirements, the available memory limit is exceeded which requires some memory to be freed up.

CPU Scheduling

CPU scheduling is a process that allows one process to use the CPU while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast, and fair.

Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute and allocates the CPU to one of them.

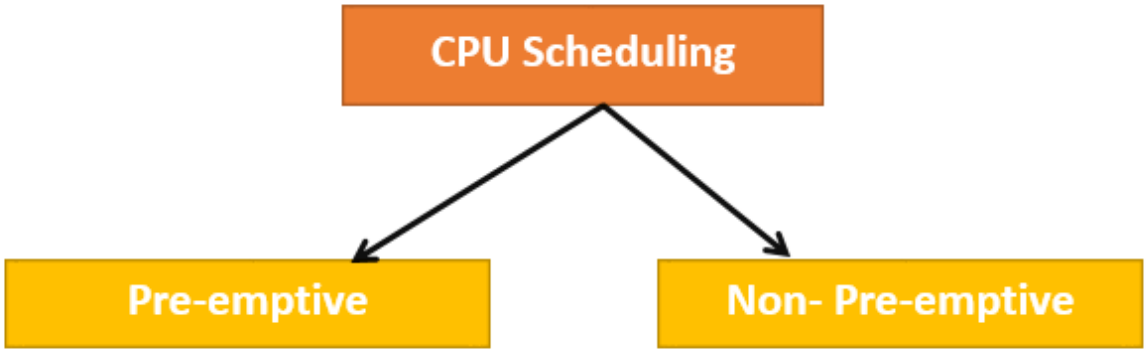
CPU Scheduling: Dispatcher

Another component involved in the CPU scheduling function is the Dispatcher. The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler. This function involves:

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program from where it left last time.

The dispatcher should be as fast as possible, given that it is invoked during every process switch. The time taken by the dispatcher to stop one process and start another process is known as the **Dispatch Latency.**

Types of CPU Scheduling



Preemptive Scheduling

In Preemptive Scheduling, the tasks are mostly assigned with their priorities. Sometimes it is important to run a task with a higher priority before another lower priority task, even if the lower priority task is still running. The lower priority task holds for some time and resumes when the higher priority task finishes its execution.

Non-Preemptive Scheduling

In this type of scheduling method, the CPU has been allocated to a specific process. The process that keeps the CPU busy will release the CPU either by switching context or terminating. It is the only method that can be used for various hardware platforms. That's because it doesn't need special hardware (for example, a timer) like preemptive scheduling.

When scheduling is Preemptive or Non-Preemptive?

To determine if scheduling is preemptive or non-preemptive, consider these four parameters:

1. A process switches from the running to the waiting state.
2. Specific process switches from the running state to the ready state.
3. Specific process switches from the waiting state to the ready state.
4. Process finished its execution and terminated.

CPU Scheduling Algorithms

The Purpose of a Scheduling algorithm

- The CPU uses scheduling to improve its efficiency.
- It helps you to allocate resources among competing processes.
- The maximum utilization of CPU can be obtained with multi-programming.
- The processes which are to be executed are in ready queue.

CPU Scheduling Criteria

The scheduling criterion is responsible for helping in the design of the good scheduler. These criteria are as follows −

CPU Utilization

The scheduling algorithm should be designed in such a way that the usage of the CPU should be as efficient as possible.

Throughput

It can be defined as the number of processes executed by the CPU in a given amount of time. It is used to find the efficiency of a CPU.

Response Time

The Response time is the time taken to start the job when the job enters the queues so that the scheduler should be able to minimize the response time.

Response time = Time at which the process gets the CPU for the first time - Arrival time

Turnaround time

Turnaround time is the total amount of time spent by the process from coming in the ready state for the first time to its completion.

Turnaround time = Burst time + Waiting time or

Turnaround time = Completion time - Arrival time

Waiting time

The Waiting time is nothing but where there are many jobs that are competing for the execution, so that the Waiting time should be minimized.

Waiting time = Turnaround time - Burst time

CPU Scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated first to the CPU. There are four types of CPU scheduling that exist.

1. First Come, First Served Scheduling (FCFS) Algorithm:

This is the simplest CPU scheduling algorithm. In this scheme, the process which requests the CPU first, that is allocated to the CPU first. The implementation of the FCFS algorithm is easily managed with a FIFO queue. When a process enters the ready queue its PCB is linked onto the rear of the queue. The average waiting time under FCFS policy is quiet long.

The FCFS algorithm is non preemptive means once the CPU has been allocated to a process then the process keeps the CPU until the release the CPU either by terminating or requesting I/O.

Characteristics of FCFS method:

- It offers non-preemptive and pre-emptive scheduling algorithm.
- Jobs are always executed on a first-come, first-serve basis
- It is easy to implement and use.
- However, this method is poor in performance, and the general wait time is quite high.

Process id	Arrival Time	Burst Time	Completion Time	Turnaround Time=CT-AT	Waiting Time=TAT-BT
P1	0	3	3	3	0
P2	1	5	8	7	2
P3	2	2	10	8	6
P4	3	4	14	11	7

Using FCFS algorithm find the average waiting time and average turnaround time if the order is P1 , P2 , P3 , P4 .

Solution: If the process arrived in the order P1 , P2 , P3 , P4 then according to the FCFS the Gantt chart will be:

P1	P2	P3	P4
0 3	8	10	14

Average waiting time = (0 + 2 + 6+ 7)/4 = 15/4 = 3.6
Average turnaround time = (3 + 7 + 8 + 11)/4 = 29/4 = 7.25

2. Shortest Job First Scheduling (SJF) Algorithm:

This algorithm associates with each process if the CPU is available. This scheduling is also known as shortest next CPU burst, because the scheduling is done by examining the length of the next CPU burst of the process rather than its total length.

The SJF algorithm may be either preemptive or non preemptive algorithm. The preemptive SJF is also known as shortest remaining time first.

Characteristics of SRT scheduling method:

- This method is mostly applied in batch environments where short jobs are required to be given preference.
- This is not an ideal method to implement it in a shared system where the required CPU time is unknown.
- Associate with each process as the length of its next CPU burst. So that operating system uses these lengths, which helps to schedule the process with the shortest possible time.

Characteristics of SJF Scheduling

- It is associated with each job as a unit of time to complete.
- In this method, when the CPU is available, the next process or job with the shortest completion time will be executed first.
- It is Implemented with non-preemptive policy.
- This algorithm method is useful for batch-type processing, where waiting for jobs to complete is not critical.
- It improves job output by offering shorter jobs, which should be executed first, which mostly have a shorter turnaround time.

Process id	Arrival Time	Burst Time	Completion Time	Turnaround Time=CT-AT	Waiting Time=TAT-BT
P1	0	3	3	3	0
P2	1	5	14	13	8
P3	2	2	5	3	1
P4	3	4	9	6	2

Solution: According to the SJF the Gantt chart will be

P1	P3	P4	P2
0 3	5	9	14

Average waiting time = (0 + 8 + 1 + 2)/4 = 11/4 = 2.5

Average turnaround time = (3+ 13 + 3 + 6)/4 = 25/4 = 5.25

Shortest Time Remaining First

Process id	Arrival Time	Burst Time	Completion Time	Turnaround Time=CT-AT	Waiting Time=TAT-BT
P1	0	8	17	17	7
P2	1	4	5	4	0
P3	2	9	26	24	15
P4	3	5	10	7	2

Solution: In this case the Gantt chart will be

P1	P2	P4	P1	P3
0 1	5	10	17	26

Average waiting time = (7 + 0 + 15 + 2)/4 = 24/4 = 27.5

Average turnaround time = (17+ 4 + 24 + 7)/4 = 52/4 = 13

3. Priority Scheduling Algorithm:

In this scheduling a priority is associated with each process and the CPU is allocated to the process with the highest priority. Equal priority processes are scheduled in FCFS manner.

- 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.

4. Round Robin Scheduling Algorithm:

This type of algorithm is designed only for the time sharing system. It is similar to FCFS scheduling with preemption condition to switch between processes. A small unit of time called quantum time or time slice is used to switch between the processes. The average waiting time under the round robin policy is quiet long.

Characteristics of Round-Robin Scheduling

- Round robin is a hybrid model which is clock-driven
- Time slice should be minimum, which is assigned for a specific task to be processed. However, it may vary for different processes.
- It is a real time system which responds to the event within a specific time limit.

Cooperating Processes

There are various processes in a computer system, which can be either independent or cooperating processes that operate in the operating system. It is considered independent when any other processes operating on the system may not impact a process. Process-independent processes don't share any data with other processes. On the other way, a collaborating process may be affected by any other process executing on the system. A cooperating process shares data with another.

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.

- **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.

THREADS

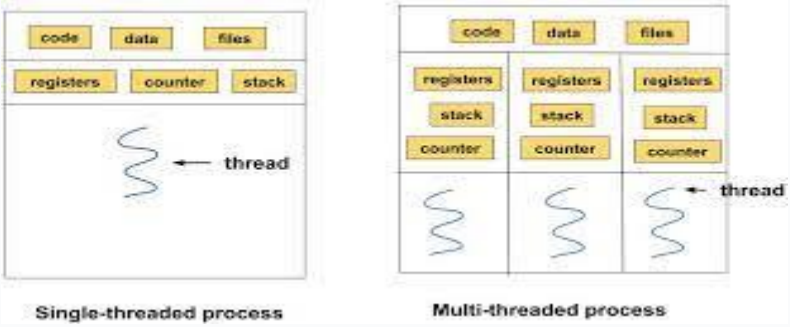
Thread is a sequential flow of tasks within a process. Threads in OS can be of the same or different types. Threads are used to increase the performance of the applications.

Each thread has its own program counter, stack, and set of registers. But the threads of a single process might share the same code and data/file. Threads are also termed as lightweight processes as they share common resources.

Threads in the operating system provide multiple benefits and improve the overall performance of the system. Some of the reasons threads are needed in the operating system are:

- Since threads use the same data and code, the operational cost between threads is low.
- Creating and terminating a thread is faster compared to creating or terminating a process.
- Context switching is faster in threads compared to processes.

Eg: While playing a movie on a device the audio and video are controlled by different threads in the background.



Components of Thread

A thread has the following three components:

1. Program Counter
2. Register Set
3. **Stack space**
4.

Types of Thread

1. User Level Thread:

User-level threads are implemented and managed by the user and the kernel is not aware of it.

- User-level threads are implemented using user-level libraries and the OS does not recognize these threads.
- User-level thread is faster to create and manage compared to kernel-level thread.
- Context switching in user-level threads is faster.
- If one user-level thread performs a blocking operation then the entire process gets blocked. Eg: POSIX threads, Java threads, etc.

2. Kernel level Thread:

Kernel level threads are implemented and managed by the OS.

- Kernel level threads are implemented using system calls and Kernel level threads are recognized by the OS.
- Kernel-level threads are slower to create and manage compared to user-level threads.
- Context switching in a kernel-level thread is slower.
- Even if one kernel-level thread performs a blocking operation, it does not affect other threads. Eg: Window Solaris.

Advantages of Threading

- Threads improve the overall performance of a program.
- Threads increases the responsiveness of the program
- Context Switching time in threads is faster.
- Threads share the same memory and resources within a process.
- Communication is faster in threads.
- Threads provide concurrency within a process.
- Enhanced throughput of the system.
- Since different threads can run parallelly, threading enables the utilization of the multiprocessor architecture to a greater extent and increases efficiency.

Issues with Threading

There are a number of issues that arise with threading. Some of them are mentioned below:

- **The semantics of fork() and exec() system calls:** The fork() call is used to create a duplicate child process. During a fork() call the issue that arises is whether the whole process should be duplicated or just the thread which made the fork() call should be duplicated. The exec() call replaces the whole process that called it including all the threads in the process with a new program.

- **Thread cancellation:** The termination of a thread before its completion is called thread cancellation and the terminated thread is termed as target thread. Thread cancellation is of two types:
    1. Asynchronous Cancellation: In asynchronous cancellation, one thread immediately terminates the target thread.
    2. Deferred Cancellation: In deferred cancellation, the target thread periodically checks if it should be terminated.
- **Signal handling:** In UNIX systems, a signal is used to notify a process that a particular event has happened. Based on the source of the signal, signal handling can be categorized as:
    1. **Asynchronous Signal:** The signal which is generated outside the process which receives it.
    2. **Synchronous Signal:** The signal which is generated and delivered in the same process.

Inter Process Communication

Inter process communication (IPC) is used for exchanging data between multiple threads in one or more processes or programs. The Processes may be running on single or multiple computers connected by a network. The full form of IPC is Inter-process communication.

It is a set of programming interface which allow a programmer to coordinate activities among various program processes which can run concurrently in an operating system. This allows a specific program to handle many user requests at the same time.
Since every single user request may result in multiple processes running in the operating system, the process may require to communicate with each other. Each IPC protocol approach has its own advantage and limitation, so it is not unusual for a single program to use all of the IPC methods.

Approaches for Inter-Process Communication

Pipes

Pipe is widely used for communication between two related processes. This is a half-duplex method, so the first process communicates with the second process. However, in order to achieve a full-duplex, another pipe is needed.

Message Passing:

It is a mechanism for a process to communicate and synchronize. Using message passing, the process communicates with each other without resorting to shared variables.

IPC mechanism provides two operations:

- Send (message)- message size fixed or variable
- Received (message)

Message Queues:

A message queue is a linked list of messages stored within the kernel. It is identified by a message queue identifier. This method offers communication between single or multiple processes with full-duplex capacity.

Direct Communication:

In this type of inter-process communication process, should name each other explicitly. In this method, a link is established between one pair of communicating processes, and between each pair, only one link exists.

Indirect Communication:

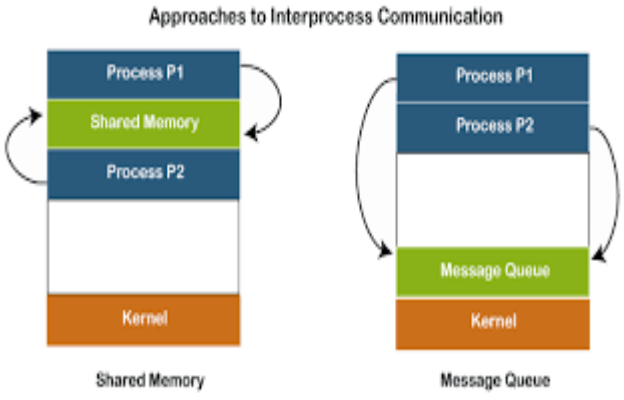
Indirect communication establishes like only when processes share a common mailbox each pair of processes sharing several communication links. A link can communicate with many processes. The link may be bi-directional or unidirectional.

Shared Memory:

Shared memory is a memory shared between two or more processes that are established using shared memory between all the processes. This type of memory requires to protected from each other by synchronizing access across all the processes.

FIFO:

Communication between two unrelated processes. It is a full-duplex method, which means that the first process can communicate with the second process, and the opposite can also happen.



Terms Used in IPC

The following are a few important terms used in IPC:

Semaphores: A semaphore is a signaling mechanism technique. This OS method either allows or disallows access to the resource, which depends on how it is set up.

Signals: It is a method to communicate between multiple processes by way of signaling. The source process will send a signal which is recognized by number, and the destination process will handle it.

Multiple-Processor Scheduling in Operating System

In multiple-processor scheduling multiple CPU's are available and hence Load Sharing becomes possible. However multiple processor scheduling is more complex as compared to single processor scheduling. In multiple processor scheduling there are cases when the processors are identical i.e. HOMOGENEOUS, in terms of their functionality, we can use any processor available to run any process in the queue.

Approaches to Multiple-Processor Scheduling –

One approach is when all the scheduling decisions and I/O processing are handled by a single processor which is called the Master Server and the other processors executes only the user code. This is simple and reduces the need of data sharing. This entire scenario is called Asymmetric Multiprocessing.
A second approach uses Symmetric Multiprocessing where each processor is self scheduling. All processes may be in a common ready queue or each processor may have its own private queue for ready processes. The scheduling proceeds further by having the scheduler for each processor examine the ready queue and select a process to execute.