PROCESS MANAGEMENT IN OPERATING SYSTEMS

PROCESS DEFINITION

Process is the execution of a program that performs the actions specified in that program. It can be defined as an execution unit where a program runs. The OS helps you to create, schedule, and terminates the processes which is used by CPU. A process created by the main process is called a child process.

Process operations can be easily controlled with the help of PCB (Process Control Block). You can consider it as the brain of the process, which contains all the crucial information related to processing like process id, priority, state, CPU registers, etc.

PROCESS MANAGEMENT

Process management involves various tasks like creation, scheduling, termination of processes, and a dead lock. Process is a program that is under execution, which is an important part of modern-day operating systems. The OS must allocate resources that enable processes to share and exchange information.

PROCESS ARCHITECTURE

Process architecture is the structural make-up of general process systems. It applies to fields such as technology, software and business processes. The architecture diagram of a process is as shown in figure 1.

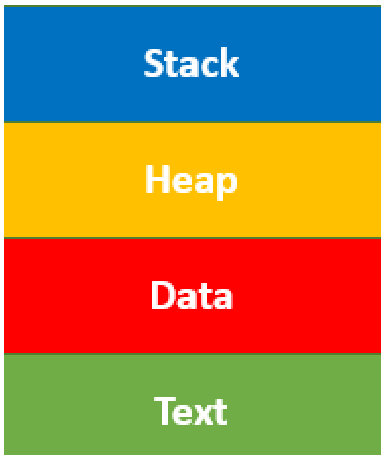


Figure 1: Process architecture Image

1. **Stack**: The Stack stores temporary data like function parameters, returns addresses, and local variables.
2. **Heap**: Allocates memory, which may be processed during its run time.
3. **Data**: It contains the variable.
4. **Text**: Text Section includes the current activity, which is represented by the value of the Program Counter.

PROCESS STATES

A process state is a condition of the process at a specific instant of time. It also defines the current position of the process. Figure 2 below illustrates process states

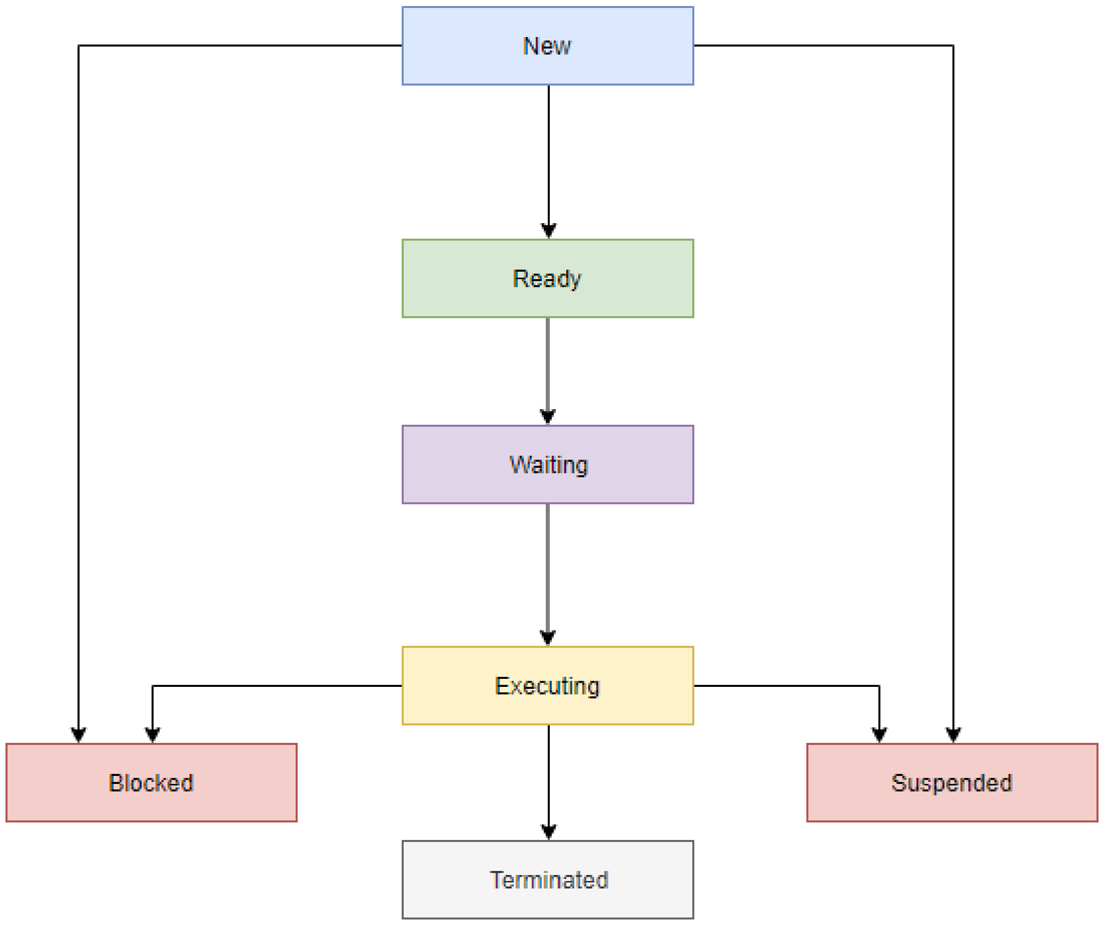


Figure 2: Process States Diagram

1. **New**
   The new process is created when a specific program calls from secondary memory/ hard disk to primary memory/ RAM a
2. **Ready**
   In a ready state, the process should be loaded into the primary memory, which is ready for execution.
3. **Waiting**
   The process is waiting for the allocation of CPU time and other resources for execution.
4. **Executing**
   The process is an execution state.
5. **Blocked**
   It is a time interval when a process is waiting for an event like I/O operations to complete.
6. **Suspended**
   Suspended state defines the time when a process is ready for execution but has not been placed in the ready queue by OS.

7. **Terminated**
   Terminated state specifies the time when a process is terminated

NOTE

After completing every step, all the resources are used by a process, and memory becomes free.

PROCESS CONTROL BLOCKS (PCB)

PCB stands for Process Control Block. It is a data structure that is maintained by the Operating System for every process. The PCB should be identified by an integer Process ID (PID). It helps you to store all the information required to keep track of all the running processes.

Every process is represented in the operating system by a process control block which is also called a *task control block.*

It is accountable for storing the contents of processor registers. These are saved when the process moves from the running state and then returns back to it. The information is quickly updated in the PCB by the OS as soon as the process makes the state transition.

Components of PCB

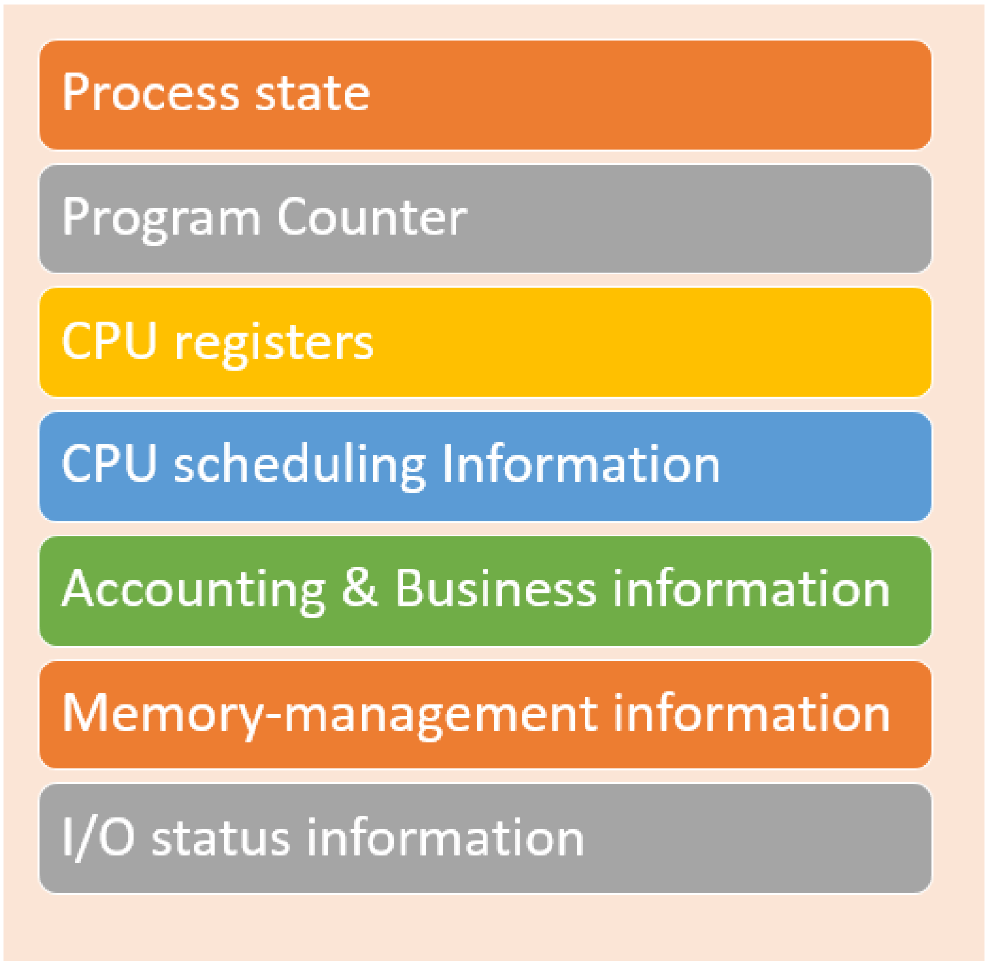

Process state
Program Counter
CPU registers
CPU scheduling Information
Accounting & Business information
Memory-management information
I/O status information

Figure 3: PCB Components

1. Process state
   A process can be new, ready, running, waiting, etc.
2. Program counter
   The program counter lets you know the address of the next instruction, which should be executed for that process.
3. CPU registers
   This component includes accumulators, index and general-purpose registers, and information of condition code.
4. CPU scheduling information
   This component includes a process priority, pointers for scheduling queues, and various other scheduling parameters.
5. Accounting and business information
   It includes the amount of CPU and time utilities like real time used, job or process numbers, etc.
6. Memory-management information
   This information includes the value of the base and limit registers, the page, or segment tables. This depends on the memory system, which is used by the operating system.
7. I/O status information
   This block includes a list of open files, the list of I/O devices that are allocated to the process, etc.

PROCESS SCHEDULING

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

Process scheduling is an essential part of a multiprogramming operating systems. There are many scheduling queues that are used to handle processes. When the processes enter the system, they are put into the job queue. The processes that are ready to execute in the main memory are kept in the ready queue. The processes that are waiting for the I/O device are kept in the device queue.

Three types of process schedulers.

Long Term Scheduler

The job scheduler or long term scheduler selects processes from the storage pool and loads them into memory for execution. The job scheduler must select a careful mixture of I/O bound and CPU bound processes to yield optimum system throughput. If it selects too many CPU bound processes then the I/O devices are idle and if it selects too many I/O bound processes then the processor has nothing to do.

Short Term Scheduler

The short term scheduler selects one of the processes from the ready queue and schedules them for execution. The short term scheduler executes much more frequently than the long term scheduler as a process may execute only for a few milliseconds.

Medium Term Scheduler

The medium term scheduler swaps out a process from main memory. It can again swap in the process later from the point it stopped executing. This is helpful in reducing the degree of multiprogramming. Swapping is also useful to improve the mix of I/O bound and CPU bound processes in the memory.

PROCESSES SYNCHRONIZATION

Processes Synchronization is the way by which processes that share the same memory space are managed in an operating system. It helps maintain the consistency of data by using variables or hardware so that only one process can make changes to the shared memory at a time.

How Process Synchronization in OS Works

Let us take a look at why exactly we need Process Synchronization. For example, if a ***process 1*** is trying to read the data present in a memory location while another ***process 2*** is trying to change the data present at the same location, there is a high chance that the data read by the ***process 1*** will be incorrect.

Elements/sections of a program:

1. ***Entry Section***: The entry Section decides the entry of a process.
2. ***Critical Section***: Critical section allows and makes sure that only one process is modifying the shared data.
3. ***Exit Section:*** The entry of other processes in the shared data after the execution of one process is handled by the Exit section.
4. ***Remainder Section:*** The remaining part of the code which is not categorized as above is contained in the Remainder section.

Critical Section Problem

A part of code that can only be accessed by a single process at any moment is known as a critical section. This means that when a lot of programs want to access and change a single shared data, only one process will be allowed to change at any given moment. The other processes have to wait until the data is free to be used.

The wait () function mainly handles the entry to the critical section, while the signal() function handles the exit from the critical section.

If we remove the critical section, we cannot guarantee the consistency of the end outcome after all the processes finish executing simultaneously.

Requirements of Synchronization

They are also referred to as the conditions that are necessary for a solution to Critical Section Problem.

1. **Mutual exclusion**: If a process is running in the critical section, no other process should be allowed to run in that section at that time.
2. **Progress**: If no process is still in the critical section and other processes are waiting outside the critical section to execute, then any one of the threads must be permitted to enter the critical section. The decision of which process will enter the critical section will be taken by only those processes that are not executing in the remaining section.
3. **No starvation:** Starvation means a process keeps waiting forever to access the critical section but never gets a chance. No starvation is also known as Bounded Waiting.
   - A process should not wait forever to enter inside the critical section.
   - When a process submits a request to access its critical section, there should be a limit or bound, which is the number of other processes that are allowed to access the critical section before it.
   - After this bound is reached, this process should be allowed to access the critical section.

<u>Solutions to the critical section problem</u>

1. **Peterson's solution**

   The solution is based on the idea that when a process is executing in a critical section, then the other process executes the rest of the code and vice-versa is also possible, i.e., this solution makes sure that only one process executes the critical section at any point in time.

2. **Synchronization Hardware**

   Hardware can occasionally assist in the solving of critical section issues. Some operating systems provide a lock feature.

   When a process enters a critical section, it is given a lock, which the process must release before the process can exit the critical section. As a result, additional processes are unable to access a critical section if anyone process is already using the section. The lock can have either of the two values, 0 or 1.

3. **Mutex Locks**

   Implementation of Synchronization hardware is not an easy method, which is why Mutex Locks were introduced.

   Mutex is a locking mechanism used to synchronize access to a resource in the critical section. In this method, we use a LOCK over the critical section. The LOCK is set when a process enters from the entry section, and it gets unset when the process exits from the exit section.

4. **Semaphores**

   A semaphore is a signaling mechanism, and a process can signal a process that is waiting on a semaphore. This differs from a mutex in that the mutex can only be notified by the process that sets the shared lock. Semaphores make use of the wait() and signal() functions for synchronization among the processes.

There are two kinds of semaphores namely

a) **Binary Semaphores**

   Binary Semaphores can only have one of two values: 0 or 1. Because of their capacity to ensure mutual exclusion, they are also known as mutex locks.

   A single binary semaphore is shared between multiple processes.

   When the semaphore is set to 1, it means some process is working on its critical section, and other processes need to wait, and if the semaphore is set to 0, that means any process can enter the critical section.

b) **Counting Semaphores**

   Counting Semaphores can have any value and are not limited to a certain area. They can be used to restrict access to a resource that has a concurrent access limit.

   Initially, the counting semaphores are set to the maximum amount of processes that can access the resource at a time. Hence, the counting semaphore indicates that a process can access the resource if it has a value greater than 0. If it is set to 0, no other process can access the resource.

Hence,

- When a process wants to use that resource, it first checks to see if the value of the counting semaphore is more than zero.
- If yes, the process can then proceed to access the resource, which involves reducing the value of the counting semaphore by one.
- When the process completes its critical section code, it can increase the value of the counting semaphore, making way for some other process to access it.

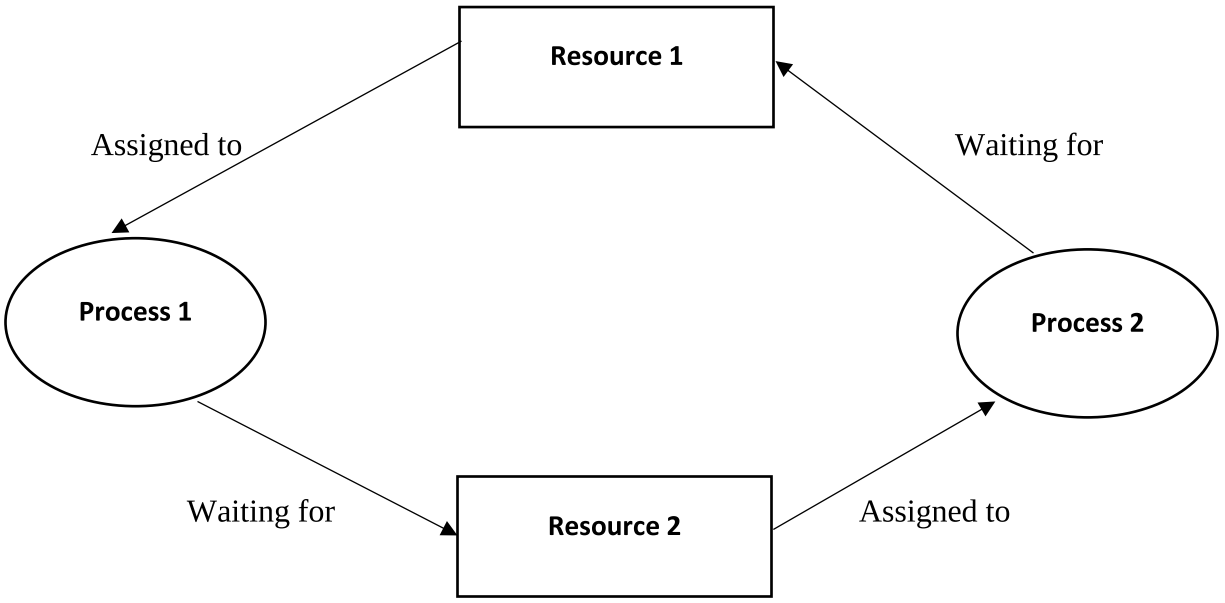
CONCLUSION

- Synchronization is the effort of executing processes such that no two processes have access to the same shared data.
- Four elements of program/data are:
  1. Entry section
  2. Critical section
  3. Exit section
  4. Reminder section
- The critical section is a portion of code that a single process can access at a specified moment in time.
- Three essential rules that any critical section solution must follow are as follows:
  1. Mutual Exclusion
  2. Progress
  3. No Starvation (Bounded waiting)
- Solutions to critical section problem are:
  1. Peterson's solution
  2. Synchronization hardware
  3. Mutex Locks
  4. Semaphore

DEADLOCK

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

Consider an example when two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they are in front of each other. A similar situation occurs in operating systems when there are two or more processes that hold some resources and wait for resources held by other(s). For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



Deadlock conditions

Deadlock can arise if the following four conditions hold simultaneously (Necessary Conditions)

1. **Mutual Exclusion:** Two or more resources are non-shareable (Only one process can use at a time)
2. **Hold and Wait:** A process is holding at least one resource and waiting for resources.
3. **No Preemption:** A resource cannot be taken from a process unless the process releases the resource.
4. **Circular Wait:** A set of processes are waiting for each other in circular form.

Methods for handling deadlock.

There are three ways to handle deadlock namely

Deadlock prevention or avoidance

The idea is to not let the system into a deadlock state.
One can zoom into each category individually, Prevention is done by negating one of above mentioned necessary conditions for deadlock.
Avoidance is kind of futuristic in nature. By using strategy of “Avoidance”, we have to make an assumption. We need to ensure that all information about resources which process will need are known to us prior to execution of the process.

Deadlock detection and recovery

Let deadlock occur, then do preemption to handle it once occurred.

Ignore the problem altogether

If deadlock is very rare, then let it happen and reboot the system.

Review Question

Describe Inter-process communication in Operating Systems clearly stating the reasons why it is needed