Concept of Concurrency

Concurrency is the execution of several instruction sequences at the same time**.** In an operating system, this happens when there are several process threads running in parallel. These threads may communicate with each other through either shared memory or message passing.

Concurrency results in sharing of resources result in problems like: - deadlocks and resources starvation.

It helps in techniques like coordinating execution of processes, memory allocation and execution scheduling for maximizing throughput.

Problems in Concurrency: -

- sharing global resources safely is difficult;
- optimal allocation of resources is difficult;
- locating programming errors can be difficult, because the contexts in which errors occur cannot always be reproduced easily

Cooperating Process

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

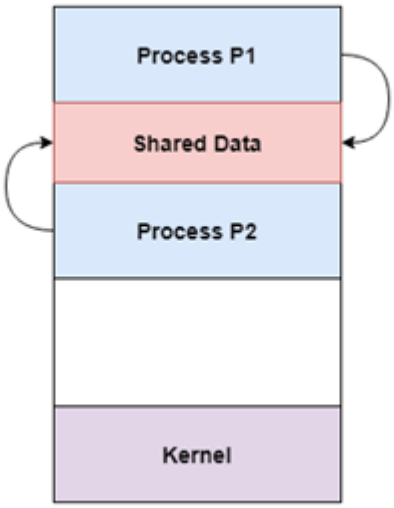
Cooperating processes can coordinate with each other using shared data or messages. Details about these are given as follows:

- **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:

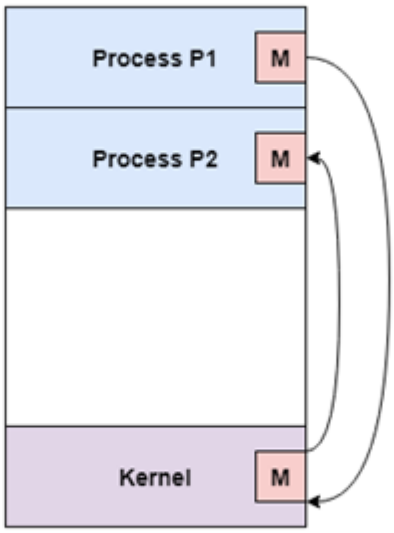


In the above diagram, Process P1 and P2 can cooperate with each other using shared data such as memory, variables, files, databases etc.

- **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.

The Critical Section Problem

The critical section is a code segment where the shared variables can be accessed. An atomic action is required in a critical section i.e. only one process can execute in its critical section at a time. All the other processes have to wait to execute in their critical sections.

A diagram that demonstrates the critical section is as follows –

Semaphores

Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization.

The definitions of wait and signal are as follows −

- **Wait**

  The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

```
wait(S)
{
while (S<=0);


S--;
}
```

- **Signal**

  The signal operation increments the value of its argument S.

```
signal(S)
{
S++;
}
```

Types of Semaphores

There are two main types of semaphores i.e. counting semaphores and binary semaphores. Details about these are given as follows:

- **Counting Semaphores**

These are integer value semaphores and have an unrestricted value domain. These semaphores are used to coordinate the resource access, where the semaphore count is the number of available resources. If the resources are added, semaphore count automatically incremented and if the resources are removed, the count is decremented.

- **Binary Semaphores**

  The binary semaphores are like counting semaphores but their value is restricted to 0 and 1. The wait operation only works when the semaphore is 1 and the signal operation succeeds when semaphore is 0. It is sometimes easier to implement binary semaphores than counting semaphores.

Advantages of Semaphores

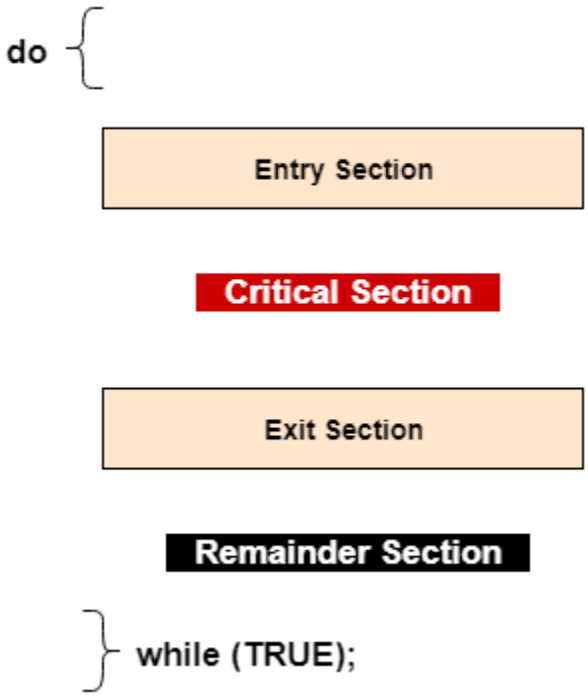
Some of the advantages of semaphores are as follows:

- Semaphores allow only one process into the critical section. They follow the mutual exclusion principle strictly and are much more efficient than some other methods of synchronization.
- There is no resource wastage because of busy waiting in semaphores as processor time is not wasted unnecessarily to check if a condition is fulfilled to allow a process to access the critical section.
- Semaphores are implemented in the machine independent code of the microkernel. So they are machine independent.

Disadvantages of Semaphores

Some of the disadvantages of semaphores are as follows –

- Semaphores are complicated so the wait and signal operations must be implemented in the correct order to prevent deadlocks.
- Semaphores are impractical for last scale use as their use leads to loss of modularity. This happens because the wait and signal operations prevent the creation of a structured layout for the system.
- Semaphores may lead to a priority inversion where low priority processes may access the critical section first and high priority processes later.



In the above diagram, the entry section handles the entry into the critical section. It acquires the resources needed

for execution by the process. The exit section handles the exit from the critical section. It releases the resources and also informs the other processes that the critical section is free.

Solution to the Critical Section Problem

The critical section problem needs a solution to synchronize the different processes. The solution to the critical section problem must satisfy the following conditions −

- **Mutual Exclusion**

  Mutual exclusion implies that only one process can be inside the critical section at any time. If any other processes require the critical section, they must wait until it is free.

- **Progress**

  Progress means that if a process is not using the critical section, then it should not stop any other process from accessing it. In other words, any process can enter a critical section if it is free.

- **Bounded Waiting**

  Bounded waiting means that each process must have a limited waiting time. It should not wait endlessly to access the critical section.

Classical Problems

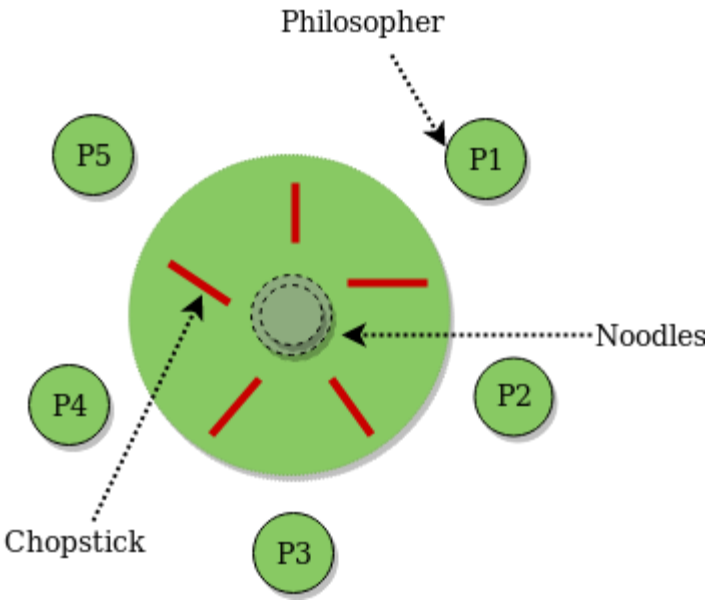
These problems are used for testing nearly every newly proposed synchronization scheme. The following

problems of synchronization are considered as classical problems:

1. Producer-Consumer Problem,
2. Dining-Philosopher's Problem,
3. Readers and Writers Problem,

1. **Producer-Consumer Problem:**
   Producer-Consumer problem is also called producer consumer problem. This problem is generalized in terms of the Producer-Consumer problem. Solution to this problem is, creating two counting semaphores "full" and "empty" to keep track of the current number of full and empty buffers respectively. Producers produce a product and consumers consume the product, but both use of one of the containers each time.

2. **Dining-Philosopher's Problem:**
   The Dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pickup the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both. This problem involves the allocation of limited resources to a group of processes in a deadlock-free and starvation-free manner.



3. **<u>Readers and Writers Problem</u>:**
   Suppose that a database is to be shared among several concurrent processes. Some of these processes may want only to read the database, whereas others may want to update (that is, to read and write) the database. We distinguish between these two types of processes by referring to the former as readers and to the latter as writers. Precisely in OS we call this situation as the readers-writers' problem. Problem parameters:
   - One set of data is shared among a number of processes.
   - Once a writer is ready, it performs its write. Only one writer may write at a time.
   - If a process is writing, no other process can read it.
   - If at least one reader is reading, no other process can write.
   - Readers may not write and only read.

Inter process communication (IPC)

Inter process communication 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.

Approaches for Inter-Process Communication

Here, are few important methods 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.

Why IPC?

Here, are the reasons for using the inter process communication protocol for information sharing:

- It helps to speedup modularity
- Computational
- Privilege separation
- Convenience
- Helps operating system to communicate with each other and synchronize their actions.