Interprocess Communication and Synchronization

Prof. V. V. Subrahmanyam
Director, SOCIS, IGNOU

Processes

Processes that are executing concurrently may be either independent processes or cooperating processes.

- **Independent Process** : Execution of one process does not affects the execution of other processes.
- **Cooperative Process** : Execution of one process affects the execution of other processes.

Reasons for Process Cooperation

- Information Sharing
- Computation Speedup
- Modularity
- Convenience

Why Interprocess Communication?

- Cooperating processes frequently need to communicate with each other to ensure tasks are correctly done.
- Cooperating processes need an inter-process communication (IPC) mechanism that will allow them to exchange data and information.

Contd…

- Sometimes, the correctness of the executing results depend on the executing sequence of cooperating processes. At this scenario, we need to enforce **synchronization** to make sure we can get the correct results.

Contd…

- Sometimes the execution of one process may require the result of another process. At this scenario, we need a **communication** mechanism for those processes to talk to each other.

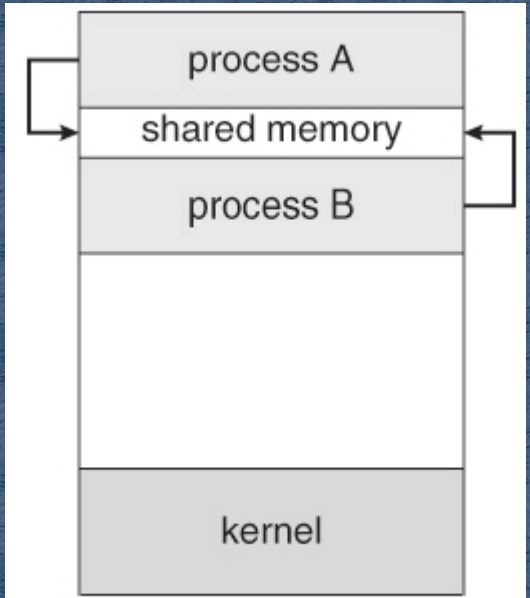
2 Approaches for IPC

- Shared Memory
- Message Passing

Shared Memory

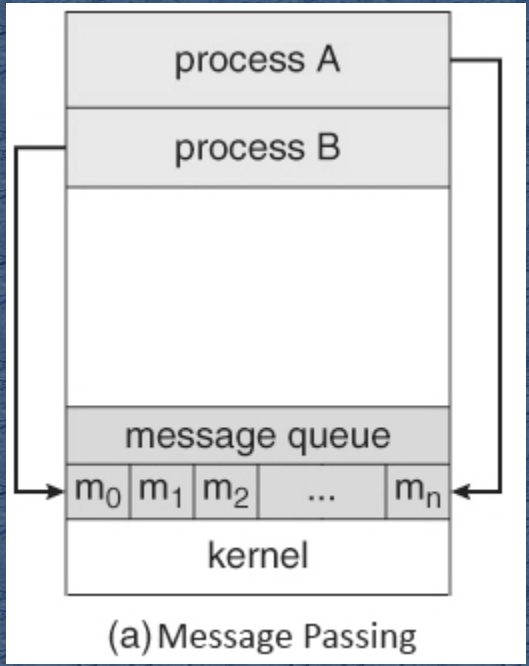
- Cooperating processes may communicate by **sharing** the same piece of memory.
- One process can write some data to a piece of shared memory, and another process can read from the same piece of memory directly.
- When doing so, you need to be careful. Otherwise, the executing result may not be the same as your expectation. To understand why, we study the concept of race condition first.

Shared Memory



(b) Shared Memory

Message Passing



(a) Message Passing

Race Condition

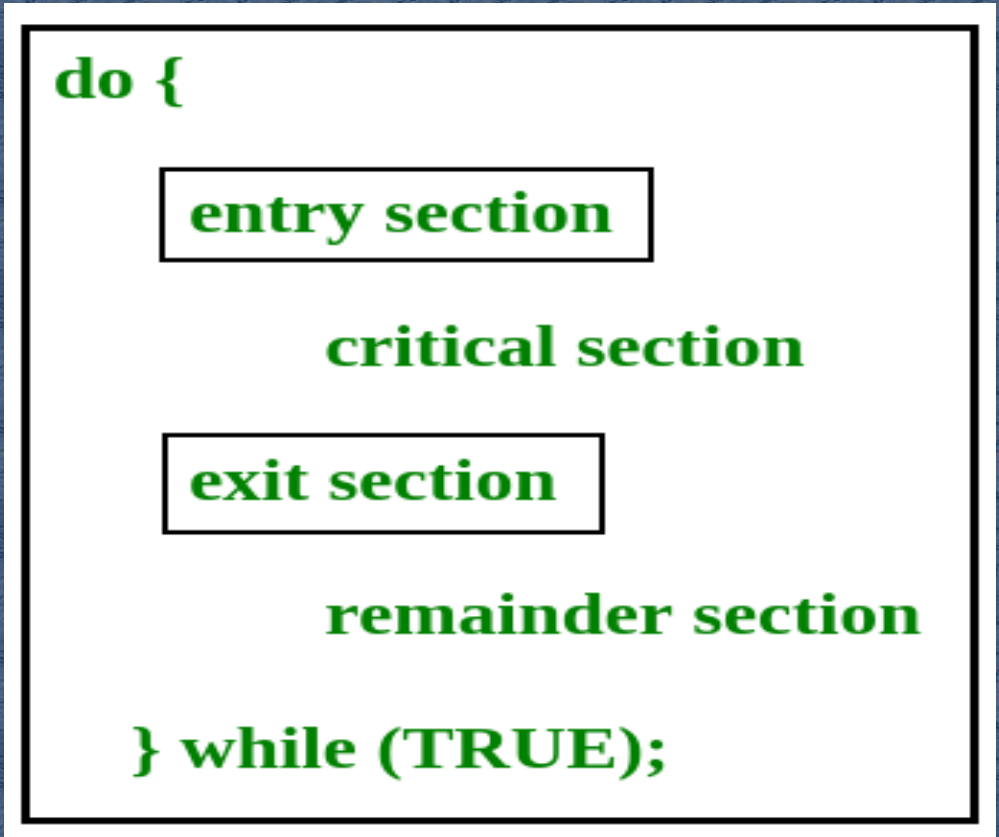
- When more than one processes are executing the same code or accessing the same memory or any shared variable in that condition there is a possibility that the output or the value of the shared variable is wrong so for that all the processes doing race to say that my output is correct this condition known as race condition.

Contd…

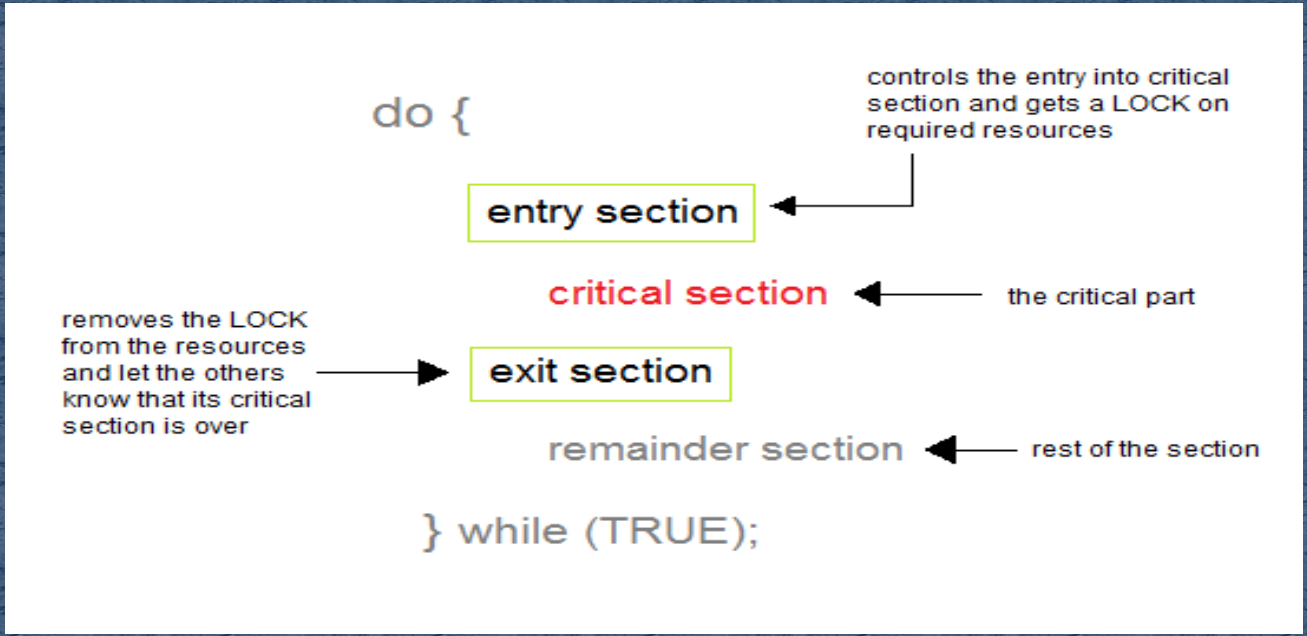
- Several processes access and process the manipulations over the same data concurrently, then the outcome depends on the particular order in which the access takes place.

Critical Section Problem

- To solve race condition problem, we define a problem called critical section problem.
- Critical section is the part of program where shared memory is accessed.
- Find solutions that arrange the cooperating processes properly, so that no two processes were ever in their critical sections at the same time.


do {
entry section
critical section
exit section
remainder section
} while (TRUE);

Critical Section



Solution Criteria to Critical Section Problem

- A solution to critical section problem must satisfy the following 3 conditions:
  - **Mutual Exclusion:** No two processes can enter their critical sections at the same time
  - **Progress:** If no process is executing its critical section, then one of the waiting processes can enter its critical section.
  - **Bounded Waiting:** No infinite wait for a process.

Process Synchronization

- Process Synchronization means sharing system resources by processes in a such a way that, concurrent access to shared data is handled thereby minimizing the chance of inconsistent data.
- Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes.
- Process Synchronization was introduced to handle problems that arose while multiple process executions.

Synchronization Hardware

- Many systems provide hardware support for critical section code. The critical section problem could be solved easily in a single-processor environment if we could disallow interrupts to occur while a shared variable or resource is being modified.
- In this manner, we could be sure that the current sequence of instructions would be allowed to execute in order without pre-emption. Unfortunately, this solution is not feasible in a multiprocessor environment.

Contd…

- Disabling interrupt on a multiprocessor environment can be time consuming as the message is passed to all the processors.
- This message transmission lag, delays entry of threads into critical section and the system efficiency decreases.

Test And Set (H/W Solution)

- TestAndSet is a hardware solution to the synchronization problem. In TestAndSet, we have a shared lock variable which can take either of the two values, 0 or 1.
- 0 Unlock
- 1 Lock

Before entering into the critical section, a process inquires about the lock. If it is locked, it keeps on waiting till it becomes free and if it is not locked, it takes the lock and executes the critical section.

Contd...

- In Test And Set, Mutual exclusion and progress are preserved but bounded waiting cannot be preserved.

```
int TestAndSet(int &lock)
{
 int initial = lock;
lock = 1;
return initial;
}
void enter_CS(X)
{
while test-and-set(X) ;
}
void leave_CS(X)
{
X = 0;
 }
```

Compare and Swap (H/w Solution)

- **Compare and swap** is a technique used when designing concurrent algorithms.
- Basically, compare and swap compares an expected value to the concrete value of a variable, and if the concrete value of the variable is equals to the expected value, swaps the value of the variable for a new variable.

Peterson's Solution

- Peterson's Solution is a classical software based solution to the critical section problem.
- In Peterson's solution, we have two shared variables:
  - boolean flag[i] :Initialized to FALSE, initially no one is interested in entering the critical section
  - int turn : The process whose turn is to enter the critical section.

```
do {

    flag[i] = TRUE ;
    turn = j ;
    while (flag[j]  &&  turn == j) ;

        critial section

    flag[i] = FALSE ;

        remainder section

} while (TRUE) ;
```

Peterson's Solution preserves all three conditions :

- Mutual Exclusion is assured as only one process can access the critical section at any time.
- Progress is also assured, as a process outside the critical section does not block other processes from entering the critical section.
- Bounded Waiting is preserved as every process gets a fair chance.

Disadvantages

Disadvantages of Peterson's Solution:

- It involves Busy waiting
- It is limited to 2 processes

Bakery Algorithm

- The **Bakery algorithm** is one of the simplest known solutions to the mutual exclusion problem for the general case of N process.
- Bakery Algorithm is a critical section solution for **N** processes. The algorithm preserves the first come first serve property.
- Before entering its critical section, the process receives a number. Holder of the smallest number enters the critical section.

Contd…

- If processes Pi and Pj receive the same number,

  if i < j

  Pi is served first;

  else

  Pj is served first.

- The numbering scheme always generates numbers in increasing order of enumeration; i.e., 1, 2, 3, 3, 3, 3, 4, 5, …

Mutex Locks

- In this approach, in the entry section of code, a LOCK is acquired over the critical resources modified and used inside critical section, and in the exit section that LOCK is released.
- As the resource is locked while a process executes its critical section hence no other process can access it.

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 alternate names of wait are P(), Down()
- The alternate names of Signal are V(), Up(), Post(), Release()

Wait operation

- 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 Operation

- The signal operation increments the value of its argument S.

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

Types of Semaphores

- There are two main types of semaphores:
  - Counting Semaphores
  - Binary Semaphores

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

- 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

- 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.

Monitors in Process Synchronization

- The monitor is one of the ways to achieve Process synchronization.
- The monitor is supported by programming languages to achieve mutual exclusion between processes. For example Java Synchronized methods. Java provides wait() and notify() constructs.

Contd…

- It is the collection of condition variables and procedures combined together in a special kind of module or a package.
- The processes running outside the monitor can't access the internal variable of the monitor but can call procedures of the monitor.
- Only one process at a time can execute code inside monitors.

```
Monitor Demo //Name of Monitor
{
variables;
condition variables;

procedure p1 {....}
prodecure p2 {....}


}
```

Syntax of Monitor

Advantages of Monitors

- Monitors have the advantage of making parallel programming easier and less error prone than using techniques such as semaphore.

Classical Problems of Synchronization

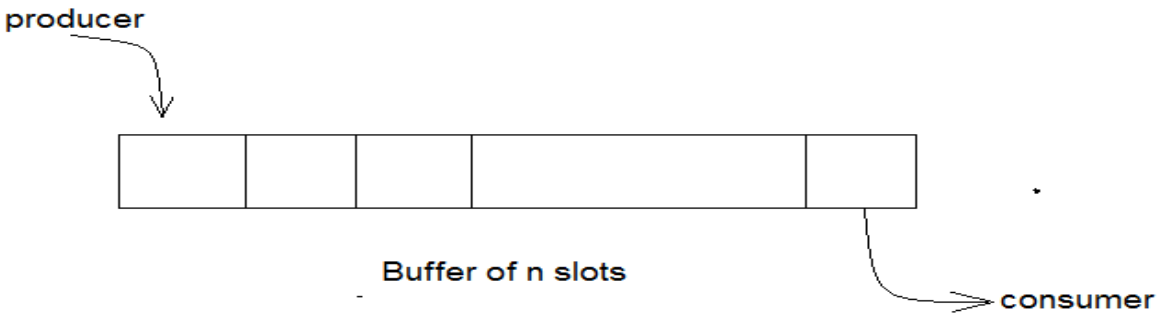
- Semaphore can be used in other synchronization problems besides Mutual Exclusion.
- Below are some of the classical problem depicting flaws of process synchronization in systems where cooperating processes are present.
  - Bounded Buffer (Producer-Consumer) Problem
  - Dining Philosophers Problem
  - The Readers Writers Problem

Bounded Buffer Problem

- This problem is generalized in terms of the **Producer Consumer problem**, where a **finite** buffer pool is used to exchange messages between producer and consumer processes.
- Because the buffer pool has a maximum size, this problem is often called the **Bounded buffer 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.

Problem Statement

- A producer tries to insert data into an empty slot of the buffer. A consumer tries to remove data from a filled slot in the buffer. As you might have guessed by now, those two processes won't produce the expected output if they are being executed concurrently.
- There needs to be a way to make the producer and consumer work in an independent manner.



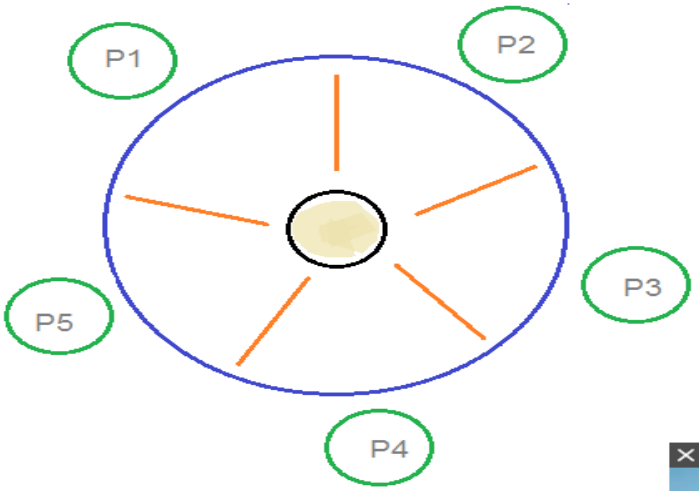
Bounded Buffer Problem

Dining Philosopher's Problem

- The dining philosopher's problem involves the allocation of limited resources to a group of processes in a deadlock-free and starvation-free manner.
- There are five philosophers sitting around a table, in which there are five chopsticks/forks kept beside them and a bowl of rice in the centre.
- When a philosopher wants to eat, he uses two chopsticks - one from their left and one from their right. When a philosopher wants to think, he keeps down both chopsticks at their original place.

Problem Statement

- At any instant, a philosopher is either eating or thinking. When a philosopher wants to eat, he uses two chopsticks - one from their left and one from their right. When a philosopher wants to think, he keeps down both chopsticks at their original place.



Dining Philosophers Problem

Readers-Writers Problem

- In this problem there are some processes (called **readers**) that only read the shared data, and never change it, and there are other processes (called **writers**) who may change the data in addition to reading, or instead of reading it.
- There are various type of readers-writers problem, most centered on relative priorities of readers and writers.

Problem Statement

- There is a shared resource which should be accessed by multiple processes.
- There are two types of processes in this context. They are **reader** and **writer**.
- Any number of **readers** can read from the shared resource simultaneously, but only one **writer** can write to the shared resource.
- When a **writer** is writing data to the resource, no other process can access the resource. A **writer** cannot write to the resource if there are non zero number of readers accessing the resource at that time.

Message Passing Method

- **Mechanism for processes to communicate and to synchronize their actions.**
- In this method, processes communicate with each other without using any kind of shared memory.
- **Message system – processes communicate with each other without resorting to shared variables.**
- **IPC facility provides two operations:**
  - **send(*message*) – message size fixed or variable**
  - **receive(*message*)**

Contd…

- If *P* and *Q* wish to communicate, they need to:
  - establish a *communication link* between them
  - exchange messages via send/receive
- Implementation of communication link
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)

Direct Communication

- **Processes must name each other explicitly:**
  - **send (*P, message*) – send a message to process P**
  - **receive(*Q, message*) – receive a message from process Q**

Contd…

- **Properties of a communication link**
  - **Links are established automatically.**
  - **A link is associated with exactly one pair of communicating processes.**
  - **Between each pair there exists exactly one link.**
  - **The link may be unidirectional, but is usually bi-directional.**

Indirect Communication

- **Messages are directed and received from mailboxes (also referred to as ports).**
  - **Each mailbox has a unique id.**
  - **Processes can communicate only if they share a mailbox.**

Contd…

- **Properties of communication link**
  - **Link established only if processes share a common mailbox**
  - **A link may be associated with many processes.**
  - **Each pair of processes may share several communication links.**
  - **Link may be unidirectional or bi-directional.**

Contd…

- **Operations**
  - **create a new mailbox**
  - **send and receive messages through mailbox**
  - **destroy a mailbox**
- **Primitives are defined as:**

  send(*A, message*) – send a message to mailbox A

  receive(*A, message*) – receive a message from mailbox A