UNIT- III

process Management & synchronization:

Concurrency:

→ Concurrency means that two or more process execution with in the same time frame and there is usually some sort of dependency between them.

→ Concurrency creates problems also because of access to shared data.

→ Concurrency refers to any form of interaction among the processes or threads.

→ Parallelism means that two are more process execution simultaneously.

→ Concurrency describe a problem, parallelism describe a solution. parallelism is one of way to implement concurrency, but it is not the only one. Another popular solution is interleaved processing.

Principle of Concurrency:

Concurrent access to shared data may result in data inconsistency. maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes.

→ Concurrency arises in the same way at different level of execution. The following are example of concurrency in different types of operating systems.

1. Concurrency in multiprogramming :

An interaction between multiple processes running on one cpu.

2. Concucy in multithreading :

An interaction between multiple threads running in one process.

3. Concurrency in multiprocessors :

An interaction between multiple cpus running multiple processes or threads.

4. Concurrency in multi computers :

An interaction between multiple computers running distributed processes or threads.

→ process synchronization is required in uni-processor System, multiprocessor system and network.

→ If more than one process exists in the system at the same time, then the processes are said to be concurrent.

→ Synchronization problems can occurs whenever two or more currcurrent processes use any shared resources.

Race condition :

→ Race condition occurs when two or more operations occurs in an undefined manner. when two or more processes are reading or writting some shared data and the final result depends on who precisely when, are called race condition.

→ There is a race condition if the outcome depends on the order of execution.

→ Race condition should be avoided because they can cause fine errors in the application and difficult to debug.

→ In banking system balance is shared variable.

After depositing the amount, it update by bank employee

$\boxed{\text{balance} = \text{balance} + \text{amount}}$ → task1

At the same time, some amount is transfer then it becomes $\boxed{\text{balance} = \text{balance} - \text{amount}}$ → task2

These two tasks are in a race to write variable balance.

→ Operating systems concerns, Design and management issues for concurrency are as follows.

1. Track of various processes is kept by operating system

2. Operating systems allocates and deallocate s/w and H/w resources to active process.

3. Operating system must protect user data and physical resources from un-authorized process.

4. process execution speed do not depends upon other process execution speed.

→ Critical section problem :

A critical section is a block of code that only one process at ~~ta~~ time can execute. So, when one process is in its critical section, no other process may be in its critical section.

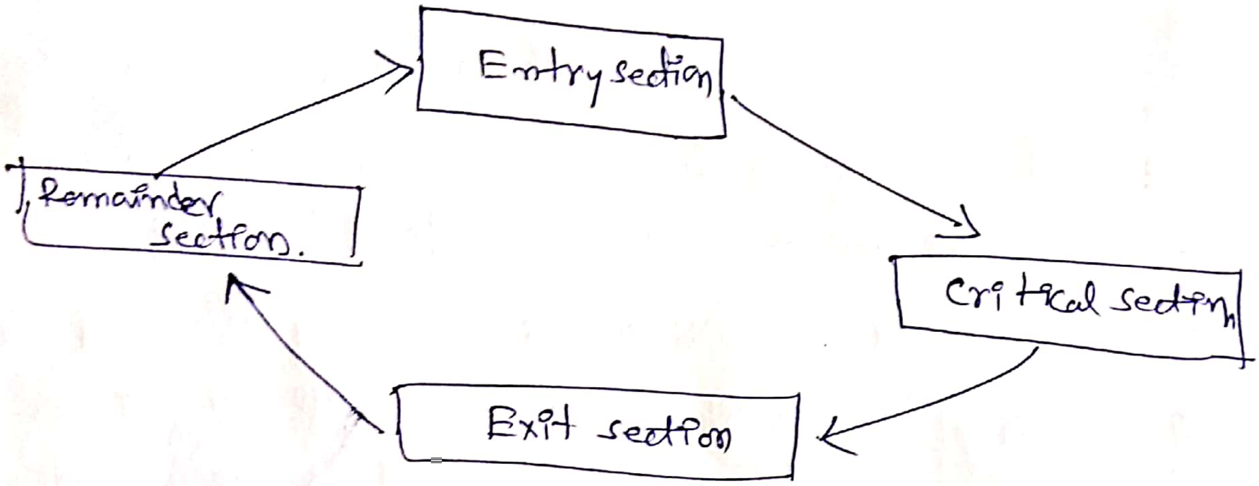
→ The critical section problems is to ensure that only one process at a time is allowed to be operating in its critical section.

→ Critical section means, process may change some common variable, writting files, updating memory location updating a process table etc. when process is accessing shared modifiable data, it is said to be in a critical section.

→ Each process takes permission from operating system to enter into the critical section.

Critical section structure of process is as follows.

1. Entry section → It is block of code execded in preparatio for entering critical section
2. Remainder section: Rest of the code is remainder section.
3. Exit section: The code exeated upon leaving the critical section.



→ Solution to critical section problem must satisfy the mutual exclusion, Lock conditions, mutex values.

Classical problems of synchronization:

The following are classical problems of Synchronisatn

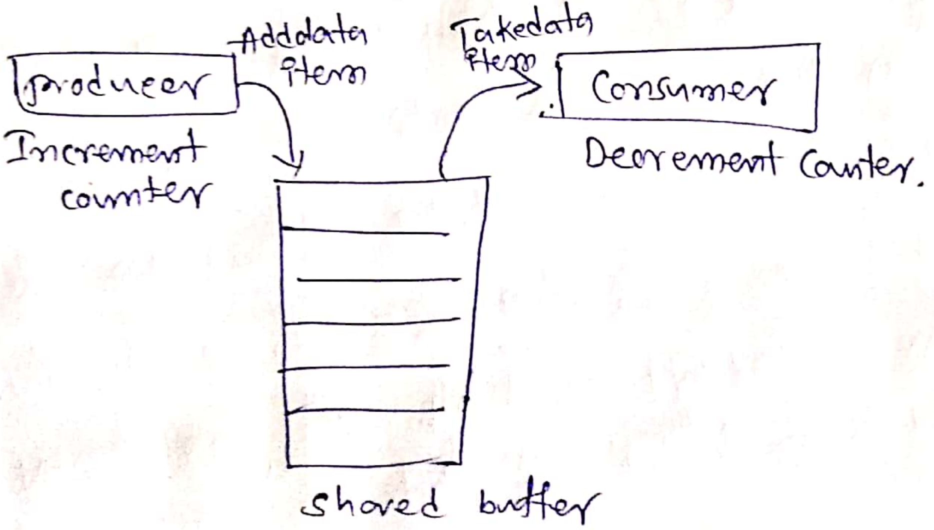
1. The producer-Consumer problem
2. Dinning philosopher problem.
3. Reader-writer problem.

1. The producer-Consumer problem.

The producer consumer problem classic problems of synchronization. producer process produce data item that consumer process consumes later.

→ Buffer is used between producer & consumer. Buffer size may be ~~fixed~~ fixed or variable.

→ The producer portion of the application generates data and stores it in a buffer, and consumer reads data from the buffer. The following figure shows producer-consumer problem.



→ The producer-consumer is also called bounded buffer problem.

→ The producer consumer problem shows the need for synchronization in system where many processes share resource.

→ The producer cannot deposit its if the buffer is full. similarly, a consumer cannot retrieve any data if the buffer is empty.

→ If the buffer is not full, a producer can deposit its data. The consumer should be allowed to retrieve a data item is buffer contains.

→ problem arises when the buffer is full and producer wants to add a new data item. Solution to this problem is that producer goes to sleep until consumer removes data item from the buffer.

→ Similar condition exists with the consumer. Consumer wants to consume data item from the buffer, but buffer is empty then go to sleep. It wake up when producer add some data items into the buffer.

→ In order to synchronize these processes, both ~~prod~~ ~~procedure and~~ producer and consumer are blocked on some condition. The producer is blocked when buffer is full, and consumer is blocked when the buffer is empty

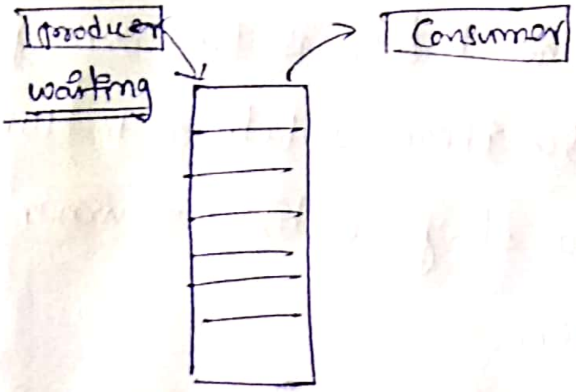
→ Example of producer consumer problem.

1. Printing word file : when user gives the print command for printing file, a word processor spool data to a buffer and data are subseguently consumed by the printer as it prints the document. printer stops printing when buffer is empty.

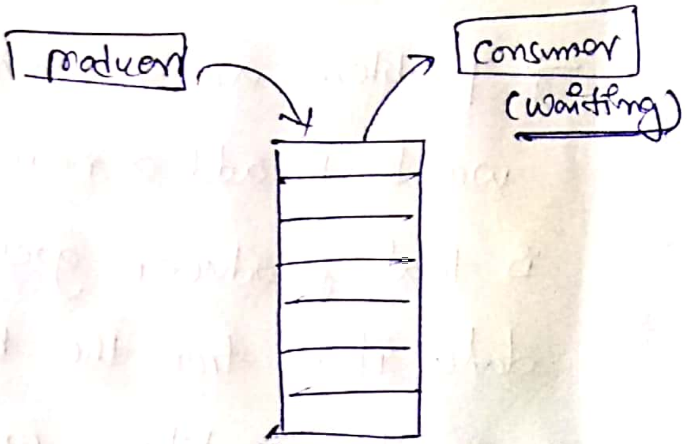
2. A compiler may produce assembly code, which is consumed by am assembler.

→ Buffer is ~~in~~ any of the following states.

1. Full buffer. 2. Empty Buffer.

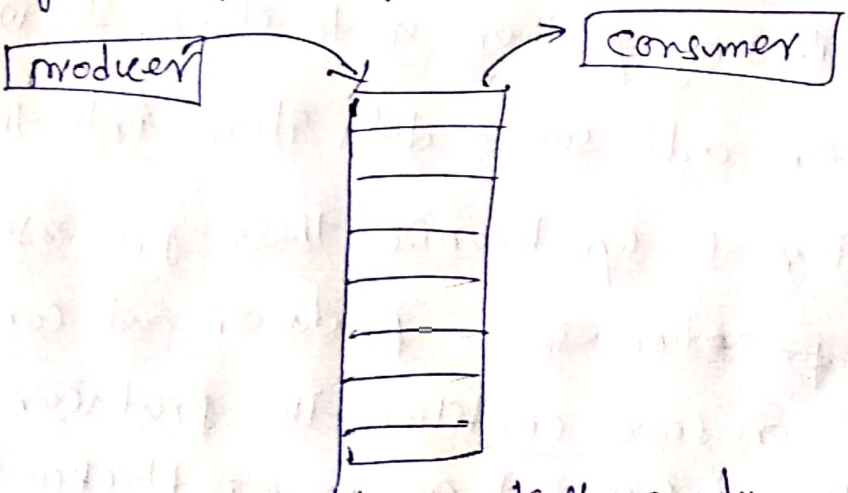


Buffer full.



Buffer empty

3. partially empty buffer



Buffer partially empty.

→ producer-Consumer problem is solved by using Semaphore, mutex and monitor. Mutual exclusion must be enforced on the buffer itself.

→ Each data item produce by producer to the shored buffer must be consumed exactly once by the consumer. Data item must be consumed in the same order (First in First out order). in which it is out into the buffer.

1.