Operating System
U3. Concurrency Control

Q-1- Write short note on interprocess communication — (5M)

Process executing in operating system

- Independent : cannot affect or be affected
- Co-operating : can affect or affected.

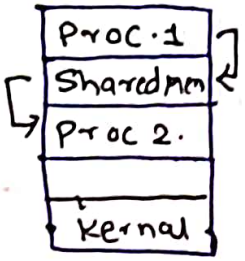
Any process that shares data with other processes is co-operating process

Reasons for IPC :

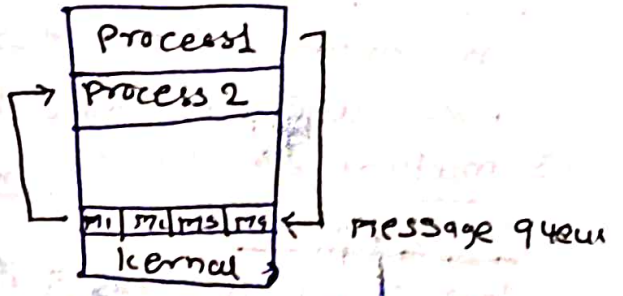
1. Information Sharing
2. Computational speedup
3. modularity
4. convinence

"IPC is a way by which multiple processes can communicate with eo/1

- **ways (working)**
  co-operating process need IPC to allow them exchange data & inform
- **model**
  1. shared memory
  2. message passing.

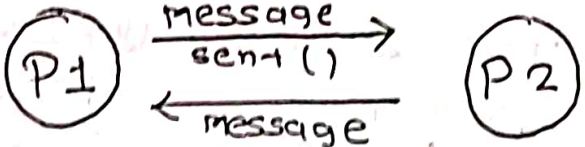


Shared memory



message passing

- **Shared memory :-**
- **Message_Passing :-**



- **Methods of IPC**
  1. Pipes
  2. Named pipes
  3. message queing
  4. semaphores
  5. shared memory
  6. sockets.

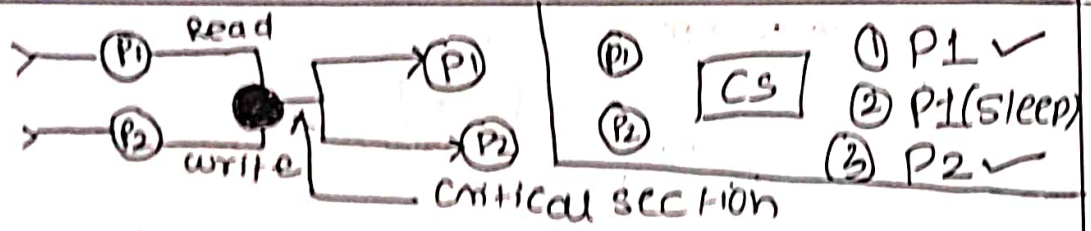
- **Mutual Exclusion**

when only one process should allow to execute in critical section called as mutual exclusion

Critical section:

The Part of the Program where the shared memory is accessed called as critical section

To avoid Race condition we use mutual exclusion.



To Prevent multiple Process to CS at a time.

- **Requirement for mutual exclusion**
  1. one Process at a time in CS
  2. In ncs process should not interface others
  3. No deadlock occurance
  4. every Process request entry for CS
  5. No Assumptions made about Proc or threads
  6. Process should remain limited time in CS.

- **Hardware Support**
  1. Interrept Disabling
  2. special machine instruction
  3. Compute & swap instruction
  4. Exchange instruction
  5. machine instruction

- **Race condition**

Race condition occurs when "multiple process or thread redds or write data at a time"

Explanation: Var = 10

P1 = Add the num by 1
P2 = Sub the num by 1

~~X = P1 (-) → sleep (-) → Var (~~
Var = X

~~Process Synchronisation~~

Critical section Problem

- Critical section is a part of segment of code where Process is changing data between shared resources.
- when one is executing no other is allowed in CS
- No two process allowed at a time.

Critical section Problem is to design a Protocal that the Processes can use to Co-operate

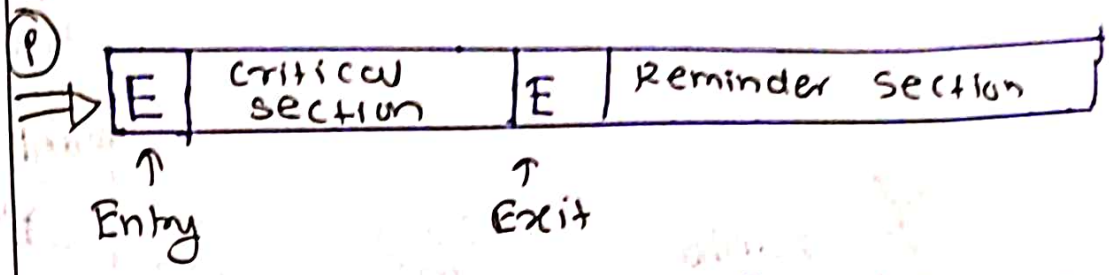
Rules: ① each Process request Permision to enter CS

② The section of code implimenting request called as entry section

③ Critical section may be followed by exit section

④ Remaing part is remaing section

```
do {
    entry section
        critical section
    exit section
        Reminder section
} while (1);
```



Solution to CS Problem: (requirements)

① Mutual Exclusion

② Progress: if no Process Executing in CS some wish to enter CS. Then which process wish to enter should allow. selection is taken by reminder section

③ Bounded waiting:
There exist a Bound or limit on the number of times a process enter a CS

Problem faced by Competing Process.

① Mutual exclusion

② Dead lock

③ Starvation.

• SEMAPHORES:

- It is technique to manage concurrent Processess by using simple integer value known as semaphore
- Non-negative variable.
- used to solve critical section prb
- Shared variable (Integer)
- Can be access by Processess through oprations: wait() & signal()

Wait() → 'P' → "TO test"

signal() → 'V' → "TO Increment"

wait() [entry]

```
P(semaphore S)
{
  while (S<=0);
  // no-operation
  S--;
}
```

if val of S<0 means some process is ready in the CS.

signal() [exit]

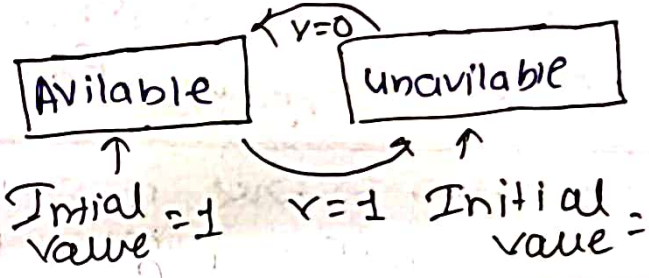
```
V(semaphore S)
{
  S++;
}
```

Signal that Signal to other proces i've used CS now you can use it.

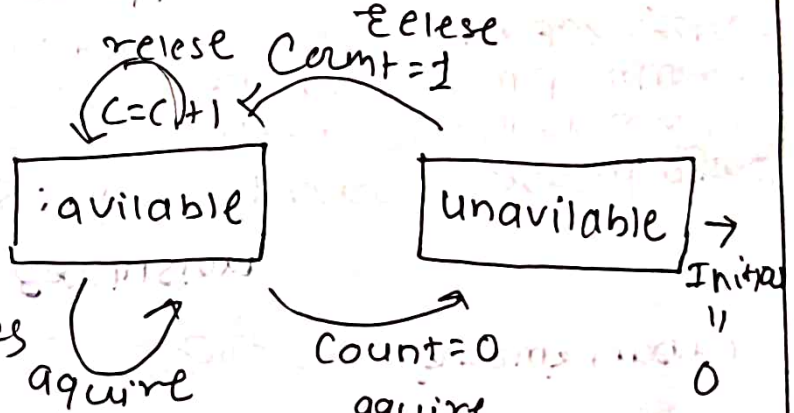
- When one process modifing the s value then none other process can't modified the 's' at that particular time

- Types
  - → Binary (mutex lock) 0 or 1
  - → Counting (integer values)

Binary:



Counting:



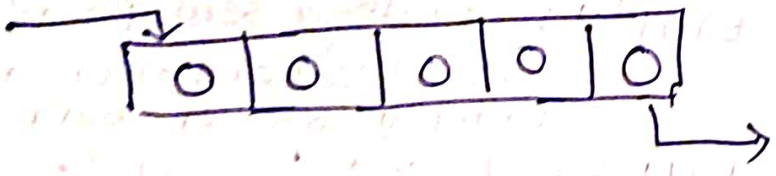
Producer :

```
do {
    wait (empty)
    wait (mutex)
    /* add data to buffer */
    signal (mutex)
    signal (Full)
} while (True)
```

consumer

```
do {
    wait (full)
    wait (mutex)
    /* remove data from buff */
    signal (mutex)
    Signal (Empty)
} while (True)
```

Initialized to



- **Readers writers Problem**
  - Database is shared between several processes
  - some want to → Read / Read / write
  - Two types → Readers / writers
  - Two Readers can access the data simultaneously. ✓
  - But a writer and another process can't access data ✗
  - writes have exclusive access to the database.

- **Soln** 2 semaphore & 1 variable (Int)
  1. M (mutex) : Binary semaphore
  2. wrt , semaphore common to both reader & writer
  3. read count (Int) = 0
     keep tracks of how many reades are reading the data.

Advantages of semaphore

1. Allow one Process at a time in CS
2. follow mutual exclusion
3. very efficient
4. No resource wastage
5. machine independent

Disadvantages:

1. uses wait() & signal() must implemented in correct order may cause deadlock
2. may leads to Priority inversion (low Priority gets high).

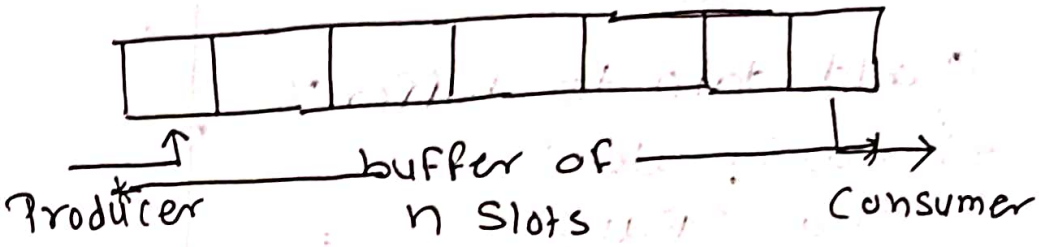
Moniter | High level abstraction for synchron

- It is abstract datatype.

```
moniter moniter_name
{ //shared variable
   Procedure P1 (..) {....}
   Procedure P2(..) {....}
   code (....) {....
  }
}
```

• Producer-consumer Problem using semaphore (Bounded buffer) (5)

⟹ There is a buffer of n slots each slot can store one unit of d



- Producer tries to insert data in empty sl
- consumer tries to remove data from filled slot
- Producer must not try to insert data
- when buffer is empty when buffer is empty
- Producer & consumer must NOT insert & remove data simultously.

solution: we will use 3 semaphore

1. m (mutex): to acquire & release lock
2. empty: counting semaphore intial val = number of slot (empty slot of buffer)
3. full : counting / intial = 0

Writer

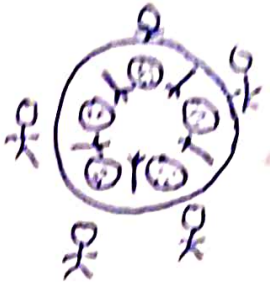
```
do{
  /*request for CS*/
  wait(wrt);
  // Perform the write
  signal(wrt);
}while(True);
```

Reader

```
do{
wait(mutex);
readcnt++;
if(readcnt==1)
  {
    wait(wrt);
    signal(mutex);
  }
wait(mutex);
readcnt--;
if(readcnt==0)
    signal(wrt);
    signal(mutex);

}while(True)
```

Philosophers Problem



State → Thinking
→ eating