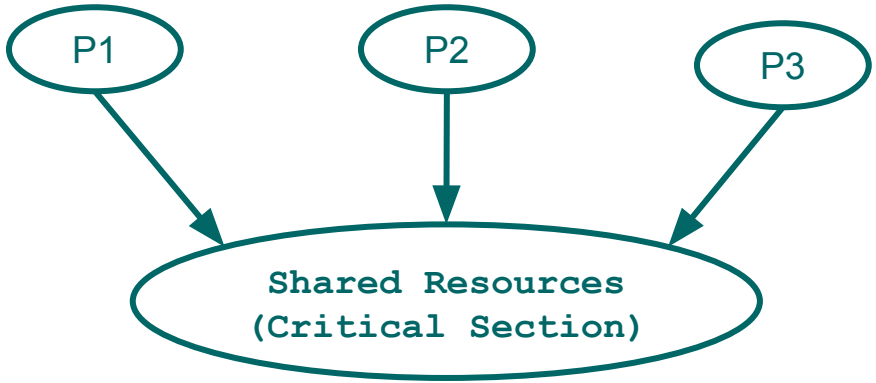
Process Synchronization

Prof. Harish D.G.
Dept. of Computer and IT
College of Engineering,Pune
www.harishgadade.com

Process Synchronization

- **Process Synchronization**

  Process Synchronization is a way to coordinate processes that use shared data. It occurs in an operating system among cooperating processes



Process Synchronization

- Types of Processes
  - Independent Processes
  - Cooperative processes
- Processes Synchronization problem assises in case of cooperative processes and also resources are shared in cooperative processes.

Critical Section Problem

- Critical section is a code segment that can be accessed by only one process at a time
- Critical section contain shared variables which need to be synchronized to maintain consistency of data variables.
- In the entry section, the process requests for entry in the critical section

- Problem

```
Do
{
    Entry Section
        Critical Section
    Exit Section
    Remaining Section
}while(TRUE);
```

Solution to Critical Section

Solution to critical section must satisfy following conditions

- **Mutual Exclusion :**

  If a process is executing in critical section, then no other process is allowed to execute in critical section

- **Progress :**

  If no process is in critical section,then no other process from outside can block it from entering in the critical section

- **Bounded Waiting :**

  A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Semaphore

- Semaphores are integer variables that are used to solve the critical section problem. After initialization, it can only be accessed by two atomic operations, **wait** and **signal**.
- Initial value of S is 1

1. Wait

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

2. Sigal

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

Solution to CS Problem using Semaphore

- Let p1,p2,p3,p4,...., pn are the processes that wants to execute in critical section

```
do
{
  wait(S);
     // Critical Section
  signal(S);
     //Remaining Section
}while(TRUE);
```

Types of Semaphore

There are two main types of semaphores i.e. counting semaphores and binary semaphores.

- **Counting Semaphores:**

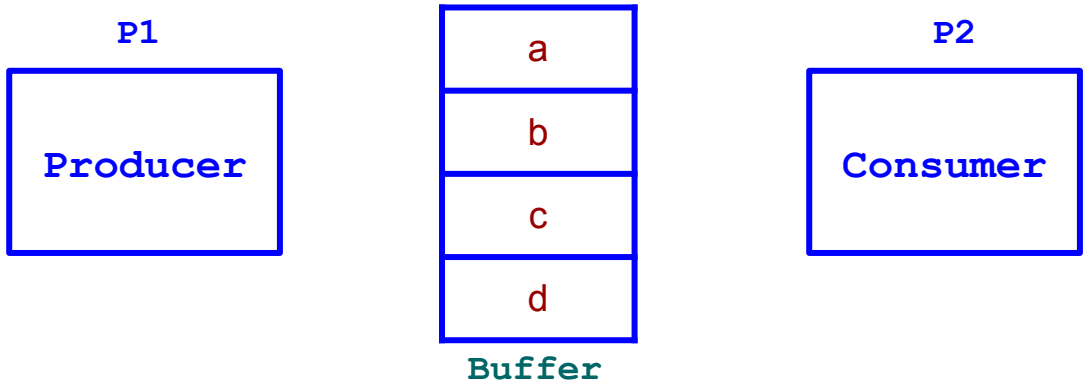
  These are integer value semaphores and have an unrestricted value domain. These semaphores are used to coordinate the resource access, where the semaphore count is the number of available resources. If the resources are added, semaphore count automatically incremented and if the resources are removed,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

Producer-Consumer Problem

We have a buffer of fixed size. A producer can produce an item and can place in the buffer. A consumer can pick items and can consume them. We need to ensure that when a producer is placing an item in the buffer, then at the same time consumer should not consume any item.



Solution to Producer-Consumer Problem

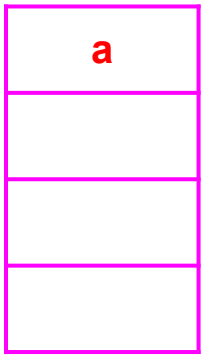
Produce-Consumer problem can be solved using Semaphore

- **Initialization of Semaphores:**

```
S = 1          // Binary Semaphore
Full = 0       // Initially all slots are empty,Thus full slots are 0
Empty = N      // Initially all slots are empty
```

P1

```
do
{
     // Produce an Item
     wait(empty);
     wait(S);
       // Place in Buffer
     signal(S)
     signal(full)
}while(TRUE)
```

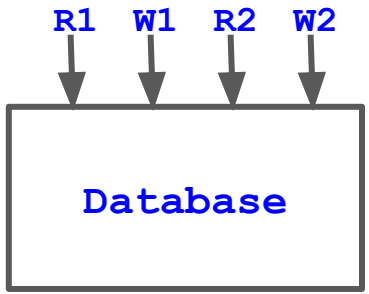


Buffer

P2

```
do
{
     wait(full);
     wait(S);
       // Remove Item from Buffer
     signal(S)
     signal(empty)
       // Consume Item
}while(TRUE)
```

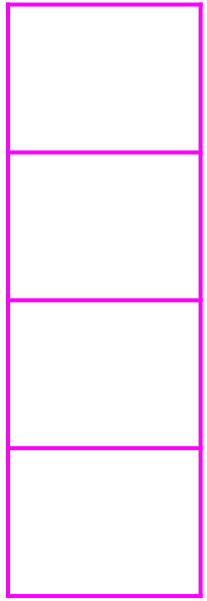
Readers-Writers Problem



R - W : Problem
W - R : Problem
W - W : Problem
R - R : No Problem

Solution to Readers-Writers Problem

```
int rc = 0
semaphore S = 1
semaphore db = 1
void reader(void)
{
do{
     wait(S);
     rc=rc+1;
     if(rc == 1) then
             wait(db)
     signal(S);
       // Data Base
     signal(S);
     rc=rc-1;
     if (rc = = 0) then
             signal(db)
     signal(s)
     Process Data
   }while(True)
}
```



Database

```
void writer(void)
{
    do
    {
         wait(db);
        // Data Base
     signal(db);
     }while(TRUE);
}
```

Lock Variables

Critical Section Solution Using Locks

```
do
{
    Acquire Lock
        CS
    Release Lock
}while(True);
```

```
do
{
    while(lock==1);
    lock==1;
      // Critical Section
    lock=0;
}while(TRUE);
```

Entry Code

Exit Code

Initial Value of Lock = 0

Lock=0 - Vacant
Lock=1 - Full

Note: It execute in user mode and does not give guarantee of Mutual Exclusion.