**Semaphore: (used for synchronization)**

If multiple process wants to access the same region of code parallely and the outcome is overlapped.

So, to avoid he problem we use semaphore, It is used to protect the critical/common region shared among multiple process.

Semaphore are classified into two types,

> Binary semaphore

only two state 0 and 1 i.e, (locked/ unlocked or available/ unavailable) Mutex implementation.

> Counting semaphore

Semaphore which allow arbitary resource count. i.e (scheduling as per the availability of resource is counting semaphore).

To create semaphore:

*#include<sys/types.h>*

*#include<sys/ipc.h>*

*#include<sys/sem.h>*

*int* ***semget****(key\_t* ***key****, int* ***nsems****, int* ***semflg****)*

Here,

**semget** is used to create semaphore.

**key** it is used as arbitary value or which is used to denote the semaphore.

**nsems** it specifies the number of semaphore.

**semflg** it specifes semaphore flag IPC\_CREAT or IPC\_EXCL. Need permission too.

Return value:

On success: sempaphore identified

On failure: -1

To perform operation on semaphore:

To perform operation like allocating resource, waiting for resource and freeing resource.

*#include<sys/types.h>*

*#include<sys/ipc.h>*

*#include<sys/sem.h>*

*int* ***semop****(int* ***semid****, struct* ***sembuf\* semops****, size\_t* ***nsemops****)*

Here,

**semid** it denotes the semaphore identifier from semget.

**semops** it is pointer to array of operations to be performed on the semaphore set.

**nsemops** it denote the number of operation on the array.

*Struct sembuf{*

*unsigned short sem\_num; // semaphore set num*

*short sem\_op; // semaphore operation*

*short sem\_flg; // Operation flag*

*};*

Here,

if **sem\_op** is -ve,

> Allocate or obtain resource.

> Block the calling process untill enough resource is freed by other process.

if **sem\_op** is 0,

> The calling process wait or sleeps until semaphore value reaches zero.

If **sem\_op** is +ve,

> Release resouces.

**nsemops** – Number of operation in that array.

To control semaphore:

*#include<sys/types.h>*

*#include<sys/ipc.h>*

*#include<sys/sem.h>*

*int* ***semctl****(int* ***semid****, int* ***semnum****, int* ***cmd, ...****)*

Here,

**semid** – It’s the identifier of the semaphore.

**semnum** – number of semaphore

**cmd** – To perform the required control operation on the semaphore.

*struct semid\_ds{*

*struct ipc\_perm sem\_perm; // Permission*

*time\_t sem\_otime; // Last semop time*

*time\_t sem\_ctime; // Last change time*

*unsigned long sem\_nsems; // Number of semaphore in set*

*}*

*union semun{*

*int val; // val for SETVAL*

*struct s emid\_ds \*buf; // Buffer for IPC\_STAT and IPC\_SET*

*unsigned short \*array; // Buffer for GETALL and SETALL*

*struc seminfo \*\_buf; // Buffer for IPC\_INFO and SEM\_INFO*

*}*

Here,

Return value,

On success: a non getative value (based on the command passed).

On failure: -1.