Exp. No. 6 Semaphore Implementation

Date:

Aim

To demonstrate the utility of semaphore in synchronization and multithreading.

Semaphore

- The POSIX system in Linux has its own built-in semaphore library.
- To use it, include `semaphore.h`.
- Compile the code by linking with `-lpthread -lrt`.
- To lock a semaphore or wait, use the **sem_wait** function.
- To release or signal a semaphore, use the **sem_post** function.
- A semaphore is initialised by using **sem_init**(for processes or threads)
- To declare a semaphore, the data type is `sem_t`.

Algorithm

1. 2 threads are being created, one 2 seconds after the first one.
2. But the first thread will sleep for 4 seconds after acquiring the lock.
3. Thus the second thread will not enter immediately after it is called, it will enter 4 – 2 = 2 secs after it is called.
4. *Stop*.

Program

```
/* C program to demonstrate working of Semaphores */

#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

sem_t mutex;

void* thread(void* arg)
{
   //wait
   sem_wait(&mutex);
   printf("\nEntered..\n");

   //critical section
   sleep(4);
```

```
    //signal
    printf("\nJust Exiting...\n");
    sem_post(&mutex);
}

int main()
{
    sem_init(&mutex, 0, 1);
    pthread_t t1,t2;
    pthread_create(&t1,NULL,thread,NULL);
    sleep(2);
    pthread_create(&t2,NULL,thread,NULL);
    pthread_join(t1,NULL);
    pthread_join(t2,NULL);
    sem_destroy(&mutex);
    return 0;
}
```

Output

```
$ gcc sem.c -lpthread

$ ./a.out

Entered..
Just Exiting...
Entered..
Just Exiting...
```

Result

Thus semaphore implementation has been demonstrated.