How to use POSIX semaphores in C language

Semaphores are very useful in process synchronization and multithreading. But how to use one in real life, for example say in C Language?

Well, we have the POSIX semaphore library in Linux systems. Let's learn how to use it.

The basic code of a semaphore is simple as presented here. But this code cannot be written directly, as the functions require to be atomic and writing code directly would lead to a context switch without function completion and would result in a mess.

The POSIX system in Linux presents its own built-in semaphore library. To use it, we have to :

1. Include semaphore.h
2. Compile the code by linking with -lpthread -lrt

To lock a semaphore or wait we can use the **sem_wait** function:

```
int sem_wait(sem_t *sem);
```

To release or signal a semaphore, we use the **sem_post** function:

```
int sem_post(sem_t *sem);
```

A semaphore is initialised by using **sem_init**(for processes or threads) or **sem_open** (for IPC).

```
sem_init(sem_t *sem, int pshared, unsigned int value);
```

Where,

- **sem** : Specifies the semaphore to be initialized.
- **pshared** : This argument specifies whether or not the newly initialized semaphore is shared between processes or between threads. A non-zero value means the semaphore is shared between processes and a value of zero means it is shared between threads.

- **value** : Specifies the value to assign to the newly initialized semaphore.

To destroy a semaphore, we can use **sem_destroy**.

```
sem_destoy(sem_t *mutex);
```

To declare a semaphore, the data type is sem_t.

Code –

```
// 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;
}
```

Compilation should be done with gcc a.c -lpthread -lrt

Explanation –

2 threads are being created, one 2 seconds after the first one.

But the first thread will sleep for 4 seconds after acquiring the lock.

Thus the second thread will not enter immediately after it is called, it will enter 4 – 2 = 2 secs after it is called.
So the output is:

```
Entered..

Just Exiting...

Entered..

Just Exiting...
```

but not:

```
Entered..

Entered..

Just Exiting...

Just Exiting...
```

This article is contributed by Suprotik Dey. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Posts:

Signals in C language
Difference between while(1) and while(0) in C language
kbhit in C language
fgets() and gets() in C language
Stopwatch using C language
C Language Introduction