Program for Process Synchronization using Semaphores

2 Comments / Programs / By Baljit Singh Saini

Semaphore is an integer variable which is accessed or modified by using two atomic operations: *wait()* and *signal()*. In C program the corresponding operations are *sem_wait()* and *sem_post()*. Here, we write a Program for Process Synchronization using Semaphores to understand the implementation of *sem_wait()* and *sem_signal()* to avoid a race condition.

Program 1: Program for process synchronization using semaphores

Q. Program creates two threads: one to increment the value of a shared variable and second to decrement the value of the shared variable. Both the threads make use of semaphore variable so that only one of the threads is executing in its critical section

```
#include<pthread.h>
#include<stdio.h>
#include<semaphore.h>
#include<unistd.h>
void *fun1();
 void *fun2();
 int shared=1; //shared variable
 sem_t s; //semaphore variable
 int main()
 {
 sem_init(&s,0,1); //initialize semaphore variable - 1st argument is address of variable, 2nd is number of processes
sharing semaphore, 3rd argument is the initial value of semaphore variable
 pthread_t thread1, thread2;
 pthread_create(&thread1, NULL, fun1, NULL);
 pthread_create(&thread2, NULL, fun2, NULL);
 pthread_join(thread1, NULL);
 pthread_join(thread2,NULL);
 printf("Final value of shared is %d\n",shared); //prints the last updated value of shared variable
 }



void *fun1()
{
    int x;
```

```
    printf("Local updation by Thread1: %d\n",x);
    sleep(1); //thread1 is preempted by thread 2
    shared=x; //thread one updates the value of shared variable
    printf("Value of shared variable updated by Thread1 is: %d\n",shared);
    sem_post(&s);
}



void *fun2()
{
    int y;
    sem_wait(&s);
    y=shared;//thread2 reads value of shared variable
    printf("Thread2 reads the value as %d\n",y);
    y--;  //thread2 increments its value
    printf("Local updation by Thread2: %d\n",y);
    sleep(1); //thread2 is preempted by thread 1
    shared=y; //thread2 updates the value of shared variable
    printf("Value of shared variable updated by Thread2 is: %d\n",shared);
    sem_post(&s);
}
```

The final value of the variable *shared* will be 1. When any one of the threads executes the wait operation the value of "s" becomes zero. Hence the other thread (even if it preempts the running thread) is not able to successfully execute the wait operation on "*s*". Thus not able to read the inconsistent value of the shared variable. This ensures that only one of the thread is running in its critical section at any given time. The output is as shown below. The working of the program is also discussed in detail.

Output

```
baljit@baljit:~/cse325$ ./a.out
Thread1 reads the value as 1
Local updation by Thread1: 2
Value of shared variable updated by Thread1 is: 2
Thread2 reads the value as 2
Local updation by Thread2: 1
Value of shared variable updated by Thread2 is: 1
Final value of shared is 1
```

Process synchronization using semaphore

How it works?

The process initializes the semaphore variable *s* to '1' using the *sem_init()* function. The initial value is set to '1' because binary semaphore is used here. If you have multiple instances of the resource then counting semaphores can be used. Next, the process creates two threads. *thread1* acquires the semaphore variable by calling *sem_wait()*. Next, it executes statements in its critical section part. We use *sleep(1)* function to preempt *thread1* and start *thread2*. This simulates a real-life scenario. Now, when *thraed2* executes *sem_wait()* it will not be able to do so because *thread1* is already in the critical section. Finally, thread1 calls *sem_post()* function. Now *thread2* will be able to acquire *s* using *sem_wait()*. This ensures synchronization among threads.

Practice Program

Q. Write a program to achieve synchronization between multiple threads. The threads try to acquire a resource that has two instances

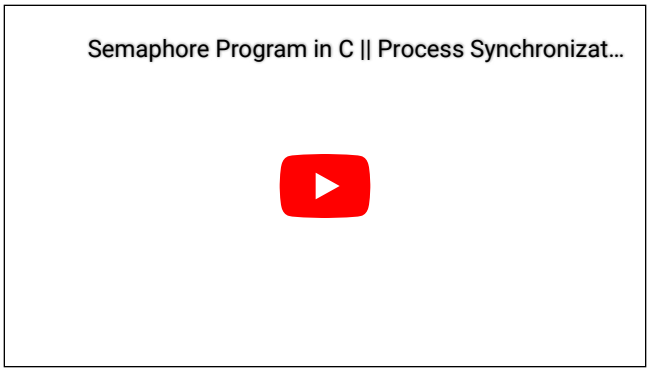
Viva questions on Program for Process Synchronization using Semaphores

Q4. What is the significance of using *sleep(1)* function in the functions *fun1()* and *fun2()*?

Q5. How to use counting semaphores?

Q6. What will be the initial value of the semaphore variable if there are 5 instances of the resource?

Video Link



Relevant Programs

Program to implement Race condition in Linux

Process Synchronization using locks

Program to create Thread in Linux

← Previous Post

Next Post →

2 thoughts on “Program for Process Synchronization using Semaphores”

ANGELITA
NOVEMBER 24, 2020 AT 3:18 AM

My partner and I stumbled over here coming from a different page
and thought I may as well check things out. I like what
I see so i am just following you. Look forward to looking at your web page again.

Log in to Reply

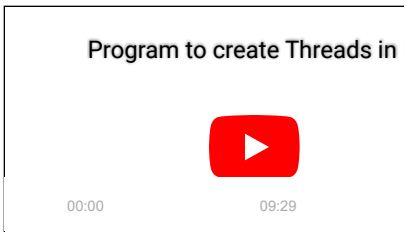
Log in to Reply

Leave a Comment

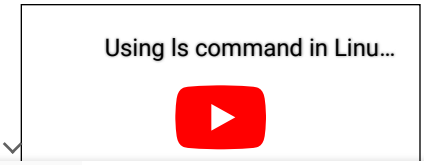
You must be logged in to post a comment.

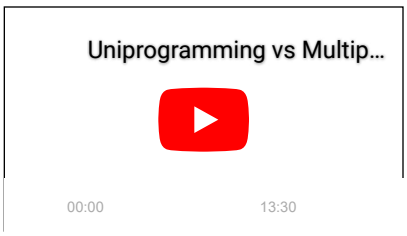
Search ...

OS Lab



Linux Essentials Series





Categories

Programs

Question Hub

UGC-NET MCQ's for OS

Uncategorized