Dextutor

Program for Process Synchronization using mutex locks

Leave a Comment / Programs / By Baljit Singh Saini

/* Program for process synchronization using locks
Program create two threads: one to increment the value of a shared variable and second to decrement the value of shared variable. Both the threads make use of locks so that only one of the threads is executing in its critical section */

```
#include<pthread.h>
#include<stdio.h>
#include<unistd.h>
void *fun1();
void *fun2();
 int shared=1; //shared variable
 pthread_mutex_t l; //mutex lock
 int main()
 {
 pthread_mutex_init(&l, NULL); //initializing mutex locks
 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("Thread1 trying to acquire lock\n");
     pthread_mutex_lock(&l); //thread one acquires the lock. Now thread 2 will not be able to acquire the lock
//until it is unlocked by thread 1
    printf("Thread1 acquired lock\n");
     x=shared;//thread one reads value of shared variable
     printf("Thread1 reads the value of shared variable as %d\n",x);
```

-15%

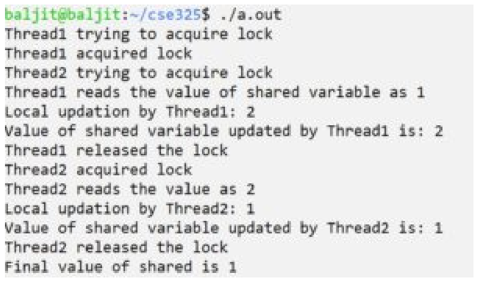
```
        pthread_mutex_unlock(&l);
        printf("Thread1 released the lock\n");
    }



void *fun2()
    {
        int y;
        printf("Thread2 trying to acquire lock\n");
        pthread_mutex_lock(&l);
        printf("Thread2 acquired lock\n");
        y=shared;//thread two reads value of shared variable
        printf("Thread2 reads the value as %d\n",y);
        y--;  //thread two increments its value
        printf("Local updation by Thread2: %d\n",y);
        sleep(1); //thread two is preempted by thread 1
        shared=y; //thread one updates the value of shared variable
        printf("Value of shared variable updated by Thread2 is: %d\n",shared);
        pthread_mutex_unlock(&l);
        printf("Thread2 released the lock\n");
    }
```

The final value of shared variable will be 1. When any one of the threads acquires the lock and is making changes to shared variable the other thread (even if it preempts the running thread) is not able to acquire the lock and thus not able to read the inconsistent value of shared variable. Thus only one of the thread is running in its critical section at any given time */

Sample Output

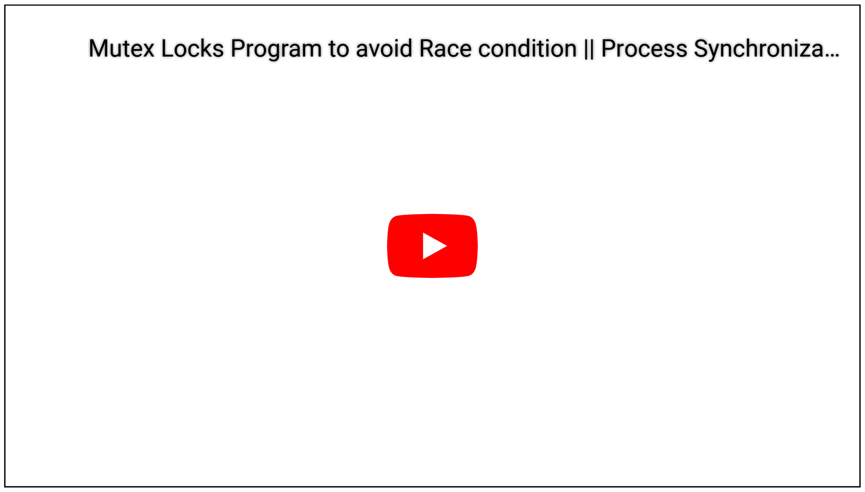


```
baljit@baljit:~/cse325$ ./a.out
Thread1 trying to acquire lock
Thread1 acquired lock
Thread2 trying to acquire lock
Thread1 reads the value of shared variable as 1
Local updation by Thread1: 2
Value of shared variable updated by Thread1 is: 2
Thread1 released the lock
Thread2 acquired lock
Thread2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

Viva Questions on Program for Process Synchronization using mutex locks

Q1. What is the initial value of the *lock* variable?
Q2. Why we use *pthread_join()* function in the above program?
Q3. Why is the second parameter in *pthread_mutex_init() NULL*?
Q4. What is the significance of using *sleep(1)* function in the functions *fun1()* and *fun2()*?
Q5. What is the use of *pthread_mutex_lock()* function?
Q6. What is the use of *pthread_mutex_unlock()* function?

Video Link


Mutex Locks Program to avoid Race condition || Process Synchroniza…

Relevant Programs

- [Program for Thread creation in Linux]
- [Simulating Race condition]
- [Process synchronization using semaphore]

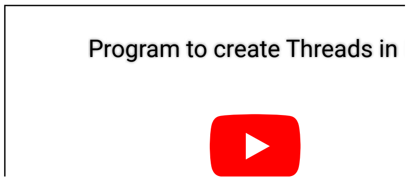
← Previous Post Next Post →

Leave a Comment

You must be logged in to post a comment.

Search …

OS Lab



Linux Essentials Series

00:00 13:14

Uniprogramming vs Multip...

00:00 13:30

-50%

₹1,119 ₹1,119

-40%

₹1,343 ₹1,119

-40% -40%

₹1,343

-40%

₹1,119

Shop Eyewear at l

Lenskart

Categories

- [Database]
  - [Entity-Relationship(ER)]
  - [Introduction to Database]
- [Linux]
  - [commands]
  - [Shell Scripting]
- [Operating System]
  - [CPU Scheduling]
  - [Deadlock]
  - [Disk Management]

- Practice Problems
- Process
- Process Synchronization
- System calls
- Thread
- Programs
- Question Hub
- UGC-NET MCQ's for OS
- Uncategorized