/* Program to show the race condition.
Program create two threads: one to increment the value of a shared variable and second to decrement thev value of shared variable. Both the threads are executed, so the final value of shared variable should be same as its initial value. But due to race condition it would not be same.
*/

```
#include<pthread.h>
#include<stdio.h>
void *fun1();
void *fun2();
int shared=1; //shared variable
int main()
{
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;
   x=shared;//thread one reads value of shared variable
   x++;  //thread one increments its value
   sleep(1);  //thread one is preempted by thread 2
   shared=x; //thread one updates the value of shared variable
}

void *fun2()
{
   int y;
   y=shared;//thread two reads value of shared variable
   y--;  //thread two increments its value
   sleep(1); //thread two is preempted by thread 1
   shared=y; //thread one updates the value of shared variable
}
```

/* the final value of shared variable should have been 1 but it will be either 2 or 0 depending upon which thread executes first. This happened because the two processes were not synchronized. When one thread was modifying the value of shared variable the other thread must not have read its value for modification. This can be achieved using locks or semaphores */

Program 2:

/* 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>
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;
   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
   x=shared;//thread one reads value of shared variable
   x++;  //thread one increments its value
   sleep(1);  //thread one is preempted by thread 2
   shared=x; //thread one updates the value of shared variable
   pthread_mutex_unlock(&l);
}

void *fun2()
{
   int y;
   pthread_mutex_lock(&l);
   y=shared;
   y--;
   sleep(1);
   shared=y;
   pthread_mutex_unlock(&l);
}
```

/* 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 */