**Ex. No. 10** **Threading and Synchronization**

**Date:**

**Aim**

To demonstrate threading and synchronization using mutex.

**Description**

- Thread synchronization is defined as a mechanism which ensures that two or more concurrent processes or threads do not simultaneously execute some particular program segment known as critical section.
- Processes' access to critical section is controlled by using synchronization techniques.
- When one thread starts executing the critical section (serialized segment of the program) the other thread should wait until the first thread finishes.
- If proper synchronization techniques are not applied, it may cause a race condition where the values of variables may be unpredictable
- A Mutex is a lock that we set before using a shared resource and release after using it.
- When the lock is set, no other thread can access the locked region of code. So this ensures a synchronized access of shared resources in the code.

**Algorithm**

1. Create two threads
2. Let the threads share a common resource, say counter
3. Even if thread2 si scheduled to start while thread was not done, access to shared resource is not done as it is locked by mutex
4. Once thread1 completes, thread2 starts execution
5. Stop

**Program**

```
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

pthread_t tid[2];
int counter;
pthread_mutex_t lock;

void* trythis(void *arg)
{
   pthread_mutex_lock(&lock);

   unsigned long i = 0;
   counter += 1;
   printf("\n Job %d has started\n", counter);

   for(i=0; i<(0xFFFFFFFF);i++);
   printf("\n Job %d has finished\n", counter);

   pthread_mutex_unlock(&lock);

   return NULL;
}

main()
{
   int i = 0;
   int error;

   if (pthread_mutex_init(&lock, NULL) != 0)
   {
      printf("\n mutex init has failed\n");
      return 1;
   }

   while(i < 2)
   {
      err = pthread_create(&(tid[i]), NULL, &trythis, NULL);
      if (error != 0)
         printf("\nThread    can't    be    created    :[%s]",
strerror(error));
      i++;
   }

   pthread_join(tid[0], NULL);
   pthread_join(tid[1], NULL);
   pthread_mutex_destroy(&lock);

   return 0;
}
```

**Output**

```
$ gcc filename.c -lpthread

$ ./a.out
Job 1 started
Job 1 finished
Job 2 started
Job 2 finished
```

**Result**

**Thus concurrent threads were synchronized using mutex lock.**