Experiment 12

Aim: Write a program for Process Synchronization using mutex lock.

- **Mutex (Mutual Exclusion) is a lock used to ensure that only one thread accesses the critical section at a time.**

Why is Synchronization Needed?

- **When multiple threads access a shared variable (shared) concurrently:**
  - **They may read/write incorrect or stale data.**
  - **This leads to unpredictable results, called a race condition.**

Sample Code:

```
#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_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);
   return 0;
}
void *fun1() {
   int x;
   printf("Thread1 trying to acquire lock\n");
   pthread_mutex_lock(&l);  // Thread acquires lock
   printf("Thread1 acquired lock\n");
   x = shared;
   printf("Thread1 reads the shared variable as: %d\n", x);
   x++;
   printf("Local updation by thread1: %d\n", x);
   sleep(1); // Thread1 is preempted by Thread2
   shared = x;  // Thread1 updates the value of shared
   printf("Value of shared variable updated by Thread1 is: %d\n", shared);
   pthread_mutex_unlock(&l);
   printf("Thread1 released the lock\n");
   return NULL;
}
void *fun2() {
   int y;
   printf("Thread2 trying to acquire lock\n");
   pthread_mutex_lock(&l);  // Thread acquires lock
   printf("Thread2 acquired lock\n");
   y = shared;
   printf("Thread2 reads the shared variable as: %d\n", y);
```

```
    y++;
    printf("Local updation by thread2: %d\n", y);
    sleep(1); // Thread2 is preempted by Thread1
    shared = y;  // Thread2 updates the value of shared
    printf("Value of shared variable updated by Thread2 is: %d\n", shared);
    pthread_mutex_unlock(&l);
    printf("Thread2 released the lock\n");
    return NULL;
}
```

Sample Screenshots:

Creating a vi file named mutex_lock.c

```
localhost:~/AK_12# vi mutex_lock.c
```

Write the C program in mutex_lock.c

```
    printf("Value of shared variable updated by Thread1 is: %d\n", shared);

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

    return NULL;
}
void *fun2() {
    int y;
    printf("Thread2 trying to acquire lock\n");

    pthread_mutex_lock(&l);  // Thread acquires lock
    printf("Thread2 acquired lock\n");

    y = shared;
    printf("Thread2 reads the shared variable as: %d\n", y);

    y++;
    printf("Local updation by thread2: %d\n", y);

    sleep(1); // Thread2 is preempted by Thread1

    shared = y;  // Thread2 updates the value of shared
    printf("Value of shared variable updated by Thread2 is: %d\n", shared);

    pthread_mutex_unlock(&l);
    printf("Thread2 released the lock\n");

    return NULL;
}
```

Compilation & Output execution:

```
localhost:~/AK_12# gcc mutex_lock.c -o mutex_lock -lpthread
localhost:~/AK_12# ./mutex_lock
Thread2 trying to acquire lock
Thread2 acquired lock
Thread2 reads the shared variable as: 1
Local updation by thread2: 2
Thread1 trying to acquire lock
Value of shared variable updated by Thread2 is: 2
Thread1 acquired lock
Thread1 reads the shared variable as: 2
Local updation by thread1: 3
Thread2 released the lock
Value of shared variable updated by Thread1 is: 3
Thread1 released the lock
Final value of shared is: 3
```