**Operating System Lab**

**22F-3168**

**Mariam Fatima**

**PROCESS SYNCHRONIZATION**

**With Two Threads**

**CODE:**

#include<pthread.h>

#include<stdio.h>

#include<unistd.h>

void \*fun1();

void \*fun2();

int shared=1;//shred memory

pthread\_mutex\_t l; //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);

}

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 untill it is unlocked bu 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);

x++; //thread one increments its value

printf("Local updation by Thread1: %d\n",x);

sleep(1); //thread one 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);

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 varable

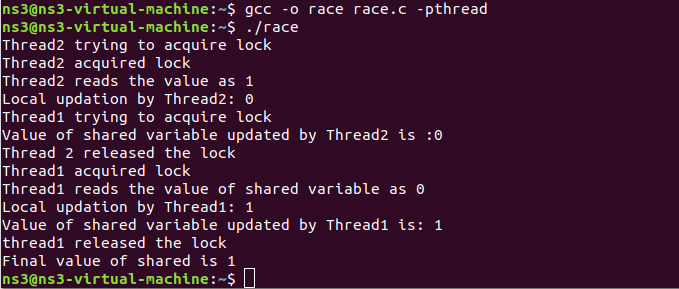
printf("Value of shared variable updated by Thread2 is :%d\n",shared);

pthread\_mutex\_unlock(&l);

printf("Thread 2 released the lock\n");

}

**OUTPUT:**



**Taking user input of No. of Threads and using one function:**

**Code:**

#include <pthread.h>

#include <stdio.h>

#include <unistd.h>

void \*threadFunction(void \*arg);

int shared = 1; // shared memory

pthread\_mutex\_t l; // lock

int main() {

int numThreads;

printf("Enter the number of threads (2 or more): ");

scanf("%d", &numThreads);

pthread\_mutex\_init(&l, NULL); // initializing mutex locks

pthread\_t threads[numThreads];

// Create threads

for (int i = 0; i < numThreads; i++) {

pthread\_create(&threads[i], NULL, threadFunction, (void \*)(intptr\_t)i);

}

// Wait for threads to finish

for (int i = 0; i < numThreads; i++) {

pthread\_join(threads[i], NULL);

}

printf("Final value of shared is %d\n", shared);

pthread\_mutex\_destroy(&l); // destroy mutex

return 0;

}

void \*threadFunction(void \*arg) {

int threadId = (intptr\_t)arg; // Cast argument back to integer

int localValue;

if (threadId % 2 == 0) { // Even thread id increments

printf("Thread %d trying to acquire lock\n", threadId);

pthread\_mutex\_lock(&l); // thread acquires the lock

printf("Thread %d acquired lock\n", threadId);

localValue = shared; // read value

printf("Thread %d reads the value of shared variable as %d\n", threadId, localValue);

localValue++; // increment

printf("Local updation by Thread %d: %d\n", threadId, localValue);

sleep(1); // simulate preemption

shared = localValue; // update shared value

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

pthread\_mutex\_unlock(&l);

printf("Thread %d released the lock\n", threadId);

} else { // Odd thread id decrements

printf("Thread %d trying to acquire lock\n", threadId);

pthread\_mutex\_lock(&l); // thread acquires the lock

printf("Thread %d acquired lock\n", threadId);

localValue = shared; // read value

printf("Thread %d reads the value of shared variable as %d\n", threadId, localValue);

localValue--; // decrement

printf("Local updation by Thread %d: %d\n", threadId, localValue);

sleep(1); // simulate preemption

shared = localValue; // update shared value

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

pthread\_mutex\_unlock(&l);

printf("Thread %d released the lock\n", threadId);

}

return NULL;

}

