#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#include <stdio.h>

/\*

This program provides a possible solution for producer-consumer problem using mutex and semaphore.

I have used 5 producers and 5 consumers to demonstrate the solution. You can always play with these values.

\*/

#define MaxItems 5 // Maximum items a producer can produce or a consumer can consume

#define BufferSize 5 // Size of the buffer

sem\_t empty;

sem\_t full;

int in = 0;

int out = 0;

int buffer[BufferSize];

pthread\_mutex\_t mutex;

void \*producer(void \*pno)

{

int item;

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

item = rand(); // Produce an random item

sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

buffer[in] = item;

printf("Producer %d: Insert Item %d at %d\n", \*((int \*)pno),buffer[in],in);

in = (in+1)%BufferSize;

pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

}

}

void \*consumer(void \*cno)

{

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

sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

int item = buffer[out];

printf("Consumer %d: Remove Item %d from %d\n",\*((int \*)cno),item, out);

out = (out+1)%BufferSize;

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty);

}

}

int main()

{

pthread\_t pro[5],con[5];

pthread\_mutex\_init(&mutex, NULL);

sem\_init(&empty,0,BufferSize);

sem\_init(&full,0,0);

int a[5] = {1,2,3,4,5}; //Just used for numbering the producer and consumer

for(int i = 0; i < 5; i++) {

pthread\_create(&pro[i], NULL, (void \*)producer, (void \*)&a[i]);

}

for(int i = 0; i < 5; i++) {

pthread\_create(&con[i], NULL, (void \*)consumer, (void \*)&a[i]);

}

for(int i = 0; i < 5; i++) {

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

}

for(int i = 0; i < 5; i++) {

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

}

pthread\_mutex\_destroy(&mutex);

sem\_destroy(&empty);

sem\_destroy(&full);

return 0;

}

…………………………………………………………………………………………………………….

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

/\*

This program provides a possible solution for first readers writers problem using mutex and semaphore.

I have used 10 readers and 5 producers to demonstrate the solution. You can always play with these values.

\*/

sem\_t wrt;

pthread\_mutex\_t mutex;

int cnt = 1;

int numreader = 0;

void \*writer(void \*wno)

{

sem\_wait(&wrt);

cnt = cnt\*2;

printf("Writer %d modified cnt to %d\n",(\*((int \*)wno)),cnt);

sem\_post(&wrt);

}

void \*reader(void \*rno)

{

// Reader acquire the lock before modifying numreader

pthread\_mutex\_lock(&mutex);

numreader++;

if(numreader == 1) {

sem\_wait(&wrt); // If this id the first reader, then it will block the writer

}

pthread\_mutex\_unlock(&mutex);

// Reading Section

printf("Reader %d: read cnt as %d\n",\*((int \*)rno),cnt);

// Reader acquire the lock before modifying numreader

pthread\_mutex\_lock(&mutex);

numreader--;

if(numreader == 0) {

sem\_post(&wrt); // If this is the last reader, it will wake up the writer.

}

pthread\_mutex\_unlock(&mutex);

}

int main()

{

pthread\_t read[10],write[5];

pthread\_mutex\_init(&mutex, NULL);

sem\_init(&wrt,0,1);

int a[10] = {1,2,3,4,5,6,7,8,9,10}; //Just used for numbering the producer and consumer

for(int i = 0; i < 10; i++) {

pthread\_create(&read[i], NULL, (void \*)reader, (void \*)&a[i]);

}

for(int i = 0; i < 5; i++) {

pthread\_create(&write[i], NULL, (void \*)writer, (void \*)&a[i]);

}

for(int i = 0; i < 10; i++) {

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

}

for(int i = 0; i < 5; i++) {

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

}

pthread\_mutex\_destroy(&mutex);

sem\_destroy(&wrt);

return 0;

}

………………………………….

#include<stdio.h>  
#include<stdlib.h>  
#include<pthread.h>  
#include<semaphore.h>  
#include<unistd.h>  
  
sem\_t room;  
sem\_t chopstick[5];  
  
void \* philosopher(void \*);  
void eat(int);  
int main()  
{  
int i,a[5];  
pthread\_t tid[5];  
  
sem\_init(&room,0,4);  
  
for(i=0;i<5;i++)  
sem\_init(&chopstick[i],0,1);  
  
for(i=0;i<5;i++){  
a[i]=i;  
pthread\_create(&tid[i],NULL,philosopher,(void \*)&a[i]);  
}  
for(i=0;i<5;i++)  
pthread\_join(tid[i],NULL);  
}  
  
void \* philosopher(void \* num)  
{  
int phil=\*(int \*)num;  
  
sem\_wait(&room);  
printf("\nPhilosopher %d has entered room",phil);  
sem\_wait(&chopstick[phil]);  
sem\_wait(&chopstick[(phil+1)%5]);  
  
eat(phil);  
sleep(2);  
printf("\nPhilosopher %d has finished eating",phil);  
  
sem\_post(&chopstick[(phil+1)%5]);  
sem\_post(&chopstick[phil]);  
sem\_post(&room);  
}  
  
void eat(int phil)  
{  
printf("\nPhilosopher %d is eating",phil);  
}

…………………………………………………………………………………………………………..

AIM: Write a program to show the race condition. The program creates two threads-one to increment the value of a shared variable and the second to decrement the value of the shared variable.

void \*fun1();

void \*fun2();

int shared=1;

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);

}

void \*fun1()

{

int x;

x=shared;

printf("Thread1 reads the value of shared variable as %d\n",x);

x++;

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

sleep(1);

shared=x;

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

}

void \*fun2()

{

int y;

y=shared;

printf("Thread2 reads the value as %d\n",y);

y--;

printf("Local updation by Thread2: %d\n",y);

sleep(1);

shared=y;

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

}

………………………………………………..

To write a program to execute Race Condition in Reverse. In this change the order of these 2 threads and display the final values of the shared variable.

void \*fun1();

void \*fun2();

int shared=1;

int main()

{

pthread\_t thread1, thread2;

pthread\_create(&thread2, NULL, fun2, NULL);

pthread\_create(&thread1, NULL, fun1, NULL);

pthread\_join(thread2, NULL);

pthread\_join(thread1,NULL);

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

}

void \*fun1()

{

int x;

x=shared;

printf("Thread1 reads the value of shared variable as %d\n",x);

x++;

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

sleep(1);

shared=x;

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

}

void \*fun2()

{

int y;

y=shared;

printf("Thread2 reads the value as %d\n",y);

y--;

printf("Local updation by Thread2: %d\n",y);

sleep(1);

shared=y;

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

}

…………………………………………………………………………………………………………………