ASSIGNMENT – 5

DEKKER’S AND PETERSON ALGORITHM

Name- Ashutosh Soni

Id - 2018ucp1505

Question: Implement of Dekker’s Algorithm for two threads

Answer:

**Code for the following:**

// Dekkers algorithm implementation

// Free from satrvation

// Mutual Exlusion satsisfied

// Free from deadlock

#include<bits/stdc++.h>

#include<pthread.h>

using namespace std;

int turn=1;

bool wantp=false,wantq=false;

int x=0;

// Critical Section

void critical\_section(){

x++;

}

// process P

void\* p(){

while(1){

if(x>=50){

return NULL;

}

wantp=true;

while(wantq){

if(turn==2){

wantp=false;

while(turn!=1){

}

wantp=true;

}

}

cout<<"Critical Section of P starts here"<<endl;

critical\_section();

cout<<"Critical Section of P ends here"<<endl;

turn=2;

wantp=false;

}

}

// process Q

void\* q(){

while(1){

if(x>=50){

return NULL;

}

wantq=true;

while(wantp){

if(turn==1){

wantq=false;

while(turn!=2){

}

wantq=true;

}

}

cout<<"Critical Section of Q starts here"<<endl;

critical\_section();

cout<<"Critical Section of Q ends here"<<endl;

turn=1;

wantq=false;

}

}

// start P

void\* start\_p(void\* arg){

p();

return (void\*)0;

}

// start q

void\* start\_q(void\* arg){

q();

return (void\*)0;

}

int main(){

pthread\_t pid,qid;

// creating two thread

pthread\_create(&pid,NULL,&start\_p,NULL);

pthread\_create(&qid,NULL,\*start\_q,NULL);

// Joining threads

pthread\_join(pid,NULL);

pthread\_join(qid,NULL);

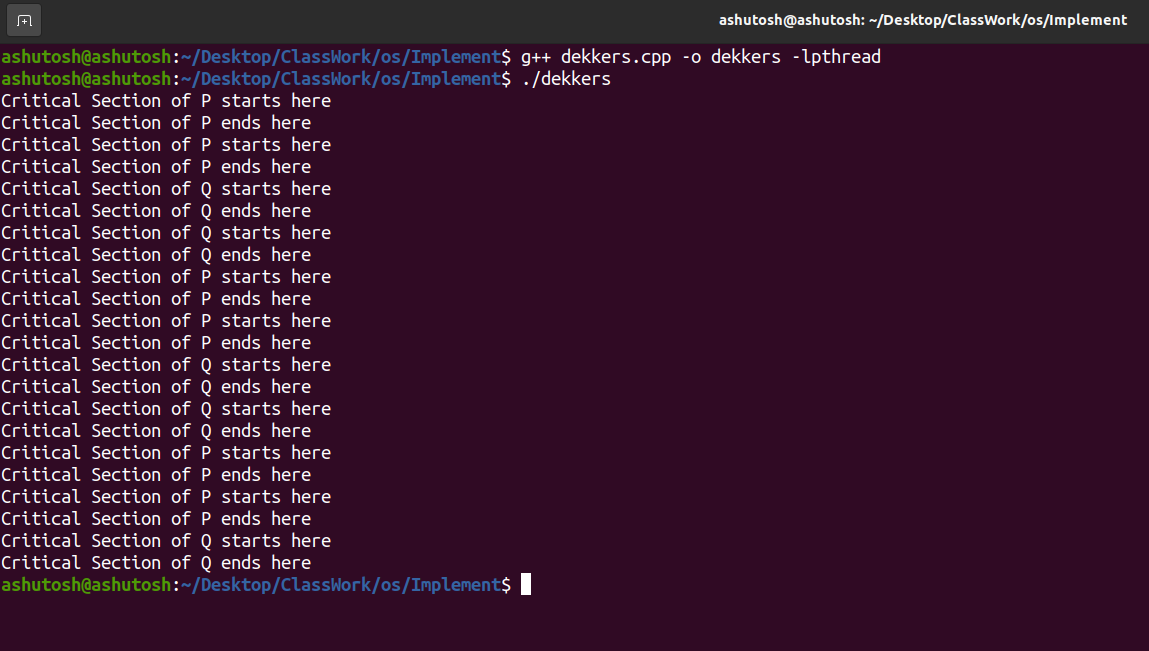
// Exit

pthread\_exit(NULL);

return 0;

}

Output for the following:



Question: Implement of Peterson’s Algorithm for two threads.

Answer:

Code for the following:

#include <stdio.h>

#include <pthread.h>

int n= 20;

int flag[20]; //change 10 with number n

int turn;

const int MAX = 100;

int ans = 0;

//in start of program

void lock\_init()

{

int i;

for(i=0;i<n;i++)

{

flag[i] = 0;

}

turn = 0;

}

// Before entering critical section

void lock(int self)

{

//flag[self]=1 show that process self want to enter in critical section

if(self <=n-1 && self >=0)

{

flag[self] = 1;

}

// first give chance to another process

turn = n-self;

//wait untill other process are in critical section

if(n-1-self != self)

{

while (flag[n-1-self]==1 && turn==n-self) ;

}

else if(n-2-self >= 0)

{

while (flag[n-2-self]==1 && turn==n-self) ;

}

}

//when goes out of critical section

void unlock(int self)

{

//flag[self] = 0 show that process self going out of critical section

if(self <=n && self >=0)

{

flag[self] = 0;

}

}

//every process run the same function

void\* func(void \*s)

{

int i = 0;

int\* temp = (int\*)s;

int self=\*temp;

lock(self);

//critical section starts here

printf("Thread Entered: %d\n", self);

//ans variable changed by every thread

for (i=0; i<MAX; i++)

ans++;

//critical section ends here

unlock(self);

return (void\*)0;

}

// Driver code

int main()

{

pthread\_t threads[n];

lock\_init();

void \*retvals[n];

int count;

for (count = 0; count < n; ++count)

{

int \*temp= &count;

void\* temp1 =(void\*)temp;

if (pthread\_create(&threads[count], NULL,func,temp1) != 0)

{

fprintf(stderr, "error: Cannot create thread # %d\n", count);

break;

}

}

for (int i = 0; i < n; ++i)

{

if (pthread\_join(threads[i], &retvals[i]) != 0)

{

fprintf(stderr, "error: Cannot join thread # %d\n", i);

}

}

//if both are same then we say that our solution is correct

printf("Ans variable after each thread: %d | Ans should be : %d\n", ans, MAX\*n);

return 0;

}

Output for the following:

