EXCERSICE (CONTINUED) Scheduling

SRTF(SHORTEST REMAINING TIME FIRST ) C++ IMPLEMENTATION

#include <iostream>

using namespace std;

int main() {

int n,i,j,temp=0,tat=0;

int arrTime[10],execTime[10],nexecTime[10],pro[10],npro[10];

cout<<"Enter number of processes:"<<endl;

cin>>n;

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

pro[i]=i;

npro[i]=i;

}

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

cout<<"Enter execution time of process "<<i<<":";

cin>>execTime[i];

cout<<"Enter arrival time of process "<<i<<":";

cin>>arrTime[i];

cout<<endl;

}

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

tat = tat+execTime[i];

}

for(i=0;i<n-1;i++) {

for(j=0;j<n-i-1;j++) {

if(arrTime[j]>arrTime[j+1]) {

temp=arrTime[j+1];

arrTime[j+1]=arrTime[j];

arrTime[j]=temp;

temp=pro[j+1];

pro[j+1]=pro[j];

pro[j]=temp;

}

}

}

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

cout<<" P"<<pro[i]<<" ";

execTime[pro[i]]=execTime[pro[i]]-1;

}

for(i=0;i<n-1;i++) {

for(j=0;j<n-i-1;j++) {

if(execTime[j]>execTime[j+1]) {

temp=execTime[j+1];

execTime[j+1]=execTime[j];

execTime[j]=temp;

temp=npro[j+1];

npro[j+1]=npro[j];

npro[j]=temp;

}

}

}

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

if(execTime[i]!=0) {

for(j=0;j<execTime[i];j++) {

cout<<" P"<<npro[i]<<" ";

}

}

}

cout<<endl;

for(i=0;i<tat+1;i++) {

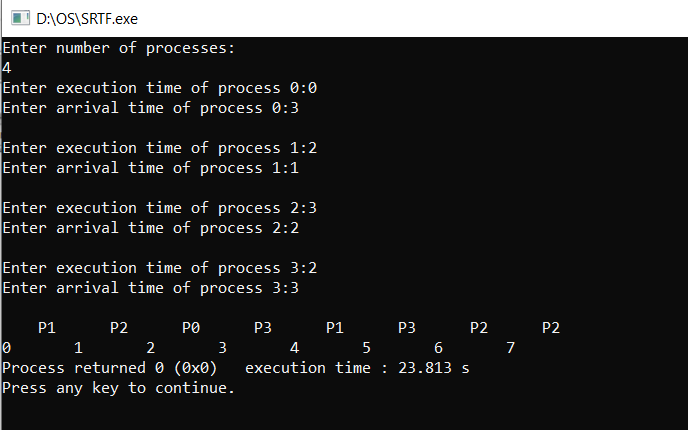
cout<<i<<"\t";

}

return 0;

}

OUTPUT-



Exercise (Based on Classical Scheduling problem)

**AIM:** To implement and understand the solutions of producer consumer and reader writer problems using seamaphore.

**THEORY:**

**PRODUCER AND CONSUMER PROBLEM**

*Problem Statement* – We have a buffer of fixed size. A producer can produce an item and can place in the buffer. A consumer can pick items and can consume them. We need to ensure that when a producer is placing an item in the buffer, then at the same time consumer should not consume any item. In this problem, buffer is the critical section.

To solve this problem, we need two counting semaphores – Full and Empty. “Full” keeps track of number of items in the buffer at any given time and “Empty” keeps track of number of unoccupied slots.

Initialization of semaphores –  
mutex = 1  
Full = 0 // Initially, all slots are empty. Thus full slots are 0  
Empty = n // All slots are empty initially

Solution for Producer –

do{

//produce an item

wait(empty);

wait(mutex);

//place in buffer

signal(mutex);

signal(full);

}while(true)

Solution for Consumer –

do{

wait(full);

wait(mutex);

// remove item from buffer

signal(mutex);

signal(empty);

// consumes item

}while(true)

C++ code Solution (only executable on linux based systems)

#include <iostream>

#include <pthread.h>

#include <semaphore.h>

#include <random>

#include <unistd.h>

using namespace std;

#define BUFFER\_SIZE 10

int buffer[BUFFER\_SIZE];

int index=0;

sem\_t full,empty;

pthread\_mutex\_t mutex;

void\* produce(void\* arg){

while(1){

sleep(1);

sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

int item = rand()%100;

buffer[index++] = item;

cout<<"Produced "<<item<<endl;

pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

}

}

void\* consume(void\* arg){

while(1){

sleep(1);

sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

int item = buffer[--index];

cout<<"Consumed "<<item<<endl;

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty);

}

}

int main(){

pthread\_t producer,consumer;

sem\_init(&empty,0,BUFFER\_SIZE);

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

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

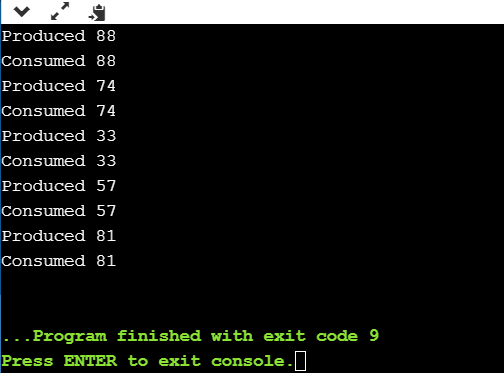
pthread\_create(&producer,NULL,produce,NULL);

pthread\_create(&consumer,NULL,consume,NULL);

pthread\_exit(NULL);

}

OUTPUT:



**READER AND WRITER PROBLEM**

*Problem parameters*:

* One set of data is shared among a number of processes
* Once a writer is ready, it performs its write. Only one writer may write at a time
* If a process is writing, no other process can read it
* If at least one reader is reading, no other process can write
* Readers may not write and only read

*Solution when Reader has the Priority over Writer:*

(priority means, no reader should wait if the share is currently opened for reading.)

Three variables are used: mutex, wrt, readcnt to implement solution

semaphore mutex, wrt;

semaphore mutex is used to ensure mutual exclusion when readcnt is updated i.e. when any reader enters or exit from the critical section and semaphore wrt is used by both readers and writers

int readcnt;

readcnt tells the number of processes performing read in the critical section, initially 0

**Functions for sempahore :**

– wait() : decrements the semaphore value.

– signal() : increments the semaphore value.

**Writer process:**

Writer requests the entry to critical section.

If allowed i.e. wait() gives a true value, it enters and performs the write. If not allowed, it keeps on waiting.

It exits the critical section.

do {

// writer requests for critical section

wait(wrt);

// performs the write

// leaves the critical section

signal(wrt);

} while(true);

**Reader process:**

Reader requests the entry to critical section.

If allowed:

it increments the count of number of readers inside the critical section. If this reader is the first reader entering, it locks the wrt semaphore to restrict the entry of writers if any reader is inside.

It then, signals mutex as any other reader is allowed to enter while others are already reading.

After performing reading, it exits the critical section. When exiting, it checks if no more reader is inside, it signals the semaphore “wrt” as now, writer can enter the critical section.

If not allowed, it keeps on waiting.

do {

// Reader wants to enter the critical section

wait(mutex);

// The number of readers has now increased by 1

readcnt++;

// there is atleast one reader in the critical section

// this ensure no writer can enter if there is even one reader

// thus we give preference to readers here

if (readcnt==1)

wait(wrt);

// other readers can enter while this current reader is inside

// the critical section

signal(mutex);

// current reader performs reading here

wait(mutex);   // a reader wants to leave

readcnt--;

// that is, no reader is left in the critical section,

if (readcnt == 0)

signal(wrt);         // writers can enter

signal(mutex); // reader leaves

} while(true);