****

Assignment 2

CSE-325

Sec-01

Submitted to:

Tanni Mittra

Senior Lecturer

Department of Computer Science & Engineering

Submitted by:

Name: Hasib Ar Rafiul Fahim

ID: 2019-1-60-036

Ans to the question no. 1

#include<bits/stdc++.h>

#include<iostream>

using namespace std;

#define SIZE 1000

int countjob=0;

int countready=0;

class StructureProcess{

public:

int processID;

string processName;

string processStatus;

int burst;

int arrivalTime;

int priority;

static int processCounter;

public:

Process(){}

void setProcess(string pN, int bur, int aT, int pr){

processName=pN;

burst=bur;

arrivalTime=aT;

priority=pr;

}

string getprocessName(){

return processName;

}

int getburst(){

return burst;

}

int getarrivalTime(){

return arrivalTime;

}

int getpriority(){

return priority;

}

};

StructureProcess JOB\_QUEUE[SIZE];

StructureProcess READY\_QUEUE[SIZE];

StructureProcess DEVICE\_QUEUE[SIZE];

int job\_queue\_front=0, job\_queue\_rear = 0;

int ready\_queue\_front=0, ready\_queue\_rear = 0;

int device\_queue\_front=0, device\_queue\_rear = 0;

int GANTT\_CHART[1000];

void initGanttChart(){

/\* for producing gantt chart \*/

}

void createProcess(){

string processName;

int burst,arrivalTime,priority;

cout<< "Enter process name, burst, arrival time and priority: \n";

cin>>processName;

cin>>burst;

cin>>arrivalTime;

cin>>priority;

JOB\_QUEUE[countjob].processName=processName;

JOB\_QUEUE[countjob].burst=burst;

JOB\_QUEUE[countjob].arrivalTime=arrivalTime;

JOB\_QUEUE[countjob].priority=priority;

countjob++;

job\_queue\_rear++;

cout<<"Process Transfered to job queue \n";

}

void showJobQueue(){

for(int i=job\_queue\_front;i<countjob;i++){

cout<<"Process number " <<i+1<<": ";

cout<<JOB\_QUEUE[i].processName <<"\n";

cout<<JOB\_QUEUE[i].burst<<"\n";

cout<<JOB\_QUEUE[i].arrivalTime<<"\n";

cout<<JOB\_QUEUE[i].priority<<"\n";

}

}

void showReadyQueue(){

for(int i=ready\_queue\_front;i<countready;i++){

cout<<"Process number " <<i+1<<": ";

cout<<READY\_QUEUE[i].processName <<"\n";

cout<<READY\_QUEUE[i].burst<<"\n";

cout<<READY\_QUEUE[i].arrivalTime<<"\n";

cout<<READY\_QUEUE[i].priority<<"\n";

}

}

void readyProcess(){

cout<<"Transferring process to ready queue \n " ;

for(int i=ready\_queue\_front;i<job\_queue\_rear;i++){

READY\_QUEUE[i].processName=JOB\_QUEUE[i].processName;

READY\_QUEUE[i].burst=JOB\_QUEUE[i].burst;

READY\_QUEUE[i].arrivalTime=JOB\_QUEUE[i].arrivalTime;

READY\_QUEUE[i].priority=JOB\_QUEUE[i].priority;

countready++;

ready\_queue\_rear++;

job\_queue\_front++;

}

}

void fcfs(){

int wt[countready], tat[countready], total\_wt = 0, total\_tat = 0;

wt[0] = 0;

for (int i = 1; i < countready ; i++ ) wt[i] = READY\_QUEUE[i-1].burst + wt[i-1] ;

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

tat[i] = READY\_QUEUE[i].burst + wt[i];

cout << "Processes "<< " Burst time "

<< " Waiting time " << " Turn around time\n";

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

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

cout << " " << i+1 << "\t\t" << READY\_QUEUE[i].burst <<"\t "

<< wt[i] <<"\t\t " << tat[i] <<endl;

}

cout << "Average waiting time = "

<< (float)total\_wt / (float)countready;

cout << "\nAverage turn around time = \n"

<< (float)total\_tat / (float)countready;

}

void sjf(){

int wt[countready], tat[countready], total\_wt = 0, total\_tat = 0;

int rt[countready];

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

rt[i] = READY\_QUEUE[i].burst;

int complete = 0, t = 0, minm = INT\_MAX;

int shortest = 0, finish\_time;

bool check = false;

while (complete != countready) {

for (int j = 0; j < countready; j++) {

if ((READY\_QUEUE[j].arrivalTime <= t) &&

(rt[j] < minm) && rt[j] > 0) {

minm = rt[j];

shortest = j;

check = true;

}

}

if (check == false) {

t++;

continue;

}

rt[shortest]--;

minm = rt[shortest];

if (minm == 0)

minm = INT\_MAX;

if (rt[shortest] == 0) {

complete++;

check = false;

finish\_time = t + 1;

wt[shortest] = finish\_time -

READY\_QUEUE[shortest].burst -

READY\_QUEUE[shortest].arrivalTime;

if (wt[shortest] < 0)

wt[shortest] = 0;

}

t++;

}

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

tat[i] = READY\_QUEUE[i].burst + wt[i];

cout << "Processes "

<< " Burst time "

<< " Waiting time "

<< " Turn around time\n";

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

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

cout << " " << READY\_QUEUE[i].processName << "\t\t"

<< READY\_QUEUE[i].burst << "\t\t " << wt[i]

<< "\t\t " << tat[i] << endl;

}

cout << "\nAverage waiting time = "

<< (float)total\_wt / (float)countready;

cout << "\nAverage turn around time = \n"

<< (float)total\_tat / (float)countready;

}

void srtf(){

int x[100];

int waiting[100],turnaround[100],completion[100];

int i,j,smallest,countt=0,time;

double avg=0,tt=0,end;

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

x[i]=READY\_QUEUE[i].burst;

READY\_QUEUE[9].burst=9999;

for(time=0; countt!=countready; time++)

{

smallest=9;

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

{

if(READY\_QUEUE[i].arrivalTime<=time && READY\_QUEUE[i].burst<READY\_QUEUE[smallest].burst && READY\_QUEUE[i].burst>0 )

smallest=i;

}

READY\_QUEUE[smallest].burst--;

if(READY\_QUEUE[smallest].burst==0)

{

countt++;

end=time+1;

completion[smallest] = end;

waiting[smallest] = end - READY\_QUEUE[smallest].arrivalTime - x[smallest];

turnaround[smallest] = end - READY\_QUEUE[smallest].arrivalTime;

}

}

cout<<"Process"<<"\t"<< "burst-time"<<"\t"<<"arrival-time" <<"\t"<<"waiting-time" <<"\t"<<"turnaround-time"<< "\t"<<"completion-time"<<endl;

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

{

cout<<"p"<<i+1<<"\t\t"<<x[i]<<"\t\t"<<READY\_QUEUE[i].arrivalTime<<"\t\t"<<waiting[i]<<"\t\t"<<turnaround[i]<<"\t\t"<<completion[i]<<endl;

avg = avg + waiting[i];

tt = tt + turnaround[i];

}

cout<<"\n\nAverage waiting time ="<<avg/countready;

cout<<" Average Turnaround time ="<<tt/countready<<endl;

}

void priorityScheduling(){

int wt[20],tat[20],i,j,total=0,pos,temp,avg\_wt,avg\_tat;

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

{

pos=i;

for(j=i+1;j<countready;j++)

{

if(READY\_QUEUE[j].priority<READY\_QUEUE[pos].priority)

pos=j;

}

temp=READY\_QUEUE[i].priority;

READY\_QUEUE[i].priority=READY\_QUEUE[pos].priority;

READY\_QUEUE[pos].priority=temp;

temp=READY\_QUEUE[i].burst;

READY\_QUEUE[i].burst=READY\_QUEUE[pos].burst;

READY\_QUEUE[pos].burst=temp;

}

wt[0]=0;

for(i=1;i<countready;i++)

{

wt[i]=0;

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

wt[i]+=READY\_QUEUE[j].burst;

total+=wt[i];

}

avg\_wt=total/countready;

total=0;

cout<<"\nBurst Time \tWaiting Time\tTurnaround Time";

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

{

tat[i]=READY\_QUEUE[i].burst+wt[i];

total+=tat[i];

cout<<READY\_QUEUE[i].burst<<"\t\t "<<wt[i]<<"\t\t\t"<<tat[i];

}

avg\_tat=total/countready;

cout<<"\n\nAverage Waiting Time="<<avg\_wt;

cout<<"\nAverage Turnaround Time="<<avg\_tat;

}

void preemptivePriorityScheduling(){

int x[10];

int waiting[10],turnaround[10],completion[10];

int i,j,smallest,countt=0,time;

double avg=0,tt=0,endd;

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

x[i]=READY\_QUEUE[i].burst;

READY\_QUEUE[9].priority=-1;

for(time=0; countt!=countready; time++)

{

smallest=9;

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

{

if(READY\_QUEUE[i].arrivalTime<=time && READY\_QUEUE[i].priority>READY\_QUEUE[smallest].priority && READY\_QUEUE[i].burst>0 )

smallest=i;

}

READY\_QUEUE[smallest].burst--;

if(READY\_QUEUE[smallest].burst==0)

{

countt++;

endd=time+1;

completion[smallest] = endd;

waiting[smallest] = endd - READY\_QUEUE[smallest].arrivalTime - x[smallest];

turnaround[smallest] = endd - READY\_QUEUE[smallest].arrivalTime;

}

}

cout<<"Process"<<"\t"<< "burst-time"<<"\t"<<"arrival-time" <<"\t"<<"waiting-time" <<"\t"<<"turnaround-time"<< "\t"<<"completion-time"<<"\t"<<"Priority"<<endl;

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

{

cout<<"p"<<i+1<<"\t\t"<<x[i]<<"\t\t"<<READY\_QUEUE[i].arrivalTime<<"\t\t"<<waiting[i]<<"\t\t"<<turnaround[i]<<"\t\t"<<completion[i]<<"\t\t"<<READY\_QUEUE[i].priority<<endl;

avg = avg + waiting[i];

tt = tt + turnaround[i];

}

cout<<"\n\nAverage waiting time ="<<avg/countready;

cout<<" Average Turnaround time ="<<tt/countready<<endl;

}

void roundRobin(){

int wt[countready], tat[countready], total\_wt = 0, total\_tat = 0;

int quantum=2;

int rem\_bt[countready];

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

rem\_bt[i] = READY\_QUEUE[i].burst;

int t = 0;

while (1)

{

bool done = true;

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

{

if (rem\_bt[i] > 0)

{

done = false;

if (rem\_bt[i] > quantum)

{

t += quantum;

rem\_bt[i] -= quantum;

}

else

{

t = t + rem\_bt[i];

wt[i] = t - READY\_QUEUE[i].burst;

rem\_bt[i] = 0;

}

}

}

if (done == true)

break;

}

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

tat[i] = READY\_QUEUE[i].burst + wt[i];

cout << "Processes "<< " Burst time "

<< " Waiting time " << " Turn around time\n";

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

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

cout << " " << i+1 << "\t\t" << READY\_QUEUE[i].burst <<"\t "

<< wt[i] <<"\t\t " << tat[i] <<endl;

}

cout << "Average waiting time = "

<< (float)total\_wt / (float)countready;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)countready;

}

void executeProcess(){

int choice;

cout << "Enter Choice: ";

cin >> choice;

switch(choice){

case 1:

fcfs();

break;

case 2:

sjf();

break;

case 3:

srtf();

break;

case 4:

priorityScheduling();

break;

case 5:

preemptivePriorityScheduling();

break;

case 6:

roundRobin();

break;

default:

cout << "Unrecognized Option. " << endl;

}

}

void reexecuteProcess(){

readyProcess();

executeProcess();

}

int main(){

string command;

do{

cout << "> ";

cin >> command;

if(command=="create-process"){

createProcess();

}

else if(command=="ready-process"){

readyProcess();

}

else if(command=="show-job-queue"){

showJobQueue();

}

else if(command=="show-ready-queue"){

showReadyQueue();

}

else if(command=="execute-process"){

executeProcess();

}

else if(command=="reexecute-process"){

reexecuteProcess();

}

else if(command=="exit"){

cout << "Thank You." << endl;

exit(0);

}

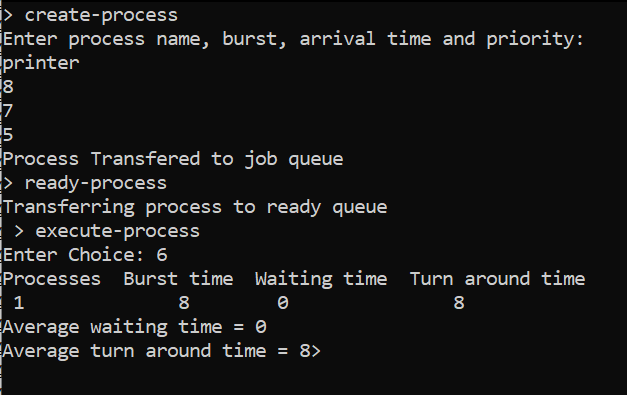
else{

cout << "Unrecognized Command. Try again." << endl;

}

} while(true);

}



Ans to the question no. 2

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#include <assert.h>

#define Max\_stu\_size 2000

int done=0;

int totalRequests=0;

int totalSessions=0;

int tutoringNow=0;

void \*student\_thread(void \*student\_id);

void \*tutor\_thread(void \*tutor\_id);

void \*coordinator\_thread();

int student\_num=0;

int tutor\_num=0;

int help\_num=0;

int chair\_num=0;

int occupied\_chair\_num=0;

int newArrivedStudentQueue[Max\_stu\_size];

int tutorFinishedQueue[Max\_stu\_size];

int priorityQueue[Max\_stu\_size][2];

int student\_priority[Max\_stu\_size];

int student\_ids[Max\_stu\_size];

int tutor\_ids[Max\_stu\_size];

sem\_t sem\_student;

sem\_t sem\_coordinator;

pthread\_mutex\_t seatLock;

pthread\_mutex\_t queueLock;

pthread\_mutex\_t tutorFinishedQueueLock;

int main(int argc, const char \* argv[]) {

if (argc != 5){

printf("Usage: <# of Students> <# of tutors> <# of chairs> <# of help>\n");

exit(-1);

}

student\_num=atoi(argv[1]);

tutor\_num=atoi(argv[2]);

chair\_num=atoi(argv[3]);

help\_num=atoi(argv[4]);

if(student\_num > Max\_stu\_size || tutor\_num > Max\_stu\_size){

printf("Max student number is: %d; Max tutor number is: %d\n", Max\_stu\_size, Max\_stu\_size);

exit(-1);

}

int i;

for(i=0;i<student\_num;i++){

newArrivedStudentQueue[i]=-1;

tutorFinishedQueue[i]=-1;

priorityQueue[i][0]=-1;

priorityQueue[i][1]=-1;

student\_priority[i]=0;

}

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

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

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

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

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

pthread\_t students[student\_num];

pthread\_t tutors[tutor\_num];

pthread\_t coordinator;

assert(pthread\_create(&coordinator,NULL,coordinator\_thread,NULL)==0);

for(i = 0; i < student\_num; i++)

{

student\_ids[i] = i + 1;

assert(pthread\_create(&students[i], NULL, student\_thread, (void\*) &student\_ids[i])==0);

}

for(i = 0; i < tutor\_num; i++)

{

tutor\_ids[i] = i + student\_num + 1;

assert(pthread\_create(&tutors[i], NULL, tutor\_thread, (void\*) &tutor\_ids[i])==0);

}

pthread\_join(coordinator, NULL);

for(i =0; i < student\_num; i++)

{

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

}

for(i =0; i < tutor\_num; i++)

{

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

}

return 0;

}

void \*student\_thread(void \*student\_id){

int id\_student=\*(int\*)student\_id;

while(1){

if(student\_priority[id\_student-1]>=help\_num) {

pthread\_mutex\_lock(&seatLock);

done++;

pthread\_mutex\_unlock(&seatLock);

sem\_post(&sem\_student);

pthread\_exit(NULL);

}

float programTime=(float)(rand()%200)/100;

usleep(programTime);

pthread\_mutex\_lock(&seatLock);

if(occupied\_chair\_num>=chair\_num){

printf("St: Student %d found no empty chair. Will try again later.\n",id\_student);

pthread\_mutex\_unlock(&seatLock);

continue;

}

occupied\_chair\_num++;

totalRequests++;

newArrivedStudentQueue[id\_student-1]=totalRequests;

printf("St: Student %d takes a seat. Empty chairs = %d\n",id\_student,chair\_num-occupied\_chair\_num);

pthread\_mutex\_unlock(&seatLock);

sem\_post(&sem\_student);

while(tutorFinishedQueue[id\_student-1]==-1);

printf("St: Student %d received help from Tutor %d.\n",id\_student,tutorFinishedQueue[id\_student-1]-student\_num);

pthread\_mutex\_lock(&tutorFinishedQueueLock);

tutorFinishedQueue[id\_student-1]=-1;

pthread\_mutex\_unlock(&tutorFinishedQueueLock);

pthread\_mutex\_lock(&seatLock);

student\_priority[id\_student-1]++;

pthread\_mutex\_unlock(&seatLock);

}

}

void \*tutor\_thread(void \*tutor\_id){

int id\_tutor=\*(int\*)tutor\_id;

int studentTutoredTimes;

int studentSequence;

int id\_student;

while(1){

if(done==student\_num){

pthread\_exit(NULL);

}

studentTutoredTimes=help\_num-1;

studentSequence=student\_num\*help\_num+1;

id\_student=-1;

sem\_wait(&sem\_coordinator);

pthread\_mutex\_lock(&seatLock);

int i;

for(i=0;i<student\_num;i++){

if(priorityQueue[i][0]>-1 && priorityQueue[i][0]<=studentTutoredTimes

&& priorityQueue[i][1]<studentSequence){

studentTutoredTimes=priorityQueue[i][0];

studentSequence=priorityQueue[i][1];

id\_student=student\_ids[i];

}

}

if(id\_student==-1) {

pthread\_mutex\_unlock(&seatLock);

continue;

}

priorityQueue[id\_student-1][0]=-1;

priorityQueue[id\_student-1][1]=-1;

occupied\_chair\_num--;

tutoringNow++;

pthread\_mutex\_unlock(&seatLock);

float tutorTime=(float)(rand()%200)/1000;

usleep(tutorTime);

pthread\_mutex\_lock(&seatLock);

tutoringNow--;

totalSessions++;

printf("Tu: Student %d tutored by Tutor %d. Students tutored now = %d. Total sessions tutored = %d\n",id\_student,id\_tutor-student\_num,tutoringNow,totalSessions);

pthread\_mutex\_unlock(&seatLock);

pthread\_mutex\_lock(&tutorFinishedQueueLock);

tutorFinishedQueue[id\_student-1]=id\_tutor;

pthread\_mutex\_unlock(&tutorFinishedQueueLock);

}

}

void \*coordinator\_thread(){

while(1){

if(done==student\_num){

int i;

for(i=0;i<tutor\_num;i++){

sem\_post(&sem\_coordinator);

}

pthread\_exit(NULL);

}

sem\_wait(&sem\_student);

pthread\_mutex\_lock(&seatLock);

int i;

for(i=0;i<student\_num;i++){

if(newArrivedStudentQueue[i]>-1){

priorityQueue[i][0]=student\_priority[i];

priorityQueue[i][1]=newArrivedStudentQueue[i];

printf("Co: Student %d with priority %d in the queue. Waiting students now = %d. Total requests = %d\n",student\_ids[i],student\_priority[i],occupied\_chair\_num,totalRequests);

newArrivedStudentQueue[i]=-1;

sem\_post(&sem\_coordinator);

}

}

pthread\_mutex\_unlock(&seatLock);

}



Answer to the question no. 3

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

const int PAPER = 1;

const int TOBACCO = 2;

const int MATCHES = 3;

typedef struct Smoker{

char smokerName[10];

int item;

} smoker;

typedef struct Agent{

char agentName[10];

int item1;

int item2;

} agent;void printAgents(agent\* ag);

void printSmoker(agent\* ag, smoker\* sm);

char\* getIngredientName(int choice);

int main(){

agent ag;

smoker sm[3] = {{"Paper", 1}, {"Tobacco", 2}, {"Matches", 3}};

int choice = 0;

strcpy(ag.agentName, "Agent");

ag.item1 = 0;

ag.item2 = 0;

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

printAgents(&ag);

printSmoker(&ag, sm);

}

return 0;

}

void printAgents(agent\* ag){

ag->item1 = 1 + rand() % 3;

while(1){

ag->item2 = 1 + rand() % 3;

if(ag->item1 != ag->item2)

break;

}

printf("\nAgent provides ingredients - %s, %s\n", getIngredientName(ag->item1),getIngredientName(ag->item2));

}

void printSmoker(agent\* ag, smoker\* sm){

int i = 0;

while(sm[i].item != 0){

if((sm[i].item != ag->item1) && (sm[i].item!= ag->item2)){

printf("Smoker with %s is smoking his cigarette\n", sm[i].smokerName);

ag->item1 = 0;

ag->item2 = 0;

break;

}

i++;

}

}char\* getIngredientName(int choice)

{

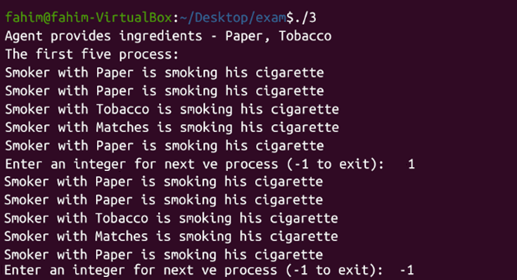
if(choice == PAPER) return "Paper";

else if(choice == TOBACCO) return "Tobacco";

else if(choice == MATCHES) return "Matches";

else return "None";

}



Ans to the question no. 4

Regressive round-robin scheduler favors the CPU bound types of processes. That's because during computation, a CPU bound processes uses its entire quantum time, which makes the processor stay active at all times, and its priority level is boosted.

Regressive round-robin scheduler does not favor the I/O bound processes because they use up the whole time waiting for the I/O device, which gives them lower priority, and the process depends on speed of the processor cores of the I/O device.

It favors CPU bound type of a process since those processes actively consume CPU time.

Ans to the question no. 5

Here:

Let’s assume, sem\_t m1=1, m2=1, m3=0, m4=0;

**Process-1**

While(true){

Down(m1);

Up(m3);

Up(m1);

}

**Process-2**

While(true){

Down(m2);

Up(m4);

Up(m2);

}

**Process-3**

While(true){

Down(m3);

Down(m4);

Up(m4);

Up(m3);

}

**Process-4**

While(true){

Down(m4);

Up(m4);

}