Text

Description automatically generated

**ASSIGNMENT-2 (MID-2)**

Course Title: Operating System

Course Code: CSE325

Section: 01

Submitted to:

Tanni Mittra

Senior Lecturer

Department of Computer Science & Engineering

Submitted by:

Name: Adri Saha

ID: 2019-1-60-024

Date of submission: 12th May 2021

**Ques no 1.**

You know the core concepts behind process management of operating system. OS handles creation of processes, manage different queues such as job, ready and I/O queues. It also uses different process scheduling algorithms to execute processes by maintaining proper scheduling schemes. Now your task is to build a process simulator Your simulator will have following menu page:

|  |
| --- |
| Create Process |
| Ready Process |
| Show Job Queue |
| Show Ready Queue |
| Execute Process |
| Re-Execute Process |
| Exit |

Your simulator will receive a command as input and perform the task as per the command. You should follow the instructions stated below:

a) Whenever you create a process, your process will have the following properties:

* processID;
* processName;
* processStatus;
* burst;
* arrivalTime;
* priority;
* processCounter;

b) For job queue, ready queue and device queue you can use advanced data structures like Linked List or Vector

c) Ready Process function transferring processes from Job Queue to Ready Queue. Here once process is ready, shift the process from new to ready state

d) Show ready queue option is used to print ready queue

e) Inside the execute process function you have to select different scheduling algorithm. For each of the scheduling algorithm create appropriate gantt chart. Create submenu inside execute process function.

"CPU Scheduling Algorithm Selection"

(1) First-Come-First-Served (FCFS)

(2) Shortest-Job-First (SJF)

(3) Preemptive SJF (shortest-remaining-time-first)

(4) Priority-based Scheduling

(5) Preemptive Priority-based Scheduling

(6) Round-Robin Scheduling

f) Inside Re-execute process you have to set process status as ready of ready queue and start execution again.

g) A sample code is assigned for your process simulator.

Ans to the ques no-1

#include<bits/stdc++.h>

#include <queue>

#include <algorithm>

#include <iomanip>

#include <queue>

#include <list>

#include <iterator>

using namespace std;

struct Process

{

public:

int processID;

string processName;

string processStatus;

int burst;

int arrivalTime;

int priority;

int processCounter;

public:

Process() {}

void setProcess(string processName, int burst, int arrivalTime, int priority)

{

this->processName=processName;

this->burst=burst;

this->arrivalTime=arrivalTime;

this->priority=priority;

}

// design appropriate getter and setter methods

};

int SIZE=10000,gin=0;

/\* you may use other advanced data structures such as Linked List or Vector \*/

Process JOB\_QUEUE[10000];

Process READY\_QUEUE[10000];

Process DEVICE\_QUEUE[10000];

/\* keeping track of processes in the queue \*/

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

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

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

int GANTT\_CHART[1000];

void initGanttChart()

{

/\* for producing gantt chart \*/

}

/\* initiating a process \*/

void createProcess()

{

int n;

cout<<"\n-------Process Creation-----\n";

cout<<"How many process you have: ";

cin>>n;

Process p[n];

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

{

p[i].processID=i+1;

cout<<"Process\_name\t ProcessStatus\t processCounter\t arrivalTime\t Burst\_time\t Priority\n";

cin>>p[i].processName>>p[i].processStatus>>p[i].processCounter>>p[i].arrivalTime>>p[i].burst>>p[i].priority;

JOB\_QUEUE[job\_queue\_rear].setProcess(p[i].processName,p[i].burst,p[i].arrivalTime,p[i].priority);

JOB\_QUEUE[job\_queue\_rear].processName=p[i].processName;

JOB\_QUEUE[job\_queue\_rear].processID=p[i].processID;

JOB\_QUEUE[job\_queue\_rear].processCounter=p[i].processCounter;

job\_queue\_rear++;

}

/\* Write code to create process and put the process into the job queue \*/

}

/\* printing job queue \*/

void showJobQueue()

{

cout<<"P\_Id\t\tName\t\tStatus\t\tArrival\_Time\t\tBurst\_Time\t\t\n\n";

for(Process i: JOB\_QUEUE)

{

cout<<"\n-------JOB QUEUE-----\n";

if(i.processCounter!=0)

{

cout<<i.processID<<"\t"<<i.processName<<"\t"<<i.processStatus<<"\t"<<i.arrivalTime<<"\t"<<i.burst<<"\t"<<endl;

}

}

}

/\* printing ready queue \*/

void showReadyQueue()

{

for(int i=0; i<ready\_queue\_rear; i++)

{

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

}

cout<<endl;

}

/\* once user is ready, shifting process from new -> ready state \*/

void readyProcess()

{

for(int i=0; i<ready\_queue\_rear; i++)

{

cout<<DEVICE\_QUEUE[i].processName<<" ";

}

/\* displaying gantt chart \*/

void displayGanttChart(int clock)

{

for(int i=0; i<ready\_queue\_rear; i++)

{

cout<<GANTT\_CHART[i]<<" ";

}

cout<<endl;

}

void fcfs()

{

createProcess();

for(int i=0; i<job\_queue\_rear-1; i++ )

{

for(int j=0; j<job\_queue\_rear-i-1; j++)

{

if( JOB\_QUEUE[j].burst>JOB\_QUEUE[j+1].burst )

{

Process temp=JOB\_QUEUE[j];

JOB\_QUEUE[j]=JOB\_QUEUE[j+1];

JOB\_QUEUE[j+1]=temp;

}

}

}

for(int i=0; i<ready\_queue\_rear; i++)

{

GANTT\_CHART[i]=READY\_QUEUE[i].processID;

}

displayGanttChart(gin);

gin++;

int sum=0;

cout<<"\n-------FCFS is-----\n";

cout<<"Id\t\tName\t\tstatus\t\tArrivalTime\t\tBurst\t\tSum\n\n";

for(int i=0; i<100000; i++)

{

if(JOB\_QUEUE[i].processCounter!=0)

{

sum+=JOB\_QUEUE[i].burst;

cout<<JOB\_QUEUE[i].processID<<"\t"<<JOB\_QUEUE[i].processName<<"\t"<<JOB\_QUEUE[i].processStatus<<"\t"<<JOB\_QUEUE[i].arrivalTime<<"\t"<<JOB\_QUEUE[i].burst<<" "<<sum<<endl;

}

}

cout<<endl;

}

void sjf()

{

createProcess();

for(int i=0; i<job\_queue\_rear-1; i++ )

{

for(int j=0; j<job\_queue\_rear-i-1; j++)

{

if(JOB\_QUEUE[j].arrivalTime>JOB\_QUEUE[j+1].arrivalTime )

{

Process temp=JOB\_QUEUE[j];

JOB\_QUEUE[j]=JOB\_QUEUE[j+1];

JOB\_QUEUE[j+1]=temp;

}

}

}

for(int i=0; i<ready\_queue\_rear; i++)

{

GANTT\_CHART[i]=READY\_QUEUE[i].processID;

}

displayGanttChart(gin);

gin++;

int sum=0;

for(int i=0; i<100000; i++)

{

if( JOB\_QUEUE[i].processCounter!=0 )

{

sum+=JOB\_QUEUE[i].burst;

cout<<"\n-------SJF is-----\n";

cout<<JOB\_QUEUE[i].processID<<"\t"<<JOB\_QUEUE[i].processName<<"\t\t"<<JOB\_QUEUE[i].processStatus<<"\t"<<JOB\_QUEUE[i].arrivalTime<<"\t"<<JOB\_QUEUE[i].burst<<"\t"<<sum<<endl;

}

}

cout<<endl;

}

bool isBurst()

{

for(int i=0; i<ready\_queue\_rear; i++)

{

if( READY\_QUEUE[i].burst>0 )

{

return false;

}

}

return true;

}

void sorting(double t)

{

for(int i=0; i<ready\_queue\_rear; i++)

{

for(int j=i; j<ready\_queue\_rear; j++)

{

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

{

Process temp=READY\_QUEUE[i];

READY\_QUEUE[i]=READY\_QUEUE[j];

READY\_QUEUE[j]=temp;

}

}

}

}

void sortTime()

{

for(int i=0; i<ready\_queue\_rear; i++)

{

for(int j=i; j<ready\_queue\_rear; j++)

{

if(JOB\_QUEUE[i].arrivalTime>JOB\_QUEUE[j].arrivalTime)

{

Process temp=JOB\_QUEUE[i];

JOB\_QUEUE[i]=JOB\_QUEUE[j];

JOB\_QUEUE[j]=temp;

}

}

}

}

void srtf()

{

createProcess();

sortTime();

while(!isBurst())

{

sorting( READY\_QUEUE[0].arrivalTime );

GANTT\_CHART[gin++]=READY\_QUEUE[0].processID;

READY\_QUEUE[0].burst--;

}

}

void priorityScheduling()

{

createProcess();

int tem;

Process t;

cout<<"\n-------Priority Scheduling-----\n";

cout<<"id\t\tname\t\tstatus\t\tarrivalTime\t\tBurst\t\tsum"<<endl;

for(int i=0; i<ready\_queue\_rear; i++)

{

tem=i;

for(int j=i+1; j<ready\_queue\_rear; j++)

{

if(JOB\_QUEUE[tem].priority>JOB\_QUEUE[j].priority )

{

tem=j;

}

}

t=JOB\_QUEUE[i];

JOB\_QUEUE[i]=JOB\_QUEUE[tem];

JOB\_QUEUE[tem]=t;

}

for(int i=0; i<ready\_queue\_rear; i++)

{

GANTT\_CHART[i]=READY\_QUEUE[i].processID;

}

displayGanttChart(gin);

int sum=0;

for(int i=0; i<100000; i++)

{

if( JOB\_QUEUE[i].processCounter!=0 )

{

sum+=JOB\_QUEUE[i].burst;

cout<<JOB\_QUEUE[i].processID<<" "<<JOB\_QUEUE[i].processName<<" "<<JOB\_QUEUE[i].processStatus<<" "<<JOB\_QUEUE[i].arrivalTime<<" "<<JOB\_QUEUE[i].burst<<" "<<sum<<endl;

}

}

cout<<endl;

}

void sortPr(int ta,int prio)

{

for(int i=0; i<ready\_queue\_rear; i++)

{

for(int j=i; j<ready\_queue\_rear; j++)

{

if( JOB\_QUEUE[i].burst>JOB\_QUEUE[j].burst && JOB\_QUEUE[i].arrivalTime<=ta && JOB\_QUEUE[i].priority>prio )

{

Process temp=JOB\_QUEUE[i];

JOB\_QUEUE[i]=JOB\_QUEUE[j];

JOB\_QUEUE[j]=temp;

}

}

}

}

void preemptivePriorityScheduling()

{

createProcess();

gin=0;

sortTime();

while(!isBurst())

{

sortPr(JOB\_QUEUE[0].arrivalTime,JOB\_QUEUE[0].priority);

GANTT\_CHART[gin++]=JOB\_QUEUE[0].processID;

JOB\_QUEUE[0].burst--;

}

}

void roundRobin()

{

createProcess();

int n;

int tq;

struct Process p[100];

float avg\_turnaround\_time;

float avg\_waiting\_time;

float avg\_response\_time;

float cpu\_utilisation;

int total\_turnaround\_time = 0;

int total\_waiting\_time = 0;

int total\_response\_time = 0;

int total\_idle\_time = 0;

float throughput;

int burst\_remaining[100];

int idx;

for(int i=0; i<ready\_queue\_rear; i++)

{

for(int j=i; j>=1; j--)

{

if( READY\_QUEUE[j].arrivalTime<READY\_QUEUE[j-1].arrivalTime )

{

Process temp=READY\_QUEUE[j];

READY\_QUEUE[j]=READY\_QUEUE[j-1];

READY\_QUEUE[j-1]=temp;

}

else if(READY\_QUEUE[j].arrivalTime==READY\_QUEUE[j-1].arrivalTime)

{

if( READY\_QUEUE[j].burst<READY\_QUEUE[j-1].burst )

{

Process temp=READY\_QUEUE[j];

READY\_QUEUE[j]=READY\_QUEUE[j-1];

READY\_QUEUE[j-1]=temp;

}

}

}

// sort(READY\_QUEUE,READY\_QUEUE+ready\_queue\_rear,compare1);

queue<int> q;

int current\_time = 0;

q.push(0);

int total\_Time=0;

total\_Time+=READY\_QUEUE[0].arrivalTime+READY\_QUEUE[0].burst;

for(int i=1; i<ready\_queue\_rear; i++)

{

if( READY\_QUEUE[i].arrivalTime>total\_Time )

{

total\_Time=READY\_QUEUE[i].arrivalTime;

total\_Time+=READY\_QUEUE[i].burst;

}

}

}

}

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()

{

Process JOB\_QUEUE[100000];

Process READY\_QUEUE[100000];

Process DEVICE\_QUEUE[100000];

}

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);

}

/\*

execute-process

1

4

p yes 1 1 3 1

q yes 2 2 4 3

r yes 3 1 2 2

s yes 4 4 4 4

\*/

Text

Description automatically generated

Figure 1: Output of ques no 1(FCFS)

Text

Description automatically generated

A picture containing text

Description automatically generated

A picture containing text

Description automatically generated

Figure 1: Output of ques no 1

**Ques no 2.**

The computer science department of EWU runs a programming club to help undergraduate students with their programming assignments. The club has a coordinator and several tutors to assist the students. The waiting area of the center has several chairs. Initially, all the chairs are empty. The coordinator is waiting for the students to arrive. The tutors are either waiting for the coordinator to notify that there are students waiting or they are busy tutoring. The tutoring area is different from the waiting area. A student, while programming for his project, decides to go to a club to get help from a tutor. After arriving at the club, the student sits in an empty chair in the waiting area and waits to be called for tutoring. If no chairs are available, the student will go back to programming and come back to the club later. Once a student arrives, the coordinator queues the student based on the student’s priority, and then the coordinator notifies an idle tutor. A tutor, once woken up, finds the student with the highest priority and begins tutoring. A tutor after helping a student, waits for the next student. A student after receiving help from a tutor goes back to programming. The priority of a student is based on the number of times the student has visited the club. A student visiting the club for the first time gets the highest priority. In general, a student visiting for the ith time has a priority higher than the priority of the student visiting for the kth time for any k > i. If two students have the same priority, then the student who came first has a higher priority.

Using POSIX threads, mutex locks, and semaphores implement a solution that synchronizes the activities of the coordinator, tutors, and the students. The total number of students, the number of tutors, the number of chairs, and the number of times a student seeks a tutor’s help are passed as command line arguments. Once a student thread takes the required number of help from the tutors, it should terminate. Once all the student threads are terminated, the tutor threads, the coordinator thread, and the main program should be terminated. Your program should work for any number of students, tutors, chairs and help sought. Allocate memory for data structures dynamically based on the input parameter(s).

**Ans to the ques no-2**

**Explanation:** Here, the following is the output of a student thread (x and y are ids): St: If a student x sits down. The number of vacant seats equals the number of empty chairs. or student x couldn't find an empty seat. Student x was then assisted by Tutor y. The following is the output of coordinator threads (where x is the id and p is the priority): Student x has priority p in the queue, according to the coordinator. The number of students who are not waiting is equal to the number of students who are currently waiting. Total requests = the total number of tutoring requests (notifications) received so far. After tutoring a student, the output of a tutor thread. Student x is being tutored by Tutor y. Students who are currently being tutored = the number of students who are currently being tutored. As a result, the software does not produce anything else.

#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <string.h>

#include <pthread.h>

#include <semaphore.h>

#include <assert.h>

#define Max\_stu\_size 500

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

int done=0;

int totalRequests=0;

int totalSessions=0;

int tutoringNow=0;

int student\_num=0;

int tutor\_num=0;

int help\_num=0,chair\_num=0;

int occupied\_chair\_num=0;

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

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

void \*coordinator\_thread();

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;

void \*thread\_stu(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;

sleep(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;

pthread\_mutex\_unlock(&seatLock);

sem\_post(&sem\_student);

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

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 \*thread\_tutor(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;

sleep(tutorTime);

pthread\_mutex\_lock(&seatLock);

tutoringNow--;

totalSessions++;

pthread\_mutex\_unlock(&seatLock);

pthread\_mutex\_lock(&tutorFinishedQueueLock);

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

pthread\_mutex\_unlock(&tutorFinishedQueueLock);

}

}

void \*thread\_Coordinate()

{

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];

newArrivedStudentQueue[i]=-1;

sem\_post(&sem\_coordinator);

}

}

pthread\_mutex\_unlock(&seatLock);

}

}

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

{

if (argc != 5)

{

printf("----Usage----\n <Number of Students> \t <Number of tutors> <number of chairs> <Number of help>\n");

printf("Compilation complete\n");

exit(0);

}

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)

{

exit(0);

}

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,thread\_Coordinate,NULL)==0);

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

{

student\_ids[i] = i + 1;

assert(pthread\_create(&students[i], NULL, thread\_stu, (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, thread\_tutor, (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);

}

exit(0);

}

Text

Description automatically generated

Figure 2: Output of ques no 2

**Ques no 3.**

There are three smokers and one agent. Each smoker continuously rolls a cigarette and then smokes it. But to roll and smoke a cigarette, the smoker needs three ingredients: tobacco, paper, and matches. One of the smokers has paper, another has tobacco, and the third has matches. The agent has infinite supply of all three materials. The agent places two of the ingredients on the table. The smoker who has the remaining ingredient then makes and smokes the cigarette, signaling the agent on completion. The agent then puts out another two of the three ingredients and the cycle repeats.

You must output the progress of the processes. E.g. when the agent places the two ingredients you should print that information. When a smoker gets to roll and smoke a cigarette, output which smoker got the ingredients and so-on. Show the run of your processes for a while until around 10 cigarettes are rolled.

Ans to the ques no-3

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <termios.h>

#include <sys/wait.h>

int getch(void)

{

struct termios oldattr, newattr;

int ch;

tcgetattr( STDIN\_FILENO, &oldattr );

newattr = oldattr;

newattr.c\_lflag &= ~( ICANON | ECHO );

tcsetattr( STDIN\_FILENO, TCSANOW, &newattr );

ch = getchar();

tcsetattr( STDIN\_FILENO, TCSANOW, &oldattr );

return ch;

}

enum Ingredients

{

None,

Paper,

Tobacco,

Matches

};

typedef struct smoker

{

char smokerID[25];

int Item;

} SMOKER;

typedef struct agent

{

char AgentID[25];

int Item1;

int Item2;

} AGENT;

char\* GetIngredientName(int Item)

{

if(Item == Paper)

return "Paper";

else if(Item == Tobacco)

return "Tobacco";

else if(Item==Matches)

return "Matches";

}

void GetAgentIngredients(AGENT\* agent)

{

agent->Item1=(rand()%3)+1;

while(1)

{

agent->Item2=(rand()%3)+1;

if(agent->Item1 != agent->Item2)

break;

}

printf("\nSmoker got Ingredients until 10 cigarettes are rolled- %s,%s\n\n",GetIngredientName(agent->Item1),GetIngredientName(agent->Item2));

}

void GiveIngredientToSmoker(AGENT\* agent,SMOKER\* smoker)

{

int index=0;

while(smoker[index].Item!='\0')

{

if((smoker[index].Item!=agent->Item1)&&(smoker[index].Item!=agent->Item2));

{

printf("\nSmoker-\%s\"-is smoking his cigarette\n\n",smoker[index].smokerID);

agent->Item1=None;

agent->Item2=None;

break;

}

index++;

}

}

void main()

{

AGENT agent;

SMOKER smoker[4]= {{"Smoker with paper",Paper},{"Smoker with tobacco",Tobacco},{"Smoker with Matches",Matches},{"\0",None}};

int userChoice=0;

strcpy(agent.AgentID,"Agent");

agent.Item1=None;

agent.Item2=None;

while(1)

{

GetAgentIngredients(&agent);

GiveIngredientToSmoker(&agent,smoker);

printf("Press ESC to exit or any key to continue\t");

userChoice=getch();

//scanf("%d",&userChoice);

if(userChoice==10)

break;

}

}

Text

Description automatically generated

Figure 3: Output of ques no 3

**Ques no 4.**

A variation of the round-robin scheduler is the regressive round-robinscheduler. This scheduler assigns each process a time quantum and a priority. The initial value of a time quantum is 50 milliseconds. However, every time a process has been allocated the CPU and uses its entire time quantum (does not block for I/O), 10 milliseconds is added to its time quantum, and its priority level is boosted. (The time quantum for a process can be increased to a maximum of 100 milliseconds.) When a process blocks before using its entire time quantum, its time quantum is reduced by 5 milliseconds, but its priority remains the same. What type of process (CPU-bound or I/O-bound) does the regressive round-robin scheduler favor? Explain.

Ans to the ques no-4

Regressive round robin prefers CPU-bound process forms. Round Robin is a process scheduling algorithm that uses a preemptive approach. Each process is given a fixed amount of time to complete, which is referred to as a quantum. After a process has run for a set amount of time, it is preempted, and another process runs for the same amount of time. Round-robin scheduling allows each ready task to run in a cyclic queue for a set period of time. This algorithm also allows for process execution without starvation. To save the states of preempted systems, context switching is used.

This is since scheduler can reward certain processes up to 5 times. The reasoning is that if a CPU bound process consumes the entire slice, it was performing computation and the processor was “active” the entire time. It's likely that a CPU-bound process will finish soon, in which case the scheduler will reward it. Here, we see the initial time quantum is 50ms. It is increasing 10ms to its time quantum in every time of process. And with that the priority level is also boosted. But time quantum can’t be exceeding 100ms. A process can be block in mid if burst time is less than the modified time quantum. As the process ends before reaching the time quantum. When a process blocks before using its entire time quantum, the time quantum reduced 5ms to adjust & prevent CPU loss. But still the priority remains same. To assume the situation, we can think like that, initial TQ=50ms.

|  |  |  |  |
| --- | --- | --- | --- |
| Processes alloacated in CPU | Time Quantum | Arrival Time (Assume) | Burst Time  (Assume) |
| P1 | 50ms | 0ms | 50ms |
| P2 | 60ms | 10ms | 60ms |
| P3 | 70ms | 40ms | 80ms |
| P4 | 80ms | 60ms | 70ms |
| P5 | 85ms (reduced 5) | 80ms | 75ms |
| P6 | 100ms | 100ms | 100ms |

CPU Bound implies that the rate at which a process runs is limited by the CPU's speed. A CPU-bound job is one that performs calculations on a small collection of numbers, such as multiplying small matrices. I/O Bound implies that the rate at which a process moves is constrained by the I/O subsystem's speed.

I/O-bound processes are not punished with lower priorities, but they should be given a smaller slice if they spend the entire time waiting for an I/O unit. The idea is that such a process is dependent on the speed of an I/O system that is unknown to the processor cores. Moreover, since CPU-bound forms of processes consume CPU time, it favors them.

**Ques no 5.**

A program is decomposed into 4 sub-processes. All the sub-processes can execute in parallel. However, some processes can execute only after some other processes can finish execution. This is described by the following precedence constraints:

* Process 3 can start only after processes 1 and 2 finish execution
* Process 4 can start only after 2 finish execution.

Provide a pseudo code with semaphores to accomplish this.

Ans to the ques no-5

Semaphore is an integer variable which is used in mutual exclusion manner by various concurrent cooperative process to achieve synchronization. In binary semaphore the values of semaphore only can be 1 or 2. Binary semaphore is also known as mutex lock. It is used to implement the solution of critical section problems with multiple processes. Here, a program has 4 sub processes. And there are 2 conditions.

Process 3 can start only after processes 1 and 2 finishes. And process 4 can start only after process 2 finish execution. So, it Can be

1. P1>P2>P4>P3

It can be process P1 & P2 comes first. By chance P2 finish its execution & P1 still in critical section while executing. As P2 finishes execution P4 comes in ready queue & P3 become blocked. As there is priority which is P4 start just after P2 finish. Then after that P4 & P1 both finishes. So, the 1st condition become true that process 1 & 2 both finish its execution. And So, Process 3 start then.

1. P2>P4>P1>P3

It also can be occurred if we take, P2 process first. For condition 2, P4 start executing just after P2 finish executing & P3 will become block. Then after a wait operation we can unblock P3 & then execute P1. After P1 & P2 both finishes, P3 can start executing for condition 1.

Now we can implement these by using counting or binary semaphore. Here, I am using binary semaphore for simplification. The pseudo code & algorithm are in follows:

1. For 1. P1>P2>P4>P3

I will use 3V(Signal/Up), 1P(Wait/Down), 2V(Up)

Assume, S1 & S2 are 2 binary semaphores.

Int S1=0, S2=0;

**For P1**: Up (Semaphore S1) {

If (suspended list==0) //if suspended or block list empty

{

S1.value=1; //S1= from 0 to 1 means successful operation so P1 enter the critical section

}

else {

select a process from suspend list and wakeup (); //not required for process 1 as block list is empty at first

}

} now, p1 in CS

**For P2**: Up (Semaphore S2) {

If (suspended list==0) //if suspended or block list empty

{

S2.value=1; //S2= from 0 to 1 means successful operation so P1 enter the critical section

}

else {

select a process from suspend list and wakeup (); //not required for process 2 as block list is empty

}

} now, p2 in CS

**For P3**: Up (Semaphore S2) {

If (suspended list==0) //if suspended or block list empty

{

S2.value=1; //S2= from 1 to 1 means unsuccessful operation so P3 will enter the suspended list

}

else {

select a process from suspend list and wakeup (); //not come in this

}

} now, p3 in block list

**For P4**:

Down (Semaphore S2) {

If (S2.value==1)

S2.value=0; //successful operation

}

else {

Block this process and place in suspended list.

Sleep ();}

} So, now P4 is in CS

**For P3**: Up (Semaphore S2) {

If (suspended list==0) //if suspended or block list empty

{

S2.value=1; //S2= from 0 to 1 means successful operation so P3 will enter the CS

}

else {

select a process from suspend list and wakeup (); //not come in this

}

} now, p3 in CS

So, all the processes execution successfully like that.

Moreover, if we want the 2nd option P2>P4>P1>P3 which is also correct. We must apply 1V,1P, 1V,1P,1V which can be complex.

For P2: up (semaphore S2);

For P3 block: down (semaphore S1)

For P4: up (semaphore S1)

For P1: down (semaphore S2)

For P3: up (semaphore S1)

The codes are same. So, this is how we can implement the priority-based process scheduling by using semaphore.