

Simulation Based Assignment

Of

Operating System

Name : Hriday Sharma

Reg No. : 11603080

Roll No. : B47

Section : K1640

Email : hridaysharma0344@gmail.com

Github Link:<https://github.com/hridaysharma0344/OS_project>

**Code:** Mention solution code assigned to you

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

struct process{

int name; //stores process as integer value

float burst\_time; //time taken by process in CPU

float arrival\_time;

float status;

float waiting\_time

float turnAroundTime

};

void display(struct process proc[],int num){

int i;

printf(“ Details\n");

printf("Process\tBurst-time\tArrival-time\n");

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

printf("p%d\t%f\t%f\n",proc[i].name,proc[i].burst\_time,proc[i].arrival\_time);

}

printf("\n");

}

void sort(struct process proc[],int num){

int i,j;

struct process temp;

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

for(j=i+1;j<num;j++{

if(proc[i].arrival\_time>proc[j].arrival\_time){

temp=proc[i];

proc[i]=proc[j];

proc[j]=temp;

}

}

}

void schedule(struct process proc[],int num,int sum){

int i,j;

float f,avgWaitingTime=0,avgTurnaroundTime=0;

sort(proc,num);

printf("\nProcess\tBurst Time\tArrival Time\tWaiting Time\tTurn-Around Time\n");

for(f=proc[0].arrival\_time;f<(float)sum;){

float pr=-9999;//priority

int nxt;//hold the the next process

float temp;

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

if(proc[i].arrival\_time<=f && proc[i].status!=1){

temp=(proc[i].burst\_time + (f - proc[i].arrival\_time)) / proc[i].burst\_time;

if(pr<temp){ pr=temp;

nxt=i;

}

}

}

f=proc[nxt].burst\_time+f;

proc[nxt].waiting\_time=f-(proc[nxt].arrival\_time)-(proc[nxt].burst\_time);

proc[nxt].turnAroundTime=f-proc[nxt].arrival\_time;

avgWaitingTime+=proc[nxt].waiting\_time; avgTurnaroundTime+=proc[nxt].turnAroundTime;

proc[nxt].status=1;

printf("p%d\t%f\t%f",proc[nxt].name,proc[nxt].burst\_time,proc[nxt].arrival\_time);

printf("\t%f\t%f\n",proc[nxt].waiting\_time,proc[nxt].turnAroundTime);

}

printf("Average waiting time=%f\n",avgWaitingTime/num);

printf("Average turn-around time=%f\n",avgTurnaroundTime/num);

}

int main(){

int i;//loop variable

float arrival,burst,sum=0;

static int num;//holds number of process

struct process proc[10];//array of struct to hold multiple propeties of a single process

printf("number of process:");

scanf("%d",&num);

printf("\n data for the processes%d processes:\n",num);

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

proc[i].name=i+1;

printf("Process p%d\n",proc[i].name);

printf("Burst time:");

scanf("%f",&burst);

proc[i].burst\_time=burst;

printf("Arrival time:");

scanf("%f",&arrival);

proc[i].arrival\_time=arrival;

proc[i].status=0;

sum+=proc[i].burst\_time;

printf("\n");

}

display(proc,num);

sort(proc,num);

schedule(proc,num,sum);

}

1. Explain the problem in terms of operating system concept?

Ans. Shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJN is a non-premeptive algorithm.

* Shortest Job first has the advantage of having minimum average waiting time among all scheduling algorithms.
* It is a Greedy Algorithm.
* It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of aging.
* It is practically infeasible as Operating System may not know burst time and therefore may not sort them. While it is not possible to predict execution time, several methods can be used to estimate the execution time for a job, such as a weighted average of previous execution times. SJF can be used in specialized environments where accurate estimates of running time are available.

2. Write the algorithm for proposed solution of the assigned problem.

1. SJF is a scheduling algorithm that assigns to each process the length of its next CPU burst/execution time. CPU is then given to the process with the minimal CPU burst from the waiting queue. SJF is provably optimal, in that for a given set of processes and their CPU bursts/execution times it gives the least average waiting time for each process. The average waiting time for a process is defined by:
2. WS=(W1+W2+...+Wn)/n [1], where Wk=Wk-1+tk-1 [2] is the waiting time for a kth process and ti is the execution time/length of next CPU burst of the ith process; 1<= k, i <=n (actually, the execution time of the last process in the queue, tn, does not affect any waiting times), and W0=0.
3. If we replace [2] into [1], we get: WS=((n-1)t1+(n-2)t2+...+(n-k)tk+... +tn-1)/n [3]
4. Now let us suppose that we have an arbitrary set of n CPU bursts, { t1, t2, ... , tn }.
5. The average process waiting time in such a set is given by [3]. If we take from that set two processes, k-j and k, such that tk-j>tk , k>j and switch them, the new average waiting time is:
6. WS1=((n-1)t1+...+(n-k+j)tk+... +(n-k)tk-j+...+tn-1)/n [4]
7. If we subtract [4] from [3] we get:
8. WS-WS1= ((n-k+j)tk-j+(n-k)tk-(n-k+j)tk-(n-k)tk-j)/n = j(tk-j-tk)/n > 0, therefore WS1 < WS.
9. By repeating the process over and over and putting shorter jobs in front of longer ones, we will eventually completely order the starting set and achieve the minimal average waiting time:
10. WSof ordered set<...<WS1< WS. Since we started with an arbitrary set, this completes this short and simple optimality proof of the SJF.
11. The main drawback of the SJF alogrithm is that, of course, in modern operating system enviroments we cannot precisely determine the length of the process' next CPU burst. We can at best approximate it, and that only by a certain degree and in certain conditions.

3. Calculate complexity of implemented algorithm. (Student must specify complexity of each line of code along with overall complexity)

Solution: Both the [Insert](https://en.wikipedia.org/wiki/Binary_heap#Insert) and the [Extract-Min](https://en.wikipedia.org/wiki/Binary_heap#Extract) operations have a running time of O(log n). There are n tasks and each task has to be inserted to the heap and later extracted from the heap, resulting in a running time of O(n log n).

The reason Insert has a running time of O(log n) is that when inserting we first add the new task to the heap final position. This does not necessarily maintain the [heap property](https://xlinux.nist.gov/dads/HTML/minheapprop.html), as the priority of the new task might be better (having a smaller key) than the priority of its parent. This is why the Insert operation involves the [Heapify-Up procedure](https://www.cs.duke.edu/courses/spring05/cps130/lectures/littman.lectures/lect08/node17.html) whose purpose is to restore the heap order. The Heapify-Up procedure has a running time of O(log n).

The reason Extract-Min has a running time of O(log n) is that in the Extract-Min operation we first delete the root of the heap (the first task) and then move the last task to the first position (that is, replacing the root with the task residing in the last position). As this possibly violates the heap property, the Extract-Min involves the execution of the [Heapify-Down procedure](https://www.cs.duke.edu/courses/spring05/cps130/lectures/littman.lectures/lect08/node16.html). The Heapify-Down procedure also has a running time of O(log n).

5. If you have implemented any additional algorithm to support the solution, explain the need and usage of the same.

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

struct process{

int name; //stores process as integer value

float burst\_time; //time taken by process in CPU

float arrival\_time;

float status;

float waiting\_time

float turnAroundTime

};

void display(struct process proc[],int num){

int i;

printf(“ Details\n");

printf("Process\tBurst-time\tArrival-time\n");

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

printf("p%d\t%f\t%f\n",proc[i].name,proc[i].burst\_time,proc[i].arrival\_time);

}

printf("\n");

}

void sort(struct process proc[],int num){

int i,j;

struct process temp;

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

for(j=i+1;j<num;j++{

if(proc[i].arrival\_time>proc[j].arrival\_time){

temp=proc[i];

proc[i]=proc[j];

proc[j]=temp;

}

}

void schedule(struct process proc[],int num,int sum){

int i,j;

float f,avgWaitingTime=0,avgTurnaroundTime=0;

sort(proc,num);

printf("\nProcess\tBurst Time\tArrival Time\tWaiting Time\tTurn-Around Time\n");

for(f=proc[0].arrival\_time;f<(float)sum;){

float pr=-9999;//priority

int nxt;//hold the the next process

float temp;

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

if(proc[i].arrival\_time<=f && proc[i].status!=1){

temp=(proc[i].burst\_time + (f - proc[i].arrival\_time)) / proc[i].burst\_time;

if(pr<temp){ pr=temp;

nxt=i;

}

}

}

7.) Explain all the test cases applied on the solution of assigned problem.

P1 is arrived at time = 0, so it will be executed first. Service Time of this process is 3. So this process is completed at time=3.

At time=3, there is only one process that is arrived which is P2. All other processes arrive later. So this process is now executed. Service time of this process is 6, so this process is completed at time=3+6=9.

Now at time=9, there are three processes which are P3, P4 and P5 (which arrived at time= 4, 6 and 8 respectively). Since the service time of P5 is 2 which is minimum as compared to that of P3 and P4, so P5 is now executed and it gets completed at time=9+2=11.

At time=11, we have two processes which are P3 and P4 (which are arrived at time= 4 and 6respectively). Since the service time of P3 is 4 which is less as compared to that of P4, so P4 is executed now and it gets completed at time= 11+4=15

At time=15, we have only one process which is P4. So it is executed now. Since service time of this process is 5, so it gets completed at time = 15+5 = 20.

8. Have you made minimum 5 revisions of solution on GitHub?

GitHub Link: <https://github.com/hridaysharma0344/OS_project>