**Program Code**

* **First Come First Serve Scheduling**

#include <stdio.h>

#include <stdlib.h>

# define max 25

int queue[max],front=-1,rear=-1;

static int total=0;

void enqueue(int key)

{

if(rear==max-1)

{

printf("Overflow");

}

else

{

rear++;

queue[rear]=key;

}

}

void selectionSort(int arr[], int n)

{

int i, j, min,temp;

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

{

min = i;

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

if (arr[j] < arr[min]) {

min = j;

}

}

if(min!=i){

temp = arr[i];

arr[i] = arr[min];

arr[min] = temp;

}

}

}

void process\_queue(int queue[],int ct[],int bt[],int at[],int size){

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

int j = queue[i];

if((i==0 && at[j-1]!=0)||(i!=0 && at[j-1]>total)){

printf("\ninvalid input given!!\n");

exit(0);

}

ct[j-1]=total+bt[j-1];

total=ct[j-1];

}

}

int search\_queue(int value,int size){

if(size==0)

return size;

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

if(value==queue[i] && queue[i]!=0)

return 1;

return 0;

}

void main(){

int k,count=0;

int n,prev;

int \*process; //process id

int \*wait; //waiting time

int \*bt; //burst time

int \*arrival; //arrival time

int \*arrival\_dummy;

int \*completion; //completion time

int \*tat; //turn around time

int totaltat=0,totalwt=0;

printf("Enter the number of processes: ");

scanf("%d",&n);

process = (int\*)malloc(n \* sizeof(int));

wait = (int\*)malloc(n \* sizeof(int));

bt = (int\*)malloc(n \* sizeof(int));

arrival = (int\*)malloc(n \* sizeof(int));

arrival\_dummy = (int\*)malloc(n \* sizeof(int));

completion = (int\*)malloc(n \* sizeof(int));

tat = (int\*)malloc(n \* sizeof(int));

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

process[i]=i+1;

printf("Enter the arrival time for the process %d:",i+1);

scanf("%d",&arrival[i]);

arrival\_dummy[i]=arrival[i];

printf("Enter the burst time for the process %d:",i+1);

scanf("%d",&bt[i]);

}

selectionSort(arrival\_dummy,n);

k=0;

//gangtt chart

while(k<n){

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

if(arrival[i]==arrival\_dummy[k]){

if(!search\_queue(process[i],n)){ //to avoid same process to be enqueued again

enqueue(process[i]);

k++;

}

else{

printf("Value is already there:%d\n",process[i]);

if(++count>max){

k=n-1; //getting out of loop

break;

}

}

}

}

}

//displaying gangtt chart

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

printf("%d\t",queue[i]);

}

process\_queue(queue,completion,bt,arrival,n);

printf("\n");

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

tat[i]=completion[i]-arrival[i];

wait[i]=tat[i]-bt[i];

printf("Waiting time : %d and turnaround time :%d of process %d\n",wait[i],tat[i],i+1);

totaltat+=tat[i];

totalwt+=wait[i];

}

//avg waiting and turnaround time

printf("\n");

printf("avg waiting time:%.2f",(float)totalwt/n);

printf("\n");

printf("avg turnaround time:%.2f\n",(float)totaltat/n);

}

* **Shortest Job First Scheduling**

#include <stdio.h>

#include <stdlib.h>

struct sjf{

int at;

int bt;

int pno;

int tat;

int wt;

};

struct sjf \*s1;

void display(int);

void selectionSort(int n)

{

int i, j;

struct sjf temp;

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

{

temp=s1[i];

for (j = i-1; j >= n; j--) {

if (temp.at >s1[j].at) {

s1[j+1]=s1[j];

}

else

break;

}

s1[j+1]=temp;

}

display(n);

}

void selectBt(int num){ //sort on the basis on burst time with same arrival time

int i,j;

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

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

if(s1[i].at!=s1[j].at){

break;

}

else if((s1[i].at==s1[j].at)&&(s1[i].bt>s1[j].bt)){

struct sjf temp = s1[i];

s1[i]=s1[j];

s1[j]=temp;

}

}

}

}

void sortAll(int num){ //sort when the burst time is less but arrival time is more

//sort on the basic of process that have less burst time but more arrival time

int total=0;

for(int i=1;i<num;i++){

if(total<s1[i-1].at)

total=s1[i-1].at;

total+=s1[i-1].bt;

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

if(total>=s1[i].at){

if(s1[j].bt<s1[i].bt){

struct sjf temp= s1[i];

s1[i]=s1[j];

s1[j]=temp;

}

else if((s1[j].bt==s1[i].bt)&&(s1[i].pno>s1[j].pno)){

struct sjf temp= s1[i];

s1[i]=s1[j];

s1[j]=temp;

}

}

else

break;

}

}

}

void display(int num){

printf("Process | AT \t | BT");

printf("\n");

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

printf("%d \t | %d \t | %d",s1[i].pno,s1[i].at,s1[i].bt);

printf("\n");

}

printf("Gantt chart is \n");

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

printf("%d\t",s1[i].pno);

}

float total\_time=0,avg\_wt=0,avg\_tat=0;

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

if(total\_time<s1[i].at){

total\_time=s1[i].at;

}

total\_time+=s1[i].bt;

s1[i].tat=total\_time-s1[i].at;

s1[i].wt=s1[i].tat-s1[i].bt;

}

//for calculating the average waiting time and turn arounf time

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

avg\_wt+=s1[i].wt;

avg\_tat+=s1[i].tat;

}

printf("Average waiting time: %.2f ms\n",avg\_wt/num);

printf("Average turn around time: %.2f ms \n",avg\_tat/num);

}

void main(){

static int total = 0;

int num;

printf("Enter the total number of process: ");

scanf("%d",&num);

s1 = (struct sjf\*)malloc(num \* sizeof(struct sjf));

//asking for arrival time and burst time

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

{

s1[i].pno = i+1;

printf("What is the arrival time of process %d: ",i+1);

scanf("%d",&s1[i].at);

printf("What is the burst time of process %d: ",i+1);

scanf("%d",&s1[i].bt);

}

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

printf("%d \t %d \t %d",s1[i].pno,s1[i].at,s1[i].bt);

printf("\n");

}

selectionSort(num); //sort on the basis of arrival time

selectBt(num); //sort on the basis on burst time with same arrival time

sortAll(num); //sort when the burst time is less but arrival time is more

display(num);

}

* **Priority (Non-preemptive) Scheduling**

#include <stdio.h>

#include <stdlib.h>

struct priority{

int at;

int bt;

int pno;

int tat;

int wt;

int priority;

};

struct priority \*s1;

void selectionSort(int n)

{

int i, j, min;

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

{

min = i;

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

if (s1[min].at >s1[j].at) {

min = j;

}

}

if(min!=i){

struct priority temp= s1[min];

s1[min]=s1[i];

s1[i]=temp;

}

}

}

void sortByPriority(int n){

int i,j;

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

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

if(s1[i].at!=s1[j].at){

break;

}

else if(s1[i].at==s1[i].at && s1[i].priority>s1[j].priority){

struct priority temp=s1[i];

s1[i]=s1[j];

s1[j]=temp;

}

}

}

}

void display(int num){

printf("Process | AT \t | BT \t | Priority");

printf("\n");

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

printf("%d \t | %d \t | %d \t | %d",s1[i].pno,s1[i].at,s1[i].bt,s1[i].priority);

printf("\n");

}

printf("Gantt chart is \n");

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

printf("%d\t",s1[i].pno);

}

float total\_time=0,avg\_wt=0,avg\_tat=0;

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

if(total\_time<s1[i].at){

total\_time=s1[i].at;

}

total\_time+=s1[i].bt;

s1[i].tat=total\_time-s1[i].at;

s1[i].wt=s1[i].tat-s1[i].bt;

}

//for calculating the average waiting time and turn arounf time

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

avg\_wt+=s1[i].wt;

avg\_tat+=s1[i].tat;

}

printf("Average waiting time: %.2f ms\n",avg\_wt/num);

printf("Average turn around time: %.2f ms \n",avg\_tat/num);

}

void priority(int n){

int i,j,total=0;

//processes with higher priority coming later

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

if(total<s1[i-1].at)

total=s1[i-1].at;

total+=s1[i-1].bt;

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

if(total>=s1[j].at){

if(s1[j].priority < s1[i].priority){

struct priority temp = s1[i];

s1[i] = s1[j];

s1[j] = temp;

}

else if (s1[j].priority == s1[i].priority && s1[i].pno > s1[j].pno) {

struct priority temp = s1[i];

s1[i] = s1[j];

s1[j] = temp;

}

}

else

break;

}

}

}

void main(){

static int total = 0;

int num;

printf("Enter the total number of process: ");

scanf("%d",&num);

s1 = (struct priority\*)malloc(num \* sizeof(struct priority));

//asking for arrival time and burst time

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

{

s1[i].pno = i+1;

printf("What is the arrival time of process %d: ",i+1);

scanf("%d",&s1[i].at);

printf("What is the burst time of process %d: ",i+1);

scanf("%d",&s1[i].bt);

printf("What is the priority of process %d: ",i+1);

scanf("%d",&s1[i].priority);

}

selectionSort(num);

sortByPriority(num);

priority(num);

display(num);

}

* **Round Robin Scheduling**

#include <stdio.h>

#include <stdlib.h>

struct rr{

int at;

int bt;

int pno;

int tat;

int wt;

int remain;

};

struct rr \*s1;

void selectionSort(int n)

{

int i, j, min;

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

{

min = i;

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

if (s1[min].at >s1[j].at) {

min = j;

}

}

if(min!=i){

struct rr temp= s1[min];

s1[min]=s1[i];

s1[i]=temp;

}

}

}

void roundrobin(int num,float q){

int i,j,count=0;

printf("Round Robin sorted chart on the basis of AT is given. \n");

printf("Process | AT \t | BT \t");

printf("\n");

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

printf("%d \t | %d \t | %d",s1[i].pno,s1[i].at,s1[i].bt);

printf("\n");

}

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

s1[i].remain=s1[i].bt;

}

printf("\n");

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

printf("%d",s1[i].remain);

printf("\n");

}

count=0;

printf("%d",count);

printf("\n");

printf("Gantt chart is \n");

for(i=0;count!=num;i=(i+1)%num){

if(s1[i].remain!=0){

printf("%d \t",s1[i].pno);

if(q<s1[i].remain){

s1[i].remain-=q;

}

else{

s1[i].remain=0;

count++;

if(count==num)

break;

}

}

}

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

s1[i].remain=s1[i].bt;

float total\_time=0,avg\_wt=0,avg\_tat=0;

//calculations

for(i=0,count=0;count!=num;i=(i+1)%num){

if(s1[i].remain<=q){

if(total\_time<s1[i].at){

total\_time=s1[i].at;

}

total\_time+=s1[i].remain;

s1[i].tat=total\_time-s1[i].at;

s1[i].wt=s1[i].tat-s1[i].bt;

s1[i].remain=0;

count++;

}

else{

if(total\_time<s1[i].at){

total\_time=s1[i].at;

}

total\_time+=q;

s1[i].remain-=q;

}

}

//for calculating the average waiting time and turn around time

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

avg\_wt+=s1[i].wt;

avg\_tat+=s1[i].tat;

}

printf("\n");

printf("Average waiting time: %.2f ms\n",avg\_wt/num);

printf("Average turn around time: %.2f ms \n",avg\_tat/num);

}

void main(){

static int total = 0;

int num;

float slice;

printf("Enter the total number of process: ");

scanf("%d",&num);

s1 = (struct rr\*)malloc(num \* sizeof(struct rr));

printf("What is the timeslice of processes");

scanf("%f",&slice);

//asking for arrival time and burst time

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

{

s1[i].pno = i+1;

printf("What is the arrival time of process %d: ",i+1);

scanf("%d",&s1[i].at);

printf("What is the burst time of process %d: ",i+1);

scanf("%d",&s1[i].bt);

}

selectionSort(num); //sort by arrival time

roundrobin(num,slice);

}

**Output**

* **First Come First Serve Scheduling**

Enter the number of processes: 3

Enter the arrival time for the process 1:4

Enter the burst time for the process 1:3

Enter the arrival time for the process 2:0

Enter the burst time for the process 2:3

Enter the arrival time for the process 3:2

Enter the burst time for the process 3:3

Value is already there:2

2 3 1

Waiting time : 2 and turnaround time :5 of process 1

Waiting time : 0 and turnaround time :3 of process 2

Waiting time : 1 and turnaround time :4 of process 3

avg waiting time:1.00

avg turnaround time:4.00

* **Shortest Job First Scheduling**

Enter the total number of process: 4

What is the arrival time of process 1: 3

What is the burst time of process 1: 2

What is the arrival time of process 2: 0

What is the burst time of process 2: 3

What is the arrival time of process 3: 1

What is the burst time of process 3: 5

What is the arrival time of process 4: 2

What is the burst time of process 4: 2

1 3 2

2 0 3

3 1 5

4 2 2

Process | AT | BT

1 | 3 | 2

2 | 0 | 3

3 | 1 | 5

4 | 2 | 2

Gantt chart is

1 2 3 4 Average waiting time: 5.75 ms

Average turn around time: 8.75 ms

Process | AT | BT

1 | 3 | 2

4 | 2 | 2

2 | 0 | 3

3 | 1 | 5

Gantt chart is

1 4 2 3 Average waiting time: 4.75 ms

Average turn around time: 7.75 ms

* **Priority (Non-preemptive) Scheduling**

Enter the total number of process: 5

What is the arrival time of process 1: 3

What is the burst time of process 1: 4

What is the priority of process 1: 3

What is the arrival time of process 2: 0

What is the burst time of process 2: 4

What is the priority of process 2: 5

What is the arrival time of process 3: 2

What is the burst time of process 3: 4

What is the priority of process 3: 1

What is the arrival time of process 4: 4

What is the burst time of process 4: 4

What is the priority of process 4: 4

What is the arrival time of process 5: 5

What is the burst time of process 5: 3

What is the priority of process 5: 2

Process | AT | BT | Priority

2 | 0 | 4 | 5

3 | 2 | 4 | 1

5 | 5 | 3 | 2

1 | 3 | 4 | 3

4 | 4 | 4 | 4

Gantt chart is

2 3 5 1 4 Average waiting time: 4.80 ms

Average turn around time: 8.60 ms

* **Round Robin Scheduling**

Enter the total number of process: 4

What is the timeslice of processes2

What is the arrival time of process 1: 5

What is the burst time of process 1: 3

What is the arrival time of process 2: 0

What is the burst time of process 2: 4

What is the arrival time of process 3: 1

What is the burst time of process 3: 6

What is the arrival time of process 4: 4

What is the burst time of process 4: 4

Round Robin sorted chart on the basis of AT is given.

Process | AT | BT

2 | 0 | 4

3 | 1 | 6

4 | 4 | 4

1 | 5 | 3

4

6

4

3

0

Gantt chart is

2 3 4 1 2 3 4 1 3

Average waiting time: 6.00 ms

Average turn around time: 8.75 ms