Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.  
  
  1. SJF (pre-emptive &; Non-pre-emptive)  
2. Priority (pre-emptive &; Non-pre-emptive)  
3. Round Robin (Experiment with different quantum sizes for RR algorithm)

#include <stdio.h>

#include <stdbool.h>

#define MAX\_PROCESSES 10

struct Process {

int pid;

int arrival\_time;

int burst\_time;

int priority;

int remaining\_time;

int turnaround\_time;

int waiting\_time;

};

void sjf\_nonpreemptive(struct Process processes[], int n) {

// Sort the processes based on burst time in ascending order

int i,j,count=0,m;

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

{

if(processes[i].arrival\_time==0)

count++;

}

if(count==n||count==1)

{

if(count==n)

{

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

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

if (processes[j].burst\_time > processes[j + 1].burst\_time) {

struct Process temp = processes[j];

processes[j] = processes[j + 1];

processes[j + 1] = temp;

}

}

}

}

else

{

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

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

if (processes[j].burst\_time > processes[j + 1].burst\_time) {

struct Process temp = processes[j];

processes[j] = processes[j + 1];

processes[j + 1] = temp;

}

}

}

}

}

int total\_time = 0;

double total\_turnaround\_time = 0;

double total\_waiting\_time = 0;

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

printf("Process %d is executing for %d unit of time\n",processes[i].pid,processes[i].burst\_time);

total\_time += processes[i].burst\_time;

processes[i].turnaround\_time = total\_time - processes[i].arrival\_time;

processes[i].waiting\_time = processes[i].turnaround\_time - processes[i].burst\_time;

total\_turnaround\_time += processes[i].turnaround\_time;

total\_waiting\_time += processes[i].waiting\_time;

}

printf("Process\tTurnaround Time\tWaiting Time\n");

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

printf("%d\t%d\t\t%d\n", processes[i].pid, processes[i].turnaround\_time, processes[i].waiting\_time);

}

printf("Average Turnaround Time: %.2f\n", total\_turnaround\_time / n);

printf("Average Waiting Time: %.2f\n", total\_waiting\_time / n);

}

void sjf\_preemptive(struct Process processes[], int n) {

int total\_time = 0,i,p\_total\_time=0;

int completed = 0;

while (completed < n) {

int shortest\_burst = -1;

int next\_process = -1;

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

if (processes[i].arrival\_time <= total\_time && processes[i].remaining\_time > 0) {

if (shortest\_burst == -1 || processes[i].remaining\_time < shortest\_burst) {

shortest\_burst = processes[i].remaining\_time;

next\_process = i;

}

}

}

if (next\_process == -1) {

total\_time++;

continue;

}

processes[next\_process].remaining\_time--;

total\_time++;

if (processes[next\_process].remaining\_time == 0) {

completed++;

printf("Process %d is executed for %d time units\n",processes[next\_process].pid,total\_time-p\_total\_time);

p\_total\_time=total\_time;

processes[next\_process].turnaround\_time = total\_time - processes[next\_process].arrival\_time;

processes[next\_process].waiting\_time = processes[next\_process].turnaround\_time - processes[next\_process].burst\_time;

}

else

{

printf("Process %d is executed for %d time units\n",processes[next\_process].pid,total\_time-p\_total\_time);

p\_total\_time=total\_time;

}

}

double total\_turnaround\_time = 0;

double total\_waiting\_time = 0;

printf("Process\tTurnaround Time\tWaiting Time\n");

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

printf("%d\t%d\t\t%d\n", processes[i].pid, processes[i].turnaround\_time, processes[i].waiting\_time);

total\_turnaround\_time += processes[i].turnaround\_time;

total\_waiting\_time += processes[i].waiting\_time;

}

printf("Average Turnaround Time: %.2f\n", total\_turnaround\_time / n);

printf("Average Waiting Time: %.2f\n", total\_waiting\_time / n);

}

void priority\_nonpreemptive(struct Process processes[], int n) {

// Sort the processes based on priority in ascending order

int i,j,count=0,m;

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

{

if(processes[i].arrival\_time==0)

count++;

}

if(count==n||count==1)

{

if(count==n)

{

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

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

if (processes[j].priority > processes[j + 1].priority) {

struct Process temp = processes[j];

processes[j] = processes[j + 1];

processes[j + 1] = temp;

}

}

}

}

else

{

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

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

if (processes[j].priority > processes[j + 1].priority) {

struct Process temp = processes[j];

processes[j] = processes[j + 1];

processes[j + 1] = temp;

}

}

}

}

}

int total\_time = 0;

double total\_turnaround\_time = 0;

double total\_waiting\_time = 0;

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

printf("Process %d is executing for %d unit of time\n",processes[i].pid,processes[i].burst\_time);

total\_time += processes[i].burst\_time;

processes[i].turnaround\_time = total\_time - processes[i].arrival\_time;

processes[i].waiting\_time = processes[i].turnaround\_time - processes[i].burst\_time;

total\_turnaround\_time += processes[i].turnaround\_time;

total\_waiting\_time += processes[i].waiting\_time;

}

printf("Process\tTurnaround Time\tWaiting Time\n");

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

printf("%d\t%d\t\t%d\n", processes[i].pid, processes[i].turnaround\_time, processes[i].waiting\_time);

}

printf("Average Turnaround Time: %.2f\n", total\_turnaround\_time / n);

printf("Average Waiting Time: %.2f\n", total\_waiting\_time / n);

}

void priority\_preemptive(struct Process processes[], int n) {

int total\_time = 0,i,p\_total\_time=0;

int completed = 0;

while (completed < n) {

int highest\_priority = -1;

int next\_process = -1;

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

if (processes[i].arrival\_time <= total\_time && processes[i].remaining\_time > 0) {

if (highest\_priority == -1 || processes[i].priority < highest\_priority) {

highest\_priority = processes[i].priority;

next\_process = i;

}

}

}

if (next\_process == -1) {

total\_time++;

continue;

}

processes[next\_process].remaining\_time--;

total\_time++;

if (processes[next\_process].remaining\_time == 0) {

completed++;

printf("Process %d is executed for %d time units\n",processes[next\_process].pid,total\_time-p\_total\_time);

p\_total\_time=total\_time;

processes[next\_process].turnaround\_time = total\_time - processes[next\_process].arrival\_time;

processes[next\_process].waiting\_time = processes[next\_process].turnaround\_time - processes[next\_process].burst\_time;

}

else

{

printf("Process %d is executed for %d time units\n",processes[next\_process].pid,total\_time-p\_total\_time);

p\_total\_time=total\_time;

}

}

double total\_turnaround\_time = 0;

double total\_waiting\_time = 0;

printf("Process\tTurnaround Time\tWaiting Time\n");

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

printf("%d\t%d\t\t%d\n", processes[i].pid, processes[i].turnaround\_time, processes[i].waiting\_time);

total\_turnaround\_time += processes[i].turnaround\_time;

total\_waiting\_time += processes[i].waiting\_time;

}

printf("Average Turnaround Time: %.2f\n", total\_turnaround\_time / n);

printf("Average Waiting Time: %.2f\n", total\_waiting\_time / n);

}

int main() {

int n, quantum,i,choice;

struct Process processes[MAX\_PROCESSES];

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

scanf("%d", &n);

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

printf("Process %d\n", i + 1);

printf("Enter arrival time: ");

scanf("%d", &processes[i].arrival\_time);

printf("Enter burst time: ");

scanf("%d", &processes[i].burst\_time);

printf("Enter priority: ");

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

processes[i].pid = i + 1;

processes[i].remaining\_time = processes[i].burst\_time;

processes[i].turnaround\_time = 0;

processes[i].waiting\_time = 0;

}

printf("\nSelect a scheduling algorithm:\n");

printf("1. SJF\n");

printf("2. SRTF\n");

printf("3. Priority Non-preemptive\n");

printf("4. Priority Preemptive\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("\nSJF Scheduling:\n");

sjf\_nonpreemptive(processes, n);

break;

case 2:

printf("\nSRTF Scheduling:\n");

sjf\_preemptive(processes, n);

break;

case 3:

printf("\nPriority Non-preemptive Scheduling:\n");

priority\_nonpreemptive(processes, n);

break;

case 4:

printf("\nPriority Preemptive Scheduling:\n");

priority\_preemptive(processes, n);

break;

default:

printf("Invalid choice!\n");

return 1;

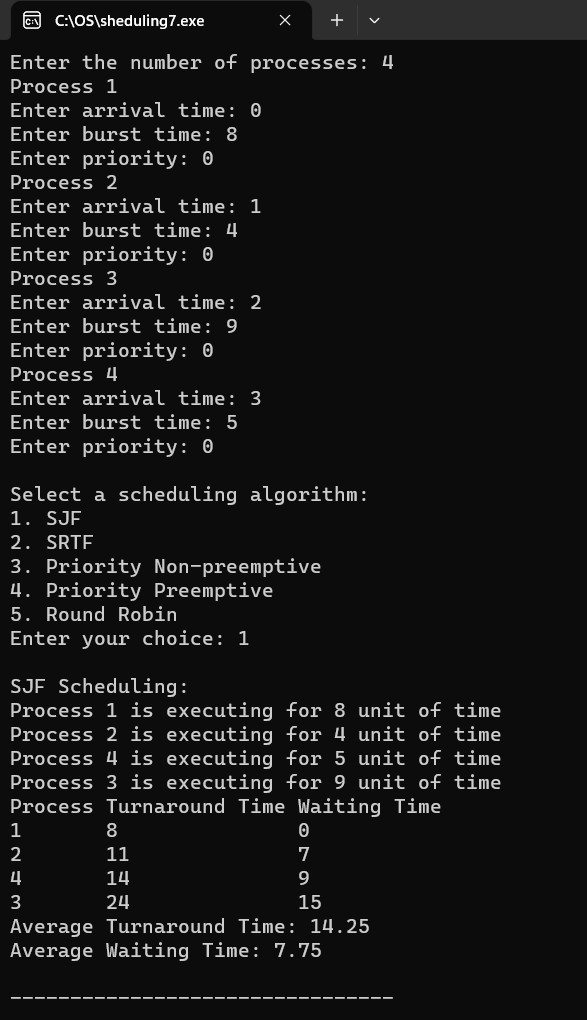
}

return 0;

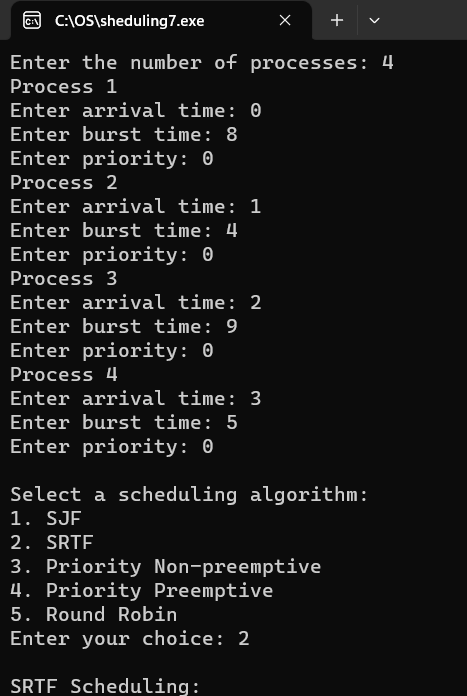
}

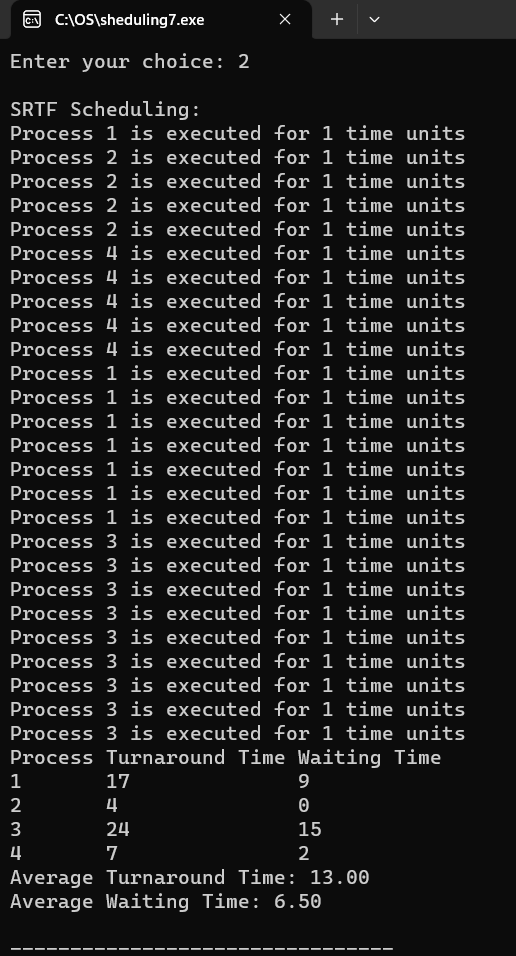
OUTPUT

SJF Scheduling

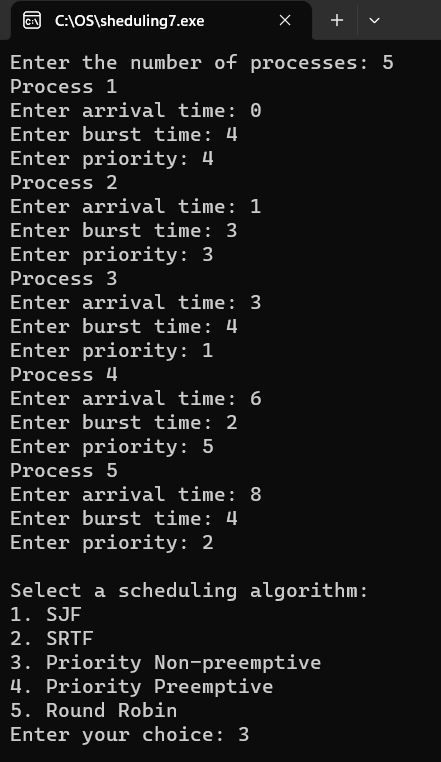


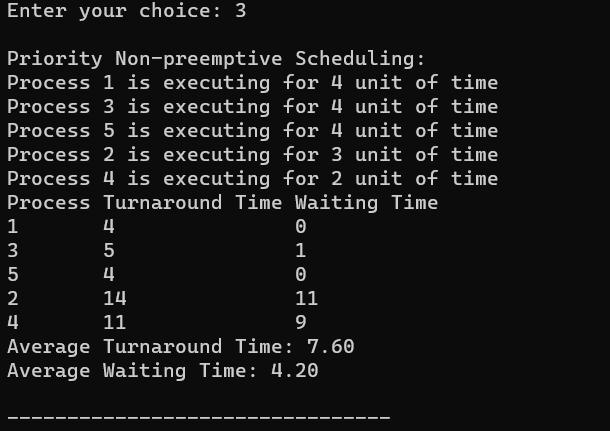
SRTF



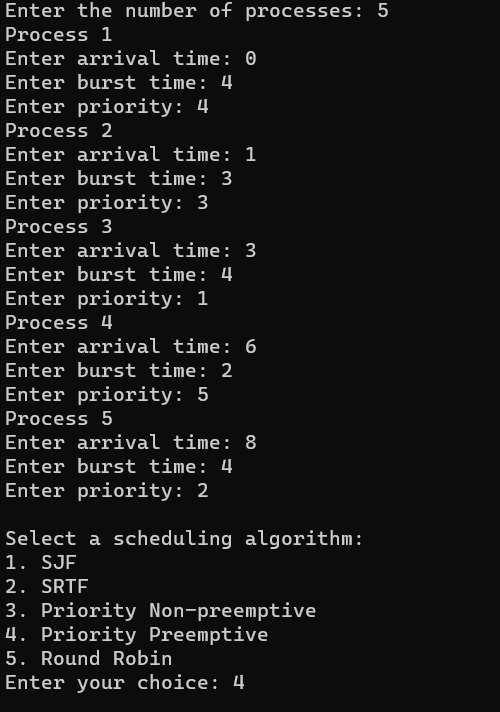


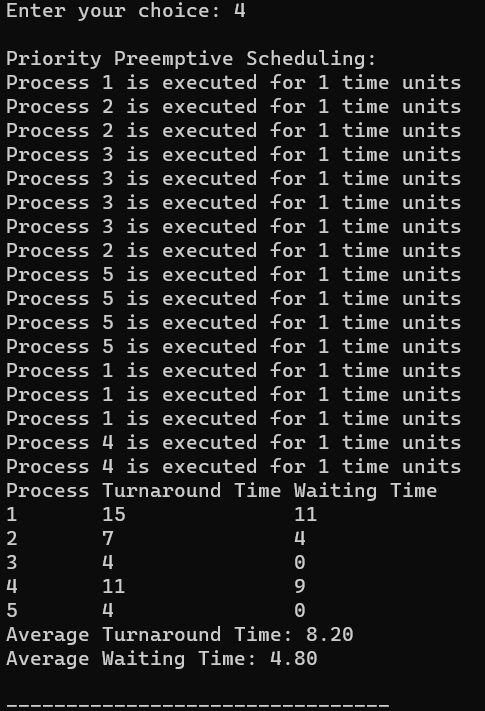
Priority non-preemptive





Priority-preemptive





ROUND ROBIN

#include<stdio.h>

#include<limits.h>

#include<stdbool.h>

struct P{

int AT,BT,ST[20],WT,FT,TAT,pos;

};

int quant;

int main(){

int n,i,j;

// Taking Input

printf("Enter the no. of processes :");

scanf("%d",&n);

struct P p[n];

printf("Enter the quantum \n");

scanf("%d",&quant);

printf("Enter the process numbers \n");

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

scanf("%d",&(p[i].pos));

printf("Enter the Arrival time of processes \n");

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

scanf("%d",&(p[i].AT));

printf("Enter the Burst time of processes \n");

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

scanf("%d",&(p[i].BT));

// Declaring variables

int c=n,s[n][20];

float time=0,mini=INT\_MAX,b[n],a[n];

// Initializing burst and arrival time arrays

int index=-1;

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

b[i]=p[i].BT;

a[i]=p[i].AT;

for(j=0;j<20;j++){

s[i][j]=-1;

}

}

int tot\_wt,tot\_tat;

tot\_wt=0;

tot\_tat=0;

bool flag=false;

while(c!=0){

mini=INT\_MAX;

flag=false;

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

float p=time+0.1;

if(a[i]<=p && mini>a[i] && b[i]>0){

index=i;

mini=a[i];

flag=true;

}

}

// if at =1 then loop gets out hence set flag to false

if(!flag){

time++;

continue;

}

//calculating start time

j=0;

while(s[index][j]!=-1){

j++;

}

if(s[index][j]==-1){

s[index][j]=time;

p[index].ST[j]=time;

}

if(b[index]<=quant){

time+=b[index];

b[index]=0;

}

else{

time+=quant;

b[index]-=quant;

}

if(b[index]>0){

a[index]=time+0.1;

}

// calculating arrival,burst,final times

if(b[index]==0){

c--;

p[index].FT=time;

p[index].WT=p[index].FT-p[index].AT-p[index].BT;

tot\_wt+=p[index].WT;

p[index].TAT=p[index].BT+p[index].WT;

tot\_tat+=p[index].TAT;

}

} // end of while loop

// Printing output

printf("Process number ");

printf("Arrival time ");

printf("Burst time ");

printf("\tStart time");

j=0;

while(j!=10){

j+=1;

printf(" ");

}

printf("\t\tFinal time");

printf("\tWait Time ");

printf("\tTurnAround Time \n");

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

printf("%d \t\t",p[i].pos);

printf("%d \t\t",p[i].AT);

printf("%d \t",p[i].BT);

j=0;

int v=0;

while(s[i][j]!=-1){

printf("%d ",p[i].ST[j]);

j++;

v+=3;

}

while(v!=40){

printf(" ");

v+=1;

}

printf("%d \t\t",p[i].FT);

printf("%d \t\t",p[i].WT);

printf("%d \n",p[i].TAT);

}

//Calculating average wait time and turnaround time

double avg\_wt,avg\_tat;

avg\_wt=tot\_wt/(float)n;

avg\_tat=tot\_tat/(float)n;

//Printing average wait time and turnaround time

printf("The average wait time is : %lf\n",avg\_wt);

printf("The average TurnAround time is : %lf\n",avg\_tat);

return 0;

}

OUTPUT

