|  |  |
| --- | --- |
| **Student Name:** | **Mukul Kathayat** |
| **Student ID:** | 11612208 |
| **Roll :** | A 05 |
| **Email Address:** | Mukulkathayat97@gmail.com |
| **Git-Hub Link:** | <https://github.com/mukulkathayat/Priority-Scheduling> |

**Code**:

#include<stdio.h>

#include<string.h> #include<stdlib.h> int n; struct Process

{

char process[5]; int arr\_t; double estRun\_t; double priority; int wait\_t;

double turnAround\_t;

};

struct Process \*obj; int i,min\_arrtime=65537; int cur\_time=0,i=0,j=0,p=0; char cur\_process[5];

double avgTurnAround\_t=0,avgWait\_t=0; double runTime=100000000.0; void findMinTime()

{

min\_arrtime=65537;

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

{

if(obj[i].priority==-1) continue;

if(min\_arrtime==obj[i].arr\_t)

{

if(obj[i].estRun\_t<runTime)

{

strcpy(cur\_process,obj[i].process); runTime=obj[i].estRun\_t; min\_arrtime=obj[i].arr\_t;

}

}

else if(min\_arrtime>obj[i].arr\_t)

{

strcpy(cur\_process,obj[i].process); runTime=obj[i].estRun\_t;

min\_arrtime=obj[i].arr\_t;

}

}

cur\_time=min\_arrtime;

return;

}

void findMaxPriority()//FUNCTION TO FIND THE MAXIMUM PRIORITY

{

double max\_priority=-10000000.0; for(i=0;i<n;i++)

{

if(obj[i].priority==-1)

{

continue;

}

else

{

if(max\_priority<obj[i].priority)

{

max\_priority=obj[i].priority;

strcpy(cur\_process,obj[i].process);

}

}

}

return;

} void main()

{

printf("enter the number of processes: "); scanf("%d",&n);

printf("\n\t\tEnter the details of the %d processes one by one\n\n",n); obj=(struct Process\*)malloc(sizeof( struct Process)\*n); for(i=0;i<n;i++)

{ printf("Enter the %d Process Name: ",i+1); scanf("%s",obj[i].process); printf("Enter its Arrival Time: "); scanf("%d",&(obj[i].arr\_t)); printf("Enter its Burst Time : "); scanf("%lf",&(obj[i].estRun\_t));

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n"); if(min\_arrtime==obj[i].arr\_t)

{

if(obj[i].estRun\_t<runTime)

{

strcpy(cur\_process,obj[i].process); runTime=obj[i].estRun\_t; min\_arrtime=obj[i].arr\_t;

}

}

else if(min\_arrtime>obj[i].arr\_t)

{

strcpy(cur\_process,obj[i].process); runTime=obj[i].estRun\_t;

min\_arrtime=obj[i].arr\_t;

}

}

cur\_time=min\_arrtime; min\_arrtime=65537;

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

{

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

{

if(obj[i].priority!=-1 && strcmp(obj[i].process,cur\_process)==0)

{

obj[i].priority=-1;

obj[i].wait\_t=cur\_time-obj[i].arr\_t; cur\_time=cur\_time+(int)obj[i].estRun\_t; obj[i].turnAround\_t=cur\_time-obj[i].arr\_t;

}

}

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

{

if(obj[i].priority==-1)

continue;

else if(cur\_time<obj[i].arr\_t)

{

p=1;

continue;

}

else

{

p=-1; break;

}

}

if(p==1)

{

findMinTime();

}

else

{

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

{

if(obj[i].priority!=-1 && cur\_time>obj[i].arr\_t)

{

obj[i].wait\_t=cur\_time-obj[i].arr\_t; obj[i].priority=1+(obj[i].wait\_t/obj[i].estRun\_t);

}

}

findMaxPriority();

}

}

//final printing

printf("\n\t\t\t\tRESULTS AFTER CALCULATIONS\n\n");

printf("\n Process Arrival time Est\_RunTime WaitingTime TurnAroundTime StartTime\n\n");

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

{

printf("\n %s %3d %3.0lf %6d %4.0lf %3d

",obj[i].process,obj[i].arr\_t,obj[i].estRun\_t,obj[i].wait\_t,obj[i].turnAround\_t,obj[i].arr\_t+obj[i].wait\_t);

avgTurnAround\_t=avgTurnAround\_t+obj[i].turnAround\_t; avgWait\_t=avgWait\_t+obj[i].wait\_t;

}

printf("\n\n\n\t\tAverage\_Waiting\_Time of all the processes is %.2lf units",avgWait\_t/n); printf("\n\n\t\tAverage\_TurnAround\_Time of all the processes is %.2lf units\n\n\n\n",avgTurnAround\_t/n);

}

1. **Description**

It’s a process scheduling mechanism done by the scheduler in the operating systems. According to the given scheduling approach it’s a non-preemptive( it is not moved from running queue without the process requesting it), shortest job first in nature approach (in which the shortest job is given the most priority) but here we also have to consider the waiting time of each and every processes, the process which has been waiting for a long time is increased with its priority which prevents the case of starvation on the basis of the formula given in the question i.e:

*Priority*=1+*Waiting time*/ *Estimated runtime*

1. **Algorithms**

 **Algorithm to find the minimum arrival time of the process i.e.**

* + In this function the priority of the process is assumed to be (-1) whichever process has been executed completely.
  + In this function it’s also checked that if the two process arrives at the same time then which process is given more priority.

**# all the required variables are declared globally**

**void findMinTime(struct Process \*obj)**

{

int min\_arrtime=65537;

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

{

if(obj[i].priority== -1)

continue;

if(min\_arrtime==obj[i].arr\_t)

{

if(obj[i].estRun\_t<runTime)

{

strcpy(cur\_process,obj[i].process); runTime=obj[i].estRun\_t;

min\_arrtime=obj[i].arr\_t;

}

}

else if(min\_arrtime>obj[i].arr\_t)

{

strcpy(cur\_process,obj[i].process); runTime=obj[i].estRun\_t;

min\_arrtime=obj[i].arr\_t;

}

}

cur\_time=min\_arrtime;

return;

}

 **Algorithm to find the maximum priority, if the current clock time of the cpu is greater than atleast one of the processes**

void findMaxPriority(struct Process \*obj)

{

double max\_priority=-10000000.0; for(i=0;i<n;i++)

{

if(obj[i].priority== -1)

continue;

else if(max\_priority<obj[i].priority)

{

max\_priority=obj[i].priority;

strcpy(cur\_process,obj[i].process);

}

}

return;

}

* **After every process execution we have to calculate the Turnaround time of that process and have to set the priority of the process to (-1), algorithm for that is:**

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

{

if(obj[i].priority!= -1 && strcmp(obj[i].process, cur\_process)==0)

{

obj[i].priority= -1; //process already done

obj[i].wait\_t=cur\_time-obj[i].arr\_t; //new wait time calculation

cur\_time=cur\_time+(int)obj[i].estRun\_t; //

obj[i].turnAround\_t=cur\_time-obj[i].arr\_t;

}

}

* **Algorithm to calculate the priority after every process execution :** for(i=0;i<n;i++)

{

if(obj[i].priority!=-1 && cur\_time>obj[i].arr\_t)

{

obj[i].wait\_t=cur\_time-obj[i].arr\_t; obj[i].priority=1+(obj[i].wait\_t/obj[i].estRun\_t);

}

}

1. **Complexity** : The complexity of each algorithms given above is of O(n) but since the algorithms are processed after each process execution so it effectively becomes O(n^2).

1. **Boundary conditions /constraints:**

* + Any numerical value should be taken under the range of integer i.e. based on the compiler.
  + Process name should be taken in the string.
  + Time shouldn’t be negative.

**printf("Enter its Arrival Time: ");**

**scanf("%d",&(obj[i].arr\_t)); if(obj[i].arr\_t<0)**

**{**

**printf("\n\t\t\ttime in negative error");**

**exit(0);**

**}**

* + **Burst time shouldn’t be negative. printf("Enter its Burst Time : "); scanf("%lf",&(obj[i].estRun\_t));**

**if(obj[i].estRun\_t<0)**

**{**

**printf("\n\t\t\tEstimated run time in negative error");**

**exit(0);**

**}**

1. **Test cases:**

|  |  |  |
| --- | --- | --- |
| **Process** | **Arrival time** | **Burst time** |
| **P1** | **0** | **20** |
| **P2** | **5** | **36** |
| **P3** | **13** | **19** |
| **P4** | **17** | **42** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Process** | **Arrival time** | **Burst time** | **Start time** | **Waiting time** | **Turn around time** |
| **P1** | **0** | **20** | **0** | **0** | **20** |
| **P2** | **5** | **36** | **20** | **15** | **51** |
| **P3** | **13** | **19** | **56** | **43** | **62** |
| **P4** | **17** | **42** | **75** | **58** | **100** |

**Priority of each process:**

**After p1 executed:**

P1= -1

P2= 3.4

P3=3.714

P4=15 there fore p2 will execute now

**After p2 executed:**

P1= -1

P2=-1

P3= 1.44

P4=2.07 there fore p3 will execute now

**After p3 executed:** p4 will execute there fore p4 will execute now

At first process p1 will start then after waiting for 20 units p2 will start and then after p2 ends p3 starts and then p4

**Avg turn around time= (20+51+62+100)/4=58.25 units**

**Avg waiting time = (0+15+43+58)/4=29.00 units**

**GITHUB COMMIT**