Manishi Parmar

11804462

[Parmarmanishi12345@gmail.com](mailto:Parmarmanishi12345@gmail.com)

Manishi510(GitHub ID)

Question:-

1. Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed. In answering the questions, use non-preemptive scheduling, and base all decisions on the information you have at the time the decision must be made.

Process Arrival Time Burst Time

*P*1 0.0 8

*P*2 0.4 4

*P*3 1.0 1

a. What is the average turnaround time for these processes with the FCFS scheduling algorithm?

b. What is the average turnaround time for these processes with the SJF scheduling algorithm?

c. Compute what average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes *P*1 and *P*2 are waiting during this idle time, so their waiting time may increase.

Answer:-

1. FCFS

P1 P2 P3

|  |  |  |
| --- | --- | --- |
|  |  |  |

0 8 12 13

As we know that ,

Turn around Time = Final time – Arrival Time

|  |  |
| --- | --- |
| process | TAT |
| P1 | 8 - 0 = 8 |
| P2 | 12 - 0.4 =11.6 |
| P3 | 13 - 1 = 12 |
| Average TAT | = 10.53 |

1. SJF

P1 P3 P2

|  |  |  |
| --- | --- | --- |
|  |  |  |

0 8 9 13

As we know that ,

Turn around Time = Final time – Arrival Time

|  |  |
| --- | --- |
| process | TAT |
| P1 | 8 - 0 = 8 |
| P2 | 13 - 0.4 =12.6 |
| P3 | 9 - 1 = 8 |
| Average TAT | = 9.53 |

1. For this case , we name it as Future Knowledge Acknowledgement so……

P3 P2 P1

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

0 1 2 6 14

As we know that ,

Turn around Time = Final time – Arrival Time

|  |  |
| --- | --- |
| process | TAT |
| P1 | 14 - 0 = 14 |
| P2 | 06 - 0.4 = 5.6 |
| P3 | 02 - 1 = 01 |
| Average TAT | = 6.87 |

CODE:-

----SJF----

#include<iostream>

using namespace std;

int mat[10][6];

void swap(int \*a, int \*b)

{

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

void arrangeArrival(int num, int mat[][6])

{

    for(int i=0; i<num; i++)

    {

        for(int j=0; j<num-i-1; j++)

        {

            if(mat[j][1] > mat[j+1][1])

            {

                for(int k=0; k<5; k++)

                {

                    swap(mat[j][k], mat[j+1][k]);

                }

            }

        }

    }

}

void completionTime(int num, int mat[][6])

{

    int temp, val;

    mat[0][3] = mat[0][1] + mat[0][2];

    mat[0][5] = mat[0][3] - mat[0][1];

    mat[0][4] = mat[0][5] - mat[0][2];

    for(int i=1; i<num; i++)

    {

        temp = mat[i-1][3];

        int low = mat[i][2];

        for(int j=i; j<num; j++)

        {

            if(temp >= mat[j][1] && low >= mat[j][2])

            {

                low = mat[j][2];

                val = j;

            }

        }

        mat[val][3] = temp + mat[val][2];

        mat[val][5] = mat[val][3] - mat[val][1];

        mat[val][4] = mat[val][5] - mat[val][2];

        for(int k=0; k<6; k++)

        {

            swap(mat[val][k], mat[i][k]);

        }

    }

}

int main()

{

    int num, temp;

    cout<<"Enter number of Process: ";

    cin>>num;

    cout<<"...Enter the process ID...\n";

    for(int i=0; i<num; i++)

    {

        cout<<"...Process "<<i+1<<"...\n";

        cout<<"Enter Process Id: ";

        cin>>mat[i][0];

        cout<<"Enter Arrival Time: ";

        cin>>mat[i][1];

        cout<<"Enter Burst Time: ";

        cin>>mat[i][2];

    }

    cout<<"Before Arrange...\n";

    cout<<"Process ID\tArrival Time\tBurst Time\n";

    for(int i=0; i<num; i++)

    {

        cout<<mat[i][0]<<"\t\t"<<mat[i][1]<<"\t\t"<<mat[i][2]<<"\n";

    }

    arrangeArrival(num, mat);

    completionTime(num, mat);

    cout<<"Final Result...\n";

    cout<<"Process ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n";

 for(int i=0; i<num; i++)

    {

        cout<<mat[i][0]<<"\t\t"<<mat[i][1]<<"\t\t"<<mat[i][2]<<"\t\t"<<mat[i][4]<<"\t\t"<<mat[i][5]<<"\n";

    }

}

b. -------- FCFS-------

#include<stdio.h>

void findWaitingTime(int processes[], int n,int bt[], int wt[])

{

// waiting time for first process is 0

wt[0] = 0;

// calculating waiting time

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

wt[i] = bt[i-1] + wt[i-1] ;

}

// Function to calculate turn around time

void findTurnAroundTime( int processes[], int n,

int bt[], int wt[], int tat[])

{

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

tat[i] = bt[i] + wt[i];

}

//Function to calculate average time

void findavgTime( int processes[], int n, int bt[])

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

//Function to find waiting time of all processes

findWaitingTime(processes, n, bt, wt);

//Function to find turn around time for all processes

findTurnAroundTime(processes, n, bt, wt, tat);

//Display processes along with all details

printf("Processes Burst time Waiting time Turn around time\n");

// Calculate total waiting time and total turn

// around time

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

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

printf(" %d ",(i+1));

printf(" %d ", bt[i] );

printf(" %d",wt[i] );

printf(" %d\n",tat[i] );

}

int s=(float)total\_wt / (float)n;

int t=(float)total\_tat / (float)n;

printf("Average waiting time = %d",s);

printf("\n");

printf("Average turn around time = %d ",t);

}

int main()

{

//process id's

int processes[] = { 1, 2, 3};

int n = sizeof processes / sizeof processes[0];

//Burst time of all processes

int burst\_time[] = {10, 5, 8};

findavgTime(processes, n, burst\_time);

return 0;

}

Code 2:

#include <bits/stdc++.h>

using namespace std;

struct Process {

int pid; // Process ID

int bt; // Burst Time

int art; // Arrival Time

};

// Function to find the waiting time for all

// processes

void findWaitingTime(Process proc[], int n,

int wt[])

{

int rt[n];

// Copy the burst time into rt[]

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

rt[i] = proc[i].bt;

int complete = 0, t = 0, minm = INT\_MAX;

int shortest = 0, finish\_time;

bool check = false;

// Process until all processes gets

// completed

while (complete != n) {

// Find process with minimum

// remaining time among the

// processes that arrives till the

// current time`

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

if ((proc[j].art <= t) &&

(rt[j] < minm) && rt[j] > 0) {

minm = rt[j];

shortest = j;

check = true;

}

}

if (check == false) {

t++;

continue;

}

// Reduce remaining time by one

rt[shortest]--;

// Update minimum

minm = rt[shortest];

if (minm == 0)

minm = INT\_MAX;

// If a process gets completely

// executed

if (rt[shortest] == 0) {

// Increment complete

complete++;

check = false;

// Find finish time of current

// process

finish\_time = t + 1;

// Calculate waiting time

wt[shortest] = finish\_time -

proc[shortest].bt -

proc[shortest].art;

if (wt[shortest] < 0)

wt[shortest] = 0;

}

// Increment time

t++;

}

}

// Function to calculate turn around time

void findTurnAroundTime(Process proc[], int n,

int wt[], int tat[])

{

// calculating turnaround time by adding

// bt[i] + wt[i]

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

tat[i] = proc[i].bt + wt[i];

}

// Function to calculate average time

void findavgTime(Process proc[], int n)

{

int wt[n], tat[n], total\_wt = 0,

total\_tat = 0;

// Function to find waiting time of all

// processes

findWaitingTime(proc, n, wt);

// Function to find turn around time for

// all processes

findTurnAroundTime(proc, n, wt, tat);

// Display processes along with all

// details

printf("Processes Burst time Waiting time Turn around time\n");

// Calculate total waiting time and

// total turnaround time

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

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

printf(" %d ",(i+1));

printf(" %d ", bt[i] );

printf(" %d",wt[i] );

printf(" %d\n",tat[i]

}

printf("Average waiting time = %d",s);

printf("\n");

printf("Average turn around time = %d ",t);

}

// Driver code

int main()

{

Process proc[] = { { 1, 6, 1 }, { 2, 8, 1 },

{ 3, 7, 2 }, { 4, 3, 3 } };

int n = sizeof(proc) / sizeof(proc[0]);

findavgTime(proc, n);

return 0;

}

Problems: -

* In SJF Scheduling, a process with high burst time may suffer starvation. Starvation is the process in which a process with higher burst time is kept on waiting and waiting , but is not allocated to the CPU. It’s prevented by **aging.**
* Total execution time must be known beforehand of a process.
* IN FCFS, If the CPU gets the processes of the higher burst time at the front end of the ready queue then the processes of lower burst time may get blocked which means they may never get the CPU if the job in the execution has a very high burst time. This is called **starvation**.

Algorithm: - SJF

* Sort all the process according to the arrival time.
* Then select that process which has minimum arrival time and minimum Burst time.
* After completion of process make a pool of process which after till the completion of previous process and select that process among the pool which is having minimum Burst time.

FCFS

* First in, first out (FIFO), also known as first come, first served (FCFS), is the simplest scheduling algorithm. FIFO simply queues processes in the order that they arrive in the ready queue.  
  In this, the process that comes first will be executed first and next process starts only after the previous gets fully executed.

Constraints: -

After execution of simulation codes of FCFS and SJF I come to the conclusion that SJF algorithm is more competitive than FCFS If we talk about performance because It has minimal average waiting that is very important. SJF main flaw is a very long process to be not executed at the time required, thing that does not happen to FCFS algorithm.

  🡪  for(int i=1; i<num; i++)

    {

        temp = mat[i-1][3];

        int low = mat[i][2];

        for(int j=i; j<num; j++)

        {

            if(temp >= mat[j][1] && low >= mat[j][2])

            {

                low = mat[j][2];

                val = j;

            }

        }

Complexity: -

O(nlogn) – SJF

O(n) - FCFS

Boundary conditions: -

There is only one boundary condition that it is a non-preemitive approach it has to wait long SJF main flaw is a very long process to be not executed at the time required, thing that does not happen to FCFS algorithm.

Test Cases: -

1. FCFS for solving processes of both queue with proper selection and comparison.
2. Minimization of waiting time.
3. SJF for solving and then using the future knowledge acknowledge approach for the least average least turn around time.