Name:- Ajinkya Narendra Mahajan

Class:- TY-AI&DS-B

Roll no :- 60

Lab 8:- **Implementation of FCFC and SRTF**

1. **FCFS**

// C program for implementation of FCFS

// scheduling

#include<stdio.h>

// Function to find the waiting time for all

// processes

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[])

{

// calculating turnaround time by adding

// bt[i] + wt[i]

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);

}

// Driver code

int main()

{

//process id's

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

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

//Burst time of all processes

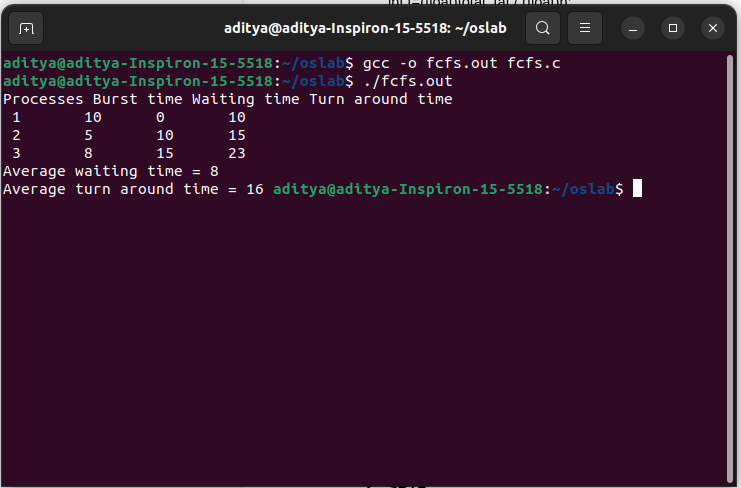
int burst\_time[] = {10, 5, 8};

findavgTime(processes, n, burst\_time);

return 0;

}

**Output:-**

****

1. **SRTF**

// C++ program to implement Shortest Remaining Time First

// Shortest Remaining Time First (SRTF)

#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

cout << " P\t\t"

<< "BT\t\t"

<< "WT\t\t"

<< "TAT\t\t\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];

cout << " " << proc[i].pid << "\t\t"

<< proc[i].bt << "\t\t " << wt[i]

<< "\t\t " << tat[i] << endl;

}

cout << "\nAverage waiting time = "

<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

// Driver code

int main()

{

Process proc[] = { { 1, 6, 2 }, { 2, 2, 5 },

{ 3, 8, 1 }, { 4, 3, 0}, {5, 4, 4} };

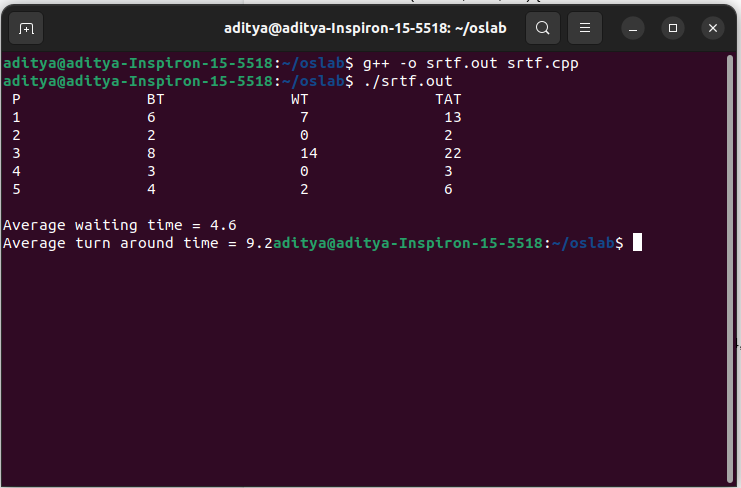
int n = sizeof(proc) / sizeof(proc[0]);

findavgTime(proc, n);

return 0;

}

**Output:-**

****

Lab 9:- **Priority Scheduling- Preemptive by considering High No High Priority**

#include<stdio.h>

// structure representing a structure

struct priority\_scheduling {

// name of the process

char process\_name;

// time required for execution

int burst\_time;

// waiting time of a process

int waiting\_time;

// total time of execution

int turn\_around\_time;

// priority of the process

int priority;

};

int main() {

// total number of processes

int number\_of\_process;

// total waiting and turnaround time

int total = 0;

// temporary structure for swapping

struct priority\_scheduling temp\_process;

// ASCII numbers are used to represent the name of the process

int ASCII\_number = 65;

// swapping position

int position;

// average waiting time of the process

float average\_waiting\_time;

// average turnaround time of the process

float average\_turnaround\_time;

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

// get the total number of the process as input

scanf("%d", & number\_of\_process);

// initializing the structure array

struct priority\_scheduling process[number\_of\_process];

printf("\nPlease Enter the Burst Time and Priority of each process:\n");

// get burst time and priority of all process

for (int i = 0; i < number\_of\_process; i++) {

// assign names consecutively using ASCII number

process[i].process\_name = (char) ASCII\_number;

printf("\nEnter the details of the process %c \n", process[i].process\_name);

printf("Enter the burst time: ");

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

printf("Enter the priority: ");

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

// increment the ASCII number to get the next alphabet

ASCII\_number++;

}

// swap process according to high priority

for (int i = 0; i < number\_of\_process; i++) {

position = I;

for (int j = i + 1; j < number\_of\_process; j++) {

// check if priority is higher for swapping

if (process[j].priority > process[position].priority)

position = j;

}

// swapping of lower priority process with the higher priority process

temp\_process = process[i];

process[i] = process[position];

process[position] = temp\_process;

}

// First process will not have to wait and hence has a waiting time of 0

process[0].waiting\_time = 0;

for (int i = 1; i < number\_of\_process; i++) {

process[i].waiting\_time = 0;

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

// calculate waiting time

process[i].waiting\_time += process[j].burst\_time;

}

// calculate total waiting time

total += process[i].waiting\_time;

}

// calculate average waiting time

average\_waiting\_time = (float) total / (float) number\_of\_process;

// assigning total as 0 for next calculations

total = 0;

printf("\n\nProcess\_name \t Burst Time \t Waiting Time \t Turnaround Time\n");

printf("------------------------------------------------------------\n");

for (int i = 0; i < number\_of\_process; i++) {

// calculating the turn around time of the processes

process[i].turn\_around\_time = process[i].burst\_time + process[i].waiting\_time;

// calculating the total turnaround time.

total += process[i].turn\_around\_time;

// printing all the values

printf("\t %c \t\t %d \t\t %d \t\t %d", process[i].process\_name, process[i].burst\_time, process[i].waiting\_time, process[i].turn\_around\_time);

printf("\n-----------------------------------------------------------\n");

}

// calculating the average turn\_around time

average\_turnaround\_time = (float) total / (float) number\_of\_process;

// average waiting time

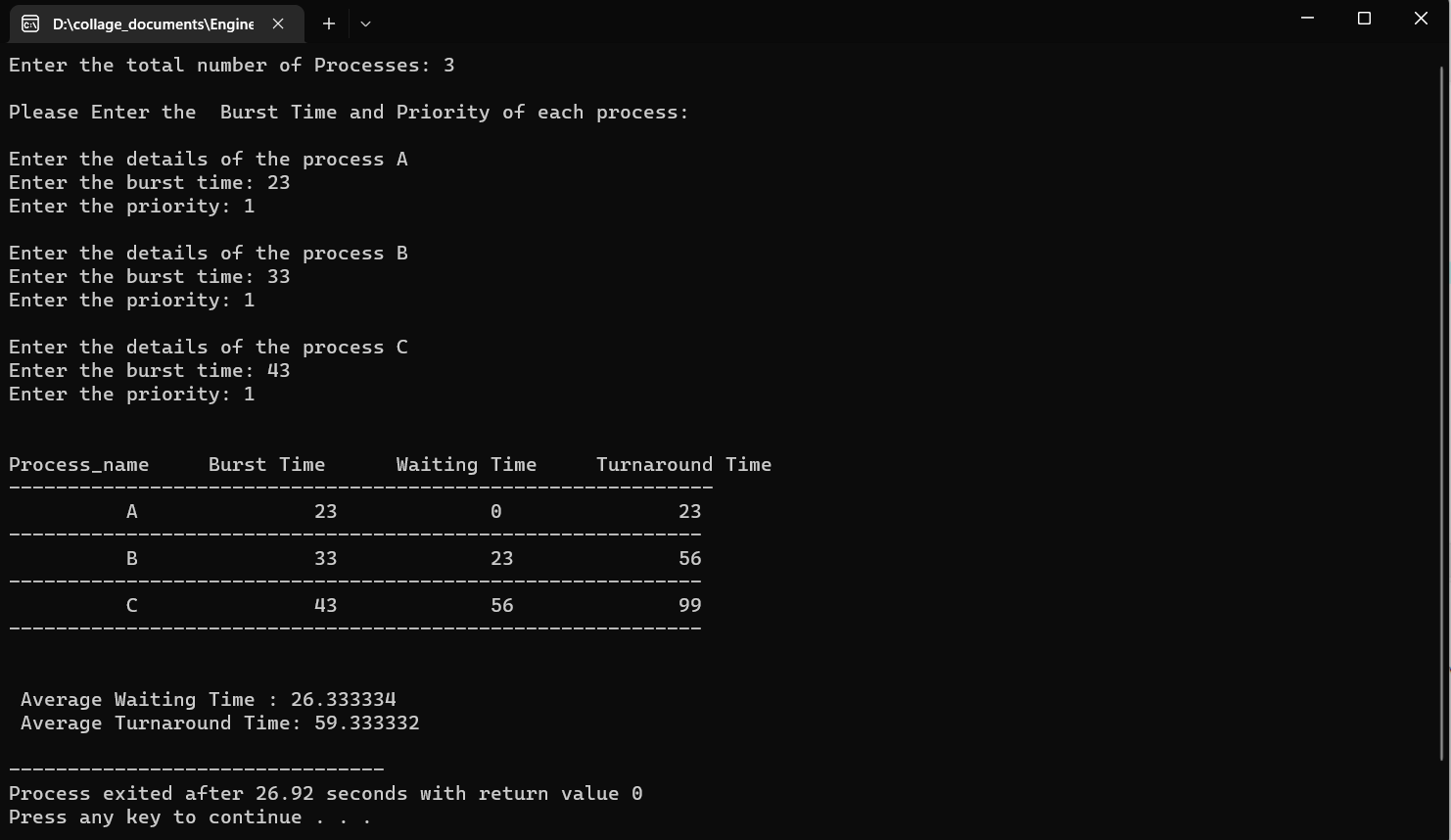
printf("\n\n Average Waiting Time : %f", average\_waiting\_time);

// average turnaround time

printf("\n Average Turnaround Time: %f\n", average\_turnaround\_time);

return 0;

}

**Output:- **