**Lab 5: Scheduling Methods**

**Objectives:**

The primary objective of a scheduling method in a lab report is to efficiently allocate resources and tasks to optimize productivity and minimize idle time. It aims to ensure that all processes are completed within the stipulated time frame while maintaining a high level of accuracy and consistency. By implementing a systematic scheduling approach, the method seeks to balance workload distribution and prevent bottlenecks that could delay overall progress. Additionally, it aims to enhance resource utilization, ensuring that equipment and personnel are used to their full potential. Ultimately, the scheduling method strives to improve the overall efficiency and effectiveness of lab operations, leading to more reliable and timely results.

1. **Write a menu driven program where the user gets to choose her/his choice of scheduling method.**

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct process {

int pid, AT, BT, WT, TAT;

};

struct process a[10];

int queue[100];

int front = -1;

int rear = -1;

void inputArrivalTime(int n, int \*arrivalTime);

void inputBurstTime(int n, int \*burstTime, int \*p);

void swap(int \*x, int \*y);

void sortingFCFS(int \*p, int n, int \*arrivalTime, int \*burstTime);

void sortingSJF(int n, int \*at, int \*bt, int \*p);

float calculateAvgTATime(int \*TATime, int n);

float calculateAvgWaitingTime(int \*WaitingTime, int n);

void FCFS(int n, int \*TATime, int \*arrivalTime, int \*burstTime, int \*waitingTime, int \*p);

void SJF(int n, int \*TATime, int \*arrivalTime, int \*burstTime, int \*waitingTime, int \*p);

void SRTN(int n);

void RR(int n);

void insert(int n);

int delete();

int main() {

int choice;

while (1) {

printf("\nProcess scheduling algorithm: \n");

printf("1. First Come First Serve\n");

printf("2. Shortest Job First (non-preemptive)\n");

printf("3. Shortest Remaining Time Next (preemptive)\n");

printf("4. Round Robin\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

int n;

float avgWaiting, avgTA;

if (choice == 5) {

printf("Terminating program!!\n");

exit(0);

}

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

scanf("%d", &n);

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

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

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

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

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

switch (choice) {

case 1:

printf(" >>>>First Come First Serve<<<<\n");

printf("Number of processes: %d\n", n);

printf("Enter the arrival time : ");

inputArrivalTime(n, arrivalTime);

printf("Enter the burst time : ");

inputBurstTime(n, burstTime, Process);

FCFS(n, TATime, arrivalTime, burstTime, waitingTime, Process);

printf("\n===================================================================================== \n\n");

break;

case 2:

printf(" >>>>Shortest Job First (non-preemptive)<<<<\n");

printf("Number of processes: %d\n", n);

printf("Enter the arrival time : ");

inputArrivalTime(n, arrivalTime);

printf("Enter the burst time : ");

inputBurstTime(n, burstTime, Process);

SJF(n, TATime, arrivalTime, burstTime, waitingTime, Process);

printf("\n===================================================================================== \n\n");

break;

case 3:

printf(" >>>>Shortest Remaining Time Next (preemptive)<<<<\n");

printf("Number of processes: %d\n", n);

SRTN(n);

printf("\n===================================================================================== \n\n");

break;

case 4:

printf(" >>>>Round Robin<<<<\n");

printf("Number of processes: %d\n", n);

RR(n);

printf("\n===================================================================================== \n\n");

break;

default:

printf("\n===================================================================================== \n\n");

printf("Please enter correct options!\n\n");

break;

}

free(arrivalTime);

free(burstTime);

free(waitingTime);

free(TATime);

free(Process);

}

}

void inputArrivalTime(int n, int \*arrivalTime) {

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

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

}

void inputBurstTime(int n, int \*burstTime, int \*p) {

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

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

p[i] = i + 1;

}

}

void swap(int \*x, int \*y) {

int temp = \*x;

\*x = \*y;

\*y = temp;

}

void sortingFCFS(int \*p, int n, int \*arrivalTime, int \*burstTime) {

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

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

if (arrivalTime[i] > arrivalTime[j]) {

swap(&p[i], &p[j]);

swap(&arrivalTime[i], &arrivalTime[j]);

swap(&burstTime[i], &burstTime[j]);

}

}

}

}

float calculateAvgTATime(int \*TATime, int n) {

float average, sum = 0;

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

sum += TATime[i];

average = sum / n;

return average;

}

float calculateAvgWaitingTime(int \*WaitingTime, int n) {

float average, sum = 0;

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

sum += WaitingTime[i];

average = sum / n;

return average;

}

void sortingSJF(int n, int \*at, int \*bt, int \*p) {

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

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

if (at[i] > at[j]) {

swap(&p[i], &p[j]);

swap(&at[i], &at[j]);

swap(&bt[i], &bt[j]);

} else if (at[i] == at[j]) {

if (bt[i] > bt[j]) {

swap(&p[i], &p[j]);

swap(&at[i], &at[j]);

swap(&bt[i], &bt[j]);

}

}

}

}

}

void insert(int n) {

if (front == -1)

front = 0;

rear = rear + 1;

queue[rear] = n;

}

int delete() {

int n;

n = queue[front];

front = front + 1;

return n;

}

void FCFS(int n, int \*TATime, int \*arrivalTime, int \*burstTime, int \*waitingTime, int \*p) {

float avgWaiting, avgTA;

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

int temp;

sortingFCFS(p, n, arrivalTime, burstTime);

completionTime[0] = arrivalTime[0] + burstTime[0];

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

temp = 0;

if (completionTime[i - 1] < arrivalTime[i]) {

temp = arrivalTime[i] - completionTime[i - 1];

}

completionTime[i] = completionTime[i - 1] + burstTime[i] + temp;

}

printf("\nProcess\t\tArrival Time\t\tBurst Time\t\tCompletion Time\t\tTurnAround Time\t\tWaiting Time");

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

TATime[i] = completionTime[i] - arrivalTime[i];

waitingTime[i] = TATime[i] - burstTime[i];

printf("\nP[%d]\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d", p[i], arrivalTime[i], burstTime[i], completionTime[i], TATime[i], waitingTime[i]);

}

avgWaiting = calculateAvgWaitingTime(waitingTime, n);

avgTA = calculateAvgTATime(TATime, n);

printf("\nThe average waiting time is %.3f\n", avgWaiting);

printf("The average TurnAround time is %.3f\n", avgTA);

free(completionTime);

}

void SJF(int n, int \*TATime, int \*arrivalTime, int \*burstTime, int \*waitingTime, int \*p) {

float avgWaiting, avgTA;

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

int pos;

int min = 1000;

sortingSJF(n, arrivalTime, burstTime, p);

completionTime[0] = arrivalTime[0] + burstTime[0];

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

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

if (arrivalTime[j] <= completionTime[i - 1]) {

if (burstTime[j] < min) {

min = burstTime[j];

pos = j;

}

}

}

swap(&p[i], &p[pos]);

swap(&arrivalTime[i], &arrivalTime[pos]);

swap(&burstTime[i], &burstTime[pos]);

completionTime[i] = completionTime[i - 1] + burstTime[i];

}

printf("\nProcess\t\tArrival Time\t\tBurst Time\t\tCompletion Time\t\tTurnAround Time\t\tWaiting Time");

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

TATime[i] = completionTime[i] - arrivalTime[i];

waitingTime[i] = TATime[i] - burstTime[i];

printf("\nP[%d]\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d", p[i], arrivalTime[i], burstTime[i], completionTime[i], TATime[i], waitingTime[i]);

}

avgWaiting = calculateAvgWaitingTime(waitingTime, n);

avgTA = calculateAvgTATime(TATime, n);

printf("\nThe average waiting time is %.3f\n", avgWaiting);

printf("The average TurnAround time is %.3f\n", avgTA);

free(completionTime);

}

void SRTN(int n) {

int ari[10], bur[10], total = 0, i, j, small, temp, procs[100], k, waiting[10], finish[10];

float tavg = 0.0, wavg = 0.0;

printf("Enter the arrival time: ");

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

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

}

printf("Enter the burst time: ");

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

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

waiting[i] = 0;

total += bur[i];

}

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

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

if (ari[i] > ari[j]) {

temp = ari[i];

ari[i] = ari[j];

ari[j] = temp;

temp = bur[i];

bur[i] = bur[j];

bur[j] = temp;

}

}

}

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

small = 3200;

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

if ((bur[j] != 0) && (ari[j] <= i) && (bur[j] < small)) {

small = bur[j];

k = j;

}

}

procs[i] = k;

bur[k]--;

}

k = 0;

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

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

if (procs[i] == j) {

finish[j] = i + 1;

waiting[j]++;

}

}

}

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

printf("\nP[%d]\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d", i + 1, ari[i], bur[i] + 1, finish[i], (finish[i] - ari[i] + 1), (finish[i] - ari[i] + 1) - (bur[i] + 1));

wavg = wavg + ((finish[i] - ari[i] + 1) - (bur[i] + 1));

tavg = tavg + (finish[i] - ari[i] + 1);

}

printf("\n\nThe average waiting time is %.3f\n", (wavg / n));

printf("The average TurnAround time is %.3f\n", (tavg / n));

}

void RR(int n ) {

int TQ,p,TIME=0;

int temp[10],exist[10]={0};

float total\_wt=0,total\_tat=0,Avg\_WT,Avg\_TAT;

printf("Enter the arrival time: ");

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

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

a[i].pid=i+1;

}

printf("Enter the burst time: ");

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

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

temp[i]=a[i].BT;

}

printf("Enter the time quantum: ");

scanf("%d",&TQ);

insert(0);

exist[0]=1;

while(front<=rear) {

p=delete();

if(a[p].BT>=TQ) {

a[p].BT=a[p].BT-TQ;

TIME=TIME+TQ;

} else {

TIME=TIME+a[p].BT;

a[p].BT=0;

}

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

if(exist[i]==0 && a[i].AT<=TIME) {

insert(i);

exist[i]=1;

}

}

// if process is completed

if(a[p].BT==0) {

a[p].TAT=TIME-a[p].AT;

a[p].WT=a[p].TAT-temp[p];

total\_tat=total\_tat+a[p].TAT;

total\_wt=total\_wt+a[p].WT;

} else {

insert(p);

}

}

Avg\_TAT=total\_tat/n;

Avg\_WT=total\_wt/n;

// printing of the answer

printf("\nProcess\t\tArrival Time\t\tBrust Time\t\tTurnAround Time\t\tWaiting Time");

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

printf("\nP[%d]\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d",a[i].pid,a[i].AT,temp[i],a[i].TAT, a[i].WT);

}

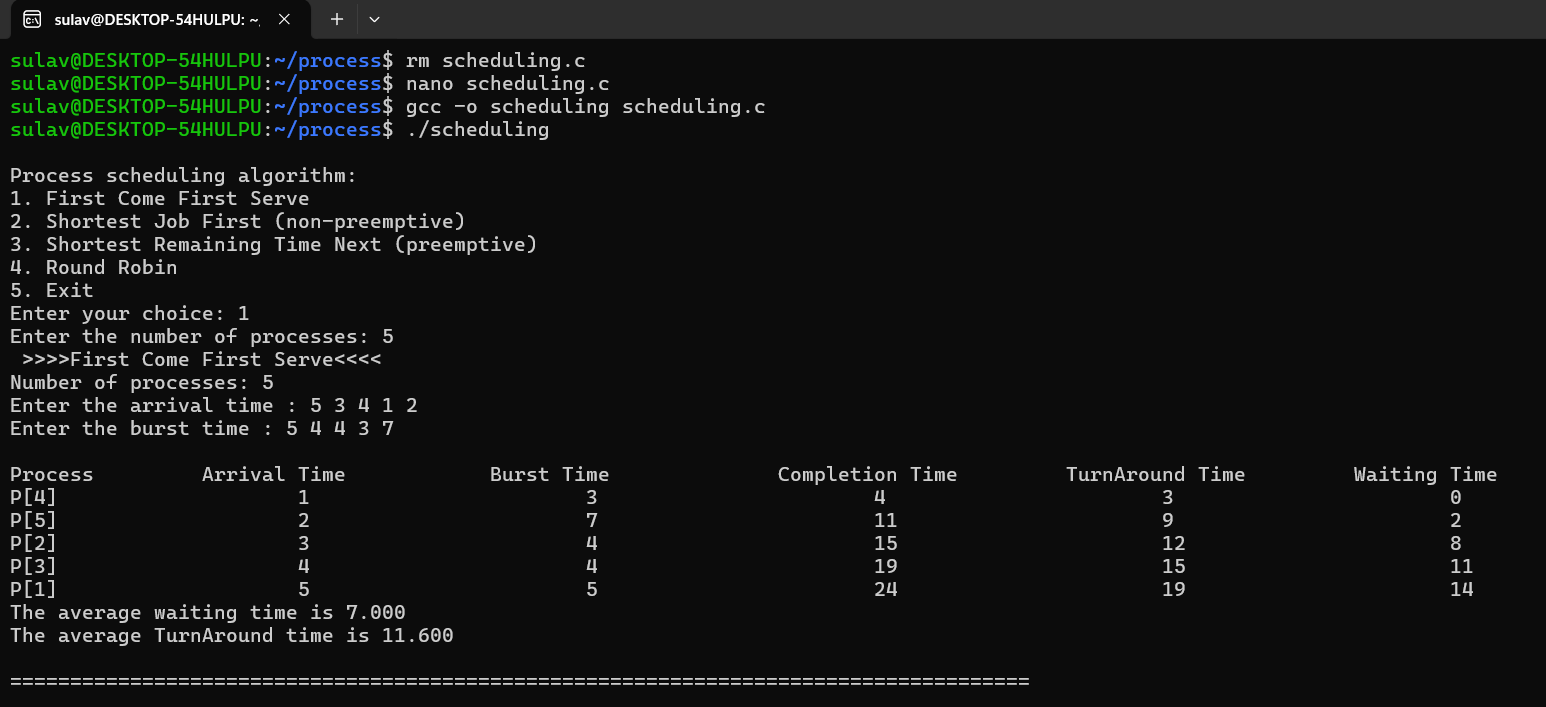
printf("\n\nThe average waiting time is %.3f\n",Avg\_WT );

printf("The average TurnAround time is %.3f\n",Avg\_TAT);

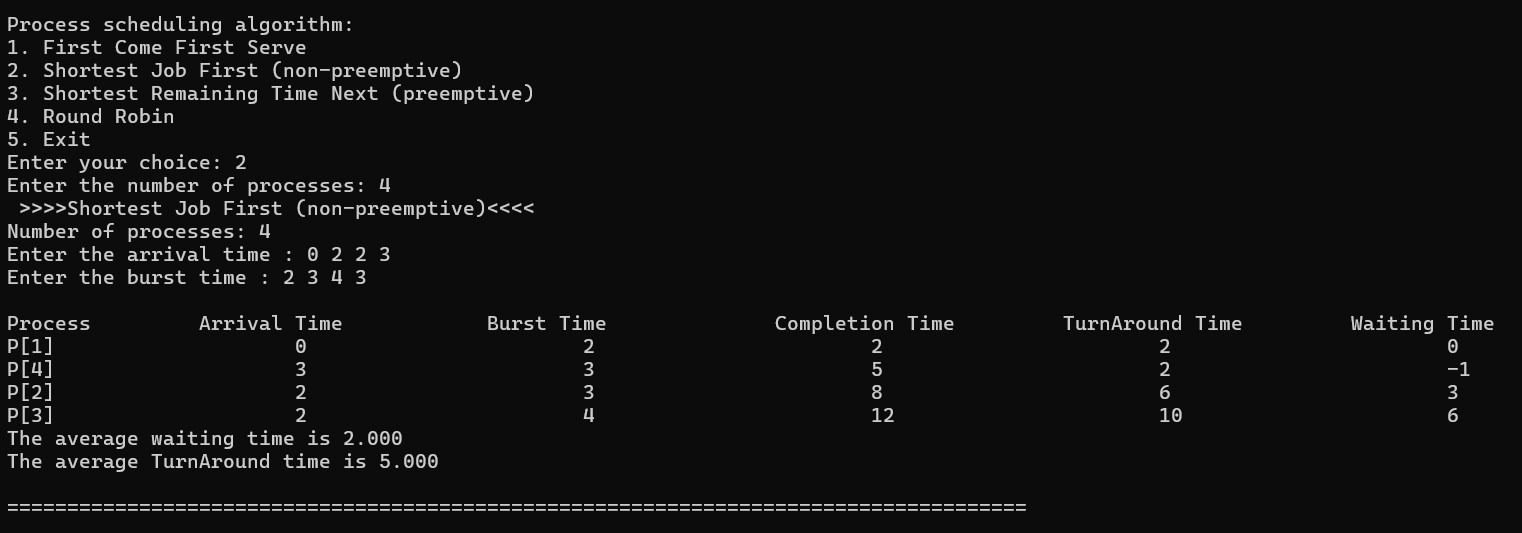
}

**Output:**

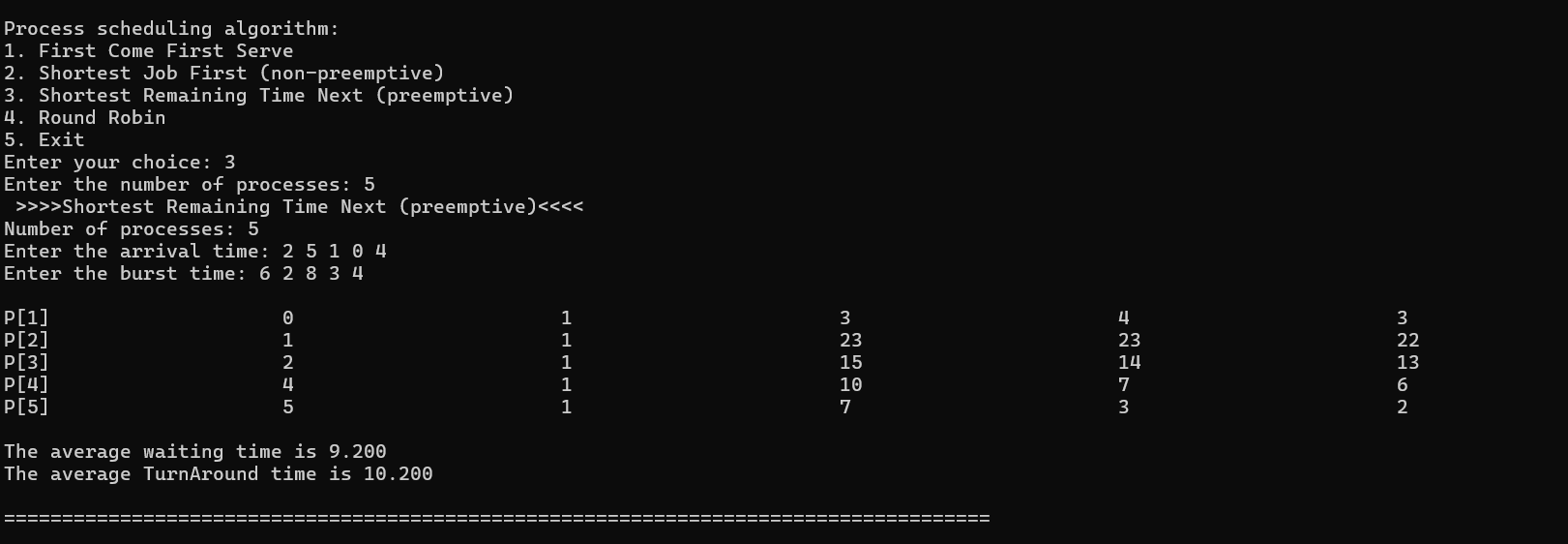
**For First Come First Serve:**

****

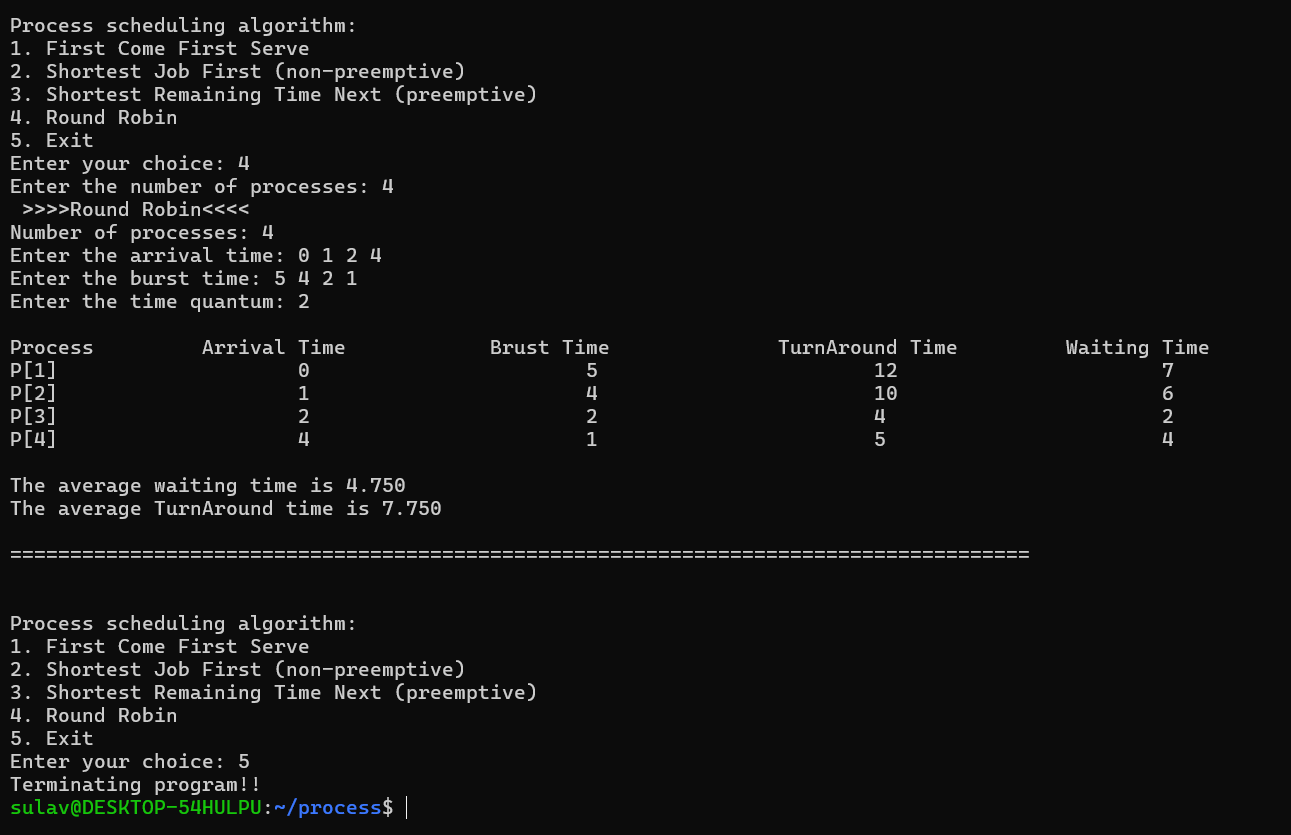
**For Shortest Job First:**

****

**For Shortest Remaining Time Next:**

****

**For Round Robin:**

****

**Conclusion:**

In conclusion, the scheduling method demonstrated its effectiveness in optimizing lab operations by ensuring efficient resource allocation and task management. It successfully minimized idle time and prevented bottlenecks, thereby enhancing overall productivity. The systematic approach to workload distribution and resource utilization resulted in timely and reliable outcomes. By maintaining a high level of accuracy and consistency, the scheduling method contributed significantly to the success of the lab's objectives. Overall, the implementation of this method has proven to be a valuable strategy for improving efficiency and achieving optimal results in the laboratory setting.