1

**OS Lab: 8, 9, 10, 11**

****

**Department of Computer Science**

**Iqra University Islamabad**

**Operating System**

**Maqsood Ahmed**

**ID: 38186**

**Lab: 08**

**To write a C program for implementation of Priority scheduling algorithms.**

**Source Code:**

#include <stdio.h>

#include <stdbool.h>

// Structure to represent a process

struct Process {

int pid; // Process ID

int priority; // Priority of the process

int burstTime; // Burst time of the process

int arrivalTime; // Arrival time of the process

int waitingTime; // Waiting time of the process

int turnAroundTime; // Turnaround time of the process

bool isCompleted; // Flag to check if the process is completed

};

// Function to find the next process to schedule based on priority and arrival time

int findNextProcess(struct Process p[], int n, int currentTime) {

int minPriority = 100000, nextProcessIndex = -1;

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

if (!p[i].isCompleted && p[i].arrivalTime <= currentTime) {

if (p[i].priority < minPriority) {

minPriority = p[i].priority;

nextProcessIndex = i;

}

}

}

return nextProcessIndex;

}

// Function to calculate waiting and turnaround times

void calculateTimes(struct Process p[], int n) {

int currentTime = 0, completed = 0;

while (completed < n) {

int idx = findNextProcess(p, n, currentTime);

if (idx == -1) {

currentTime++; // If no process is ready, increment time

continue;

}

// Schedule the process

currentTime += p[idx].burstTime;

p[idx].waitingTime = currentTime - p[idx].arrivalTime - p[idx].burstTime;

if (p[idx].waitingTime < 0) p[idx].waitingTime = 0; // No negative waiting time

p[idx].turnAroundTime = currentTime - p[idx].arrivalTime;

p[idx].isCompleted = true;

completed++;

}

}

// Function to calculate average waiting and turnaround times

void calculateAverages(struct Process p[], int n, float \*avgWait, float \*avgTurnAround) {

int totalWait = 0, totalTurnAround = 0;

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

totalWait += p[i].waitingTime;

totalTurnAround += p[i].turnAroundTime;

}

\*avgWait = (float)totalWait / n;

\*avgTurnAround = (float)totalTurnAround / n;

}

// Function to display the process information

void displayProcesses(struct Process p[], int n) {

printf("\nPID\tPriority\tBurst Time\tArrival Time\tWaiting Time\tTurnaround Time\n");

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

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

printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",

p[i].pid, p[i].priority, p[i].burstTime, p[i].arrivalTime,

p[i].waitingTime, p[i].turnAroundTime);

}

}

// Main function

int main() {

int n;

float avgWait, avgTurnAround;

// Input: Number of processes

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

scanf("%d", &n);

struct Process processes[n];

// Input: Process details

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

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

processes[i].pid = i + 1;

printf("Enter priority (lower number means higher priority): ");

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

printf("Enter burst time: ");

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

printf("Enter arrival time: ");

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

processes[i].isCompleted = false; // Mark process as not completed

}

// Step 1: Calculate times

calculateTimes(processes, n);

// Step 2: Calculate averages

calculateAverages(processes, n, &avgWait, &avgTurnAround);

// Step 3: Display processes and metrics

displayProcesses(processes, n);

// Step 4: Display average waiting and turnaround times

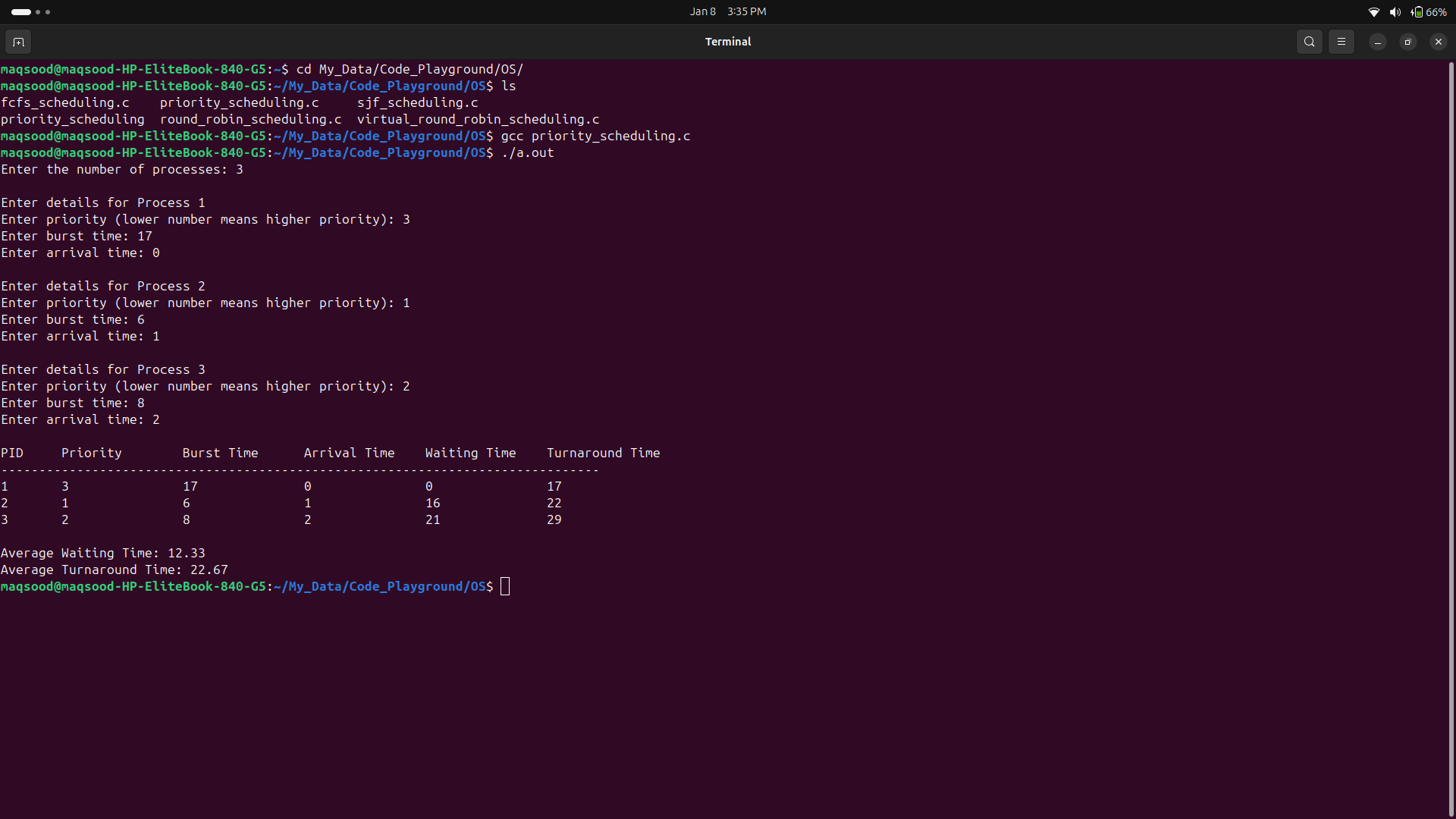
printf("\nAverage Waiting Time: %.2f\n", avgWait);

printf("Average Turnaround Time: %.2f\n", avgTurnAround);

return 0;

}

**OUTPUT:**



**Lab: 09**

**To write a C program for implementation of Round Robin scheduling algorithm.**

**Source Code:**

#include <stdio.h>

#include <stdbool.h>

// Structure to represent a process

struct Process {

int pid; // Process ID

int burstTime; // Burst time of the process

int remainingTime; // Remaining time of the process

int arrivalTime; // Arrival time of the process

int waitingTime; // Waiting time of the process

int turnAroundTime; // Turnaround time of the process

};

// Function to calculate waiting and turnaround times using Round Robin

void calculateTimes(struct Process p[], int n, int timeQuantum) {

int currentTime = 0, completed = 0;

bool processCompleted[n]; // Array to track completed processes

for (int i = 0; i < n; i++) processCompleted[i] = false;

while (completed < n) {

bool found = false;

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

if (p[i].remainingTime > 0 && p[i].arrivalTime <= currentTime) {

found = true;

if (p[i].remainingTime > timeQuantum) {

// Execute for a time quantum

currentTime += timeQuantum;

p[i].remainingTime -= timeQuantum;

} else {

// Execute for remaining time and mark as complete

currentTime += p[i].remainingTime;

p[i].remainingTime = 0;

// Calculate turnaround and waiting times

p[i].turnAroundTime = currentTime - p[i].arrivalTime;

p[i].waitingTime = p[i].turnAroundTime - p[i].burstTime;

if (p[i].waitingTime < 0) p[i].waitingTime = 0;

processCompleted[i] = true;

completed++;

}

}

}

// If no process was ready, increment time

if (!found) currentTime++;

}

}

// Function to calculate average waiting and turnaround times

void calculateAverages(struct Process p[], int n, float \*avgWait, float \*avgTurnAround) {

int totalWait = 0, totalTurnAround = 0;

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

totalWait += p[i].waitingTime;

totalTurnAround += p[i].turnAroundTime;

}

\*avgWait = (float)totalWait / n;

\*avgTurnAround = (float)totalTurnAround / n;

}

// Function to display the process information

void displayProcesses(struct Process p[], int n) {

printf("\nPID\tBurst Time\tArrival Time\tWaiting Time\tTurnaround Time\n");

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

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

printf("%d\t%d\t\t%d\t\t%d\t\t%d\n",

p[i].pid, p[i].burstTime, p[i].arrivalTime,

p[i].waitingTime, p[i].turnAroundTime);

}

}

// Main function

int main() {

int n, timeQuantum;

float avgWait, avgTurnAround;

// Input: Number of processes

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

scanf("%d", &n);

struct Process processes[n];

// Input: Process details

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

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

processes[i].pid = i + 1;

printf("Enter burst time: ");

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

printf("Enter arrival time: ");

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

processes[i].remainingTime = processes[i].burstTime; // Initialize remaining time

}

// Input: Time Quantum

printf("\nEnter the time quantum: ");

scanf("%d", &timeQuantum);

// Step 1: Calculate times

calculateTimes(processes, n, timeQuantum);

// Step 2: Calculate averages

calculateAverages(processes, n, &avgWait, &avgTurnAround);

// Step 3: Display processes and metrics

displayProcesses(processes, n);

// Step 4: Display average waiting and turnaround times

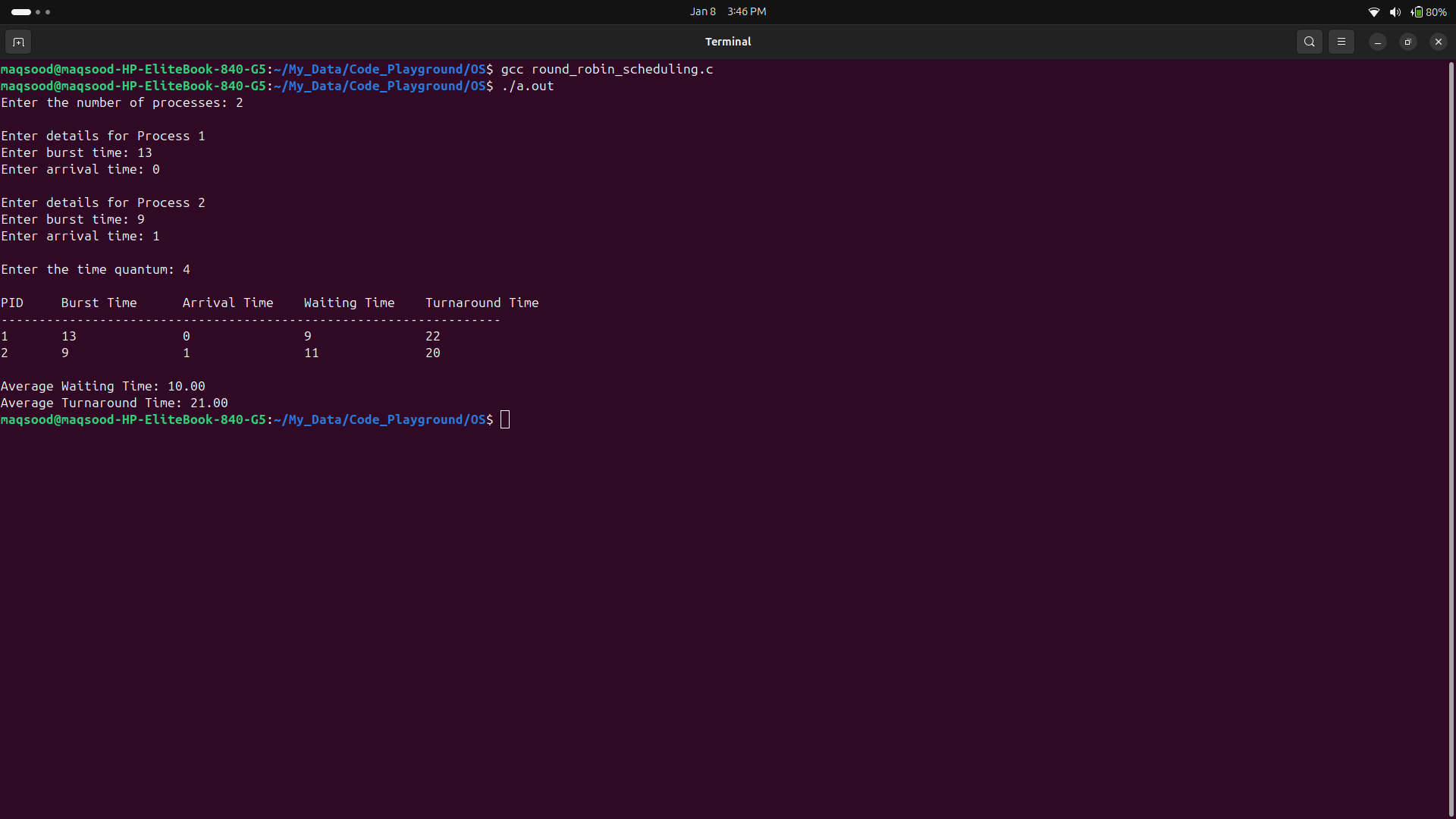
printf("\nAverage Waiting Time: %.2f\n", avgWait);

printf("Average Turnaround Time: %.2f\n", avgTurnAround);

return 0;

}

**OUTPUT:**

****

**Lab: 10**

**To write a C program for implementation of FCFS scheduling algorithms.**

**Source Code:**

#include <stdio.h>

#include <stdbool.h>

// Structure to represent a process

struct Process {

int pid; // Process ID

int burstTime; // Burst time of the process

int arrivalTime; // Arrival time of the process

int waitingTime; // Waiting time of the process

int turnAroundTime; // Turnaround time of the process

};

// Function to sort processes by arrival time

void sortByArrivalTime(struct Process p[], int n) {

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

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

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

struct Process temp = p[i];

p[i] = p[j];

p[j] = temp;

}

}

}

}

// Function to calculate waiting and turnaround times

void calculateTimes(struct Process p[], int n) {

int currentTime = 0;

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

if (currentTime < p[i].arrivalTime) {

currentTime = p[i].arrivalTime; // Wait for the process to arrive

}

p[i].waitingTime = currentTime - p[i].arrivalTime;

currentTime += p[i].burstTime;

p[i].turnAroundTime = p[i].waitingTime + p[i].burstTime;

}

}

// Function to calculate average waiting and turnaround times

void calculateAverages(struct Process p[], int n, float \*avgWait, float \*avgTurnAround) {

int totalWait = 0, totalTurnAround = 0;

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

totalWait += p[i].waitingTime;

totalTurnAround += p[i].turnAroundTime;

}

\*avgWait = (float)totalWait / n;

\*avgTurnAround = (float)totalTurnAround / n;

}

// Function to display the process information

void displayProcesses(struct Process p[], int n) {

printf("\nPID\tBurst Time\tArrival Time\tWaiting Time\tTurnaround Time\n");

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

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

printf("%d\t%d\t\t%d\t\t%d\t\t%d\n",

p[i].pid, p[i].burstTime, p[i].arrivalTime,

p[i].waitingTime, p[i].turnAroundTime);

}

}

// Main function

int main() {

int n;

float avgWait, avgTurnAround;

// Input: Number of processes

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

scanf("%d", &n);

struct Process processes[n];

// Input: Process details

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

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

processes[i].pid = i + 1;

printf("Enter burst time: ");

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

printf("Enter arrival time: ");

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

}

// Step 1: Sort by arrival time

sortByArrivalTime(processes, n);

// Step 2: Calculate times

calculateTimes(processes, n);

// Step 3: Calculate averages

calculateAverages(processes, n, &avgWait, &avgTurnAround);

// Step 4: Display processes and metrics

displayProcesses(processes, n);

// Step 5: Display average waiting and turnaround times

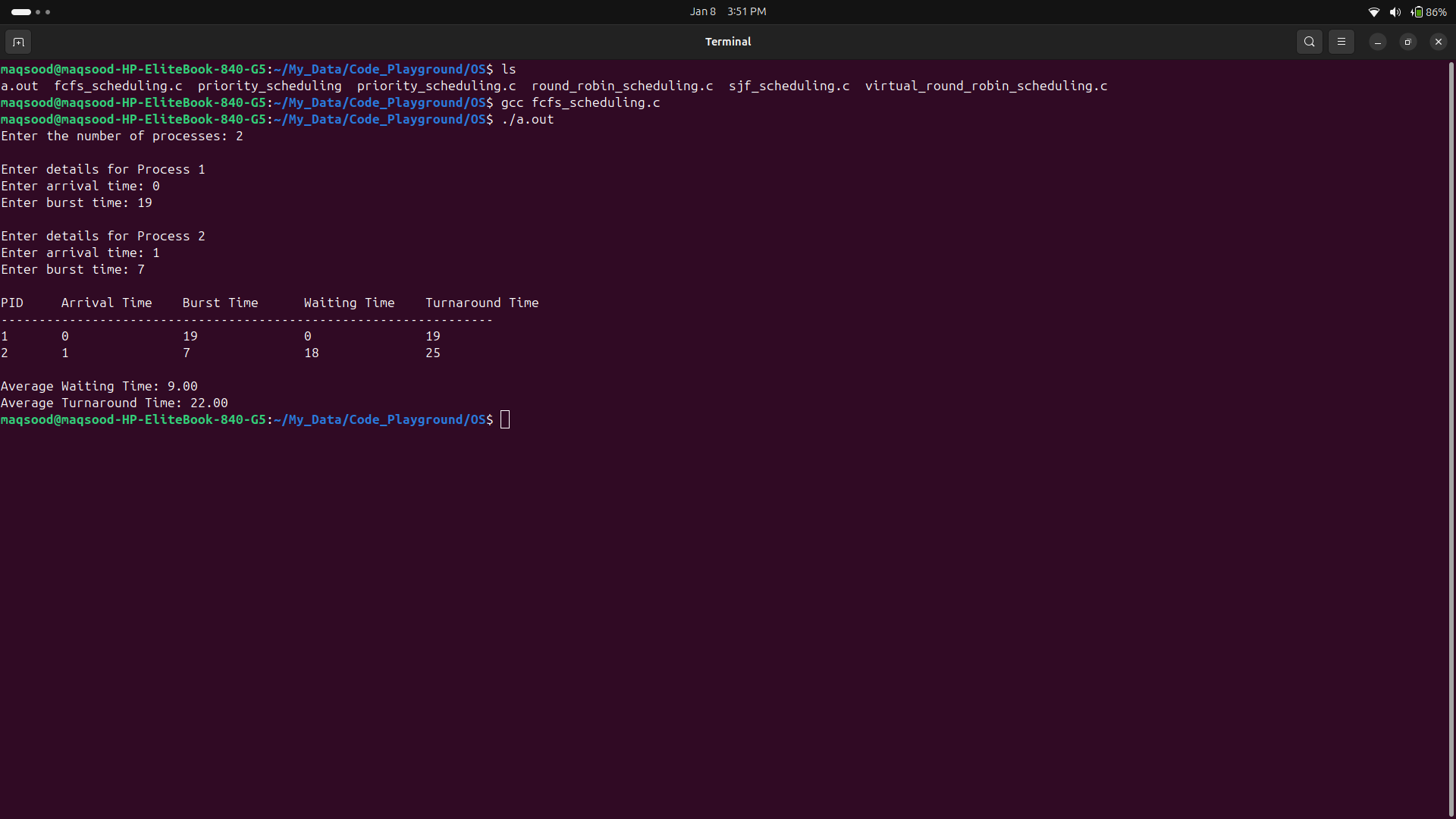
printf("\nAverage Waiting Time: %.2f\n", avgWait);

printf("Average Turnaround Time: %.2f\n", avgTurnAround);

return 0;

}

**OUTPUT:**

****

**Lab: 11**

**To write a C program for implementation of FCFS scheduling algorithms.**

**Source Code:**

#include <stdio.h>

#include <stdbool.h>

// Structure to represent a process

struct Process {

int pid; // Process ID

int burstTime; // Burst time of the process

int arrivalTime; // Arrival time of the process

int waitingTime; // Waiting time of the process

int turnAroundTime; // Turnaround time of the process

bool isCompleted; // Flag to check if the process is completed

};

// Function to find the next process to execute

int findNextProcess(struct Process p[], int n, int currentTime) {

int shortestJobIndex = -1;

int minBurstTime = 100000; // A large number to compare

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

if (!p[i].isCompleted && p[i].arrivalTime <= currentTime) {

if (p[i].burstTime < minBurstTime) {

minBurstTime = p[i].burstTime;

shortestJobIndex = i;

}

}

}

return shortestJobIndex;

}

// Function to calculate waiting and turnaround times

void calculateTimes(struct Process p[], int n) {

int currentTime = 0, completed = 0;

while (completed < n) {

int idx = findNextProcess(p, n, currentTime);

if (idx == -1) {

currentTime++; // If no process is ready, increment time

continue;

}

// Execute the shortest job

currentTime += p[idx].burstTime;

p[idx].waitingTime = currentTime - p[idx].arrivalTime - p[idx].burstTime;

if (p[idx].waitingTime < 0) p[idx].waitingTime = 0; // No negative waiting time

p[idx].turnAroundTime = currentTime - p[idx].arrivalTime;

p[idx].isCompleted = true;

completed++;

}

}

// Function to calculate average waiting and turnaround times

void calculateAverages(struct Process p[], int n, float \*avgWait, float \*avgTurnAround) {

int totalWait = 0, totalTurnAround = 0;

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

totalWait += p[i].waitingTime;

totalTurnAround += p[i].turnAroundTime;

}

\*avgWait = (float)totalWait / n;

\*avgTurnAround = (float)totalTurnAround / n;

}

// Function to display the process information

void displayProcesses(struct Process p[], int n) {

printf("\nPID\tBurst Time\tArrival Time\tWaiting Time\tTurnaround Time\n");

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

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

printf("%d\t%d\t\t%d\t\t%d\t\t%d\n",

p[i].pid, p[i].burstTime, p[i].arrivalTime,

p[i].waitingTime, p[i].turnAroundTime);

}

}

// Main function

int main() {

int n;

float avgWait, avgTurnAround;

// Input: Number of processes

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

scanf("%d", &n);

struct Process processes[n];

// Input: Process details

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

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

processes[i].pid = i + 1;

printf("Enter burst time: ");

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

printf("Enter arrival time: ");

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

processes[i].isCompleted = false; // Mark process as not completed

}

// Step 1: Calculate times

calculateTimes(processes, n);

// Step 2: Calculate averages

calculateAverages(processes, n, &avgWait, &avgTurnAround);

// Step 3: Display processes and metrics

displayProcesses(processes, n);

// Step 4: Display average waiting and turnaround times

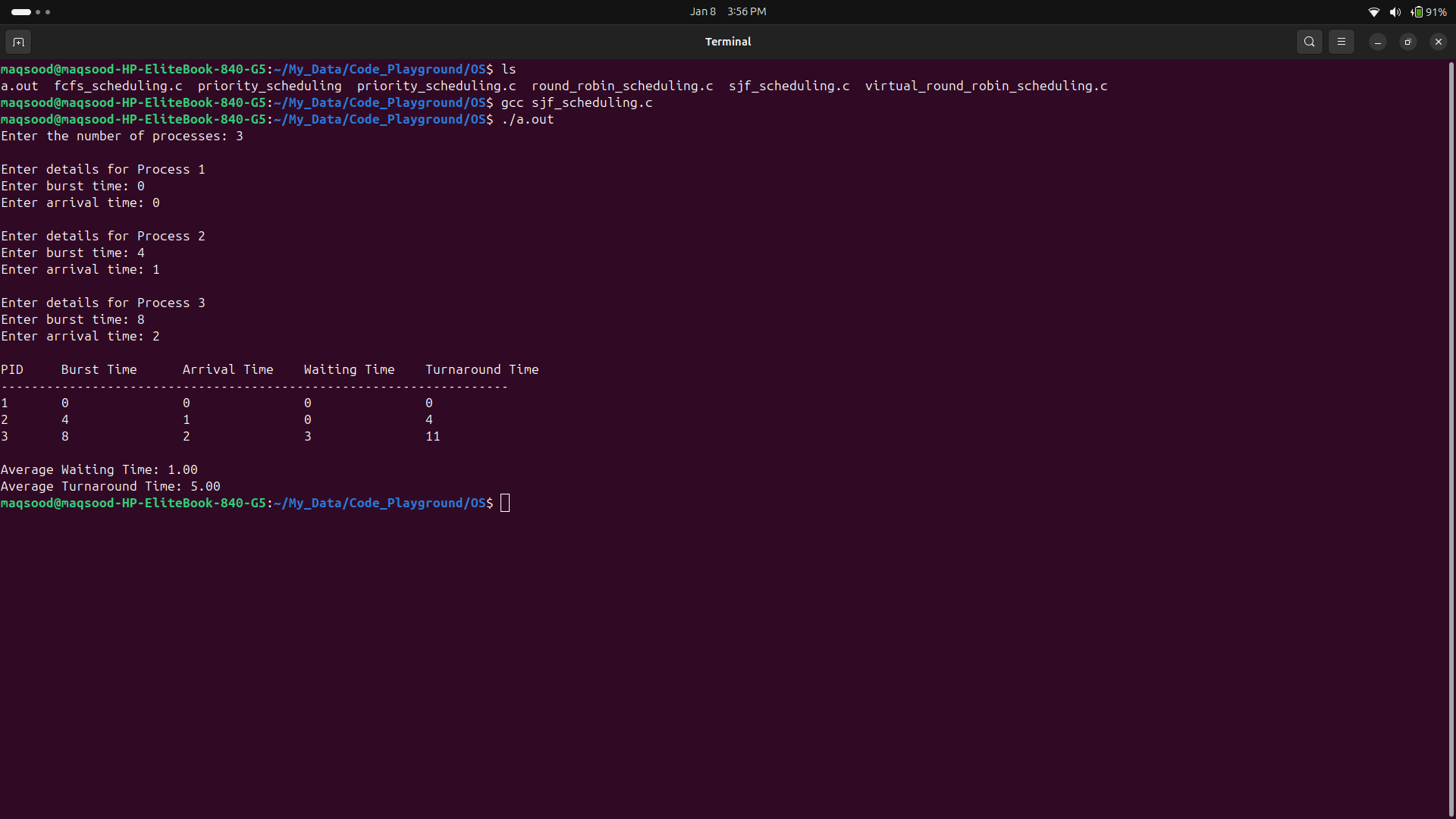
printf("\nAverage Waiting Time: %.2f\n", avgWait);

printf("Average Turnaround Time: %.2f\n", avgTurnAround);

return 0;

}

**OUTPUT:**

****

**The End**