**Name – Omkar Arun Shinde (122B1B258)**

**Assignment No. 03**

**Problem Statement :** Write a program to implement scheduling algorithms – FCFS, SJF, Round Robin and Priority.

**Code :**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

static int Count = 0;

struct Process

{

char pname[10];

int AT;

int BT;

int PR;

float CT;

float RT;

int Queue;

float TAT;

float WT;

int flag;

};

void PrintAll(struct Process P[], int n)

{

float TTAT = 0;

float TWT = 0;

printf("\nProcess\tArrival\tBurst\tPriority\tCompletion\tTurnaround\tWaiting\n");

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

{

printf("%s\t%d\t%d\t%d\t\t%.2f\t\t%.2f\t\t%.2f\n", P[i].pname, P[i].AT, P[i].BT, P[i].PR, P[i].CT, P[i].TAT, P[i].WT);

TTAT = TTAT + P[i].TAT;

TWT = TWT + P[i].WT;

}

float AvgTAT, AvgWT;

AvgTAT = TTAT / n;

AvgWT = TWT/ n;

printf("\nAverage Turnaround Time: %.2f\n", AvgTAT);

printf("Average Waiting Time: %.2f\n", AvgWT);

}

int Partition(struct Process P[], int low, int high)

{

int Pivot = P[high].AT;

int i = low - 1;

for (int j = low; j < high; j++)

{

if (P[j].AT < Pivot)

{

i++;

struct Process temp = P[i];

P[i] = P[j];

P[j] = temp;

}

}

struct Process temp = P[i + 1];

P[i + 1] = P[high];

P[high] = temp;

return i + 1;

}

void QuickSort(struct Process P[], int low, int high)

{

if (low < high)

{

int Pivot = Partition(P, low, high);

QuickSort(P, low, Pivot - 1);

QuickSort(P, Pivot + 1, high);

}

}

void CalculateTime(struct Process P[], int n)

{

int currentTime = 0;

printf("\nGantt Chart:\n|");

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

if(currentTime < P[i].AT) {

printf(" IDLE |");

currentTime = P[i].AT;

}

printf(" %s |", P[i].pname);

P[i].CT = currentTime + P[i].BT;

P[i].TAT = P[i].CT - P[i].AT;

P[i].WT = P[i].TAT - P[i].BT;

currentTime = P[i].CT;

}

printf("\n");

PrintAll(P, n);

}

void PrintGanttSchedule(struct Process P[], int Schedule[], int n) {

printf("\nGantt Chart: \n\n");

int CurrentTime = 0;

int index;

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

printf("----------------");

}

printf("\n|");

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

index = Schedule[i];

if (CurrentTime < P[index].AT) {

printf("\tIdle\t|");

CurrentTime = P[index].AT;

}

printf("\t%s\t|", P[index].pname);

CurrentTime = P[index].CT;

}

printf("\n");

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

printf("----------------");

}

printf("\n");

printf("0");

CurrentTime = 0;

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

index = Schedule[i];

if (CurrentTime < P[index].AT) {

printf("\t\t%d", P[index].AT);

CurrentTime = P[index].AT;

}

printf("\t\t%.1f", P[index].CT);

CurrentTime = P[index].CT;

}

printf("\n");

}

void PrioritySchedule(struct Process P[], int n) {

int CurrentTime = 0;

int i = 0;

int Schedule[n];

while(i < n) {

int minPR = 999;

int minIndex = -1;

int j = 0;

while(j < n) {

if(P[j].AT <= CurrentTime && P[j].PR < minPR && !P[j].flag) {

minPR = P[j].PR;

minIndex = j;

}

j++;

}

if(minIndex != -1) {

P[minIndex].CT = CurrentTime + P[minIndex].BT;

P[minIndex].TAT = P[minIndex].CT - P[minIndex].AT;

P[minIndex].WT = P[minIndex].TAT - P[minIndex].BT;

P[minIndex].flag = 1;

Schedule[i] = minIndex;

CurrentTime = P[minIndex].CT;

Count++;

i++;

}

else {

CurrentTime++;

Count++;

}

}

PrintGanttSchedule(P, Schedule, n);

}

void SJFSchedule(struct Process P[], int n) {

int CurrentTime = 0;

int i = 0;

int Schedule[n];

while(i < n) {

int minBT = 999;

int minIndex = -1;

int j = 0;

while(j < n) {

if(P[j].AT <= CurrentTime && P[j].BT < minBT && !P[j].flag) {

minBT = P[j].BT;

minIndex = j;

}

j++;

}

if(minIndex != -1) {

P[minIndex].CT = CurrentTime + P[minIndex].BT;

P[minIndex].TAT = P[minIndex].CT - P[minIndex].AT;

P[minIndex].WT = P[minIndex].TAT - P[minIndex].BT;

P[minIndex].flag = 1;

Schedule[i] = minIndex;

CurrentTime = P[minIndex].CT;

i++;

}

else {

CurrentTime++;

}

}

PrintGanttSchedule(P, Schedule, n);

}

void RoundRobin(struct Process P[], int n, int tq) {

int currentTime = 0;

int completed = 0;

int \*Schedule = (int\*)malloc(1000 \* sizeof(int));

int count = 0;

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

P[i].RT = P[i].BT;

}

while(completed < n) {

bool foundProcess = false;

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

if(P[i].RT > 0 && P[i].AT <= currentTime) {

foundProcess = true;

int executionTime = (P[i].RT < tq) ? P[i].RT : tq;

P[i].RT -= executionTime;

currentTime += executionTime;

Schedule[count++] = i;

if(P[i].RT == 0) {

completed++;

P[i].CT = currentTime;

P[i].TAT = P[i].CT - P[i].AT;

P[i].WT = P[i].TAT - P[i].BT;

}

}

}

if(!foundProcess) {

currentTime++;

Schedule[count++] = -1;

}

}

PrintGanttChart(P, n, Schedule, count);

PrintAll(P, n);

}

void PrintGanttChart(struct Process P[], int n, int Schedule[], int count) {

printf("\nGantt Chart:\n|");

int currentTime = 0;

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

if(Schedule[i] == -1) {

printf(" IDLE |");

} else {

printf(" %s |", P[Schedule[i]].pname);

}

}

printf("\n");

}

void ResetParameters(struct Process P[], int n) {

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

P[i].flag = 0;

P[i].CT = 0;

P[i].TAT = 0;

P[i].WT = 0;

P[i].RT = P[i].BT;

}

}

int main()

{

int n, tq, choice;

printf("Enter number of Processes: ");

scanf("%d", &n);

struct Process P[n];

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

{

printf("Enter Process Name: ");

scanf("%s", P[i].pname);

printf("Enter Arrival Time: ");

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

printf("Enter Burst Time: ");

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

printf("Enter Priority of the process: ");

scanf("%d", &P[i].PR);

P[i].flag = 0;

}

do {

ResetParameters(P, n);

printf("\*\*\*\*\*\*\*\*\*\*Menu\*\*\*\*\*\*\*\*\*\*\n");

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

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

printf("3. Priority Scheduling\n");

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

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: {

QuickSort(P, 0, n - 1);

CalculateTime(P, n);

}

break;

case 2: {

SJFSchedule(P, n);

PrintAll(P, n);

}

break;

case 3: {

PrioritySchedule(P, n);

PrintAll(P, n);

}

break;

case 4: {

printf("Enter the time quantum: ");

scanf("%d", &tq);

RoundRobin(P, n, tq);

}

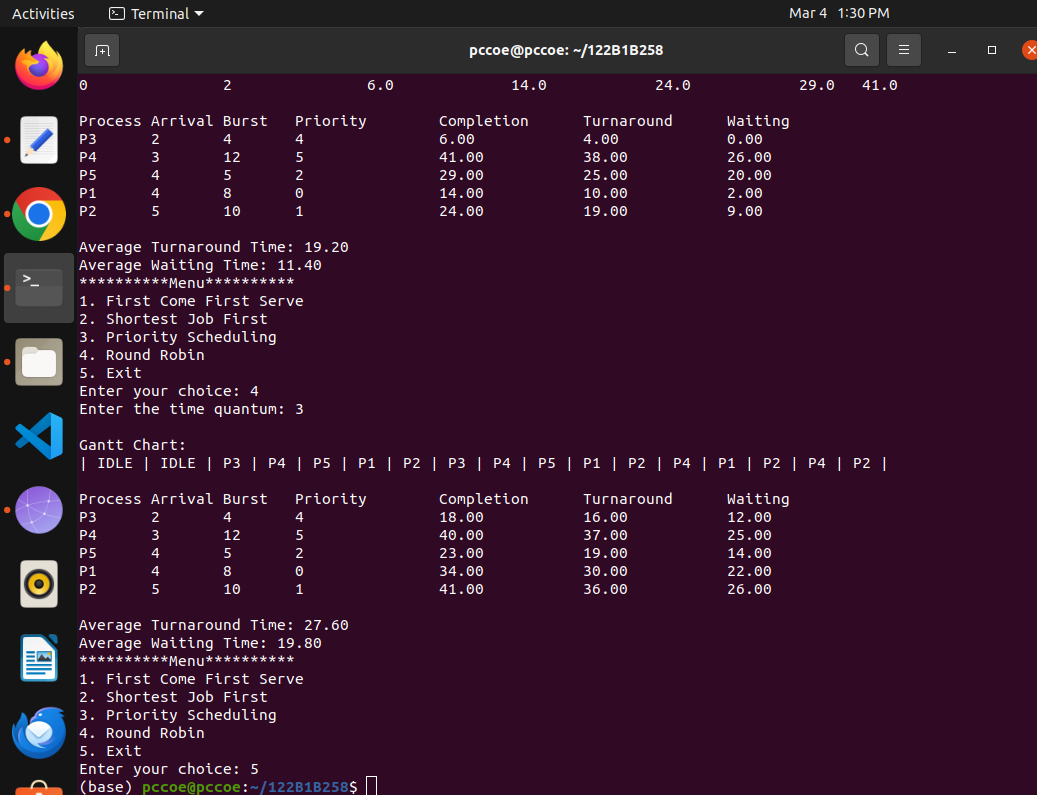
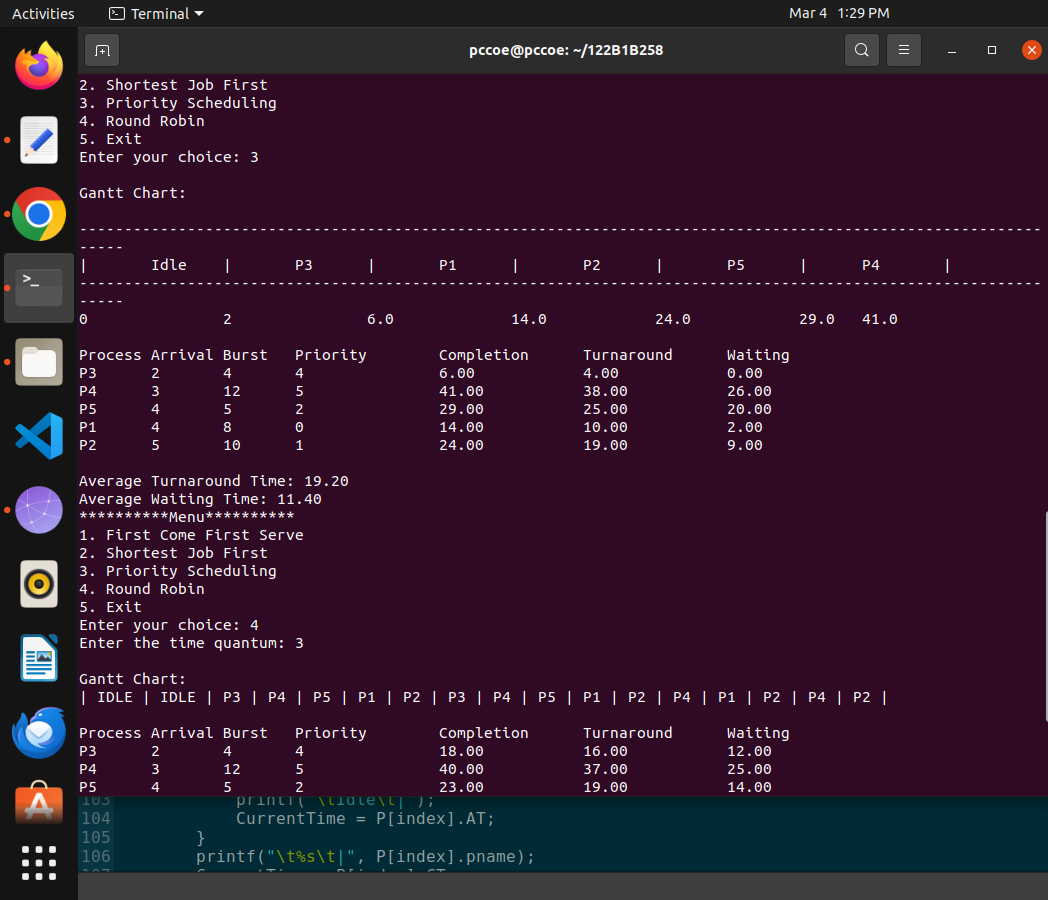
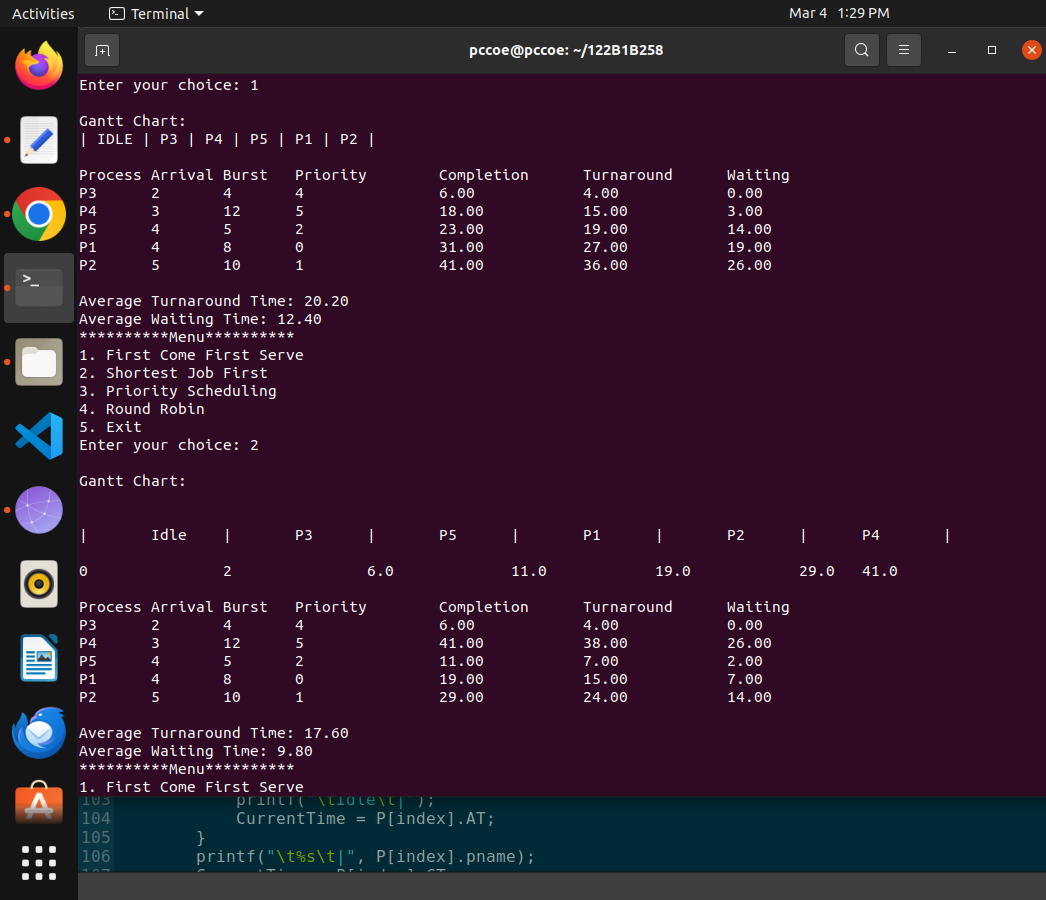
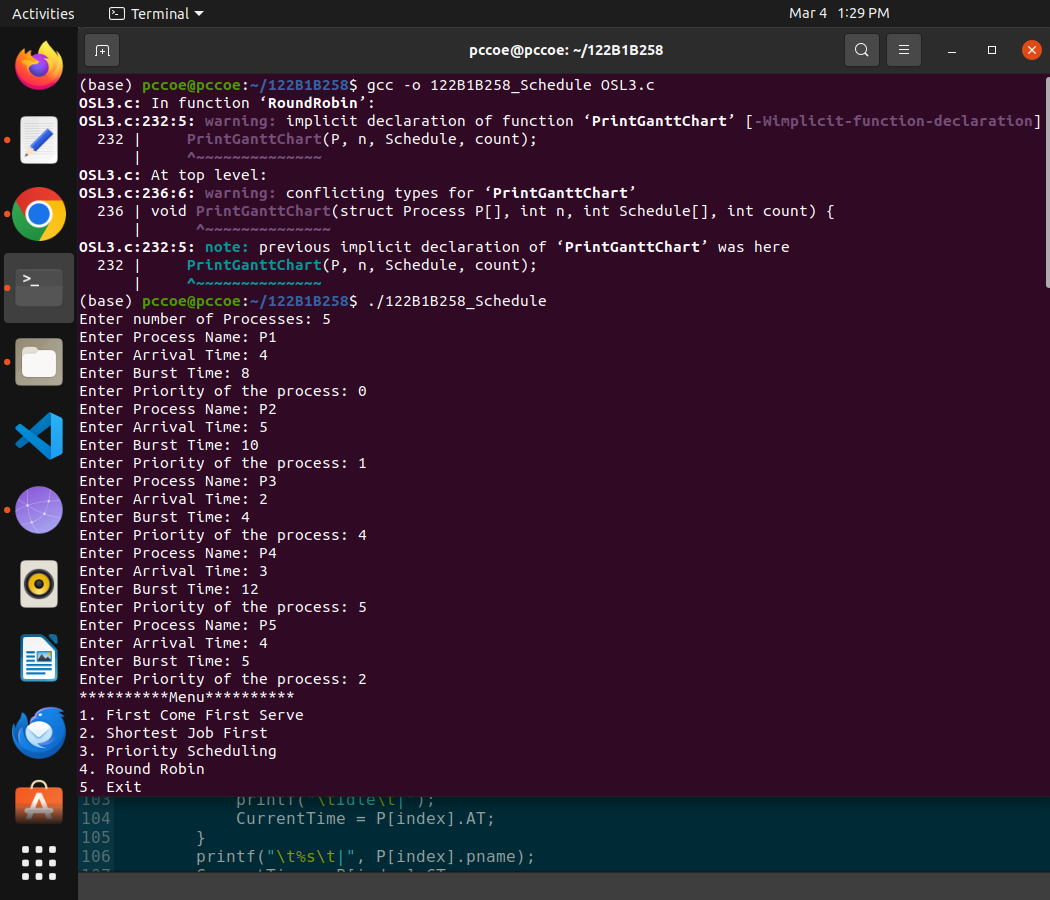
}

}while(choice != 5);

return 0;

}

**Output :**

****