**LAB-5**

**Write a menu driven program where the user gets to choose**

**Her / His choice of scheduling method.**

1. ls
2. mkdir process
3. cd process
4. nano schedule.c

P**rogram in nano window:**

#include <stdio.h>

#include <stdlib.h>

// Function prototypes

void fcfs(int processes[], int n, int bt[], int at[]);

void roundRobin(int processes[], int n, int bt[], int at[], int quantum);

void sjf(int processes[], int n, int bt[], int at[]);

void srtn(int processes[], int n, int bt[], int at[]);

int main() {

int choice;

int n;

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

scanf("%d", &n);

int processes[n];

int burstTime[n];

int arrivalTime[n];

printf("Enter burst time and arrival time for each process:\n");

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

printf("Process %d:\n", i + 1);

printf("Burst Time: ");

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

printf("Arrival Time: ");

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

processes[i] = i + 1; // Process IDs start from 1

}

do {

// Display menu

printf("\nSelect Scheduling Method:\n");

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

printf("2. Round Robin (RR)\n");

printf("3. Shortest Job First (SJF)\n");

printf("4. Shortest Remaining Time Next (SRTN)\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

fcfs(processes, n, burstTime, arrivalTime);

break;

case 2: {

int quantum;

printf("Enter time quantum for Round Robin: ");

scanf("%d", &quantum);

roundRobin(processes, n, burstTime, arrivalTime, quantum);

break;

}

case 3:

sjf(processes, n, burstTime, arrivalTime);

break;

case 4:

srtn(processes, n, burstTime, arrivalTime);

break;

case 5:

printf("Exiting program.\n");

break;

default:

printf("Invalid choice. Please enter a valid option.\n");

}

} while (choice != 5);

return 0;

}

// Function to perform First Come First Serve (FCFS) scheduling

void fcfs(int processes[], int n, int bt[], int at[]) {

int waitingTime[n], turnaroundTime[n];

// Calculate waiting time for each process

waitingTime[0] = 0;

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

int waitTime = waitingTime[i - 1] + bt[i - 1];

// If the process has arrived, consider arrival time in waiting time calculation

waitingTime[i] = (at[i] > waitTime) ? at[i] : waitTime;

}

// Calculate turnaround time for each process

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

turnaroundTime[i] = waitingTime[i] + bt[i];

}

// Display results

printf("\nFCFS Scheduling:\n");

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

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

printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i], bt[i], at[i], waitingTime[i], turnaroundTime[i]);

}

float avgWaitingTime = 0, avgTurnaroundTime = 0;

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

avgWaitingTime += waitingTime[i];

avgTurnaroundTime += turnaroundTime[i];

}

avgWaitingTime /= n;

avgTurnaroundTime /= n;

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

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

}

// Function to perform Round Robin (RR) scheduling

void roundRobin(int processes[], int n, int bt[], int at[], int quantum) {

int waitingTime[n], turnaroundTime[n];

int remainingTime[n];

// Initialize remaining time for each process

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

remainingTime[i] = bt[i];

}

int time = 0; // Current time

int flag; // Flag to check if any process is remaining

do {

flag = 0;

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

if (at[i] <= time && remainingTime[i] > 0) {

flag = 1;

if (remainingTime[i] > quantum) {

// Process executes for a time quantum

time += quantum;

remainingTime[i] -= quantum;

} else {

// Process finishes execution

time += remainingTime[i];

waitingTime[i] = time - bt[i];

remainingTime[i] = 0;

}

}

}

} while (flag);

// Calculate turnaround time for each process

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

turnaroundTime[i] = waitingTime[i] + bt[i];

}

// Display results

printf("\nRound Robin Scheduling:\n");

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

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

printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i], bt[i], at[i], waitingTime[i], turnaroundTime[i]);

}

float avgWaitingTime = 0, avgTurnaroundTime = 0;

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

avgWaitingTime += waitingTime[i];

avgTurnaroundTime += turnaroundTime[i];

}

avgWaitingTime /= n;

avgTurnaroundTime /= n;

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

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

}

// Function to perform Shortest Job First (SJF) scheduling

void sjf(int processes[], int n, int bt[], int at[]) {

int waitingTime[n], turnaroundTime[n];

int temp[n];

// Copy burst times to temp array

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

temp[i] = bt[i];

}

// Sort processes based on burst time and arrival time

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

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

if (temp[j] > temp[j + 1] || (temp[j] == temp[j + 1] && at[j] > at[j + 1])) {

// Swap processes

int tempProcess = processes[j];

processes[j] = processes[j + 1];

processes[j + 1] = tempProcess;

// Swap burst times

int tempBurstTime = temp[j];

temp[j] = temp[j + 1];

temp[j + 1] = tempBurstTime;

// Swap arrival times

int tempArrivalTime = at[j];

at[j] = at[j + 1];

at[j + 1] = tempArrivalTime;

}

}

}

// Calculate waiting time for each process

waitingTime[0] = 0;

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

int waitTime = waitingTime[i - 1] + temp[i - 1];

// If the process has arrived, consider arrival time in waiting time calculation

waitingTime[i] = (at[i] > waitTime) ? at[i] : waitTime;

}

// Calculate turnaround time for each process

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

turnaroundTime[i] = waitingTime[i] + temp[i];

}

// Display results

printf("\nShortest Job First (SJF) Scheduling:\n");

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

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

printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i], temp[i], at[i], waitingTime[i], turnaroundTime[i]);

}

float avgWaitingTime = 0, avgTurnaroundTime = 0;

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

avgWaitingTime += waitingTime[i];

avgTurnaroundTime += turnaroundTime[i];

}

avgWaitingTime /= n;

avgTurnaroundTime /= n;

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

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

}

// Function to perform Shortest Remaining Time Next (SRTN) scheduling

void srtn(int processes[], int n, int bt[], int at[]) {

int waitingTime[n], turnaroundTime[n];

int remainingTime[n];

// Initialize remaining time for each process

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

remainingTime[i] = bt[i];

}

int time = 0; // Current time

int completed = 0; // Number of completed processes

while (completed < n) {

int shortest = -1; // Index of the shortest remaining time process

// Find the process with the shortest remaining time

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

if (at[i] <= time && remainingTime[i] > 0 && (shortest == -1 || remainingTime[i] < remainingTime[shortest])) {

shortest = i;

}

}

// Update time and remaining time for the selected process

time += 1;

remainingTime[shortest] -= 1;

// Check if the process is completed

if (remainingTime[shortest] == 0) {

completed++;

waitingTime[shortest] = time - bt[shortest];

turnaroundTime[shortest] = waitingTime[shortest] + bt[shortest];

}

}

// Display results

printf("\nShortest Remaining Time Next (SRTN) Scheduling:\n");

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

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

printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i], bt[i], at[i], waitingTime[i], turnaroundTime[i]);

}

float avgWaitingTime = 0, avgTurnaroundTime = 0;

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

avgWaitingTime += waitingTime[i];

avgTurnaroundTime += turnaroundTime[i];

}

avgWaitingTime /= n;

avgTurnaroundTime /= n;

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

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

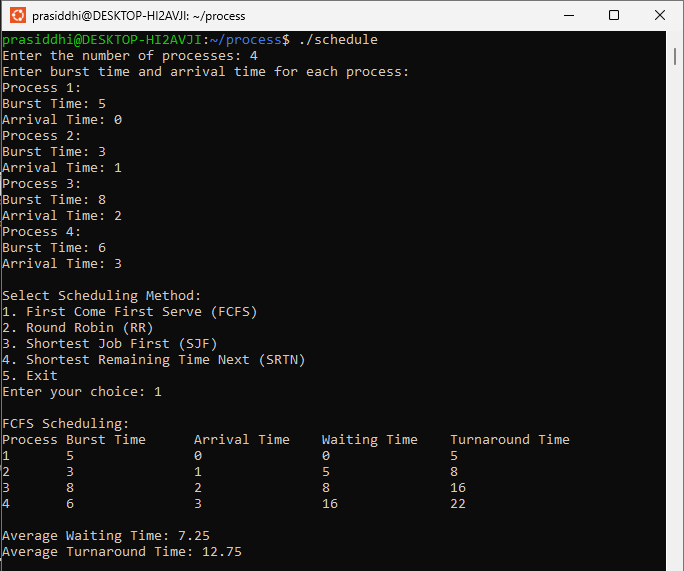
}

1. Now compile and run the code using the following commands:

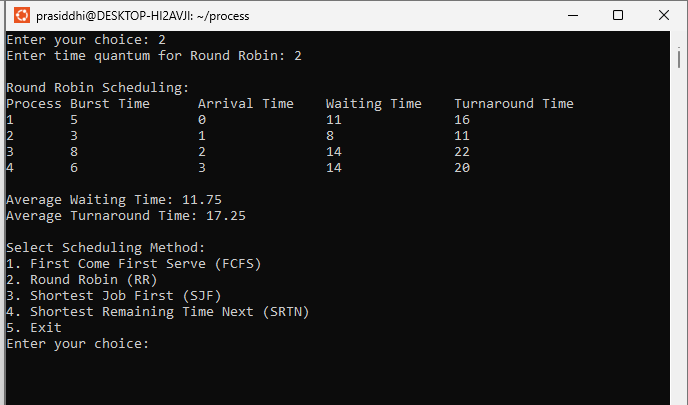
* Press ctrl+x to save file
* gcc –o schedule schedule.c –lpthread
* ./schedule

**Output:**

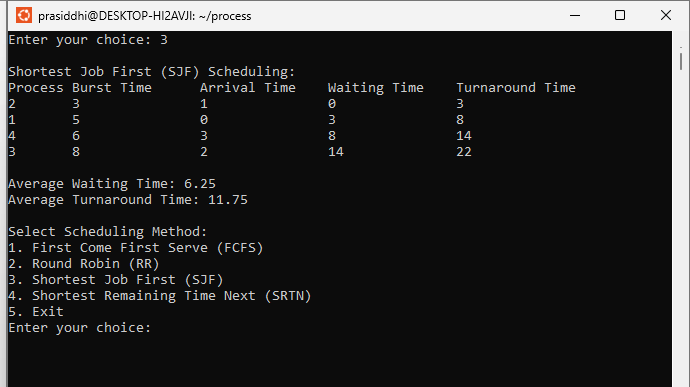
1. **For First Come First Serve**

****

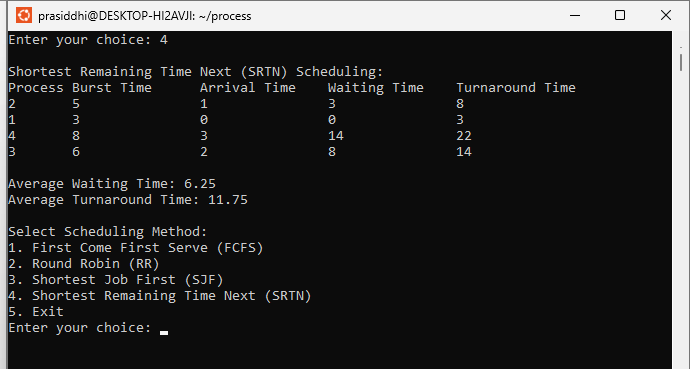
1. **For Round Robin**

****

1. **For Shortest Job First**

****

1. **For Shortest Remaining Time Next**

****