Operating System Lab Assignment: II

[4ITRC2]

Manshaa Kakani | BE: II Year | Information Technology (IT): A

Table of Contents

[ Aim: 2](#_Toc195026310)

[ To perform: 2](#_Toc195026311)

[ To Submit: 2](#_Toc195026312)

[ How to Submit: 2](#_Toc195026313)

[ C Codes and outputs 2](#_Toc195026314)

# Aim:

To create C programs for the different scheduling algorithms.

# To perform:

Create and execute C programs for following CPU Scheduling Algorithms:

1. First Come First Serve (FCFS)

2. Shortest Job First (SJF)

3. Round Robin Scheduling

# To Submit:

C Codes for the above scheduling algorithms with their outputs.

# How to Submit:

Prepare your content in word/notepad, with every page watermarked with your name and roll number. Upload on Github and submit the link in the google form attached.

# C Codes and outputs

1. First Come First Serve [FCFS]

#include <stdio.h>

struct Process {

int id;

int arrival\_time;

int burst\_time;

int completion\_time;

int turnaround\_time;

int waiting\_time;

};

int main() {

int n, i;

printf("Enter number of processes: ");

scanf("%d", &n);

struct Process processes[n];

printf("Enter Arrival and Burst time for each process:\n");

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

processes[i].id = i + 1;

printf("Process %d (Arrival Burst): ", processes[i].id);

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

}

int current\_time = 0;

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

if (current\_time < processes[i].arrival\_time) {

current\_time = processes[i].arrival\_time;

}

processes[i].completion\_time = current\_time + processes[i].burst\_time;

processes[i].turnaround\_time = processes[i].completion\_time - processes[i].arrival\_time;

processes[i].waiting\_time = processes[i].turnaround\_time - processes[i].burst\_time;

current\_time = processes[i].completion\_time;

}

float total\_tat = 0, total\_wt = 0;

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

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

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

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

processes[i].id, processes[i].arrival\_time, processes[i].burst\_time,

processes[i].completion\_time, processes[i].turnaround\_time, processes[i].waiting\_time);

total\_tat += processes[i].turnaround\_time;

total\_wt += processes[i].waiting\_time;

}

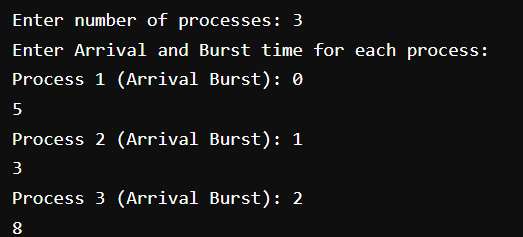
printf("Average Turnaround Time: %.2f\n", total\_tat / n);

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

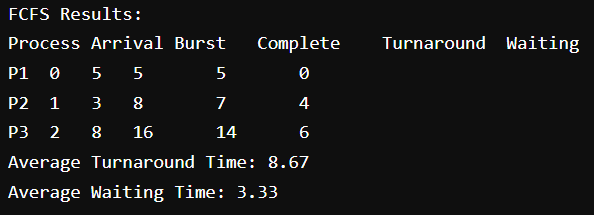
return 0;

}

Sample Input:



Output:



1. Shortest Job First [SJF]

#include <stdio.h>

#include <stdlib.h> // For qsort

struct Process {

int id;

int arrival\_time;

int burst\_time;

int completion\_time;

int turnaround\_time;

int waiting\_time;

};

int compareArrivalTime(const void \*a, const void \*b) {

return ((struct Process \*)a)->arrival\_time - ((struct Process \*)b)->arrival\_time;

}

int main() {

int n, i, current\_time = 0;

printf("Enter number of processes: ");

scanf("%d", &n);

struct Process processes[n];

printf("Enter Arrival and Burst time for each process:\n");

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

processes[i].id = i + 1;

printf("Process %d (Arrival Burst): ", processes[i].id);

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

}

qsort(processes, n, sizeof(struct Process), compareArrivalTime);

int completed = 0;

int current\_process\_index;

struct Process executed\_order[n];

int executed\_count = 0;

while (completed < n) {

current\_process\_index = -1;

int shortest\_burst = 9999;

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

int is\_completed = 0;

for (int j = 0; j < executed\_count; j++) {

if (processes[i].id == executed\_order[j].id) {

is\_completed = 1;

break;

}

}

if (processes[i].arrival\_time <= current\_time && !is\_completed) {

if (processes[i].burst\_time < shortest\_burst) {

shortest\_burst = processes[i].burst\_time;

current\_process\_index = i;

}

}

}

if (current\_process\_index == -1) {

current\_time++;

} else {

processes[current\_process\_index].completion\_time = current\_time + processes[current\_process\_index].burst\_time;

processes[current\_process\_index].turnaround\_time = processes[current\_process\_index].completion\_time - processes[current\_process\_index].arrival\_time;

processes[current\_process\_index].waiting\_time = processes[current\_process\_index].turnaround\_time - processes[current\_process\_index].burst\_time;

current\_time = processes[current\_process\_index].completion\_time;

executed\_order[executed\_count++] = processes[current\_process\_index];

completed++;

}

}

float total\_tat = 0, total\_wt = 0;

printf("\nSJF Results:\n");

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

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

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

processes[i].id, processes[i].arrival\_time, processes[i].burst\_time,

processes[i].completion\_time, processes[i].turnaround\_time, processes[i].waiting\_time);

total\_tat += processes[i].turnaround\_time;

total\_wt += processes[i].waiting\_time;

}

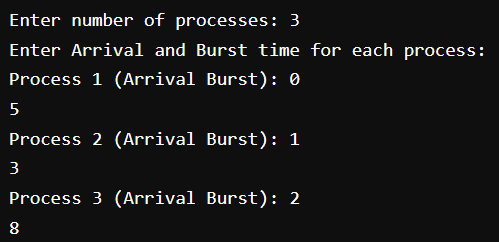
printf("Average Turnaround Time: %.2f\n", total\_tat / n);

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

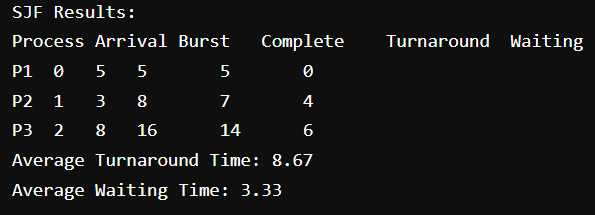
return 0;

}

Sample Input:



Output



1. Round Robin Scheduling

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

struct Process {

int id;

int arrival\_time;

int burst\_time;

int remaining\_time;

int completion\_time;

int turnaround\_time;

int waiting\_time;

bool in\_queue;

};

int main() {

int n, i, time\_quantum;

int current\_time = 0, completed = 0;

printf("Enter number of processes: ");

scanf("%d", &n);

struct Process processes[n];

printf("Enter Arrival Time and Burst Time for each process:\n");

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

processes[i].id = i + 1;

printf("Process %d (Arrival Burst): ", processes[i].id);

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

processes[i].remaining\_time = processes[i].burst\_time;

processes[i].completion\_time = 0;

processes[i].in\_queue = false;

}

printf("Enter Time Quantum: ");

scanf("%d", &time\_quantum);

int queue[100];

int front = 0, rear = 0;

queue[rear++] = 0; // Enqueue first process

processes[0].in\_queue = true;

while (completed < n) {

int index = queue[front++];

if (processes[index].remaining\_time <= time\_quantum) {

current\_time = (current\_time < processes[index].arrival\_time) ? processes[index].arrival\_time : current\_time;

current\_time += processes[index].remaining\_time;

processes[index].remaining\_time = 0;

processes[index].completion\_time = current\_time;

completed++;

} else {

current\_time = (current\_time < processes[index].arrival\_time) ? processes[index].arrival\_time : current\_time;

processes[index].remaining\_time -= time\_quantum;

current\_time += time\_quantum;

}

// Add newly arrived processes to queue

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

if (processes[i].arrival\_time <= current\_time &&

processes[i].remaining\_time > 0 &&

!processes[i].in\_queue) {

queue[rear++] = i;

processes[i].in\_queue = true;

}

}

// If current process is not finished, add it back to queue

if (processes[index].remaining\_time > 0) {

queue[rear++] = index;

}

}

float total\_tat = 0, total\_wt = 0;

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

processes[i].turnaround\_time = processes[i].completion\_time - processes[i].arrival\_time;

processes[i].waiting\_time = processes[i].turnaround\_time - processes[i].burst\_time;

total\_tat += processes[i].turnaround\_time;

total\_wt += processes[i].waiting\_time;

}

printf("\nRound Robin Results (Time Quantum: %d):\n", time\_quantum);

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

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

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

processes[i].id, processes[i].arrival\_time, processes[i].burst\_time,

processes[i].completion\_time, processes[i].turnaround\_time, processes[i].waiting\_time);

}

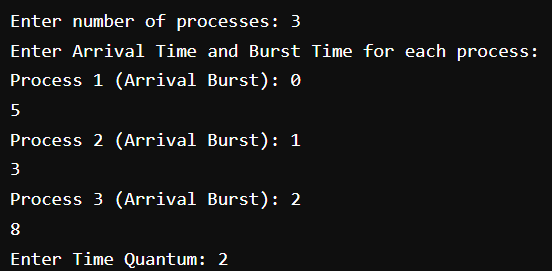
printf("\nAverage Turnaround Time: %.2f\n", total\_tat / n);

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

return 0;

}

Sample Input:



Output:

