**Name:** Kushal Kishor Shankhapal **Date:** 30/07/2023

**Roll No:** 56 **Subject:** OS Lab

**RR: Round Robin**

**Code:**

#include <stdio.h>

#define MAX\_PROCESSES 10 // Define a constant for the maximum number of processes

int main() {

int i, limit, total\_time = 0, time\_quantum;

int wait\_time = 0, turnaround\_time = 0;

int arrival\_time[MAX\_PROCESSES], burst\_time[MAX\_PROCESSES], temp\_burst\_time[MAX\_PROCESSES];

int completion\_time[MAX\_PROCESSES];

int completed[MAX\_PROCESSES] = {0}; // To keep track of completed processes

int remaining\_processes, counter;

float average\_wait\_time, average\_turnaround\_time;

// Input number of processes

printf("Enter Total Number of Processes (max %d):\n\t", MAX\_PROCESSES);

scanf("%d", &limit);

// Input arrival and burst times for each process

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

printf("Enter Details of Process[%d]\n", i + 1);

printf("Arrival Time:\t");

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

printf("Burst Time:\t");

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

temp\_burst\_time[i] = burst\_time[i]; // Copy burst times for processing

}

// Input time quantum

printf("Enter Time Quantum:\n\t");

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

printf("\nProcess ID\tAT\tBT\tCT\tTAT\tWT\n");

remaining\_processes = limit;

counter = 0;

// Main Round Robin scheduling loop

while (remaining\_processes > 0) {

int process\_found = 0;

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

if (arrival\_time[i] <= total\_time && !completed[i]) {

process\_found = 1;

if (temp\_burst\_time[i] <= time\_quantum && temp\_burst\_time[i] > 0) {

total\_time += temp\_burst\_time[i];

temp\_burst\_time[i] = 0;

completion\_time[i] = total\_time; // Record completion time

counter = 1;

} else if (temp\_burst\_time[i] > 0) {

temp\_burst\_time[i] -= time\_quantum;

total\_time += time\_quantum;

}

// Check if the process is complete

if (temp\_burst\_time[i] == 0 && counter == 1) {

remaining\_processes--;

// Calculate turnaround time and waiting time

int turnaround = completion\_time[i] - arrival\_time[i];

int wait = turnaround - burst\_time[i];

wait\_time += wait;

turnaround\_time += turnaround;

completed[i] = 1; // Mark process as completed

counter = 0;

}

}

}

// If no process was found that could be executed, advance time

if (!process\_found) {

total\_time++;

}

}

// Print process details in the original order

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

if (completed[i]) { // Only print completed processes

int turnaround = completion\_time[i] - arrival\_time[i];

int wait = turnaround - burst\_time[i];

printf("Process[%d]\t%d\t%d\t%d\t%d\t%d\n", i + 1, arrival\_time[i], burst\_time[i], completion\_time[i], turnaround, wait);

}

}

// Calculate average waiting time and turnaround time

average\_wait\_time = (float)wait\_time / limit;

average\_turnaround\_time = (float)turnaround\_time / limit;

// Print average times

printf("\nAverage Waiting Time:\t%f", average\_wait\_time);

printf("\nAverage Turnaround Time:\t%f\n", average\_turnaround\_time);

return 0;

}

**Output:**

pl-13@pl13-OptiPlex-3020:~/Kushal\_Assignments-main/OS\_Lab/RR$ gcc RR\_1.c

pl-13@pl13-OptiPlex-3020:~/Kushal\_Assignments-main/OS\_Lab/RR$ ./a.out

Enter Total Number of Processes (max 10):

4

Enter Details of Process[1]

Arrival Time: 0

Burst Time: 8

Enter Details of Process[2]

Arrival Time: 1

Burst Time: 4

Enter Details of Process[3]

Arrival Time: 2

Burst Time: 9

Enter Details of Process[4]

Arrival Time: 3

Burst Time: 5

Enter Time Quantum:

4

Process ID AT BT CT TAT WT

Process[1] 0 8 20 20 12

Process[2] 1 4 8 7 3

Process[3] 2 9 26 24 15

Process[4] 3 5 25 22 17

Average Waiting Time: 11.750000

Average Turnaround Time: 18.250000