UNIVERSITY OF INFORMATION TECHNOLOGY AND SCIENCES

Team Name : Cloud

Project Name : Cloud Drone (IOT base)

Name : Motaleb Hossain   ID : 2125051071

Name : Reduanul Karim Abid ID: 2125051058

Name: Md Ebrahim ID: 21250151062

Name: Haemaet Uddin ID: 2125051068

Batch : 50

Section : 7B1

Course title : Operating System Lab

Course Code :412

Submitted to : *Saima Siddique Tashfia*

Project Title: CPU Scheduling Simulator

Objective:

To simulate basic CPU scheduling algorithms such as **First Come First Serve (FCFS)**, **Shortest Job First (SJF)**, and **Round Robin (RR)** to compare their performance based on turnaround time and waiting time.

Features:

1. Simulate multiple CPU scheduling algorithms.
2. Accept process details (arrival time, burst time, and time quantum for Round Robin).
3. Calculate and display:
    - Waiting time for each process.
    - Turnaround time for each process.
    - Average waiting time and average turnaround time.

```
#include <stdio.h>



#define MAX_PROCESSES 100



typedef struct {

    int pid;        // Process ID

    int arrival;    // Arrival time

    int burst;      // Burst time

    int remaining;  // Remaining burst time (for RR)

    int waiting;    // Waiting time

    int turnaround; // Turnaround time

    int completed;

} Process;
```

```
// Function to sort processes by arrival time
void sort_by_arrival(Process processes[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (processes[j].arrival > processes[j + 1].arrival) {
                Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }
}

// Function for FCFS Scheduling
void fcfs(Process processes[], int n) {
    printf("\n--- FCFS Scheduling ---\n");
    sort_by_arrival(processes, n);

    int time = 0;
    float total_waiting = 0, total_turnaround = 0;

    for (int i = 0; i < n; i++) {
        if (time < processes[i].arrival) {
            time = processes[i].arrival; // CPU idle time
        }

        processes[i].waiting = time - processes[i].arrival;
        time += processes[i].burst;
        processes[i].turnaround = processes[i].waiting + processes[i].burst;

        total_waiting += processes[i].waiting;
        total_turnaround += processes[i].turnaround;

        printf("Process %d: Waiting = %d, Turnaround = %d\n",
               processes[i].pid, processes[i].waiting, processes[i].turnaround);
    }

    printf("Average Waiting Time: %.2f\n", total_waiting / n);
    printf("Average Turnaround Time: %.2f\n", total_turnaround / n);
}

// Function for Round Robin Scheduling
void round_robin(Process processes[], int n, int quantum) {
    printf("\n--- Round Robin Scheduling ---\n");
    int time = 0, completed = 0;
    float total_waiting = 0, total_turnaround = 0;
```

```
while (completed < n) {
    for (int i = 0; i < n; i++) {
        if (processes[i].remaining > 0) {
            if (processes[i].remaining > quantum) {
                time += quantum;
                processes[i].remaining -= quantum;
            } else {
                time += processes[i].remaining;
                processes[i].waiting = time - processes[i].arrival - processes[i].burst;
                processes[i].turnaround = time - processes[i].arrival;
                processes[i].remaining = 0;
                completed++;
                total_waiting += processes[i].waiting;
                total_turnaround += processes[i].turnaround;

                printf("Process %d: Waiting = %d, Turnaround = %d\n",
                       processes[i].pid, processes[i].waiting, processes[i].turnaround);
            }
        }
    }
}
```

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

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

}



// Function for Shortest Job First Scheduling

void sjf(Process processes[], int n) {

    printf("\n--- Shortest Job First (Non-Preemptive) Scheduling ---\n");

    int time = 0, completed = 0;

    float total_waiting = 0, total_turnaround = 0;



    while (completed < n) {

        int min_burst = 9999, shortest = -1;



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

            if (!processes[i].completed && processes[i].arrival <= time && processes[i].burst <
min_burst) {

                min_burst = processes[i].burst;

                shortest = i;

            }

        }



        if (shortest == -1) {

            time++; // CPU idle time

        } else {
            processes[shortest].waiting = time - processes[shortest].arrival;
            time += processes[shortest].burst;
            processes[shortest].turnaround = processes[shortest].waiting + processes[shortest].burst;
            processes[shortest].completed = 1;

            completed++;
            total_waiting += processes[shortest].waiting;
            total_turnaround += processes[shortest].turnaround;

            printf("Process %d: Waiting = %d, Turnaround = %d\n",
                   processes[shortest].pid, processes[shortest].waiting, processes[shortest].turnaround);
        }
    }

    printf("Average Waiting Time: %.2f\n", total_waiting / n);
    printf("Average Turnaround Time: %.2f\n", total_turnaround / n);
}

int main() {
    int n, quantum;
    Process processes[MAX_PROCESSES];

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

```
scanf("%d", &n);


for (int i = 0; i < n; i++) {
    processes[i].pid = i + 1;
    printf("Enter arrival time and burst time for process %d: ", i + 1);
    scanf("%d %d", &processes[i].arrival, &processes[i].burst);
    processes[i].remaining = processes[i].burst; // Initialize remaining burst time
    processes[i].completed = 0; // Initialize as not completed
}


printf("Enter time quantum for Round Robin: ");
scanf("%d", &quantum);


fcfs(processes, n);
sjf(processes, n);


// Reset remaining time and completed flag for Round Robin
for (int i = 0; i < n; i++) {
    processes[i].remaining = processes[i].burst;
    processes[i].completed = 0;
}


round_robin(processes, n, quantum);
```

```
    return 0;

}
```

Result :

```
Enter the number of processes: 3
Enter arrival time and burst time for process 1: 0 5
Enter arrival time and burst time for process 2: 1 3
Enter arrival time and burst time for process 3: 2 8
Enter time quantum for Round Robin: 4
FCFS Scheduling
Process 1: Waiting = 0, Turnaround = 5
Process 2: Waiting = 4, Turnaround = 7
Process 3: Waiting = 9, Turnaround = 17
Average Waiting Time: 4.33
Average Turnaround Time: 9.67

Shortest Job First (Non-Preemptive) Scheduling
Process 1: Waiting = 0, Turnaround = 5
Process 2: Waiting = 4, Turnaround = 7
Process 3: Waiting = 9, Turnaround = 17
Average Waiting Time: 4.33
Average Turnaround Time: 9.67

--- Round Robin Scheduling ---
Process 1: Waiting = 0, Turnaround = 5
Process 2: Waiting = 4, Turnaround = 7
Process 3: Waiting = 10, Turnaround = 18
```

Average Waiting Time: 4.67

Average Turnaround Time: 10.00

Conclusion : CPU scheduling algorithms are critical for efficient process management in operating systems, as they determine the order and timing with which processes are executed on the CPU. By simulating algorithms like First Come First Serve (FCFS) and Shortest Job First (SJF)**,** we can observe their impact on system performance metrics such as Waiting Time and Turnaround Time. Round Robin is fair and suitable for multitasking systems, but it requires careful selection of the time quantum.