

DEPARTMENT OF INFORMATION TECHNOLOGY

INSTITUTE OF ENGINEERING AND TECHNOLOGY , INDORE



LAB ASSIGNMENT OF OPERATING SYSTEM

SUBJECT CODE: 4ITRC2

LAB ASSIGNMENT - 05

NAME : GHANSHYAM BATANE

ROLLNO : 23I4128

CLASS : BE 2ND YEAR IT-B

1. First Come First Serve (FCFS) Scheduling

This scheduling algorithm processes jobs in the order they arrive.

- **Definition:** FCFS is the simplest CPU scheduling algorithm. The process that arrives first in the queue gets executed first. It operates like a queue (FIFO - First In, First Out).
- **Working:**
 - The CPU is allocated to the process that arrives first.
 - Once a process starts execution, it runs until completion (non-preemptive).
- **Advantages:**
 - Simple and easy to implement.
- **Disadvantages:**
 - Poor average waiting time when long processes arrive first.
- **Code:**

```
#include <stdio.h>
```

```
void findWaitingTime(int processes[], int n, int bt[], int wt[]) {
```

```
    wt[0] = 0; // First process has no waiting time
```

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

```
        wt[i] = bt[i - 1] + wt[i - 1];
```

```
}
```

```
void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {
```

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

```
        tat[i] = bt[i] + wt[i];
```

```
}
```

```
void findAverageTime(int processes[], int n, int bt[]) {
```

```
    int wt[n], tat[n];
```

```
    findWaitingTime(processes, n, bt, wt);
```

```
    findTurnAroundTime(processes, n, bt, wt, tat);
```

```

printf("Processes Burst Time Waiting Time Turnaround Time\n");
for (int i = 0; i < n; i++)
    printf("%d      %d      %d      %d\n", processes[i], bt[i], wt[i], tat[i]);

float total_wt = 0, total_tat = 0;
for (int i = 0; i < n; i++) {
    total_wt += wt[i];
    total_tat += tat[i];
}
printf("\nAverage waiting time = %.2f", total_wt / n);
printf("\nAverage turnaround time = %.2f\n", total_tat / n);
}

int main() {
    int processes[] = {1, 2, 3};
    int n = sizeof processes / sizeof processes[0];
    int burst_time[] = {10, 5, 8};

    findAverageTime(processes, n, burst_time);
    return 0;
}

```

- **Output :**

Active code page: 65001

D:\operating system>cd "d:\operating system\" && gcc fcfs.c -o fcfs && "d:\operating system\"fcfs

Process	Burst Time	Waiting Time	Turnaround Time
1	6	0	6
2	8	6	14
3	7	14	21
4	3	21	24

Average Waiting Time: 10.25
 Average Turnaround Time: 16.25
 d:\operating system>

2. Shortest Job First (SJF) Scheduling

SJF schedules jobs based on the shortest burst time.

- **Definition:** SJF selects the process with the smallest burst time and executes it first. It can be **preemptive** (interruptible) or **non-preemptive** (once started, it runs till completion).
- **Working:**
 - The process with the shortest execution time is selected first.
 - If two processes have the same burst time, FCFS is used.
- **Advantages:**
 - Gives the lowest average waiting time.
 - Efficient CPU utilization.
- **Disadvantages:**
 - Requires prior knowledge of burst times, which may not always be possible.
- **Code:**

```
#include <stdio.h>
```

```
void findWaitingTime(int processes[], int n, int bt[], int wt[]) {
```

```
    wt[0] = 0;
```

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

```
        wt[i] = bt[i - 1] + wt[i - 1];
```

```
}
```

```
void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {
```

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

```
        tat[i] = bt[i] + wt[i];
```

```
}
```

```

void findAverageTime(int processes[], int n, int bt[]) {
    int wt[n], tat[n];

    findWaitingTime(processes, n, bt, wt);

    findTurnAroundTime(processes, n, bt, wt, tat);

    printf("Processes Burst Time Waiting Time Turnaround Time\n");
    for (int i = 0; i < n; i++)
        printf("%d\t\t\t%d\t\t\t%d\t\t\t%d\n", processes[i], bt[i], wt[i], tat[i]);

    float total_wt = 0, total_tat = 0;
    for (int i = 0; i < n; i++) {
        total_wt += wt[i];
        total_tat += tat[i];
    }

    printf("\nAverage waiting time = %.2f", total_wt / n);
    printf("\nAverage turnaround time = %.2f\n", total_tat / n);
}

void sortProcessesByBurstTime(int processes[], int bt[], int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = i + 1; j < n; j++)
            if (bt[i] > bt[j]) {
                int temp = bt[i];
                bt[i] = bt[j];
                bt[j] = temp;

                temp = processes[i];
                processes[i] = processes[j];
                processes[j] = temp;
            }
}

```

```
int main() {  
    int processes[] = {1, 2, 3};  
    int n = sizeof processes / sizeof processes[0];  
    int burst_time[] = {6, 8, 7};  
  
    sortProcessesByBurstTime(processes, burst_time, n);  
    findAverageTime(processes, n, burst_time);  
    return 0;  
}
```

- **Output :**

```
D:\operating system>cd "d:\operating system\" && gcc sjs.c -o sjs && "d:\operating system\"sjs  
Processes  Burst Time  Waiting Time  Turnaround Time  
1           6           0             6  
3           7           6            13  
2           8          13            21  
  
Average waiting time = 6.33  
Average turnaround time = 13.33  
  
d:\operating system>
```

3. Round Robin Scheduling

This algorithm executes each job for a fixed time quantum in a cyclic order.

- **Definition:** RR scheduling assigns a fixed time quantum (time slice) to each process in a cyclic order. If a process is not finished within its time slice, it goes to the end of the queue.
- **Working:**
 - A fixed time slice (quantum) is assigned.
 - Each process gets CPU time in a circular manner.
 - If a process doesn't complete within the quantum, it is preempted and moved to the back of the queue.
- **Advantages:**
 - Ensures **fairness** as all processes get equal CPU time.
 - **Avoids starvation** because every process eventually gets executed.
- **Disadvantages:**
 - High context switching overhead if the quantum is too small.
 - If the quantum is too large, it behaves like FCFS.
- **Code :**

```
#include <stdio.h>
```

```
void findWaitingTime(int processes[], int n, int bt[], int wt[], int quantum) {
```

```
    int rem_bt[n];
```

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

```
        rem_bt[i] = bt[i];
```

```
    int t = 0;
```

```
    while (1) {
```

```

int done = 1;
for (int i = 0; i < n; i++) {
    if (rem_bt[i] > 0) {
        done = 0;
        if (rem_bt[i] > quantum) {
            t += quantum;
            rem_bt[i] -= quantum;
        } else {
            t += rem_bt[i];
            wt[i] = t - bt[i];
            rem_bt[i] = 0;
        }
    }
}
if (done)
    break;
}
}

```

```

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {
    for (int i = 0; i < n; i++)
        tat[i] = bt[i] + wt[i];
}

```

```

void findAverageTime(int processes[], int n, int bt[], int quantum) {
    int wt[n], tat[n];
    findWaitingTime(processes, n, bt, wt, quantum);
    findTurnAroundTime(processes, n, bt, wt, tat);

    printf("Processes  Burst Time  Waiting Time  Turnaround Time\n");
    for (int i = 0; i < n; i++)
        printf("%d\t\t\t\t\t %d\t\t\t\t\t %d\t\t\t\t\t %d\n", processes[i], bt[i], wt[i], tat[i]);
}

```



```

float total_wt = 0, total_tat = 0;
for (int i = 0; i < n; i++) {
    total_wt += wt[i];
    total_tat += tat[i];
}

printf("\nAverage waiting time = %.2f", total_wt / n);
printf("\nAverage turnaround time = %.2f\n", total_tat / n);
}

int main() {
    int processes[] = {1, 2, 3};
    int n = sizeof processes / sizeof processes[0];
    int burst_time[] = {24, 3, 3};
    int quantum = 4;

    findAverageTime(processes, n, burst_time, quantum);
    return 0;
}

```

- **Output :**

```

d:\operating system>cd "d:\operating system\" && gcc roundrobin.c -o roundrobin && "d:\operating system\"roundrobin
Processes Burst Time Waiting Time Turnaround Time
1         24         6         30
2          3         4          7
3          3         7         10

Average waiting time = 5.67
Average turnaround time = 15.67

```

GHANSHYAM BATANE (23/4131)