**DEPARTMENT OF INFORMATION TECHNOLOGY**

**INSTITUTE OF ENGINEERING AND TECHNOLOGY , INDORE**

****

**LAB ASSIGNMENT OF OPERATING SYSTEM**

**SUBJECT CODE: 4ITRC2**

**LAB ASSIGNMENT - 05**

**NAME : ARUSH DIXIT**

**ROLLNO : 23I4118**

**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.
* Fair as it executes processes in order of arrival.

 **Disadvantages**:

* **Convoy Effect**: A short job may have to wait for a long job to finish.
* 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];

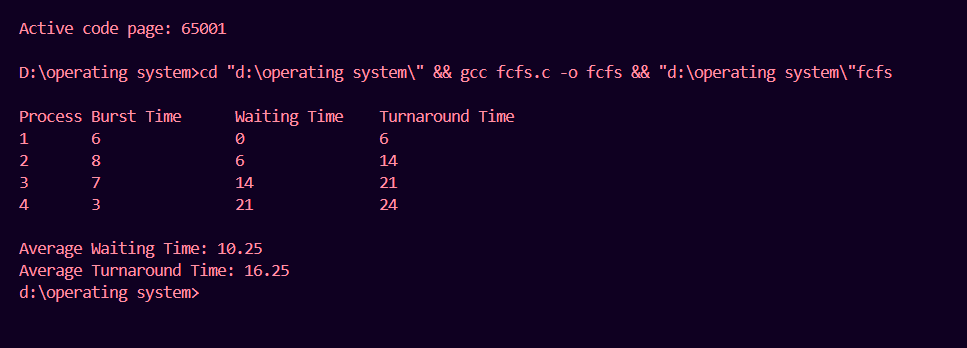
int burst\_time[] = {10, 5, 8};

findAverageTime(processes, n, burst\_time);

return 0;

}

Output



**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**:
  + **Starvation**: Long processes may never get executed if short processes keep arriving.
  + 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 %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);

}

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];

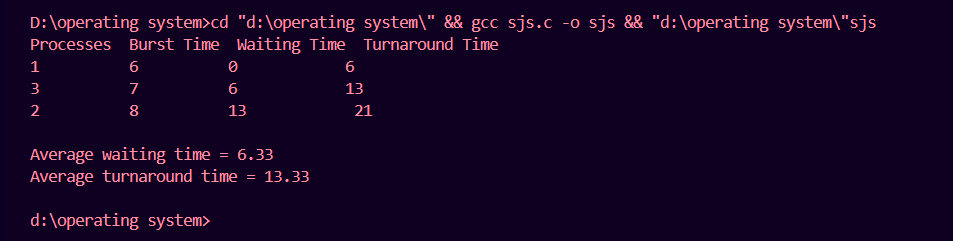
int burst\_time[] = {6, 8, 7};

sortProcessesByBurstTime(processes, burst\_time, n);

findAverageTime(processes, n, burst\_time);

return 0;

}



**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.

#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 %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[] = {24, 3, 3};

int quantum = 4;

findAverageTime(processes, n, burst\_time, quantum);

return 0;

}

Output

