**SVKM’S NMIMS**

**MUKESH PATEL SCHOOL OF TECHNOLOGY MANAGEMENT& ENGINEERING**

**(Campus Name)**

**Academic Year: 2022-2023**

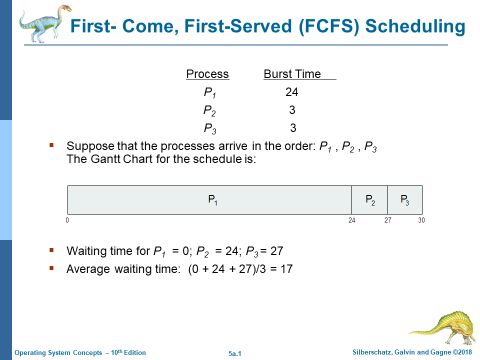
# **Practical 2-First Come First Serve Scheduling algorithm**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Roll\_No | SAP-ID | BATCH | DATE |
| Jal Bafana | K005 | 70102300054 | K1 | 10.01.2025 |

Dear all,

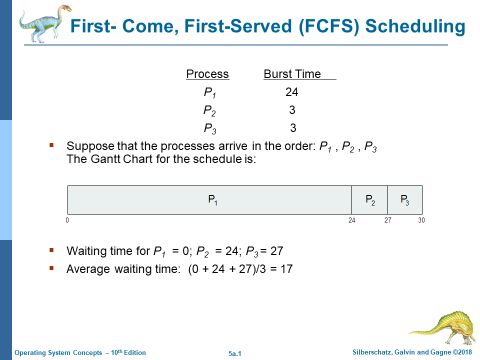
Kindly complete the following task with your name in output file.

### **Example 1:-**



Example 2

.



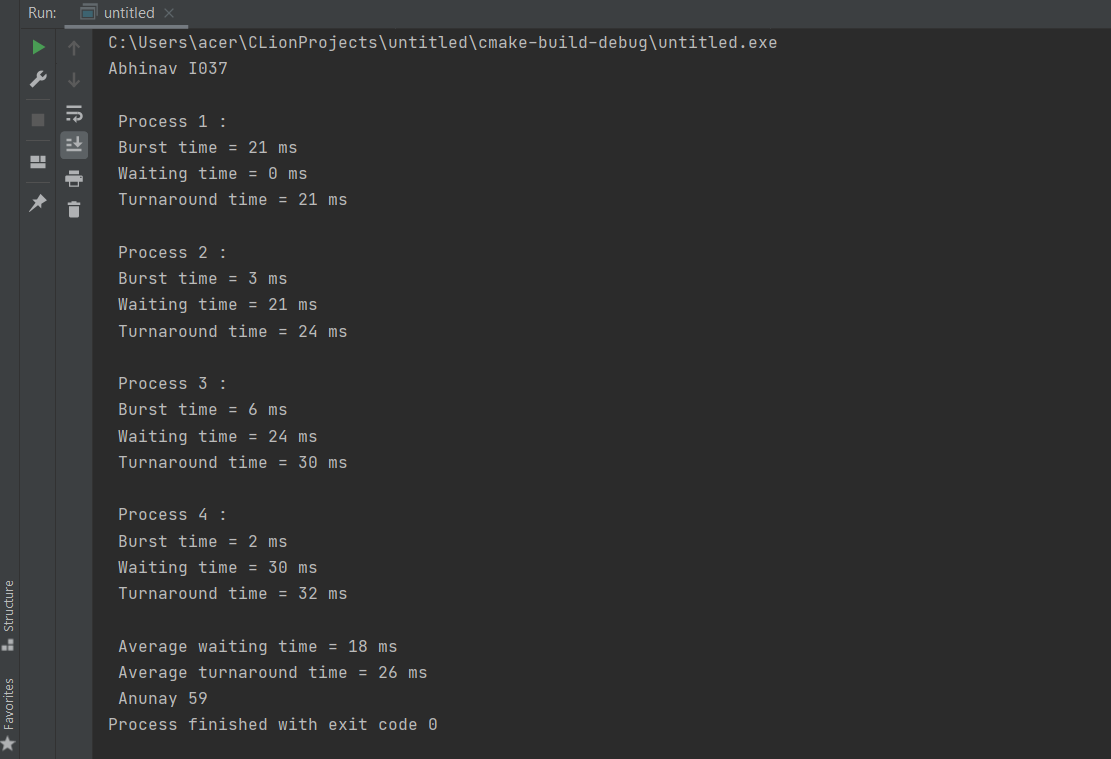
Example 3

|  |  |  |
| --- | --- | --- |
| Process | Arrival time | Burst time |
| P1 | 0 | 5 |
| P2 | 1 | 6 |
| P3 | 2 | 7 |

Theoretical calculation:

Program:

#include<stdio.h>  
  
void WaitingTime(int processes[], int n,int bt[], int wt[])  
{  
// waiting time for first process is 0  
 wt[0] = 0;  
// calculating waiting time  
 for (int i = 1; i < n ; i++ )  
 wt[i] = bt[i-1] + wt[i-1] ;  
}  
  
void TurnaroundTime(int processes[], int n,int bt[], int wt[], int ta[])  
{  
// waiting time for first process is 0  
 ta[0]=0;  
// calculating turnaround time  
 for (int i = 0; i < n ; i++ )  
 ta[i] = wt[i] + bt[i] ;  
}  
  
//Function to calculate average time  
void avgTime( int processes[], int n, int bt[])  
{  
 int wt[n], total\_wt = 0;  
 int ta[n];  
 int total\_ta=0;  
  
 WaitingTime(processes, n, bt, wt);  
 TurnaroundTime(processes,n,bt,wt,ta);  
  
 for (int i=0; i<n; i++)  
 {  
 total\_wt = total\_wt + wt[i];  
 total\_ta = total\_ta + ta[i];  
 printf("\n Process %d :",(i+1));  
 printf("\n Burst time = %d ms", bt[i] );  
 printf(" \n Waiting time = %d ms",wt[i] );  
 printf(" \n Turnaround time = %d ms \n",ta[i] );  
 }  
 int s=(float)total\_wt / (float)n;  
 int t=(float)total\_ta / (float)n;  
 printf("\n Average waiting time = %d ms",s);  
 printf("\n Average turnaround time = %d ms", t);  
 printf("\n Anunay 59");  
}  
int main()  
{  
 printf("Abhinav I037\n");  
 int processes[] = {1, 2, 3, 4};  
 int n = sizeof processes / sizeof processes[0];  
 int burst\_time[] = {21, 3, 6, 2};  
 avgTime(processes, n, burst\_time);  
  
 return 0;  
}



**Example 2:-**

# List of processes with (Process ID, Burst Time)

processes = [(1, 5), (2, 3), (3, 8), (4, 6)]

# Initialize waiting time and turnaround time lists

waiting\_time = [0] \* len(processes)

turnaround\_time = [0] \* len(processes)

# Calculate waiting time

for i in range(1, len(processes)):

waiting\_time[i] = processes[i - 1][1] + waiting\_time[i - 1]

# Calculate turnaround time

for i in range(len(processes)):

turnaround\_time[i] = processes[i][1] + waiting\_time[i]

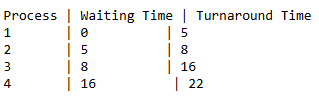
# Print the results

print("Process | Waiting Time | Turnaround Time")

for i in range(len(processes)):

print(f"{processes[i][0]} | {waiting\_time[i]} | {turnaround\_time[i]}")

Result screenshot:



## **Conclusion:-**

The First Come, First Serve (FCFS) scheduling algorithm executes processes in the order of arrival. While simple, it often leads to inefficiencies like high waiting times, especially when long processes precede shorter ones (convoy effect). Since FCFS is non-preemptive, a process runs to completion once started, delaying shorter tasks.

### Ways to Reduce Waiting Time in FCFS:

1. **Shortest Job First (SJF):** Schedule processes with shorter burst times to reduce delays.
2. **Shortest Remaining Time First (SRTF):** Use a preemptive approach where shorter processes interrupt longer ones.
3. **Assign Priorities:** Run higher-priority processes first to minimize waiting times.
4. **Multilevel Queue Scheduling:** Categorize processes and prioritize shorter ones within each group.
5. **Time-Slicing (Round Robin):** Allocate fixed time slices to prevent delays for shorter processes.

Switching to algorithms like SJF, SRTF, or Round Robin improves waiting times and overall system efficiency.

References:

1. <https://www.geeksforgeeks.org/preemptive-and-non-preemptive-scheduling/>
2. <https://www.guru99.com/fcfs-scheduling.html>
3. <https://www.javatpoint.com/os-fcfs-scheduling>