Fatima Jinnah Women University

Department Of Software Engineering

PROJECT

Course Title
Operating System (105)

Submitted To
Dr. Mehreen Sirshar

Submitted By
Mariam Fatima

Registration No: 2021-BSE-020

Date of Submission
January 04, 2022

Contents

TICKET COUNTER SCHEDULING SYSTEM

- **Ticket Counter Scheduling System** is the real-life example of **First Come First Serve Algorithm.**
- In this example, the process of **buying a movie ticket at the ticket counter** and **the allocation of seat in the cinema** are scheduled.

FIRST COME FIRST SERVE (FCFS)

- **First Come First Serve (FCFS)** is a scheduling algorithm that **executes queued requests/processes in order of their arrival.**
- In this scheduling algorithm, requests/processes are **served according to the queue manner.**

WHY USED FCFS?

- We have used FCFS and not SJF because we **do not know the time each person will take buying a ticket at the ticket counter.**

FCFS IN TICKET COUNTER SCHEDULING SYSTEM

- In this scheduling algorithm, the person who arrives first in the queue **first buys theticket** and **first gets the seat** and then the next one. This will continue until the last person in the queue purchases the ticket and gets the seat.

ADVANTAGES OF FCFS

- Simple
- User-Friendly
- Easy to understand
- Easy to implement

DISADVANTAGES OF FCFS

- Non-preemptive nature (processes wait until the current program completes).
- Convoy Effect (Short processes behind long processes).
- Long waiting time.
- Incompatible with time-sharing systems.

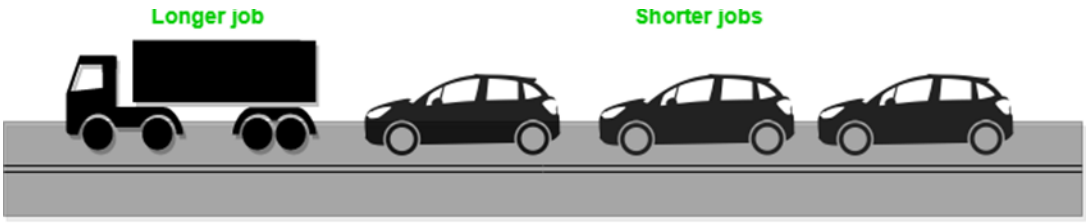
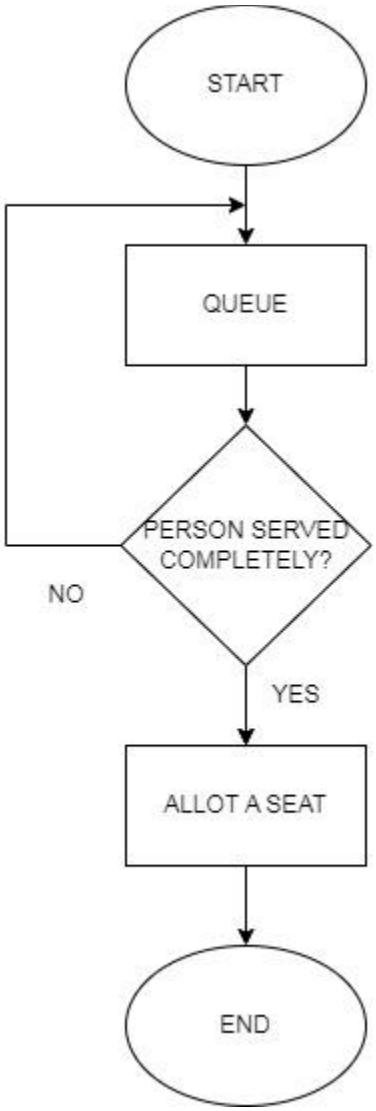


Figure - The Convey Effect, Visualized

FLOW CHART



ALGORITHM

- Start
- First person standing in the queue will get ticket first.
- If first person served completely
- Seat in the cinema allotted.
- Else in case of any issue (person having not enough money to buy a ticket and wants to go to ATM first)
- Person will again have to stand in the queue.
- End

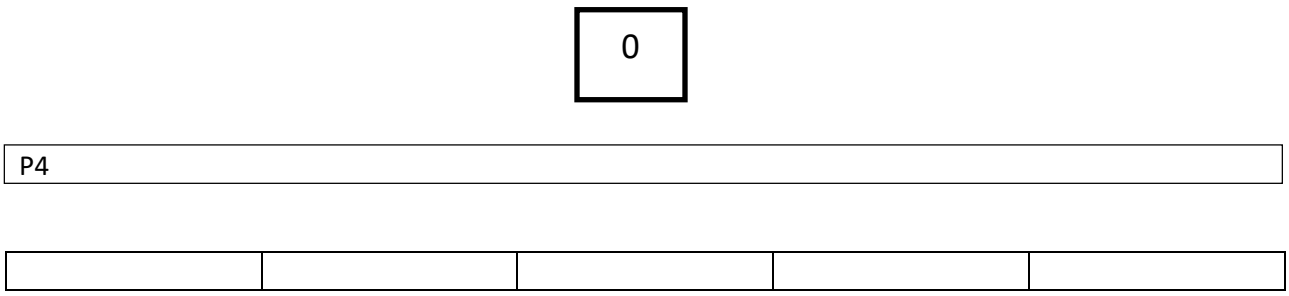
GANTT CHART

- Five processes (people) arriving at different times at the ticket counter. Each process (person) has a different burst time.

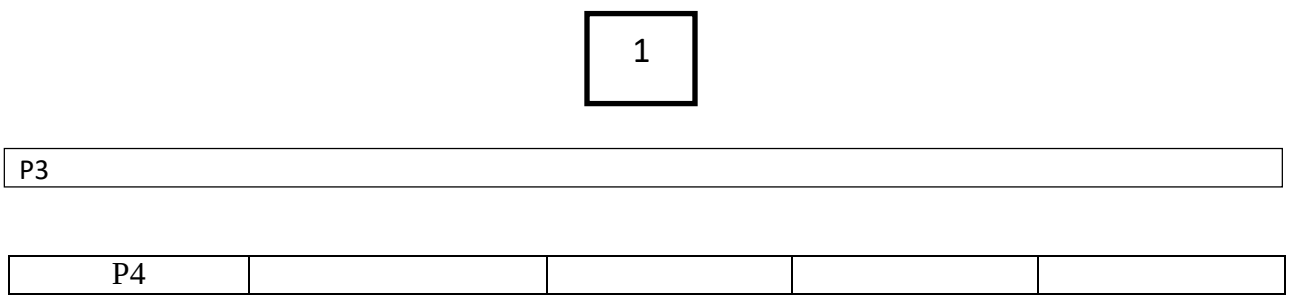
Processes	Arrival Time	Burst Time	Completion Time
P1	2	6	17
P2	5	2	23
P3	1	8	11
P4	0	3	3
P5	4	4	21

- Using the FCFS scheduling algorithm, these processes (people) are handled as follows.

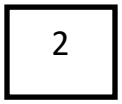
1. The process begins with P4 which has arrival time 0.



2. At time=1, P3 arrives. P4 is still executing. Hence, P3 is kept in a queue.



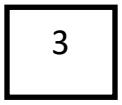
3. At time= 2, P1 arrives, which is kept in the queue.



P3	P1

P4				

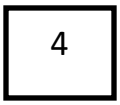
4. At time=3, P4 process completes its execution.



P3	P1

P4				

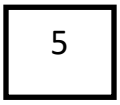
5. At time=4, P3, which is first in the queue, starts execution.



P1	P5

P4	P3			

6. At time =5, P2 arrives, which is kept in a queue.



P1	P5	P2

P4	P3			

7. At time 11, P3 completes its execution.

11

P1	P5	P2

P4	P3			

8. At time=11, P1 starts execution. It has a burst time of 6. It completes execution at time interval 17.

17

P5	P2

P4	P3	P1		

9. At time=17, P5 starts execution. It has a burst time of 4. It completes execution at time=21.

21

P2

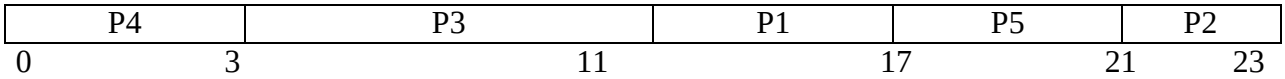
P4	P3	P1	P5	

10. At time=21, P2 starts execution. It has a burst time of 2. It completes execution at time interval 23.

23

P4	P3	P1	P5	P2

FINAL GANTT CHART



WAITING TIME

(Start Time – Arrival Time)

Processes (People)	Waiting Time
P1	11 – 2 = 9
P2	21 – 5 = 16
P3	3 – 1 = 2
P4	0 – 0 = 0
P5	17 – 4 = 13

AVERAGE WAITING TIME

(Total Waiting Time / No. of Waiting Time)

= 0 + 2 + 9 + 13 + 16 / 5

= 40 / 5

= 8

TURN AROUND TIME

(Completion Time - Arrival Time)

Processes (People)	Turn Around Time
P1	17 – 2 = 15
P2	23 – 5 = 18
P3	11 – 1 = 10
P4	3 – 0 = 3
P5	21 – 4 = 17

AVERAGE TURN AROUND TIME

(Total Turn Around Time / No. of Turn Around Time)

= 15 + 18 + 10 + 3 + 17 / 5

= 63 / 5

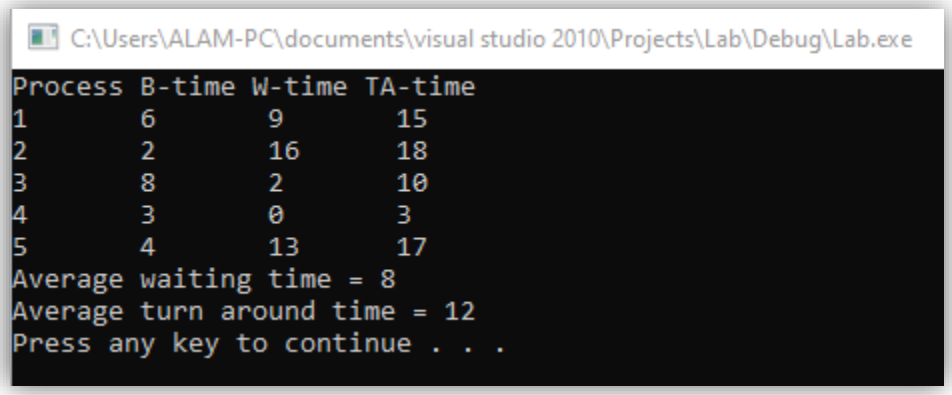
=12.5

CALCULATION OF AVERAGE WAITING AND TURN AROUND TIME

```
#include"stdafx.h"
#include<iostream>
using namespace std;
void Waiting_Time(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 TurnAround_Time( int processes[], int n, int bt[], int wt[], int tat[])
{
for (int i = 0; i < n ; i++)
tat[i] = bt[i] + wt[i];
}
void Average_Time( int processes[], int n, int bt[])
{
int wt[n], tat[n], total_wt = 0, total_tat = 0;
Waiting_Time(processes, n, bt, wt);
TurnAround_Time(processes, n, bt, wt, tat);
cout << "P"<< " B-time " << " W-time " << " TA-time" <<endl;
for (int i=0; i<n; i++)
{
total_wt = total_wt + wt[i];
```

```
total_tat = total_tat + tat

cout << " " << i+1 << "\t" << bt[i] <<"\t " << wt[i] <<"\t\t " << tat[i] <<endl;

}

cout << "Average waiting time = " << (float)total_wt / (float)n <<endl;

cout << "Average turn around time = " << (float)total_tat / (float)n <<endl;

}

int main()

{

int processes[] = { 1, 2, 3, 4 ,5};

int n = sizeof processes / sizeof processes[0];

int burst_time[] = {6, 2, 8, 3, 4};

Average_Time(processes, n, burst_time);

system("pause");

return 0;

}
```

Output:



REFERENCES

- www.geeksforgeeks.org
- www.ecomputernotes.com