Course Name:	Operating Systems and Compilers	Semester:	VI
Date of Performance:	07 / 02 / 2025	Batch No.:	B - 2
Faculty Name:	Prof. Nilesh Lakade	Roll No.:	16014022050
Faculty Sign & Date:		Grade/Marks:	___ / 25

Experiment No.: 3

Title: Implementation of Basic Process management algorithms – non-pre-emptive (FCFS, SJF)

Aim and Objective of the Experiment:

To implement basic Non –Pre-emptive Process management algorithms (FCFS, SJF).

COs to be achieved:

CO2: Describe the problems related to process concurrency and the different synchronization mechanisms available to solve them.

Theory:

Most systems have a large number of processes with short CPU bursts interspersed between I/O requests and a small number of processes with long CPU bursts. To provide good time-sharing performance, we may preempt a running process to let another one run. The ready list, also known as a run queue, in the operating system keeps a list of all processes that are ready to run and not blocked on some I/O or other system request, such as a semaphore. Then entries in this list are pointers to the process control block, which stores all information and state about a process.

When an I/O request for a process is complete, the process moves from the *waiting* state to the *ready* state and gets placed on the run queue.

The process scheduler is the component of the operating system that is responsible for deciding whether the currently running process should continue running and, if not, which process should run next. There are four events that may occur where the scheduler needs to step in and make this decision:

1. The current process goes from the *running* to the *waiting* state because it issues an I/O request or some operating system request that cannot be satisfied immediately.
2. The current process terminates.
3. A timer interrupt causes the scheduler to run and decide that a process has run for its allotted interval of time and it is time to move it from the *running* to the *ready* state.
4. An I/O operation is complete for a process that requested it and the process now moves from the *waiting* to the*ready* state. The scheduler may then decide to pre-empt the currently-running process and move this *ready* process into the *running* state.

The decisions that the scheduler makes concerning the sequence and length of time that processes may run is called the scheduling algorithm (or scheduling policy). These decisions are not easy ones,

as the scheduler has only a limited amount of information about the processes that are ready to run.

A good scheduling algorithm should:

1. Be fair – give each process a fair share of the CPU, allow each process to run in a reasonable amount of time.
2. Be efficient – keep the CPU busy all the time.
3. Maximize throughput – service the largest possible number of jobs in a given amount of time; minimize the amount of time users must wait for their results.
4. Minimize response time – interactive users should see good performance
5. Minimize overhead – don't waste too many resources. Keep scheduling time and context switch time at a minimum.
6. Maximize resource use – favor processes that will use underutilized resources. There are two motives for this. Most devices are slow compared to CPU operations. We'll achieve better system throughput by keeping devices busy as often as possible. The second reason is that a process may be holding a key resource and other, possibly more important, processes cannot use it until it is released. Giving the process more CPU time may free up the resource quicker.
7. Avoid indefinite postponement – every process should get a chance to run eventually.

Implementation details:

First Come, First Serve Scheduling:

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

int main() {
    int bt[20], at[10], n, i, st[10], ft[10], wt[10], ta[10], rt[10];
    int totwt = 0, totta = 0;
    double awt, ata;
    char pn[10][10];

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    printf("\nEnter process name, arrival time (AT) & burst time (BT):\n");
    for (i = 0; i < n; i++) {
        printf("For process %d: ", i + 1);
        scanf("%s %d %d", pn[i], &at[i], &bt[i]);
    }

    for (i = 0; i < n; i++) {
        if (i == 0 || at[i] > ft[i - 1])
            st[i] = at[i];
        else
            st[i] = ft[i - 1];
        rt[i] = st[i] - at[i];
```

```
        ft[i] = st[i] + bt[i];
        ta[i] = ft[i] - at[i];
        wt[i] = ta[i] - bt[i];
        totwt += wt[i];
        totta += ta[i];
    }

    awt = (double)totwt / n;
    ata = (double)totta / n;

    printf("\n%-12s %-8s %-8s %-8s %-8s %-8s %-8s\n", "ProcessName", "AT", "BT",
"CT", "TAT", "WT", "RT");
    printf("-------------------------------------------------------\n");
    for (i = 0; i < n; i++) {
        printf("%-12s %-8d %-8d %-8d %-8d %-8d %-8d\n", pn[i], at[i], bt[i], ft[i],
ta[i], wt[i], rt[i]);
    }

    printf("\nAverage Waiting Time: %.2f", awt);
    printf("\nAverage Turnaround Time: %.2f", ata);

    return 0;
}
```

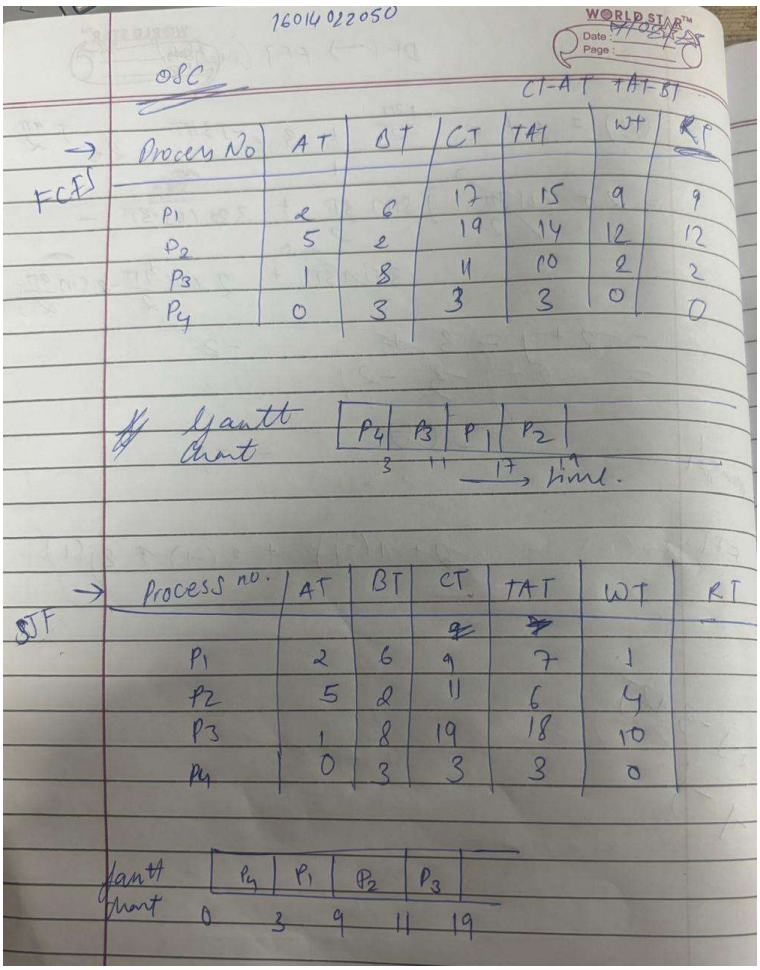
```
PS C:\Users\admin\OneDrive\Desktop\sem 6\operating system and compilers> cd "c:\Users\admin\One
Drive\Desktop\sem 6\operating system and compilers\" ; if ($?) { gcc fcfs.c -o fcfs } ; if ($?)
 { .\fcfs }
Enter the number of processes: 4

Enter process name, arrival time (AT) & burst time (BT):
For process 1: p1 2 6
For process 2: p2 5 2
For process 3: p3 1 8
For process 4: p4 0 3

ProcessName  AT       BT       CT       TAT      WT       RT
-------------------------------------------------------
p4           0        3        3        3        0        0
p3           1        8        11       10       2        2
p1           2        6        17       15       9        9
p2           5        2        19       14       12       12

Average Waiting Time: 5.75
Average Turnaround Time: 10.50
PS C:\Users\admin\OneDrive\Desktop\sem 6\operating system and compilers>
```

Shortest Job First Scheduling:

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

int main() {
    int bt[20], at[20], n, i, j, st[20], ft[20], wt[20], ta[20], rt[20];
    int totwt = 0, totta = 0, completed = 0, current_time = 0;
    double awt, ata;
    char pn[20][10];
    int is_completed[20] = {0};

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    printf("\nEnter process name, arrival time (AT) & burst time (BT):\n");
    for (i = 0; i < n; i++) {
        printf("For process %d: ", i + 1);
        scanf("%s %d %d", pn[i], &at[i], &bt[i]);
    }

    while (completed != n) {
        int shortest_job = -1;
        int min_burst = __INT_MAX__;

        for (i = 0; i < n; i++) {
            if (at[i] <= current_time && !is_completed[i]) {
                if (bt[i] < min_burst) {
                    min_burst = bt[i];
                    shortest_job = i;
                }
            }
        }

        if (shortest_job == -1) {
            current_time++;
            continue;
        }

        st[shortest_job] = current_time;
        ft[shortest_job] = st[shortest_job] + bt[shortest_job];
        rt[shortest_job] = st[shortest_job] - at[shortest_job];
        ta[shortest_job] = ft[shortest_job] - at[shortest_job];
        wt[shortest_job] = ta[shortest_job] - bt[shortest_job];
```

```
        totwt += wt[shortest_job];
        totta += ta[shortest_job];

        is_completed[shortest_job] = 1;
        completed++;
        current_time = ft[shortest_job];
    }

    awt = (double)totwt / n;
    ata = (double)totta / n;

    printf("\n%-12s %-8s %-8s %-8s %-8s %-8s %-8s\n", "ProcessName", "AT", "BT",
"CT", "TAT", "WT", "RT");
    printf("-------------------------------------------------------\n");
    for (i = 0; i < n; i++) {
        printf("%-12s %-8d %-8d %-8d %-8d %-8d %-8d\n", pn[i], at[i], bt[i], ft[i],
ta[i], wt[i], rt[i]);
    }

    printf("\nAverage Waiting Time: %.2f", awt);
    printf("\nAverage Turnaround Time: %.2f", ata);

    return 0;
}
```

```
PS C:\Users\admin\OneDrive\Desktop\sem 6\operating system and compilers> cd "c:\Users\admin\One
Drive\Desktop\sem 6\operating system and compilers\" ; if ($?) { gcc sfj.c -o sfj } ; if ($?) {
 .\sfj }
Enter the number of processes: 4

Enter process name, arrival time (AT) & burst time (BT):
For process 1: p1 2 6
For process 2: p2 5 2
For process 3: p3 1 8
For process 4: p4 0 3

ProcessName  AT       BT       CT       TAT      WT       RT
-------------------------------------------------------
p1           2        6        9        7        1        1
p2           5        2        11       6        4        4
p3           1        8        19       18       10       10
p4           0        3        3        3        0        0

Average Waiting Time: 3.75
Average Turnaround Time: 8.50
PS C:\Users\admin\OneDrive\Desktop\sem 6\operating system and compilers>
```



16014022050

OSC

FCFS

Process No	AT	BT	CT	TAT	WT	RT
P1	2	6	17	15	9	9
P2	5	2	19	14	12	12
P3	1	8	11	10	2	2
P4	0	3	3	3	0	0

Gantt chart: P4 | P3 | P1 | P2 (3, 11, 17, 19 time.)

SJF

Process no.	AT	BT	CT	TAT	WT	RT
P1	2	6	9	7	1	
P2	5	2	11	6	4	
P3	1	8	19	18	10	
P4	0	3	3	3	0	

Gantt chart: P4 | P1 | P2 | P3 (0, 3, 9, 11, 19)

Post Lab Subjective/Objective type Questions:

1. What is the ready state of a process?

a) when process is scheduled to run after some execution
b) when process is unable to run until some task has been completed
c) when process is using the CPU
d) none of the mentioned

Ans: A

2. A process stack does not contain

a) function parameters
b) local variables
c) return addresses
d) PID of child process

Ans: D

3. A process can be terminated due to

a) normal exit
b) fatal error
c) killed by another process
d) all of the mentioned

Ans: D

Conclusion:

We implemented First-Come-First-Serve (FCFS) and Shortest Job First (SJF) scheduling algorithms, where FCFS executes processes in arrival order regardless of burst time, while SJF prioritizes processes with shorter burst times to minimize average waiting time and improve system efficiency.

Signature of faculty in-charge with Date: