Batch: A2	**Roll No.: 16010121045**
Experiment No. 03	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

TITLE: Implementation of Basic Process management algorithms – Non Pre-emptive (FCFS , SJF, priority)

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

Expected Outcome of Experiment:

CO 2. To understand the concept of process, thread and resource management.

Books/ Journals/ Websites referred:

1. **Silberschatz A., Galvin P., Gagne G. "Operating Systems Principles", Willey Eight edition.**
2. **Achyut S. Godbole , Atul Kahate "Operating Systems" McGraw Hill Third Edition.**
3. **William Stallings, "Operating System Internal & Design Principles", Pearson.**
4. **Andrew S. Tanenbaum, "Modern Operating System", Prentice Hall.**

Pre Lab/ Prior Concepts:

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 preempt 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.

Description of the application to be implemented:

First-Come, First-Served Scheduling:

```
public class fcfs {
    static void CalculateWaitingTime(int at[], int bt[], int
N) {
        int[] wt = new int[N];
        wt[0] = 0;
        System.out.print("P.No.\tArrival Time\t" + "Burst
Time\tWaiting Time\n");
        System.out.print("1" + "\t\t" + at[0] + "\t\t" +
bt[0] + "\t\t" + wt[0] + "\n");
        for (int i = 1; i < 5; i++) {
            wt[i] = (at[i - 1] + bt[i - 1] + wt[i - 1]) -
at[i];
            System.out.print(i + 1 + "\t\t" + at[i] + "\t\t"
+ bt[i] + "\t\t" + wt[i] + "\n");
        }
        float average;
        float sum = 0;
        for (int i = 0; i < 5; i++)
            sum = sum + wt[i];
        average = sum / 5;
        System.out.println("Average waiting time = " +
average);
    }

    public static void main(String args[]) {
        int N = 5;
        int at[] = { 0, 1, 2, 3, 4 };
        int bt[] = { 4, 3, 1, 2, 5 };
        CalculateWaitingTime(at, bt, N);
    }
}
```

Output

```
cd "/Users/pargatsinghdhanjal/Desktop/Coding/OS/" && javac fcfs.java && java fcfs
❯ cd "/Users/pargatsinghdhanjal/Desktop/Coding/OS/" && javac fcfs.java && java fcfs
P.No.    Arrival Time    Burst Time      Waiting Time
1               0               4               0
2               1               3               3
3               2               1               5
4               3               2               5
5               4               5               6
Average waiting time = 3.8
```

Shortest job first :

```
import java.util.*;

public class sjf {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.println("enter no of process:");
        int n = sc.nextInt();
        int pid[] = new int[n];
        int at[] = new int[n];
        int bt[] = new int[n];
        int ct[] = new int[n];
        int ta[] = new int[n];
        int wt[] = new int[n];
        int f[] = new int[n];
        int st = 0, tot = 0;
        float avgwt = 0, avgta = 0;
        for (int i = 0; i < n; i++) {
            System.out.println("enter process " + (i + 1) + "
arrival time:");
            at[i] = sc.nextInt();
            System.out.println("enter process " + (i + 1) + "
brust time:");
            bt[i] = sc.nextInt();
            pid[i] = i + 1;
            f[i] = 0;
        }
        boolean a = true;
        while (true) {
            int c = n, min = 999;
```

```
            if (tot == n)
                break;
            for (int i = 0; i < n; i++) {
                if ((at[i] <= st) && (f[i] == 0) && (bt[i] <
min)) {
                    min = bt[i];
                    c = i;
                }
            }
            if (c == n)
                st++;
            else {
                ct[c] = st + bt[c];
                st += bt[c];
                ta[c] = ct[c] - at[c];
                wt[c] = ta[c] - bt[c];
                f[c] = 1;
                tot++;
            }
        }
        System.out.println("\npid arrival brust complete turn
waiting");
        for (int i = 0; i < n; i++) {
            avgwt += wt[i];
            avgta += ta[i];
            System.out.println(pid[i] + "\t" + at[i] + "\t" +
bt[i] + "\t" + ct[i] + "\t" + ta[i] + "\t" + wt[i]);
        }
        System.out.println("\naverage tat is " + (float)
(avgta / n));
        System.out.println("average wt is " + (float) (avgwt
/ n));
        sc.close();
    }
}
```

```
❯ cd "/Users/pargatsinghdhanjal/Desktop/Coding/OS/"
va sjf
enter no of process:
3
enter process 1 arrival time:
0
enter process 1 brust time:
3
enter process 2 arrival time:
0
enter process 2 brust time:
1
enter process 3 arrival time:
1
enter process 3 brust time:
2

pid arrival brust complete turn waiting
1       0       3       6       6       3
2       0       1       1       1       0
3       1       2       3       2       0

average tat is 3.0
average wt is 1.0
```

Conclusion: Through this experiment we understood the concept of non-pre-emptive scheduling algorithm and implemented First Come First Serve and Shortest Job First algorithm in C++ language.

Post Lab Objective 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

2. A process stack does not contain
a) function parameters
b) local variables
c) return addresses
d) PID of child process

3. A process can be terminated due to
a) normal exit
b) fatal error
c) killed by another process
d) all of the mentioned

Date: _____ **Signature of faculty in-charge**