॥ वसुधैव कुटुम्बकम् ॥

**SYMBIOSIS INTERNATIONAL (DEEMED UNIVERSITY)**

**(Established under section 3 of the UGC Act 1956)**

**Re-accredited by NAAC with "A" Grade**

**Name of the Student: Udayrajsinh Jhala** **PRN: 22070122240**

**Semester:** IV **Year:** AY 23-24

| | |
|---|---|
| **Subject Title: Operating Systems Lab** | |
| **EXPERIMENT No**: 7 | |
| **TITLE**: FCFS Scheduling Algorithm | **DoP**: 1-03-2023 |

**Aim:** Implement a C++ program to demonstrate the FCFS algorithm.

**Learning Outcome:** 1. To understand the scheduling algorithm
2. To Demonstrate the FCFS algorithm

**Hardware/Software:**

- Hardware requirements:
  - A PC with a minimum of 4GB RAM.
  - A processor with 1.6GHz clock speed or faster.
  - Sufficient hard disk space to store the code and compiler.
- Software Requirements:
  - Visual Studio Code (VS Code)
  - C++ compiler such as GCC
  - C/C++ extension for VS Code. This extension is used to compile and run C++ code directly from VS Code.

**Theory:**

Scheduling algorithms are used in operating systems to allocate system resources to processes in an efficient manner. Scheduling algorithms determine which process gets access to the CPU and when, based on predefined criteria such as priority, execution time, and resource requirements. One of the most basic scheduling algorithms is First-Come, First-Served (FCFS).

FCFS is a non-preemptive scheduling algorithm where the processes are executed in the order they arrive. The process that arrives first is executed first and so on. FCFS is easy to understand and implement, but it has several drawbacks. One of the major drawbacks of FCFS is that it can lead to long waiting times for processes with large execution times. This is because, in FCFS, the processes are executed in the order they arrive, without taking into consideration their execution time or priority. This can lead to a situation where a process with a long execution time occupies

the CPU, while other shorter processes have to wait, leading to increased waiting times and reduced system throughput

The features of the FCFS algorithm include:

- Simple and easy to understand.
- Non-preemptive means a process will keep using the CPU until it completes or voluntarily gives up the CPU.
- Follows the concept of First-Come, First-Served, which means the process that arrives first is executed first.

The drawbacks of the FCFS algorithm include:

- Long waiting times for processes with large execution times.
- Poor performance in a multi-programming environment, as it can lead to poor utilization of the CPU.
- Does not take into consideration the priority or execution time of the processes, which can lead to reduced system throughput.

**Algorithm:**

1. Initialize variables n, burst_time, arrival_time, waiting_time, turnaround_time, completion_time, avg_waiting_time, and avg_turnaround_time.
2. Read the number of processes n from the user.
3. Read the burst time for each process and store it in the burst_time array.
4. Check if all processes have arrived at the same instant or not by comparing the burst time of each process.
5. If all processes have arrived at the same instant, set the arrival time to 0 for all processes.
6. Else, read the arrival time for each process and store it in the arrival_time array.
7. Calculate the completion time for each process using the formula: completion_time[0] = burst_time[0] + arrival_time[0]; completion_time[i] = max(completion_time[i-1], arrival_time[i]) + burst_time[i];
8. Calculate the waiting time for each process using the formula: waiting_time[0] = 0; waiting_time[i] = max(0, completion_time[i-1] - arrival_time[i]);
9. Calculate the turnaround time for each process using the formula: turnaround_time[i] = waiting_time[i] + burst_time[i];
10. Calculate the average waiting time and average turnaround time using the formula: avg_waiting_time = sum(waiting_time) / n; avg_turnaround_time = sum(turnaround_time) / n;
11. Display the process details using the following format: Process Burst Time Arrival Time Waiting Time Turnaround Time Completion Time p1 bt1 at1 wt1 tat1 ct1 p2 bt2 at2 wt2 tat2 ct2 ... pn btn atn wtn tatn ctn
12. Display the average waiting time and average turnaround time.
13. End.

**Code:**

```
#include <iostream>
#include <iomanip>
using namespace std;

struct Process
{
```

```
    int burst_time;
    int arrival_time;
    int waiting_time;
    int turnaround_time;
    int completion_time;
};

void CalcCompletionTime(Process process[], int n)
{
              process[0].completion_time    =    process[0].burst_time    +
process[0].arrival_time;
    for (int i = 1; i < n; i++)
    {
         process[i].completion_time = (max(process[i - 1].completion_time,
process[i].arrival_time) + process[i].burst_time);
    }
}

void CalcWaitingTime(Process process[], int n)
{
    process[0].waiting_time = 0;
    for (int i = 1; i < n; i++)
    {
         process[i].waiting_time = max(0, process[i - 1].completion_time -
process[i].arrival_time);
    }
}

void CalcTurnaroundTime(Process process[], int n)
{
    for (int i = 0; i < n; i++)
    {
                process[i].turnaround_time  =  process[i].waiting_time  +
process[i].burst_time;
    }
}

int AvgWaiting(Process process[], int n)
{
    int avg, sum;
    for (int i = 0; i < n; i++)
    {
        sum = sum + process[i].waiting_time;
    }
```

```
    avg = sum / n;
    return avg;
}

int AvgTurnaround(Process process[], int n)
{
    int avg, sum;
    for (int i = 0; i < n; i++)
    {
        sum = sum + process[i].turnaround_time;
    }
    avg = sum / n;
    return avg;
}

void swap(Process *a, Process *b)
{
    Process temp = *a;
    *a = *b;
    *b = temp;
}

void display(Process process[], int n)
{
      cout<<"Process      Arrival Time      Burst Time      Completion Time
Turnaround Time    Waiting Time";
    for(int i=0;i<n;i++)
    {
        cout<<"\n"<<i+1;
        cout<<setw(15)<<process[i].arrival_time;
        cout<<setw(15)<<process[i].burst_time;
        cout<<setw(15)<<process[i].completion_time;
        cout<<setw(25)<<process[i].turnaround_time;
        cout<<setw(15)<<process[i].waiting_time;
    }
}

void SortProcesses(Process a[], int n)
{
    for(int i=0;i<n-1;i++)
    {
        for (int j=0;j<n-1-i;j++)
        {
            if(a[j].arrival_time>a[j+1].arrival_time)
```

```
            {
                swap(&a[j],&a[j+1]);
            }
        }
    }
}

int main()
{
    int n, i;
    int avg_waiting_time, avg_turnaround_time;
    cout << "Enter number of processes: ";
    cin >> n;
    Process process[n];
    for (i = 0; i < n; i++)
    {
        cout << "Enter burst time for process " << i + 1<<": ";
        cin >> process[i].burst_time;
        cout << "Enter arrival time for process " << i + 1<<": ";
        cin >> process[i].arrival_time;
    }


    SortProcesses(process, n);
    CalcCompletionTime(process, n);
    CalcWaitingTime(process, n);
    CalcTurnaroundTime(process, n);
    avg_waiting_time = AvgWaiting(process, n);
    avg_turnaround_time = AvgTurnaround(process, n);
    display(process, n);
    cout<<"\nAverage Waiting Time: "<<avg_waiting_time;
    cout<<"\nAverage Turnaround time: "<<avg_turnaround_time;

    return 0;
}
```

**Input and Output:**

```
Enter number of processes: 5
Enter burst time for process 1: 2
Enter arrival time for process 1: 3
Enter burst time for process 2: 4
Enter arrival time for process 2: 5
Enter burst time for process 3: 6
Enter arrival time for process 3: 2
Enter burst time for process 4: 9
Enter arrival time for process 4: 6
Enter burst time for process 5: 8
Enter arrival time for process 5: 0
Process    Arrival Time    Burst Time    Completion Time    Turnaround Time    Waiting Time
1              0              8              8                  8                0
2              2              6              14                 12               6
3              3              2              16                 13               11
4              5              4              20                 15               11
5              6              9              29                 23               14
Average Waiting Time: 9
Average Turnaround time: 23
```

**Conclusion:**

The experiment conducted involved the implementation of a C++ program to demonstrate the FCFS algorithm, resulting in an improved understanding of scheduling algorithms and their implementation. The experiment allowed for the demonstration of the FCFS algorithm in scheduling processes and calculating waiting time, turnaround time, and completion time. It helped to understand the working mechanism of an FCFS Algorithm.