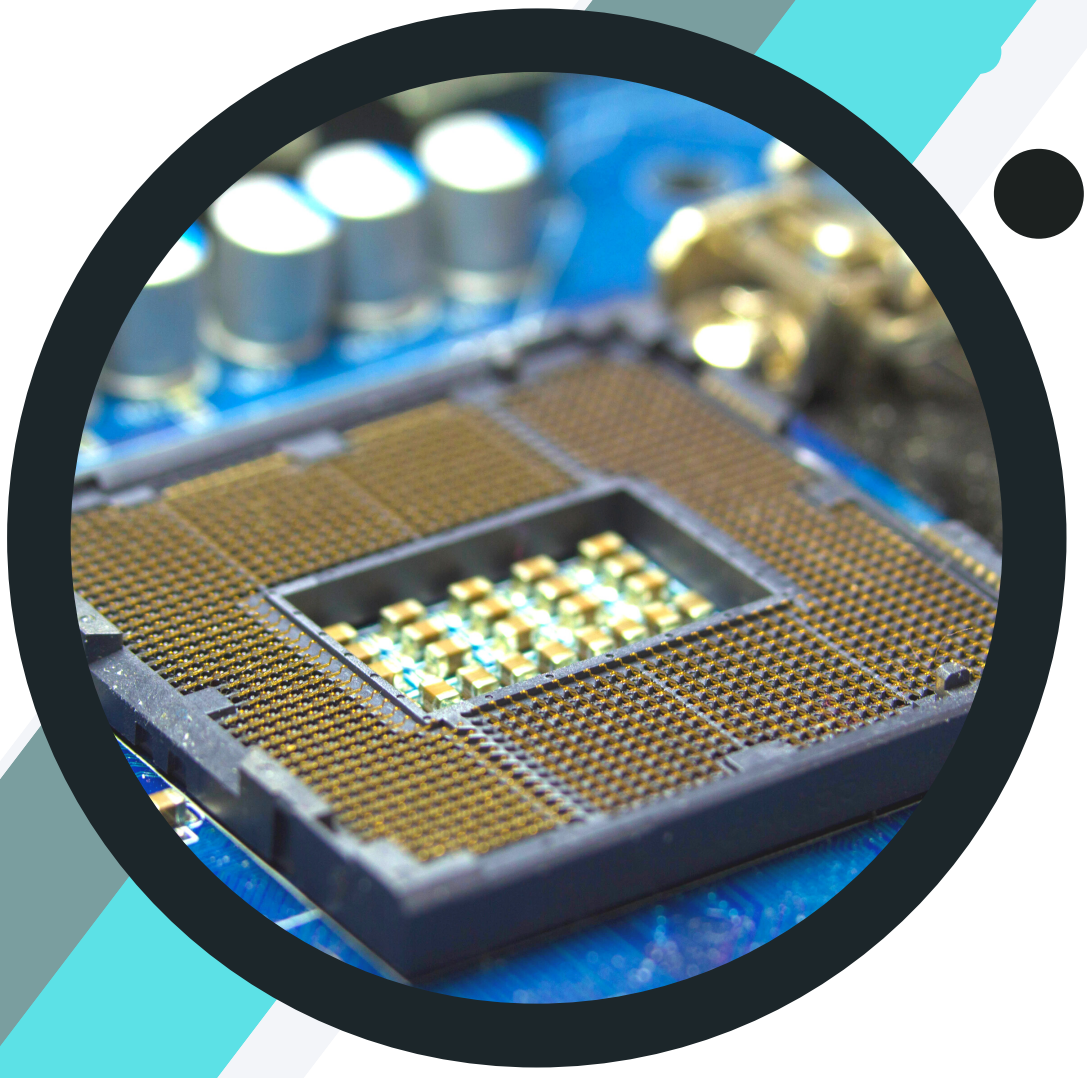
MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
A Constituent Institution of Manipal University

CPU-SCHEDULING ALGORITHM VISUALISER

OS-PROJECT
Section-C



Prepared By:	Rolllno:
Saurabh kumar Mishra	53
Raghav Sharma	54
Pradyumn Kamath	60
Soumyajit Saha	65

INDEX

ABSTRACT

Scheduling of the central processing unit (CPU) switches the CPU between different operations, which has a profound impact. Because the processor is a key resource, CPU scheduling becomes crucial to achieving the OS's design objectives. To maximize CPU usage, the OS should permit as many processes as feasible to run continuously.

The construction of high-quality scheduling algorithms that meet the scheduling objectives is a prerequisite for creating a CPU scheduler with high efficiency.

Using this knowledge, we have created a CPU scheduling visualizer to help intuitively demonstrate the parameters used to schedule processes using four types of algorithms.

WHY DO WE NEED CPU-SCHEDULING?

A procedure to finish its execution desires each CPU time and I/O time. In a multiprogramming system, there may be one procedure for the usage of the CPU even as any other is ready for I/O, whereas, in a uni programming system, time spent ready for I/O is wasted as the CPU is idle at this time. The multiprogramming may be finished with the aid of using the usage of procedure scheduling.

The goals of a scheduling set of rules are as follows:

- Maximize the CPU utilization, which means that hold the CPU as busy as possible.
- Fair allocation of CPU time to each procedure
- Maximize the Throughput
- Minimize the turnaround time
- Minimize the ready time
- Minimize the reaction time

Preemptive Scheduling Algorithms
In these algorithms, processes are assigned with priority. Whenever a high-priority process comes in, the lower-priority process that occupies the CPU is preempted. It releases the CPU, and the high-priority process takes the CPU for its execution.

Non-Preemptive Scheduling Algorithms
In these algorithms, we cannot preempt the process. Once a process runs in the CPU, it will release it either by context switching or terminating. Often, these are the types of algorithms that can be used because of hardware limitations.

There are some important terminologies to know for understanding the scheduling algorithms:

- Arrival Time: This is when a process arrives in the ready queue.
- Completion Time: This is when a process completes its execution.
- Burst Time: This is the time required by a process for CPU execution.
- Turn-Around Time: This is the difference in time between completion time and arrival time.

This can be calculated as follows:

Turn Around Time = Completion Time – Arrival Time.

- Waiting Time: This is the difference in time between turnaround time and burst time.

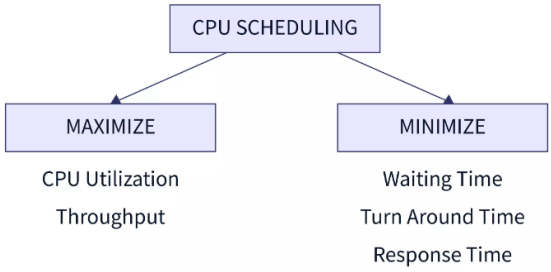
This can be calculated as follows:

Waiting Time = Turn Around Time – Burst Time.

Throughput: The number of processes completing their execution per unit of time.

CPU Scheduling Criteria:

A CPU scheduling algorithm tries to maximize and minimize the following:



Maximize:

CPU utilization: The main task in which the operating system must ensure that the CPU remains as busy as possible. It can range from 0 to 100 percent. However, for the RTOS, it can range from 40 percent for low-level and 90 percent for the high-level system.

Throughput:

The number of processes that finish their execution per unit of time is known as Throughput. So, when the CPU is busy executing the process, that time work is being done, and the work completed per unit of time is called Throughput.

Minimize:

Waiting time: Waiting time is an amount that a specific process needs to wait in the ready queue.

Response time:

The time request was submitted until the first response was produced.

Turnaround Time: Turnaround time is the amount of time to execute a specific process. It calculates the time spent waiting to get into the memory, waiting in the queue, and executing on the CPU. The period between the time of process submission to the completion time is the turnaround time.

Interval Timer:

Timer interruption is a method that is closely related to preemption. When a certain process gets the CPU allocation, a timer may be set to a specified interval. Both timer interruption and preemption force a process to return the CPU before its CPU burst is complete.

The most multi-programmed operating system uses some form of a timer to prevent a process from tying up the system forever.

ALGORITHMS

There are mainly six types of process scheduling algorithms

1. First Come First Serve (FCFS)
2. Shortest-Job-First (SJF) Scheduling
3. Shortest Remaining Time First Scheduling
4. Round Robin Scheduling
5. Priority Scheduling
6. Multilevel Queue Scheduling

WE HAVE DEMONSTRATED THE MOST POPULAR 4 TYPES OF SCHEDULING ALGORITHMS USED.

- ***First Come First Serve (FCFS)***
- ***Shortest-Job-First (SJF) Scheduling***
- ***Shortest Remaining Time First Scheduling***
- ***Round Robin Scheduling***

FCFS-SCHEDULING

The complete phrase for FCFS is **First Come First Serve**. The easiest and simplest CPU scheduling algorithm is this one. The process that asks the CPU receives the CPU allocation first in this kind of procedure. A FIFO queue can be used to manage this scheduling technique.

The process' PCB (Process Control Block) is connected to the tail of the queue as it enters the ready queue. Therefore, CPU should be allocated to the process at the front of the queue as soon as it becomes available.

SJF-SCHEDULING

SRT stands for **Shortest Remaining Time** in its entire form. A different name for it is SJF preemptive scheduling. The task closest to completion will receive the procedure in this approach. The completion of an older process won't be delayed by a newer ready-state process thanks to this technique.

SRTF-SCHEDULING

The scheduling technique is known as SJF, or "**Shortest Job First**," which chooses the process with the quickest execution time to run next. Both preemptive and non-preemptive scheduling strategies are possible. The average time that other processes are left waiting to be executed greatly decreases.

RR-SCHEDULING

The oldest and simplest scheduling algorithm is **round-robin**. The round-robin principle, in which each person gets an equal amount of something, in turn, inspired the name of this method. It is primarily employed for multitasking scheduling algorithms. This algorithmic approach aids in the execution of tasks without starving.

METHODOLOGY

- Scheduling algorithms tell the CPU which will be the next process to have CPU time.
- The main goal of scheduling algorithms is to Maximize Throughput.
- Scheduling algorithms can be preemptive and non-preemptive.
- First Come, First Serve, Shortest Job First, Shortest Remaining Time First, and Round Robin are four widely used scheduling algorithms, each with its own advantages and disadvantages.
- The best scheduling algorithms depend on the situation, needs, and hardware and software capabilities.

Keeping these following points in mind, we have created a basic visualization software to calculate and demonstrate properly widely used CPU scheduling with the help of an interactive website.

TECH-STACK USED

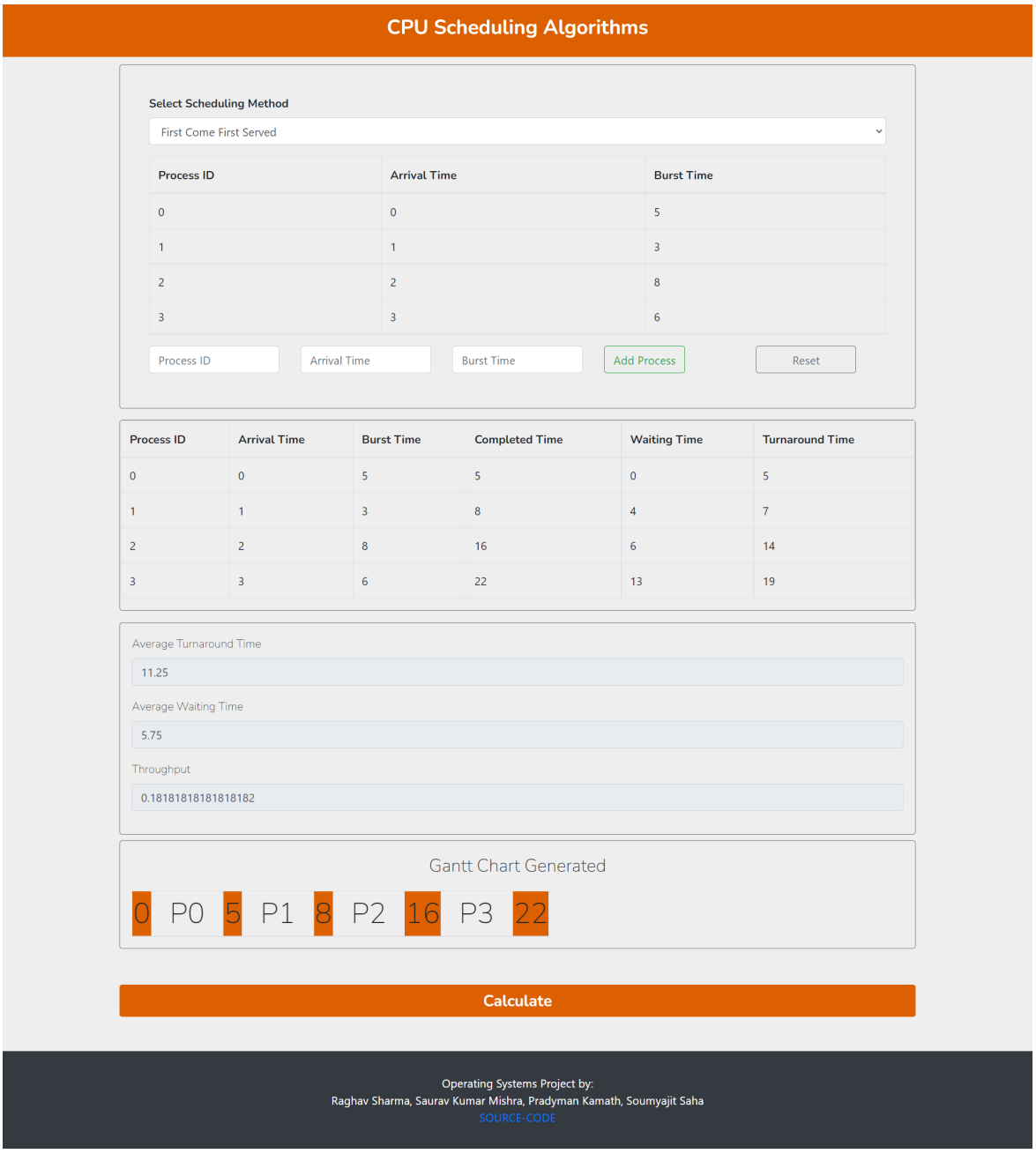
- **JAVASCRIPT** - All algorithms used in the project are implemented in JAVASCRIPT.
- **HTML** - Outlines the front-end part is done in HTML.
- **CSS** - Customization to all the front end is done in CSS.
- **BOOTSTRAP** - It is added to make the website modular.

IMPLEMENTATION

1-FCFS-SCHEDULING

```
function firstComeFirstServed() {
    var time = 0;
    var queue = [];
    var completedList = [];
    while (processList.length > 0 || queue.length > 0) {
     addToQueue();
     while (queue.length == 0) {
      time++;
      addToQueue();
     }
     // Dequeue from queue and run the process.
     process = queue.shift();
     for (var i = 0; i < process.burstTime; i++) {
      time++
      addToQueue();
     }
     process.completedTime = time;
     process.turnAroundTime = process.completedTime - process.arrivalTime;
     process.waitingTime = process.turnAroundTime - process.burstTime;
     completedList.push(process);
    }
    function addToQueue() {
     for (var i = 0; i < processList.length; i++) {
      if (time >= processList[i].arrivalTime) {
       var process = {
        processID: processList[i].processID,
        arrivalTime: processList[i].arrivalTime,
        burstTime: processList[i].burstTime
       }
       processList.splice(i, 1);
       queue.push(process);
      }
     }
   }
```

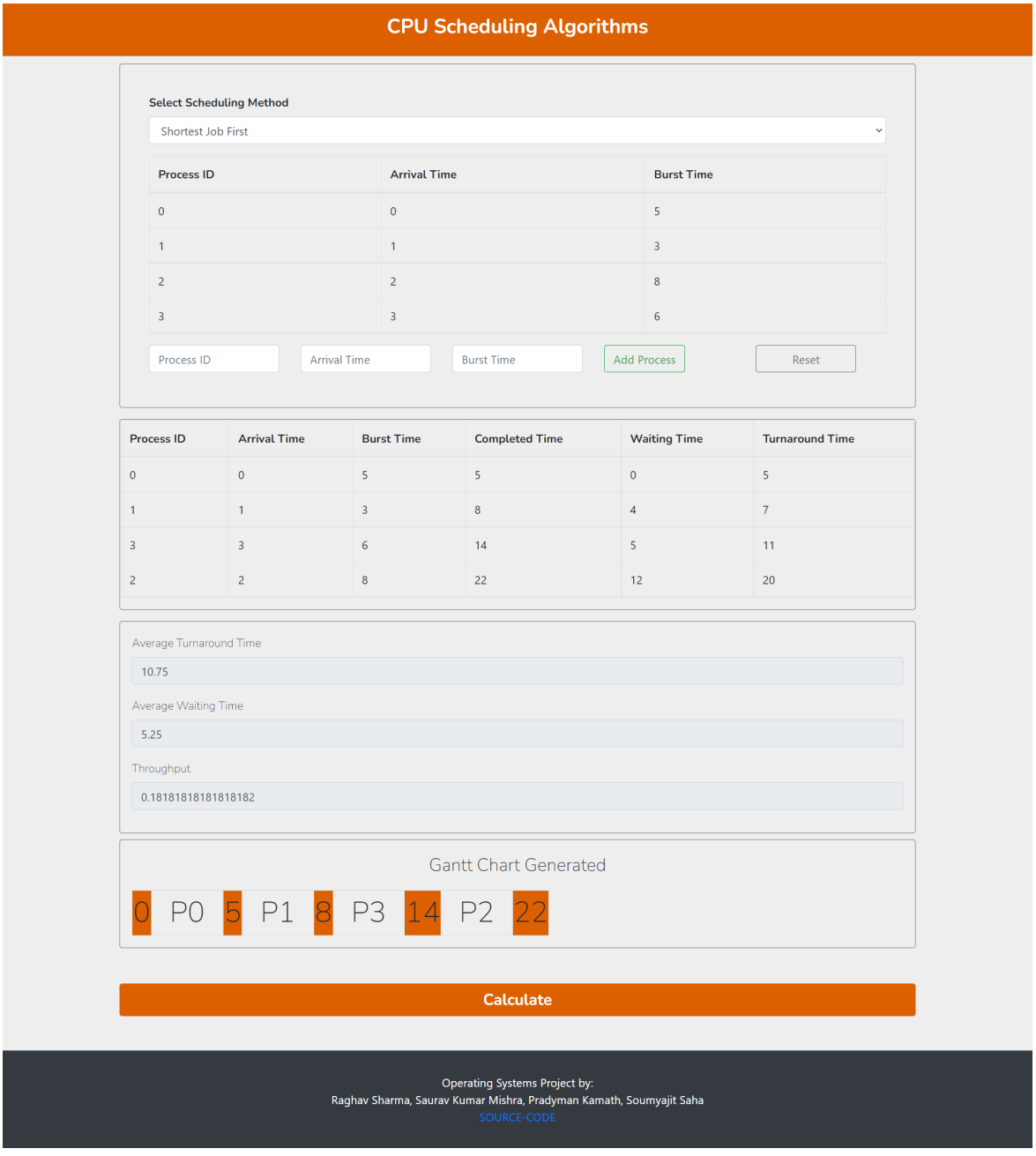
Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



2-SJF-SCHEDULING

```
function shortestJobFirst() {
    var completedList = [];
    var time = 0;
    var queue = [];
    while (processList.length > 0 || queue.length > 0) {
     addToQueue();
     while (queue.length == 0) {
      time++;
      addToQueue();
     }
     processToRun = selectProcess();
     for (var i = 0; i < processToRun.burstTime; i++) {
      time++;
      addToQueue();
     }
     processToRun.processID = processToRun.processID;
     processToRun.arrivalTime = processToRun.arrivalTime;
     processToRun.burstTime = processToRun.burstTime;
     processToRun.completedTime = time;
     processToRun.turnAroundTime = processToRun.completedTime -
processToRun.arrivalTime;
     processToRun.waitingTime = processToRun.turnAroundTime -
processToRun.burstTime;
     completedList.push(processToRun);
    }
    function addToQueue() {
     for (var i = 0; i < processList.length; i++) {
      if (processList[i].arrivalTime === time) {
       var process = {
        processID: processList[i].processID,
        arrivalTime: processList[i].arrivalTime,
        burstTime: processList[i].burstTime
       }
       processList.splice(i, 1);
       queue.push(process);
      }
     }
    }
```

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



3-SRTF-SCHEDULING

```
function shortestRemainingTimeFirst() {
    var completedList = [];
    var time = 0;
    var queue = [];
    while (processList.length > 0 || queue.length > 0) {
     addToQueue();
     while (queue.length == 0) {
      time++;
      addToQueue();
     }
     selectProcessForSRTF();
     runSRTF();
    }

    function addToQueue() {
     for (var i = 0; i < processList.length; i++) {
      if (processList[i].arrivalTime === time) {
       var process = {
        processID: processList[i].processID,
        arrivalTime: processList[i].arrivalTime,
        burstTime: processList[i].burstTime
       }
       processList.splice(i, 1);
       queue.push(process);
      }
     }
    }
```

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16

4-RR-SCHEDULING

```
function roundRobin() {
    var timeQuantum = $('#timeQuantum');
    var timeQuantumVal = parseInt(timeQuantum.val(), 10);
    if (timeQuantum.val() == '') {
      alert('Please enter time quantum');
      timeQuantum.addClass('is-invalid');
      return;
    }
    var completedList = [];
    var time = 0;
    var queue = [];

    while (processList.length > 0 || queue.length > 0) {
      addToQueue();
      while (queue.length == 0) {
        time++;
        addToQueue();
      }
      selectProcessForRR();
    }

    function addToQueue() {
      for (var i = 0; i < processList.length; i++) {
        if (processList[i].arrivalTime === time) {
          var process = {
            processID: processList[i].processID,
            arrivalTime: processList[i].arrivalTime,
            burstTime: processList[i].burstTime
          }
          processList.splice(i, 1);
          queue.push(process);
        }
      }
    }
```

Process Id	Arrival time	Burst time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

If the CPU scheduling policy is Round Robin with time quantum = 2 unit, calculate the average waiting time and average turn around time.

CPU Scheduling Algorithms

Select Scheduling Method

Round Robin

Time Quantum :

2

Process ID	Arrival Time	Burst Time
1	0	5
2	1	3
3	2	1
4	3	2
5	4	3

Process ID | Arrival Time | Burst Time | Add Process | Reset

Process ID	Arrival Time	Burst Time	Completed Time	Waiting Time	Turnaround Time
3	2	1	5	2	3
4	3	2	9	4	6
2	1	3	12	8	11
1	0	5	13	8	13
5	4	3	14	7	10

Average Turnaround Time

8.6

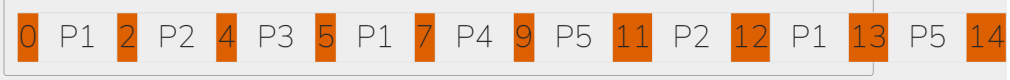
Average Waiting Time

5.8

Throughput

0.35714285714285715

Gantt Chart Generated



Calculate

Operating Systems Project by:
Raghav Sharma, Saurav Kumar Mishra, Pradyman Kamath, Soumyajit Saha
SOURCE-CODE

CONCLUSION

Working on this operating systems mini-project gave us an in-depth grasp of how CPU scheduling algorithms compute critical parameters and use them to schedule processes. We also learned how to utilize a variety of tech stacks for the project, including javascript, HTML, CSS, and bootstrap. By developing the project and analyzing the results, we could visualize the Gantt chart generated by our algorithm to schedule processes.

REFERENCE

https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/#fcfs

https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/#sjf

https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/#rr

https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/#srtf

https://afteracademy.com/blog/process-scheduling-algorithms-in-the-operating-system

https://www.guru99.com/round-robin-scheduling-example.html

https://www.tutorialspoint.com/operating_system/os_process_scheduling_algorithms.htm

SAMPLE INPUTS FOR THE PROJECT

FCFS
https://afteracademy.com/blog/process-scheduling-algorithms-in-the-operating-system

SJF
https://www.tutorialspoint.com/operating_system/os_process_scheduling_algorithms.htm

SRTF
https://www.javatpoint.com/os-srtf-scheduling-algorithm

RR
https://www.javatpoint.com/os-round-robin-scheduling-example