A Course Based Project Report

On

Scheduling Algorithms with graphical representation and analysis

Submitted in partial fulfillment of requirement

for the completion of the

Operating Systems Laboratory

I B.Tech Computer Science and Engineering

of

VNR VJIET

By

P. SUNIL	**23071A05J2**
P. KARTHIK REDDY	**23071A05J4**
P. HARIN GUPTHA	**23071A05J5**
P. BHARATH	**23071A05J6**
P. VENKAT REDDY	**23071A0J3**

2024-2025

VALLURIPALLI NAGESWARA RAO
VIGNANA JYOTHI INSTITUTE OF ENGINEERING &TECHNOLOGY
(AUTONOMOUS INSTITUTE)
NAAC ACCREDITED WITH 'A++' GRADE
NBA Accreditation for B. Tech Programs
Vignana Jyothi Nagar, Bachupally, Nizampet (S.O), Hyderabad 500090
Phone no: 040-23042758/59/60, Fax: 040-23042761
Email:postbox@vnrvjiet.ac.in Website: www.vnrvjiet.ac.in

A Project Report On

Scheduling Algorithms with graphical representation and analysis

Submitted in partial fulfillment of requirement
for the completion of the
Data Structures Laboratory

B.Tech Computer Science and Engineering

of

VNRVJIET

2024-2025

Under the Guidance

of

DR. D N VASUNDHARA
Assistant Professor
Department of CSE

VNR VIGNANA JYOTHI INSTITUTE OF ENGINEERING & TECHNOLOGY

(AUTONOMUS INSTITUTE)

NAAC ACCREDITED WITH 'A++' GRADE

CERTIFICATE

This is to certify that the project entitled "Scheduling Algorithms with graphical representation and analysis" submitted in partial fulfillment for the course of Data Structures laboratory being offered for the award of Batch (CSE-C) by VNRVJIET is a result of the bonafide work carried out by **23071A05J2**, **23071A05J3, 23071A05J4, 23071A05J5** and **23071A05J6** during the year **2024-2025**.

This has not been submitted for any other certificate or course.

.

Signature of Faculty **Signature of Head of the Department**

ACKNOWLEDGEMENT

An endeavor over a long period can be successful only with the advice and support of many well-wishers. We take this opportunity to express our gratitude and appreciation to all of them.

We wish to express our profound gratitude to our honorable **Principal Dr. C.D.Naidu** and **HOD Dr.S.Nagini**, **CSE Department**, **VNR Vignana Jyothi Institute of Engineering and Technology** for their constant and dedicated support towards our career moulding and development.

With great pleasure we express our gratitude to the internal guide **DR. DN VASUNDHARA, Assistant Professor, CSE department** for his timely help, constant guidance, cooperation, support and encouragement throughout this project as it has urged us to explore many new things.

Finally, we wish to express my deep sense of gratitude and sincere thanks to our parents, friends and all our well-wishers who have technically and non-technically contributed to the successful completion of this course-based project.

DECLARATION

We hereby declare that this Project Report titled "**Scheduling Algorithms with graphical representation and analysis**" submitted by us of Computer Science & Engineering in **VNR Vignana Jyothi Institute of Engineering and Technology**, is a bonafide work undertaken by us and it is not submitted for any other certificate /Course or published any time before.

Name & Signature of the Students:

P SUNIL	**23071A05J2**
P VENKAT REDDY	**23071A05J3**
P KARTHIK REDDY	**23071A05J4**
P HARIN GUPTHA	**23071A05J5**
P BHARATH	**23071A05J6**

Date:

TABLE OF CONTENTS

ABSTRACT

Efficient CPU scheduling is fundamental to enhancing system performance, reducing waiting time, and ensuring optimal utilization of processor resources in modern computing environments. With the increasing complexity of multi-programming systems, the need for flexible and intelligent scheduling algorithms has become more critical. This project presents a comprehensive simulation of nine distinct CPU scheduling algorithms, each tailored to address specific execution scenarios within an operating system.

The system supports a wide range of scheduling strategies, including First Come First Serve (FCFS), Shortest Job First (SJF), Shortest Remaining Job First (SRJF), Longest Job First (LJF), Longest Remaining Job First (LRJF), Priority-based scheduling (Preemptive and Non-Preemptive), Round Robin (RR), and Highest Response Ratio Next (HRRN). Each algorithm is implemented with support for multiple CPU and I/O burst times per process, reflecting real-world operating system behavior. The simulation accounts for context switching time, allowing a realistic approximation of scheduling overhead.

One of the core features of the project is its interactive visualization, which includes both a Gantt Chart and a Timeline Chart to represent the execution sequence of processes. The animation of time logs enhances user understanding of how each algorithm behaves over time. Furthermore, the system allows detailed comparisons between algorithms based on performance metrics such as Average Completion Time, Turnaround Time, Waiting Time, and Response Time.
Special emphasis is placed on analyzing the Round Robin algorithm with varying time quanta to observe its performance sensitivity. The user can input processes, observe state transitions (Remain, Ready, Running, Block, Terminate), and understand the decision-making mechanisms of each scheduling approach.

Developed using HTML, CSS, Vanilla JavaScript, Google Charts, and Chart.js, the application provides a dynamic and user-friendly interface. This simulation serves as a valuable educational tool for students and researchers to study and compare CPU scheduling techniques in an interactive manner.

In conclusion, this project offers a versatile and educational platform to explore the behavior and efficiency of various CPU scheduling algorithms. By simulating real-time process execution and allowing metric-based comparisons, it provides insights into the strengths and trade-offs of different scheduling methods, thereby contributing to a deeper understanding of operating system concepts.

INTRODUCTION

In modern operating systems, process scheduling is a crucial task that directly impacts system performance, efficiency, and responsiveness. With multiple processes competing for CPU time, the operating system must determine the order in which these processes are executed. CPU scheduling algorithms play a fundamental role in achieving this goal by managing the execution flow of processes in a fair and optimized manner. Each algorithm is designed with specific criteria, such as minimizing waiting time, turnaround time, or giving priority to critical processes.

This project focuses on simulating and comparing nine widely-used CPU scheduling algorithms: First Come First Serve (FCFS), Shortest Job First (SJF), Shortest Remaining Job First (SRJF), Longest Job First (LJF), Longest Remaining Job First (LRJF), Priority-based (Preemptive and Non-Preemptive), Round Robin (RR), and Highest Response Ratio Next (HRRN). These algorithms cover both preemptive and non-preemptive scheduling strategies. The system allows users to define multiple CPU and I/O burst times for each process, offering a realistic and flexible simulation environment.

To aid in understanding and analysis, the application provides visual outputs such as Gantt Charts, Timeline Charts, and animated logs of the execution process. Context switching is also accounted for, which adds a layer of realism to the simulation. Users can interactively observe how each algorithm handles different process arrival patterns and burst times. A comparative analysis is also available, allowing users to evaluate algorithms based on performance metrics like average completion time, waiting time, turnaround time, and response time.

The project is built using core web technologies including HTML, CSS, JavaScript, Google Charts, and Chart.js. It serves as an educational tool for students, educators, and developers to explore and understand the working principles of various CPU scheduling strategies. By visualizing complex scheduling behaviors and enabling metric-based comparison, the system bridges the gap between theoretical learning and practical understanding of operating system process management.

METHODOLOGY

The methodology for developing the CPU Scheduling Algorithms simulator focuses on accurately implementing and comparing various scheduling strategies used in operating systems. The process involves several well-defined phases: system initialization and input handling, scheduling logic execution, visualization and animation, and performance metrics comparison. Each phase is carefully structured to ensure interactive learning, effective simulation, and meaningful performance analysis.

1. This phase sets up the environment by initializing all necessary variables and data structures. The user is prompted to input essential details such as the number of processes, their arrival times, CPU burst times, I/O burst times, priorities (if required), and context switching time. The system allows dynamic and flexible inputs to accommodate real-world-like process behaviors. Processes are stored in structured arrays or objects for efficient access and manipulation.

2. The core of the simulator is the implementation of nine different CPU scheduling algorithms: FCFS, SJF, SRJF, LJF, LRJF, Priority (Preemptive and Non-Preemptive), Round Robin, and HRRN. Each algorithm has its own function or module encapsulating the logic to determine which process runs next based on its specific criteria (arrival time, burst time, priority, etc.). Both preemptive and non-preemptive scheduling strategies are supported. Context switching is factored into the simulation to ensure accurate timing and realistic processor behavior.

3. Once scheduling decisions are made, the simulator generates a Gantt Chart and Timeline Chart to visually represent process execution over time. Animations of the time log help users observe how processes move through different states (ready, running, waiting, terminated) as time progresses. Google Charts and Chart.js are utilized to make the visualization interactive, colorful, and informative. The inclusion of context switch animations provides deeper insights into how frequent switches affect overall performance.

4. After running each algorithm, the simulator computes key performance metrics: Average Completion Time, Turnaround Time, Waiting Time, and Response Time. These metrics are displayed in a tabular format and compared across all algorithms. A special module compares different time quantum values for the Round Robin algorithm, showing how performance changes with time slice variations. This comparative study helps users understand trade-offs between fairness, responsiveness, and throughput in scheduling.

OBJECTIVES

The primary objective of this CPU Scheduling Algorithms project is to design and simulate various scheduling strategies to evaluate their impact on process management in an operating system. This system is designed with the following specific objectives:

1. **Implementation of Multiple Scheduling Algorithms**
    - To accurately implement nine different CPU scheduling algorithms including FCFS, SJF, SRJF, LJF, LRJF, Priority (Preemptive and Non-Preemptive), Round Robin, and HRRN.
    - To allow the user to switch between algorithms for comparative evaluation.
2. **Interactive Process Input Handling**
    - To enable users to dynamically enter process data such as arrival time, burst time, I/O burst time, priority, and context switch time.
    - To provide flexibility in simulating real-world process behaviors with various characteristics.
3. **Process Execution Simulation and Visualization**
    - To simulate the execution of processes based on the selected algorithm and animate the process timeline.
    - To display Gantt Charts and Timeline Charts using Google Charts and Chart.js for visual understanding.
4. **Handling of Context Switching**
    - To include context switch duration in the simulation for realistic CPU behavior.
    - To reflect the overhead and impact of context switching on overall process execution time.
5. **Comparative Analysis of Time Quantum in Round Robin**
    - To compare how different time quantum values affect the performance of the Round Robin algorithm.
    - To help users understand the trade-off between responsiveness and throughput.
6. **Performance Metrics Evaluation**
    - To calculate and display key metrics such as Average Completion Time, Turnaround Time, Waiting Time, and Response Time.
    - To compare these metrics across different algorithms for better insight into their efficiencies.
7. **Educational and Analytical Tool**
    - To provide a hands-on, visual learning experience for students and learners studying operating system concepts.
    - To serve as a platform for analyzing and improving scheduling techniques through experimentation.

By meeting these objectives, the system aims to offer a comprehensive and educational environment for understanding CPU scheduling techniques and their practical implications on process management in modern operating systems.

FLOW OF EXECUTION

CPU Scheduling Algorithms Flow

Start Program
|
V
User Input Phase
|
V
Initialize Data Structures
|
V
User Selection
|
V
Scheduling and Execution
|
V
Log and Visualize Output
|
V
Display Summary
|
V
End Program

IMPLEMENTATION OF PROGRAM

Code:

Index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Scheduling Algorithms</title>
    <link rel="icon" href="favicon.png">
    <link rel="stylesheet" href="style.css">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.7.0/css/font-awesome.min.css">

</head>

<body>

    <h1>Scheduling Algorithms</h1>
    <h3>Instructions :</h3>
    <ol class="instructions">
        <li>Tiebreaker is Process ID.</li>
        <li>When burst time is criteria, total burst time is taken into
consideration.</li>
        <li>Context Switch not considered at start and end of processes.</li>
    </ol>
    <h3>Preferences :</h3>
    <ol class="preferences">
        <li>
            Priority : 1 is
            <button id="priority-toggle-btn">
                <span id="priority-preference">high</span>
                <i class="fa fa-refresh"></i>
            </button>
        </li>
        <form id="algorithms-form">
            <label for="algo"><h3>Algorithms : </h3></label>
            <select name="algo" id="algo">
                <option value="fcfs">First Come First Serve (FCFS)</option>
                <option value="sjf">Shortest Job First (SJF)</option>
                <option value="ljf">Longest Job First (LJF)</option>
                <option value="srtf">Shortest Remaining Job First
(SRTF)</option>
                <option value="lrtf">Longest Remaining Job First (LRTF)</option>
                <option value="rr">Round Robin</option>
                <option value="pnp">Priority (Non Preemptive)</option>
                <option value="pp">Priority (Preemptive)</option>
                <option value="hrrn">Highest Response Ratio Next (HRRN)</option>
```

```
            </select>
        </form>
        <br>
        <table class="main-table">
            <thead>
                <tr>
                    <th class="process-id">Process ID</th>
                    <th class="priority hide">Priority</th>
                    <th class="arrival-time">Arrival Time</th>
                    <th class="process-time" colspan="1">Process Time</th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td class="process-id" rowspan="2">P1</td>
                    <td class="priority hide" rowspan="2"><input type="number"
min="1" step="1" value="1"></td>
                    <td class="arrival-time" rowspan="2"><input type="number"
min="0" step="1" value="0"> </td>
                    <td class="process-time cpu process-heading"
colspan="">CPU</td>
                    <td class="process-btn"><button type="button" class="add-
process-btn">+</button></td>
                    <td class="process-btn"><button type="button" class="remove-
process-btn">-</button></td>
                </tr>
                <tr>
                    <td class="process-time cpu process-input"><input
type="number" min="1" step="1" value="1"> </td>
                </tr>
            </tbody>
        </table>
        <br>
        <button type="button" class="add-btn">Add Process</button>
        <button type="button" class="remove-btn">Delete Process</button>
        <div id="context-switch-div">
            <br>
            <label for="context-switch">Context Switch Time : </label>
            <input type="number" name="Context Switch" id="context-switch"
min="0" step="1" value="0">
        </div>
        <div id="time-quantum" class="hide">
            <br>
            <label for="tq">Time Quantum : </label>
            <input type="number" name="Time Quantum" id="tq" min="1" step="1"
value="1">
        </div>
        <br>
        <button type="button" id="calculate">Calculate</button>
        <button type="button" id="reset"
onClick="window.location.reload();">Reset</button>
        <div id="output"></div>
        <script type="text/javascript"
src="https://www.gstatic.com/charts/loader.js"></script>
        <script
src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.min.js"
integrity="sha512-
d9xgZrVZpmmQlfonhQUvTR7lMPtO7NkZMkA0ABN3PHCbKA5nqylQ/yWlFAyY6hYgdF1Qh6nYiuADWwKB
```

```
4C2WSw==" crossorigin="anonymous"></script>
        <script src="script.js"></script>

</body>

</html>
```

Scrpit.js

```
let priorityPreference = 1; //priority preferences change
document.getElementById("priority-toggle-btn").onclick = () => {
    let currentPriorityPreference = document.getElementById("priority-
preference").innerText;
    if (currentPriorityPreference == "high") {
        document.getElementById("priority-preference").innerText = "low";
    } else {
        document.getElementById("priority-preference").innerText = "high";
    }
    priorityPreference *= -1;
};

let selectedAlgorithm = document.getElementById('algo');

function checkTimeQuantumInput() {
    let timequantum = document.querySelector("#time-quantum").classList;
    if (selectedAlgorithm.value == 'rr') {
        timequantum.remove("hide");
    } else {
        timequantum.add("hide");
    }
}

function checkPriorityCell() {
    let prioritycell = document.querySelectorAll(".priority");
    if (selectedAlgorithm.value == "pnp" || selectedAlgorithm.value == "pp") {
        prioritycell.forEach((element) => {
            element.classList.remove("hide");
        });
    } else {
        prioritycell.forEach((element) => {
            element.classList.add("hide");
        });
    }
}

selectedAlgorithm.onchange = () => {
    checkTimeQuantumInput();
    checkPriorityCell();
```

```
};

function inputOnChange() { //onchange EventListener for input
    let inputs = document.querySelectorAll('input');
    inputs.forEach((input) => {
        if (input.type == 'number') {
            input.onchange = () => {
                let inputVal = Number(input.value);
                let isInt = Number.isInteger(inputVal);
                if (input.parentNode.classList.contains('arrival-time') ||
input.id == 'context-switch') //min 0 : arrival time
                {
                    if (!isInt || (isInt && inputVal < 0)) {
                        input.value = 0;
                    } else {
                        input.value = inputVal;
                    }
                } else //min 1 : time quantum, priority, process time
                {
                    if (!isInt || (isInt && inputVal < 1)) {
                        input.value = 1;
                    } else {
                        input.value = inputVal;
                    }
                }
            }
        }
    });
}
inputOnChange();
let process = 1;
//resize burst time rows size on +/-

function gcd(x, y) {
    while (y) {
        let t = y;
        y = x % y;
        x = t;
    }
    return x;
}

function lcm(x, y) {
    return (x * y) / gcd(x, y);
}

function lcmAll() {
    let result = 1;
    for (let i = 0; i < process; i++) {
        result = lcm(result, document.querySelector(".main-table").rows[2 * i +
```

```
2].cells.length);
    }
    return result;
}

function updateColspan() { //update burst time cell colspan
    let totalColumns = lcmAll();
    let processHeading = document.querySelector("thead .process-time");
    processHeading.setAttribute("colspan", totalColumns);
    let processTimes = [];
    let table = document.querySelector(".main-table");
    for (let i = 0; i < process; i++) {
        let row = table.rows[2 * i + 2].cells;
        processTimes.push(row.length);
    }
    for (let i = 0; i < process; i++) {
        let row1 = table.rows[2 * i + 1].cells;
        let row2 = table.rows[2 * i + 2].cells;
        for (let j = 0; j < processTimes[i]; j++) {
            row1[j + 3].setAttribute("colspan", totalColumns / processTimes[i]);
            row2[j].setAttribute("colspan", totalColumns / processTimes[i]);
        }
    }
}

function addremove() { //add remove bt-io time pair add event listener
    let processTimes = [];
    let table = document.querySelector(".main-table");
    for (let i = 0; i < process; i++) {
        let row = table.rows[2 * i + 2].cells;
        processTimes.push(row.length);
    }
    let addbtns = document.querySelectorAll(".add-process-btn");
    for (let i = 0; i < process; i++) {
        addbtns[i].onclick = () => {
            let table = document.querySelector(".main-table");
            let row1 = table.rows[2 * i + 1];
            let row2 = table.rows[2 * i + 2];
            let newcell1 = row1.insertCell(processTimes[i] + 3);
            newcell1.innerHTML = "IO";
            newcell1.classList.add("process-time");
            newcell1.classList.add("io");
            newcell1.classList.add("process-heading");
            let newcell2 = row2.insertCell(processTimes[i]);
            newcell2.innerHTML = '<input type="number" min="1" step="1"
value="1">';
            newcell2.classList.add("process-time");
            newcell2.classList.add("io");
            newcell2.classList.add("process-input");
            let newcell3 = row1.insertCell(processTimes[i] + 4);
```

```
            newcell3.innerHTML = "CPU";
            newcell3.classList.add("process-time");
            newcell3.classList.add("cpu");
            newcell3.classList.add("process-heading");
            let newcell4 = row2.insertCell(processTimes[i] + 1);
            newcell4.innerHTML = '<input type="number" min="1" step="1"
value="1">';
            newcell4.classList.add("process-time");
            newcell4.classList.add("cpu");
            newcell4.classList.add("process-input");
            processTimes[i] += 2;
            updateColspan();
            inputOnChange();
        };
    }
    let removebtns = document.querySelectorAll(".remove-process-btn");
    for (let i = 0; i < process; i++) {
        removebtns[i].onclick = () => {
            if (processTimes[i] > 1) {
                let table = document.querySelector(".main-table");
                processTimes[i]--;
                let row1 = table.rows[2 * i + 1];
                row1.deleteCell(processTimes[i] + 3);
                let row2 = table.rows[2 * i + 2];
                row2.deleteCell(processTimes[i]);
                processTimes[i]--;
                table = document.querySelector(".main-table");
                row1 = table.rows[2 * i + 1];
                row1.deleteCell(processTimes[i] + 3);
                row2 = table.rows[2 * i + 2];
                row2.deleteCell(processTimes[i]);
                updateColspan();
            }
        };
    }
}
addremove();

function addProcess() {
    process++;
    let rowHTML1 = `
                            <td class="process-id" rowspan="2">P${process}</td>
                            <td class="priority hide" rowspan="2"><input
type="number" min="1" step="1" value="1"></td>
                            <td class="arrival-time" rowspan="2"><input
type="number" min="0" step="1" value="0"> </td>
                            <td class="process-time cpu process-heading"
colspan="">CPU</td>
                            <td class="process-btn"><button type="button"
class="add-process-btn">+</button></td>
```

```
                        <td class="process-btn"><button type="button"
class="remove-process-btn">-</button></td>
                    `;
    let rowHTML2 = `
                         <td class="process-time cpu process-input"><input
type="number" min="1" step="1" value="1"> </td>
                    `;
    let table = document.querySelector(".main-table tbody");
    table.insertRow(table.rows.length).innerHTML = rowHTML1;
    table.insertRow(table.rows.length).innerHTML = rowHTML2;
    checkPriorityCell();
    addremove();
    updateColspan();
    inputOnChange();
}

function deleteProcess() {
    let table = document.querySelector(".main-table");
    if (process > 1) {
        table.deleteRow(table.rows.length - 1);
        table.deleteRow(table.rows.length - 1);
        process--;
    }
    updateColspan();
    inputOnChange();
}

document.querySelector(".add-btn").onclick = () => { //add row event listener
    addProcess();
};
document.querySelector(".remove-btn").onclick = () => { //remove row event
listener
    deleteProcess();
};
//------------------------
class Input {
    constructor() {
        this.processId = [];
        this.priority = [];
        this.arrivalTime = [];
        this.processTime = [];
        this.processTimeLength = [];
        this.totalBurstTime = [];
        this.algorithm = "";
        this.algorithmType = "";
        this.timeQuantum = 0;
        this.contextSwitch = 0;
    }
}
class Utility {
```

```
    constructor() {
        this.remainingProcessTime = [];
        this.remainingBurstTime = [];
        this.remainingTimeRunning = [];
        this.currentProcessIndex = [];
        this.start = [];
        this.done = [];
        this.returnTime = [];
        this.currentTime = 0;
    }
}
class Output {
    constructor() {
        this.completionTime = [];
        this.turnAroundTime = [];
        this.waitingTime = [];
        this.responseTime = [];
        this.schedule = [];
        this.timeLog = [];
        this.contextSwitches = 0;
        this.averageTimes = []; //ct,tat,wt,rt
    }
}
class TimeLog {
    constructor() {
        this.time = -1;
        this.remain = [];
        this.ready = [];
        this.running = [];
        this.block = [];
        this.terminate = [];
        this.move = []; //0-remain->ready 1-ready->running 2-running->terminate
3-running->ready 4-running->block 5-block->ready
    }
}

function setAlgorithmNameType(input, algorithm) {
    input.algorithm = algorithm;
    switch (algorithm) {
        case 'fcfs':
        case 'sjf':
        case 'ljf':
        case 'pnp':
        case 'hrrn':
            input.algorithmType = "nonpreemptive";
            break;
        case 'srtf':
        case 'lrtf':
        case 'pp':
            input.algorithmType = "preemptive";
```

```
            break;
        case 'rr':
            input.algorithmType = "roundrobin";
            break;
    }
}

function setInput(input) {
    for (let i = 1; i <= process; i++) {
        input.processId.push(i - 1);
        let rowCells1 = document.querySelector(".main-table").rows[2 * i -
1].cells;
        let rowCells2 = document.querySelector(".main-table").rows[2 * i].cells;
        input.priority.push(Number(rowCells1[1].firstElementChild.value));
        input.arrivalTime.push(Number(rowCells1[2].firstElementChild.value));
        let ptn = Number(rowCells2.length);
        let pta = [];
        for (let j = 0; j < ptn; j++) {
            pta.push(Number(rowCells2[j].firstElementChild.value));
        }
        input.processTime.push(pta);
        input.processTimeLength.push(ptn);
    }
    //total burst time for each process
    input.totalBurstTime = new Array(process).fill(0);
    input.processTime.forEach((e1, i) => {
        e1.forEach((e2, j) => {
            if (j % 2 == 0) {
                input.totalBurstTime[i] += e2;
            }
        });
    });
    setAlgorithmNameType(input, selectedAlgorithm.value);
    input.contextSwitch = Number(document.querySelector("#context-
switch").value);
    input.timeQuantum = Number(document.querySelector("#tq").value);
}

function setUtility(input, utility) {
    utility.remainingProcessTime = input.processTime.slice();
    utility.remainingBurstTime = input.totalBurstTime.slice();
    utility.remainingTimeRunning = new Array(process).fill(0);
    utility.currentProcessIndex = new Array(process).fill(0);
    utility.start = new Array(process).fill(false);
    utility.done = new Array(process).fill(false);
    utility.returnTime = input.arrivalTime.slice();
}

function reduceSchedule(schedule) {
    let newSchedule = [];
```

```
    let currentScheduleElement = schedule[0][0];
    let currentScheduleLength = schedule[0][1];
    for (let i = 1; i < schedule.length; i++) {
        if (schedule[i][0] == currentScheduleElement) {
            currentScheduleLength += schedule[i][1];
        } else {
            newSchedule.push([currentScheduleElement, currentScheduleLength]);
            currentScheduleElement = schedule[i][0];
            currentScheduleLength = schedule[i][1];
        }
    }
    newSchedule.push([currentScheduleElement, currentScheduleLength]);
    return newSchedule;
}

function reduceTimeLog(timeLog) {
    let timeLogLength = timeLog.length;
    let newTimeLog = [],
        j = 0;
    for (let i = 0; i < timeLogLength - 1; i++) {
        if (timeLog[i] != timeLog[i + 1]) {
            newTimeLog.push(timeLog[j]);
        }
        j = i + 1;
    }
    if (j == timeLogLength - 1) {
        newTimeLog.push(timeLog[j]);
    }
    return newTimeLog;
}

function outputAverageTimes(output) {
    let avgct = 0;
    output.completionTime.forEach((element) => {
        avgct += element;
    });
    avgct /= process;
    let avgtat = 0;
    output.turnAroundTime.forEach((element) => {
        avgtat += element;
    });
    avgtat /= process;
    let avgwt = 0;
    output.waitingTime.forEach((element) => {
        avgwt += element;
    });
    avgwt /= process;
    let avgrt = 0;
    output.responseTime.forEach((element) => {
        avgrt += element;
```

```
    });
    avgrt /= process;
    return [avgct, avgtat, avgwt, avgrt];
}

function setOutput(input, output) {
    //set turn around time and waiting time
    for (let i = 0; i < process; i++) {
        output.turnAroundTime[i] = output.completionTime[i] -
input.arrivalTime[i];
        output.waitingTime[i] = output.turnAroundTime[i] -
input.totalBurstTime[i];
    }
    output.schedule = reduceSchedule(output.schedule);
    output.timeLog = reduceTimeLog(output.timeLog);
    output.averageTimes = outputAverageTimes(output);
}

function getDate(sec) {
    return (new Date(0, 0, 0, 0, sec / 60, sec % 60));
}

function showGanttChart(output, outputDiv) {
    let ganttChartHeading = document.createElement("h3");
    ganttChartHeading.innerHTML = "Gantt Chart";
    outputDiv.appendChild(ganttChartHeading);
    let ganttChartData = [];
    let startGantt = 0;
    output.schedule.forEach((element) => {
        if (element[0] == -2) { //context switch
            ganttChartData.push([
                "Time",
                "CS",
                "grey",
                getDate(startGantt),
                getDate(startGantt + element[1])
            ]);

        } else if (element[0] == -1) { //nothing
            ganttChartData.push([
                "Time",
                "Empty",
                "black",
                getDate(startGantt),
                getDate(startGantt + element[1])
            ]);

        } else { //process
            ganttChartData.push([
                "Time",
```

```
                "P" + element[0],
                "",
                getDate(startGantt),
                getDate(startGantt + element[1])
            ]);
        }
        startGantt += element[1];
    });
    let ganttChart = document.createElement("div");
    ganttChart.id = "gantt-chart";

    google.charts.load("current", { packages: ["timeline"] });
    google.charts.setOnLoadCallback(drawGanttChart);

    function drawGanttChart() {
        var container = document.getElementById("gantt-chart");
        var chart = new google.visualization.Timeline(container);
        var dataTable = new google.visualization.DataTable();

        dataTable.addColumn({ type: "string", id: "Gantt Chart" });
        dataTable.addColumn({ type: "string", id: "Process" });
        dataTable.addColumn({ type: 'string', id: 'style', role: 'style' });
        dataTable.addColumn({ type: "date", id: "Start" });
        dataTable.addColumn({ type: "date", id: "End" });
        dataTable.addRows(ganttChartData);
        let ganttWidth = '100%';
        if (startGantt >= 20) {
            ganttWidth = 0.05 * startGantt * screen.availWidth;
        }
        var options = {
            width: ganttWidth,
            timeline: {
                showRowLabels: false,
                avoidOverlappingGridLines: false
            }
        };
        chart.draw(dataTable, options);
    }
    outputDiv.appendChild(ganttChart);
}

function showTimelineChart(output, outputDiv) {
    let timelineChartHeading = document.createElement("h3");
    timelineChartHeading.innerHTML = "Timeline Chart";
    outputDiv.appendChild(timelineChartHeading);
    let timelineChartData = [];
    let startTimeline = 0;
    output.schedule.forEach((element) => {
        if (element[0] >= 0) { //process
            timelineChartData.push([
```

```
                "P" + element[0],
                getDate(startTimeline),
                getDate(startTimeline + element[1])
            ]);
        }
        startTimeline += element[1];
    });
    timelineChartData.sort((a, b) => parseInt(a[0].substring(1, a[0].length)) -
parseInt(b[0].substring(1, b[0].length)));
    let timelineChart = document.createElement("div");
    timelineChart.id = "timeline-chart";

    google.charts.load("current", { packages: ["timeline"] });
    google.charts.setOnLoadCallback(drawTimelineChart);

    function drawTimelineChart() {
        var container = document.getElementById("timeline-chart");
        var chart = new google.visualization.Timeline(container);
        var dataTable = new google.visualization.DataTable();

        dataTable.addColumn({ type: "string", id: "Process" });
        dataTable.addColumn({ type: "date", id: "Start" });
        dataTable.addColumn({ type: "date", id: "End" });
        dataTable.addRows(timelineChartData);

        let timelineWidth = '100%';
        if (startTimeline >= 20) {
            timelineWidth = 0.05 * startTimeline * screen.availWidth;
        }
        var options = {
            width: timelineWidth,
        };
        chart.draw(dataTable, options);
    }
    outputDiv.appendChild(timelineChart);
}

function showFinalTable(input, output, outputDiv) {
    let finalTableHeading = document.createElement("h3");
    finalTableHeading.innerHTML = "Final Table";
    outputDiv.appendChild(finalTableHeading);
    let table = document.createElement("table");
    table.classList.add("final-table");
    let thead = table.createTHead();
    let row = thead.insertRow(0);
    let headings = [
        "Process",
        "Arrival Time",
        "Total Burst Time",
        "Completion Time",
```

```
        "Turn Around Time",
        "Waiting Time",
        "Response Time",
    ];
    headings.forEach((element, index) => {
        let cell = row.insertCell(index);
        cell.innerHTML = element;
    });
    let tbody = table.createTBody();
    for (let i = 0; i < process; i++) {
        let row = tbody.insertRow(i);
        let cell = row.insertCell(0);
        cell.innerHTML = "P" + (i + 1);
        cell = row.insertCell(1);
        cell.innerHTML = input.arrivalTime[i];
        cell = row.insertCell(2);
        cell.innerHTML = input.totalBurstTime[i];
        cell = row.insertCell(3);
        cell.innerHTML = output.completionTime[i];
        cell = row.insertCell(4);
        cell.innerHTML = output.turnAroundTime[i];
        cell = row.insertCell(5);
        cell.innerHTML = output.waitingTime[i];
        cell = row.insertCell(6);
        cell.innerHTML = output.responseTime[i];
    }
    outputDiv.appendChild(table);

    let tbt = 0;
    input.totalBurstTime.forEach((element) => (tbt += element));
    let lastct = 0;
    output.completionTime.forEach((element) => (lastct = Math.max(lastct,
element)));

    let cpu = document.createElement("p");
    cpu.innerHTML = "CPU Utilization : " + (tbt / lastct) * 100 + "%";
    outputDiv.appendChild(cpu);

    let tp = document.createElement("p");
    tp.innerHTML = "Throughput : " + process / lastct;
    outputDiv.appendChild(tp);
    if (input.contextSwitch > 0) {

        let cs = document.createElement("p");
        cs.innerHTML = "Number of Context Switches : " + (output.contextSwitches
- 1);
        outputDiv.appendChild(cs);
    }
}
```

```
function toggleTimeLogArrowColor(timeLog, color) {
    let timeLogMove = ['remain-ready', 'ready-running', 'running-terminate',
'running-ready', 'running-block', 'block-ready'];
    timeLog.move.forEach(element => {
        document.getElementById(timeLogMove[element]).style.color = color;
    });
}

function nextTimeLog(timeLog) {
    let timeLogTableDiv = document.getElementById("time-log-table-div");

    let arrowHTML = `
    <p id = "remain-ready" class = "arrow">&rarr;</p>
    <p id = "ready-running" class = "arrow">&#10554;</p>
    <p id = "running-ready" class = "arrow">&#10554;</p>
    <p id = "running-terminate" class = "arrow">&rarr;</p>
    <p id = "running-block" class = "arrow">&rarr;</p>
    <p id = "block-ready" class = "arrow">&rarr;</p>
    `;
    timeLogTableDiv.innerHTML = arrowHTML;

    let remainTable = document.createElement("table");
    remainTable.id = "remain-table";
    remainTable.className = 'time-log-table';
    let remainTableHead = remainTable.createTHead();
    let remainTableHeadRow = remainTableHead.insertRow(0);
    let remainTableHeading = remainTableHeadRow.insertCell(0);
    remainTableHeading.innerHTML = "Remain";
    let remainTableBody = remainTable.createTBody();
    for (let i = 0; i < timeLog.remain.length; i++) {
        let remainTableBodyRow = remainTableBody.insertRow(i);
        let remainTableValue = remainTableBodyRow.insertCell(0);
        remainTableValue.innerHTML = 'P' + (timeLog.remain[i] + 1);
    }
    timeLogTableDiv.appendChild(remainTable);

    let readyTable = document.createElement("table");
    readyTable.id = "ready-table";
    readyTable.className = 'time-log-table';
    let readyTableHead = readyTable.createTHead();
    let readyTableHeadRow = readyTableHead.insertRow(0);
    let readyTableHeading = readyTableHeadRow.insertCell(0);
    readyTableHeading.innerHTML = "Ready";
    let readyTableBody = readyTable.createTBody();
    for (let i = 0; i < timeLog.ready.length; i++) {
        let readyTableBodyRow = readyTableBody.insertRow(i);
        let readyTableValue = readyTableBodyRow.insertCell(0);
        readyTableValue.innerHTML = 'P' + (timeLog.ready[i] + 1);
    }
    timeLogTableDiv.appendChild(readyTable);
```

```
    let runningTable = document.createElement("table");
    runningTable.id = "running-table";
    runningTable.className = 'time-log-table';
    let runningTableHead = runningTable.createTHead();
    let runningTableHeadRow = runningTableHead.insertRow(0);
    let runningTableHeading = runningTableHeadRow.insertCell(0);
    runningTableHeading.innerHTML = "Running";
    let runningTableBody = runningTable.createTBody();
    for (let i = 0; i < timeLog.running.length; i++) {
        let runningTableBodyRow = runningTableBody.insertRow(i);
        let runningTableValue = runningTableBodyRow.insertCell(0);
        runningTableValue.innerHTML = 'P' + (timeLog.running[i] + 1);
    }
    timeLogTableDiv.appendChild(runningTable);

    let blockTable = document.createElement("table");
    blockTable.id = "block-table";
    blockTable.className = 'time-log-table';
    let blockTableHead = blockTable.createTHead();
    let blockTableHeadRow = blockTableHead.insertRow(0);
    let blockTableHeading = blockTableHeadRow.insertCell(0);
    blockTableHeading.innerHTML = "Block";
    let blockTableBody = blockTable.createTBody();
    for (let i = 0; i < timeLog.block.length; i++) {
        let blockTableBodyRow = blockTableBody.insertRow(i);
        let blockTableValue = blockTableBodyRow.insertCell(0);
        blockTableValue.innerHTML = 'P' + (timeLog.block[i] + 1);
    }
    timeLogTableDiv.appendChild(blockTable);

    let terminateTable = document.createElement("table");
    terminateTable.id = "terminate-table";
    terminateTable.className = 'time-log-table';
    let terminateTableHead = terminateTable.createTHead();
    let terminateTableHeadRow = terminateTableHead.insertRow(0);
    let terminateTableHeading = terminateTableHeadRow.insertCell(0);
    terminateTableHeading.innerHTML = "Terminate";
    let terminateTableBody = terminateTable.createTBody();
    for (let i = 0; i < timeLog.terminate.length; i++) {
        let terminateTableBodyRow = terminateTableBody.insertRow(i);
        let terminateTableValue = terminateTableBodyRow.insertCell(0);
        terminateTableValue.innerHTML = 'P' + (timeLog.terminate[i] + 1);
    }
    timeLogTableDiv.appendChild(terminateTable);
    document.getElementById("time-log-time").innerHTML = "Time : " +
timeLog.time;
}

function showTimeLog(output, outputDiv) {
```

```
    reduceTimeLog(output.timeLog);
    let timeLogDiv = document.createElement("div");
    timeLogDiv.id = "time-log-div";
    timeLogDiv.style.height = (15 * process) + 300 + "px";
    let startTimeLogButton = document.createElement("button");
    startTimeLogButton.id = "start-time-log";
    startTimeLogButton.innerHTML = "Start Time Log";
    timeLogDiv.appendChild(startTimeLogButton);
    outputDiv.appendChild(timeLogDiv);

    document.querySelector("#start-time-log").onclick = () => {
        timeLogStart = 1;
        let timeLogDiv = document.getElementById("time-log-div");
        let timeLogOutputDiv = document.createElement("div");
        timeLogOutputDiv.id = "time-log-output-div";

        let timeLogTableDiv = document.createElement("div");
        timeLogTableDiv.id = "time-log-table-div";

        let timeLogTime = document.createElement("p");
        timeLogTime.id = "time-log-time";

        timeLogOutputDiv.appendChild(timeLogTableDiv);
        timeLogOutputDiv.appendChild(timeLogTime);
        timeLogDiv.appendChild(timeLogOutputDiv);
        let index = 0;
        let timeLogInterval = setInterval(() => {
            nextTimeLog(output.timeLog[index]);
            if (index != output.timeLog.length - 1) {
                setTimeout(() => {
                    toggleTimeLogArrowColor(output.timeLog[index], 'red');
                    setTimeout(() => {
                        toggleTimeLogArrowColor(output.timeLog[index], 'black');
                    }, 600);
                }, 200);
            }
            index++;
            if (index == output.timeLog.length) {
                clearInterval(timeLogInterval);
            }
            document.getElementById("calculate").onclick = () => {
                clearInterval(timeLogInterval);
                document.getElementById("time-log-output-div").innerHTML = "";
                calculateOutput();
            }
        }, 1000);
    };
}

function showRoundRobinChart(outputDiv) {
```

```
let roundRobinInput = new Input();
setInput(roundRobinInput);
let maxTimeQuantum = 0;
roundRobinInput.processTime.forEach(processTimeArray => {
    processTimeArray.forEach((time, index) => {
        if (index % 2 == 0) {
            maxTimeQuantum = Math.max(maxTimeQuantum, time);
        }
    });
});
let roundRobinChartData = [
    [],
    [],
    [],
    [],
    []
];
let timeQuantumArray = [];
for (let timeQuantum = 1; timeQuantum <= maxTimeQuantum; timeQuantum++) {
    timeQuantumArray.push(timeQuantum);
    let roundRobinInput = new Input();
    setInput(roundRobinInput);
    setAlgorithmNameType(roundRobinInput, 'rr');
    roundRobinInput.timeQuantum = timeQuantum;
    let roundRobinUtility = new Utility();
    setUtility(roundRobinInput, roundRobinUtility);
    let roundRobinOutput = new Output();
    CPUScheduler(roundRobinInput, roundRobinUtility, roundRobinOutput);
    setOutput(roundRobinInput, roundRobinOutput);
    for (let i = 0; i < 4; i++) {
        roundRobinChartData[i].push(roundRobinOutput.averageTimes[i]);
    }
    roundRobinChartData[4].push(roundRobinOutput.contextSwitches);
}
let roundRobinChartCanvas = document.createElement('canvas');
roundRobinChartCanvas.id = "round-robin-chart";
let roundRobinChartDiv = document.createElement('div');
roundRobinChartDiv.id = "round-robin-chart-div";
roundRobinChartDiv.appendChild(roundRobinChartCanvas);
outputDiv.appendChild(roundRobinChartDiv);

new Chart(document.getElementById('round-robin-chart'), {
    type: 'line',
    data: {
        labels: timeQuantumArray,
        datasets: [{
                label: "Completion Time",
                borderColor: '#3366CC',
                data: roundRobinChartData[0]
            },
```

```
                {
                    label: "Turn Around Time",
                    borderColor: '#DC3912',
                    data: roundRobinChartData[1]
                },
                {
                    label: "Waiting Time",
                    borderColor: '#FF9900',
                    data: roundRobinChartData[2]
                },
                {
                    label: "Response Time",
                    borderColor: '#109618',
                    data: roundRobinChartData[3]
                },
                {
                    label: "Context Switches",
                    borderColor: '#990099',
                    data: roundRobinChartData[4]
                },
            ]
        },
        options: {
            title: {
                display: true,
                text: ['Round Robin', 'Comparison of Completion, Turn Around,
Waiting, Response Time and Context Switches', 'The Lower The Better']
            },
            scales: {
                yAxes: [{
                    ticks: {
                        beginAtZero: true
                    }
                }],
                xAxes: [{
                    scaleLabel: {
                        display: true,
                        labelString: 'Time Quantum'
                    }
                }]
            },
            legend: {
                display: true,
                labels: {
                    fontColor: 'black'
                }
            }
        }
    });
}
```

```
function showAlgorithmChart(outputDiv) {
    let algorithmArray = ["fcfs", "sjf", "srtf", "ljf", "lrtf", "rr", "hrrn",
"pnp", "pp"];
    let algorithmNameArray = ["FCFS", "SJF", "SRTF", "LJF", "LRTF", "RR",
"HRRN", "PNP", "PP"];
    let algorithmChartData = [
        [],
        [],
        [],
        []
    ];
    algorithmArray.forEach(currentAlgorithm => {
        let chartInput = new Input();
        let chartUtility = new Utility();
        let chartOutput = new Output();
        setInput(chartInput);
        setAlgorithmNameType(chartInput, currentAlgorithm);
        setUtility(chartInput, chartUtility);
        CPUScheduler(chartInput, chartUtility, chartOutput);
        setOutput(chartInput, chartOutput);
        for (let i = 0; i < 4; i++) {
            algorithmChartData[i].push(chartOutput.averageTimes[i]);
        }
    });
    let algorithmChartCanvas = document.createElement('canvas');
    algorithmChartCanvas.id = "algorithm-chart";
    let algorithmChartDiv = document.createElement('div');
    algorithmChartDiv.id = "algorithm-chart-div";
    algorithmChartDiv.style.height = "40vh";
    algorithmChartDiv.style.width = "80%";
    algorithmChartDiv.appendChild(algorithmChartCanvas);
    outputDiv.appendChild(algorithmChartDiv);
    new Chart(document.getElementById('algorithm-chart'), {
        type: 'bar',
        data: {
            labels: algorithmNameArray,
            datasets: [{
                    label: "Completion Time",
                    backgroundColor: '#3366CC',
                    data: algorithmChartData[0]
                },
                {
                    label: "Turn Around Time",
                    backgroundColor: '#DC3912',
                    data: algorithmChartData[1]
                },
                {
                    label: "Waiting Time",
```

```
                    backgroundColor: '#FF9900',
                    data: algorithmChartData[2]
                },
                {
                    label: "Response Time",
                    backgroundColor: '#109618',
                    data: algorithmChartData[3]
                }
            ]
        },
        options: {
            title: {
                display: true,
                text: ['Algorithm', 'Comparison of Completion, Turn Around,
Waiting and Response Time', 'The Lower The Better']
            },
            scales: {
                yAxes: [{
                    ticks: {
                        beginAtZero: true
                    }
                }],
                xAxes: [{
                    scaleLabel: {
                        display: true,
                        labelString: 'Algorithms'
                    }
                }]
            },
            legend: {
                display: true,
                labels: {
                    fontColor: 'black'
                }
            }
        }
    });
}

function showOutput(input, output, outputDiv) {
    showGanttChart(output, outputDiv);
    outputDiv.insertAdjacentHTML("beforeend", "<hr>");
    showTimelineChart(output, outputDiv);
    outputDiv.insertAdjacentHTML("beforeend", "<hr>");
    showFinalTable(input, output, outputDiv);
    outputDiv.insertAdjacentHTML("beforeend", "<hr>");
    showTimeLog(output, outputDiv);
    outputDiv.insertAdjacentHTML("beforeend", "<hr>");
    if (selectedAlgorithm.value == "rr") {
        showRoundRobinChart(outputDiv);
```

```
        outputDiv.insertAdjacentHTML("beforeend", "<hr>");
    }
    showAlgorithmChart(outputDiv);
}

function CPUScheduler(input, utility, output) {
    function updateReadyQueue(currentTimeLog) {
        let candidatesRemain = currentTimeLog.remain.filter((element) =>
input.arrivalTime[element] <= currentTimeLog.time);
        if (candidatesRemain.length > 0) {
            currentTimeLog.move.push(0);
        }
        let candidatesBlock = currentTimeLog.block.filter((element) =>
utility.returnTime[element] <= currentTimeLog.time);
        if (candidatesBlock.length > 0) {
            currentTimeLog.move.push(5);
        }
        let candidates = candidatesRemain.concat(candidatesBlock);
        candidates.sort((a, b) => utility.returnTime[a] -
utility.returnTime[b]);
        candidates.forEach(element => {
            moveElement(element, currentTimeLog.remain, currentTimeLog.ready);
            moveElement(element, currentTimeLog.block, currentTimeLog.ready);
        });
        output.timeLog.push(JSON.parse(JSON.stringify(currentTimeLog)));
        currentTimeLog.move = [];
    }

    function moveElement(value, from, to) { //if present in from and not in to
        let index = from.indexOf(value);
        if (index != -1) {
            from.splice(index, 1);
        }
        if (to.indexOf(value) == -1) {
            to.push(value);
        }
    }
    let currentTimeLog = new TimeLog();
    currentTimeLog.remain = input.processId;
    output.timeLog.push(JSON.parse(JSON.stringify(currentTimeLog)));
    currentTimeLog.move = [];
    currentTimeLog.time++;
    let lastFound = -1;
    while (utility.done.some((element) => element == false)) {
        updateReadyQueue(currentTimeLog);
        let found = -1;
        if (currentTimeLog.running.length == 1) {
            found = currentTimeLog.running[0];
        } else if (currentTimeLog.ready.length > 0) {
            if (input.algorithm == 'rr') {
```

```
                found = currentTimeLog.ready[0];
                utility.remainingTimeRunning[found] =
Math.min(utility.remainingProcessTime[found][utility.currentProcessIndex[found]]
, input.timeQuantum);
            } else {
                let candidates = currentTimeLog.ready;
                candidates.sort((a, b) => a - b);
                candidates.sort((a, b) => {
                    switch (input.algorithm) {
                        case 'fcfs':
                            return utility.returnTime[a] -
utility.returnTime[b];
                        case 'sjf':
                        case 'srtf':
                            return utility.remainingBurstTime[a] -
utility.remainingBurstTime[b];
                        case 'ljf':
                        case 'lrtf':
                            return utility.remainingBurstTime[b] -
utility.remainingBurstTime[a];
                        case 'pnp':
                        case 'pp':
                            return priorityPreference * (input.priority[a] -
input.priority[b]);
                        case 'hrrn':
                            function responseRatio(id) {
                                let s = utility.remainingBurstTime[id];
                                let w = currentTimeLog.time -
input.arrivalTime[id] - s;
                                return 1 + w / s;
                            }
                            return responseRatio(b) - responseRatio(a);
                    }
                });
                found = candidates[0];
                if (input.algorithmType == "preemptive" && found >= 0 &&
lastFound >= 0 && found != lastFound) { //context switch
                    output.schedule.push([-2, input.contextSwitch]);
                    for (let i = 0; i < input.contextSwitch; i++,
currentTimeLog.time++) {
                        updateReadyQueue(currentTimeLog);
                    }
                    if (input.contextSwitch > 0) {
                        output.contextSwitches++;
                    }
                }
            }
            moveElement(found, currentTimeLog.ready, currentTimeLog.running);
            currentTimeLog.move.push(1);
            output.timeLog.push(JSON.parse(JSON.stringify(currentTimeLog)));
```

```
            currentTimeLog.move = [];
            if (utility.start[found] == false) {
                utility.start[found] = true;
                output.responseTime[found] = currentTimeLog.time -
input.arrivalTime[found];
            }
        }
        currentTimeLog.time++;
        if (found != -1) {
            output.schedule.push([found + 1, 1]);

utility.remainingProcessTime[found][utility.currentProcessIndex[found]]--;
            utility.remainingBurstTime[found]--;

            if (input.algorithm == 'rr') {
                utility.remainingTimeRunning[found]--;
                if (utility.remainingTimeRunning[found] == 0) {
                    if
(utility.remainingProcessTime[found][utility.currentProcessIndex[found]] == 0) {
                        utility.currentProcessIndex[found]++;
                        if (utility.currentProcessIndex[found] ==
input.processTimeLength[found]) {
                            utility.done[found] = true;
                            output.completionTime[found] = currentTimeLog.time;
                            moveElement(found, currentTimeLog.running,
currentTimeLog.terminate);
                            currentTimeLog.move.push(2);
                        } else {
                            utility.returnTime[found] = currentTimeLog.time +
input.processTime[found][utility.currentProcessIndex[found]];
                            utility.currentProcessIndex[found]++;
                            moveElement(found, currentTimeLog.running,
currentTimeLog.block);
                            currentTimeLog.move.push(4);
                        }

output.timeLog.push(JSON.parse(JSON.stringify(currentTimeLog)));
                        currentTimeLog.move = [];
                        updateReadyQueue(currentTimeLog);
                    } else {
                        updateReadyQueue(currentTimeLog);
                        moveElement(found, currentTimeLog.running,
currentTimeLog.ready);
                        currentTimeLog.move.push(3);

output.timeLog.push(JSON.parse(JSON.stringify(currentTimeLog)));
                        currentTimeLog.move = [];
                    }
                    output.schedule.push([-2, input.contextSwitch]);
                    for (let i = 0; i < input.contextSwitch; i++,
```

```
currentTimeLog.time++) {
                        updateReadyQueue(currentTimeLog);
                    }
                    if (input.contextSwitch > 0) {
                        output.contextSwitches++;
                    }
                }
            } else { //preemptive and non-preemptive
                if
(utility.remainingProcessTime[found][utility.currentProcessIndex[found]] == 0) {
                    utility.currentProcessIndex[found]++;
                    if (utility.currentProcessIndex[found] ==
input.processTimeLength[found]) {
                        utility.done[found] = true;
                        output.completionTime[found] = currentTimeLog.time;
                        moveElement(found, currentTimeLog.running,
currentTimeLog.terminate);
                        currentTimeLog.move.push(2);
                    } else {
                        utility.returnTime[found] = currentTimeLog.time +
input.processTime[found][utility.currentProcessIndex[found]];
                        utility.currentProcessIndex[found]++;
                        moveElement(found, currentTimeLog.running,
currentTimeLog.block);
                        currentTimeLog.move.push(4);
                    }

output.timeLog.push(JSON.parse(JSON.stringify(currentTimeLog)));
                    currentTimeLog.move = [];
                    if (currentTimeLog.running.length == 0) { //context switch
                        output.schedule.push([-2, input.contextSwitch]);
                        for (let i = 0; i < input.contextSwitch; i++,
currentTimeLog.time++) {
                            updateReadyQueue(currentTimeLog);
                        }
                        if (input.contextSwitch > 0) {
                            output.contextSwitches++;
                        }
                    }
                    lastFound = -1;
                } else if (input.algorithmType == "preemptive") {
                    moveElement(found, currentTimeLog.running,
currentTimeLog.ready);
                    currentTimeLog.move.push(3);

output.timeLog.push(JSON.parse(JSON.stringify(currentTimeLog)));
                    currentTimeLog.move = [];
                    lastFound = found;
                }
            }
```

```
        } else {
            output.schedule.push([-1, 1]);
            lastFound = -1;
        }
        output.timeLog.push(JSON.parse(JSON.stringify(currentTimeLog)));
    }
    output.schedule.pop();
}

function calculateOutput() {
    let outputDiv = document.getElementById("output");
    outputDiv.innerHTML = "";
    let mainInput = new Input();
    let mainUtility = new Utility();
    let mainOutput = new Output();
    setInput(mainInput);
    setUtility(mainInput, mainUtility);
    CPUScheduler(mainInput, mainUtility, mainOutput);
    setOutput(mainInput, mainOutput);
    showOutput(mainInput, mainOutput, outputDiv);
}

document.getElementById("calculate").onclick = () => {
    calculateOutput();
};
```

Styles.css

```
* {
    margin: 0;
    padding: 0;
    font-size: 17px;
}

body {
    margin: 10px 20px;
}

.instructions {
    font-size: 14px;
    margin: 10px 20px;
}

h1 {
    font-size: 26px;
}
```

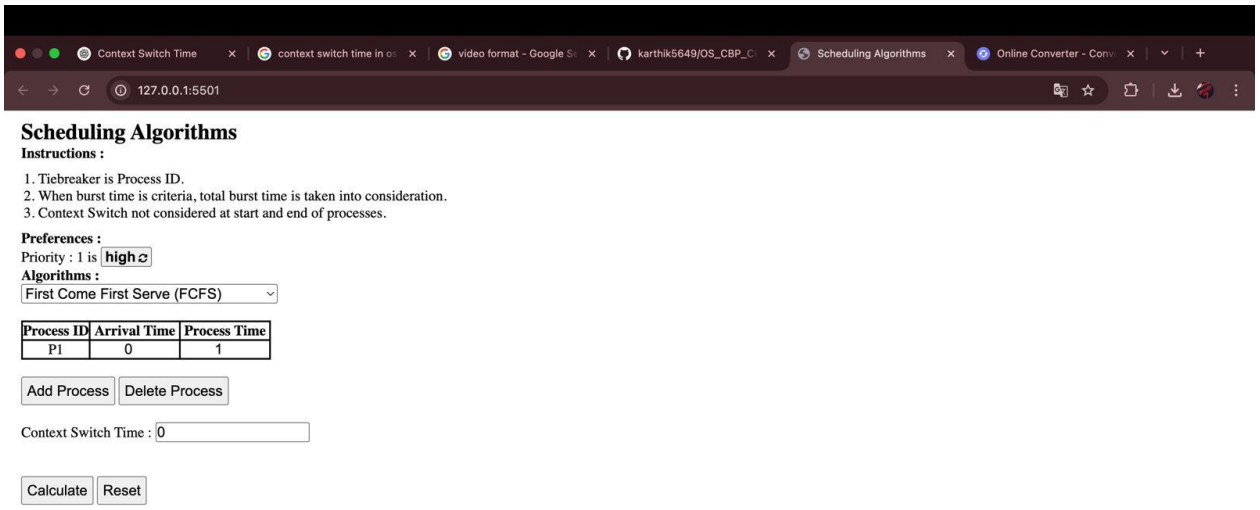
```
table,
th,
td {
    border-collapse: collapse;
    text-align: center;
    border: 2px solid black;
}

td {
    padding: 0 2px;
}

table input {
    text-align: center;
    border: none;
    width: 100px;
}

.process-btn {
    height: 5px;
}

.process-btn button {
    height: 20px;
    width: 20px;
    font-size: 16px;
    background-color: transparent;
    border: none;
    font-weight: bolder;
}

.hide {
    display: none;
}

button {
    padding: 5px;
}

.schedule-table-process,
.schedule-table-process td {
    border: 1px solid black;
}

.schedule-table-process {
    margin-left: 6px;
}

.schedule-table-time,
```

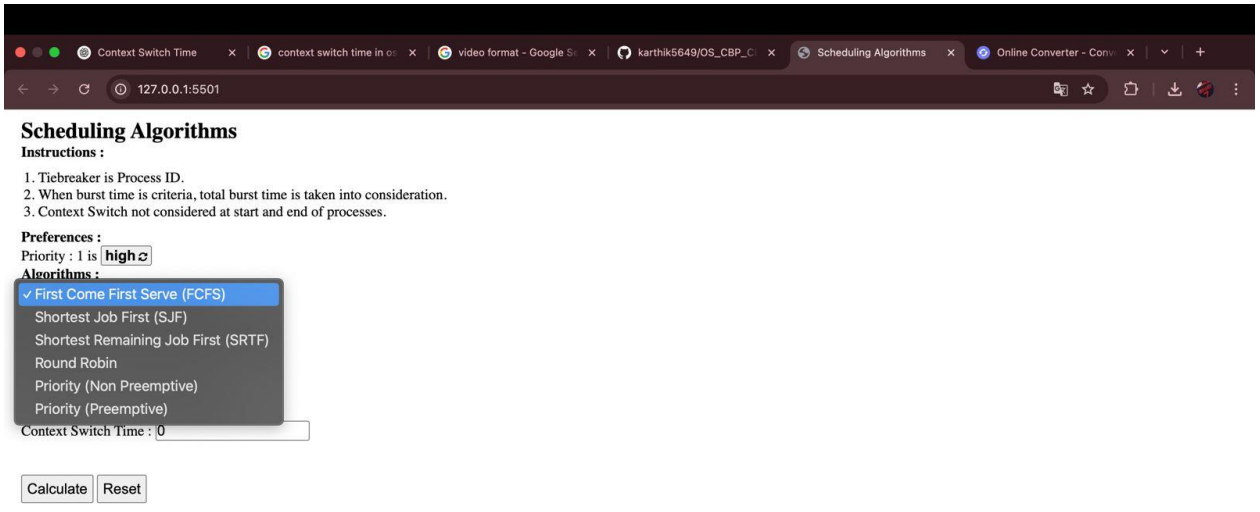
```
.schedule-table-time td {
    text-align: left;
    border: 1px solid white;
}

#algorithm-chart-div {
    width: 80%;
    height: 50%;
}

#round-robin-chart-div {
    height: 40%;
    width: 80%;
}

#gantt-chart {
    height: 250px;
    overflow-x: scroll;
    overflow-y: hidden;
}

#timeline-chart {
    overflow-x: scroll;
    overflow-y: hidden;
    height: 250px;
}

#output {
    margin-top: 20px;
}

#priority-toggle-btn {
    font-size: 12px;
    font-weight: bold;
    padding: 0 0;
    width: 60px;
}

.preferences {
    list-style: none;
}

thead {
    font-weight: bold;
}

#time-log-table-div {
    margin-top: 10vh;
    position: relative;
}
```

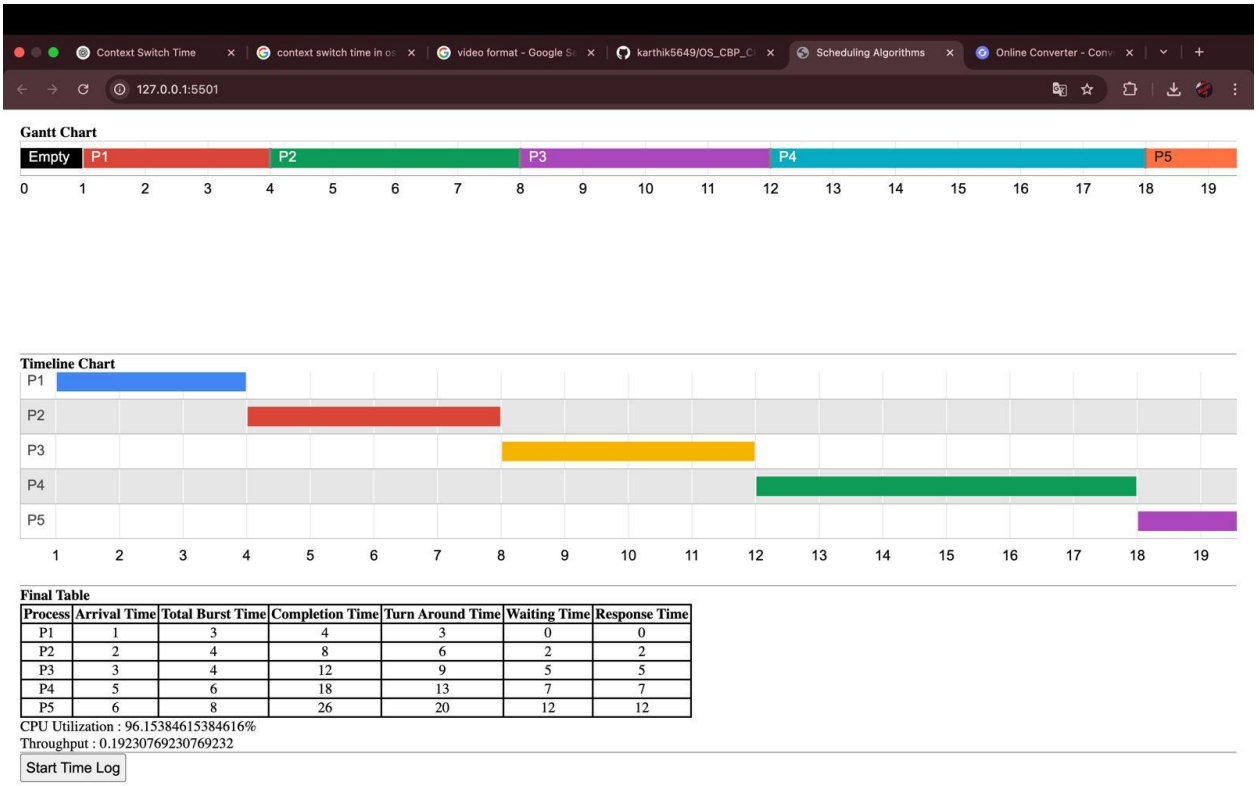
```
.time-log-table {
    width: 10%;
    position: absolute;
}

#remain-table {
    left: 7.5%;
}

#ready-table {
    left: 32.5%;
}

#running-table {
    right: 32.5%;
}

#block-table {
    left: 45%;
    top: 8vw;
}

#terminate-table {
    right: 7.5%;
}

.arrow {
    position: absolute;
    font-size: 100px;
}

#remain-ready {
    left: 20.5%;
    top: -62px;
}

#ready-running {
    transform: rotate(180deg);
    left: 47%;
    top: -15px;
    font-size: 80px;
}

#running-ready {
    left: 47%;
    top: -60px;
    font-size: 80px;
}
```

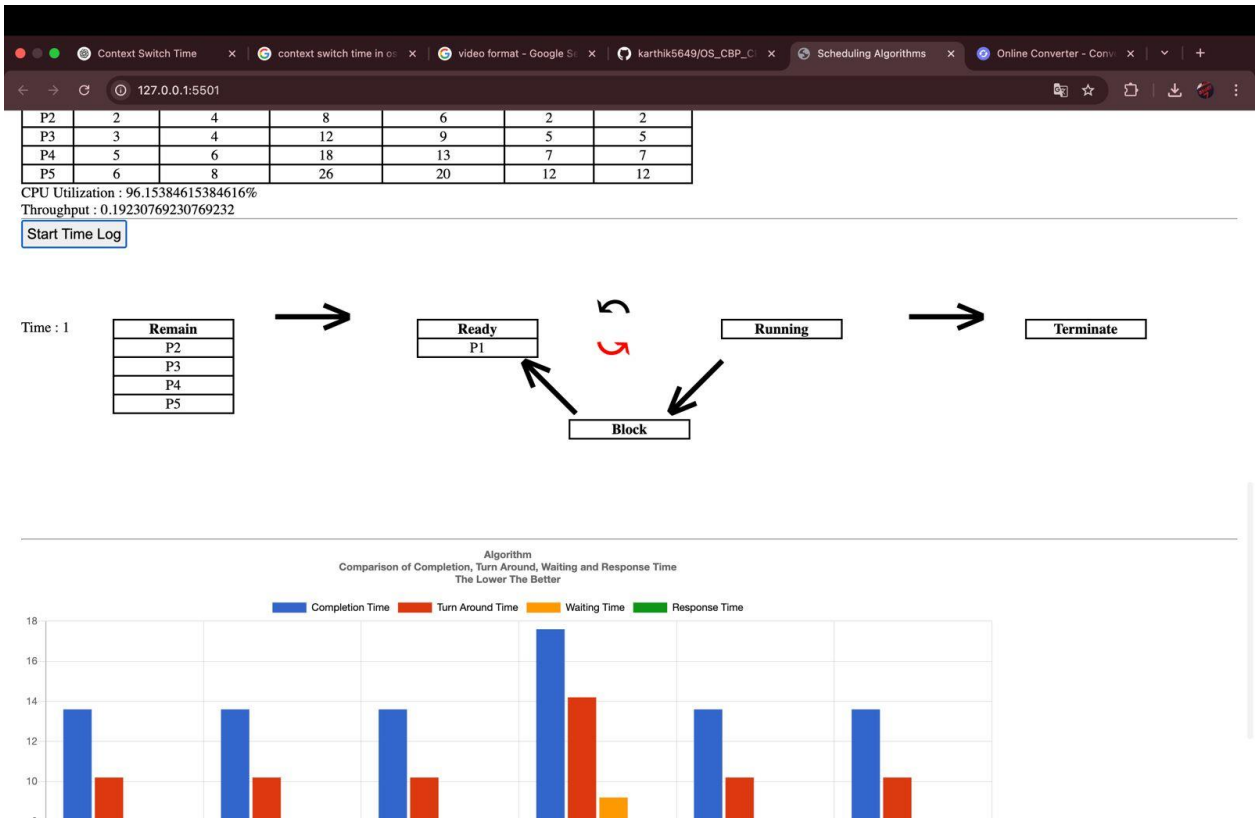
```
#running-terminate {
    right: 20.5%;
    top: -62px;
}

#running-block {
    transform: rotate(135deg);
    left: 52%;
    top: 20px;
}

#block-ready {
    transform: rotate(225deg);
    left: 40%;
    top: 20px;
}
```

Output:


Context Switch Time
context switch time in os
video format - Google Se
karthik5649/OS_CBP_C
Scheduling Algorithms
Online Converter - Conv
127.0.0.1:5501
Scheduling Algorithms
Instructions :
1. Tiebreaker is Process ID.
2. When burst time is criteria, total burst time is taken into consideration.
3. Context Switch not considered at start and end of processes.
Preferences :
Priority : 1 is high
Algorithms :
First Come First Serve (FCFS)
Process ID
Arrival Time
Process Time
P1
0
1
Add Process
Delete Process
Context Switch Time : 0
Calculate
Reset


Context Switch Time
context switch time in os
video format - Google Se
karthik5649/OS_CBP_C
Scheduling Algorithms
Online Converter - Conv
127.0.0.1:5501
Scheduling Algorithms
Instructions :
1. Tiebreaker is Process ID.
2. When burst time is criteria, total burst time is taken into consideration.
3. Context Switch not considered at start and end of processes.
Preferences :
Priority : 1 is high
Algorithms :
First Come First Serve (FCFS)
Shortest Job First (SJF)
Shortest Remaining Job First (SRTF)
Round Robin
Priority (Non Preemptive)
Priority (Preemptive)
Context Switch Time : 0
Calculate
Reset


Context Switch Time
context switch time in os
video format - Google Se
karthik5649/OS_CBP_C
Scheduling Algorithms
Online Converter - Conv
127.0.0.1:5501
Gantt Chart
Empty
P1
P2
P3
P4
P5
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Timeline Chart
P1
P2
P3
P4
P5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Final Table
Process | Arrival Time | Total Burst Time | Completion Time | Turn Around Time | Waiting Time | Response Time
P1 | 1 | 3 | 4 | 3 | 0 | 0
P2 | 2 | 4 | 8 | 6 | 2 | 2
P3 | 3 | 4 | 12 | 9 | 5 | 5
P4 | 5 | 6 | 18 | 13 | 7 | 7
P5 | 6 | 8 | 26 | 20 | 12 | 12
CPU Utilization : 96.153846153846 16%
Throughput : 0.19230769230769232
Start Time Log


Context Switch Time
context switch time in os
video format - Google Se
karthik5649/OS_CBP_C
Scheduling Algorithms
Online Converter - Conv
127.0.0.1:5501
P2 2 4 8 6 2 2
P3 3 4 12 9 5 5
P4 5 6 18 13 7 7
P5 6 8 26 20 12 12
CPU Utilization : 96.153846153846 16%
Throughput : 0.19230769230769232
Start Time Log
Time : 1
Remain
P2
P3
P4
P5
Ready
P1
Running
Terminate
Block
Algorithm
Comparison of Completion, Turn Around, Waiting and Response Time
The Lower The Better
Completion Time
Turn Around Time
Waiting Time
Response Time
18
16
14
12
10

CONCLUSION

In conclusion, the CPU scheduling algorithms implemented in this project represent a critical component of efficient process management in modern computing systems. The project explores various scheduling techniques—including First-Come-First-Serve (FCFS), Shortest Job First (SJF), Round Robin (RR), and Priority Scheduling—to address key performance metrics such as turnaround time, waiting time, and CPU utilization.

By simulating these algorithms, the project demonstrates how effective scheduling decisions can significantly enhance the overall performance of an operating system. Each algorithm serves specific scenarios: FCFS ensures fairness by serving processes in order of arrival, SJF optimizes average waiting time by prioritizing shorter tasks, Round Robin offers time-shared multitasking suitable for interactive systems, and Priority Scheduling allows critical tasks to be served ahead of less urgent ones.

The use of structured programming and dynamic data structures like arrays and process records has enabled precise tracking of each process's lifecycle—from arrival and execution to completion. This systematic approach ensures accurate computation of essential performance metrics, which are crucial for evaluating the efficiency and fairness of different scheduling strategies.

Furthermore, the project highlights the trade-offs involved in scheduling policy selection. While some algorithms reduce waiting time, others ensure fairness or responsiveness. The ability to simulate and compare these trade-offs equips developers, system architects, and researchers with practical tools for selecting the most appropriate strategy based on application needs.
The iterative coding and testing process also reinforced the importance of modularity and error handling in system-level programs. By breaking down the scheduling logic into clear functional components, the program not only became easier to debug and maintain but also more scalable for future enhancements, such as incorporating multi-core scheduling or I/O-bound task handling.

As computing environments continue to evolve—toward more real-time, high-performance, and parallel-processing systems—the relevance of optimized CPU scheduling becomes even more vital. The work presented in this project lays a strong foundation for exploring advanced topics such as preemptive multitasking, hybrid scheduling, and real-time system optimization.
Ultimately, this project serves as a practical and educational tool for understanding the core principles of operating system design and performance optimization. It prepares students and developers to contribute meaningfully to the development of responsive, efficient, and fair computing environments.

FUTURISTIC SCOPE

Looking ahead, the CPU Scheduling Algorithms implemented in this project present numerous possibilities for advancement in both academic research and real-world applications. As computing environments evolve to meet the demands of scalability, real-time processing, and energy efficiency, scheduling mechanisms must also adapt to ensure optimal performance. Below are several forward-looking directions for the future development of this project:

1. Multicore and Parallel Processing Support:
Future extensions can include scheduling algorithms specifically designed for multicore and multi-threaded systems. This would involve efficient allocation of tasks across cores, load balancing, and synchronization handling to optimize throughput and minimize latency in parallel computing environments.

2. Real-Time Operating System (RTOS) Integration:
Adapting the scheduling algorithms for use in real-time systems—such as embedded devices, medical equipment, and automotive systems—can ensure deterministic behavior and timely task completion. Support for hard and soft deadlines could be incorporated to extend the usability of the system.

3. Machine Learning-Based Scheduling:
By integrating AI and machine learning, the system could learn from historical data to dynamically select the most efficient scheduling algorithm based on workload patterns, process behavior, or system performance indicators. This would create an adaptive scheduler capable of self-optimization.

4. Energy-Efficient Scheduling:
Future iterations can explore energy-aware scheduling strategies that reduce power consumption without compromising performance. This is especially relevant in mobile devices and data centers, where energy efficiency is a critical metric.

5. Cloud and Distributed System Compatibility:
Expanding the project to support distributed environments and cloud-based architectures would involve scheduling processes across virtual machines or containers. This opens up avenues for integration with Kubernetes, Docker Swarm, or other orchestration platforms.

6. Interactive Simulation and Visualization Tools:
To enhance learning and research utility, a graphical interface with real-time simulation of process scheduling, Gantt chart generation, and performance analytics can be developed. This will aid students, educators, and developers in visualizing complex scheduling behavior.

7. Dynamic Scheduling in OS Kernels:
The project can be extended into a real or simulated kernel module where dynamic task scheduling is implemented at the OS level. This would allow testing of algorithms under real-world constraints like I/O interrupts, shared memory, and varying load conditions.

8. Customizable Algorithm Framework:
A modular framework could be developed to allow users or researchers to define their own scheduling

REFERENCES

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts (10th Edition). Wiley.

→ Comprehensive coverage of CPU scheduling algorithms, process management, and performance metrics.

2. Stallings, W. (2018). Operating Systems: Internals and Design Principles (9th Edition). Pearson.

→ In-depth discussion on scheduling criteria, types of schedulers, and real-time systems.

3. Tanenbaum, A. S., & Bos, H. (2015). Modern Operating Systems (4th Edition). Pearson.

→ Provides theoretical and practical explanations of different scheduling policies.

4. GeeksforGeeks. (n.d.). CPU Scheduling Algorithms. Retrieved from:

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

5. TutorialsPoint. (n.d.). CPU Scheduling. Retrieved from:

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

6. Programiz. (n.d.). CPU Scheduling Algorithms with Examples. Retrieved from:

https://www.programiz.com/operating-system/cpu-scheduling

7. W3Schools. (n.d.). HTML, CSS, and JavaScript References. Retrieved from:

https://www.w3schools.com/

8. MDN Web Docs (Mozilla Developer Network). (n.d.). JavaScript and Web APIs. Retrieved from:

https://developer.mozilla.org/

9. Chart.js. (n.d.). Documentation. Retrieved from:

https://www.chartjs.org/docs/latest/

10. Google Charts. (n.d.). Google Chart Tools. Retrieved from:

https://developers.google.com/chart