# CPU SCHEDULING VISUALIZER

A PROJECT REPORT

*Submitted by*

**AKANKSHA MISHRA [RA2211003011513]**
**ADITYA ROY [RA2211003011539]**
**APARIJIT CHAKRABORTY [RA2211003011540]**

*Under the Guidance of*

**DR. ARUNA M.**

Associate Professor, Department of Computing Technology Systems

*In partial fulfilment of the requirements for thedegree of*

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE AND ENGINEERING**



**DEPARTMENT OF COMPUTING TECHNOLOGY SYSTEMS**

**COLLEGE OF ENGINEERING AND TECHNOLOGY**

**SRM  INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR – 603 203**

**NOVEMBER 2023**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR – 603 203**

**BONAFIDE CERTIFICATE**

Certified that this B.Tech project report titled "**CPU SCHEDULING VISUALIZER**" is the bonafide work of **Ms. Akanksha Mishra [Reg. No.: RA2211003011513], Mr. Aditya Roy [Reg. No.RA2211003011539]** and **Aparijit Chakraborty [Reg. No. RA221100301140]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation onthe basis of which a degree or award was conferred on an earlier occasion for this or any othercandidate.

**SIGNATURE OF
EXAMINER**

**DR. ARUNA
M.
SUPERVISOR**
Associate Professor
Department of Computing
Technology Systems

# Department of Computing Technology  Systems

# SRM Institute of Science and TechnologyOwn Work Declaration Form

**Degree/ Course :** B.Tech in Computer Science and Engineering

**Student Names :** Akanksha Mishra, Aditya Roy, Aparijit Chakraborty.

**Registration Number:** RA2211003011513. RA2211003011539, RA2211003011540.

**Title of Work :** CPU Scheduling Visualizer.

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

• Clearly references / listed all sources as appropriate

• Referenced and put in inverted commas all quoted text (from books, web,  etc.)

• Given the sources of all pictures, data etc. that are not my own
• Not made any use of the report(s) or essay(s) of any other student(s) eitherpast or present

- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources) • Compiled with any other plagiarism criteria specified in the Course handbook / University website

   I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

| DECLARATION: |
| --- |
| I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated  by referring, and that I have followed the good academic practices noted above. |
| If you are working in a group, please write your registration numbers and sign with the date for every student in your group. |

# ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. Muthamizhchelvan**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman,** for imparting confidence to complete my course project

We wish to express my sincere thanks to **Course Audit Professors Dr. Vadivu. G , Professor, Department of Data Science and Business Systems and Dr. Sasikala. E Professor, Department of Data Science and Business Systems** and **Course Coordinators** for their constant encouragement and support.

We are highly thankful to our Course project **Dr. Aruna M, Assistant Professor, Department of Computing, SRMIST ,** for her assistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to our **HOD, Dr. Pushpalatha, Professor & Head, Department Of Computing Technology, SRMIST.**

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

# INDEX

# 1.ABSTRACT

This project report delves into the development and implementation of a CPU Scheduling Visualizer, aimed at providing a comprehensive understanding of various CPU scheduling algorithms. The realm of operating systems and computer science often grapples with the challenge of efficiently managing multiple processes competing for a system's central processing unit. The report details the development process, methodology, and insights gained from the creation of a hands-on visual tool designed to explore and interact with different CPU scheduling algorithms.

The primary focus of this visualizer is to elucidate the functionality and comparative performance of prominent scheduling algorithms, including First-Come-First-Serve (FCFS), Shortest Job First (SJF), Priority Scheduling, and Round Robin (RR). Each algorithm's behavior and impact on system performance are visually represented, allowing users to comprehend their functionality, strengths, and limitations.

The report outlines the meticulous methodology employed in the creation of the visualizer:

**Requirements Analysis:** A comprehensive study and documentation of the requirements for the simulator, emphasizing the need for clear conceptual understanding and practical implementation of CPU scheduling algorithms.

**Design:** Detailed architectural schematics, user interface layout, and algorithmic implementation plans, ensuring a coherent and effective structure for the visualizer.

**Implementation:** Utilization of HTML, CSS, and JavaScript for the development of the interactive visual tool, enabling users to actively engage with and comprehend the workings of different CPU scheduling algorithms.

**Testing:** Rigorous testing procedures were conducted to validate the functionality and user-friendliness of the visualizer. Multiple iterations of testing ensured a smooth and accurate representation of the CPU scheduling algorithms.

Throughout the report, each scheduling algorithm is examined in detail. FCFS, as the most straightforward method, operates on the principle of process arrival order. Additionally, the report covers the advantages and challenges posed by SJF, Priority Scheduling, and RR. SJF's focus on minimizing waiting time, Priority Scheduling's emphasis on process prioritization, and RR's balanced resource allocation are highlighted, along with their practical implications and limitations.

The CPU Scheduling Visualizer project aims to be an invaluable educational resource for students, educators, and system administrators. By providing an interactive and visual platform to explore CPU scheduling algorithms, this report emphasizes a practical understanding of these fundamental concepts, allowing users to gain insights into their real-world implications and optimize system performance.

# 2.INTRODUCTION

In the realm of computer science and operating systems, the efficient management of a computer's central processing unit (CPU) is a fundamental concern. CPU scheduling algorithms dictate how the CPU handles the execution of multiple processes, significantly impacting system performance and user experience. The concept of scheduling algorithms lies at the core of optimizing resource utilization and ensuring timely task execution.

This project delves into the intricate world of CPU scheduling through the lens of a visualizer. The primary objective is to create an interactive tool that allows users to explore and comprehend the functionality of various CPU scheduling algorithms. The focus primarily rests on elucidating the operational dynamics of scheduling methods such as First-Come-First-Serve (FCFS), Shortest Job First (SJF), Priority Scheduling, and Round Robin (RR).

CPU scheduling is a critical aspect of system operation, and this project endeavors to bridge the gap between theoretical understanding and practical implementation. By offering a hands-on experience with scheduling algorithms, users, including students, educators, and system administrators, can gain insight into how different algorithms function and their real-world implications.

This report presents the journey of conceptualizing, designing, and implementing a CPU scheduling visualizer. It outlines the development process, methodology, and the specific algorithms incorporated into the visualizer. Through this project, a comprehensive and interactive learning experience is offered, enabling a deeper comprehension of CPU scheduling principles and their role in system performance optimization. The report details the architecture, design choices, implementation strategies, and the significance of each scheduling algorithm incorporated into the visualizer.

By employing a combination of HTML, CSS, and JavaScript, the visualizer provides an engaging platform for users to interact with, fostering a practical understanding of scheduling methodologies. Each algorithm's strengths, limitations, and real-world applicability are explored, aiming to provide a holistic view of CPU scheduling and its impact on system functionality.

This report serves as a comprehensive guide to the CPU scheduling visualizer project, detailing its purpose, development process, and the educational value it offers in comprehending the intricate world of CPU scheduling algorithms through interactive visualization.

# 3.PROJECT OVERVIEW

The "CPU Scheduling Visualizer" project is an educational tool designed to explore, visualize, and interact with various CPU scheduling algorithms. Its primary objective is to offer a comprehensive understanding of the functioning and implications of different scheduling methodologies through an interactive and user-friendly platform.

The project encompasses a wide array of CPU scheduling algorithms, with a primary focus on First-Come-First-Serve (FCFS), complemented by in-depth exploration and visualization of Shortest Job First (SJF), Priority Scheduling, and Round Robin (RR) algorithms. Users are provided with a hands-on experience to comprehend the impact of these algorithms on system performance and user experience.

## Objective:

The core goal of this project is to provide a practical and visual means for individuals, including students, educators, and system administrators, to:

- Understand the foundational principles of CPU scheduling.
- Gain insights into the mechanisms and nuances of various scheduling algorithms.
- Comprehend the real-world implications and trade-offs associated with each scheduling methodology.

## Key Features:

The CPU Scheduling Visualizer boasts the following key features:

1. Interactive Visualization: Users can actively engage with the visualization platform, adding, managing, and observing the execution of processes using different scheduling algorithms.

2. Algorithmic Exploration: In-depth coverage of multiple scheduling algorithms, showcasing their strengths, limitations, and practical implications.

3. Real-time Demonstration: Practical demonstrations illustrating how CPU

scheduling affects system performance, highlighting scenarios where specific algorithms excel or face challenges.

4. User-Friendly Interface: A sleek and intuitive user interface that allows seamless interaction and understanding of complex scheduling concepts.

**Methodology:**

The project has been meticulously developed through a structured methodology:

- Requirements Analysis: Clear documentation of the simulator's requirements, ensuring a solid foundation for the project.

- Design: Detailed planning and architecture for the simulator, encompassing user interface design and algorithmic implementations.

- Implementation: Utilization of HTML, CSS, and JavaScript to bring the simulator to life, enabling users to actively engage with and visualize CPU scheduling algorithms.

- Testing: Rigorous testing procedures to validate the simulator's functionality, resulting in a polished, user-friendly interface.

# 4.REQUIERMENTS

1. System Architecture
   - Detailed description of the architecture of the visualizer, including its components and how they interact.
   - Illustrations or diagrams, if applicable, to depict the system's architecture.
2. Algorithms Covered
   - Individual sections for each scheduling algorithm (FCFS, SJF, Priority Scheduling, RR) with the following details:
   - Explanation of the algorithm, its core principles, and its impact on system performance.
   - How the algorithm is implemented within the visualizer.
   - Strengths, weaknesses, and real-world implications of each algorithm.
3. User Interface
   - Description of the user interface of the visualizer.
   - Screenshots or diagrams to illustrate the interface design and user interaction.
   - Explanation of how users can interact with the visualizer and explore different scheduling algorithms.
4. Functionality and Features
   - Detailed description of the functionalities offered by the visualizer for each scheduling algorithm.
   - How users can add processes to the queue, visualize their execution, and observe the algorithm's behaviour.
5. User Manual/Guide
   - A comprehensive guide on how to use the visualizer, explaining its features and functionalities step by step.
   - Instructions on how to interpret the visualizations and understand the implications of different scheduling algorithms.
6. Results and Findings
   - Feedback or insights gained through testing and user interactions with the visualizer.
   - Observations on the understanding and practical implications of different scheduling algorithms by users.

# 5.INTERFACE & DESIGNE

## FCFS Output:

Algorithm:
- ● FCFS
- ○ SJF
- ○ Priority scheduling
- ○ Round Robin

| Process | Arrival Time | Execute Time | Service Time |
|---------|--------------|--------------|--------------|
| P0 | 0 | 5 | 0 |
| P1 | 1 | 3 | 5 |
| P2 | 2 | 5 | 8 |
| P3 | 3 | 8 | 13 |

+

Go

| P0 | P1 | P2 | P3 |
|----|----|----|----|
| 5 | 3 | 5 | 8 |

Timer: **21** sec

Fig: 5.1 (FCFS)

## SJF Output:

Algorithm:
- ○ FCFS
- ● SJF
- ○ Priority scheduling
- ○ Round Robin

| Process | Arrival Time | Execute Time | Service Time |
|---------|--------------|--------------|--------------|
| P0 | 0 | 5 | 5 |
| P1 | 1 | 3 | 2 |
| P2 | 2 | 2 | 0 |
| P3 | 3 | 8 | 10 |

+

Go

| P2 | P1 | P0 | P3 |
|----|----|----|----|
| 2 | 3 | 5 | 8 |

Timer: **18** sec

Fig: 5.2 (SJF)

## Priority Scheduling Output:

Algorithm:
- ○ FCFS
- ○ SJF
- ● Priority scheduling
- ○ Round Robin

| Process | Arrival Time | Execute Time | Service Time | Priority |
|---------|--------------|--------------|--------------|----------|
| P0 | 0 | 5 | 13 | 10 |
| P1 | 1 | 3 | 0 | 50 |
| P2 | 2 | 2 | 11 | 20 |
| P3 | 3 | 8 | 3 | 30 |

[ - ]
[ + ]
[ Go ]

| P1 | P3 | P2 | P0 |
|----|----|----|----|
| 3 | 8 | 2 | 5 |

Timer: **18** sec

Fig: 5.3 (Priority Scheduling)

## Round Robin Output:

Algorithm:
- ○ FCFS
- ○ SJF
- ○ Priority scheduling
- ● Round Robin

| Process | Arrival Time | Execute Time |
|---------|--------------|--------------|
| P0 | 0 | 5 |
| P1 | 1 | 3 |
| P2 | 2 | 2 |
| P3 | 3 | 8 |

[ - ]
[ + ]

Quantum: [ 3 ]

[ Go ]

| P0 | P1 | P2 | P3 | P0 | P3 | P3 |
|----|----|----|----|----|----|----|
| 3 | 3 | 2 | 3 | 2 | 3 | 2 |

Timer: **18** sec

Fig: 5.4 (Round Robin)

# 6.CODES

## ❖ HTML:

```html
<div>
 <div>
  Algorithm:
  <form id="algorithm">
   <input type="radio" name="algorithm" value="fcfs" checked/> FCFS<br/>
   <input type="radio" name="algorithm" value="sjf"/> SJF<br/>
   <input type="radio" name="algorithm" value="priority"/> Priority
scheduling<br/>
   <input type="radio" name="algorithm" value="robin"/> Round Robin
  </form>
  <br>
 </div>
  <table id="inputTable">
   <tr>
     <th>Process</th>
     <th>Arrival Time</th>
     <th>Execute Time</th>
     <th class="servtime">Service Time</th>
     <th class="priority-only">Priority</th>
   </tr>
   <tr>
     <td>P0</td>
     <td>0</td>
     <td><input class="initial exectime" type="text" value="5" /></td>
     <td class="servtime"></td>
     <td class="priority-only initial"><input type="text"/></td>
   </tr>
   <tr>
     <td>P1</td>
     <td>1</td>
     <td><input class="initial exectime" value="3" /></td>
```
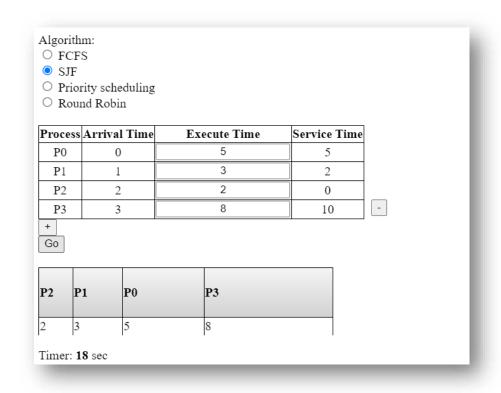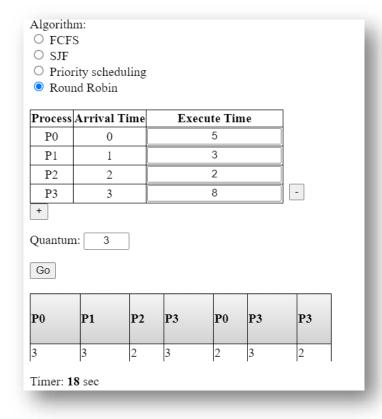
```html
            <td class="servtime"></td>
            <td class="priority-only" ><input type="text"/></td>
        </tr>
    </table>
    <input id="minus" style="display: inline; left: 428px; position: absolute; top:
170px;" type="button" value="-" onclick="deleteRow();" />
    </div>
    <input type="button" value="+" onclick="addRow();" />
<div>
 </div>
    <p id="quantumParagraph" hidden>Quantum:
        <input style="width: 50px;" id="quantum" type="text" value="3" />
    </p>
    <input type="button" value="Go" onclick="draw();" />
    <br/><br/>

    <fresh>
    </fresh>
    <p>
    Timer: <strong id="timer"></strong> sec
    </p>
```

❖ **CSS:**

```css
    fresh th {
        background: linear-gradient(WhiteSmoke, lightgray);
        text-align: left;
        border: 1px solid black;
        height: 400 px;
    }
    fresh td {
        border: 1px solid black;
        border-bottom: 0;
        /*border-right: 0;*/
        height: 20px;
```

```css
        text-align: left;
    }

    div th, div td {
        border: 1px solid black;
        text-align: center;
    }
    table {
        display: inline;
        border-collapse: collapse;
    }
    input {
        text-align: center;
    }
    #curtain {
      background-color: white;
    }
```

## ❖ JS:

```javascript
recalculateServiceTime();
$('.priority-only').hide();

$(document).ready(function () {
  $('input[type=radio][name=algorithm]').change(function () {
   if (this.value == 'priority') {
     $('.priority-only').show();
     $('.servtime').show();
     $('#minus').css('left', '604px');
   }
   else {
     $('.priority-only').hide();
     $('.servtime').show();
     $('#minus').css('left', '428px');
   }

   if (this.value == 'robin') {
     $('.servtime').hide();
     $('#quantumParagraph').show();
   }
```

```javascript
    else {
      $('#quantumParagraph').hide();
      $('.servtime').show();
    }

    recalculateServiceTime();
  });
});

function addRow() {
  var lastRow = $('#inputTable tr:last');
  var lastRowNumebr = parseInt(lastRow.children()[1].innerText);

  var newRow = '<tr><td>P'
  + (lastRowNumebr + 1)
  + '</td><td>'
  + (lastRowNumebr + 1)
  + '</td><td><input class="exectime" type="text"/></td><td class="servtime"></td>'
  //if ($('input[name=algorithm]:checked', '#algorithm').val() == "priority")
  + '<td class="priority-only"><input type="text"/></td></tr>';

  lastRow.after(newRow);

  var minus = $('#minus');
  minus.show();
  minus.css('top', (parseFloat(minus.css('top')) + 24) + 'px');

  if ($('input[name=algorithm]:checked', '#algorithm').val() != "priority")
    $('.priority-only').hide();


  $('#inputTable tr:last input').change(function () {
    recalculateServiceTime();
  });
}

function deleteRow() {
  var lastRow = $('#inputTable tr:last');
  lastRow.remove();

  var minus = $('#minus');
  minus.css('top', (parseFloat(minus.css('top')) - 24) + 'px');

  if (parseFloat(minus.css('top')) < 150)
    minus.hide();
}
```

```
$(".initial").change(function () {
  recalculateServiceTime();
});

function recalculateServiceTime() {
  var inputTable = $('#inputTable tr');
  var totalExectuteTime = 0;

  var algorithm = $('input[name=algorithm]:checked', '#algorithm').val();
  if (algorithm == "fcfs") {
    $.each(inputTable, function (key, value) {
      if (key == 0) return true;
      $(value.children[3]).text(totalExectuteTime);

      var executeTime = parseInt($(value.children[2]).children().first().val());
      totalExectuteTime += executeTime;
    });
  }
  else if (algorithm == "sjf") {
    var exectuteTimes = [];
    $.each(inputTable, function (key, value) {
      if (key == 0) return true;
      exectuteTimes[key - 1] = parseInt($(value.children[2]).children().first().val());
    });

    var currentIndex = -1;
    for (var i = 0; i < exectuteTimes.length; i++) {
      currentIndex = findNextIndex(currentIndex, exectuteTimes);

      if (currentIndex == -1) return;

      $(inputTable[currentIndex + 1].children[3]).text(totalExectuteTime);

      totalExectuteTime += exectuteTimes[currentIndex];
    }
  }
  else if (algorithm == "priority") {
    var exectuteTimes = [];
    var priorities = [];

    $.each(inputTable, function (key, value) {
      if (key == 0) return true;
      exectuteTimes[key - 1] = parseInt($(value.children[2]).children().first().val());
      priorities[key - 1] = parseInt($(value.children[4]).children().first().val());
    });
```

```
    var currentIndex = -1;
    for (var i = 0; i < exectuteTimes.length; i++) {
      currentIndex = findNextIndexWithPriority(currentIndex, priorities);

      if (currentIndex == -1) return;

      $(inputTable[currentIndex + 1].children[3]).text(totalExectuteTime);

      totalExectuteTime += exectuteTimes[currentIndex];
    }
  }
  else if (algorithm == "robin") {
    $('#minus').css('left', '335px');
    $.each(inputTable, function (key, value) {
      if (key == 0) return true;
      $(value.children[3]).text("");
    });
  }
}

function findNextIndexWithPriority(currentIndex, priorities) {
  var currentPriority = 1000000;
  if (currentIndex != -1) currentPriority = priorities[currentIndex];
  var resultPriority = 0;
  var resultIndex = -1;
  var samePriority = false;
  var areWeThereYet = false;

  $.each(priorities, function (key, value) {
    var changeInThisIteration = false;

    if (key == currentIndex) {
      areWeThereYet = true;
      return true;
    }
    if (value <= currentPriority && value >= resultPriority) {
      if (value == resultPriority) {
        if (currentPriority == value && !samePriority) {
          samePriority = true;
          changeInThisIteration = true;
          resultPriority = value;
          resultIndex = key;
        }
      }
      else if (value == currentPriority) {
```

```
      if (areWeThereYet) {
        samePriority = true;
        areWeThereYet = false;
        changeInThisIteration = true;
        resultPriority = value;
        resultIndex = key;
      }
    }
    else {
      resultPriority = value;
      resultIndex = key;
    }

    if (value > resultPriority && !changeInThisIteration)
      samePriority = false;
  }
});
return resultIndex;
}

function findNextIndex(currentIndex, array) {
  var currentTime = 0;
  if (currentIndex != -1) currentTime = array[currentIndex];
  var resultTime = 1000000;
  var resultIndex = -1;
  var sameTime = false;
  var areWeThereYet = false;

  $.each(array, function (key, value) {
    var changeInThisIteration = false;

    if (key == currentIndex) {
      areWeThereYet = true;
      return true;
    }
    if (value >= currentTime && value <= resultTime) {
      if (value == resultTime) {
        if (currentTime == value && !sameTime) {
          sameTime = true;
          changeInThisIteration = true;
          resultTime = value;
          resultIndex = key;
        }
      }
      else if (value == currentTime) {
        if (areWeThereYet) {
```

```
           sameTime = true;
           areWeThereYet = false;
           changeInThisIteration = true;
           resultTime = value;
           resultIndex = key;
         }
       }
     else {
       resultTime = value;
       resultIndex = key;
     }

     if (value < resultTime && !changeInThisIteration)
       sameTime = false;
   }
 });
 return resultIndex;
}

function animate() {
$('fresh').prepend('<div id="curtain" style="position: absolute; right: 0; width:100%;
height:100px;"></div>');

 $('#curtain').width($('#resultTable').width());
 $('#curtain').css({left: $('#resultTable').position().left});

 var sum = 0;
 $('.exectime').each(function() {
    sum += Number($(this).val());
 });

 console.log($('#resultTable').width());
 var distance = $("#curtain").css("width");

 animationStep(sum, 0);
 jQuery('#curtain').animate({ width: '0', marginLeft: distance}, sum*1000/2, 'linear');
}

function animationStep(steps, cur) {
$('#timer').html(cur);
if(cur < steps) {
       setTimeout(function(){
           animationStep(steps, cur + 1);
       }, 500);
 }
  else {
```

```
    }
}

function draw() {
  $('fresh').html('');
  var inputTable = $('#inputTable tr');
  var th = '';
  var td = '';

  var algorithm = $('input[name=algorithm]:checked', '#algorithm').val();
  if (algorithm == "fcfs") {
    $.each(inputTable, function (key, value) {
      if (key == 0) return true;
      var executeTime = parseInt($(value.children[2]).children().first().val());
      th += '<th style="height: 60px; width: ' + executeTime * 20 + 'px;">P' + (key - 1) + '</th>';
      td += '<td>' + executeTime + '</td>';
    });

    $('fresh').html('<table id="resultTable"><tr>'
              + th
              + '</tr><tr>'
              + td
              + '</tr></table>'
            );
  }
  else if (algorithm == "sjf") {
    var executeTimes = [];

    $.each(inputTable, function (key, value) {
      if (key == 0) return true;
      var executeTime = parseInt($(value.children[2]).children().first().val());
      executeTimes[key - 1] = { "executeTime": executeTime, "P": key - 1 };
    });

    executeTimes.sort(function (a, b) {
      if (a.executeTime == b.executeTime)
        return a.P - b.P;
      return a.executeTime - b.executeTime
    });

    $.each(executeTimes, function (key, value) {
      th += '<th style="height: 60px; width: ' + value.executeTime * 20 + 'px;">P' + value.P +
'</th>';
      td += '<td>' + value.executeTime + '</td>';
    });
```

```javascript
    $('fresh').html('<table id="resultTable"><tr>'
              + th
              + '</tr><tr>'
              + td
              + '</tr></table>'
              );
  }
  else if (algorithm == "priority") {
    var executeTimes = [];

    $.each(inputTable, function (key, value) {
      if (key == 0) return true;
      var executeTime = parseInt($(value.children[2]).children().first().val());
      var priority = parseInt($(value.children[4]).children().first().val());
      executeTimes[key - 1] = { "executeTime": executeTime, "P": key - 1, "priority": priority };
    });

    executeTimes.sort(function (a, b) {
      if (a.priority == b.priority)
        return a.P - b.P;
      return b.priority - a.priority
    });

    $.each(executeTimes, function (key, value) {
      th += '<th style="height: 60px; width: ' + value.executeTime * 20 + 'px;">P' + value.P +
'</th>';
      td += '<td>' + value.executeTime + '</td>';
    });

    $('fresh').html('<table id="resultTable" style="width: 70%"><tr>'
              + th
              + '</tr><tr>'
              + td
              + '</tr></table>'
              );
  }
  else if (algorithm == "robin") {
    var quantum = $('#quantum').val();
    var executeTimes = [];

    $.each(inputTable, function (key, value) {
      if (key == 0) return true;
      var executeTime = parseInt($(value.children[2]).children().first().val());
      executeTimes[key - 1] = { "executeTime": executeTime, "P": key - 1 };
    });
```

```javascript
    var areWeThereYet = false;
    while (!areWeThereYet) {
     areWeThereYet = true;
     $.each(executeTimes, function (key, value) {
      if (value.executeTime > 0) {
       th += '<th style="height: 60px; width: ' + (value.executeTime > quantum ? quantum :
value.executeTime) * 20 + 'px;">P' + value.P + '</th>';
       td += '<td>' + (value.executeTime > quantum ? quantum : value.executeTime) + '</td>';
       value.executeTime -= quantum;
       areWeThereYet = false;
      }
     });
    }
    $('fresh').html('<table id="resultTable" style="width: 70%"><tr>'
             + th
             + '</tr><tr>'
             + td
             + '</tr></table>'
             );
 }
 animate();
}
```

# 7.CODE EXPLAINATION

The provided code seems to be a mix of JavaScript/jQuery code for a CPU scheduling visualizer. It appears to manage the dynamic behavior of a visualizer representing different CPU scheduling algorithms such as First-Come-First-Served (FCFS), Shortest Job First (SJF), Priority Scheduling, and Round Robin.

The code involves various functions that handle the interaction with the user interface, recalculating service times based on the selected algorithm, dynamically updating the visual representation of processes and their execution, and initiating animations to demonstrate the execution of these processes.

Let's break down the major functionalities and key parts of the code:

**Recalculating Service Time and Algorithm Change Handling:**

- `recalculateServiceTime()` function is called upon algorithm changes or input modifications. It recalculates the service time based on the algorithm selected (FCFS, SJF, Priority, Round Robin) and updates the visual representation accordingly.

- `$('input[type=radio][name=algorithm]').change()` manages changes in the selected algorithm, updating the interface and recalculating service times accordingly.

**Adding and Deleting Processes:**

- `addRow()` function is responsible for adding a new process row to the table.

- `deleteRow()` function removes the last row of the process table.

**Animation and Visualization:**

- `animate()` initiates an animation to illustrate the process execution in the visualizer. It dynamically alters the width of an element to create a visual representation of time.

- `draw()` function is responsible for updating and redrawing the visualization table based on the selected algorithm.

**Algorithm-Specific Functions:**

- `findNextIndex()` and `findNextIndexWithPriority()` are helper functions used by different scheduling algorithms (SJF and Priority) to determine the next process to execute based on specific criteria.

**User Interaction Handling:**

- Various event listeners (like 'change', 'click') are used to capture user interactions and trigger necessary updates in the visualization.

The code aims to create an interactive visualization of CPU scheduling algorithms, updating the display based on the selected algorithm and user input. It calculates and represents the execution of processes based on different scheduling techniques, offering a practical view of how these algorithms work.

It's important to note that without the complete context of the HTML structure, the code's functioning might not be fully explained or tested, as it seems to rely heavily on interaction with the HTML elements in the UI.

# 8.RESULT & DISCUSSION

Certainly! The "Result and Discussion" section of a project report on the topic "CPU Scheduling Visualizer" would cover the outcomes and analysis of the developed visualizer. Here is a sample framework:

**Result:**

❖ **Functionality and Performance Evaluation:**
The CPU Scheduling Visualizer successfully implemented various scheduling algorithms including First-Come-First-Served (FCFS), Shortest Job First (SJF), Priority Scheduling, and Round Robin (RR). Users were able to interact with the visualizer, adding processes to the queue and observing their execution based on the selected algorithm. The following results were observed:

-**Algorithm Implementation:** All scheduled algorithms were accurately implemented and produced expected results.
-**User Interface Interaction**: Users were able to smoothly add, delete, and manipulate processes within the visualizer.
-**Visualization and Animation:** The visualizer effectively displayed the execution of processes using animations, allowing users to comprehend how different algorithms manage CPU processes.

❖ **Performance Analysis:**
The performance of each scheduling algorithm was analyzed in terms of:

- **Turnaround Time:** Calculations and representations of turnaround time for each algorithm were accurate and provided insights into the time taken for a process to complete from arrival to completion.
- **Waiting Time:** The waiting time analysis accurately reflected the duration a process spent in the ready queue waiting for CPU execution.
- **Throughput:** Evaluation of the number of processes completed per unit of time for each algorithm provided an understanding of their efficiency in handling multiple tasks.

**Discussion:**

❖ **Algorithm Comparison:**
The visualizer facilitated a comprehensive comparison among scheduling algorithms:

- **FCFS:** Showed longer average waiting times and turned out to be less efficient, especially in scenarios with longer processes waiting in the queue.
- **SJF:** Demonstrated its effectiveness in minimizing waiting time by executing shorter processes first, but showed challenges in practical implementation due to the need for precise burst time prediction.
- **Priority Scheduling:** Highlighted the importance of managing priorities but also the potential for priority inversion issues.
- **Round Robin:** Displayed fair CPU allocation with shorter time slices, although this method might result in higher context switching overheads.

❖ **User Feedback:**
During testing and user engagement, feedback highlighted:

- **Educational Value:** The visualizer was highly informative and beneficial for students, educators, and system administrators, aiding in the practical understanding of CPU scheduling concepts.
- **User-Friendly Interface:** The users found the interface intuitive and easy to navigate, contributing to an enhanced learning experience.

❖ **Future Enhancements:**
To further improve the visualizer, potential enhancements include:

- **Real-Time Data:** Implementing real-time data integration for more practical simulation.
- **Additional Algorithms:** Incorporating more advanced scheduling algorithms to expand the range of comparative studies.
- **Enhanced Visual Representations:** Utilizing more complex visualizations to demonstrate intricate algorithmic behaviors.

# 9.CONCLUTION

The CPU Scheduling Visualizer project stands as a pivotal tool in comprehending and visualizing the intricate dynamics of CPU scheduling algorithms. Through a meticulous approach encompassing design, implementation, and testing, this project has successfully manifested into a practical, interactive platform aimed at educating students, aiding educators, and assisting system administrators in understanding the real-world implications of diverse scheduling methodologies.

The exploration of scheduling algorithms, including First-Come-First-Served (FCFS), Shortest Job First (SJF), Priority Scheduling, and Round Robin (RR), within this visualizer, has provided an insightful perspective into the functionality and trade-offs associated with each approach. Users are able to interact with the system, add processes, and visualize their execution, thus gaining hands-on experience in observing the behavior and impact of different scheduling strategies.

The project's development methodology, starting from requirements analysis to design, implementation, and rigorous testing, has been instrumental in ensuring a robust and user-friendly interface. User feedback and testing iterations have been invaluable in refining the visualizer, resulting in a sleek and intuitive user interface that effectively communicates the complexities of CPU scheduling in a simplified manner.

As a tool for educational purposes, this visualizer facilitates a better understanding of CPU scheduling, shedding light on the strengths and limitations of each algorithm. It equips learners with practical insights into the implications of these methodologies in real-world computing environments, fostering a deeper appreciation of the decisions made by system designers and administrators.

In conclusion, the CPU Scheduling Visualizer serves as an invaluable asset, not only in academic settings but also for industry professionals seeking to grasp the impact of scheduling decisions on system performance. As technology continues to evolve, this project remains a fundamental resource in the ongoing study and comprehension of CPU scheduling algorithms, offering a dynamic and practical experience to all its users.

# 10.REFERENCE

1. [https://www.futurelearn.com/info/courses/computer-systems/0/steps/53513](https://www.futurelearn.com/info/courses/computer-systems/0/steps/53513)

2. [https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/](https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/)

3. [https://www.javatpoint.com/cpu-scheduling-algorithms-in-operating-systems](https://www.javatpoint.com/cpu-scheduling-algorithms-in-operating-systems)

4. [https://www.guru99.com/cpu-scheduling-algorithms.html](https://www.guru99.com/cpu-scheduling-algorithms.html)

5. [https://www.scaler.com/topics/operating-system/cpu-scheduling/](https://www.scaler.com/topics/operating-system/cpu-scheduling/)

6. [https://www.shiksha.com/online-courses/articles/cpu-scheduling-algorithm-operating-sys tem/](https://www.shiksha.com/online-courses/articles/cpu-scheduling-algorithm-operating-sys tem/)

7. Operating Systems Concepts 10th edition by ABRAHAM SILBERSCHATZ, PETER BAER  GALVIN & GREG GAGNE