Sched-ITS: An Interactive Tutoring System to Teach CPU Scheduling Concepts in an Operating Systems Course

Bharath Kumar Koya

Wright State University

Follow this and additional works at: https://corescholar.libraries.wright.edu/etd_all

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Repository Citation
https://corescholar.libraries.wright.edu/etd_all/1745

This Thesis is brought to you for free and open access by the Theses and Dissertations at CORE Scholar. It has been accepted for inclusion in Browse all Theses and Dissertations by an authorized administrator of CORE Scholar. For more information, please contact library-corescholar@wright.edu.
SCHED – ITS: AN INTERACTIVE TUTORING SYSTEM TO TEACH CPU SCHEDULING CONCEPTS IN AN OPERATING SYSTEMS COURSE

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science

By

BHARATH KUMAR KOYA
B.E, Andhra University, India, 2015

2017
Wright State University
April 24, 2017

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Bharath Kumar Koya ENTITLED SCHED-ITS: An Interactive Tutoring System to Teach CPU Scheduling Concepts in an Operating System Course BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science.

_____________________________________
Adam R. Bryant, Ph.D.
Thesis Director

_____________________________________
Mateen M. Rizki, Ph.D.
Chair, Department of Computer Science and Engineering

Committee on Final Examination

_____________________________________
Adam R. Bryant, Ph.D.

_____________________________________
Mateen M. Rizki, Ph.D.

_____________________________________
Yong Pei, Ph.D.

_____________________________________
Robert E.W. Fyffe, Ph.D.
Vice President for Research and Dean of the Graduate School
ABSTRACT

Koya, Bharath Kumar. M.S., Department of Computer Science and Engineering, Wright State University, 2017. Sched-ITS: AN Interactive Tutoring System to Teach CPU Scheduling Concepts in an Operating Systems Course

Operating systems is an essential course in computer science curriculum, which helps students to develop a mental model of how computer operating systems work. The internal mechanisms and processes of an operating system (OS) are often complex, non-deterministic and intangible which makes them difficult for students to understand. One such concept is central processing unit (CPU) scheduling. CPU scheduling forms the basis of the multiprogramming in an OS.

In practice, OS courses involve classroom lectures describing high-level abstractions of the concepts, and students complete programming assignments to apply the material in a more concrete way. Depending on the programming assignments, this approach may leave students with only a theoretical understanding of OS ideas, which may be different from the actual way these concepts are implemented in an OS. What many students require is a practical knowledge of OS implementation to supplement the high-level presentations of concepts taught in class or presented in a textbook.

To bridge the gap between the operating system theory and practical implementation, this research describes the development of an interactive simulation to present the theories involved in CPU scheduling in visualizations and simulations. This thesis discusses a prototype interactive tutoring system (ITS) named as Sched-ITS. The
tool covers all the important algorithms of CPU scheduling such as first-come, first-serve (FCFS), round robin (RR), shortest job first (SJF), shortest remaining time first (SRTF), priority with pre-emption, and priority without pre-emption.

Sched-ITS also provides graphical visualization of how context switches occur during CPU scheduling in a real operating system. Sched-ITS makes use of the JavaFX framework for visualization and Perf-tool for tracing an OS’s scheduling activities. It presents scheduling activities of background processes as well as pre-defined or user-defined processes. Sched-ITS can display scheduling order changes for different algorithms for the same set of processes in a Linux operating system.
## Contents

Abstract ........................................................................................................................................ iii

List of Figures ............................................................................................................................... vii

1. Introduction .............................................................................................................................. 1
   1.1. Overview ........................................................................................................................... 1
   1.2. Statement of Problem ......................................................................................................... 5
   1.3. Scope ................................................................................................................................... 5
   1.4. Significance of the study ...................................................................................................... 6

2. Background & Literature Review .............................................................................................. 8
   2.1. Overview ............................................................................................................................ 8
      2.1.1. Kernel overview ........................................................................................................... 8
      2.1.2. Different Types of Processes ...................................................................................... 13
      2.1.3. Process Scheduling .................................................................................................... 14
      2.1.4. Scheduler ................................................................................................................... 15
      2.1.5. Scheduling Criteria .................................................................................................... 17
      2.1.6. Scheduling Algorithms .............................................................................................. 18
      2.1.7. Linux Kernel Overview ............................................................................................... 20
      2.1.8. Linux Monitoring Tools .............................................................................................. 21
   2.2. Literature Review ................................................................................................................ 27
      2.2.1. CPU Scheduling Simulation projects ........................................................................ 27
   2.3. Related Real-Time Scheduler Visualization Projects .......................................................... 31
      2.3.1. Scheduler Visualization Projects .............................................................................. 31
      2.3.2. Scheduling Simulation Projects ................................................................................ 33

3. Design and Implementation ......................................................................................................... 38
   3.1. Simulation mode .................................................................................................................. 39
      3.1.1. System Model .............................................................................................................. 39
      3.1.2. Tasks (Jobs) ................................................................................................................. 40
      3.1.3. Resources ................................................................................................................... 41
      3.1.4. Events ......................................................................................................................... 41
      3.1.5. Simulation Parameters ............................................................................................... 41
      3.1.6. Simulator .................................................................................................................... 42
      3.1.7. Simulation Thread ........................................................................................................ 43
      3.1.8. Simulator Operation .................................................................................................. 44
List of Figures

Figure 1. Different states in life cycle of a process ........................................... 10
Figure 2. Different fields in Process Control Block ................................................. 11
Figure 3. Context Switching ................................................................................... 13
Figure 4. Different Queues in Process Scheduling ................................................. 15
Figure 5. Flow diagram of the Tool ....................................................................... 38
Figure 6. Main Frame in the simulation mode ......................................................... 47
Figure 7. Memory Pane (Top left pane) .................................................................. 47
Figure 8. Job Pool Pane (Top right pane) ................................................................. 49
Figure 9. Queue Pane (Middle Pane) ...................................................................... 50
Figure 10. Gantt Chart (Bottom Pane) .................................................................... 51
Figure 11. Window to input data manually .............................................................. 53
Figure 12. High-level design of the Real-Time mode ............................................. 57
Figure 13. Commands in Perf Tool ....................................................................... 58
Figure 14. Tracing the scheduler switch tracepoint .............................................. 58
Figure 15. Output of Perf script command .............................................................. 59
Figure 16. Visualization of a sample trace of Background Processes .................. 67
Figure 17. Frame to select the options for the scheduling .................................... 68
Figure 18. Visualization of a sample trace of Pre-defined set of Processes ............ 70
Figure 19. Frame to select the User-defined set of processes ............................. 71
Figure 20. Visualization of a sample trace of the User-defined set of Processes........ 72
Figure 21. Simulation Window before the start of the simulation.......................... 77
Figure 22. Simulation window at 10 th step of the simulation................................. 78
Figure 23. Simulation window at 35 th step of the simulation................................. 78
Figure 24. Simulation window at 65 th step of the simulation................................. 79
Figure 25. Simulation window at the end of the simulation ................................. 79
Figure 26. Visualization of a sample trace of the first subsection of the Real-Time mode ................................................................. 80
Figure 27. Visualization for the set with different processes under SCHED_NORMAL Scheduling Policy ............................................................................................................. 81
Figure 28. Visualization for the set with different processes under SCHED_FIFO Scheduling Policy ............................................................................................................. 81
Figure 29. Visualization for the set with different processes under SCHED_RR Scheduling Policy ............................................................................................................. 82
Figure 30. Visualization for the set with different processes under SCHED_BATCH Scheduling Policy ............................................................................................................. 82
Figure 31. Visualization for the set with same processes under SCHED_NORMAL Scheduling Policy ............................................................................................................. 84
Figure 32. Visualization for the set with same processes under SCHED_FIFO Scheduling Policy ............................................................................................................. 84
Figure 33. Visualization for the set with same processes under SCHED_RR scheduling policy ............................................................................................................. 85
Figure 34. Visualization for the set with same processes under SCHED_BATCH scheduling policy ................................................................. 85
1. Introduction

1.1. Overview

Undergraduate students in computer science programs must complete a course covering the internals and design of an operating system (OS). University instructors teach this course by presenting the concepts at a theoretical level from a textbook and assigning programming assignments to supplement the student’s understanding with practical implementation details.

One of the greatest challenge for students in learning OS internals is the complexity of the concepts. OS concepts and processes are very detailed and interrelated with one another. When students are only exposed to high-level theories, they can find it difficult to imagine how these concepts are implemented and mentally simulate how the processes work. Our working hypothesis is that if it takes less mental work to imagine OS concepts and processes, students can understand them more quickly. Hence, a tool that reduces the work involved to imagine OS concepts and processes can help reduce students’ stress and increases their ability to learn the material. A proven way to reduce students’ work in imagining complex concepts and processes is present simplified (or abstracted) visual representations of them.
In computing, scheduling is the process of assigning work to a resource so a computational element can complete it. Work that requires scheduling may make use of concrete computational elements like the central processing unit (CPU) or virtual computational elements like threads and processes. Resources are objects that computational elements require to complete work. In an OS, these include memory units, processors, files, and network ports. A scheduler is software that assigns a set of computational elements to a set of resources to ensure work is carried out and completed. A scheduler should maximize the time that resources are in use (or busy) so as to complete work more efficiently and achieve a higher quality of service for users. Scheduling is fundamental to any computational system and an intrinsic part of the execution model that makes it possible to execute multiple tasks on a single resource.

One of the fundamental OS concepts is process scheduling (or CPU scheduling), which schedules execution tasks to threads of execution and schedules system resources to those threads to maximize CPU utilization. Most modern-day OSs run in a multiprogramming environment, in which several processes compete to run on the same CPU. OSs must have scheduling criteria to avoid problems like deadlock, busy waiting, or even a crash of the OS. Scheduling algorithms attempt to efficiently share the CPU and resources among multiple processes. In a multiprogramming environment, an OS kernel has a program called a CPU scheduler. The scheduler implements one or more scheduling algorithms to select and efficiently distribute the CPU and resources among multiple processes according to various criteria. This in turn optimizes the performance of the system. In an interactive environment, the scheduler is used to optimize
multiprogramming, which creates an illusion for the user that all the processes are running at the same time on the same processor.

CPU scheduling employs various algorithms. These scheduling algorithms distribute resources among competing tasks that request system resources. Most computing and network devices (like network switches and routers, disk drives, file managers, operating systems, and embedded systems) use scheduling algorithms. The goals of any scheduling algorithm are to increase resource utilization, avoid deadlock, and decrease resource starvation among all contending parties that need to use the resource. Many scheduling algorithms are optimizations to improve upon the first-come, first-serve (FCFS) queuing of events. The most common scheduling algorithms used in CPU scheduling are FCFS, round robin (RR), shortest job first (SJF), and priority-based scheduling. These basic scheduling algorithms act as a reference so students can compare and understand the goals, purpose, rationale, and mechanisms of various scheduling optimizations and can analyze their effectiveness.

Of all the algorithms, SJF is the most widely-implemented in real operating systems (Silberschatz, 2006). In many textbooks and courses, this algorithm acts as a benchmark by which students can compare other algorithms. SJF selects the process with shortest required sequence of instructions (CPU burst) from the OS’s event queue to allocate it to the CPU. When all processes in the event queue have similar CPU bursts, FCFS is used to select among the processes. If the processes’ CPU bursts are too different, RR can be used to achieve fairness among contending processes.
Even though there are many CPU scheduling algorithms, the greatest challenge in learning CPU scheduling is to understand the sequence of actions to schedule a process and how those action sequences relate to process metrics. There are many sources available to learn these concepts such as textbooks and video lectures. A problem with these learning sources is that they lack of interactivity, and thus do not allow students to experiment with the ideas and reinforce cause-and-effect relationships that are implicit in the algorithms. Most textbooks introduce scheduling using static diagrams. Although the pictures present the concepts correctly, they provide little assistance for the students to memorize and correctly predict the algorithms’ behavior under various conditions. Moreover, some textbooks only present the most basic scheduling algorithms with ideal processes under ideal conditions and do not present detailed execution of each of the steps.

Another shortcoming is that textbooks rarely cover the scheduling algorithms as they are employed in a real operating system. This significantly hinders the learning opportunities for curious students. Considering these challenges, there exists a need for an interactive learning environment that presents both theoretical and practical concepts of CPU scheduling through visualization and simulation.

Sched-ITS presents visualizations and simulations of CPU scheduling. It incorporates a workbench environment allowing students to probe, modify, and observe scheduling behavior under different workloads both in simulation and in a real-time environment. The simulation environment allows the user to specify scheduling parameters for illustrative processes and to observe the scheduling patterns used in particular algorithms. The real-time environment allows the user to see the scheduling
behavior of processes in a running operating system and to probe a set of processes and observe their scheduling patterns along with some scheduling parameters. This application also allows users to tailor the complexity of the processes it displays according to their knowledge of the concepts.

1.2. Statement of Problem

There is a need to incorporate supplementary interactive learning aids in OS courses which simplify the tasks of imagining CPU scheduling and bridge the gap between theoretical and practical knowledge on the subject. In addition, use of a real operating system to teach scheduling concepts can supplement theoretical knowledge students acquire from lectures and textbooks. Unfortunately, scheduling algorithms run in the OS kernel, making it difficult for any instructor to give a live demonstration of scheduling as it is used in a real OS. Furthermore, due to the non-deterministic nature of how processes are introduced and work completed, it is difficult to show how changing algorithms and algorithm parameters affect how OSs complete work under actual operating conditions.

1.3. Scope

The simulations in Sched-ITS (and presented in this thesis) involve visualizations of the scheduling algorithms presented in the textbook *Operating Systems Concepts* by Silberschatz, Galvin, & Gagne, (9th edition, published 2006). Sched-ITS does not simulate the multi-level feedback algorithm which is less frequently used or algorithms
incorporating real-time deadlines (Silberschatz, 2006). The algorithms in this textbook differ slightly from algorithms implemented real operating systems, so this tool provides visualizations to represent those more advanced algorithms as well.

This thesis study addresses the following research questions:

Q1: How does existing literature portray the constraints and limitations of currently-available CPU scheduling tutorial and simulation systems?

Q2: How can these limitations and constraints be used to define the software requirements for Sched-ITS?

Q3: How can Sched-ITS present learners useful interactions with the scheduling patterns in a running operating system such as Linux?

Q4: What testing activities are needed in measuring the quality of the tool and determining whether its simulation are comparable with theoretical predictions?

1.4. Significance of the study

This research aims to simplify learners’ work in imagining scheduling concepts by simulating those concepts in a graphical and interactive environment. Sched-ITS even visualizes scheduling in a running Linux OS, which helps learners solidify their knowledge. This tool presents CPU scheduling concepts at both a theoretical level and a practical one and can be used to present live demonstrations of scheduling activity in theory and in a running Linux OS. It can also be helpful to developers for observing
scheduling for a particular workload in a Linux environment so they can make changes to improve its performance.
2. Background & Literature Review

2.1. Overview

2.1.1. Kernel overview

The main component in an OS is the kernel. One of the important modules of the kernel is the scheduler. The responsibility of the scheduler is to manage system resources across several contexts. First, it provides an interface for the kernel to access hardware and software resources, which helps the kernel communicate with applications that request these resources using systems calls. Second, the kernel allocates resources such as CPU and memory among many contending processes. Third, the scheduler maps subsystems in the kernel software to resources the kernel is managing. The scheduler creates and destroys processes, and schedules processes for execution on the CPU.

2.1.1.1. Program

A program is a file containing code and data that are executed by the CPU. When executed, the OS loads the code and data into memory, creates a process, and schedules the threads of that process on the CPU to complete the process task. A program is a static entity containing all the information needed by an OS to create a process that executes on
the CPU. It resides at a single location, typically in secondary memory such as a hard disk drive, and cannot perform any actions on its own.

2.1.1.2. Process

A process is an entity in an operating system that represents an instance of the program, but which a system can execute. It is the entity that represents the core unit of work in an OS. Processes contain executable code, handles to open system objects, and unique identifiers. While a program is a passive set of instructions, a process is the representation that permits the CPU’s execution of those instructions.

Several processes can be instances of the same static program. For example, when a user opens the same program several times, it creates several instances of the same program, each in its own process. The state of a process changes as it executes. A multitasking kernel defines a set of states for the processes. The states of a process are:

1. **New**: When a system creates a new process.
2. **Ready**: When the process is ready to execute on the CPU.
3. **Running**: When the CPU is executing the process’ instructions.
4. **Waiting**: The process is waiting for an event to happen before it executes again.
5. **Terminated**: When the process terminates after completing its execution.
2.1.1.3. **Process Control Block (PCB)**

Every process has a data structure called the PCB, which resides in the kernel’s memory space. Each PCB contains the information needed to manage the process to which it belongs. Many OS units involving scheduling, I/O management, and performance management access and modify process’ PCBs. The combined current state of all PCBs in an OS represents the current state of the OS. The OS uses pointers inside the PCBs to create the queues of processes, and to manage the processes. These queues include *ready*, *waiting*, *terminated*, and *new*.

PCBs have three categories of data:

1. Process identification data,
2. Process state data, and
3. Process control data.

Process control data contains information that is important for the OS to manage all processes in the system and for those processes to perform work. These include:
The program counter (PC),
Process ID (PID),
CPU registers,
I/O status information,
Memory management information,
Process state, and
List of open files.

Figure 2. Fields in the Process Control Block
Image from URL: www.genome.gov/Pages/Hyperion/DIR/VIP/Glossary/Illustration/rna.shtml

2.1.1.4. Process Priority

Each process in an operating system has a priority which is a value assigned by the kernel. The scheduler makes use of this priority to make scheduling decisions. There are two types of priorities, namely static and dynamic priorities. When the system creates a process, it will have a priority that the scheduling algorithm can change in response to its selection criteria. Then the scheduler usually selects the process with the highest priority to schedule its execution.
2.1.1.5. *Process Pre-emption*

Even in an OS with a single CPU, it appears to the user that several processes are running concurrently. However, only one process can run at a time in a single-processor system. Processes usually are started and stopped several times before their completion. This act of blocking a process is pre-emption. Linux is one example of a pre-emptive OS.

2.1.1.6. *Context Switching*

Context switching is the mechanism of storing and restoring the context of a process to and from the CPU to ensure that when the process executes again, its execution resumes from the interruption point. The process control block holds the context of its respective process. Context switching is the key mechanism involved in multitasking as it enables multiple processes to share the same CPU. Thus, when the process gets a turn to execute again, its previous context is loaded from its process control block, which reloads the program counter and registers back into the CPU.
2.1.2. Different Types of Processes

Any process falls into one of the four categories below based on the behavior and performance as follows:

- **CPU-bound process**: If a process mostly involves CPU computation, then it is a CPU-bound process. It does not have as many I/O operations as it does CPU computations.
• **I/O-bound process:** If a process involves mostly I/O operations, then it is an I/O-bound process. These type of processes do not require as much of the CPU’s time to complete their execution.

• **Interactive processes:** If a process requires extensive interaction from the user, then it is an interactive process. These processes must only take a small time to meet the user’s needs.

• **Mixed processes:** These processes combine all the above functions. Their behavior depends on the mix of the types of operation they perform.

2.1.3. Process Scheduling

The CPU can run only on one process at a time. Therefore, there is a need to share the CPU among different processes. Process scheduling is the selection of a process from a long queue of ready processes based on a particular strategy. The scheduler manages the scheduling activity of the CPU, handles the removal of a running process from the CPU, and selects another process to execute. The operating system loads more than one program into the executable memory at a time and sharing the CPU, using time multiplexing.

Any operating system maintains all the process control blocks (PCBs) in process scheduling queues. There is a separate queue for each process state. PCBs of all the processes in the same execution state will be in the same Process Scheduling queue. When process state changes, its PCB will be unlinked from its current queue and linked to another queue based on the state. The main scheduling queues in an operating system are the Job Queue, the Ready Queue, and Device Queues (Wait Queues). The Job Queue
holds the list of all the processes in the system. Ready Queue has the set of all processes, which are ready and waiting to execute on CPU. All these ready processes reside in the main memory. The respective I/O device Wait Queues has the set of all processes, which are waiting due to unavailability of an I/O device.

![Different Queues in Process Scheduling](image)

The operating system can use distinct policies to manage each queue, like FIFO, RR, or Priority. The scheduler in the operating system determines when to move and how to move a process between different queues.

### 2.1.4. Scheduler

Scheduler is the software or program that deals with the handling of processes scheduling in any operating system. The primary job of any Scheduler is to pick which
process to run. Schedulers are of three types: the Long Term Scheduler, the Short Term Scheduler, and the Medium Term Scheduler.

2.1.4.1. **Long Term Scheduler**

The task of the long-term scheduler is to determine which programs to admit into the system for processing, which means selecting processes from the queue and then loading them into the main memory for execution. Therefore, it is also named as "Job Scheduler." The primary objective of the long-term scheduler is to provide a balanced load of jobs, like a mix of I/O bound processes and CPU bound processes. It also has control over the degree of multiprogramming. Under a stable degree of multiprogramming, it is the job of the long-term scheduler to maintain the average rate of process creation equal to the average rate of processes leaving the system. In some systems, like Time-sharing operating systems, there is no Long Term Scheduler. The Long Term Scheduler appears when a process changes from the new to the ready state.

2.1.4.2. **Short Term Scheduler**

The job of the Short Term Scheduler is to enhance the system performance by the criteria chosen. It allocates the CPU to a process by selecting one of the processes from a list of ready processes. Therefore, it is also called the CPU scheduler. It is the program that changes the state of a process from the ready state to the running. It is sometimes referred to as the Dispatcher because it makes the decision in selecting the next process to be executed. Short Term Schedulers are often faster than the Job Schedulers.
2.1.4.3. Medium Term Scheduler

A running process changes its state from running to waiting when it makes an I/O request. In the waiting state, a process cannot make any advancement towards completion. In this state, moving of the waiting process to secondary storage to create room for new processes. Medium Term Scheduler is the program or software that deals with swapping a process out of the memory. It is also known as the Process Swapping Scheduler. A process state changes to suspended, when it swaps out. Swapping is necessary to enhance the process mix. The Medium Term Scheduler speed of operation is in between Short Term Scheduler and Long Term Scheduler speed. It is a part of the time sharing systems.

2.1.5. Scheduling Criteria

The central part of any scheduler is its scheduling policy, which says how to select the next process. Usually, scheduling relies on the following criteria: Turn-Around Time, Response Time, CPU Utilization, Throughput, and Waiting Time.

1. Turn-Around Time – The time a process takes to complete from its submission time.

2. Response Time – The time interval between the issuance of the command and the program’s response in an interactive environment.

3. CPU Utilization – The number of cycles of CPU doing useful work. In real time systems, CPU utilization should be in between 40% load (light load) to 90% load (heavy load).

4. Throughput – The number of processes completed per unit time.
5. Waiting Time – The time a process waits in the Ready Queue for the CPU to execute it.

In any operating system, Developers optimize all of the criteria as mentioned earlier to have a better performance.

2.1.6. Scheduling Algorithms

A Scheduler schedules different processes on the same CPU based on a particular Scheduling Algorithm. The traditional scheduling algorithms, in theory, are as follows:

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

The algorithms mentioned above are either preemptive or non-preemptive. When it comes to non-preemptive algorithms, if a process enters the running state, it cannot be pre-empted until its execution finishes or its time slice expires. However, in preemptive scheduling, it preempts a running process if another with higher priority is ready to execute (when it enters the Ready Queue).

2.1.6.1. First-Come-First-Serve Scheduling

It is the simplest of all the scheduling algorithms. With this algorithm, the scheduler allocates the CPU to the processes that request the CPU first. Processes that are...
ready to execute are present in the Ready Queue. Under this scheduling policy, the scheduler picks the process that is at the front (or head) of the queue. New processes that are ready to be executed adds to the end (or tail) of the queue.

2.1.6.2. **Round Robin Scheduling**

Time-sharing systems mainly uses this algorithm. In this algorithm, each process in the ready queue gets a time slice to execute on CPU called time quantum. Thus, here the ready queue is a circular queue. The process that is at the front of the ready queue executes for next time quantum. Once time slice expires, Scheduler adds to the end (tail) of the ready queue. The process next in the queue executes for the next time quantum.

2.1.6.3. **Shortest-Job-First Scheduling**

This algorithm requires the burst time of all the ready processes to decide on the next process. Once all the CPU bursts are available, it selects the process with the shortest burst. If any of the two processes have the same burst time, then the process that arrived first gets the CPU to execute. This algorithm exists in theory only because there is no prior information of processes burst time. This algorithm is a benchmark for other algorithms to compare.

2.1.6.4. **Shortest-Remaining-Time-First Scheduling**

This algorithm is the preemptive version of the shortest job first algorithm. The job that is closest to completion takes the CPU. However, it can also be pre-empted if a new process with shorter time to complete is available in the ready queue. It is not implementable in interactive systems as CPU burst is not known prior itself.
2.1.6.5. **Priority based Scheduling**

This algorithm uses the priority as a parameter to schedule a bunch of processes. Every process in an operating system has a priority associated with it. The process that has the highest priority in the ready queue is the next process. Either a system assigns priority to a process internally or a user can assign externally.

2.1.7. **Linux Kernel Overview**

Linux is one of the monolithic Unix-like operating systems. Traditional computer systems like personal computers and servers and in various embedded systems like routers, smart TVs uses this OS. The primary entity of the Linux operating system is the Linux kernel. A kernel is a software that interfaces with the hardware in the computer. It interfaces all the applications that are running in the user space to the physical device. Linux kernel is one of the monolithic kernel available in the market. Thus, the entire operating system runs in privileged mode. Kernel tracepoints can monitor the events that take place in the kernel.

2.1.7.1. **Kernel Tracepoints**

Tracepoint is a marker within the kernel, which when enabled can trace and output the particular activity of the running kernel. These tracepoints are the base to develop many performances monitoring tools and debugging tools. A user can either enable or disable a tracepoint. When disabled it has no effect, except a space penalty. However, when enabled, the function provided by the user is called every time a tracepoint executes in the context. When the called function execution finishes, execution returns to the
tracepoint site. Tracepoints are present at significant locations of the source code to pass a set of parameters to the user space. Linux kernel provides some tracepoints without the need of probing. There are scheduler tracepoints built in to observe the scheduler activity. Some of the trace points present in the Linux kernel to monitor the scheduler activity are:

1. sched:sched_stat_wait
2. sched:sched_stat_sleep
3. sched:sched_stat_iowait
4. sched:sched_switch
5. sched:sched_stat_blocked
6. sched:sched_stat_runtime

Tracing the above tracepoints provides the necessary Scheduler event information.

2.1.8. Linux Monitoring Tools

This section gives an overview of the monitoring tools that are available to trace the scheduler activity in the Linux kernel. There are many monitoring tools available. The most used ones are:

2.1.8.1. Pid Stat

This tool monitors the single tasks in the Linux kernel. This tool can trace a list of processes using the s-option. There are different options in the tool; each has its functionality.
S option: This option displays the stack memory usage. We get information about the stack size allocated to a process and stack memory used by the process.

W option: This option reports the process switching. We get information about the number of voluntary and non-voluntary context switches. Voluntary context switches are the context switches that happen due to unavailability of resources. Whereas, non-voluntary context switches occur when a process is pre-empted by other processes or due to its time slice expired,

d option: This option is used to report the I/O statistics of a process. We get information about the amount of data read and written by the process.

r command: This option is used to report the page faults and memory utilization for a particular process. Page faults are of two types: minor page faults and major page faults. We get information about the minor and major page faults.

2.1.8.2. Proc File System

The proc file system comes built-in with many releases of the Linux. It is the one of the widely used systems monitoring tool because of its simple API, compatibility with POSIX and high security. It is a virtual file system that allows a user to monitor as well as manipulate the kernel data structures without any extra memory. Unlike many other system-monitoring tools, it will provide direct access to the kernel. The main important feature of the Proc file system is that it imposes a low overhead on the overall system along with run-time monitoring. However, Proc file system has two main disadvantages that the size of the code that was included to support the proc file system is large and it is
not portable to all version of Linux. Moreover, proc file system is the base for many tools. These include Top and Sysstat tools. In Proc file system, /proc/sys allows the user to modify the kernel using the echo command. Some of the useful subdirectories in the proc file system are-

- **proc/sys/kernel**: It contains information about general kernel behavior and broad parameters.
- **Proc/sys/vm**: It contains information about the virtual memory, which when modified either enhances or degrades the performance.
- **Proc/devices**: It provides information about the devices that are available to the kernel.
- **Proc/sys/dev**: It provides information about the device specific parameters.
- **Proc/modules**: The kernel modules are present in this location in the proc file system.
- **Proc/cmdline**: It contains information about the kernel command line.
- **Proc/meminfo**: It provides information about the memory usage.

### 2.1.8.3. **Top**

Top is one of the tools that provides the real-time report of the process activities in the system. It allows the user to filter the monitored data using commands. It interacts with the user through the command line. Top output has four areas- Summary area, columns header, task area and prompt line.

- **Summary area**: First five lines in the output is the summary area, which gives an overview of the system activities. The first line provides information
about command name, system uptime, current system time, the load on the CPU and the number of users. The second line presents information about the number of processes in the system. The third line gives information about the CPU usage from the last refresh to the current time in the system. The last two line presents information about the memory usage of total physical memory, free memory, swap memory.

- **Columns header:** It includes different headings of the columns.
- **Prompt area:** The line after the summary is the prompt area. The user to perform subcommands of Top using prompt commands can use this area. This area also displays the error messages.
- **Task area:** This area has the details of all the processes in the system. This information includes process ids, username, process priority, nice value, total virtual memory per process, shared memory, process state, CPU usage.

Top provides the user with many options to monitor and provides a well-structured information. However, its main disadvantage is that it imposes a significant overhead as it uses too many system calls to get the required information.

### 2.1.8.4. Sysstat Project

The sysstat tool makes use of the proc pseudo file system to get the raw data about the processes in the system. It then makes use of the raw data to develop a display and database from the execution history. The raw data collected is stored for a period, and then the Sysstat calculates the mean values to make it available to the user as per his interest. Sysstat is a combination of different tools like iostat, pidstat, mpstat, sar,
nfsiostat. This tool finds its use in many commercial UNIX systems. This tool provides information like process ids, process names, CPU utilization, CPU load, memory usage, virtual memory usage, paging, network interfaces, interrupts, context switches and queues. The sysstat tool facilitates the users to use the data from proc file system without the need to learn the skills required to extract the data manually.

2.1.8.5. Mpstat

The Mpstat command gives information about the activities of each processor in the system along with average global activities. In this tool, there is an option to select the time span between each report using interval parameter. The value 0 means that the data is from system start up. Mpstat makes use of the proc pseudo file system to get the required information. For example, it gets CPU load and interrupts information from /proc/stat, Number of interrupts derived from /proc/interrupts, and the software interrupts information obtained from /proc/softirqs.

2.1.8.6. Gnome Desktop Environmental Tool

This tool is the equivalent of the task manager in the Windows operating system. It is open source software, which consists of a desktop environment and a graphical user interface for the user to play around. This tool has information about the system status, pre-emptive of processes, memory usage, and some process-level statistics. This tool also provides the option to kill process directly from the graphical interface.
2.1.8.7. \textit{Perf}

\textit{Perf} means performance counters for Linux. It is one of the dominant performance analyzing tools. It can trace the CPU performance counters and tracepoints available in the Linux. It is a lightweight profiling tool that is available in the Linux kernel under Linux tools Common packages. Performance counters are the hardware registers that store the count of the events like the total number of instructions executed; cache misses. Perf traces these counters to get the dynamic behavior of the kernel. Perf provides a way to trace these performance counters to get per task parameters and per CPU parameters, in which many other tools fail. One can use tracepoints to get details of TCP events, system calls, and file system operations. Tracepoints cause no overhead when not in use. Perf provides a rich interface for probing new tracepoints as per the need. Perf also traces the static tracepoints available in the Linux kernel. This tool interfaces through the command line. Some of the user space commands that are available in the perf package are:

\begin{itemize}
  \item Perf top: This shows the live event count of the system.
  \item Perf stat: To show the system-wide statistics.
  \item Perf Record: To record trace points of specific interest for later use.
  \item Perf report: To show the output generated by the perf record in the human readable form.
  \item Perf annotates: To annotate the source code with events.
\end{itemize}
2.2. Literature Review

2.2.1. CPU Scheduling Simulation projects

Using animations as an interactive approach towards scheduling algorithms is not new. There are many projects present for illustrating the scheduling algorithms behaviors. Each of the tools uses simulation to explain the scheduling algorithms using hypothetical processes with configurable parameters accompanied by graph and provide the timing statistics. Some of the examples are below:

2.2.1.1. OSCAL

OSCAL is one of the examples, which presents the student with an environment that simulates the standard scheduling algorithms. The user will be in a state of confusion when viewing the static image in the textbooks, as he cannot understand what steps of the algorithm created the image. Applications like this give the systematic implementation of the algorithm improving the understanding of the student. Moreover, a user can stall the simulation of the algorithm using the active controls when confusion occurs in understanding the steps. Pausing the simulation will effectively help the students to separate the aspects of the algorithm with the sparsest understanding to the user allowing him/her to avoid the potential pitfall associated with misdirected studying. (Zhaogd, 2003)
2.2.1.2. **RCOS**

The intention of this tool is to help students understand the internal working of an operating system. It is an animated tool, which runs on the simulated hardware. It demonstrates the general concepts in the operating systems in a controlled animation environment. This tool provides the students with an environment, which enables them to modify the experiment and compare different algorithms and data structures. This tool explains many concepts like process scheduling, inter-process communication, disk management, file systems and so on. (Jones & Newman, 2001)

2.2.1.3. **CPU Scheduling Algorithm Simulator**

“The Design and Development of a CPU Scheduling Algorithm Simulator” by Sukanya Suranauwarat is another perfect example of applications generating simulations for the CPU scheduling algorithms. It is a Java-based simulator that presents the concepts for scheduling for a single CPU with the use of animations. This application simulates First-come First-serve (FCFS), Round Robin (RR), Shortest Job First (SJF), Priority scheduling and Multilevel Feedback Queues (MLFQ) algorithms. This tool has some unique features. The developers have used realistic models for the processes, so that they can be easily configurable. They also used graphical animation to depict the process state at different steps of the algorithm making the understanding for the user much easier. Finally, they included a mode to test the knowledge gained by the students in making their scheduling decisions at various stages of algorithm simulation. (Suranauwarat, 2012)
2.2.1.4. **CPU Scheduling Simulation**

CPU scheduling simulation by Huda M. Salih is another example of using simulations to depict the concepts of CPU scheduling. This tool presents some of the scheduling algorithms in a visual way using graphical representation. The presenters of this tool selected pie chart to represent the different states in scheduling algorithm implementation. The slices in the pie chart represent all the processes in the simulation. The size of the slice represents the particular process time. The slice that is out of the graph represents the currently running process. There are also smart scheduling windows that give the list of algorithms and their description to select one. This tool also includes the options to create a thread, kill a process, block a process, changing the priority dynamically, suspending a process. This option enables the user to test their knowledge with experimenting. (Sailah & Fadil, 2009 )

2.2.1.5. **William Stalling Animations**

William Stallings animations depict the concepts using web applets. These animations cover most of the topics in the operating system course. The animations include process description and control, concurrency: mutual exclusion and synchronization, concurrency: deadlock and starvation, virtual memory, memory management, I/O management, uniprocessor scheduling, disk scheduling. All the concepts are explained using animations with start and pause options. These animations illustrates the process scheduling by depicting the different states in the life cycle of a process and various algorithms to schedule. There are no options to configure the process
related parameters to view the desired animation. The animations lack the user interactivity. (William, 2000)

2.2.1.6. **OSCAR**

OSCAR is a big online repository to view the simulations in different fields. OSCAR stands for the open source courseware animations repository. It includes different animation and simulations from different fields like biology, biochemistry, chemical engineering, computer science, earth sciences, civil engineering, mechanical engineering, and physics. Computer science animations include bubble sort, merge sort, insertion sort, linear search, mobile internet protocol, I-TCP simulation, process scheduling simulation, stored linked lists, stack and queue. A graph illustrates the scheduling, in which the x-axis in the graph represents the scheduling time with dots that represent processes when switched. Gantt charts displays the scheduling order. There are no options to configure the simulation.

2.2.1.7. **SOsim**

SOsim is a simulator for operating systems education. With the visual facilities, it serves as a supplement to better learn and teach modern day operating systems course. It covers the concepts like process management, internal system structures, processor management, and virtual memory management. Some of the unique features include process control block visualization, ability to create processor bound and I/O bound process, traces all the process states systematically. (Maia et. al., 2005)
2.2.1.8. Summary

All the above simulators discussed use hypothetical processes to explain the scheduling algorithms. Moreover, none of them uses real processes to illustrate the scheduling. Therefore, the user cannot have an experience with real scheduling in an operating system. Hence, this thesis tries to include the visualization of the scheduling in a real operating system.

2.3. Related Real-Time Scheduler Visualization Projects

Several projects are available to simulate or visualize the scheduling of the tasks in real time systems. Some of them focus on business projects or production and therefore they are irrelevant to this study. The following section presents an overview of some of the projects.

2.3.1. Scheduler Visualization Projects

Among all the projects that focus on task scheduling generated by the operating system, only a few of them visualizes it. In general, any project executes the simulation first, and then visualizes the results in a graphical way. Since visualization is the primary focus of this work, the following sections provide an overview of some of the projects.

2.3.1.1. Paje

Paje is a unique tool that visualizes the scheduling of the parallel applications running on distributed systems. However, this tool does not have the capability to trace
the scheduling in the operating system in a distributed environment. Therefore, to generate a visualization for the scheduling, this tool requires an additional external tool, which can produce a trace of the scheduling of the application under consideration. Once the external tool generates the trace, Paje imports it and starts the simulation. Paje can reproduce the thread states, concurrent primitives like semaphores in the visualization. Paje development used Objective-C, and its architecture connects all the components in the data flow to make the extension of the Paje easy. (De Kergommeaux et. al., 2000)

2.3.1.2. **Jedule**

Jedule is a program that presents the current schedule graphically. This tool also requires an external tool to trace the scheduler activity before visualizing. This tool generates the visualization of the given in a custom extensible markup language (XML) format. This tool aims at creating the overview of the whole schedule trace of a parallel program. This tool development used Java language and published under GNU license. (Hunold et. al., 2010)

2.3.1.3. **Summary**

Some other tools like Visual trace explorer (ViTE) or gltracevis, which have the similar functionality as the tools discussed above. Among the tools discussed, some of them have interesting features like interactive graphical representation. However, none of them has the capability to trace the scheduling to generate the visualization.
2.3.2. Scheduling Simulation Projects

None of the schedule visualization projects presented in the previous section has the ability to produce the data needed for visualization by themselves. Therefore, the simulation projects gained importance as they can generate the data necessary and then simulate it. The following section will provide an overview of some of the projects.

2.3.2.1. GHOST

GHOST stands for general hard real-time oriented simulator tool. It is a real-time scheduling simulator. This simulator supports both soft and hard deadlines. It groups the tasks into classes and handles each of the class by its scheduler. There is a superior scheduler, which selects the class and invokes the corresponding class scheduler. It also supports no deadline and resource allocation protocols. It has some unique features. One of that is new scheduling policies can add to the simulator apart from the existing scheduling policies. New scheduling policies development uses C language. This tool can also generate the graphical representation of the scheduling. Drawbacks of this tool are it supports only the single core operations, and the schedule visualization lacks the detailed description of the information when compared to some other related projects. This tool can run in either event driven or timely manner. This tool implementatation uses C language and not open sourced to the public. (sensini et. al., 1997)

2.3.2.2. Vizzscheduler

It is one of scheduling visualization projects. LogP cost model uses this tool as a part of a framework to develop and evaluate the scheduling algorithms. This tool is
capable of generating the task graph diagram and Gantt chart for the scheduling of given task set by a scheduler under test. Java debug interface connects to the visualization system to the scheduler under consideration. The visualization updates whenever the visualization systems receive the breakpoint making the tool capable of generating the live view of the scheduling to control it. This tool development used Java language and not open sourced. Moreover, this tool has no maintenance since 2002. (Lowe & Liebrich, 2001)

2.3.2.3. MAST

MAST is a software that can analyze different real-time schedulers. This tool makes use of the theoretical approaches using appropriate heuristics to test the scheduling ability of a real-time scheduler for the given task set. This tool allows the creation and saving of the system mode in a special purpose text format. SIM_MAST is a tool that was to develop to add additional features. SIM_MAST simulates the behavior of the existing saved models, but cannot generate the graphical representation of the simulation. Both the tools development used Ada and GTKAda and published under GPL. (Harbour et. al., 2001)

2.3.2.4. STRESS

It is one of the first real-time simulation projects for scheduling. It mainly focuses on analyzing and simulating the hard real-time applications. This tool provides a closed simulation environment and has its programming language called as STRESS. This programming language allows the user to define semaphores, system configuration, and the network configuration. This tool allows a task to control other tasks in the
environment making it possible to set one's scheduler to test its scheduling ability. This a graphical user interface to present the graphical representation of the scheduling. One drawback of the tool is that it does not allow soft deadlines. (Audsley et. al., 1994)

2.3.2.5. **Schesim**

It is a simulator for real-time applications. This tool generated accurate results by allowing the user to describe the control flow of the tasks to be scheduled. It also allows defining the inter-task communication like shared variables. There is some single core, and multi-core algorithms are included in the tool to make the comparisons easy. It is open sourced. Therefore, inclusion of new scheduling algorithms is just the modification of source code. Inclusion of new algorithms needs the understanding of the whole architecture. This tool cannot generate the graphical representation of the simulation. The output of the simulation is a log file. This file is the input to other tools that can visualize the log files. This tool used Ruby for the development. (Matsubara et. al., 2012)

2.3.2.6. **STORM**

It is real-time multi-processor scheduling simulator. It supports multiple cores as well as shared resources. This tool used Java language for the development. Java classes in the tool represents all the components like cores, tasks, resources. So new elements implementation is just the creation of new Java classes. An XML file with Java classes name needs to specify the simulation. Then the simulator populates the list of events internally to generate the statistical data needed to create the graphical representation of the schedule. STROM also provides a graphical user interface with command line input
to control the scheduling. This tool is not open sources as well as does not allow commercial use. (Urunuela et. al., 2010)

2.3.2.7. **RTsim**

RTsim is a real-time task-scheduling simulator, with an intention to support teaching. This tool supports both single and multi-processor systems. Visualization uses gantt chart and statistical data to present the information. It includes a variety of scheduling algorithms and protocols. The source code is not available to the public. (Manacero, 2001)

2.3.2.8. **Realtss**

It is one of the real time scheduling simulators that is open sourced and focuses on research and teaching. This tool used TCL language for the development. This tool also allows the inclusion of new schedulers written in either TCL or C. It provides a GUI to configure the simulation but does not provide any graphical representation of the simulation. Realtss uses an external tool called Kiwi to display the graph of the simulation. Its license is under GPL and supports both soft and hard deadlines. (Diaz et. al., 2007)

2.3.2.9. **RTSSim**

RTSSim is a simulation framework that supports the embedded systems. Embedded systems mainly focus on time and resource usage. Each task is a C program that executes in the simulation environment. It does not include the multiple cores, and the only available scheduling policy is a pre-emptive fixed priority. Therefore, this does
not allow the change of the task priorities at runtime. This tool requires an additional external tool to generate the graphical representation of the simulation. RTSSim uses Percepio tracealyzer as the external tool. (Kraft, 2009)

2.3.2.10. Summary

Most of the simulation tools discussed above requires an external tool to generate the graphical representation after they perform the simulation. In general, any tool executes the simulation first, and then visualizes the results in a graphical way allowing the user to focus on particular details by pausing the animation at specific intervals. However, it is not an optimal way to visualize the scheduling because it requires expensive preparation to simulate and save the results. In additional user needs to specify his part of interest before the simulation is executed to get full details of a particular event.
3. **Design and Implementation**

This research develops a Java tool to explain the concepts of CPU scheduling both theoretically and practically. The application design consists of two modes. One mode simulates the theoretical concepts and another mode to show the practical implementation of the concepts. The two modes are:

1. Simulation mode
2. Real-time mode

![Flow diagram of the Tool](image-url)
3.1. Simulation mode

One of the aims of this thesis in addition to visualization is the development of a simulation to explain the theoretical concepts of Scheduling. Shannon defined Simulation as "the process of designing the model of a real system and then conducting experiments with the model for the purpose of understanding the behavior of the system and evaluating various strategies for the operation of the system"[shannon,1998]. Hence, the model is an essential part of the development of a simulator. A model is an abstraction of the system that contains all the necessary information to draw the inference for the simulation result. It should neither oversimplify the system nor make it too detail. If the model is too detailed, it will make the calculation too complex on which the quality of the results will depend. The objective of any scheduling simulator is to simulate arbitrary processes with random parameters. So each scheduling processes under consideration should simulate on its model with entities like resources, tasks and scheduling algorithms. For a typical simulation system to handle all these entities there is a necessity to define them in an understandable and precise way.

3.1.1. System Model

For the reasons discussed above, this research develops a meta-model that describes the simulation and its elements. The meta-model should be flexible to incorporate the future changes and requirements. This meta-model is also called as system model. The system model consists of multiple tasks, resources. In addition to them, scheduling policies (scheduling algorithms) are an integral part of any scheduling simulation model. The
model design should allow different scheduling policies. Hence, the scheduling policies are not contained within the system model but referred at run time.

An instance of a system model is the simulation model. A simulation model describes the correct number of tasks, resources with their particular parameters and properties. It acts as an interface between the visualization and simulation. It allows the addition of new elements or modification of existing elements in the simulation.

3.1.2. Tasks (Jobs)

A variety of parameters characterizes each task in the simulation. Some of the parameters are:

- Job number: Every task has a unique number, which is similar to the process Id.
- Arrival Time: The time at which a process arrives in the simulation to do work
- Burst Time: The time or CPU cycles a task needs to complete its execution with the use of CPU resource.
- I/O Time: The time a task needs to complete its execution with the use of I/O device resource.
- Priority: This field is used in the scheduling policies to select a particular task
- Finish Time: This field gives the information about the time at which a task has finished its execution and terminated.
3.1.3. **Resources**

A resource is an entity that only one task can access exclusively at a time. In this simulator, Central processing unit (CPU) is one of the resources that many tasks accesses. Usually, accessing a resource involves some protocols. Scheduling policies or algorithms are the protocols that select a task to access the CPU in a scheduling environment. There are different scheduling policies that depend on various parameters in assigning a task to the CPU. I/O devices are other resources that multiple tasks share. In this simulator, tasks access them on a First-come First-serve basis.

3.1.4. **Events**

A simulation generates many events during its execution. These events include the creation of the task, allocation of resources to the task, modifying the state of a task. All the events are graphically represented using the animations. Hence, events act as an interface between the visualization and simulation. The inclusion of code at specific locations in the simulator code allows the addition of new events.

3.1.5. **Simulation Parameters**

Simulation parameters like the scheduling policy that should be used to schedule in the simulation model and other simulation parameters are not included directly in the system design. However, they should be available in the same file to be available when the simulation model loads. Therefore, fields must be included for the user to preselect these parameters when opening the model file.
There is a parameter included for selecting the speed of the simulation. This parameter allows the speed up or slowdown of the simulation by multiplying with the timing values in the simulation model. The user can edit the scheduling speed parameter at any point during the simulation.

Simulation parameters also include a map that assigns every task a distinct color value. The visualization system uses these values during visualization to fill the particular task with the assigned color to distinguish it from other tasks in the simulation model. The color set is predefined.

3.1.6. Simulator

For the simulator to drive the simulation, it modifies the simulation model. Hence, Simulator is the key element of this tool. Therefore, it should be accessible in all the different classes. Its design is a singleton class, which means it does not allow the creation of more than a single instance at any point. This class access the information from all the other classes and objects to make changes during the simulation.

A simulation can be either static or dynamic. In a static simulation environment, all the objects and their parameters are predefined and are not allowed to change at any point in the simulation. Moreover, time has no role in a static environment. However, Time plays a crucial role in any scheduling environment. Therefore, the simulation has to be dynamic for simulating a scheduling environment.

There are two ways to simulate a dynamic environment either discretely or continuously. In a dynamic continuous simulation, all the events occur systematically without the need
of any interaction from the user. The main advantage of any continuous simulation would be the smooth visualization for each step in the simulation. However, this simulation has the disadvantage of fixed interval between steps, which makes it impossible to visualize the events with shorter duration. Whereas in dynamic discrete simulation, the execution pauses after every step awaiting the user interaction. One main advantage of such simulations is that there is fixed time interval between steps, which makes it possible for the user take his own time to understand the step executed and then go to the next. However, the discrete simulation takes more time to simulate when compared to the continuous simulations. Therefore, the development of this simulator considers a mix of both continuous and discrete simulation. The graphical user interface provides pause and resume buttons to pause and resume the continuous simulation at any point. The same simulation changes to discrete by clicking the next step button continuously to get the next stages of the simulation. The simulation transforms from discrete to continuous or from continuous to discrete at any instance during the simulation. The simulation time in the simulator is independent of the real time world time.

3.1.7. Simulation Thread

A single thread couples both the visualization and simulation to have a visualization for every simulation step executed. The simulation window class contains the simulator thread as simulateaction() method. This research implements this thread as a timeline event. There is a separate method named pauseaction() to implement the son. The implementation of this thread is private. It improves the encapsulation, and does not allow other classes to access this thread in an unintentional way. Even though the
simulation time is independent of the real world time, visualization is in synchronous to the simulation time. Therefore, whenever the simulation thread pauses, the simulation time is also paused. The simulation speed parameter introduced in the previous section enables the simulation time to accelerate or decelerate based on the value set by the user. The simulation thread is also responsible for the visualization. Therefore, after every simulation step, simulator invokes the visualization method to update the visuals on the screen. The visualization contains animations highlighting the changes in the queues.

3.1.8. Simulator Operation

The simulation window opens when the user selects the simulation mode button through the graphical user interface. As the GUI is implemented in JavaFX when the simulation window opens initialize() method is immediately called. This method initializes all the GUI components on the screen and local variables with appropriate values and states. The simulation starts with input from the user. The user can set the simulation options and his preferences by drawing the side panel. The user needs to specify his choice for the number of tasks and algorithm to simulate. The simulator can simulate tasks ranging from two to eight and can generate all the parameters for the tasks automatically. The user can also give his parameters to the tasks by selecting Add Own Data radio button on the GUI. On selecting the button, a pop-up window appears with fields for providing input parameters for the tasks. These parameters include arrival time, burst time, I/O time, and priority. Upon filling the fields with appropriate values, the user needs to click the add process button to save the task parameters. The simulation requires at least two tasks data to start the simulation. Finally, the user needs to save all his preferences and tasks data by
clicking the save button in the side panel. Upon clicking the save button, a call to the
saveAction() method inside panel control class will occur. This method creates the main
queue by calling MainQueue() class. Then data loads into the table and graphical
elements representing the tasks with their IDs populates onto the GUI. The simulation
starts when the user clicks the Simulate button. The simulation thread now starts
executing by calling the nextStep() method. This method intern calls the workstep()
method in the simulation class performing the next step in the simulation. The simulation
class updates the appropriate objects and their parameters like the queues, process states.
Tasks are moved from one queue to another queue by calling the addJob() and
removeJob() methods in the respective queues.

When there are jobs in the queues to be scheduled, the scheduler selects a process from a
list of waiting process based on the algorithm under simulation. Process selected by the
scheduler uses the CPU. WORKINCPU() method updates the process execution, by
subtracting a one for every step from the process burst time. When the process burst time
is zero, the scheduler checks whether the process has any I/O activity. If it does, then the
process state changes from running to terminated and added to termination queue or its
state changes from running to waiting and added to I/O queue. If the I/O device is free,
then the first process in the I/O queue gets the access. Process execution in updated by
calling WORKINIO() method, which subtracts one from the I/O time of the process for
every step. When a process finishes its execution on the I/O device, it either terminates
changing the state from waiting to terminate or added to the ready queue changing state
from waiting to ready. The simulation thread executes until there are no more tasks to be
performed by calling the nextStep() method continuously or discretely. When all the tasks
finish their execution, the finished field in the simulation class is marked as true to stop the simulation thread. During the simulation, every step updates visually.

3.1.9. Visualization

Visualization of simulation of CPU scheduling with different queues requires animation of the concepts for a better understanding. This research compared different visualization frameworks like Java Swing, AWT, and JavaFX. JavaFX is the appropriate over the stable and proven Swing framework because of the advantages it has. A shortcoming of swing include no binding concepts, API's are less consistent, and customization is difficult. However, JavaFX has many advantages that include –

- Cleaner API, which will be much easier to work.
- Built-in animation system for creating animations.
- Availability of CSS styling.
- Less code when compared to swing from the same work.
- The binding mechanism, which makes it simple to integrate the GUI components with underlying model.

Simulation mode simulates different scheduling algorithms as selected by the user. The frame consists of six areas as shown. Each area has a different purpose in the simulation. The main frame of this mode is as shown in the below figure.
Figure 6. Main Frame in the simulation mode

Figure 7. Memory Pane (Top left pane)

The Top left pane is the memory pane. This pane explains the different memories involved in the life cycle of a process. This pane has the secondary memory, main memory, CPU, and I/O device. All the programs and OS files reside on the secondary
memory along with other files. The number of programs that reside on the secondary memory is equal to the number of processes in the simulation. A program gets loaded into main memory as a process whenever needed. At any time, operating system files called kernel resides on the part of the main memory. The kernel is the key component in any operating system that manages all the activities. Kernel files include the memory management module, scheduler, dispatcher, I/O management module. When a program is ready to execute, an instance of the program called process appears in the main memory. Every process has a data structure corresponding to it called Process control block. This block has all the control information corresponding to a process. This process control block resides in the kernel. Therefore, the simulator design should be such that whenever a process enters the main memory, a process control block corresponding to that process should appear in the Kernel part of the main memory. Animation helps the user to understand, at what instance of the time a process loads into the main memory. CPU block in the area represents the CPU in the system, holding the currently executing process. When there are no processes to run, CPU is idle. The I/O block represents the I/O activity in the system. This block holds the process using the I/O device. When a process terminates, Scheduler removes its instance from the main memory. However, its corresponding program resides in the secondary memory.
The Top right pane is the Job Pool pane. It has the data corresponding every process in the simulation. The simulator generates the data for each process upon selecting the number of processes from the menu. The fields generated are Arrival time, Burst time 1, I/O time, burst time 2, Priority. The remaining fields updates for each step during simulation. There are fields to indicate the start time and finish time of a process. During execution, the percentage field shows the percentage of each process completed. Simulator calculates turn-around time after each process completes its execution. The Remain field indicate the remaining burst time of a process. The simulator also accepts the manual input from the user. The fields in the Job pool updates for every step providing the user with useful information.

![Job Pool Table](image-url)
Figure 9. Queue Pane (Middle Pane)

The Middle pane in the simulator frame shows different queues through which each process migrates during execution. The job queue has all the processes that are present in the simulation that are not yet ready to execute and their current state is "NEW." Once they are ready, they move from the job queue to the ready queue and their state changes from "NEW" to "READY." All the processes that are ready for execution are present in the ready queue. The scheduler selects the appropriate process among the processes that are in the ready queue when it finds the CPU is idle. The process scheduled to run appears on the CPU module in the area. The Scheduler decision for selecting a particular process corresponding to an algorithm will appear in the information pane. When the burst time of a process completes and if it has an I/O event, then it enters the I/O queue to be scheduled. When a particular I/O device is free, the process that is at the first of the I/O queue gets a chance. When the I/O activity finishes, the process enters the tail of the ready queue and waits for its chance to get scheduled on CPU. When all the processes terminates, they enter the terminated queue and simulation finishes. There is a timer at the top right corner of this pane to show the simulation time, which increases by one after every step.
The last but one pane is the Gantt chart pane. It shows the Gantt chart for the simulation with a different color to each process. Simulator populates the pane for every step with a block with a number indicating the currently executing process. When the CPU is idle, a white block appears stating that no process is running. The Gantt chart for a simulation is as shown in the below figure.

The top middle pane is the information pane. It gives information about the changes and decisions made by the scheduler. Changes include process migration from one to another, scheduler decision in selecting a particular process, the current process executing, the process using the I/O device, pre-emption activity, process arrivals and termination, time slice expiration. This pane updates the information for every helping the user to gain better insight into the simulation.

The Bottom area has the controls for the simulation. These controls include simulate button, pause button, next step button, back step button, help button and a slider to control the simulation speed. The user can simulate either in continuous or discrete manner. To run continuously, the user needs to click the simulate button and adjust the simulation speed. There is an option to pause the simulation at any point by clicking the pause button and restarted using the simulate button. To run the same simulation in a discrete way, the user needs to click the next step button to get the next step. The user can
go back in the simulation by just clicking the back step button. Simulation changes from
discrete to continuous to discrete at any point. The help button opens a new window
giving the steps to execute the simulation. This window also presents some of the
concepts related to CPU scheduling.

In addition to the areas mentioned above, a side panel provides the simulation
options. This area includes the options to select the number of processes, scheduling
algorithm and option to add own data for simulation. The user can choose 2 to 8
processes for the simulation. Next, he needed to select the algorithm he wants to simulate
from the list of available algorithms. Algorithms include First-come First-serve, Round
Robin, shortest job first, shortest remaining time first, Priority scheduling with pre-
emption and without pre-emption. It is better to view the simulation at a slower speed to
have a better understanding of the changes. The user can choose between auto-generated
data or enter his values for the process data. To add own data, the user needs to click add
own data radio button. This pop ups a new window with fields to add data. These fields
include Burst time1, I/O time, Burst time 2, priority. After giving the appropriate values,
clicking the add button submits the process data. The user needs to input data at least for
two processes. After populating the processes, clicking the save button sends the data to
the simulator. Then it fills the data in Job Pool and creates the process components in the
simulation. Process data generates automatically, when the user selects the 'auto generate'
radio button. The side panel also has the buttons to restart and to reset the simulation
options to default values.
3.2. Real-Time Mode

This mode presents the user with a chance to experience the real processor scheduling that takes place in a Linux environment. The aim of this mode is to visualize the process scheduling. The design of this mode consists of three phases.

The first phase involves developing the backend to interact with the Linux kernel for getting the required scheduling information. The first step in this process is to determine the system resources and programming constructs responsible for carrying out the scheduling activities in the Linux operating system. After determining the required system resources that were necessary to gather the scheduler information, research proceeded to find if any existing software could provide similar functionality. One finding is the "procps." This tool provides the users with up to minute statistical
information regarding process statistics, priority, status and other fields. The user can interact with this tool using the command line. Of the vast information provided by the tool, only a few key attributes are of interest to the development of this application. Moreover, it does not cover some of the required information like scheduling order among the processes and system calls. Further research claims that tracing some of the trace points built into the kernel leads to the access of scheduling information. It forced further research to find the tools that are capable of tracing the tracepoints in the Linux kernel. There are some tools available like eBPF, Dtrace, System Tap. All these are capable of tracing the scheduling activities, but one shortcoming of these tracers is that they are running in the user space creating an overhead. Moreover, there is a necessity to install these tools on the host computer before using them to trace. It led further research to find the suitable software that does not create any overhead. The result of the research is Perf_events, which is the main tracing tool that many of Linux administrators use. Its source is available in the Linux kernel and can be easily included using the Linux-tool-common package. This tool traces and dumps the output to a file called "perf.data", which it does dynamic buffering and post processing of the data is required. A portion of the development of the backend requires studying the kernel source code to find the tracepoints that can trace necessary information for scheduling activities. Next, the research proceeded to observe the use cases of the perf_events tools to use it as per the requirement to generate the required data to make visualization. Tracing the Sched_switch tracepoint can give information about the scheduler activity.

After being able to generate the required information, the design effort shifted to the second phase. It starts with creating the initial sketch to determine the look and feel
of the application for user interaction. The sketch has been modified several times to find the perfect solution that provides a balance between the following three aspects.

1. Ease of use it provides to the user
2. Implementation flexibility
3. Capability coincidence with the graphical user interface development library.

Once the sketch achieves the optimal balance, the next step is to place the graphical components that interface with the user, on the application as a design step that awaited integration with the back end functionality.

The interface design has a motive to present the users with an easily understandable and interactive display that graphically presents the real scheduler behavior in the Linux kernel. The real time mode is further divided into three subsections.

1. The first subsection allows the user to view the scheduling order among the current running background processes on the Linux for a fixed time interval on all the cores. The interface in this subsection is so simple that, the user just needs to click the generate button to display the scheduling order. All the blocks in the graph represent the processes, where the height and length of any block have no significance. With this, the user can know the different processes that are running in the background.

2. The second subsection allows the user to select a set of processes from the predefined set of processes and then execute them to observe the order in which the scheduler schedules them. Designing of this subsection is a two-step process. The first step is to collect the input from the user, and the next one is to display
the scheduling activity for the execution instance generated by user input. The process set has different severity levels imposed on the CPU among various entities. The ones present in the predefined set are:

- High Processor Bound process
- Moderate Processor Bound process
- Low Processor Bound process
- High I/O Bound process
- Moderate I/O Bound process
- Low I/O Bound process

3. The third subsection allows the user to run their specific programs and observe the scheduling order. It also presents some interesting features like memory allocated, and time elapsed to complete execution. It helps the student to write their specific programs and trace their behavior, which helps in increasing the conceptual understanding of a user practically.

The third phase is the final phase of the development, which involves associating and verifying the graphical interface with proper back-end functionality. It involves creating paths for input to travel from graphical interface to the underlying backend and correspondingly a connection from the backend to the graphical interface to show the output of execution.

The overall design of the real time mode is as shown in the following figure:
The first step is to start the tracing. After the application gathers the input from the user, the process set population and information retrieval process takes place. The functions and the procedures executed in the second step belong to the group that performs the tasks required for retrieving and analyzing the CPU resource allocation data. The output of the first stage is the record of scheduling activities, which is the input to the second stage. The second stage generates a graphical visualization of the data recorded by making necessary comparisons.

3.2.1. Tracing the Scheduler activity

There are many tools in the market to provide the list of currently active processes in a Linux operating system. However, none of them is capable of visualizing the scheduling activities. To know the scheduling, one should trace all the processes on the target host system. It requires the addition of tracepoints to the kernel functions at appropriate locations. Whenever the execution enters the marked function, it will send the required information to the user space.

Linux kernel developers developed a tool ‘perf’ to measure the system performance. This thesis makes use of this tool to trace the scheduler switch activity. Perf
can record the trace of any activity using the record command. The different commands and options available in the tool are as shown in the below figure.

Figure 13. Commands in Perf Tool

Many trace points are readily for use in the Linux kernel. One can know the scheduler activity by tracing the Sched_switch tracepoint. 'Sched' is one of the commands available in the Perf tool to trace the whole scheduler activity. Hence, the combination of Record and Sched commands can trace the sched_switch tracepoint. Scheduler switch information is not sufficient to know the full scheduling activity. Command to trace the full scheduler activity is

- Perf record -a -e sched

Figure 14. Tracing the scheduler switch tracepoint
The option -a specifies that the trace is for all the cores and the option –e is for selecting a particular command in the perf tool. On executing the above command, perf tool starts tracing and continues until the user uses the break command (ctrl+Z). Once the trace completes, it will generate an intermediate file called ‘perf.data.’ This file is a binary file, which is useless until converted to a useful format. Perf provides three commands to convert the binary file into an understandable form. The three commands are Annotate, Script, and Report. Of the three commands, the script is the only command that presents the scheduling activity along with time instances. The output for sample trace is as shown in the below figure.

![Figure 15. Output of Perf script command](image)

### 3.3. Implementation of Real-Time mode

This section presents the implementation of the real-time mode in the tool. The real-time mode has three subsections, and all the three have a similar implementation that differs only in some aspects.

Required information to generate the visualization will be traced using the ‘Perf’ tool as explained in the previous section. However, Perf is a command line tool that
interfaces with the user through the command line. Hence, it will become a two-step process for the user to trace the scheduling activity using Perf and then give the output as input to the visualization system in the tool to generate the plot of the scheduling activity. The tool designed in the thesis will create script files dynamically with necessary commands to trace using the Perf tool to avoid the two-step process. The x-term application executes this script files in the background and then caches the output to text files for later use by visualization system.

3.3.1. Tracing the Background Processes

This subsection of the real time mode presents the user with scheduling of the currently executing processes in the Linux system. The first step is to create the script file with necessary commands to trace the scheduler activity. Script file will be created dynamically, whenever the user clicks the ‘Trace' button in the interface. The script file generated is ‘Wholetrace.sh,' which contains the following commands.

```
#!/bin/bash

perf sched record sleep 0.5

perf sched script>script.txt

perf sched latency>latencywhole.txt
```

The first command records the scheduling activity for an interval of half a second. The tracing time is limited to half second to simplify the visualization. The second command creates a text file named ‘script.txt' that contains the output of the script
command in the Perf tool. The third command captures the latencies into the file ‘latencywhole.txt.’

The next step in the implementation is to make the script file executable in the host system. It is achieved using ‘chmod’ command. The command executes in the background using Process Builder concept in Java, which creates parallel processes in the background. The X-Term application executes this command.

After making the script file executable, next step is to execute it. Executing the commands in the script files requires ‘sudo’ privileges from the user, which in turn requires system password from the user. This tool pop ups the X-Term application to get the sudo privileges from the user. Once, the user enters correct system password; script file executes creating the required text files as explained above.

Next step is to process the information in the text files to generate the visualization. The visualization system processes the text in the file line by line and adds all the processes in the trace to a set. There are two more sets created in parallel, one for holding the process ids and other for holding the colors associated with each process. Next step is to generate the visualization, which requires processing the ‘script.txt’ file one more time to get the context switches information. For every context switch found in the file, a block representing the process switched adds to the graph in the visualization. The following section explains the different Elements in the visualization.
3.3.2. Tracing the Pre-defined set of Processes

Implementation of this subsection will be similar to the previous one, except that the trace is for a pre-defined set of processes instead of all the running processes in the host system. To observe the scheduling order among a set of processes, they must run on the same core. It requires the use of an additional tool called ‘Taskset,' that can populate a process on the particular core as selected. The usage of the tool is:

- `taskset -c core processname`

Taskset is a command line tool that interfaces with the user through the command line. The above command populates a new process with a particular core preference. The `-c` option in the command specifies the core preference. For example, to execute a process named Firefox on core 2 of the system, the command is:

- `taskset -c 2 Firefox`

This subsection also provides the user a chance to observe the scheduling under different scheduling policies available in the Linux distributions. No interface is readily available in Linux distributions to switch the scheduler between the various scheduling policies available. It further pushed to research to find a way to switch between policies from the Java software developed itself. The result of the research is that, if a process is scheduled using a particular scheduling policy, all the child processes it forks will be under the same scheduling policy as of the parent. It further pushed the research to find a tool that is capable of scheduling a particular process under a specific scheduling policy. The finding of the research is the tool named 'Sched tool,' that is capable of scheduling a process
under a particular policy. The tool installation of the tool is easy and requires the use of some commands to switch the scheduling policy. The command used for switching is:

- `schedtool -N $`

The above command executes within a process to change the scheduling policy of that process. The `-N` option in the command selects a particular scheduling policy. There are multiple values of n corresponding to each scheduling policy.

- N=0, for SCHED_NORMAL
- N=1, for SCHED_FIFO
- N=2, for SCHED_RR
- N=3, for SCHED_BATCH

To schedule a process under SCHED_FIFO or SCHED_RR, we need to specify the static priority for the process using `-p` option followed by priority (0 to 99). The X-Term application populates all the processes in the set. Therefore, the schedtool schedules the X-Term application under the required scheduling policy. For example, to schedule under the SCHED_FIFO scheduling policy, the command is:

- `schedtool -1 -p 99 $`

This subsection generates a script file with all the commands to switch the scheduling policy and populate the processes on the particular core when the user clicks the save button in the graphical user interface. When the user selects a scheduling set with a high
processor bound process and a moderate I/O bound process, and the scheduling policy as
SCHED_FIFO, the commands in the script file generated are:

```bash
#!/bin/bash

schedtool -1 -p 99 $$

perf sched record sleep 3

taskset -c 2 ./hp &

taskset -c 2 ./mi &

ps>file.txt

sleep 5

perf sched script>trace.txt

perf sched latency>latencywhole.txt

pkill -TERM -P ${$}
```

In this subsection, the tracing of the scheduling activity will take place for three seconds. The `ps` command executes within the parent process to list the child processes that it forks. So, the `ps` command executes within the X-Term application after all the required processes are populated. A temporary text file named ‘file.txt’ caches the output of the `ps` command. This file contains the process Ids of the processes that are forked. To filter the processes of interest from scheduling trace obtained, visualization system uses this information.
Similar to the previous subsection, once the Perf tool completes the recording of scheduling activity, next the script command executes to cache the output to 'trace.txt' file. The script file includes the pkill command at the end to kill all the processes that are forked. If not, they will be left as Zombie processes, running continuously. Next step is to process the 'trace.txt' file to filter the processes set of interest using the process ids in 'file.txt' file. Next step is to make comparisons to generate the visualization. The next section will provide the full details of the visualization.

3.3.3. Tracing the User-defined set of Processes

The implementation of this subsection is similar to the previous subsection, except that the trace is for the user-defined set of processes instead of the pre-defined set of processes. This subsection also generates an intermediate script file to execute the commands in the background. A sample script file created for running processes named Firefox and Chrome on the second core of the CPU under SCHED_RR scheduling policy is as follows:

```bash
#!/bin/bash

schedtool -2 -p 99 $$

perf sched record sleep 3

taskset -c 2 ./Firefox &

taskset -c 2 ./chrome &

ps>file.txt
```
The next steps in the implementation will be same as that of the previous subsection. The next step is to generate the visualization, once it generates the necessary text files.

3.3.4. Visualization of Real-Time Mode

This research visualizes the real time mode also using the JavaFX framework. This mode comprises of three subsections, whose visualization frame looks similar. However, the frame for selecting the options to trace will differ.

3.3.4.1. Visualization for the Trace of Background Processes

The first subsection visualizes the scheduling of the background processes that are currently executing. It starts by just clicking the Trace button in options frame. After getting the required trace information, a frame with scheduling visualization will appear as shown in the following figure.
Figure 16. Visualization of a sample trace of Background Processes

The frame consists of a scrollable graph with the scheduling order. The Y-axis represents the Cores available in the system, while the x-axis represents the time of the trace. The blocks in the graph represent each process with a unique color corresponding to the process. Using different colors for each process helps in distinguishing one process from the other in the graph. The information panel beside the graph pane presents the color associated with each process. This subsection neglects the runtimes of processes and considers them as one unit to simply the visualization. The blank spaces represent the idle time on the graph. The information pane gives information about the Process Id, priority, Process tag and color. The pane in the bottom provides runtime information of every background process along with average delay, maximum delay and context switches. The bottom left area contains information about the total runtime, total ideal time, total trace time and total context switches during the trace.
3.3.4.2. **Visualization of the trace of the Pre-defined set of Processes**

Visualization in the second subsection looks similar to the first subsection, except that it is the trace of scheduling activity is for the predefined set of processes selected by the user. An options frame contains a list of processes that are available for the user to select. The options frame is as shown in the below figure:

![Frame to select options for the scheduling](image)

**Figure 17.** Frame to select the options for the scheduling

This subsection restricts the maximum number of processes to schedule to six to simplify the visualization for better understanding. The user needs to click on the process of his interest to add it to the list of processes to be scheduled. The processes selected by the user will appear in the selected processes window for reference. User can add same process multiple times to list by just clicking its corresponding button multiple times. The
user can delete any process from the list by clicking Delete last process button. He can also reset the selection made by just clicking the Reset button and start the selection again. All the processes in the set must run on the same core. Hence, the options frame provides the user with an option to select his core of interest from the list of available cores. The top right area in the options frame allows the user to select one of the scheduling policy from the list of available scheduling policies. The list of policies for selection is SCHED_FIFO, SCHED_OTHER (SCHED_NORMAL), SCHED_RR, and SCHED_BATCH. Scheduling policy selected is pre-set to SCHED_OTHER, as it is the scheduling policy that the scheduler in Linux OS uses. Core selected is pre-set to first core "Core0". The minimum number of processes to select for scheduling is two. Once the appropriate number of processes are added to the list of processes to be scheduled, the user needs to save his preferences by clicking the ‘SAVE’ button. Saving the preferences generates the required script files and processes source code files in the backend. The trace starts once the user clicks the "Run&Trace" button. The trace will last for two minutes and once it completes the visualization frame pop-ups. The frame for the visualization is as shown in the below figure.
Figure 18. Visualization of a sample trace of Pre-defined set of Processes

The frame consists of a scrollable plot with scheduling order among the predefined set of processes. The plot is an endless list of CPU bursts with idle time in between them. In the graph, Y-axis represent the processes traced, while the x-axis represents the time of the trace. A block represents each process under consideration with a unique color to distinguish from others. Block height has no significance, while its length represents the runtime of the corresponding process. Graph presents all the processes in parallel to make it more understandable. When a process gets its turn, all other processes will be in waiting state. The blank spaces represent the idle time in the process lifetime. The information pane gives information about the Process Id, priority, Process tag and color. The pane in the bottom provides runtime information about every background process along with average delay, maximum delay and context switches. The bottom left area contains information about the total runtime, the total ideal time, the total trace time, the total context switches, core and algorithm in the schedule. The user can
schedule the same process set using different scheduling policy and compare them to understand the effect of scheduling policy on scheduling. The user can also schedule different processes with different behavior to observe the fondness of scheduler.

3.3.4.3. **Visualization of Trace of the User-defined set of processes**

Visualization in this subsection is same as the second subsection, except that it is for user-defined set of processes. The options frame is also similar, except it does not list the predefined set of processes. Instead, it got "Choose File" button that opens the file selector window to choose the file to be scheduled. All other options are same for selecting the core, scheduling policy. This options window, File chooser window, and visualization for a sample trace is as shown in the below figures.

![Figure 19. Frame to select the User-defined set of processes](image-url)
A different color is set to each process to make the distinction easier. The interface allows the user to remove any selected process from the list before execution starts. Once the tracing completes, a graph will appear on the screen that presents the observed scheduling activity of the CPU on the observation set selected by the user. The information is presented using horizontal bar chart (Gantt Chart) where each block length represents the time elapsed by them on CPU. Visualization system assigns each process tag with a unique color in the display chart area to reduce the visual strain in distinguishing one process from the other process in the bar chart. There is also statistics window showing the some of the features like the total number of context switches during the time elapsed and time allocated to each process in the observation set.
4. Testing and Results

In software engineering, validation and tests are the vital aspects that should be done to any new tool developed. Validation is necessary to show that the tool works as expected, while tests are necessary to find the defects in the system designed. Hence, there is a necessity to prove that a system is reliable and stable. This section provides the validation and tests conducted. This research tests this tool at different layers, starting from unit test to a comprehensive system test.

4.1. Development Testing

Development Testing is the process of testing during development phase itself to find errors and bugs. Usually, developers and programmers carry out this testing during the development phase. There are three phases in Development testing: Unit Testing, Component Testing, and System Testing.

4.2. Unit Testing

Unit Testing is the process of testing program component and their functionality. The first step is to test the functions in the code by calling them with variety sets of parameters that cover all the cases and then comparing the result with the expected result.
Next step is to test the objects by testing all of its methods by modifying the attributes repetitively to put the object in every possible state during the simulation. This type of testing is called White-Box testing. The White-Box testing will have access to the source code and the programs internal structure.

4.3. Component Testing

Any software component is usually an integration of several individual components to provide a particular functionality. These components consist several individual interfacing objects and functions. Therefore, component testing tests the functionality given by a software component.

The simulator in the tool developed is tested for every particular functionality it provides during the development phase. The following section provides some of the functionality tests.

4.3.1. Component testing in Simulation mode

The implementation of Main Queue in the simulator requires the function fill() in the Queue class to Linux. This function is responsible for creating Jobs in the simulation and tested to check its functionality, i.e. whether it creates the correct number of Jobs with appropriate attributes. The implementation of table content class requires several functions in the Job class to a total number. These functions are responsible for updating the several attributes a Job for every simulation step executed and tested with different sets of attributes to affirm its right functionality.
Next step is testing various scheduling algorithms. In the implementation, the function nextstep() is responsible for updating the Ready Queue, Terminate Queue, I/O Queue. This function is also responsible for selecting the next Job based on the algorithm. All the algorithms implementation also has the method named nextstepIo(), which is in charge of selecting the next Job to be executing the I/O operations. These functions are tested with different jobs with different attributes to check whether the job selected same as the expected theoretical result.

4.3.2. Component Testing in Real-Time mode

The implementation of the second subsection in real time mode requires the execution of tasks on a particular core, which uses 'Taskset' tool. This research schedules different processes on different cores available to test the performance of this tool. The implementation also requires the switching of the scheduling policies. Therefore, switching the scheduling policy of the parent process(X-Term) achieves this, and then it forks the necessary child processes to check the correctness of switching.

All the processes in the set populates as background process before starting the trace. Each process in the pre-define set has a different functionality, and this research tested them separately. The last step is to test the visualization of the Scheduling activity in all the three subsections of the real time mode with different sets of processes to check correctness of the tool.
4.4. System Testing

During the development, once all the software components are integrated together, the next step is to perform the system testing. It is the last phase of development testing, also named Black Box testing. It does not have any access to the source code and internal structure. The main goal of System testing is to check that the components are correctly integrated, compatible with each other, interacting appropriately and to test that they are transferring correct data at the right time.

Test cases are written to check the correctness of the simulation steps, preconditions, inputs, post conditions and expected outputs. The completion of the test begins the comparison of the output of the simulation with the theoretical result. If the output is different from the expected output, the code is analyzed to find the errors and then source code is modified to fix the errors.

The system testing is also done for the real time mode to check the correctness. The second and third subsection of the real-time mode is tested with different sets of processes to check the output under different scheduling policies.

4.5. Results

This section will provide the results obtained in the simulation and real-time mode implemented in this tool.
4.5.1. Simulation Mode

The simulation of eight processes scheduled under Round Robin scheduling algorithm at some random instances are as shown in the following figures.

Figure 21. Simulation Window before the start of the simulation
Figure 22. Simulation window at 10th step of the simulation

Figure 23. Simulation window at 35th step of the simulation
Figure 24. Simulation window at 65th step of the simulation

Figure 25. Simulation window at the end of the simulation
4.5.2. Real-Time Mode

This section will provide visualization of the scheduling generated in the three subsections of the real-time mode.

The first subsection provides the trace of the processes that are currently running. The visualization generated at random instant is as shown in the figure.

![Visualization of a sample trace of the first subsection of the Real-Time mode](image)

The second subsection provides the trace of scheduling activity for the predefined set of processes. Graph created for the process set containing six different processes (High Processor Bound, Moderate Processor Bound, Low Processor Bound, ...
High I/O Bound, Moderate I/O Bound, and Low I/O Bound) scheduled on Core1 of the CPU under scheduling policies is as listed below.

Figure 27. Visualization for the set with different processes under SCHED_NORMAL Scheduling Policy

Figure 28. Visualization for the set with different processes under SCHED_FIFO Scheduling Policy
Figure 29. Visualization for the set with different processes under SCHED_RR Scheduling Policy

Figure 30. Visualization for the set with different processes under SCHED_BATCH Scheduling Policy
One can observe from the above figures that scheduling differ for the same set of process under different scheduling policies. Here, the process set contains different processes with different severity levels imposed on the CPU. The visualization under SCHED_NORMAL and SCHED_BATCH scheduling policies reveal the favoritism of the scheduler for some processes in the set. The processes it favors are the High I/O Bound and Low I/O bound processes. Visualizations also reveal that time slice allocated to process differs under different scheduling policies. The scheduler allocates big time slices under SCHED_RR scheduling policy. As all the processes in the set are infinite loop processes, they execute continuously until terminated. Therefore, visualization under SCHED_FIFO scheduling policy reveals that only one process in the set executes, while others still wait to get their turn. Visualizations also infer that time slices allocated under SCHED_NORMAL and SCHED_BATCH are almost similar. As the processes in the set are different, each process gets a different time slice, when compared to other processes. Fields at the bottom of the visualizations show that the number of context switches for each process are less in SCHED_RR scheduling policy, when compared to other policies for the same period of trace.

Next, Visualizations created for the process set with the same process (three of the Moderate Processor Bound processes) under different scheduling algorithms is shown in the below figures.
Figure 31. Visualization for the set with same processes under SCHED_NORMAL Scheduling Policy

Figure 32. Visualization for the set with same processes under SCHED_FIFO Scheduling Policy
Figure 33. Visualization for the set with same processes under SCHED_RR scheduling policy

Figure 34. Visualization for the set with same processes under SCHED_BATCH scheduling policy
The above visualizations reveals that when all the processes in the set are same, scheduler shows equal favoritism to all the processes. One can observe that each process in set gets an equal time slice to execute under all scheduling policies. However, SCHED_FIFO scheduling policy is a different case, where only one process executes because all the processes in the set have an infinite loop of execution.

Visualizations in the third subsection of the Real-Time mode will be similar to those of second subsection as presented above. Therefore, they are not included here.
5. Conclusion and Future Work

5.1. Conclusion

The purpose of this research was to develop a prototype for an Interactive Tutoring System that teaches the CPU Scheduling concepts in an operating system. This project considered the drawbacks in the existing CPU simulators and the currently used CPU tutoring systems to design the architecture and the features for the system. The primary motivation of this thesis is the need for bridging the gap between the theoretical concepts of CPU Scheduler and the practical implementation in an interactive way.

The initial chapters (1-2) of this document briefly introduced the concepts of CPU Scheduling in an operating system. The first chapter explained the motivation, scope, purpose and the significance of this research. Further, the discussion continued with a brief analysis on the existing literature available on the existing CPU Scheduling simulators and CPU tutoring tools. This second chapter mainly presents the various Linux monitoring tools and performance measuring tools, which are useful to trace the kernel activity. This study reviewed the behavior of the present real-time visualization systems and applied their working patterns in designing this tool.

The chapters (3-4) provided the details of the design, implementation and the test execution results of the software. Chapters 3 described the design architecture of this tool
and represented it in a flow chart. The tool involved two different modes of operation i.e. the Simulation mode and the Real-time mode. The simulation mode created an environment to simulate the different CPU Scheduling algorithms in theory, whereas, the real-time mode devised a way to visualize the scheduling in Linux operating system. Later, this chapter addressed the implementation of this software, which used JAVA programming for the development and the JavaFX framework for the visualization. Chapter 4 elucidated the various testing methods such as Unit testing, Component testing and System testing involved in defining the quality of the tool. This chapter also included the screenshots of the results at important steps of the simulation followed by a crisp explanation. Further, this chapter also had the results for all the three sub-sections of the real-time mode, which demonstrated that the libraries and the application satisfied the specifications and the initial requirements.

The rest of this chapter addresses the significant future improvements using this “Sched–ITS” tool, and the underlying user experiments.

5.2. Future Work

To increase the effectiveness of the tool, at this point, the application requires feedback from the users to improve the features of the tool. A detailed strategy to obtain the user feedback involves defining the Questionnaire to user, Selection of the Participants, Testing Procedure followed by the analysis. The Sched-ITS tool may also have an additional “practice” mode to enhance the understanding by guessing and validating the output of the future steps in the execution. This tool only covered the visualization of
scheduling in only one real operating system i.e. Linux. However, Windows OS being a widely used operating system, there is need to extend the usability of this tool to work in the Windows environment. This project can further include the other concepts in the operating system course like File Systems, Disk Management or Virtual Memory, which involves simulation, visualization and tracing in Linux OS.
Bibliography


