ASYNOPSIS ON

CUSTOMTASKSCHEDULERSIMULATOR

Submittedinpartialfulfilmentoftherequirementfortheawardofthedegreeof

BACHELOROFTECHNOLOGY

In

ComputerScience&Engineering

Submittedby:

AkshitDumka	**2261081**
DiyanshRana	**2261203**
PiyushKumar	**2261419**
SirajMehra	**2261542**

UndertheGuidanceof

Prince Kumar

Assistant Professor

ProjectTeamID:81

DepartmentofComputerScience&Engineering

GraphicEraHillUniversity,Bhimtal,Uttarakhand

March-2025

CANDIDATE'SDECLARATION

We hereby certify that the work which is being presented in the Synopsis entitled **"Custom Task Scheduler Simulator"** in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science & Engineering of the Graphic Era Hill University, Bhimtal campus and shall be carried out by the undersigned under the supervision of **Anubhav Bewerwal, Assistant Professor**, Department of Computer Science & Engineering, Graphic Era Hill University, Bhimtal.

Akshit Dumka	**2261081**
DiyanshRana	**2261203**
PiyushKumar	**2261419**
SirajMehra	**2261542**

The above mentioned students shall be working under the supervision of the undersigned on the **"Custom Task Scheduler Simulator"**

Signature	Signature
Supervisor	**HeadoftheDepartment**

InternalEvaluation(ByDPRC Committee)

StatusoftheSynopsis: Accepted/Rejected

Any Comments:

Nameof theCommitteeMembers: **Signaturewith Date**

1.

2.

TableofContents

ChapterNo.	**Description**	**PageNo.**
Chapter1	IntroductionandProblem Statement	4
Chapter2	Background/LiteratureSurvey	6
Chapter3	Objectives	8
Chapter4	HardwareandSoftware Requirements	9
Chapter5	Possible Approach/Algorithms	10
	References	14

Chapter1

IntroductionandProblemStatement

1. Introduction

Efficient process scheduling is at the heart of every modern operating system. As computer systems are expected to handle multiple tasks simultaneously, the way the **CPU schedules and allocates time to processes** becomes critical in determining system performance, responsiveness, and user experience.

In a typical multitasking operating system, several processes compete for CPU time. The **CPU Scheduler** decides the order in which these processes are executed based on specific criteria, such as arrival time, burst time, and priority. Over the years, multiple CPUscheduling algorithms have been developed, each optimized for different performance goals. Common examples include **First-Come, First-Served (FCFS)**, **Shortest Job First (SJF)**, **Round Robin (RR)**, and **Priority Scheduling**.

However, understanding these algorithms through theory alone can be challenging. Students and beginners often struggle to grasp the impact each scheduling method has on metrics such as **waiting time**, **turnaround time**, **response time**, and **CPU utilization**. This is where our project, the **Custom Task Scheduler Simulator**, becomes extremely useful.

The **Custom Task Scheduler Simulator** is an interactive and educational tool developed to simulate the working of various CPU scheduling algorithms. It allows users to define their own set of processes, specifying attributes like **arrival time**, **burst time**, and **priority**. Once the data is input, users can choose from multiple scheduling algorithms and observe howthese processes are executed over time.

One of the key features of the simulator is its ability to **visually represent process execution using Gantt charts**, making it easier to understand how each algorithm schedules tasks. It also computes and displays important performance metrics for each process and the overall system, allowing for easy comparison between algorithms.

Theprojectisespeciallybeneficialfor:

- **Studentsandeducators**,asalearningtooltovisualizeandcomparescheduling methods.
- **Developers and researchers**, to analyze the behavior of scheduling algorithms under different workloads.
- **Anyone interested in operating system design**, looking to understand the real-time impact of CPU scheduling.

In summary, the Custom Task Scheduler Simulator bridges the gap between theoretical understanding and real-world application. By providing an intuitive and practical platform to experiment with scheduling algorithms, the simulator helps users gain a deeper and more hands-on understanding of core operating system concepts.

2. ProblemStatement

Modern operating systems are designed to handle multiple processes running simultaneously. One of the key responsibilities of an operating system is to **efficiently schedule processes on the CPU**, ensuring that all tasks are executed fairly and within an acceptable time frame. To achieve this, various **CPU scheduling algorithms** are used, such as **First-Come-First-Serve (FCFS)**, **Shortest Job First (SJF)**, **Round Robin (RR)**, and **Priority Scheduling**. Each algorithm is suited to different types of workloads and comes with its own trade-offs in terms of performance metrics like **waiting time**, **turnaround time**, **response time**, **throughput**, and **CPU utilization**.

Despite the importance of these concepts, students and professionals often find it difficult to fully understand and compare scheduling techniques using only theoretical explanations. Manual calculations for scheduling tasks can be time-consuming, error-prone, and do not offer real-time visual insights. Moreover, most educational tools and simulators that do exist are either overly simplified or too complex, offering little room for **customization or performance comparison**.

There is a clear need for a **customizable, interactive, and educational simulator** that enables users to:

- **Manuallyinputtaskparameters**likeprocessID,arrivaltime,bursttime,andpriority.
- **Selectfrommultipleschedulingalgorithms**(bothpreemptiveandnon-preemptive).
- **Visualize**howprocessesarescheduledusing**Ganttchartsortimeline views**.
- **Automaticallycomputeanddisplaykeyperformancemetrics**foreachprocessand for the entire system.
- **Compare results** between algorithms to understand which performs best in different scenarios.

The lack of such a tool limits both learning and experimentation. Therefore, this project proposes the development of a **Custom Task Scheduler Simulator** that addresses thesegaps. The simulator will serve as a bridge between theory and practice by providing a hands- on, visual approach to learning and analyzing CPU scheduling techniques.

Byofferingflexibility,interactivity,andperformanceinsights,thistoolcangreatlybenefit:

- **Students**studyingoperatingsystemsorcomputerarchitecture.
- **Educators**lookingfordemonstrative tools.
- **Researchersanddevelopers**analyzingschedulingbehaviorinreal-world-like conditions.

Chapter2

Background/LiteratureSurvey

1. IntroductiontoCPU Scheduling

Inthefieldofoperatingsystems,**CPUscheduling**isoneofthemostcriticalandfundamental concepts. It refers to the method by which the operating system decides **which process gets to use the CPU** at any given time. Since a system may have multiple active processes (more than the number of available CPUs), scheduling ensures that all processes are handled efficiently and fairly. The ultimate aim is to optimize system performance in terms of **CPU utilization, throughput, response time, waiting time**, and **turnaround time**.

Various **CPU scheduling algorithms** have been proposed and studied over the years, each designedtoperformoptimallyinspecificsituations.Thesealgorithmsformavitalpartofany operating system and are crucial in areas such as **real-time systems, multitasking environments**, and **embedded systems**.

2. ExistingSchedulingAlgorithms

Several well-established scheduling algorithms have been researched and implemented in real-world operating systems:

- **First-Come-First-Serve(FCFS):**
  The simplest scheduling algorithm, where the process that arrives first gets executed first. It is **non-preemptive** and easy to implement but suffers from the **convoy effect**, where shorter tasks get stuck behind longer ones.
- **Shortest- Job- First (SJF):**
  This algorithm selects the process with the **shortest burst time** for execution. Whileit minimizes average waiting time, it requires **accurate prediction of process burst time**, which is often not feasible in real-time systems. It can be **preemptive (Shortest Remaining Time First)** or non-preemptive.
- **Round-Robin(RR):**
  Widelyusedin**time-sharingsystems**,itassignsafixedtimequantumtoeachprocess in the ready queue. RR is fair and responsive but can lead to **high waiting time** if the quantum is too small or too large.
- **Priority-Scheduling:**
  Each process is assigned a priority, and the CPU is allocated to the process with the highestpriority.Canbepreemptiveornon-preemptive.Akeyissuehereis**starvation**, where low-priority processes may never get scheduled.

These algorithms are the **foundation of CPU scheduling theory** and are used as benchmarks for evaluating new scheduling techniques.

3. SimulationToolsandStudies

Several researchers and educators have developed tools to simulate CPU scheduling algorithms. These simulators are typically used in **academic settings** to help students understand how different algorithms work in practice. Some notable examples include:

- **Gantt chart-based visual simulators:** These are simple tools where users input processes,andthetoolgeneratesaGanttcharttorepresentprocessexecutionvisually.
- **Open-sourceschedulingsimulators:**Thesearecommand-lineorGUItoolsavailable on GitHub and other platforms that support limited scheduling types and metrics.
- **University-based simulators:** Some universities have built Java or Python-based CPU schedulers as learning aids, but these are often limited in scope and not user-customizable.

Despitetheseefforts,**manyexistingtoolslack**:

- Flexibilityinchoosingalgorithmsordefiningcustomprocesses.
- Detailedanalysisofresults(e.g.,averagewaitingtime,responsetime,etc.).
- Clean,intuitiveuserinterfacesthatsupportreal-time interaction.
- Supportforboth**preemptiveandnon-preemptivemodes**inasingletool.

4. NeedforaCustomTaskSchedulerSimulator

There exists a clear gap between **theoretical understanding** and **practical visualization** of how scheduling algorithms perform in different scenarios. Many students find it difficult to understandconceptslikecontextswitching,starvation,andpriorityinversionwithoutactually **seeing these scenarios play out**.

The**CustomTaskSchedulerSimulator**projectisintendedtofillthisgapby providing:

- A**flexibleinputsystem**foruserstodefinecustomtasklistswithparameterslike arrival time, burst time, and priority.
- A**clean and interactive UI** to simulate the scheduling in both **preemptive and non-preemptive** settings.
- **Visualoutputs**,suchasGanttcharts,thathelpusersunderstandhowtasksaremanaged.
- **Performance analytics**, including turnaround time, waiting time, and response time for individual processes and the system overall.
- A**comparison module** to analyze and contrast the efficiency of each algorithm under different process sets.

Chapter3

Objectives

1. **ToDesignandImplementaCustomizableCPUSchedulingSimulator** The primary objective of this project is to develop a simulator that allows users to define custom tasks or processes with parameters such as **arrival time**, **burst time**, and **priority**. The simulator should support the creation, editing, and deletion of tasks before simulation

2. **ToSimulateMultipleCPUSchedulingAlgorithms(PreemptiveandNon- Preemptive)** The simulator aims to support the most commonly used CPU scheduling algorithms, including:
   - **First-Come-First-Serve(FCFS)**
   - **ShortestJobFirst(SJF)**
   - **RoundRobin(RR)**
   - **PriorityScheduling**Eachalgorithmshouldbeimplementedinboth **preemptive**and**non-preemptive**formswhereapplicable.

   .

3. **To Provide Visual Representation of Scheduling using Gantt Charts**Visualization plays a key role in understanding process scheduling. One of the project's core objectives is to provide a clear, **interactive Gantt chart-based visualization** of how processes are scheduled over time.

4. **To Analyze and Compare Algorithm Performance Using Key Metrics** Thesimulatorshouldcomputeanddisplayperformancemetricssuchas:
   - **Waiting Time**
   - **Turnaround Time**
   - **ResponseTime**
   - **CPU Utilization**
   - **Throughput** This objective enables users to evaluate and compare the efficiency of each algorithm under different workloads, helping them choose the best-suited strategy for specific use cases.

5. **To Develop an Educational Tool for Learning and Demonstration Purposes** Finally, the simulator is intended to serve as a **learning aid** for students, educators, and professionals. By offering a user-friendly interface and real-time feedback, the tool will make it easier to demonstrate the impact of CPU scheduling in operating systems. It will also act as a practical extension of theoretical knowledge, improving conceptual clarity through experimentation and visualization.

Chapter4

HardwareandSoftwareRequirements

HardwareRequirements

S. No	Nameof theHardware	Specification
1	Processor	IntelCorei5/i7 orequivalent
2	RAM	Minimum4GB
3	HardDisk	Minimum500 GB
4	Monitor	StandardHDdisplay

Software Requirements

S. No	Nameof theSoftware	Specification
1	OperatingSystem	Windows/Linux/MacOS
2	DevelopmentTools	VisualStudioCode,JupyterNotebook
3	Programming Language	Frontend(GUI/Web),Python,Visualization:matplotlib
4	Frameworks/Libraries	Tkinter,matplotlib,Numpy

Chapter5

PossibleApproach/Algorithms

1. IntroductiontoSchedulingAlgorithms

A **CPU scheduling algorithm** is a method by which the operating system determines the order in which processes are given access to the CPU. The purpose of this module in the simulator is to allow the user to select different scheduling algorithms and observe how the system behaves under each.

Intheproposedsimulator,thefollowingalgorithmswillbeimplemented:

- **First-Come-First-Serve(FCFS)**
- **ShortestJobFirst(SJF)**(PreemptiveandNon-preemptive)
- **PriorityScheduling**(PreemptiveandNon-preemptive)
- **RoundRobin(RR)**

Eachalgorithmwillbeimplementedinsuchawaythat it:

- Acceptsuser-definedinputs:arrivaltime,bursttime,priority, etc.
- Schedulestasks accordingly.
- Producesoutput:Ganttchart,waitingtime,turnaroundtime,andresponse time.

2. First-Come-First-Serve(FCFS)

Overview:

- Non-preemptivealgorithm.
- Tasksareexecutedintheorderoftheirarrival.
- Simpleandeasytoimplement,butcancauselongerwaittimesforshortprocesses.

Approach:

1. Sortallprocessesbasedonarrivaltime.
2. Executethefirstprocessentirely,thenmovetothenextone.
3. Calculatewaitingtimeandturnaroundtime.

Complexity:

- TimeComplexity:**O(nlogn)**(duetosorting).
- Bestforbatchprocessingsystems.

3. ShortestJobFirst(SJF)

Non-preemptive SJF:

- Choosesthetaskwiththeshortestbursttimeamongtheprocessesthathave arrived.

PreemptiveSJF(ShortestRemainingTimeFirst):

- TheCPUisgiventotheprocesswiththeleastremainingburst time.
- Ifanewprocessarriveswithashorterbursttimethantheremainingtimeofthe current process, the CPU is preempted.

Approach:

1. Keepapriorityqueuebasedonbursttime.
2. Ateachtimeunit,checkfortheprocesswiththeshortestbursttime.
3. Preemptifnecessary.
4. UpdateGanttchartaccordingly.

Complexity:

- TimeComplexity:**O(n^2)**orbetterwithmin-heap(priorityqueue).

4. PriorityScheduling

Non-preemptive:

- CPUisallocatedtotheprocesswiththehighestpriority(lowernumber=higher priority).
- Iftwoprocesseshavethesamepriority,FCFSis used.

Preemptive:

- Ifaprocess arriveswithahigherprioritythan thecurrentone, theCPUis preempted.

Approach:

1. Maintainapriority queuesortedby priority.
2. Ateverytimeunit, checkifahigher prioritytaskhas arrived.
3. Preemptifnecessary.
4. Continueexecution,updatingallnecessarystatistics.

Complexity:

- TimeComplexity:**O(n^2)**oroptimizedwithpriorityqueues.

IssuestoHandle:

- **Starvation:**Canbesolvedby**aging**(increasingpriorityofwaitingprocessesover time).

5. RoundRobin(RR)

Overview:

- Time-sharingschedulingalgorithm.
- Eachprocessisassignedafixed**time quantum**.
- If a process does not finish within its time quantum, it is preempted and added back to the queue.

Approach:

1. Addallreadyprocessestoaqueue.
2. Foreachtimeslice, assigntheCPU tothefront process.
3. Ifbursttime>timequantum,subtract timequantumandaddtheprocess back.
4. Continueuntilallprocessesarecomplete.
5. Trackandcalculatethemetrics.

Complexity:

- TimeComplexity:**O(n)**percycle.
- Efficientfortime-sharingsystemsand multitasking.

6. PerformanceMetrics Calculation

Afterscheduling,thefollowingmetricsarecalculatedforeachalgorithm:

- **WaitingTime(WT):**Timeaprocessspendsinthereadyqueue.
- **TurnaroundTime(TAT):**Totaltimefromarrivaltocompletion.
- **ResponseTime(RT):**TimefromarrivaluntilthefirstCPUallocation.
- **Throughput:**Numberofprocessescompletedperunit time.
- **CPUUtilization:**PercentageofCPUusage.

These metrics are visualized for comparison using **bar graphs** and **tabular formats** in thesimulator.

7. GanttChartVisualizationApproach

- The simulator will use a **horizontal bar chart (Gantt chart)** to represent scheduling visually.
- Eachtaskwillberepresentedbyablock,sizedaccordingtoitsCPUtimeandpositioned based on start time.
- TheGanttchartwillhelp users:
  - Understandpreemption.
  - Identifyprocessexecution order.
  - SpotidleCPUtimes (if any).

Table4.1 PseudocodeoftheCustomTaskSchedulerAlgorithm

```
StartProgram


→Inputtaskdetails(ProcessID,ArrivalTime,BurstTime, Priority)
→Storetasksinalist


→Displayscheduling options:
  1. FCFS
  2. SJF(Preemptive/Non-preemptive)
  3. Priority(Preemptive/Non-preemptive)
  4. RoundRobin


→Getuserchoiceandtimequantum(ifRound Robin)


→Basedonselected algorithm:
  - Sortorqueuetasksaccordingly
  - Simulateexecutionstep-by-step
  - Tracktimeandcalculate:
      • WaitingTime
      • TurnaroundTime
      • ResponseTime


→ Display:
  - Ganttchart
  - Tasktablewithallmetrics
  - Averagestats


EndProgram
```

References

1. Smith,R.(2007).*AnOverviewoftheTesseractOCREngine*.ICDAR.
2. Raffel,C.etal.(2020).*ExploringtheLimitsofTransferLearningwithaUnifiedText- toText Transformer (T5)*. JMLR.
3. Devlin,J.etal.(2019).*BERT:Pre-trainingofDeepBidirectionalTransformers*. NAACLHLT.
4. Campos, R. et al. (2020). *YAKE! Keyword Extraction from Single Documents*. Information Sciences.
5. Bojanowski, P., Grave, E., Joulin,A., &Mikolov, T. (2017). Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, 5, 135–146.
6. Loshchilov,I.,&Hutter,F.(2019).DecoupledWeightDecayRegularization. *InternationalConferenceonLearningRepresentations(ICLR)*.
7. Ramesh,A.,Pavlov,M.,Goh,G.,etal.(2021).Zero-ShotText-to-ImageGeneration. *InternationalConferenceonMachineLearning(ICML)*.
8. Mihalcea,R.,&Tarau,P.(2004).TextRank:BringingOrderintoTexts.*EMNLP*.