ACM Open Project 2024

CPU Scheduler

Name: Anvit Gupta

Enrolment number: 22114009

Batch: B.Tech. CSE 3rd Year

# Introduction

The CPU Scheduler is a sophisticated system designed to efficiently manage the execution of processes on a CPU, ensuring optimal utilization and performance. **This scheduler employs different algorithms for different types of processes to determine the order in which processes are executed**, balancing the need for responsiveness with the goal of maximizing throughput. The integrated frontend component provides a user-friendly interface that visually represents the scheduling decisions through a Gantt chart.

This visualization tool offers a clear and intuitive way to understand the sequence and duration of process executions, making it easier to analyse and optimize the performance of the scheduling strategy. Whether for educational purposes, research, or practical implementation in various computing environments, our CPU Scheduler and its accompanying Gantt chart visualization offer a comprehensive solution for managing and studying CPU process scheduling.

# Getting started

1. Install GCC compiler on Linux Ubuntu.
2. Install SFML environment into Ubuntu.
3. Open the folder where “main.cpp” is present, in the Linux terminal.
4. Compile and link the file using the command – **“g++ main.cpp -o main -lsfml-graphics -lsfml-window -lsfml-system”**. Input all the data of the processes required.
5. Execute the “main.exe” formed using the command – **“./main”**.

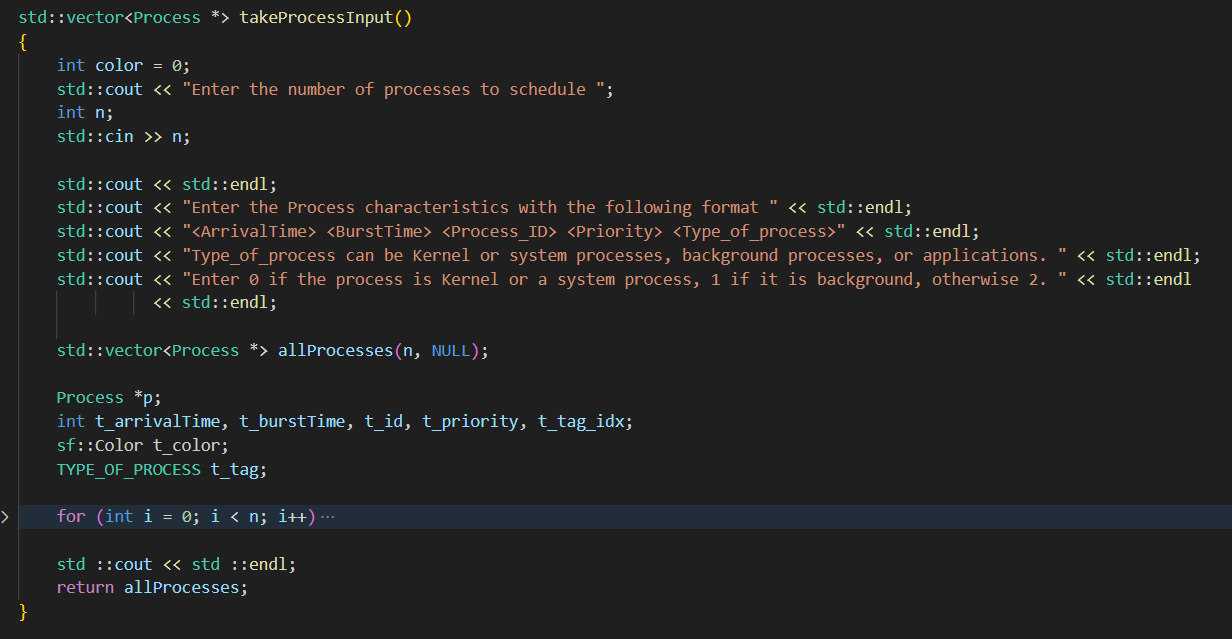
It will show the output of the scheduler in Gantt chart format.

# Code Walkthrough

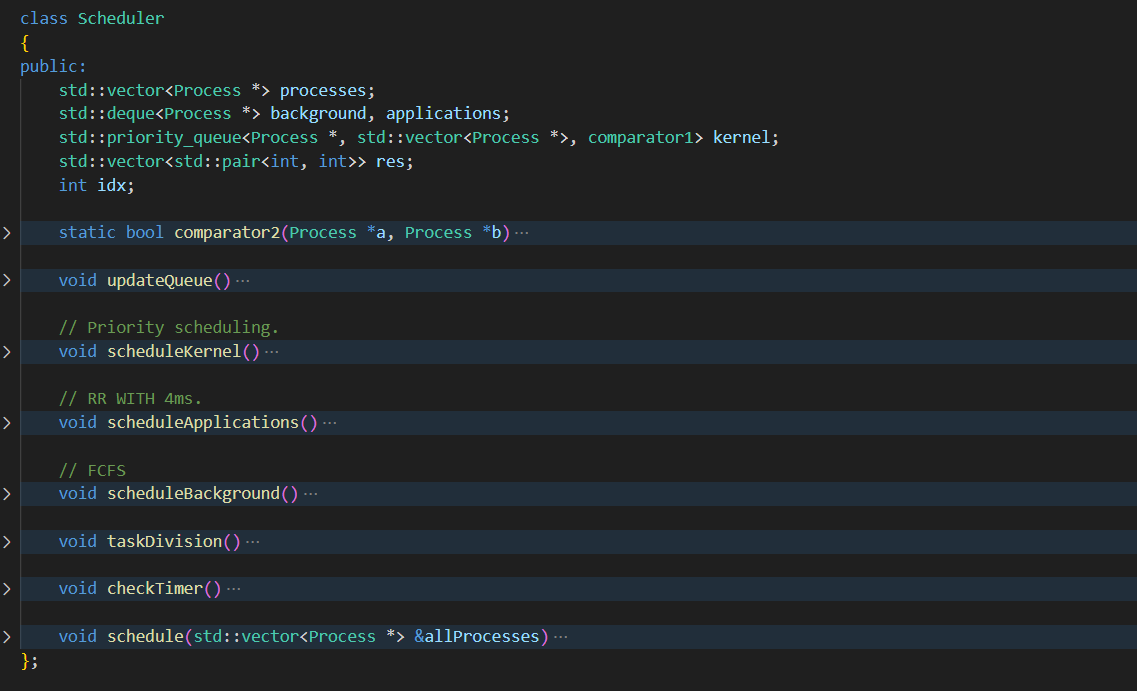
1. Define the state variables and structure of the process.
   1. STATE: A process can be in any of the following states at a time –
      1. Ready: It is ready to run on CPU and can be scheduled at any time by the scheduler.
      2. Running: It is running on the CPU at this moment.
      3. Blocked: It goes to Input-Output or is blocked momentarily for an event to occur.
      4. Terminated: It is completed and is about to exit the process queue.
   2. TYPE\_OF\_PROCESS: The tag by which the scheduler distinguishes between various processes. A process can be Kernel or a system process, background or daemon process (in Unix) or applications process.



1. Take Input of processes data



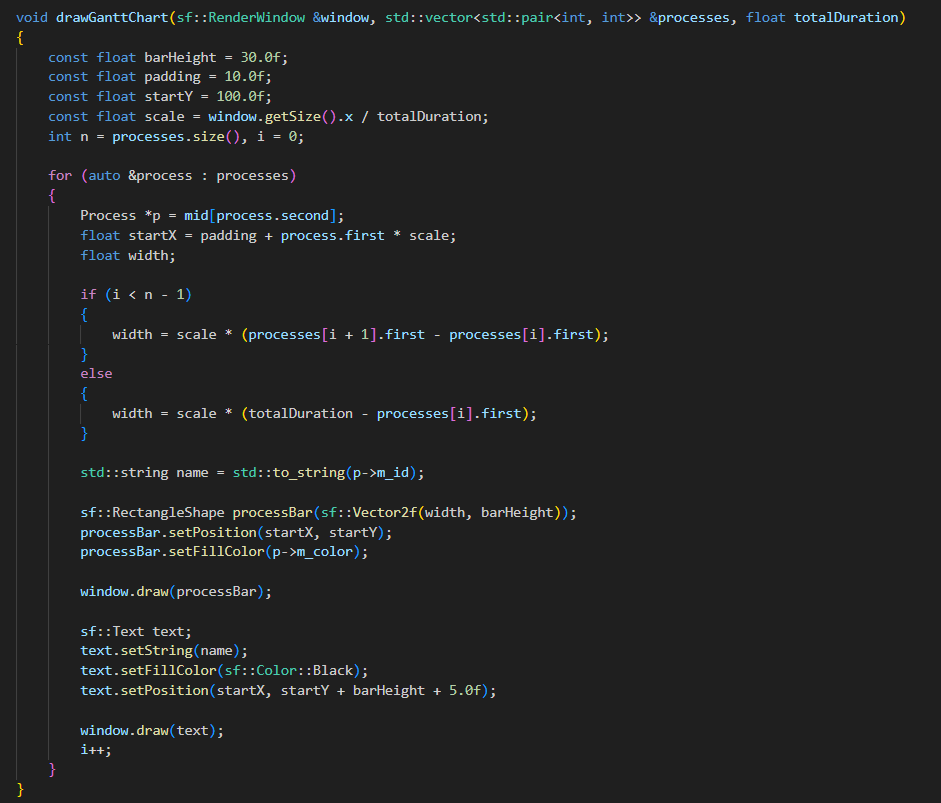
1. The core structure of Scheduler



It consists of several functions, discussed briefly below:

1. Schedule function: It will take all the process data and store it in the state of the class. Also, it will sort all the processes in order of Arrival time.

1. CheckTimer function: It will take all the processes data in “std::vector” format and will do the following task:
   1. **Categorization** of processes (**“update Queue”** function): Processes are categorized into three parts – kernel processes, applications and background processes. All the three types of processes are scheduled via different scheduling algorithms.
   2. **Task Division** among different types of processes:
      1. Schedulekernel: Kernel process are system processes which must completed with strict timeline and require highest priority among all the types of processes. And that’s why, **if there is any kernel process in the system, then Applications or Background can’t run on CPU.** **Pre-emptive Priority scheduling** is used to schedule kernel processes.
      2. ScheduleApplications: Applications require responsiveness rather than throughput, and hence, **Round Robin with 4ms** is used to schedule them. These processes run on CPU only when there are no kernel processes pending in the system.
      3. ScheduleBackground: Background processes run on CPU when there are no Kernel and Applications processes available. They are scheduled by using the simplest scheduling algorithm called **First come First serve (FCFS) algorithm**.
2. Draw Gantt chart:



The above function will draw the Gantt chart after the scheduler schedules all the process inputted by the user.

This function uses SFML library in C++, which is the simplest library to do desktop and graphics programming in C++.

# Tools and Technologies

This project leverages a combination of powerful tools and technologies to create an efficient and user-friendly CPU Scheduler:

* C++: The core logic of the CPU Scheduler is implemented in C++, providing robust performance and efficient memory management necessary for handling complex scheduling algorithms.
* SFML (Simple and Fast Multimedia Library): The frontend visualization, including the Gantt chart, is built using SFML. This multimedia library is chosen for its simplicity and efficiency in rendering graphics, making it ideal for creating interactive and visually appealing interfaces.

By integrating these technologies, the project achieves a high level of performance, reliability, and user engagement.

# Learning Takeaways

Working on this CPU Scheduler project has provided numerous valuable insights and skills, including:

**Understanding of CPU Scheduling Algorithms**: Gaining in-depth knowledge of various CPU scheduling algorithms such as First Come First Serve (FCFS), Shortest Job Next (SJN), Round Robin (RR), and Priority Scheduling. This includes understanding their mechanisms, advantages, disadvantages, and appropriate use cases.

**Proficiency in C++ Programming**: Enhancing proficiency in C++ by implementing complex scheduling algorithms and handling system-level programming tasks. This project has reinforced object-oriented programming concepts, memory management, and efficient data structures.

**Graphics Programming with SFML**: The entirely new skill that I learn in this project. Acquiring skills in graphics programming using the Simple and Fast Multimedia Library (SFML). Learning how to create interactive and dynamic visualizations, such as the Gantt chart, to represent scheduling processes effectively.

**Debugging and Performance Optimization:** Gaining experience in debugging complex systems and optimizing performance. Learning to use various tools and techniques to identify and resolve bottlenecks, ensuring the scheduler operates efficiently under different load conditions.

# Modifications in Future

1. **Support for dynamic Priorities**: If the process is allowed to have the dynamic priority and the System supports dynamic priority, then **if the waiting time of a process exceeds a threshold value, its priority will increase by a unit** with the clock timer.
2. **Range based scheduling**: If the range of priorities of the process available in the system at this moment is greater than some threshold value, then priority scheduling can be done to schedule them. But if lesser, then Round robin can be applied to improve on the responsiveness of the system having nearly same priority process.

# Resources and References

<https://www.sfml-dev.org/tutorials/2.6/start-linux.php>

<https://www.youtube.com/playlist?list=PLlrATfBNZ98dudnM48yfGUldqGD0S4FFb>

<https://www.studytonight.com/operating-system/cpu-scheduling>