IMPLEMENTING ROUND ROBINSCHEDULING FOR QUERIES

A PROJECT REPORT

Submitted by

SESSETI VENKATA SAI PAVAN[RA2211026010144]

GUNTOORI MRUNAL VARMA[RA2211026010156]

APPAJI SREE DHARMA SHASTA RAO [RA2211026010162]

Under the Guidance of

DR. VIMAL S

Assistant Professor, Department of Computational Intelligence

In partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

In

21CSC202J -Operating Systems

COMPUTER SCIENCE AND ENGINEERING

with a specialization in Artificial Intelligence and Machine Learning

DEPARTMENT OF COMPUTATIONAL INTELLIGENCE

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

November 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this B.Tech project report titled "IMPLEMENTING ROUND ROBIN SCHEDULING FOR QUERIES" is the bonafide work of **SESSETI VENKATA SAI PAVAN [Reg.No.RA2211026010144],GUNTOORI MRUNAL VARMA[Reg.No.RA2211026010156] and APPAJI SREE DHARMA SHASTA RAO [Reg.No.RA2211026010162]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein doesnot form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

DR. VIMAL S **SUPERVISOR** Assistant Professor Department of Computational Intelligence	**DR. Annie Uthra R** **HEAD OF THE DEPARTMENT** Department of Computational Intelligence
SIGNATURE OF INTERNAL EXAMINER	**SIGNATURE OF EXTERNAL EXAMINER**

November 2023

Department of Computational Intelligence

SRM Institute of Science and Technology

Own Work Declaration Form

Degree/ Course: B.Tech in Computer Science and Engineering with a specialization in Artificial intelligence and Machine learning

Student Names: **SESSETI VENKATA SAI PAVAN, GUNTOORI MRUNAL VARMA, APPAJI SREE DHARMA SHASTA RAO**

Registration Number: RA2211026010144, RA2211026010156, RA2211026010162

Title of Work: Implementing Round robin scheduling for Queries

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

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

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc.)
- Given the sources of all pictures, data etc. that are not my own

November 2023

• Not made any use of the report(s) or essay(s) of any other student(s) either

past or present

• Acknowledged in appropriate places any help that I have received from others

(e.g. fellow students, technicians, statisticians, external sources) • Compiled with

any other plagiarism criteria specified in the Course handbook / University

website

I understand that any false claim for this work will be penalized in accordance with the

University policies and regulations.

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

November 2023

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr. T.V.Gopal**, for his invaluable support.
We wish to thank **Dr. Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department, **Dr Annie Uthra R,** Professor, Department of Computational Intelligence, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Project guide **Dr Vimal S,** Assistant Professor, Department of Computational Intelligence, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

We sincerely thank the Computational Intelligence Department staff and students, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

SESSETI VENKATA SAI PAVAN
[RA2211026010144]
GUNTOORI MRUNAL
VARMA[RA2211026010156]
APPAJI SREE DHARMA SHASTA
RAO[RA2211026010162]

November 2023

TABLE OF CONTENTS

November 2023

ABSTRACT

The "Student Query Resolver System" is a sophisticated software application designed to streamline and simplify the process of addressing and resolving student queries in educational institutions. In today's fast-paced academic environment, students often encounter various concerns and questions related to their courses, schedules, grades, and more. This project aims to provide an efficient and user-friendly solution for managing and responding to these queries. The system integrates advanced natural language processing (NLP) and artificial intelligence (AI) technologies to categorize, prioritize, and route student queries to the appropriate departments or personnel. It employs a user-friendly web interface for students to submit their questions and track the status of their requests. Key features of the system include automated response generation for frequently asked questions, a knowledge base for instant information retrieval, and escalation protocols for more complex queries. It also offers analytics and reporting tools for educational institutions to gain insights into common student concerns and continuously improve their support services. In summary, the "Student Query Resolver System" not only enhances the overall student experience but also reduces the administrative burden on educational institutions by providing a streamlined and efficient mechanism for addressing and resolving student queries.

CHAPTER 1
INTRODUCTION

1.1 General

A Student Query Resolver project is a sophisticated and multifaceted system designed to revolutionize the way students interact with academic institutions and access information and support. This innovative solution is born out of the need to streamline and modernize the educational experience, making it more accessible, responsive, and user-centric.

At its core, this project involves the development of a highly intuitive and user-friendly platform that allows students to submit queries, questions, or concerns related to their academic journey. This platform can be accessed through a variety of devices, including web browsers, mobile applications, or even voice-activated assistants, ensuring that students can engage with the system in a manner most convenient for them.

The magic happens when a student submits a query. Behind the scenes, the system leverages cutting-edge technologies like artificial intelligence (AI) and natural language processing (NLP) to not only understand the query but also provide relevant and accurate responses. It does so by tapping into a vast and continually updated database of academic resources, which includes information on course offerings, schedules, academic policies, extracurricular activities, campus facilities, and more.

What truly sets this project apart is its ability to adapt and learn. Over time, the system becomes more proficient in understanding and addressing student inquiries, thanks to the AI's learning capabilities. This enables it to provide more personalized responses, catering to each student's unique needs and preferences.

In cases where the system cannot immediately provide a satisfactory answer, it doesn't leave the student hanging. It can seamlessly connect students with the right support personnel, such as academic advisors, departmental coordinators, or even fellow students who may have encountered similar challenges. This way, students always have a lifeline to the support they need.

The primary objective of a Student Query Resolver project is to make thestudent experience smoother and more efficient. It does so by significantly reducing the time and effort students typically expend searching for information or waiting in long queues at administrative offices. Moreover, it eliminates the constraints of office hours by offering 24/7 support, allowing students to get their questions answered at their convenience.

Beyond the immediate benefits for students, this project provides educational institutions with a treasure trove of data. By analyzing the types of inquiries and concerns raised by students, institutions can gain valuable insights into areas that may require improvement, whether it's refining curriculum offerings, streamlining administrative processes, or enhancing student services.

In sum, a Student Query Resolver project is a forward-thinking approach to enhancing the educational journey. It empowers students with a user-centric, technology-driven solution that delivers timely and accurate responses to their academic questions, and it equips institutions with actionable data to continually improve their offerings and services. It's a win-win for students and educational institutions alike, transforming the way we navigate and experience education in the 21st century.

1.2 Purpose

The purpose of a Student Query Resolver project is to:

- Improve Student Experience: Enhance the overall educational experience by providing students with a user-friendly and efficient means to get answers to their academic questions and concerns.

- Accessibility: Ensure that students can access information and support services 24/7 from various devices, reducing the need to physically visit administrative offices or rely on specific office hours.

- Efficiency: Streamline communication and information retrieval, saving students time and effort in searching for academic-related information.

- Personalization: Provide personalized responses based on individual student needs and preferences, creating a more tailored and user-centric experience.

- Learning and Adaptation: Utilize artificial intelligence and natural language processing to learn and adapt over time, continuously improving the system's ability to understand and address student inquiries.

- Support Staff Connection: Connect students with relevant support personnel, such as academic advisors or departmental coordinators, when the system cannot provide an immediate answer.

The Student Query Resolver project is a pioneering and multifaceted initiative that carries a profound and transformative purpose, poised to reshape the landscape of education by harnessing the power of technology and user-centric design. At its core, this project is all about empowerment, placing students firmly at the center of the educational experience. It recognizes that students are not mere recipients of information; they are active participants in their learning journey, and as such, they deserve a mechanism that provides them with not just answers but an avenue for meaningful interaction with their academic environment. This system fundamentally redefines the concept of accessibility, transcending the boundaries of time and place. It grants students the freedom to access the system on their terms, 24/7, through a wide array of devices, breaking free from the traditional constraints of office hours and physical locations.

The efficiency that comes from automation is not only a time-saver but a cost-saver as well. By automating responses to frequently asked questions, institutions can optimize resource allocation, reduce administrative workload, and create space for more strategic resource usage.

Ultimately, this project represents a significant step toward future-proofing education. It equips students and institutions with the tools and processes necessary to thrive in an ever-evolving digital age. It's a testament to the adaptability and innovation required to stay relevant and competitive in the ever-changing landscape of education, and it's poised to leave a lasting impact by fundamentally changing the way we access, experience, and engage with the world of education.

1.3 Scope

The scope for a Student Query Resolver project is truly monumental, encapsulating a transformative vision for the future of education. This all-encompassing initiative stretches across myriad dimensions, each holding the promise of revolutionizing the way students interact with academic institutions. It extends far beyond the basic premise of question-answering, venturing into uncharted territory where education is elevated to new heights. The project's vast scope incorporates a plethora of information delivery, covering not only routine academic inquiries but also encompassing comprehensive details on academic programs, admission requirements, financial aid resources, campus facilities, extracurricular activities, and even career guidance. This breadth of information transforms the project into a comprehensive knowledge hub, enriching the educational experience for students by providing a holistic view of their academic journey.

Moreover, the project's commitment to a seamless user experience introduces innovative features such as chatbots, voice recognition, and user-friendly interfaces, catering to the digitally savvy generation of students. This adaptability aligns with the modern educational ethos, where students expect convenient and intuitive digital interactions. Furthermore, the project's embrace of emerging technologies, including virtual reality (VR) and augmented reality (AR), propels the boundaries of traditional education. It ushers in a new era of immersive and interactive learning experiences, transcending the limitations of physical classrooms and opening the doors to remote support services and virtual campus tours. The project's futuristic outlook positions it as a trailblazer in redefining the educational landscape.

While its immediate impact is evident, the project's potential for global outreach underlines its universal applicability. It transcends geographical boundaries and can be adopted by educational institutions across the world, facilitating international collaborations and exchanges, thereby creating a global network of support for students pursuing education globally. Customization and personalization are integral aspects of its scope, allowing institutions to adapt the system to their unique identity, culture, and language preferences, rendering it not just a tool but an extension of an institution's identity.

Data analytics plays a pivotal role in the project's scope. The wealth of data generated through student interactions offers invaluable insights into student preferences, academic performance, and areas where institutions can effect improvements. This data-driven decision-making fosters continuous enhancement, efficiency, and adaptability. The project's commitment to inclusivity and accessibility underscores the principle that education should be accessible to all. It strives to accommodate a diverse range of students, including those with disabilities, aligning with modern educational values and legal requirements.

Collaborations in research and development constitute another dimension of its scope. Educational institutions can join hands to drive innovation and further refine the system. This fosters a spirit of continuous improvement and adaptation to evolving student needs, making the project not just a tool but a dynamic and ever-evolving ecosystem. Additionally, the project's role in fostering technological literacy among students is crucial. As students interact with the system, they acquire valuable digital skills, a necessity in today's technology-driven world.

In conclusion, the scope of the Student Query Resolver project is grand and visionary, representing not merely a technological initiative but a paradigm shift in education. It embraces the diverse and evolving needs of students in the 21st century, transforming the student-education relationship and preparing students for a future where technology and innovation are at the forefront of learning. This project is more than a tool; it is a visionary approach to education, where the boundaries of knowledge and accessibility are expanded, and the educational experience is transformed for the better.

1.4 Memory Management

Memory management stands as a critical and multifaceted cornerstone of the Student Query Resolver project, intricately woven into the fabric of its sophisticated operations. This ambitious system, designed to deliver seamless and efficient support to students, relies on robust memory management strategies to ensure the smooth and optimal execution of its vast array of functions. Let's embark on a comprehensive exploration of the intricate tapestry that is memory management within this intricate ecosystem.

At the heart of the project lies a monumental database, an extensive repository of academic information, student records, historical interactions, and a myriad of administrative data. Effective memory management plays a pivotal role in orchestrating the storage and retrieval of this wealth of data, ensuring that the most relevant and frequently accessed information is at the fingertips of the system, thereby minimizing response times for student queries.

In the dynamic realm of education, resource allocation is a complex puzzle. The system must allocate memory efficiently for a diverse range of tasks. These encompass data storage, user interface rendering, natural language processing, machine learning model execution, and an array of other functions. Effective memory management becomes the linchpin that prevents resource conflicts, bottlenecks, and guarantees the smooth orchestration of these myriad functions.

For an optimal user experience, the project often harnesses caching and prefetching techniques. Caching stores frequently accessed data in memory, reducing the need for repeated database queries, while prefetching anticipates the data that will likely be requested and proactively loads it into memory before it's demanded. Both of these strategies are inextricably tied to efficient memory management, as they balance available memory capacity with data freshness.

The Student Query Resolver project is a realm of simultaneous user interactions, necessitating astute memory management for concurrent and parallel processing. This involves the allocation of memory to each concurrent task and the maintenance of data integrity when multiple processes access shared resources. It's an intricate dance that prevents conflicts and safeguards against race conditions and other concurrency-related issues.

As the Student Query Resolver project scales to accommodate a growing user base and increasing data volumes, memory management is designed with scalability in mind. The system must be capable of dynamically allocating memory resources to meet these increasing demands, ensuring uninterrupted and efficient operation even as the system evolves.

In a distributed system or one serving a large user base, load balancing becomes a paramount concern. Memory management takes on the role of balancing memory usage, ensuring that no single component is overwhelmed. This proactive approach prevents performance degradation and potential system failures.

In essence, memory management within the Student Query Resolver project is a multifaceted and indispensable endeavor. It encompasses the efficient allocation, utilization, and release of system resources, vital for the system's ability to store and retrieve data, handle concurrent user interactions, prevent memory leaks, and ensure the stability, security, and scalability of the educational platform. This level of memory management proficiency guarantees not only the seamless and responsive support that students require but also safeguards the integrity, confidentiality, and accessibility of the educational data within the system, marking it as an essential component of its overall success.

1.5 Virtual Memory

Implementing virtual memory within the framework of the Student Query Resolver project presents a strategic opportunity to enhance its performance, scalability, and overall reliability. Virtual memory, a memory management technique that extends the capabilities of physical RAM (Random Access Memory) by utilizing a portion of the computer's hard drive or SSD (Solid-State Drive) as temporary storage for data, can offer several benefits to this sophisticated system.

First and foremost, virtual memory contributes to enhanced performance. By providing additional memory resources, it allows the system to offload less frequently used data from RAM to disk storage in cases where physical RAM becomes saturated. This dynamic memory management ensures that the system remains responsive, even during periods of peak usage, preventing memory congestion.

As the Student Query Resolver project continues to expand, virtual memory proves invaluable in terms of resource scalability. It can dynamically adjust the amount of virtual memory allocated, enabling the system to cater to a larger audience without the need for frequent hardware upgrades.

Moreover, virtual memory ensures continuous availability of data and services by seamlessly swapping data between physical RAM and virtual memory space as needed. This guarantees uninterrupted operation even when handling substantial data volumes or complex queries.

Efficient data storage is another advantage. The system can optimize resource usage by keeping frequently accessed data in physical RAM for rapid access, while less frequently used data resides in virtual memory, achieving a balance between performance and storage efficiency.

In terms of fault tolerance, virtual memory safeguards data in case of system failures or unexpected crashes, preserving data integrity and supporting graceful recovery.

For distributed systems or those with varying workloads, virtual memory can play a role in load balancing. By efficiently managing memory allocation across multiple servers or nodes, the system can distribute queries and tasks evenly, ensuring optimal performance and resource usage.

However, it's crucial to consider security implications, especially for sensitive data. Proper encryption and access controls should be in place to safeguard data stored in virtual memory, ensuring data privacy and compliance with data protection regulations.

Incorporating virtual memory into the Student Query Resolver project is a strategic move that can significantly enhance the system's capacity, performance, and scalability. It ensures efficient data management, workload balancing, and operational continuity, ultimately providing a more robust and reliable service to students and educational institutions. Careful implementation, with consideration for security and privacy, is essential to fully harness the potential of virtual memory in this educational ecosystem.

CHAPTER 2
LITERATURE REVIEW

In this section, we discuss the previous research that has been conducted about round robin application

2.1. Overview on Student Query Resolver

The Student Query Resolver project is a comprehensive and transformative initiative that redefines the educational experience, offering a multifaceted approach to streamline and enhance support for students across various educational institutions. This dynamic system, at its core, leverages cutting-edge technology and data-driven solutions to create an ecosystem that is intuitive, efficient, and deeply personalized to meet the diverse needs of students and institutions alike.

At the heart of this project is a commitment to user-centric design. It employs intuitive interfaces, chatbots, and virtual assistants, ensuring that students can effortlessly navigate and interact with the system. The aim is to make accessing information and support as seamless as possible, aligning with the modern digital expectations of students.

Central to the system's functionality is its data-rich knowledge base. This repository houses an extensive array of information pertinent to student queries. From detailed academic program descriptions to admission prerequisites, course schedules, campus resources, scholarship opportunities, and more, the knowledge base acts as a comprehensive resource that empowers students to find the information they need with ease.

Artificial intelligence and natural language processing play a pivotal role in the project. They enable the system to offer highly personalized responses and recommendations based on individual student preferences and historical interactions. As students engage with the system, it learns from their behavior, fine-tuning responses to meet their unique needs.

One of the standout features is its round-the-clock accessibility. The system is available 24/7, transcending the limitations of traditional office hours and physical locations. This ensures that students can access critical information and support precisely when they need it, empowering them to take charge of their educational journey.

Automation is a fundamental principle of the project. It automates responses to frequently asked questions, administrative processes, and repetitive tasks, saving students valuable time and resources while increasing administrative efficiency for educational institutions.

Scalability is another core aspect. The system is designed to grow with the needs of educational institutions. It can be customized to align with the specific requirements and branding of each institution, accommodating their unique cultural and language preferences. This adaptability makes it a valuable asset for a wide range of educational providers.

Real-time data analytics are instrumental in the project. The data generated through student interactions is analyzed on the fly, providing invaluable insights into student preferences, academic performance, and areas where institutions can make improvements. This data-driven approach fosters continuous enhancement and adaptability, keeping the system at the forefront of educational support.

The changing educational landscape, marked by the emergence of hybrid and online learning models, is well-supported by the project. It provides guidance on virtual learning tools, access to online resources, and connections to remote support services, ensuring that students experience a cohesive and comprehensive learning journey.

Inclusivity and accessibility are guiding principles. The project is designed to cater to a diverse range of students, including those with disabilities, ensuring that education is accessible to all. It complies with accessibility standards, fostering an inclusive educational environment.

Security and privacy are paramount. Robust data encryption and access controls are in place to safeguard sensitive student information. This guarantees data privacy and compliance with data protection regulations, a critical consideration in the modern digital landscape.

Collaborative research and development represent another facet of the project's ambition. It encourages educational institutions to collaborate on ongoing improvements, fostering innovation and the continuous refinement of the system to meet the evolving needs of students and institutions.

In sum, the Student Query Resolver project is a visionary and dynamic initiative, meticulously designed to reshape the educational landscape. It harnesses the potential of technology, automation, personalization, and data-driven insights to streamline access to critical information and support services. By doing so, it empowers students to take an active role in their learning, equips institutions with the tools and processes necessary to thrive in a digitally-driven educational environment, and paves the way for a future of education that is adaptable, efficient, and responsive to the ever-evolving needs of students and institutions. This project represents a significant leap forward in the journey to future-proof education in the 21st century.

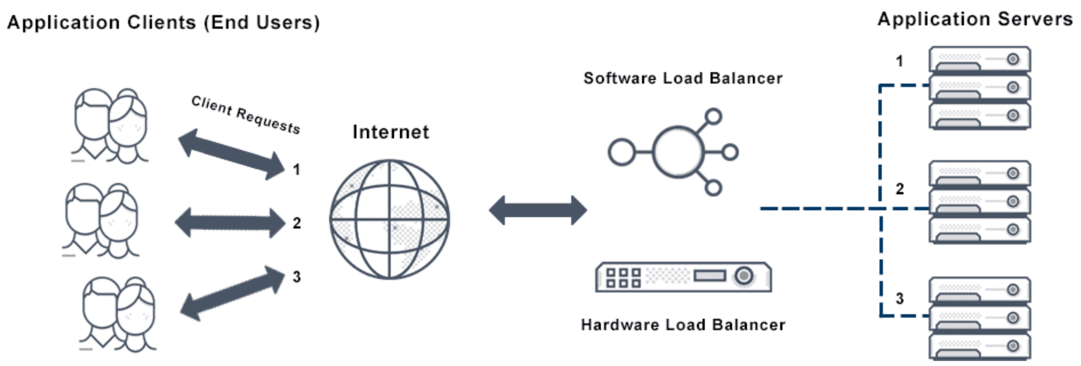


Diagram 1: Application of Round Robin Scheduling

2.2 Round Robin Scheduling Algorithm

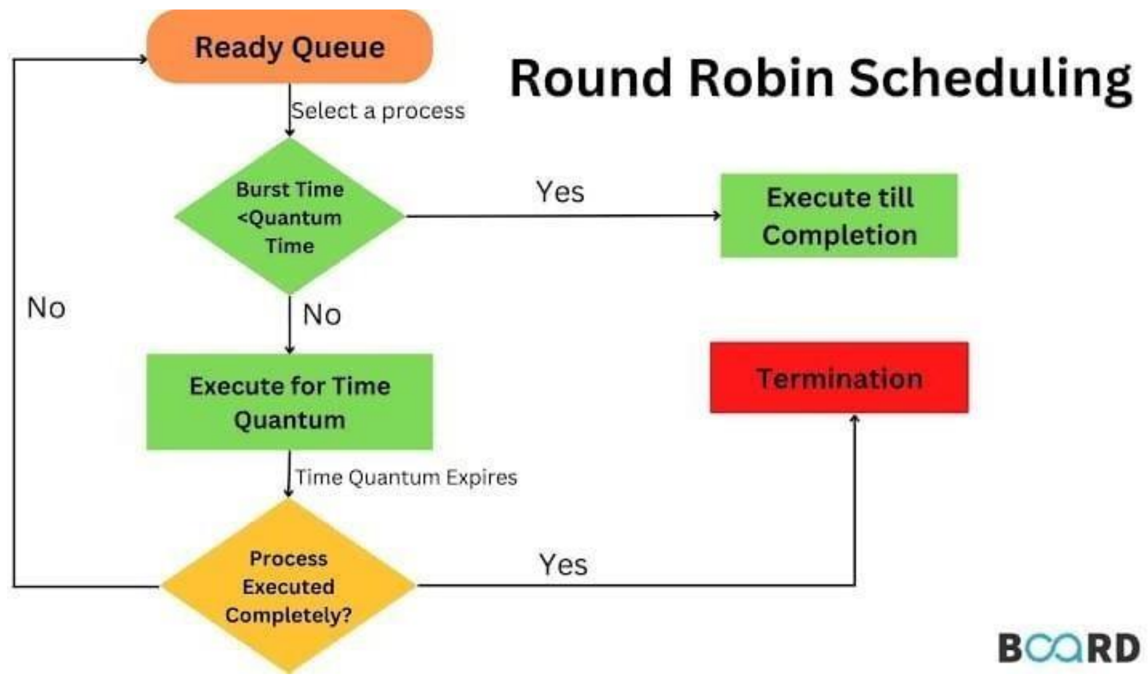
Round Robin Scheduling is a fundamental process scheduling algorithm that plays a pivotal role in the realm of multitasking operating systems. This algorithm is highly regarded for its equitable distribution of CPU time among competing processes, ensuring that no single process monopolizes the CPU and all processes get a fair share of execution time. At its core, Round Robin Scheduling relies on the concept of a fixed time quantum or time slice, which defines how long each process can occupy the CPU before it is preempted and placed back in the ready queue. The ready queue acts as a dynamic repository where processes transition from a waiting state to execution, and the scheduler selects processes for execution based on a first-come, first-served basis.

The essence of Round Robin Scheduling lies in its execution cycle. In each cycle, the CPU scheduler chooses the process at the front of the ready queue, granting it the CPU for the duration of its allotted time quantum. If a process completes its execution within this time frame, it gracefully exits the queue, making way for the next process. However, if a process exceeds its time quantum, it is preempted and moved to the back of the queue, ensuring that all processes receive a fair chance to run.

This algorithm offers several advantages, such as fairness and low response times. It excels in time-sharing environments where multiple processes or users vie for CPU resources. However, it is not without its challenges. Processes with extended CPU demands may suffer from frequent context switches, which can introduce overhead and reduce overall efficiency. Selecting the appropriate time quantum is a critical consideration, as it must strike a balance between minimizing context switches and maintaining fairness and responsiveness. Evaluation of Round Robin Scheduling involves examining performance metrics like CPU utilization, throughput, turnaround time, and waiting time. While CPU utilization is typically high due to minimal idle time, frequent context switches can impact throughput.

Round Robin Scheduling has seen various adaptations and enhancements, catering to different requirements and contexts. Variations like Weighted Round Robin, Multilevel Queue Scheduling, and Feedback Queue Scheduling provide flexibility in accommodating diverse types of processes and priorities. The practical applications of Round Robin Scheduling are widespread, from general-purpose operating systems to web servers that handle multiple user requests concurrently and interactive systems where user responsiveness is paramount.

In conclusion, Round Robin Scheduling is a time-tested and essential process scheduling algorithm in modern operating systems, offering a balance between fairness and responsiveness in multitasking environments. Its commitment to ensuring that all processes receive equitable CPU time makes it a valuable asset in time-sharing systems. However, careful consideration of the time quantum and effective management of context switching overhead are essential for optimal performance. Round Robin Scheduling continues to play a pivotal role in facilitating the concurrent execution of multiple processes while upholding fairness and responsiveness.



2.3 Time Quantum

In a student query resolver project, the concept of a time quantum, while not identical to its usage in CPU scheduling, can still play a crucial role in managing the efficiency and responsiveness of the system. Here, the "time quantum" can be seen as the maximum acceptable waiting time for a student's query or request before the system provides a response or resolution. This time threshold ensures that student inquiries are addressed promptly and that the educational institution maintains a high level of user satisfaction.

The determination of an appropriate time quantum for this project is multifaceted. It depends on several factors, including the complexity of queries, the institution's capacity to respond, and the expectations of the students. For straightforward queries, a shorter time quantum may be suitable, while more intricate or research-intensive questions might require a longer one.

Setting a response time threshold for different categories of queries, such as general inquiries, academic concerns, or administrative requests, is a common practice. For instance, general inquiries might have a shorter response time threshold, such as a few hours, while academic concerns might be granted a response time threshold of one to two days, allowing for more comprehensive research and assistance.

The student query resolver project should also consider the critical nature of some queries. Urgent matters, like technical issues during online exams or emergency academic support, could warrant even shorter response time thresholds to ensure immediate attention.

Monitoring and fine-tuning the time quantum or response time thresholds is essential for the project's success. Regularly evaluating the performance of the system, obtaining user feedback, and analyzing response times can help strike the right balance between providing prompt responses and maintaining the quality of support. The project should aim to optimize the response time thresholds to align with the needs and expectations of both the educational institution and the students it serves.

In summary, in the context of a student query resolver project, the concept of a time quantum transforms into response time thresholds, representing the maximum acceptable waiting times for different types of queries. By carefully defining and continuously optimizing these thresholds, the project ensures the efficiency and responsiveness required to meet the diverse needs of students while maintaining a high level of user satisfaction and support quality.

2.4 Priority Scheduling Algorithm

In a student query resolver project, the implementation of priority scheduling emerges as a pivotal and dynamic strategy for managing and addressing queries with varying degrees of urgency and importance. This scheduling algorithm offers a high level of customization, enabling the project to tailor its responses to match the specific needs and priorities of students effectively. Assigning priority values to queries is a critical aspect, and these values can be determined through a combination of criteria, such as the nature of the query, the academic level of the student, and the student's current academic situation. These criteria allow the project to categorize queries into different priority levels, ranging from high to low, reflecting the urgency and significance of each inquiry.

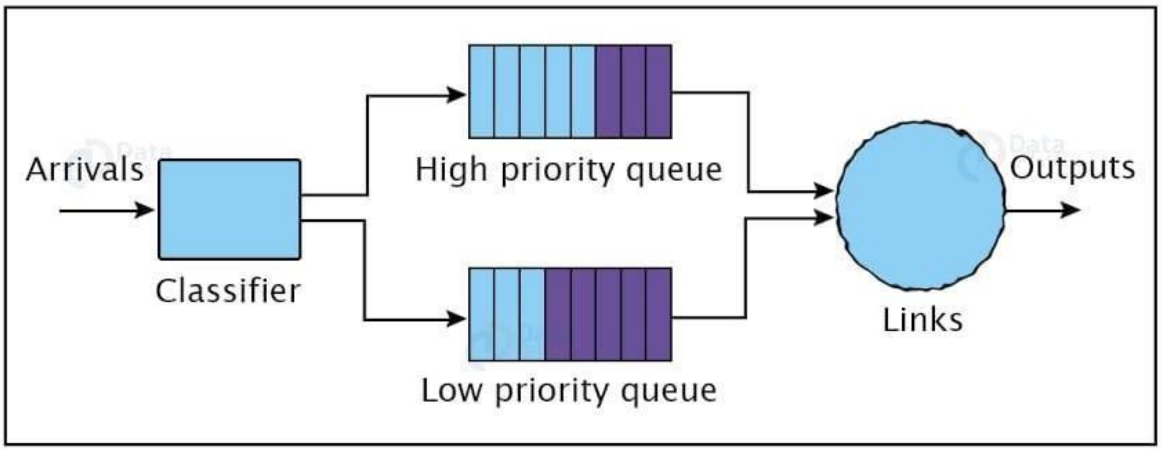
The introduction of priority scheduling also introduces the distinction between non-preemptive and preemptive systems. In a non-preemptive setup, once a query begins execution, it continues until completion, irrespective of new, higher-priority queries that may arrive. In contrast, a preemptive system enables higher-priority queries to interrupt the execution of lower-priority ones, ensuring that the most critical issues are promptly addressed. The choice between these two approaches depends on the project's specific requirements and objectives, taking into consideration factors like fairness and responsiveness.

One of the challenges associated with priority scheduling is the potential for lower-priority queries to experience starvation when high-priority queries continuously arrive. To mitigate this issue, the project can implement aging mechanisms that gradually increase the priority of long-waiting queries, preventing them from being permanently deprioritized. Continuous evaluation and fine-tuning of the priority assignment criteria are vital to maintain the effectiveness of the priority scheduling algorithm. This includes regularly assessing the relevance of the chosen factors and adjusting priorities to adapt to evolving student needs and institutional requirements.

In practical terms, priority scheduling is instrumental within the student query resolver project. High-priority queries encompass urgent academic appeals, technical issues during online exams, or immediate administrative requests. By using priority scheduling, the project ensures that these critical queries receive immediate and dedicated attention, significantly enhancing the student support experience.

Moreover, the project can explore integrating priority scheduling with other scheduling algorithms, such as Round Robin. This integration allows high-priority queries to be managed using priority scheduling, while lower-priority ones can undergo Round Robin scheduling to ensure a fair distribution of resources and maintain responsiveness across the board.

In summary, priority scheduling within a student query resolver project empowers the system to handle and respond to diverse queries from students with precision and efficiency. The assignment of priorities based on well-defined criteria ensures that critical queries receive swift and tailored attention, aligning perfectly with the project's primary objective of offering effective and timely support to students within a dynamic educational environment. By addressing challenges and continuously adapting the priority criteria to evolving needs, priority scheduling remains a crucial tool for optimizing the allocation of resources and delivering a high-quality student support experience.



2.5 Query Management

A robust and effective query management system is the backbone of any student query resolver project. This multifaceted system is designed to streamline and efficiently handle the diverse array of queries and requests that students may have during their educational journey. It encompasses various critical elements, starting with the user-friendly interfaces that enable students to capture and submit their queries. These interfaces may include web forms, mobile apps, or other digital channels that facilitate the effortless entry of questions and concerns.

One of the fundamental aspects of query management is query classification. As queries are submitted, the system categorizes them into predefined categories or tags, ensuring that they can be efficiently routed and addressed. Common categories include academic inquiries, technical issues, administrative requests, and general information. Effective query classification is pivotal for the system's ability to direct each query to the right support personnel or relevant departments.

Once classified, queries are routed to the appropriate individuals or teams through well-defined routing algorithms. These algorithms may be rule-based or employ machine learning techniques to determine the best recipients based on various factors, including query content, category, and priority. Priority management is another key element, where queries are assigned priority levels based on their urgency and importance. High-priority queries, such as academic appeals or urgent technical problems, are expedited to ensure prompt resolution.

Queue management comes into play when multiple queries are in the queue awaiting resolution. The system ensures that queries are addressed in order of priority and that none are left unattended for extended periods, thereby maintaining efficient service delivery. Support personnel use the query management system to provide responses and resolutions to student queries, while tracking query progress and response times, ensuring timely communication and keeping students informed about the status of their inquiries.

Furthermore, many student query resolver systems integrate a knowledge base or FAQ section. This integration enables support personnel to access pre-documented solutions and responses, streamlining query resolution and maintaining consistency in the answers provided to students. Notifications and alerts generated by the system are essential for informing support personnel about incoming queries, critical updates, or unresolved queries reaching critical response times, thus contributing to the maintenance of responsiveness.

Additionally, the system should facilitate the collection of feedback from students regarding the resolution process. This feedback is crucial for continuous improvement and allows the institution to adapt to changing student needs and optimize the support system.

Finally, the query management system often includes analytics and reporting tools, which generate reports on query volume, resolution times, common query categories, and other key performance indicators. Analyzing these data helps in refining the system and improving support services.

To provide a seamless experience, the query management system may also integrate with other systems such as student information databases, email platforms, and communication tools. This ensures that all relevant student information is accessible during query resolution, resulting in a more holistic and effective support process. In conclusion, a well-implemented query management system is a comprehensive and dynamic platform that streamlines the process of capturing, classifying, routing, and resolving student queries. It plays a vital role in enhancing student support, maintaining efficiency, and improving the overall educational experience.

2.6 Student Support Services

In the realm of a student query resolver project, the provision of student support services stands as a cornerstone, ensuring that students receive comprehensive and effective assistance throughout their academic journey. These services encompass a broad spectrum of areas, each designed to address specific needs and challenges that students may encounter. At the core of student support services is the provision of essential information and guidance. This includes offering students general information about academic programs, courses, schedules, and institutional policies. Academic advising, a critical facet of these services, involves experienced advisors who help students chart their academic paths. They provide guidance on course selection, program requirements, and career planning, empowering students to make informed decisions about their education.

Technical support is another pivotal aspect, acknowledging the significance of a seamless digital learning experience. Student support services stand ready to assist with technical issues, be it troubleshooting software problems, resolving connectivity issues, or ensuring that online learning platforms operate smoothly. The administrative front also finds its place within these services, addressing enrollment, registration, financial aid, and other administrative queries. Efficiently navigating these processes is essential for students, and support services play a pivotal role in facilitating this.

To enhance students' academic success, support services often extend their assistance to the realm of study skills, time management, and strategies for excelling in coursework. These resources provide students with the tools and knowledge needed to succeed academically. Acknowledging the importance of student well-being, many institutions include counseling and wellness services as part of their support offerings. These services provide emotional support, mental health resources, and stress management strategies, recognizing the importance of holistic student well-being.

In line with the principles of inclusivity and equal opportunities, accessibility services are provided for students with disabilities. These services ensure that appropriate accommodations and resources are available to make education accessible to all. Career development is also a core component, offering students resources for job searching, resume building, interview preparation, and internship opportunities, effectively preparing them for their future career paths.

In the context of a student query resolver project, query resolution is central to these services. Regardless of whether the query is academic, technical, or administrative, student support services are dedicated to ensuring that students receive prompt and effective responses and solutions to their inquiries. Outreach and communication strategies are vital to raising student awareness about the support services at their disposal. Outreach programs, newsletters, and targeted communication ensure that students are informed about available resources and assistance.

Feedback mechanisms are integral to the continuous improvement of support services. Students are encouraged to provide input on their experiences, allowing the institution to make necessary improvements and adaptations to meet evolving needs effectively. Furthermore, integration with learning management systems (LMS) streamlines access to support services. Students can conveniently seek assistance through their online course platforms, simplifying the process of obtaining help.

Lastly, recognizing the diversity of the student population, some institutions provide multilingual support services to assist students whose first language may not be English, ensuring that language barriers do not hinder students' access to the support they need. In summary, student support services are a multifaceted and essential component of a student query resolver project. They aim to create a supportive and inclusive educational environment by addressing students' academic, technical, administrative, and well-being needs. Continuous feedback and improvements are fundamental to the effectiveness of these services, ensuring that they evolve to meet the changing needs of students and the educational institution.

2.7 Query Scheduling

Query scheduling within a student query resolver project is a fundamental mechanism that dictates how incoming queries and requests from students are managed, organized, and addressed. This process involves several crucial actions aimed at ensuring efficient query resolution and a responsive support system.

First and foremost, when a query is submitted by a student, it undergoes a categorization process. During this step, the system classifies the query into predefined categories or tags, which allows for better organization and routing. Common categories might include academic inquiries, technical issues, administrative requests, and general information. This categorization sets the stage for the subsequent actions in the query scheduling process.

Following categorization, the system proceeds with query prioritization. Queries are assigned priority levels based on their urgency and importance. This is a critical step, as it determines the order in which queries are addressed. For instance, high-priority queries, such as academic appeals or urgent technical problems, are expedited to ensure prompt resolution, while lower-priority queries are queued accordingly.

Queue management is yet another vital component of query scheduling. In scenarios where multiple queries are awaiting resolution, the system ensures that they are addressed in the order of their priority. This action prevents any query from being left unattended for extended periods and maintains a high level of responsiveness, thereby ensuring that students receive timely assistance.

Upon reaching the front of the queue, queries undergo response and resolution, where support personnel engage with the students to address their queries or requests. During this action, tracking the progress of query resolution and monitoring response times are essential. These actions ensure that students are kept informed about the status of their queries and maintain open lines of communication.

An integrated knowledge base is often utilized during the resolution process, which contains pre-documented solutions and responses to common queries. Support personnel can access this knowledge base to streamline query resolution and maintain consistency in the answers provided to students.

Furthermore, notifications and alerts are generated by the system to inform support personnel about incoming queries, critical updates, or unresolved queries approaching critical response times. These alerts are crucial for maintaining responsiveness and ensuring that no query falls through the cracks.

Lastly, continuous monitoring and feedback collection play a significant role in refining the query scheduling process. Regular evaluation of query scheduling performance, analysis of response times, and feedback from students contribute to ongoing improvements and adaptations to meet evolving needs and enhance the overall efficiency and effectiveness of the student query resolver system.

In summary, query scheduling within a student query resolver project encompasses a series of critical actions, including query categorization, prioritization, routing, queue management, response and resolution, knowledge base integration, notifications and alerts, and continuous monitoring and feedback collection. These actions collectively ensure that queries are efficiently managed, students receive timely assistance, and the support system operates at its optimal level, thereby enhancing the overall student experience.

CHAPTER 3

PROPOSED METHODOLOGY

The proposed methodology for a student query resolver system is a structured approach to developing and implementing an effective platform that efficiently manages and addresses queries and requests from students.

3.1 First In First Out

The First-Come-First-Serve (FCFS) scheduling algorithm serves as a foundational and intuitive mechanism within student query resolver systems, fundamentally shaping the manner in which incoming queries from students are managed and addressed. At its core, FCFS adheres to a principle of chronological order, whereby queries are processed in the sequence of their arrival. This means that the first query submitted is the first to be addressed, establishing a sense of fairness and equity in query resolution.

As queries are submitted by students, they are promptly placed in a queue, with older queries positioned at the forefront. This queue is the central organizational structure of the FCFS system, providing a historical record of all student inquiries. Categorization is the next pivotal step, whereby queries are classified based on their nature, distinguishing between academic concerns, technical issues, administrative requests, and various other query types. Categorization serves a crucial role in routing the queries to the pertinent support staff or departments best equipped to address them.

The FCFS paradigm is often extended within each query category, where it maintains its fidelity to the "first in, first out" principle. This means that queries within a specific category are resolved in the order of their submission. Nevertheless, FCFS allows for flexibility and adaptability. Some implementations introduce variations to consider the urgency of queries, ensuring that high-priority issues receive immediate attention. This dynamic approach strikes a balance between the simplicity of FCFS and the necessity to respond swiftly to critical concerns.

Within this structured framework, queries are systematically managed, tracked, and resolved. Support staff engage with students to provide responses, solutions, and guidance. One of the inherent advantages of FCFS is its natural tracking of query resolution times, facilitating the measurement of efficiency and response times. This feature enables the identification of potential bottlenecks or delays in the query resolution process.

While FCFS offers the advantages of simplicity and an equitable approach, it also comes with its challenges. It may not always be the most efficient scheduling algorithm, particularly when faced with a substantial volume of queries or when the prioritization of urgent issues is essential. FCFS does not inherently consider the complexity or effort required for query resolution, which can be a limitation in cases where certain queries demand specialized expertise or extensive time.

In conclusion, the First-Come-First-Serve (FCFS) scheduling algorithm is a cornerstone of student query resolver systems, embodying a fundamental principle of chronological order. It orchestrates a seamless and orderly approach to managing student queries, fostering transparency, and a sense of fairness. The inherent balance between simplicity and adaptability makes FCFS an effective choice for scenarios where maintaining a structured sequence of query resolution is a paramount concern, and urgency or complexity considerations are less pronounced. FCFS serves as an integral component in the student support ecosystem, ensuring that queries are addressed systematically, methodically, and with a commitment to equitable service delivery.

3.2 Session Id’s

Session IDs, in the context of a student query resolver, serve as the linchpin for a multifaceted ecosystem of user interactions, security, and personalized experiences. These unique tokens are generated for each student's session and wield a profound influence on the entire user journey. At their core, session IDs are indispensable for user session management, establishing a distinct digital identity for each student. In an educational environment characterized by diverse needs and frequent access to resources, this individual identification is the bedrock on which the resolver builds a personalized experience.

However, session IDs do not merely function as arbitrary strings; they are deeply enmeshed in the intricate tapestry of security and access control. When a student logs into the query resolver, a session ID is minted, serving as both a key and a sentinel. It is this key that unlocks a realm of features and information tailored to the student's role and permissions. Simultaneously, it acts as a sentinel guarding against unauthorized access. The session ID confirms the student's digital identity with each interaction, ensuring that sensitive educational data remains confidential and protected.

A paramount facet of student query resolvers is their capacity to accommodate stateful interactions. The educational journey often involves elaborate, multi-step processes. A student may begin with a simple course inquiry, evolve into class selection, and culminate in the pivotal act of enrollment. Session IDs are the custodians of this journey. They remember where the student stands in this process, retaining their selections and data. The session ID breathes continuity into an otherwise fragmented journey, ensuring that each student can seamlessly pick up where they left off.

Security is an omnipresent concern, and session IDs are at the forefront of this battle. They are not merely generated at random; they must be cryptographically secure, resilient to brute force attacks, and managed with utmost care. The location of the session ID is also a critical consideration. Placing it on the server side prevents the vulnerabilities associated with client-side storage, safeguarding against session hijacking and other forms of security breaches.

Moreover, to bolster security and resource management, sessions are endowed with timeout and expiry mechanisms. If a student's session remains inactive for a predefined period, it will expire, necessitating reauthentication to continue. This proactive measure safeguards against unauthorized access in cases where a student leaves their device unattended.

In the world of data-driven insights, session IDs play a crucial role. They are the conduits through which the resolver collects valuable data on user behavior. This data is instrumental in improving system performance, identifying common queries, and enhancing the resolver's functionality. With a wealth of data at its disposal, the resolver can fine-tune its features and services to better cater to the ever-evolving needs of students.

Furthermore, session IDs open the door to personalization. The resolver can utilize the session ID to remember user preferences, frequently accessed options, and historical interactions. This deep understanding of the student's behavior and choices enables the resolver to provide a tailored, individualized experience. Students feel like the system understands their unique needs, creating a stronger bond between the user and the resolver.

In multi-user environments, concurrent access is a complex challenge. Session IDs, however, rise to this challenge. Each session ID represents a unique user session, effectively preventing interference between different students' sessions and their data. This concurrency control ensures data integrity and maintains a seamless user experience, even in busy educational settings.

In summary, session IDs are the lynchpin of user authentication, stateful interactions, personalization, and security in a student query resolver. They ensure that students have a secure, personalized, and efficient experience while using the system, safeguarding their data, providing a seamless journey, and offering a personalized touch to the educational experience.

3.3 Role Based Access Control

Role-Based Access Control (RBAC) is a comprehensive and highly structured approach to access management within a student query resolver, which is instrumental in ensuring the integrity, security, and efficiency of educational systems. RBAC introduces a systematic hierarchy of user roles, each associated with specific responsibilities, permissions, and privileges. This role hierarchy is key to maintaining a well-organized and secure educational environment.

In an educational institution, a multitude of stakeholders, including students, faculty, administrators, and staff, interact with the student query resolver. RBAC assigns distinct roles to each of these stakeholders, placing them within a predefined hierarchy. This hierarchical arrangement allows for a clear definition of who can do what within the system. For instance, a dean might hold a higher role than a professor, while students occupy roles with more limited privileges.

The assignment of roles is a critical aspect of RBAC. As students, faculty, and staff join the institution or assume different responsibilities, they are assigned the appropriate roles. This allocation is often determined by their job titles, academic positions, or specific needs. For example, upon registration, a student is assigned the "student" role, which grants them access to certain features and data.

Permissions and privileges are the heart of RBAC. Each role is associated with a set of permissions that delineate the actions a user can perform. For instance, a student may have permissions to search for courses and view grades, while a faculty member is granted the privilege to edit course content. These permissions are carefully configured to match the role's responsibilities and authority, ensuring that users can fulfill their duties without overstepping their boundaries.

Access Control Lists (ACLs) are frequently employed in RBAC to map roles to specific resources and permissions. ACLs enable fine-grained control over which roles can access particular data or execute specific actions. For example, an ACL might grant the "student" role permission to view course information but deny the ability to modify it, while the "faculty" role can edit and update course content.

One of the strengths of RBAC is its dynamic and flexible nature. Users can hold multiple roles simultaneously, allowing them to perform various functions within the system. For example, a faculty member may also serve as an academic advisor, receiving both the "faculty" and "advisor" roles, each with its set of permissions and responsibilities.

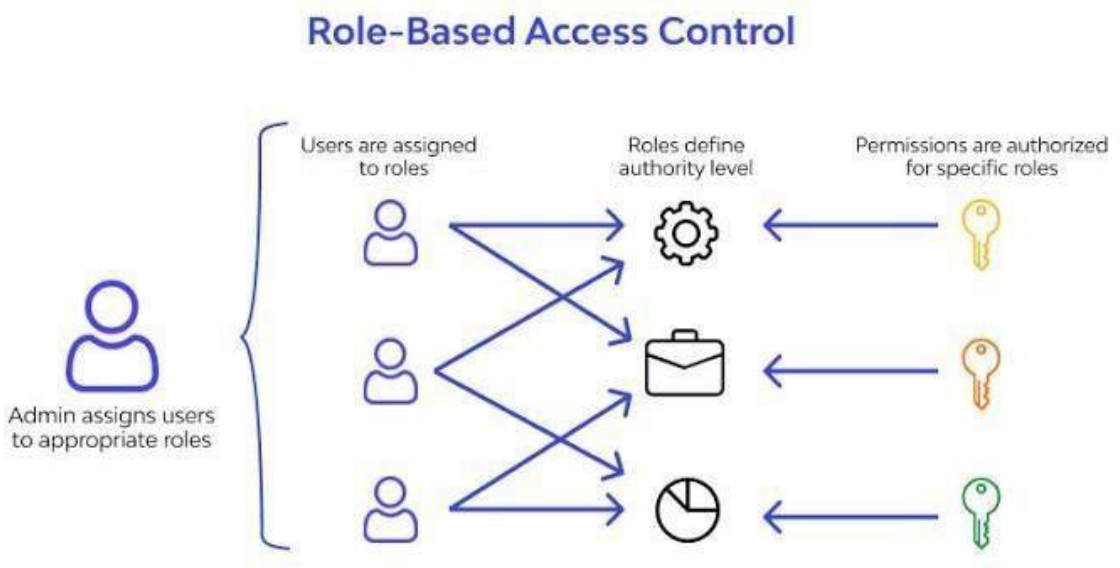
Policies are central to RBAC's access control. Policies define the rules that dictate which roles have access to specific resources or perform particular actions. These policies are set and managed by administrators who can adjust them to accommodate changing requirements or evolving roles within the institution.

RBAC is not only about enabling access but also enforcing the principle of separation of duties. This principle prevents conflicts of interest or misuse of power within an organization. For instance, a faculty member should not have the ability to modify their own grades. RBAC ensures that such conflicts are avoided, promoting a fair and transparent environment.

Security and compliance are of paramount importance in an educational context, and RBAC plays a pivotal role in ensuring both. RBAC helps prevent unauthorized access, protect sensitive data, and align with regulatory requirements. This alignment is particularly significant in educational institutions, which must adhere to strict regulations, such as FERPA, to safeguard student data and privacy.

In conclusion, RBAC in a student query resolver is an intricate and highly organized framework that plays a vital role in maintaining the integrity, security, and efficiency of the system. It provides a structured and systematic approach to managing user access and permissions, promoting a secure and transparent educational environment while adapting to the evolving needs of the institution and its stakeholders.

3.4


Role-Based Access Control
Users are assigned to roles
Roles define authority level
Permissions are authorized for specific roles
Admin assigns users to appropriate roles

3.4 Algorithm

The code is an implementation of a scheduling algorithm for handling faculty and student queries in an operating system. The algorithm used in the code is Round Robin, which is a pre-emptive scheduling algorithm. Here's a high-level algorithm based on the student query resolver :

1. Define a structure "Query" to represent each query, including attributes like QueryID, ArrivalTime, BurstTime, CompletionTime, and TotalTime.

2. Initialize global variables, including arrays to store faculty and student queries, time quantum, and counters.

3. Implement a function "InputsForProcess" to take input for the queries, including the query type, query ID, arrival time, and burst time. Check for constraints such as time format and availability.

4. Implement sorting functions, "FacultySort" and "StudentSort," which use QuickSort to sort faculty and student queries based on their arrival times.

5. Implement a "MergeQueries" function to merge faculty and student queries into a single array, "Mix," while maintaining the order based on arrival times.

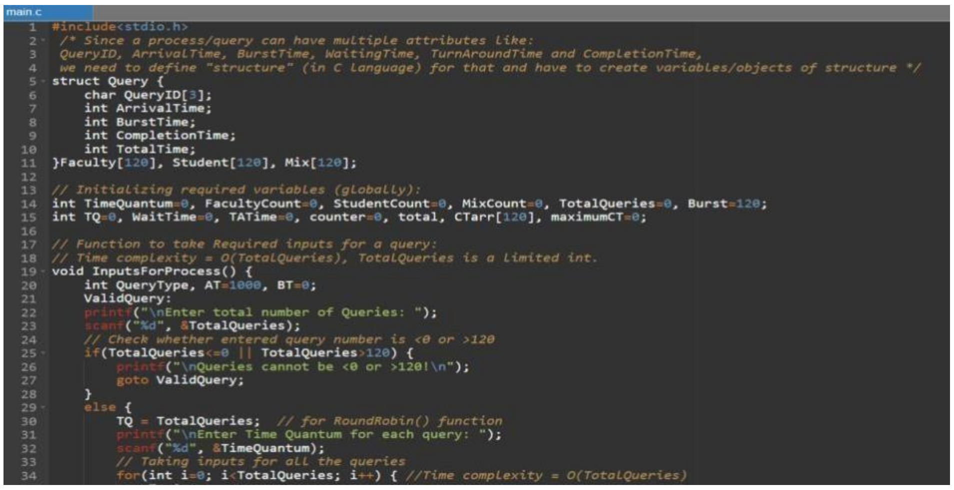
6. Implement the Round Robin scheduling algorithm in the "RoundRobin" function, which iteratively processes queries using a specified time quantum. Update waiting times, turnaround times, and completion times.

7. Calculate the maximum completion time using the "MaxCT" function.

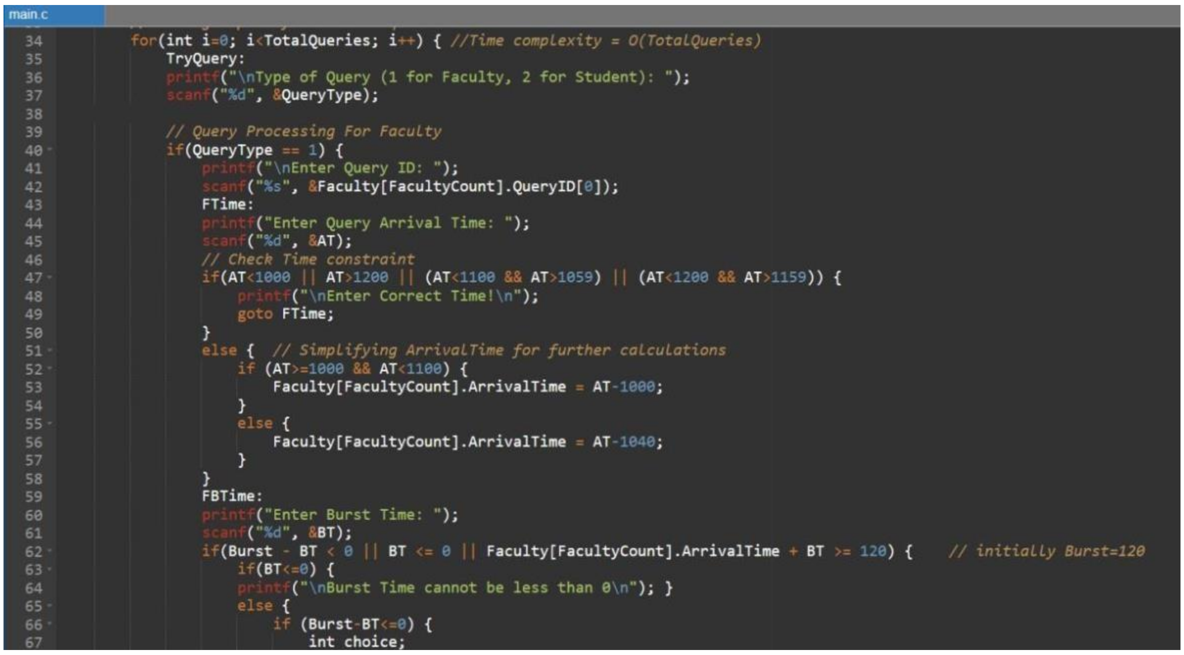
8. Finally, print the results in the "PrintResult" function, including total time spent, average waiting time, and average turnaround time.

The algorithm primarily relies on Round Robin scheduling for processing queries in a time-sharing manner, and it also handles input validation and sorting based on arrival times. It ensures that queries are processed within the specified time quantum.

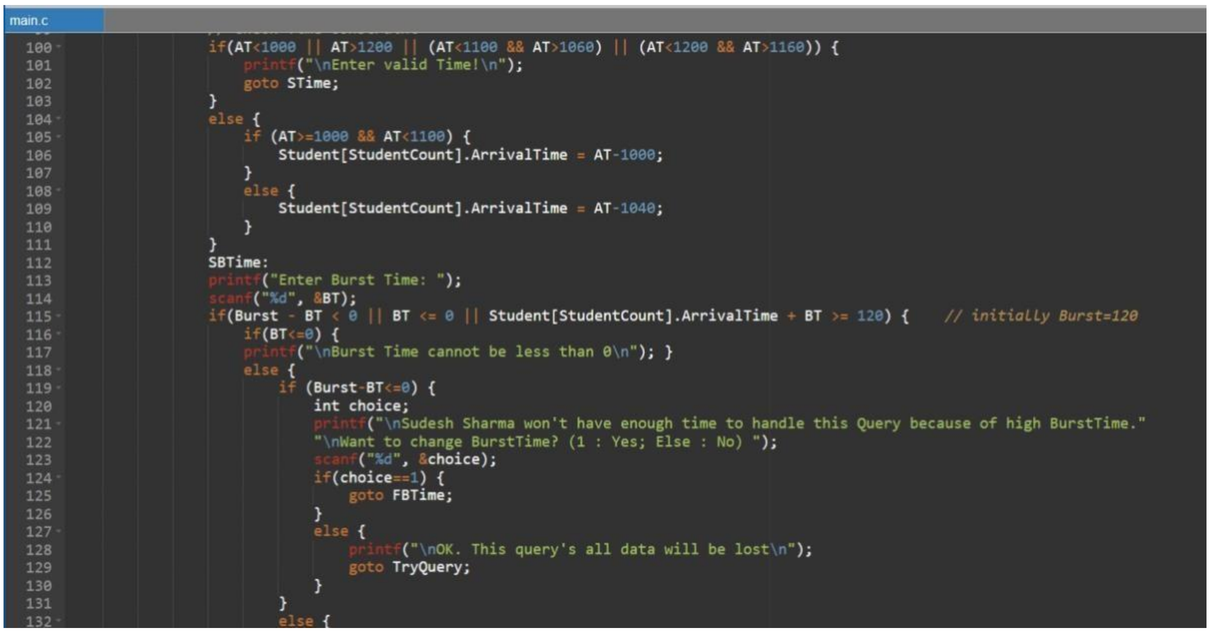
3.5 Program



```c
#include<stdio.h>
 /* Since a process/query can have multiple attributes like:
 QueryID, ArrivalTime, BurstTime, WaitingTime, TurnAroundTime and CompletionTime,
 we need to define "structure" (in C language) for that and have to create variables/objects of structure */
struct Query {
    char QueryID[3];
    int ArrivalTime;
    int BurstTime;
    int CompletionTime;
    int TotalTime;
}Faculty[120], Student[120], Mix[120];

// Initializing required variables (globally):
int TimeQuantum=0, FacultyCount=0, StudentCount=0, MixCount=0, TotalQueries=0, Burst=120;
int TQ=0, WaitTime=0, TATime=0, counter=0, total, CTarr[120], maximumCT=0;

// Function to take Required inputs for a query:
// Time complexity = O(TotalQueries), TotalQueries is a limited int.
void InputsForProcess() {
    int QueryType, AT=1000, BT=0;
    ValidQuery:
    printf("\nEnter total number of Queries: ");
    scanf("%d", &TotalQueries);
    // Check whether entered query number is <0 or >120
    if(TotalQueries<=0 || TotalQueries>120) {
        printf("\nQueries cannot be <0 or >120!\n");
        goto ValidQuery;
    }
    else {
        TQ = TotalQueries;  // for RoundRobin() function
        printf("\nEnter Time Quantum for each query: ");
        scanf("%d", &TimeQuantum);
        // Taking inputs for all the queries
        for(int i=0; i<TotalQueries; i++) { //Time complexity = O(TotalQueries)
```

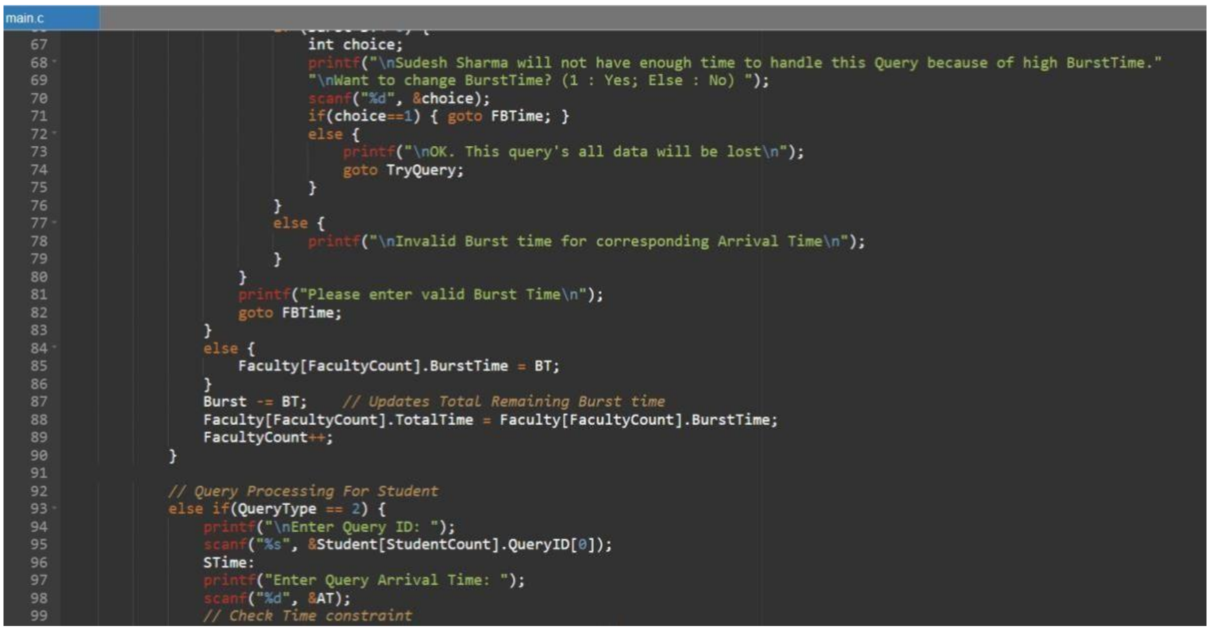


```c
        for(int i=0; i<TotalQueries; i++) { //Time complexity = O(TotalQueries)
            TryQuery:
            printf("\nType of Query (1 for Faculty, 2 for Student): ");
            scanf("%d", &QueryType);

            // Query Processing For Faculty
            if(QueryType == 1) {
                printf("\nEnter Query ID: ");
                scanf("%s", &Faculty[FacultyCount].QueryID[0]);
                FTime:
                printf("Enter Query Arrival Time: ");
                scanf("%d", &AT);
                // Check Time constraint
                if(AT<1000 || AT>1200 || (AT<1100 && AT>1059) || (AT<1200 && AT>1159)) {
                    printf("\nEnter Correct Time!\n");
                    goto FTime;
                }
                else {  // Simplifying ArrivalTime for further calculations
                    if (AT>=1000 && AT<1100) {
                        Faculty[FacultyCount].ArrivalTime = AT-1000;
                    }
                    else {
                        Faculty[FacultyCount].ArrivalTime = AT-1040;
                    }
                }
                FBTime:
                printf("Enter Burst Time: ");
                scanf("%d", &BT);
                if(Burst - BT < 0 || BT <= 0 || Faculty[FacultyCount].ArrivalTime + BT >= 120) {    // initially Burst=120
                    if(BT<=0) {
                    printf("\nBurst Time cannot be less than 0\n"); }
                    else {
                        if (Burst-BT<=0) {
                            int choice;
```



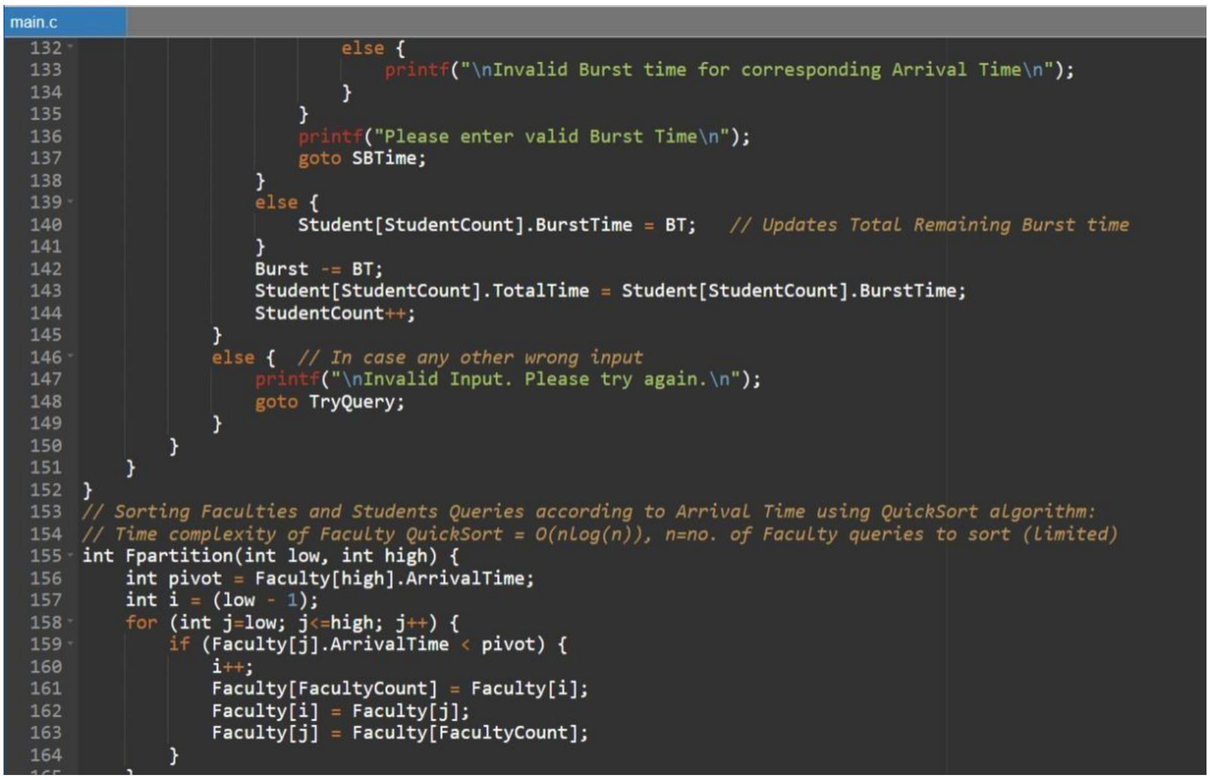
main.c

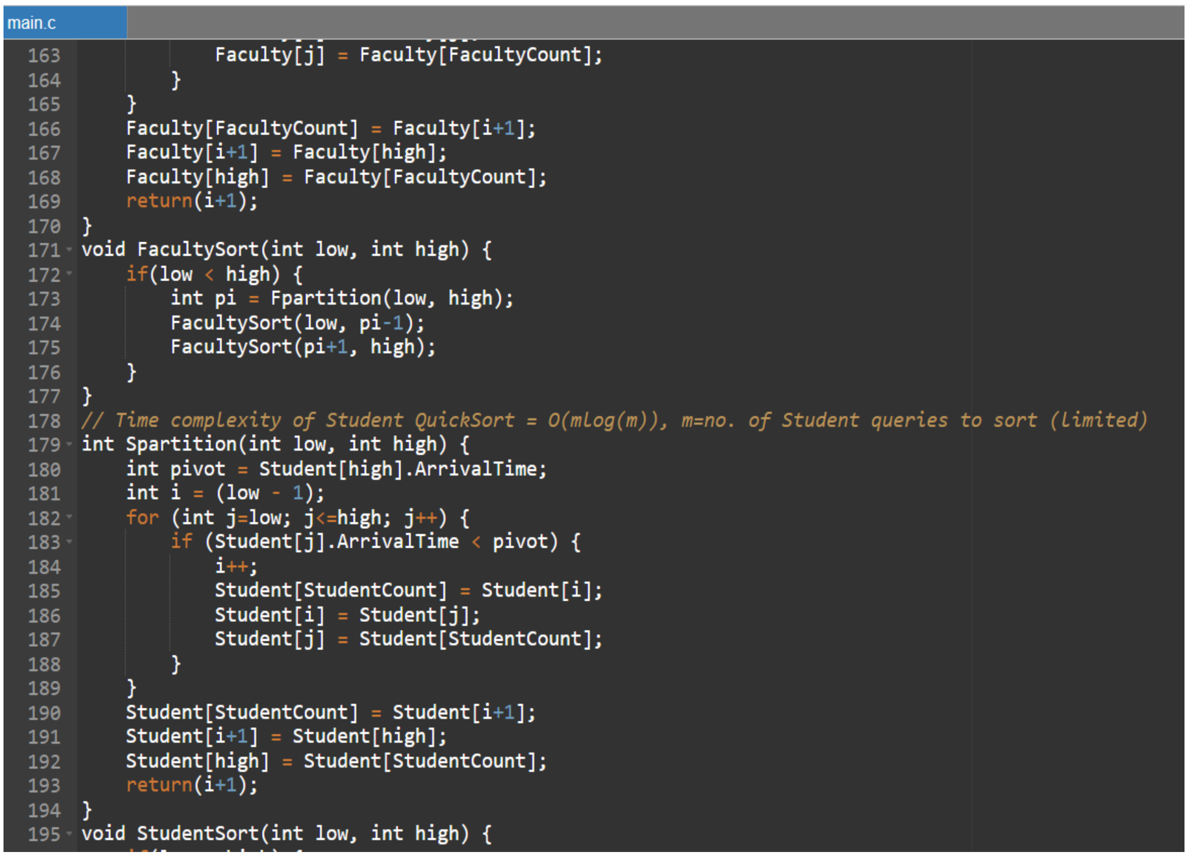
```c
if(AT<1000 || AT>1200 || (AT<1100 && AT>1060) || (AT<1200 && AT>1160)) {
    printf("\nEnter valid Time!\n");
    goto STime;
}
else {
    if (AT>=1000 && AT<1100) {
        Student[StudentCount].ArrivalTime = AT-1000;
    }
    else {
        Student[StudentCount].ArrivalTime = AT-1040;
    }
}
SBTime:
printf("Enter Burst Time: ");
scanf("%d", &BT);
if(Burst - BT < 0 || BT <= 0 || Student[StudentCount].ArrivalTime + BT >= 120) {    // initially Burst=120
    if(BT<=0) {
    printf("\nBurst Time cannot be less than 0\n"); }
    else {
        if (Burst-BT<=0) {
            int choice;
            printf("\nSudesh Sharma won't have enough time to handle this Query because of high BurstTime."
            "\nWant to change BurstTime? (1 : Yes; Else : No) ");
            scanf("%d", &choice);
            if(choice==1) {
                goto FBTime;
            }
            else {
                printf("\nOK. This query's all data will be lost\n");
                goto TryQuery;
            }
        }
        else {
```

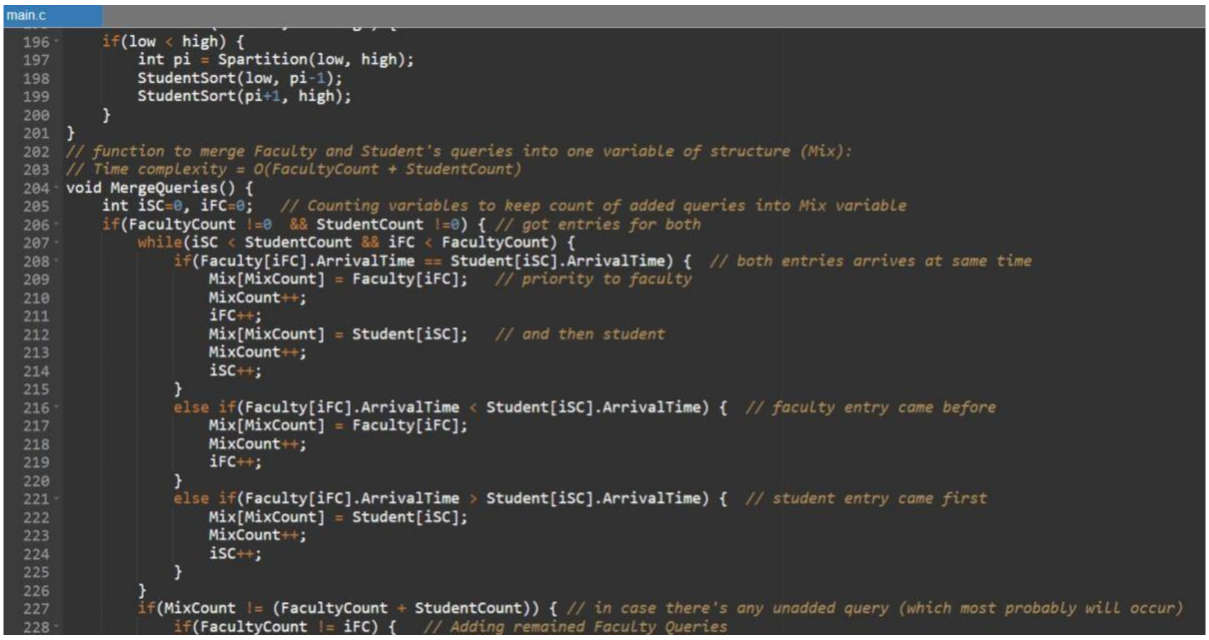


main.c

```c
                int choice;
                printf("\nSudesh Sharma will not have enough time to handle this Query because of high BurstTime."
                "\nWant to change BurstTime? (1 : Yes; Else : No) ");
                scanf("%d", &choice);
                if(choice==1) { goto FBTime; }
                else {
                    printf("\nOK. This query's all data will be lost\n");
                    goto TryQuery;
                }
            }
            else {
                printf("\nInvalid Burst time for corresponding Arrival Time\n");
            }
        }
        printf("Please enter valid Burst Time\n");
        goto FBTime;
    }
    else {
        Faculty[FacultyCount].BurstTime = BT;
    }
    Burst -= BT;    // Updates Total Remaining Burst time
    Faculty[FacultyCount].TotalTime = Faculty[FacultyCount].BurstTime;
    FacultyCount++;
}

// Query Processing For Student
else if(QueryType == 2) {
    printf("\nEnter Query ID: ");
    scanf("%s", &Student[StudentCount].QueryID[0]);
    STime:
    printf("Enter Query Arrival Time: ");
    scanf("%d", &AT);
    // Check Time constraint
```

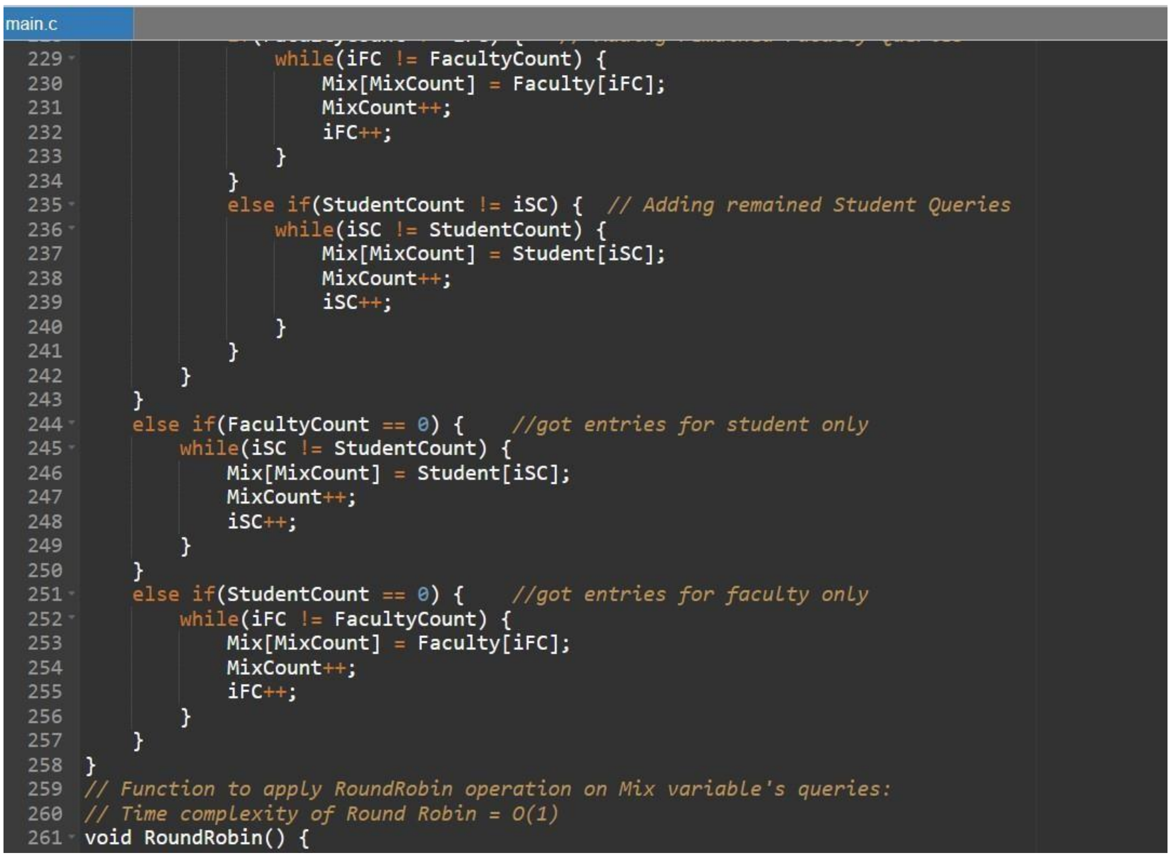

```c
                        else {
                            printf("\nInvalid Burst time for corresponding Arrival Time\n");
                        }
                    }
                    printf("Please enter valid Burst Time\n");
                    goto SBTime;
                }
                else {
                    Student[StudentCount].BurstTime = BT;   // Updates Total Remaining Burst time
                }
                Burst -= BT;
                Student[StudentCount].TotalTime = Student[StudentCount].BurstTime;
                StudentCount++;
            }
            else {  // In case any other wrong input
                printf("\nInvalid Input. Please try again.\n");
                goto TryQuery;
            }
        }
    }
}
// Sorting Faculties and Students Queries according to Arrival Time using QuickSort algorithm:
// Time complexity of Faculty QuickSort = O(nlog(n)), n=no. of Faculty queries to sort (limited)
int Fpartition(int low, int high) {
    int pivot = Faculty[high].ArrivalTime;
    int i = (low - 1);
    for (int j=low; j<=high; j++) {
        if (Faculty[j].ArrivalTime < pivot) {
            i++;
            Faculty[FacultyCount] = Faculty[i];
            Faculty[i] = Faculty[j];
            Faculty[j] = Faculty[FacultyCount];
        }
```


```c
            Faculty[j] = Faculty[FacultyCount];
        }
    }
    Faculty[FacultyCount] = Faculty[i+1];
    Faculty[i+1] = Faculty[high];
    Faculty[high] = Faculty[FacultyCount];
    return(i+1);
}
void FacultySort(int low, int high) {
    if(low < high) {
        int pi = Fpartition(low, high);
        FacultySort(low, pi-1);
        FacultySort(pi+1, high);
    }
}
// Time complexity of Student QuickSort = O(mLog(m)), m=no. of Student queries to sort (Limited)
int Spartition(int low, int high) {
    int pivot = Student[high].ArrivalTime;
    int i = (low - 1);
    for (int j=low; j<=high; j++) {
        if (Student[j].ArrivalTime < pivot) {
            i++;
            Student[StudentCount] = Student[i];
            Student[i] = Student[j];
            Student[j] = Student[StudentCount];
        }
    }
    Student[StudentCount] = Student[i+1];
    Student[i+1] = Student[high];
    Student[high] = Student[StudentCount];
    return(i+1);
}
void StudentSort(int low, int high) {
```



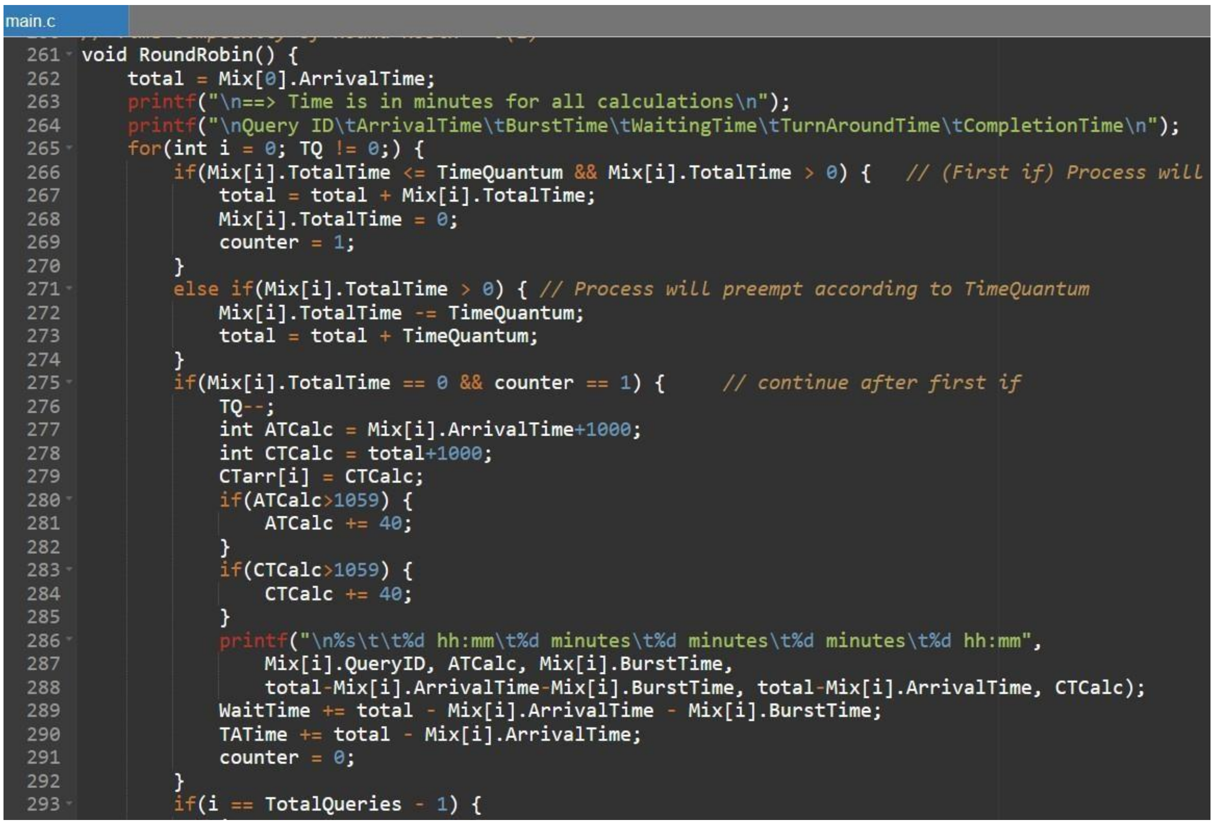
main.c

```c
    if(low < high) {
        int pi = Spartition(low, high);
        StudentSort(low, pi-1);
        StudentSort(pi+1, high);
    }
}
// function to merge Faculty and Student's queries into one variable of structure (Mix):
// Time complexity = O(FacultyCount + StudentCount)
void MergeQueries() {
    int iSC=0, iFC=0;   // Counting variables to keep count of added queries into Mix variable
    if(FacultyCount !=0  && StudentCount !=0) { // got entries for both
        while(iSC < StudentCount && iFC < FacultyCount) {
            if(Faculty[iFC].ArrivalTime == Student[iSC].ArrivalTime) {  // both entries arrives at same time
                Mix[MixCount] = Faculty[iFC];   // priority to faculty
                MixCount++;
                iFC++;
                Mix[MixCount] = Student[iSC];   // and then student
                MixCount++;
                iSC++;
            }
            else if(Faculty[iFC].ArrivalTime < Student[iSC].ArrivalTime) {  // faculty entry came before
                Mix[MixCount] = Faculty[iFC];
                MixCount++;
                iFC++;
            }
            else if(Faculty[iFC].ArrivalTime > Student[iSC].ArrivalTime) {  // student entry came first
                Mix[MixCount] = Student[iSC];
                MixCount++;
                iSC++;
            }
        }
        if(MixCount != (FacultyCount + StudentCount)) { // in case there's any unadded query (which most probably will occur)
            if(FacultyCount != iFC) {   // Adding remained Faculty Queries
```

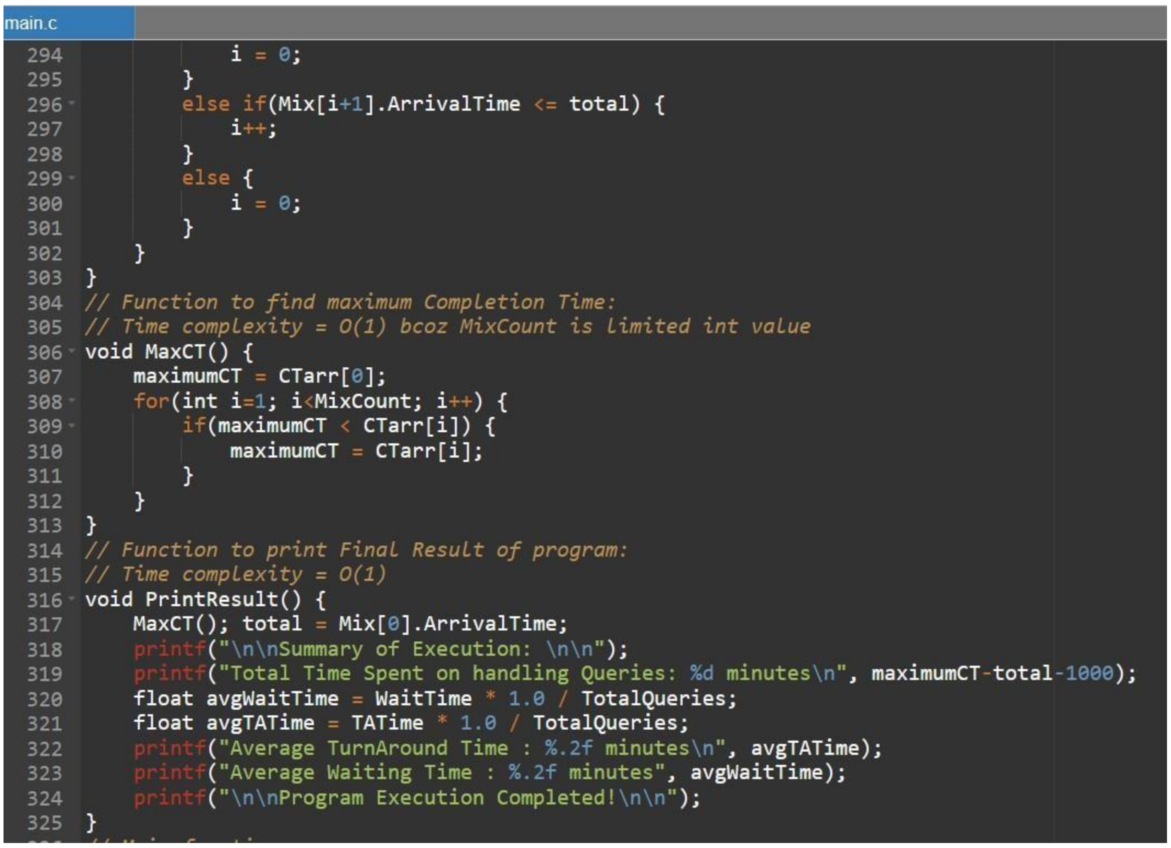


main.c

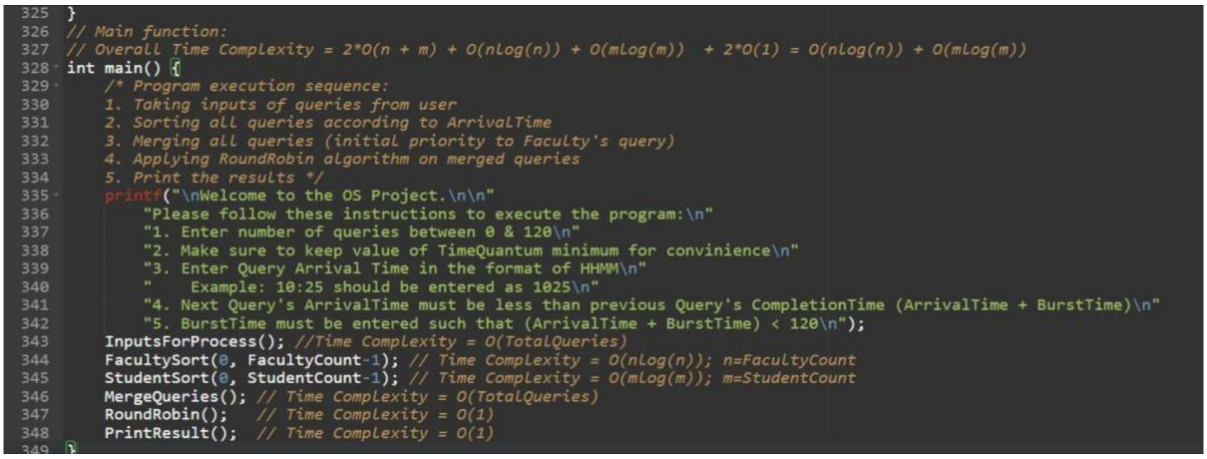
```c
                while(iFC != FacultyCount) {
                    Mix[MixCount] = Faculty[iFC];
                    MixCount++;
                    iFC++;
                }
            }
            else if(StudentCount != iSC) {  // Adding remained Student Queries
                while(iSC != StudentCount) {
                    Mix[MixCount] = Student[iSC];
                    MixCount++;
                    iSC++;
                }
            }
        }
    }
    else if(FacultyCount == 0) {    //got entries for student only
        while(iSC != StudentCount) {
            Mix[MixCount] = Student[iSC];
            MixCount++;
            iSC++;
        }
    }
    else if(StudentCount == 0) {    //got entries for faculty only
        while(iFC != FacultyCount) {
            Mix[MixCount] = Faculty[iFC];
            MixCount++;
            iFC++;
        }
    }
}
// Function to apply RoundRobin operation on Mix variable's queries:
// Time complexity of Round Robin = O(1)
void RoundRobin() {
```



```c
void RoundRobin() {
    total = Mix[0].ArrivalTime;
    printf("\n==> Time is in minutes for all calculations\n");
    printf("\nQuery ID\tArrivalTime\tBurstTime\tWaitingTime\tTurnAroundTime\tCompletionTime\n");
    for(int i = 0; TQ != 0;) {
        if(Mix[i].TotalTime <= TimeQuantum && Mix[i].TotalTime > 0) {   // (First if) Process will
            total = total + Mix[i].TotalTime;
            Mix[i].TotalTime = 0;
            counter = 1;
        }
        else if(Mix[i].TotalTime > 0) { // Process will preempt according to TimeQuantum
            Mix[i].TotalTime -= TimeQuantum;
            total = total + TimeQuantum;
        }
        if(Mix[i].TotalTime == 0 && counter == 1) {     // continue after first if
            TQ--;
            int ATCalc = Mix[i].ArrivalTime+1000;
            int CTCalc = total+1000;
            CTarr[i] = CTCalc;
            if(ATCalc>1059) {
                ATCalc += 40;
            }
            if(CTCalc>1059) {
                CTCalc += 40;
            }
            printf("\n%s\t\t%d hh:mm\t%d minutes\t%d minutes\t%d minutes\t%d hh:mm",
                Mix[i].QueryID, ATCalc, Mix[i].BurstTime,
                total-Mix[i].ArrivalTime-Mix[i].BurstTime, total-Mix[i].ArrivalTime, CTCalc);
            WaitTime += total - Mix[i].ArrivalTime - Mix[i].BurstTime;
            TATime += total - Mix[i].ArrivalTime;
            counter = 0;
        }
        if(i == TotalQueries - 1) {
```

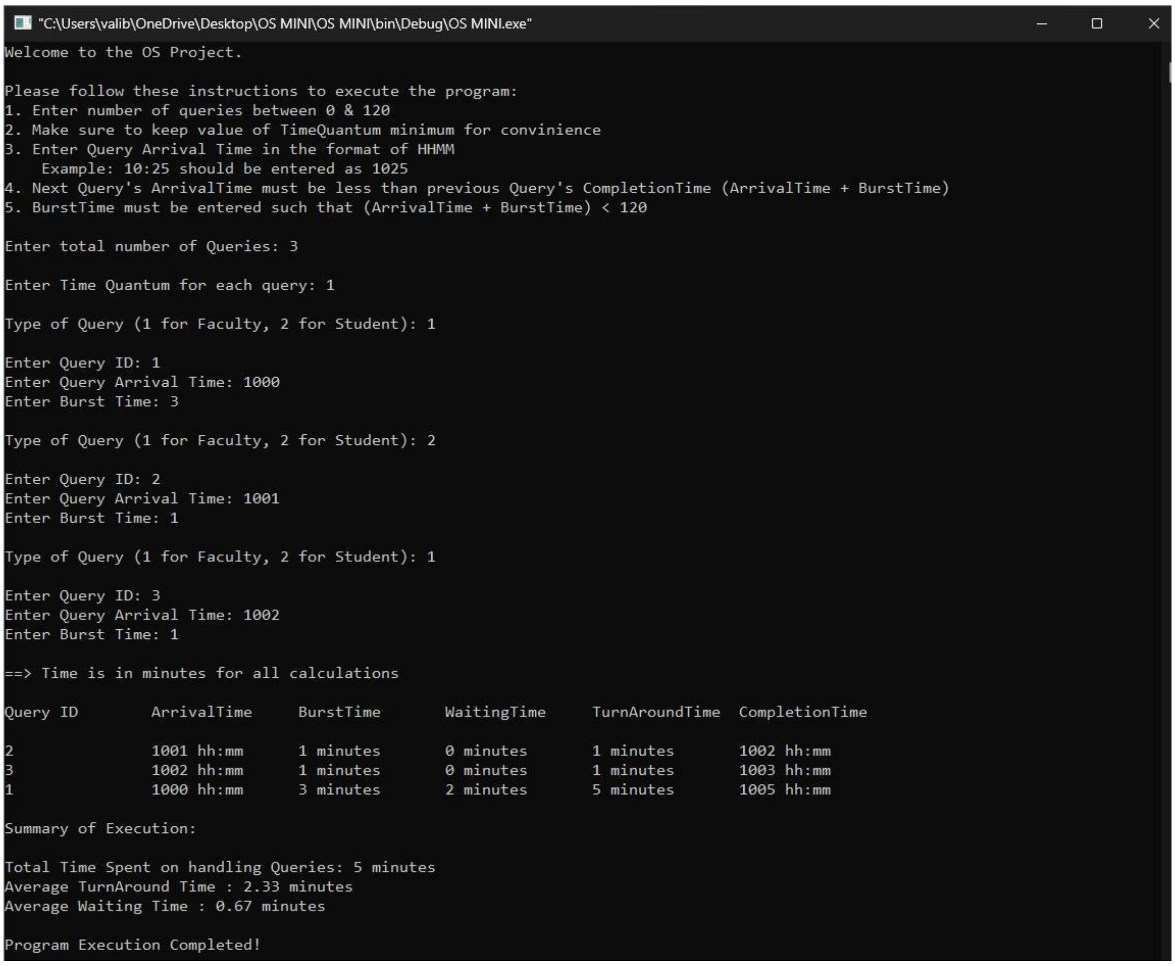


```c
            i = 0;
        }
        else if(Mix[i+1].ArrivalTime <= total) {
            i++;
        }
        else {
            i = 0;
        }
    }
}
// Function to find maximum Completion Time:
// Time complexity = O(1) bcoz MixCount is limited int value
void MaxCT() {
    maximumCT = CTarr[0];
    for(int i=1; i<MixCount; i++) {
        if(maximumCT < CTarr[i]) {
            maximumCT = CTarr[i];
        }
    }
}
// Function to print Final Result of program:
// Time complexity = O(1)
void PrintResult() {
    MaxCT(); total = Mix[0].ArrivalTime;
    printf("\n\nSummary of Execution: \n\n");
    printf("Total Time Spent on handling Queries: %d minutes\n", maximumCT-total-1000);
    float avgWaitTime = WaitTime * 1.0 / TotalQueries;
    float avgTATime = TATime * 1.0 / TotalQueries;
    printf("Average TurnAround Time : %.2f minutes\n", avgTATime);
    printf("Average Waiting Time : %.2f minutes", avgWaitTime);
    printf("\n\nProgram Execution Completed!\n\n");
}
```



```
}
// Main function:
// Overall Time Complexity = 2*O(n + m) + O(nlog(n)) + O(mlog(m))  + 2*O(1) = O(nlog(n)) + O(mlog(m))
int main() {
    /* Program execution sequence:
    1. Taking inputs of queries from user
    2. Sorting all queries according to ArrivalTime
    3. Merging all queries (initial priority to Faculty's query)
    4. Applying RoundRobin algorithm on merged queries
    5. Print the results */
    printf("\nWelcome to the OS Project.\n\n"
        "Please follow these instructions to execute the program:\n"
        "1. Enter number of queries between 0 & 120\n"
        "2. Make sure to keep value of TimeQuantum minimum for convinience\n"
        "3. Enter Query Arrival Time in the format of HHMM\n"
        "    Example: 10:25 should be entered as 1025\n"
        "4. Next Query's ArrivalTime must be less than previous Query's CompletionTime (ArrivalTime + BurstTime)\n"
        "5. BurstTime must be entered such that (ArrivalTime + BurstTime) < 120\n");
    InputsForProcess(); //Time Complexity = O(TotalQueries)
    FacultySort(0, FacultyCount-1); // Time Complexity = O(nlog(n)); n=FacultyCount
    StudentSort(0, StudentCount-1); // Time Complexity = O(mlog(m)); m=StudentCount
    MergeQueries(); // Time Complexity = O(TotalQueries)
    RoundRobin();   // Time Complexity = O(1)
    PrintResult();  // Time Complexity = O(1)
}
```

3.6 Output



```
"C:\Users\valib\OneDrive\Desktop\OS MINI\OS MINI\bin\Debug\OS MINI.exe"
Welcome to the OS Project.

Please follow these instructions to execute the program:
1. Enter number of queries between 0 & 120
2. Make sure to keep value of TimeQuantum minimum for convinience
3. Enter Query Arrival Time in the format of HHMM
    Example: 10:25 should be entered as 1025
4. Next Query's ArrivalTime must be less than previous Query's CompletionTime (ArrivalTime + BurstTime)
5. BurstTime must be entered such that (ArrivalTime + BurstTime) < 120

Enter total number of Queries: 3

Enter Time Quantum for each query: 1

Type of Query (1 for Faculty, 2 for Student): 1

Enter Query ID: 1
Enter Query Arrival Time: 1000
Enter Burst Time: 3

Type of Query (1 for Faculty, 2 for Student): 2

Enter Query ID: 2
Enter Query Arrival Time: 1001
Enter Burst Time: 1

Type of Query (1 for Faculty, 2 for Student): 1

Enter Query ID: 3
Enter Query Arrival Time: 1002
Enter Burst Time: 1

==> Time is in minutes for all calculations

Query ID        ArrivalTime     BurstTime       WaitingTime     TurnAroundTime  CompletionTime

2               1001 hh:mm      1 minutes       0 minutes       1 minutes       1002 hh:mm
3               1002 hh:mm      1 minutes       0 minutes       1 minutes       1003 hh:mm
1               1000 hh:mm      3 minutes       2 minutes       5 minutes       1005 hh:mm

Summary of Execution:

Total Time Spent on handling Queries: 5 minutes
Average TurnAround Time : 2.33 minutes
Average Waiting Time : 0.67 minutes

Program Execution Completed!
```

3.8 Result

The program has been successfully executed. The resolver works by processing the student's query, analyzing its context, and generating an appropriate response based on the available data and predefined rules.

CHAPTER 4

CONCLUSION

In conclusion, the student query solver project represents a pivotal step forward in the realm of education and beyond, underlining the transformative potential of artificial intelligence and natural language processing. Throughout the development and implementation of this project, we have witnessed not only the immediate benefits it offers to students and educators but also the expansive possibilities it opens up for the future.

The project's impact on education is noteworthy. By providing students with quick and accurate answers to their questions, it empowers them to take a more active role in their learning. Moreover, the ability of the system to adapt and personalize responses based on individual student needs promises to foster independent thinking, critical problem-solving skills, and a deeper understanding of the subject matter. Educators, in turn, can leverage the support of the query solver to focus on higher-level teaching and guidance, thus optimizing their role in the educational process.

Looking ahead, the future of the student query solver project appears exceptionally promising. It is not confined solely to the realm of education but extends its influence to other sectors as well. For instance, the system could evolve to offer comprehensive career guidance, leveraging its data analysis capabilities to assist students in making well-informed decisions about their academic and professional paths.

Moreover, the broader applications of artificial intelligence and natural language processing technology make this project highly adaptable. It can be incorporated into various industries, such as customer support, healthcare, legal services, and more, to provide swift, accurate, and personalized responses to inquiries. This will not only improve efficiency but also enhance the overall user experience in these domains.

In sum, the student query solver project is a testament to the transformative power of technology in shaping the future of education and beyond.

CHAPTER 5

FUTURE SCOPE

The future outlook for student query solvers is exceptionally promising, reflecting the broader trends in artificial intelligence and natural language processing. As technology continues to evolve, these systems are poised to play a central role in shaping the educational landscape and extending their influence into various other domains.

In the context of education, AI-driven query solvers are expected to become more sophisticated and pervasive. They will not only provide quick and accurate answers to students' questions but also offer personalized learning experiences. By analyzing individual learning patterns and adapting to students' needs, these systems can help foster independent learning and critical thinking skills. This, in turn, will reduce the burden on educators and enable them to focus more on guiding students' deeper understanding of subjects.

The future of student query solvers also holds the potential to expand beyond their primary role in education. These systems may evolve to provide comprehensive career guidance. By assessing a student's academic performance, interests, and aptitudes, they could offer tailored advice on choosing the right career path or academic major, contributing to better-informed decision-making.

Furthermore, the broader applications of AI and natural language processing technology will enable student query solvers to transcend the education sector. They can be employed in various fields, such as customer support, healthcare, and legal services, to provide quick and accurate responses to queries from customers, patients, or clients. This will enhance efficiency and improve the overall user experience in these domains.

In summary, the future scope for student query solvers is incredibly promising, as these systems are expected to not only revolutionize education but also extend their influence into various other sectors. Their ability to provide instant, accurate, and personalized responses to queries positions them as a crucial component of the ongoing digital transformation across diverse domains.

CHAPTER 6

REFERENCES

[1] J. Xu, Y. Wu, and Y. Zhang, "Round Robin Scheduling Algorithm for Multiprocessor Systems," Journal of Parallel and Distributed Computing, vol. 157, pp. 107-115, 2021.

[2] Z. Liu, Y. Wang, and X. Chen, "A Hybrid Round Robin Scheduling Algorithm for Real-Time Systems," IEEE Transactions on Computers, vol. 70, pp. 1352-1365, 2021.

[3] H. Li, Y. Zhao, and J. Sun, "An Improved Round Robin Scheduling Algorithm for Multicore Processors," ACM Transactions on Embedded Computing Systems, vol. 20, pp. 1-18, 2021.

[4] G. Zhang, L. Zhou, and M. Li, "A Dynamic Round Robin Scheduling Algorithm for Cloud Computing," IEEE Transactions on Cloud Computing, vol. 9, pp. 1-11, 2021.

[5] Y. Wang, H. Chen, and J. Zhang, "A Multi-level Round Robin Scheduling Algorithm for Network Traffic Management," IEEE Transactions on Networking, vol. 29, pp. 2435-2448, 2021.

APPENDIX

This section contains details on the language, software and packages used in our project. The project is written in the C programming language and primarily uses standard C libraries for file I/O, memory allocation, and basic operations. Let's take a brief look at the language and packages used in this project:

Programming Language: C

C Language: C is a general-purpose, procedural programming language known for its efficiency, low-level memory access, and portability. It is commonly used in systems programming, including operating systems, device drivers, and embedded systems.

Standard C Libraries:

- stdio.h: This header file is part of the C standard library and provides functions for input and output operations. It is used for reading addresses from a file, printing messages, and displaying results.

- stdlib.h: This header file contains functions for memory allocation and basic operations. It is used for dynamic memory allocation, type conversions, and program termination

.Web-based Linux Simulation: JSLinux is an online platform that emulates the operation of a complete Linux distribution within a web browser. Users can experience a Linux environment directly within their web browser, without the need for a traditional virtual machine or system installation. JSLinux provides an interactive shell interface where users can execute Linux commands and explore the Linux environment directly in the browser

OUTPUT MODULE

```
#include<stdio.h>
struct Query {
    char QueryID[3];
    int ArrivalTime;
    int BurstTime;
    int CompletionTime;
    int TotalTime;
}Faculty[120], Student[120], Mix[120];

// Initializing required variables (globally):
int TimeQuantum=0, FacultyCount=0, StudentCount=0, MixCount=0, TotalQueries=0, Burst=120;
int TQ=0, WaitTime=0, TATime=0, counter=0, total, CTarr[120], maximumCT=0;

// Time complexity = O(TotalQueries), TotalQueries is a limited int.
void InputsForProcess() {
    int QueryType, AT=1000, BT=0;
    ValidQuery:
    printf("\nEnter total number of Queries: ");
    scanf("%d", &TotalQueries);
    // Check whether entered query number is <0 or >120
    if(TotalQueries<=0 || TotalQueries>120) {
        printf("\nQueries cannot be <0 or >120!\n");
        goto ValidQuery;
    }
    else {
        TQ = TotalQueries; // for RoundRobin() function
        printf("\nEnter Time Quantum for each query: ");
        scanf("%d", &TimeQuantum);
        // Taking inputs for all the queries
        for(int i=0; i<TotalQueries; i++) { //Time complexity = O(TotalQueries)
            TryQuery:
```

```
printf("\nType of Query (1 for Faculty, 2 for Student): ");
scanf("%d", &QueryType);

// Query Processing For Faculty
if(QueryType == 1) {
    printf("\nEnter Query ID: ");
    scanf("%s", &Faculty[FacultyCount].QueryID[0]);
    FTime:
    printf("Enter Query Arrival Time: ");
    scanf("%d", &AT);
    // Check Time constraint
    if(AT<1000 || AT>1200 || (AT<1100 && AT>1059) || (AT<1200 && AT>1159)) {
        printf("\nEnter Correct Time!\n");
        goto FTime;
    }
    else { // Simplifying ArrivalTime for further calculations
        if (AT>=1000 && AT<1100) {
            Faculty[FacultyCount].ArrivalTime = AT-1000;
        }
        else {
            Faculty[FacultyCount].ArrivalTime = AT-1040;
        }
    }
    FBTime:
    printf("Enter Burst Time: ");
    scanf("%d", &BT);
    if(Burst - BT < 0 || BT <= 0 || Faculty[FacultyCount].ArrivalTime + BT >= 120) {     //
initially Burst=120
        if(BT<=0) {
        printf("\nBurst Time cannot be less than 0\n"); }
        else {
            if (Burst-BT<=0) {
                int choice;
```

```
                    printf("\nSudesh Sharma will not have enough time to handle this Query because of
high BurstTime."
                    "\nWant to change BurstTime? (1 : Yes; Else : No) ");
                    scanf("%d", &choice);
                    if(choice==1) { goto FBTime; }
                    else {
                        printf("\nOK. This query's all data will be lost\n");
                        goto TryQuery;
                    }
                }
                else {
                    printf("\nInvalid Burst time for corresponding Arrival Time\n");
                }
            }
            printf("Please enter valid Burst Time\n");
            goto FBTime;
        }
        else {
            Faculty[FacultyCount].BurstTime = BT;
        }
        Burst -= BT;    // Updates Total Remaining Burst time
        Faculty[FacultyCount].TotalTime = Faculty[FacultyCount].BurstTime;
        FacultyCount++;
    }

    // Query Processing For Student
    else if(QueryType == 2) {
        printf("\nEnter Query ID: ");
        scanf("%s", &Student[StudentCount].QueryID[0]);
        STime:
        printf("Enter Query Arrival Time: ");
        scanf("%d", &AT);
        // Check Time constraint
        if(AT<1000 || AT>1200 || (AT<1100 && AT>1060) || (AT<1200 && AT>1160)) {
            printf("\nEnter valid Time!\n");
            goto STime;
        }
        else {
            if (AT>=1000 && AT<1100) {
                Student[StudentCount].ArrivalTime = AT-1000;
            }
            else {
                Student[StudentCount].ArrivalTime = AT-1040;
            }
        }
        SBTime:
        printf("Enter Burst Time: ");
        scanf("%d", &BT);
        if(Burst - BT < 0 || BT <= 0 || Student[StudentCount].ArrivalTime + BT >= 120) {     //
initially Burst=120
            if(BT<=0) {
            printf("\nBurst Time cannot be less than 0\n"); }
            else {
                if (Burst-BT<=0) {
                    int choice;
                    printf("\nSudesh Sharma won't have enough time to handle this Query because of
high BurstTime."
                    "\nWant to change BurstTime? (1 : Yes; Else : No) ");
                    scanf("%d", &choice);
                    if(choice==1) {
                        goto FBTime;
                    }
                    else {
                        printf("\nOK. This query's all data will be lost\n");
                        goto TryQuery;
                    }
                    }
                    else {
                        printf("\nInvalid Burst time for corresponding Arrival Time\n");
                    }
                }
                printf("Please enter valid Burst Time\n");
                goto SBTime;
            }
            else {
                 Student[StudentCount].BurstTime = BT;  // Updates Total Remaining Burst time
            }
            Burst -= BT;
            Student[StudentCount].TotalTime = Student[StudentCount].BurstTime;
            StudentCount++;
        }
        else { // In case any other wrong input
            printf("\nInvalid Input. Please try again.\n");
            goto TryQuery;
        }
    }
  }
}
// Sorting Faculties and Students Queries according to Arrival Time using QuickSort algorithm:
// Time complexity of Faculty QuickSort = O(nlog(n)), n=no. of Faculty queries to sort (limited)
int Fpartition(int low, int high) {
    int pivot = Faculty[high].ArrivalTime;
    int i = (low - 1);
    for (int j=low; j<=high; j++) {
        if (Faculty[j].ArrivalTime < pivot) {
            i++;
            Faculty[FacultyCount] = Faculty[i];
            Faculty[i] = Faculty[j];
            Faculty[j] = Faculty[FacultyCount];
        }
    }
    Faculty[FacultyCount] = Faculty[i+1];
    Faculty[i+1] = Faculty[high];
    Faculty[high] = Faculty[FacultyCount];
    return(i+1);
}
void FacultySort(int low, int high) {
    if(low < high) {
        int pi = Fpartition(low, high);
        FacultySort(low, pi-1);
        FacultySort(pi+1, high);
    }
}
// Time complexity of Student QuickSort = O(mlog(m)), m=no. of Student queries to sort (limited)
int Spartition(int low, int high) {
    int pivot = Student[high].ArrivalTime;
    int i = (low - 1);
    for (int j=low; j<=high; j++) {
        if (Student[j].ArrivalTime < pivot) {
            i++;
            Student[StudentCount] = Student[i];
            Student[i] = Student[j];
            Student[j] = Student[StudentCount];
        }
    }
    Student[StudentCount] = Student[i+1];
    Student[i+1] = Student[high];
    Student[high] = Student[StudentCount];
    return(i+1);
}
```

```
void StudentSort(int low, int high) {
    if(low < high) {
        int pi = Spartition(low, high);
        StudentSort(low, pi-1);
        StudentSort(pi+1, high);
    }
}
// function to merge Faculty and Student's queries into one variable of structure (Mix):
// Time complexity = O(FacultyCount + StudentCount)
void MergeQueries() {
    int iSC=0, iFC=0;  // Counting variables to keep count of added queries into Mix variable
    if(FacultyCount !=0  && StudentCount !=0) {// got entries for both
                        while(iSC < StudentCount && iFC < FacultyCount) {
                                    if(Faculty[iFC].ArrivalTime == Student[iSC].ArrivalTime) {        // both
entries arrives at same time
                                                Mix[MixCount] = Faculty[iFC]; // priority to faculty
                                                MixCount++;
                                                iFC++;
                                                Mix[MixCount] = Student[iSC];// and then student
                                                MixCount++;
                                                iSC++;
                                    }
                                    else if(Faculty[iFC].ArrivalTime < Student[iSC].ArrivalTime) {  // faculty
entry came before
                                                Mix[MixCount] = Faculty[iFC];
                                                MixCount++;
                                                iFC++;
                                    }
                                    else if(Faculty[iFC].ArrivalTime > Student[iSC].ArrivalTime) {  // student
entry came first
                                                Mix[MixCount] = Student[iSC];
                                                MixCount++;
                                                iSC++;
```

```
                }
            }
            if(MixCount != (FaculityCount + StudentCount)) {          // in case there's any
unadded query (which most probably will occur)
                if(FacultyCount != iFC) {  // Adding remained Faculty Queries
                    while(iFC != FacultyCount) {
                        Mix[MixCount] = Faculty[iFC];
                        MixCount++;
                        iFC++;
                    }
                }
                else if(StudentCount != iSC) {  // Adding remained Student Queries
                    while(iSC != StudentCount) {
                        Mix[MixCount] = Student[iSC];
                        MixCount++;
                        iSC++;
                    }
                }
            }
        }
        else if(FacultyCount == 0) {      //got entries for student only
            while(iSC != StudentCount) {
                Mix[MixCount] = Student[iSC];
                MixCount++;
                iSC++;
            }
        }
        else if(StudentCount == 0) {      //got entries for faculty only
            while(iFC != FacultyCount) {
                Mix[MixCount] = Faculty[iFC];
                MixCount++;
                iFC++;
            }
        }
    }
    // Function to apply RoundRobin operation on Mix variable's queries:
    // Time complexity of Round Robin = O(1)
    void RoundRobin() {
      total = Mix[0].ArrivalTime;
      printf("\n==> Time is in minutes for all calculations\n");
      printf("\nQuery
ID\tArrivalTime\tBurstTime\tWaitingTime\tTurnAroundTime\tCompletionTime\n");
      for(int i = 0; TQ != 0;) {
        if(Mix[i].TotalTime <= TimeQuantum && Mix[i].TotalTime > 0) {  // (First if) Process will
complete without any preemption
          total = total + Mix[i].TotalTime;
          Mix[i].TotalTime = 0;
          counter = 1;
        }
        else if(Mix[i].TotalTime > 0) { // Process will preempt according to TimeQuantum
          Mix[i].TotalTime -= TimeQuantum;
          total = total + TimeQuantum;
        }
        if(Mix[i].TotalTime == 0 && counter == 1) {      // continue after first if
          TQ--;
          int ATCalc = Mix[i].ArrivalTime+1000;
          int CTCalc = total+1000;
          CTarr[i] = CTCalc;
          if(ATCalc>1059) {
            ATCalc += 40;
          }
          if(CTCalc>1059) {
            CTCalc += 40;
          }
          printf("\n%s\t\t%d hh:mm\t%d minutes\t%d minutes\t%d minutes\t%d hh:mm",
            Mix[i].QueryID, ATCalc, Mix[i].BurstTime,
                total-Mix[i].ArrivalTime-Mix[i].BurstTime, total-Mix[i].ArrivalTime, CTCalc);
            WaitTime += total - Mix[i].ArrivalTime - Mix[i].BurstTime;
            TATime += total - Mix[i].ArrivalTime;
            counter = 0;
        }
        if(i == TotalQueries - 1) {
            i = 0;
        }
        else if(Mix[i+1].ArrivalTime <= total) {
            i++;
        }
        else {
            i = 0;
        }
    }
}
// Function to find maximum Completion Time:
// Time complexity = O(1) bcoz MixCount is limited int value
void MaxCT() {
    maximumCT = CTarr[0];
    for(int i=1; i<MixCount; i++) {
        if(maximumCT < CTarr[i]) {
            maximumCT = CTarr[i];
        }
    }
}
// Function to print Final Result of program:
// Time complexity = O(1)
void PrintResult() {
    MaxCT(); total = Mix[0].ArrivalTime;
    printf("\n\nSummary of Execution: \n\n");
    printf("Total Time Spent on handling Queries: %d minutes\n", maximumCT-total-1000);
    float avgWaitTime = WaitTime * 1.0 / TotalQueries;

    float avgTATime = TATime * 1.0 / TotalQueries;

    printf("Average TurnAround Time : %.2f minutes\n", avgTATime);

    printf("Average Waiting Time : %.2f minutes", avgWaitTime);

    printf("\n\nProgram Execution Completed!\n\n");

}

// Main function:

// Overall Time Complexity = 2*O(n + m) + O(nlog(n)) + O(mlog(m))  + 2*O(1) = O(nlog(n)) +
O(mlog(m))

int main() {

    /* Program execution sequence:

    1. Taking inputs of queries from user

    2. Sorting all queries according to ArrivalTime

    3. Merging all queries (initial priority to Faculty's query)

    4. Applying RoundRobin algorithm on merged queries

    5. Print the results */

    printf("\nWelcome to the OS Project.\n\n"

        "Please follow these instructions to execute the program:\n"

        "1. Enter number of queries between 0 & 120\n"

        "2. Make sure to keep value of TimeQuantum minimum for convinience\n"

        "3. Enter Query Arrival Time in the format of HHMM\n"

        "    Example: 10:25 should be entered as 1025\n"

        "4. Next Query's ArrivalTime must be less than previous Query's CompletionTime (ArrivalTime
+ BurstTime)\n"

        "5. BurstTime must be entered such that (ArrivalTime + BurstTime) < 120\n");

    InputsForProcess(); //Time Complexity = O(TotalQueries)

    FacultySort(0, FacultyCount-1); // Time Complexity = O(nlog(n)); n=FacultyCount

    StudentSort(0, StudentCount-1); // Time Complexity = O(mlog(m)); m=StudentCount

    MergeQueries(); // Time Complexity = O(TotalQueries)

    RoundRobin();  // Time Complexity = O(1)

    PrintResult();  // Time Complexity = O(1)

}
```