**SECR2043 OPERATING SYSTEMS**

**SECTION 10 SESSION 2 2024/2025**

**FACULTY OF COMPUTING**

**GROUP PROJECT REPORT**

**LECTURER NAME:  DR ADNAN KHALID**

**GROUP: 02**

# BANKING LOAN MANAGEMENT SYSTEM

**PREPARED BY:**

| GROUP MEMBERS | MATRIC NUMBER |
|---|---|
| ABDUR RAHMAN | A23CS005 |
| MUHAMMAD NUR MIFTAHUL RAHMAN | A23CS0293 |
| BAQIR TSAQIB HAKIM | A23CS4039 |
| AVISHEK SAHA | A23CS0011 |

## TABLE OF CONTENTS

## Abstract

This  Project report explains how we built and tested the Bankers' Algorithm in a  banking loan Management system, using what we learned in our Operating System course. It looks at how to handle resource sharing and avoid deadlocks by applying the Bankers' Algorithm in a real example. The system was made using C++. The report provides an overview of the project's structure, including the introduction, problem statement, related works, methodology, design and implementation, results and discussion, and future work and conclusion. The demo shows how the algorithm helps manage resources and avoid deadlocks in the system.

## 1.0 Introduction

This report is about our project for the Operating Systems course, where we put theory into practice by building a bank loan management system that uses the Bankers' Algorithm. Operating systems help computers run smoothly by managing tasks and resources, and this project focuses on how the Bankers' Algorithm supports that — especially in handling resource sharing and avoiding deadlocks.

The main goal is to show how the algorithm works in a real system. We explain why we chose this topic, who it’s for, and what we hope to achieve. In the related work section, we connect our project with what we’ve learned about operating systems. The methodology part explains our planning process, what tools we used, and how we divided tasks among team members.

We also include designs and implementation , showing how the algorithm works step by step using animations. The results section explains how the system works and how well it manages resources. Lastly, the future work and conclusion section talks about the challenges we faced, what we learned, and how useful the Bankers' Algorithm is in real situations.

In short, this project uses coding and animation to clearly show how the Bankers' Algorithm works in a bank loan management system. It helps us and others better understand how operating systems handle resources and avoid problems like deadlocks.

## 2.0 Problem Statement

In a bank loan management system, managing resources and avoiding deadlocks is important to keep services running smoothly. Even though the Banker's Algorithm is used to prevent deadlocks, there are still situations where they can happen. This project focuses on analyzing when and why these potential deadlocks occur, even with the algorithm in place. By looking closely at how the system works and how resources are given out, we aim to find the exact cases that lead to problems. The goal is to understand what's missing in the current setup and suggest improvements to better prevent deadlocks and keep the system working without interruptions

## 3.0 Related Works

Effective techniques are needed to manage finite resources, such as loan capital, in real-world banking systems in order to guarantee equitable distribution and system stability. The system must avoid conflicts and deadlocks that could stop operations when several clients apply for loans at the same time. The Banker's  Algorithm's tenets are applied here. This algorithm, which was first put forth by Edsger Dijkstra, simulates whether the system will stay in a "safe state" prior to completing a resource request in order to prevent deadlocks.

Customers (processes) dynamically request and release a limited amount of funds (resources) in a banking system. For instance, even if a customer's loan request is approved, there must still be enough money left over to meet the needs of every other customer in the system. Otherwise, the system might experience a deadlock, in which certain users are left waiting for resources indefinitely. This is resolved by the Banker's Algorithm, which rejects any request that would put the system in danger.

Despite being theoretical, this algorithm captures actual risk-averse resource management techniques. Similar reasoning is frequently used by banking software systems, which assess overall liquidity and risk before granting a loan to make sure continuing operations won't be compromised. Sites like GeeksforGeeks and StudyTonight describe how this algorithm determines whether it is safe to grant a new loan or if it should be postponed by looking at available funds, remaining needs, and current allocations.

A C++ application that allows multiple users to request and repay loans is used in our project to simulate this model. Every request is examined by the system using the Banker's Algorithm, which guarantees that each approval preserves a secure system state. The project closes the gap between deadlock avoidance theory and real-world banking situations by putting this logic into practice.

## 4.0 Methodology

---

1. Definition of the Problem

   When allocating loan requests concurrently without first confirming system stability, be aware of the risk of deadlock, or insolvency as it is known in banking.

2. Collecting Requirements

   Indicate that the system needs to:

   - Take in loan applications from several "customers" (processes).
   - Ask each client to disclose their maximum loan amount up front.
   - Only simulate approvals if the system is kept safe by the remaining funds (total deposit pool less committed loans).

3. System Design:
   - Use the matrices Available, Max, Allocation, and Need to model resources (available funds), processes (customers), and claims.
   - Apply the safety algorithm by simulating allocations and determining whether a sequence exists that permits all processes to finish and release resources in a safe manner.
   - Utilize the Resource-Request Algorithm: when a loan request is made, the funds are initially allocated, a safety check is performed, and the results are confirmed or rolled back accordingly.
4. Implementation:
   - C++ was chosen for its low-level control and compatibility with operating system concepts.
   - Vectors and matrices for Available, Max, Allocation, and Need are examples of data structures.

    - The system employs a two-step algorithm to determine whether to approve or wait after each customer process specifies its maximum and moves on to the request phase.

5. Testing:
    - Create test cases with a range of loan demands and customer counts, including edge cases where the total amount of funds is approached by the cumulative requests.
    - Verify that the system recognizes safe sequences and rejects unsafe requests.
6. Demonstration:
    - Concurrent loan requests, safe checks, grant/reject decisions, and system state changes will all be demonstrated through a screen capture.

## 5.0 Design & Implementation

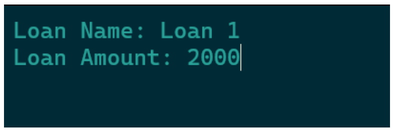
In this study, a program will be developed to simulate the mechanism behind loaning in the banking system. This simulation will implement the safety algorithm or known as banker algorithm, a popular deadlock avoidance method. The system will be developed using C++ programming language.

This program will have two roles: user and admin. Users are able to request a loan by prompting their loan identifier as well as its amount. The system then examines the user request based on predefined criteria. The criteria is such that: (1) the sum of the requested loan amount and the existing unpaid loan does not exceed the user's loan limit and (2) the system will be in safe state when accepting the user's loan request. The system will reject the requested loan if at least one of the criteria is not met. Otherwise, it will be accepted and saved to a text file. Furthermore, the user can also pay their accepted loan. This will act as a user releasing their resource and allowing the system to relocate its fund to another account. In addition to this, admin can regulate the maximum money the bank can loan. This attribute is crucial in determining whether the system is in safe or unsafe state using a safety algorithm. In order to access this program, both user and admin are required to enter their authentication data (e.g. their username and password).

## 6.0 Results & Discussion

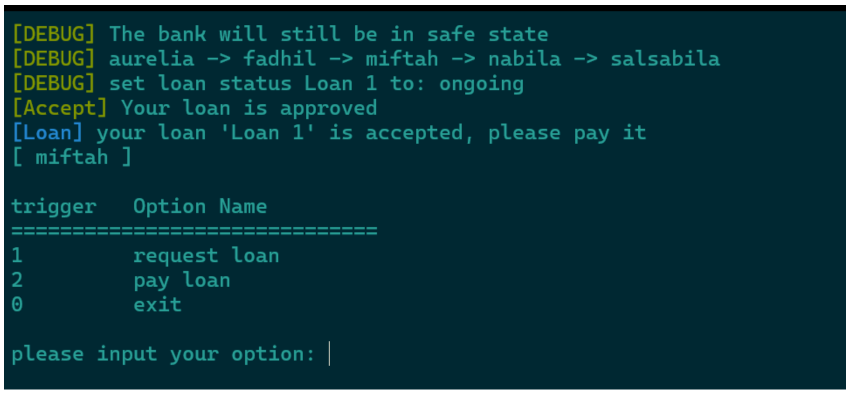
The code base for this study can be accessed in github repo as follows: https://github.com/miftahr149/OS-Project.

The first feature is user request loan. Figure 6.1 below depicts the interface on the user prompting their requested loan information. After prompting the information, the user will get one of the notifications shown in figure 6.2, 6.3, and 6.4, representing the accepted request loan, rejected request loan due to exceeding its loan limit, and rejected request loan due to unsafe state respectively. Here, it can be seen that the program will display the result of the banker's algorithm if in the safe state. Furthermore, figure 6.5 and 6.6 shows the user interface for paying the loan. In addition to this, figure 6.7 and 6.8 shows the interface for admin to set a new bank loan limit.



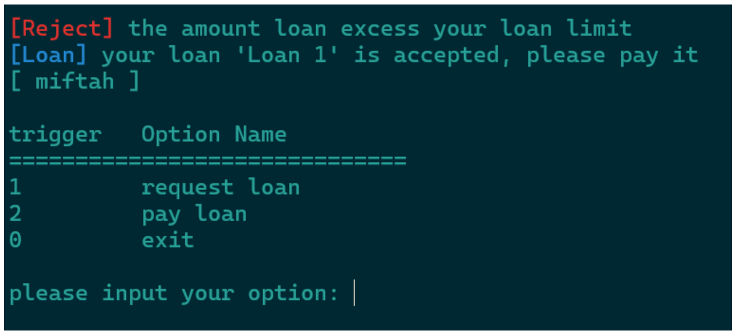
```
Loan Name: Loan 1
Loan Amount: 2000
```

Figure 6.1: User requesting loan interface



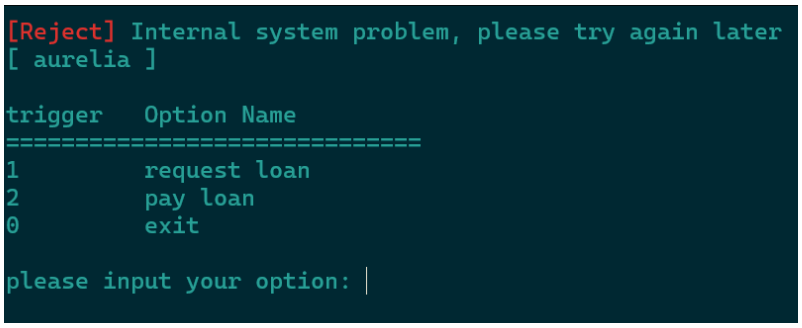
```
[DEBUG] The bank will still be in safe state
[DEBUG] aurelia -> fadhil -> miftah -> nabila -> salsabila
[DEBUG] set loan status Loan 1 to: ongoing
[Accept] Your loan is approved
[Loan] your loan 'Loan 1' is accepted, please pay it
[ miftah ]

trigger   Option Name
==============================
1         request loan
2         pay loan
0         exit

please input your option:
```

Figure 6.2: Accepted loan interface



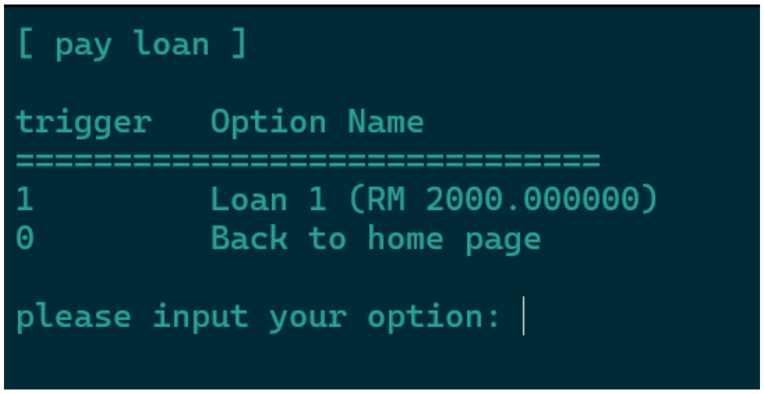
```
[Reject] the amount loan excess your loan limit
[Loan] your loan 'Loan 1' is accepted, please pay it
[ miftah ]

trigger   Option Name
==============================
1         request loan
2         pay loan
0         exit

please input your option:
```

Figure 6.3: Excess loan limit request result



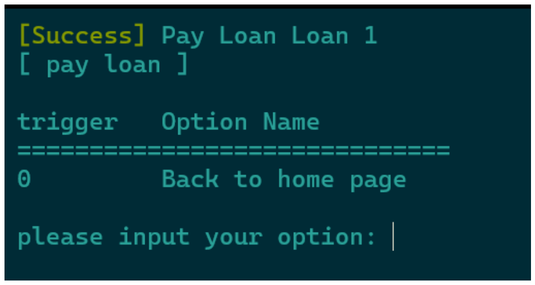
```
[Reject] Internal system problem, please try again later
[ aurelia ]

trigger   Option Name
==============================
1         request loan
2         pay loan
0         exit

please input your option:
```

Figure 6.4: Unsafe state result



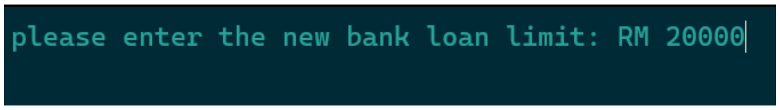
```
[ pay loan ]

trigger   Option Name
==============================
1         Loan 1 (RM 2000.000000)
0         Back to home page

please input your option:
```

Figure 6.5: Pay Loan Interface



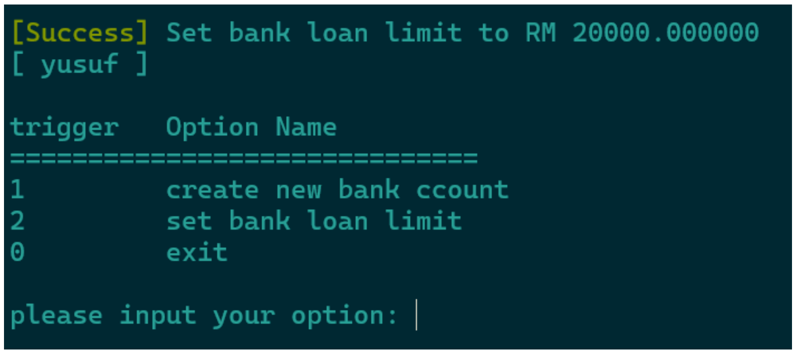
```
[Success] Pay Loan Loan 1
[ pay loan ]

trigger   Option Name
==============================
0         Back to home page

please input your option: 
```

Figure 6.6: Success Pay Loan Notification



```
please enter the new bank loan limit: RM 20000
```

Figure 6.7: Set Bank Maximum Amount Loan



```
[Success] Set bank loan limit to RM 20000.000000
[ yusuf ]

trigger   Option Name
==============================
1         create new bank ccount
2         set bank loan limit
0         exit

please input your option: 
```

Figure 6.8: Result Setting New Bank Loan Limit

## 7.0 Future Work & Conclusion

---

Since the Bankers' Algorithm has been successfully implemented into our project Banking Loan Management System, we have gained significant knowledge regarding how operating systems manage resources and prevent deadlocks. Our project has shown how these concepts can be superimposed on banking issues where loan applications have to be effectively controlled so that stability at the system level is maintained. Although, there were some things deemed imperfect in the system to lead to future enhancements that would make the system more feasible, user-friendly, and scalable.

Some users may find it difficult to utilize the command line interface for system operations. At some instance, the Graphical User Interface (GUI) can be established with any suitable framework for web technology. GUI might visualize loan request results, accepted, rejected due to limits on loans, or rejected due to unsafe states in fine dashboards, thus aiding the user to grasp system behaviour much better.

The system keeps on assuming one type of resource, which is simply the available bank fund. Enhancements in the system might be engaging in the introduction of handling multiple resources, various types of loan categories such as personal, housing, and business loans, each having distinct constraints. This would only make simulation more close to real-world banking scenarios where multiple funds or departments operate simultaneously. Presently, in our system, loans are written to a text file. Next would be connecting this system to a relational database, such as MySQL, for storing user data, transaction history, and system logs in a secure manner. This would support querying and auditing as well as scaling to larger, real banking environments.

In conclusion, the project gave a perfect opportunity to study the Bankers' Algorithm and its utility in real systems like banking loan management systems. While simulating loan requests and testing system safety with C++, we learned how to implement what we had learned in class on Operating Systems. We came to understand this algorithm as a means to avoid deadlocks and to maintain system stability while, for instance, multiple users simultaneously apply for loans. The project aided us in system design and resource allocation. The project, hence, boosted our problem-solving and teamwork abilities. Thus, it also enhanced our understanding of Operating Systems topics, such as avoiding deadlocks, and gave us practical experience in developing a realistic system.

## Acknowledgement

We want to sincerely thank everyone who supported us during the development of the Banking Loan Management System project.

Primarily, our deepest gratitude goes to our lecturer, Dr. Adnan Khalid, for his help, guidance, and encouragement from the beginning of our semester till the completion of this project. His clear explanations helped us understand concepts better and apply them in our building. In addition, the sacrifices he made, teaching us at odd times, truly made a difference in our learning experience.

Lastly, we want to thank our team members and classmates who worked together, shared ideas, and supported each other during this project. Their cooperation and teamwork made the process smoother and more enjoyable.

## References

---

1. GeeksforGeeks. (2024). *Banker's Algorithm in Operating System*. Retrieved July 4, 2025, from https://www.geeksforgeeks.org/operating-systems/bankers-algorithm-in-operating-system-2/
2. Scaler Topics. (2024). *Banker's Algorithm in OS*. Retrieved July 4, 2025, from https://www.scaler.com/topics/bankers-algorithm-in-os/
3. StudyTonight. (n.d.). *Banker's Algorithm in Operating System*. Retrieved July 4, 2025, from https://www.studytonight.com/operating-system/bankers-algorithm
4. Wikipedia. (2025). *Banker's Algorithm*. Retrieved July 4, 2025, from https://en.wikipedia.org/wiki/Banker%27s_algorithm