Bank ATM Transaction Deadlock

A MINI-PROJECT REPORT

Submitted by

SANDEEP S	**231901045**
THENMOZHI R	**231901064**
SABTHAGIRI A	**231901044**

in partial fulfilment of the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING (CYBER SECURITY)

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute

CHENNAI
MAY 2025

BONAFIDE CERTIFICATE

Certified that this project report "**BANK ATM TRANSACTION DEADLOCK**" is the Bonafide work of **"SANDEEP S (231901045), THENMOZHI R (231901064), SABTHAGIRI A (231901044)"** who carried out the project work under my supervision.

Submitted for the Practical Examination held on ______________

SIGNATURE

Ms. V. JANANEE

Assistant Professor (SG),

Computer Science and Engineering,

Rajalakshmi Engineering College,

(Autonomous),

Thandalam, Chennai - 602 105.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We express our sincere thanks to our beloved and honourable chairman **MR.S. MEGANATHAN** and the chairperson **DR.M. THANGAM MEGANATHAN** for their timely support and encouragement.

We are greatly indebted to our respected and honourable principal **Dr. S.N. MURUGESAN**, for his able support and guidance.

We express our sincere thanks to our Head of the Department **MR. Benedict JN**, for encouragement and being ever supporting force during our project work.

We also extend our sincere and hearty thanks to our internal guide **Ms. V. JANANEE,** for her valuable guidance and motivation during the completion of this project.

Our sincere thanks to our family members, friends and other staff members of computer science engineering.

1. SANDEEP S	231901045
2.THENMOZHI R	231901064
3.SABTHAGIRI A	231901044

ABSTRACT

This mini project focuses on transaction deadlocks in Automated Teller Machines (ATMs), where simultaneous transactions cause delays and failures. It explores causes like resource contention and concurrency issues, proposing solutions such as timeout mechanisms and resource allocation strategies to detect, prevent, and recover from deadlocks, enhancing ATM reliability. To develop a banker's Algorithm simulation using python with Tkinter /pyqt to provides an interactive visualization of resource allocation. User can input processes, available, resource, and maximum demands and the system will determine whether a request leads to a safe or unsafe state. This simulation will display the safe sequence highlight deadlock risks and prevent unsafe allocation. Key features include dynamic resource requests, system state visualization deadlock prevention Checks this project serves as an educational tool, helping and professionals understand resource management in multi-processing environments. Our findings contribute to enhancing the robustness of ATM systems, ensuring higher reliability and customer satisfaction in banking operations. A key component of this project is the development of an interactive Banker's Algorithm simulation using Python and GUI frameworks like Tkinter or PyQt. The simulation provides a visual and educational platform where users can input the number of processes, available resources, allocated resources, and maximum demands. Based on these inputs, the system evaluates whether a particular resource request leads to a safe or unsafe state. The simulation highlights safe execution sequences, flags deadlock risks, and prevents unsafe allocations before they occur, enabling users to visualize and understand complex resource allocation behaviour in real time.

TABLE OF CONTENTS

CHAPTER-1

INTRODUCTION

In today's fast-paced banking environment, Automated Teller Machines (ATMs) are essential for providing customers with 24/7 access to financial services such as withdrawals, deposits, and balance inquiries. These services often involve multiple users accessing shared banking resources simultaneously, leading to challenges in managing concurrent transactions. One critical issue that can arise is a deadlock, where two or more processes wait indefinitely for resources held by each other, causing the system to halt and affecting service availability. This mini project, titled "Bank ATM Transaction Deadlock Simulation," aims to simulate concurrent ATM transactions and demonstrate how deadlocks can occur in a multi-user environment. It also explores techniques for preventing or resolving deadlocks, ensuring smooth and uninterrupted transaction processing. The project includes a performance comparison between two scenarios—deadlock occurs and deadlock does not occur—through an efficiency vs. time graph. This visual analysis helps illustrate the importance of efficient resource management and concurrency control in banking systems.

1.1 KEY FEATURES

- **Deadlock Simulation**

  Simulates a realistic banking environment where multiple ATM transactions run concurrently, potentially leading to deadlock situations.

- **Multi-User Transaction Handling**

  Models multiple users accessing shared banking resources like account balances, demonstrating real-time concurrent operations.

- **Deadlock Detection and Prevention**

Implements techniques such as locking protocols, resource ordering, or wait-die/wound-wait schemes to avoid or handle deadlocks.

- **Efficiency vs. Time Graph**

Provides a comparative graph to visualize system performance under two scenarios—**deadlock occurs** vs. **deadlock avoided**—showing impact on processing time and efficiency.

- **Realistic Banking Operations**

Includes typical ATM operations like withdrawal, deposit, and balance check, making the simulation relatable and practical.

- **Concurrency Control Mechanisms**

Demonstrates concepts from operating systems and databases, such as mutual exclusion, hold and wait, and resource allocation.

- **Educational Tool**

Enhances understanding of deadlock concepts, resource management, and system performance analysis in concurrent system

CHAPTER 2

ARCHITECTURE

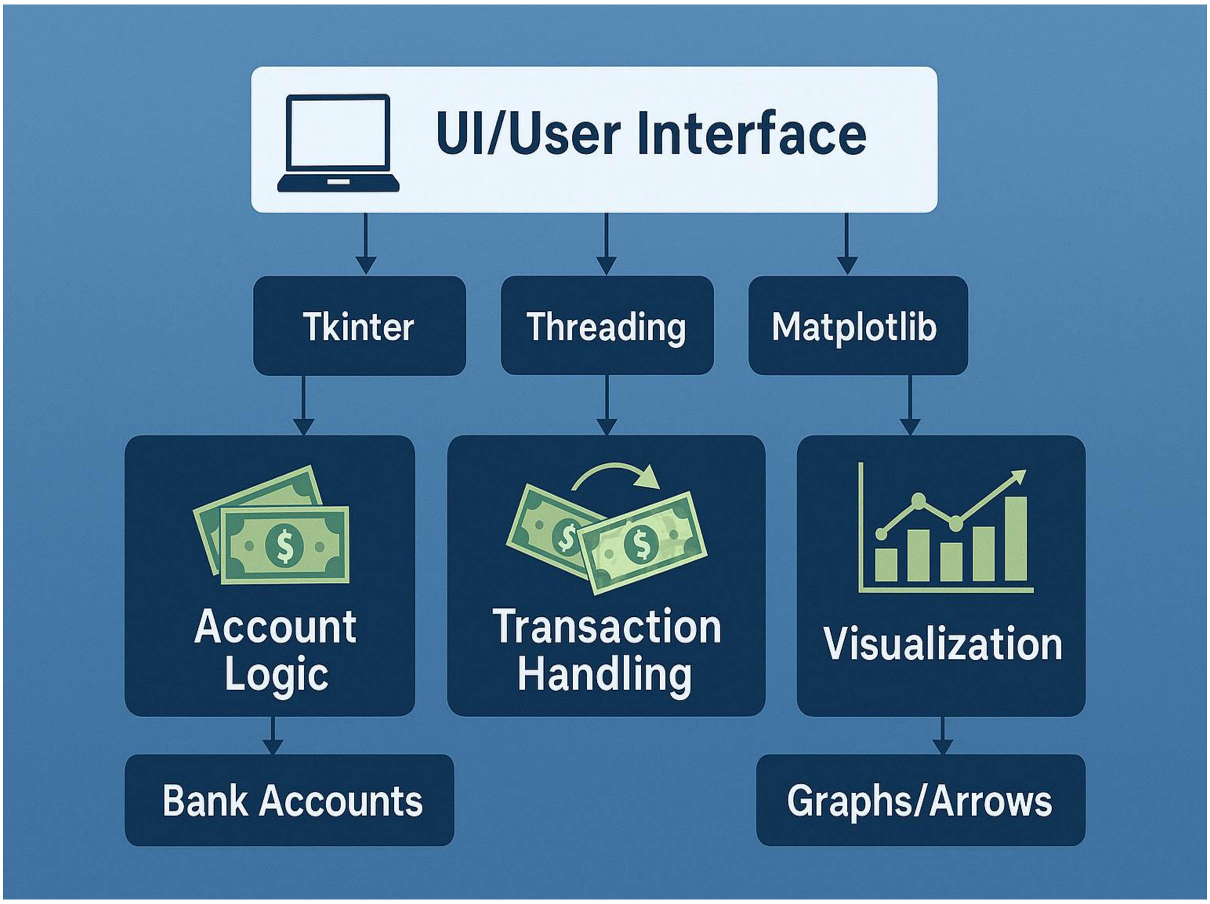


Figure 1: Bank ATM Transaction Deadlock Architecture

This diagram illustrates the architecture of a **Bank Transaction and Deadlock Simulation System**, structured around a modular design for clarity and performance.

UI Layer

- Built with **Tkinter**, this layer manages user interaction, triggering transactions and displaying visual feedback.

Core Components

- **Account Logic (via Tkinter)**

- Manages bank accounts, including balance updates and lock states.
- Implements core banking operations.

- **Transaction Handling (via Threading)**

- Executes concurrent transactions.
- Handles deadlock detection and prevention using priorities and locks.

- **Visualization (via Matplotlib)**

- Renders real-time graphs and arrows.
- Displays transaction flow and system efficiency visually.

CHAPTER 3

FLOWCHART

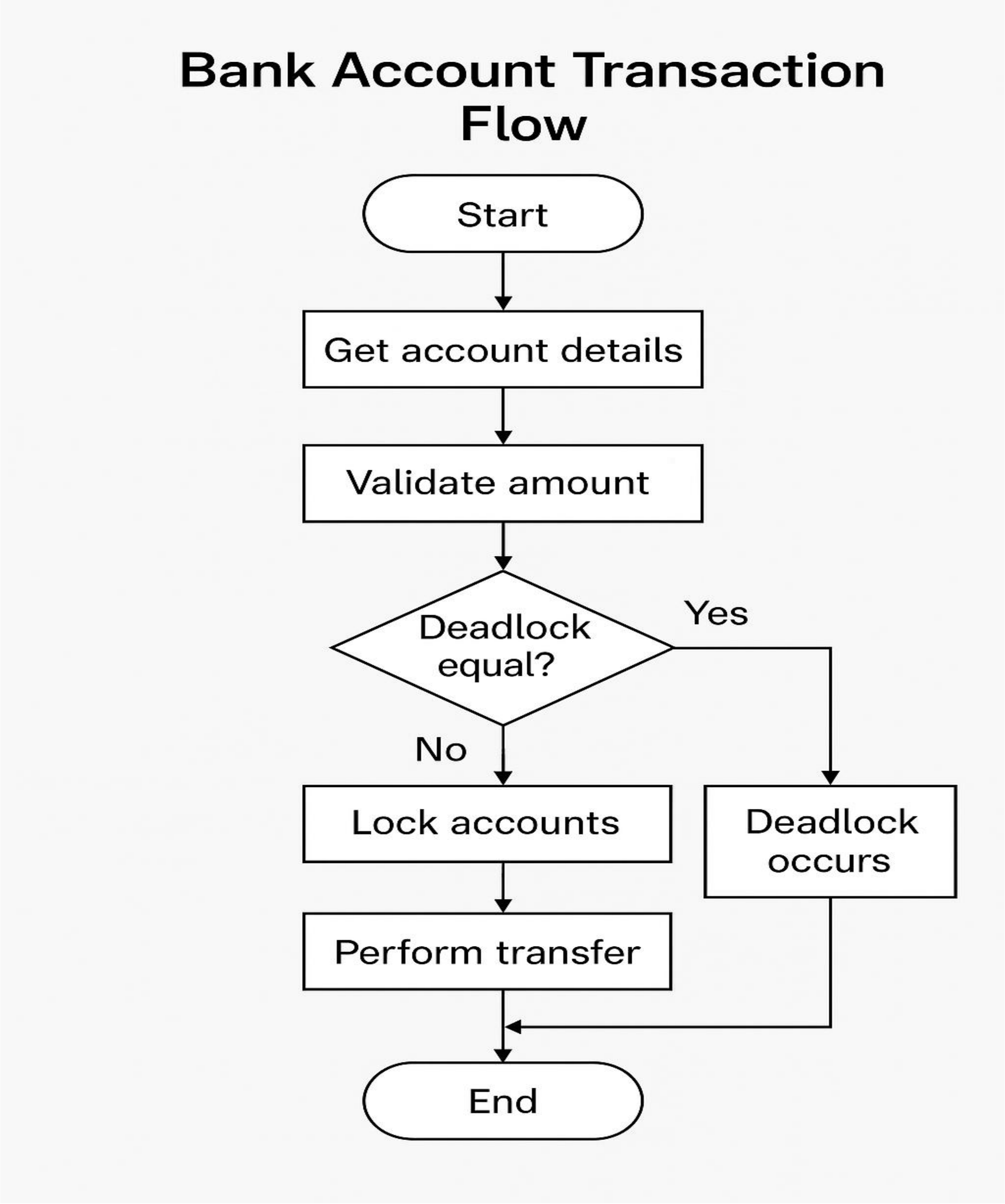


Figure 2: Bank ATM Transaction Deadlock Flowchart

- **GET ACCOUNT DETAILS**

FETCHES THE SOURCE AND DESTINATION ACCOUNT INFORMATION.

- **VALIDATE AMOUNT**

CHECKS IF THE TRANSACTION AMOUNT IS VALID (E.G., NOT NEGATIVE, SUFFICIENT BALANCE).

- **DEADLOCK EQUAL?**

CHECKS IF BOTH ACCOUNTS HAVE THE SAME PRIORITY (WHICH COULD LEAD TO DEADLOCK).

YES → DEADLOCK MAY OCCUR, AND THE PROCESS STOPS.

NO → PROCEED TO LOCK THE ACCOUNTS.

- **LOCK ACCOUNTS**

LOCKS THE INVOLVED ACCOUNTS TO ENSURE SAFE TRANSFER WITHOUT INTERFERENCE.

- **PERFORM TRANSFER**

EXECUTES THE MONEY TRANSFER BETWEEN ACCOUNTS.

- **END**

THE TRANSACTION PROCESS FINISHES.

CHAPTER-4

SCOPE OF THE PROJECT

The *Bank ATM Transaction Deadlock* project aims to simulate and analyse transaction deadlocks in ATM systems using a visual and interactive Python-based GUI built with Tkinter. The project demonstrates how deadlocks can occur during simultaneous account transactions due to resource contention and improper synchronization. It allows users to initiate multiple transactions between bank accounts, each with assigned priorities. If two accounts share the same priority, the system simulates a potential deadlock scenario, flags it, and prevents further execution to maintain system stability. In contrast, transactions between accounts with different priorities are handled safely using a priority-based locking mechanism.

The simulation includes dynamic transaction flow animations and status updates to help users better understand transaction processes. An efficiency graph is also generated, showcasing the performance differences between normal and deadlock-prone transactions. This graphical feedback helps in comparing response times and understanding the impact of deadlock prevention strategies.

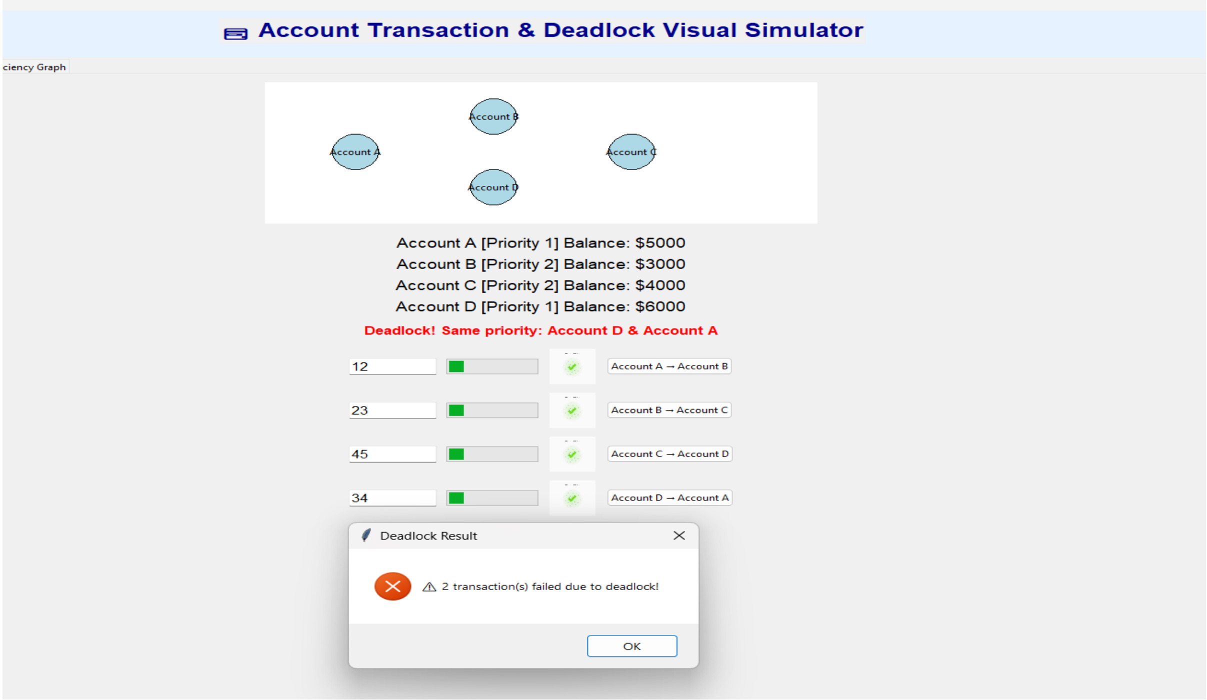
Designed as both an educational and simulation tool, this project enables students, developers, and professionals to grasp the principles of deadlock detection, prevention, and concurrency control. It serves as a foundational model for building more robust and fault-tolerant ATM systems, improving the reliability and safety of banking operations.

CHAPTER-5

CODE IMPLEMENTATION

```
def __init__(self, name, balance, priority):
    # Initializes a bank account with name, balance, and priority

def withdraw(self, amount):
    # Deducts the specified amount from the account balance

def deposit(self, amount):
    # Adds the specified amount to the account balance

def animate_transfer(label):
    # Animates the transfer icon sequence between accounts

def draw_arrow(canvas, nodes, from_acc, to_acc, amount):
    # Draws an arrow on the canvas representing a transaction and amount

def transfer_priority(from_acc, to_acc, amount, label, transactions_done, bar, anim_label,
                      canvas=None, nodes=None, eff_data=None, ax=None, canvas_fig=None):
    # Handles transaction with lock acquisition logic based on priority to avoid deadlocks

def check_final_status(transactions_done):
    # Checks if all transactions are complete and shows success or deadlock result

def update_balance_labels():
    # Updates all account balance labels in the UI

def build_transaction_ui(frame, account_pairs, transactions_done_list,
                         canvas=None, nodes=None, eff_data=None, ax=None, canvas_fig=None):
    # Creates the UI for a set of account transactions with input, progress bar, and animation
```

DEADLOCK TRANSACTION AND TIME EFFICIENCY GRAPH



Account A [Priority 1] Balance: $5000
Account B [Priority 2] Balance: $3000
Account C [Priority 2] Balance: $4000
Account D [Priority 1] Balance: $6000

Deadlock! Same priority: Account D & Account A

⚠ 2 transaction(s) failed due to deadlock!

Figure 3: Deadlock Account Transaction

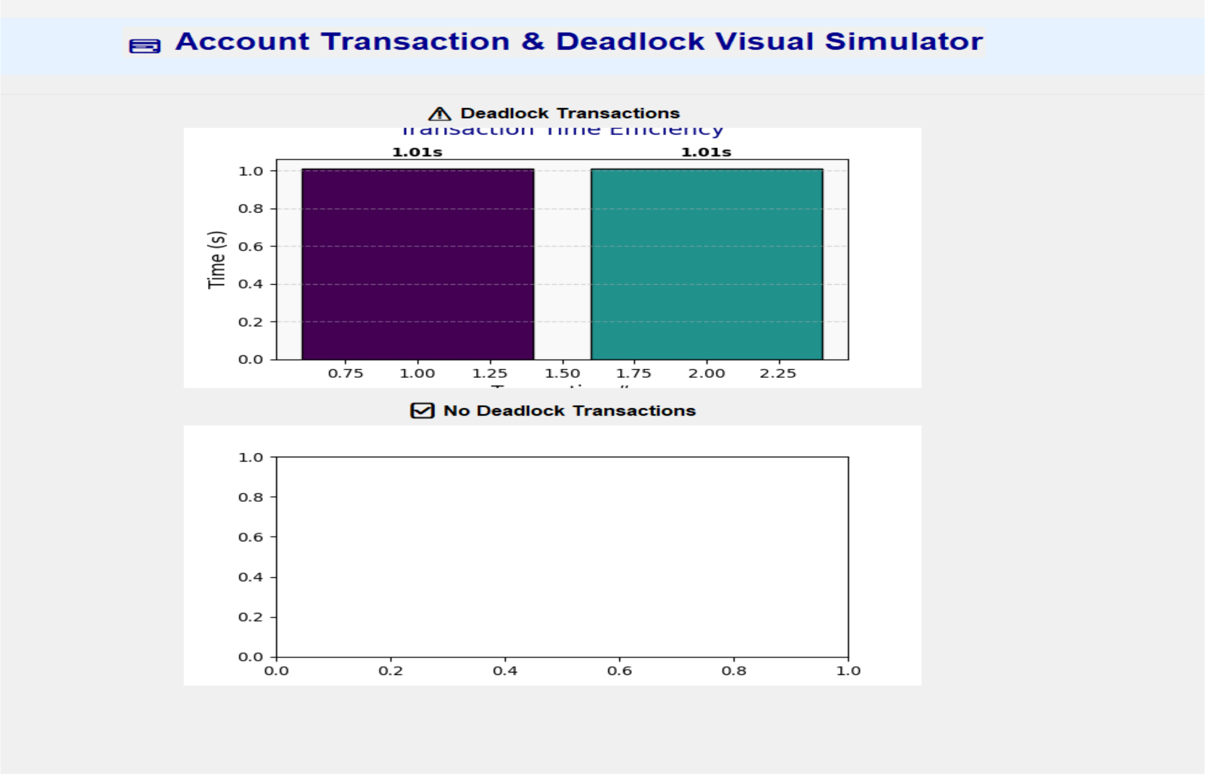
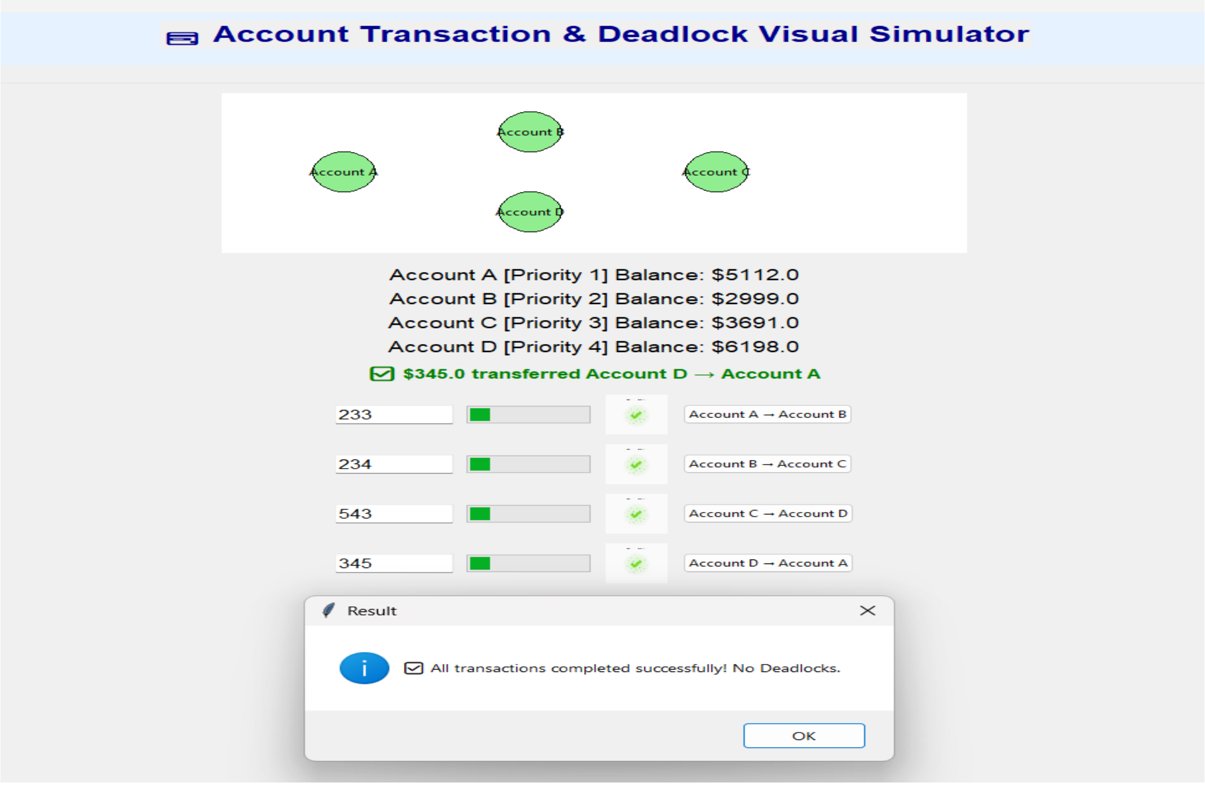


Figure 4: Deadlock Time Efficiency Graph

NO DEADLOCK TRANSACTION AND TIME EFFICIENCY GRAPH



Account Transaction & Deadlock Visual Simulator

Account A [Priority 1] Balance: $5112.0
Account B [Priority 2] Balance: $2999.0
Account C [Priority 3] Balance: $3691.0
Account D [Priority 4] Balance: $6198.0

☑ $345.0 transferred Account D → Account A

233 ✓ Account A → Account B
234 ✓ Account B → Account C
543 ✓ Account C → Account D
345 ✓ Account D → Account A

Result

☑ All transactions completed successfully! No Deadlocks.

Figure 5: No Deadlock Account Transaction

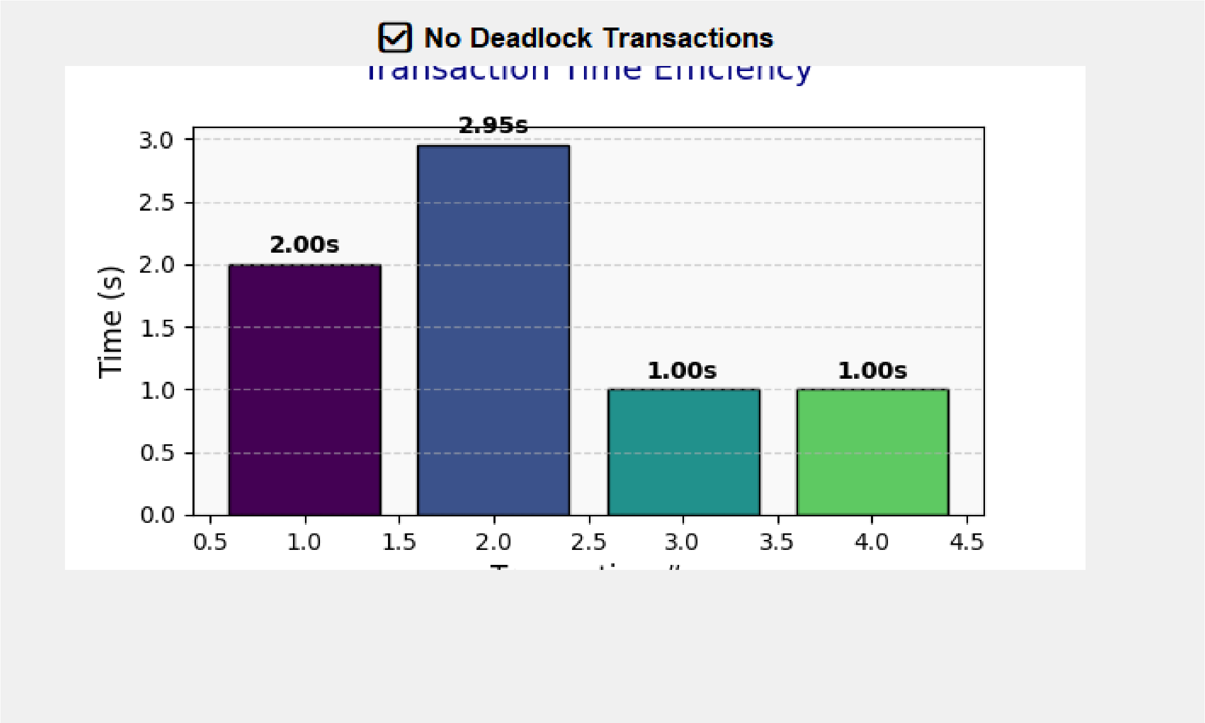


Figure 6: No Deadlock Time Efficiency Graph

CHAPTER 6

CONCLUSION

The **Bank ATM Transaction Deadlock** mini project successfully demonstrates how concurrent transactions can lead to potential deadlocks and how priority-based deadlock prevention strategies can effectively mitigate them. Through a user-friendly Tkinter GUI, the system visualizes multiple bank account transfers, lock acquisition sequences, and the occurrence or avoidance of deadlocks in real-time. The animated flows, graphical efficiency metrics, and clear error/status indicators help users intuitively grasp the complexity of thread synchronization and resource contention in banking systems.

6.1. Advantages of this project include:

- **Educational Value:** It simplifies multithreading concepts and deadlock handling for learners.
- **Visualization:** Clear, animated transaction paths enhance understanding.
- **Realism:** Simulates real-world bank scenarios like priority-based locking and concurrent transactions.
- **Scalability:** Can be extended with databases, authentication, or a web interface.

Overall, the project not only showcases deadlock detection and avoidance but also emphasizes the importance of designing secure, responsive transaction systems in financial applications.

CHAPTER 7

FUTURE WORKS

- **AI BASED PRIORITY OPERATION:**
  Integrate machine learning to dynamically assign priorities to accounts based on past transaction behaviours or success rates
- **DATABASE INTEGRATION**:
  Connect to a database like SQLite or PostgreSQL for persistent storage of account data, transactions, and logs.
- **WEB BASED VERSION**:
  Convert the application to a Flask or Django web app for accessibility through browsers and deployment on cloud platforms.
- **REAL TIME FRAUD DETECTION**:
  Add logic to detect unusual or suspicious transactions and notify the user or prevent the transfer
- **ADVANCE DEAD LOCK RECOVERY**:
  Implement additional deadlock recovery techniques like pre-emption or rollback besides priority-based prevention.
- **DETAILED LOGING SYSTEM:**
  Create a logging dashboard showing time-stamped transaction attempts, locks, and failures for analysis and debugging.
- **MULTI CURRENCY SUPPORT**:
  Enable transactions between accounts using different currencies with real-time conversion.
- **USER ROLE&AUTHENTICATION:**
  Add login systems with role-based permissions (e.g., admin for creating accounts, users for transactions
- **REAL TIME GRAPH UPDATE:**
  Make the efficiency graph update in real time with each transaction, enhancing the data visualization experience.
- **BLOCK CHAIN INTERGATION:**
  Use blockchain to ensure tamper-proof logging of transactions and priority changes for auditability.

CHAPTER 8

REFERENCES

- https://pages.cs.wisc.edu/~remzi/OSTEP/
- https://codex.cs.yale.edu/avi/os-book/OS10/slide-dir/
- https://iarjset.com/wp-content/uploads/2022/02/IARJSET.2022.9202.pdf

□