A

Mini Project Report

on

DEADLOCKS USING BANKER'S ALGORITH

submitted in partial fulfillment of requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

by

S. M. AFRIN (239Y1A05E1)
S. ROSHINI (239Y1A05E4)
S. SAMEERA (239Y1A05E6)
S. SAFIYA TASNEEM (239Y1A05E5)

under the Esteemed Supervision of

Ms. Z. SHOBHA RANI
Dept of CSE.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

K.S.R.M. COLLEGE OF ENGINEERING

(An Autonomous institution affiliated to JNTUA, Anantapuramu , Accredited by NAAC With A+ Grade)

Kadapa, Andhra Pradesh, India– 516 003

2024-2025

K.S.R.M. COLLEGE OF ENGINEERING

(An Autonomous institution affiliated to JNTUA, Anantapuramu , Accredited by NAAC With A+ Grade)

Kadapa, Andhra Pradesh, India– 516 003

VISION

To evolve as center of repute for providing quality academic programs amalgamated with creative learning and research excellence to produce graduates with leadership qualities, ethical and human values to serve the nation.

MISSION

M1:To provide high quality education with enriched curriculum blended with impactful teaching-learning practices.

M2:To promote research, entrepreneurship and innovation through industry collaborations.

M3:To produce highly competent professional leaders for contributing to Socio-economic development of region and the nation.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vision

To evolve as a recognized center of excellence in the area of Computer Science and Engineering and other related inter-disciplinary fields.

Mission

M1:: To produce competent and industry ready professionals through well balanced curriculum and innovative pedagogy.

M2::To provide conducive environment for research by establishing centre of excellence and industry collaborations.

M3:: To instill leadership qualities, ethical values among students through various co-curricular and extracurricular activities.

B. Tech. (COMPUTER SCIENCE AND ENGINEERING)

Program Educational Objectives

B.Tech-Computer Science and Engineering Program Objectives.

A graduate of the K.S.R.M.C.E, C.S.E should have a successful career in CSE or a related field, and within three to five years, should

PEO1 - : To excel in their career as competent software engineer in IT and allied organizations.

PEO2 - : To pursue higher education and to demonstrate research temper for providing solutions to engineering problems.

PEO3 - : To contribute for the societal development by exhibiting leadership, through professional, social and ethical values.

Program Outcomes

PO1 - Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2 - Problem Analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3 - Design/Development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4 - Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5 - Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6 - The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7 - Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8 - Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of engineering practice.

PO9 - Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10 - Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions .

PO11 - Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environements.

PO12 - Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes

PSOs are statements that describe what the graduates of a specific engineering program should be able to do:

PSO1 - Professional Skills: The ability to understand, analyze and develop computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics, and networking for efficient design of computer-based systems of varying complexity.

PSO2 - Problem-Solving Skills: The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success.

PSO3 - Successful Career and Entrepreneurship: The ability to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur, and a zest for higher studies.

Course Outcomes

CO1. Understand core concepts and research findings relative to human development, socialization, group dynamics and life course processes.

CO2. Identify and transfer existing ideas into new contexts and applications.

CO3. Apply and transfer academic knowledge into the real-world.

CO4: Design a component or a product applying all the relevant standards and with realistic Constraints.

CO-PO Mapping

Course Outcome	Program Outcomes												Program Specific Outcomes		
	PO1	PO2	PO3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	2	3	2										3	2	2
CO2	2	3	3					1					3	3	2
CO3	3	3	3	1							1		2	2	3
CO4	3	3	3	2				1			1		2	2	3
CO5	2	2	3										2	2	2

K.S.R.M. COLLEGE OF ENGINEERING

(An Autonomous institution affiliated to JNTUA, Anantapuramu , Accredited by NAAC With A+ Grade)

Kadapa, Andhra Pradesh, India– 516 003

CERTIFICATE

This is to certify that the Operating system mini Project Report entitled

DEADLOCKS USING BANKER'S ALGORITHM

is the bona fide work done & submitted by

S. M. AFRIN	**(239Y1A05E1)**
S. ROSHINI	**(239Y1A05E4)**
S. SAMEERA	**(239Y1A05E6)**
S. SAFIYA TASNEEM	**(239Y1A05E5)**

in the Department of Computer Science and Engineering, **K.S.R.M.C.E, Kadapa** and is submitted to **Jawaharlal Nehru Technological University Anantapur, Ananthapuramu** in partial fulfilment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering during 2023-2027.

Supervisor	**Head of the Department**
Ms.Z.Shobha rani.	**Dr.V.Venkata ramana ,M.Tech,(Ph.D).**
Assistant professor.	**Associate Professor & HOD**
Department of CSE.	**Department of CSE & Allied Branches.**

DECLARATION

We hereby declare that this Mini Project report titled “DEADLOCKS USING BANKERS ALGORUTHM” is a genuine Operating system mini project work carried out by us, in B. Tech (**Computer Science and Engineering**) degree course of Jawaharlal Nehru Technological University Anantapur and has not been submitted to any other course or University for the award of any degree by us.

Signature of the Student

S. M. AFRIN	**(239Y1A05E1)**
S. ROSHINI	**(239Y1A05E4)**
S. SAMEERA	**(239Y1A05E6)**
S. SAFIYA TASNEEM	**(239Y1A05E5)**

ACKNOWLEDGEMENTS

An endeavor over a long period can be successful only with the advice and supports of many well-wishers. We take this opportunity to express our gratitude and appreciation to all of them.

We are extremely thankful to our beloved Managing Director Dr. K Chandra Obul Reddy Garu who took keen interest and encouraged us in every effort throughout this course. We are deeply indebted to the supervisor, Ms. Z. Shobha Rani, Department of Computer Science and Engineering for valuable guidance, constructive criticism and keen interest evinced throughout the course of our Operating system mini project work. We are really fortunate to associate ourselves with such an advising and helping guide in every possible way, at all stages, for the successful completion of this work.

We express our deep sense of gratitude to Dr. V. Venkata Ramana, M.Tech., (Ph.D)., Assistant Professor and Head of Department of Computer Science and Engineering and Allied Branches for his valuable guidance and constant encouragement given to us during this Operating system mini project and the course.

We take this opportunity to express our deep gratitude and appreciation to all those who encouraged us for successfully completion of this Operating system mini Project work. We wish to express our sincere to gratitude to Dr. T NAGESWARA PRASAD, M.Tech., Ph.D. Vice Principal of K.S.R.M.C.E, Kadapa and Dr. V.S.S. MURTHY, M.Tech., Ph.D. Principal of K.S.R.M.C.E, Kadapa and for their consistent help and encouragement to complete the Operating system mini project.

We are pleased to express our heart full thanks to our faculty in Department of CSE of KSRMCE for their moral support and good wishes. Finally, we have a notation to express our sincere thanks to friends and all those who guided, inspired and helped us in the Operating system mini project work

S. M.AFRIN (239Y1A05E1)

S. ROSHINI (239Y1A05E4)

S. SAMEERA (239Y1A05E6)

S. SAFIYA TASNEEM (239Y1A05E5)

CONTENTS

ABSTRACT

Concurrency and parallelism have become essential in modern computing, enabling efficient utilization of resources and improved performance in multi-threaded and multi-process applications. However, concurrent execution introduces challenges such as race conditions, resource contention, and deadlocks. Deadlock is a critical issue in concurrent programming that occurs when multiple threads or processes are indefinitely waiting for resources held by each other, leading to a system state where execution cannot proceed.

In Python, deadlocks commonly arise in multi-threaded programs that utilize synchronization primitives such as threading.Lock, threading.RLock, threading.Semaphore, and multiprocessing.Lock. A deadlock scenario typically occurs when two or more threads attempt to acquire multiple locks in an inconsistent order, leading to circular waiting. This results in all involved threads being unable to proceed, causing the program to freeze or stall.

This paper explores the concept of deadlock in Python, including its causes, characteristics, and common scenarios where it occurs. We provide an in-depth discussion on how deadlocks can be simulated in Python using a multi-threaded environment and illustrate a classic example where improper lock acquisition results in a circular wait condition. Furthermore, we discuss different strategies to prevent and mitigate deadlocks, such as:

1. Resource Ordering – Ensuring that all threads acquire locks in a predetermined order to prevent circular waiting.

2. Timeout Mechanisms – Using timeouts when acquiring locks to detect and handle potential deadlocks.

3. Deadlock Detection – Implementing algorithms to monitor and resolve deadlocks dynamically.

4. Avoiding Nested Locks – Minimizing the use of multiple locks in a single thread to reduce the risk of deadlock

CHAPTER 1

INTRODUCTION

Introduction

In modern computing systems, resource allocation is a critical aspect of process management. However, improper allocation can lead to deadlocks, where a set of processes are indefinitely waiting for resources held by each other. Deadlocks can severely impact system performance and stability, making their prevention and avoidance essential in operating system design.

One of the most effective algorithms for deadlock avoidance is the **Banker’s Algorithm**, proposed by Edsger Dijkstra. The algorithm simulates resource allocation by considering the system’s state before
granting requests, ensuring that it never enters an unsafe state that could lead to a deadlock. By using a predefined set of maximum resource demands and available resources, the Banker's Algorithm makes intelligent decisions about whether a process should receive resources or wait until they can be safely allocated.

This project aims to implement and analyze the **Banker’s Algorithm** for deadlock avoidance. We will explore its working mechanism, applications, advantages, and limitations. Through simulations, we will demonstrate how the algorithm helps prevent deadlocks by ensuring the system remains in a safe state.

CHAPTER2

SYSTEM ANALYSIS

MODULES

Description of Deadlock Algorithm

A **deadlock** occurs in a computing system when a group of processes become stuck in a state where each process is waiting for a resource that another process holds, preventing further execution. Deadlocks typically arise in multi-threaded and multi-processing environments when resource allocation is poorly managed.

To handle deadlocks, operating systems use various **deadlock handling algorithms**, which can be categorized into four main approaches:

1. **Deadlock Prevention** – Ensures that at least one of the necessary conditions for deadlock (mutual exclusion, hold and wait, no preemption, circular wait) is never satisfied.
2. **Deadlock Avoidance** – Uses algorithms like the **Banker's Algorithm** to ensure the system never enters an unsafe state that could lead to deadlock.
3. **Deadlock Detection and Recovery** – Detects deadlocks after they occur and applies recovery techniques such as resource preemption or process termination.
4. **Ignoring Deadlocks** – Some systems, like UNIX and Windows, use a "wait-and-see" approach, assuming deadlocks are rare and relying on system restarts if necessary.

Among these approaches, **Banker's Algorithm** is widely used for deadlock avoidance. It works by evaluating resource allocation requests before granting them, ensuring that the system remains in a safe state where deadlocks cannot occur.

This project will focus on understanding deadlocks and implementing the **Banker's Algorithm** to demonstrate how deadlock avoidance can be achieved in real-world computing environments.

CHAPTER 3

PROJECT IMPLEMENTATION DETAILS

The Banker's Algorithm is a resource allocation and deadlock avoidance algorithm used in operating systems. It ensures that a system never enters an unsafe state by allocating resources carefully.

1. System Requirements

- **Programming Language:** C, C++, Python, or Java
- **Input:**
  - Number of processes
  - Number of resource types
  - Available resources for each type
  - Maximum resource requirement for each process
  - Allocated resources for each process
- **Output:**
  - Whether the system is in a safe state
  - The safe sequence if available
  - If a request can be granted safely

---

2. Functional Components

2.1 Data Structures

- `Available[]`: Holds the number of available instances for each resource.
- `Max[][]`: Defines the maximum demand of each process.
- `Allocation[][]`: Stores the number of resources allocated to each process.
- `Need[][]`: Represents remaining resource requirements for each process.

2.2 Algorithm Implementation

Step 1: Input and Initialization

- Take input for `n` processes and `m` resource types.
- Populate `Available[]`, `Max[][]`, and `Allocation[][]`.
- Compute `Need[][]` using: Need[i][j]=Max[i][j]−Allocation[i][j]Need[i][j] = Max[i][j] - Allocation[i][j]Need[i][j]=Max[i][j]−Allocation[i][j]

Step 2: Safety Algorithm

- Initialize `Work[] = Available[]` and `Finish[] = false` for all processes.

- Find a process `P[i]` such that: Need[i]≤WorkNeed[i] \leq WorkNeed[i]≤Work
- Allocate resources and update `Work[]` and `Finish[]`.
- If all processes can complete, the system is in a safe state.

Step 3: Resource Request Algorithm

- Check if the requested resources by `P[i]` do not exceed `Need[i]` and `Available[]`.
- Temporarily allocate resources.
- Run the safety algorithm to check if the system remains safe.
- If safe, grant the request; otherwise, revert changes.

3. Code Implementation

4. Testing Strategy

- **Test Case 1:** System starts in a safe state.
- **Test Case 2:** A process requests resources leading to an unsafe state.
- **Test Case 3:** Multiple processes complete successfully.

---

5. Enhancements

- GUI-based visualization
- Simulate dynamic requests and allocation changes
- Multi-threaded implementation for real-time simulation

CHAPTER 4

SOURCE CODE

```
import tkinter as tk
from tkinter import messagebox, simpledialog
from tkinter import ttk

# Sample login credentials
USER_CREDENTIALS = {"afrin": "sameera786"}

# Function for Banker's Algorithm
def bankers_algorithm():
    try:
    num_processes = int(simpledialog.askstring("Input", "Enter number
of processes:"))
    num_resources = int(simpledialog.askstring("Input", "Enter number
of resources:"))

        allocation = []
        max_need = []
        available = list(map(int, simpledialog.askstring("Input",
"Enter available resources (space-separated):").split()))

        for i in range(num_processes):
            allocation.append(list(map(int,
simpledialog.askstring("Input", f"Enter allocation for P{i} (space-
separated):").split())))
            max_need.append(list(map(int,
simpledialog.askstring("Input", f"Enter max need for P{i} (space-
separated):").split())))

        # Calculate Need Matrix
        need = [[max_need[i][j] - allocation[i][j] for j in
range(num_resources)] for i in range(num_processes)]
        finish = [False] * num_processes
        safe_sequence = []

        while len(safe_sequence) < num_processes:
            found = False
            for i in range(num_processes):
                if not finish[i] and all(need[i][j] <= available[j] for
j in range(num_resources)):
                    for j in range(num_resources):
                        available[j] += allocation[i][j]
                    safe_sequence.append(i)
                    finish[i] = True
```

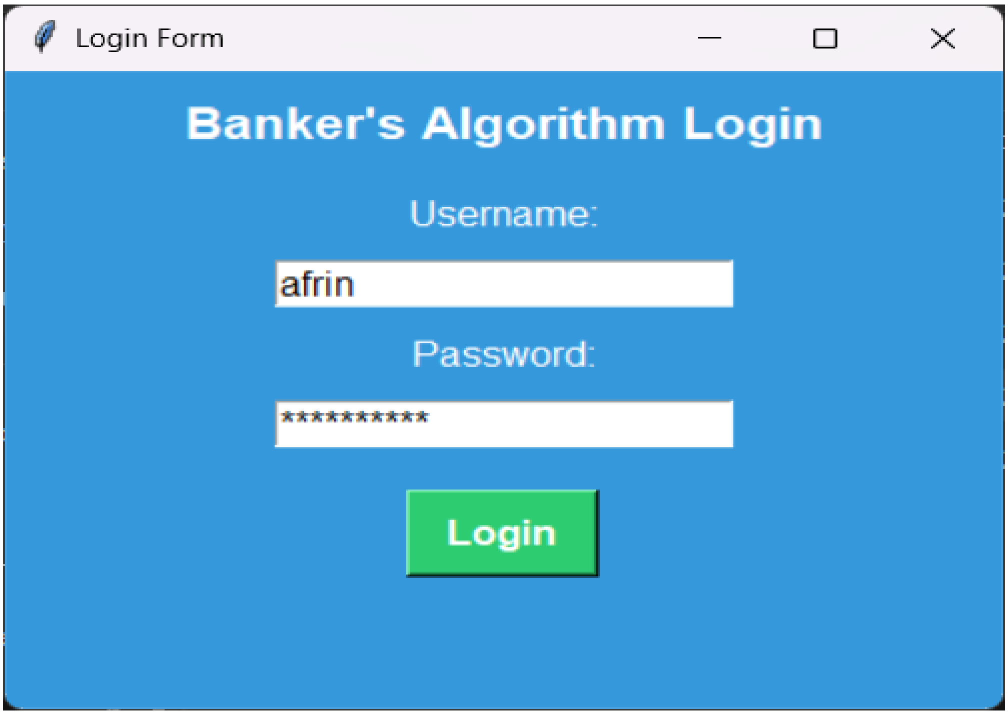
```
                    found = True
                    break
            if not found:
                messagebox.showerror("Deadlock Detected", "System is in
a Deadlock! No safe sequence found.")
                return

        messagebox.showinfo("Safe State", f"System is in a Safe
State!\nSafe Sequence: {' -> '.join(['P'+str(i) for i in
safe_sequence])}")

    except Exception as e:
        messagebox.showerror("Error", f"Invalid input: {e}")

# Login Function
def login():
    username = entry_username.get()
    password = entry_password.get()

    if username in USER_CREDENTIALS and USER_CREDENTIALS[username] ==
password:
        messagebox.showinfo("Login Successful", "Welcome to the
Banker's Algorithm Simulation!")
        root.destroy()
        bankers_algorithm()
    else:
        messagebox.showerror("Login Failed", "Invalid username or
password!")

# GUI for Login Form
root = tk.Tk()
root.title("Login Form")
root.geometry("400x300")
root.configure(bg="#3498db")

# Title Label
title_label = tk.Label(root, text="Banker's Algorithm Login",
font=("Arial", 16, "bold"), fg="white", bg="#3498db")
title_label.pack(pady=10)

# Username Label & Entry
tk.Label(root, text="Username:", font=("Arial", 12), fg="white",
bg="#3498db").pack(pady=5)
entry_username = tk.Entry(root, font=("Arial", 12))
entry_username.pack(pady=5)

# Password Label & Entry
```

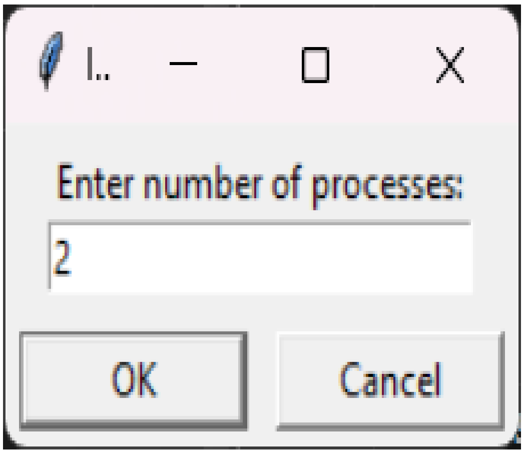
```
tk.Label(root, text="Password:", font=("Arial", 12), fg="white",
bg="#3498db").pack(pady=5)
entry_password = tk.Entry(root, show="*", font=("Arial", 12))
entry_password.pack(pady=5)

# Login Button with styling
login_button = tk.Button(root, text="Login", font=("Arial", 12,
"bold"), bg="#2ecc71", fg="white", padx=10, pady=5, command=login)
login_button.pack(pady=15)

root.mainloop()
```

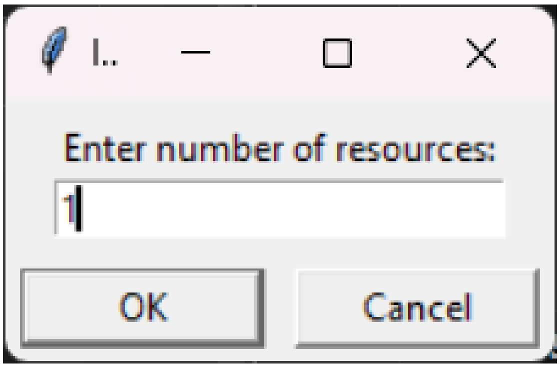
CHAPTER 5

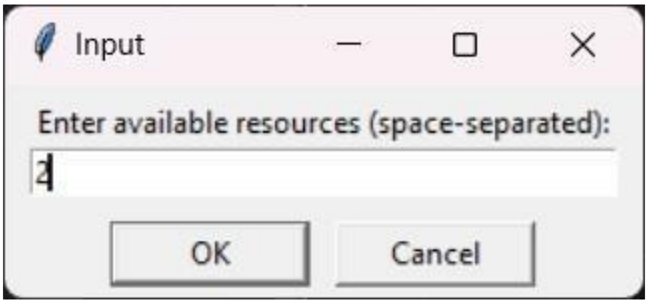
SCREENS

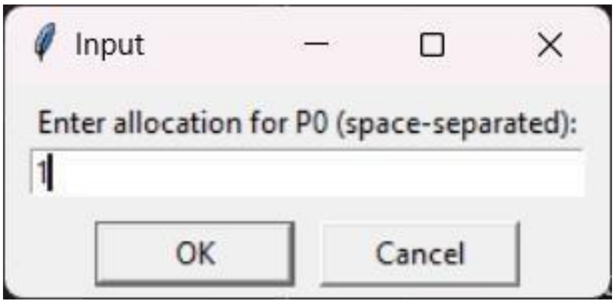
PLACE UR PROJECT SCREENS

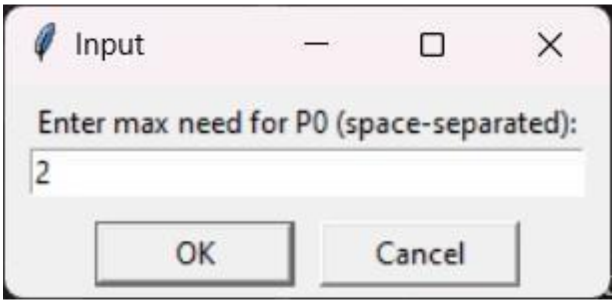


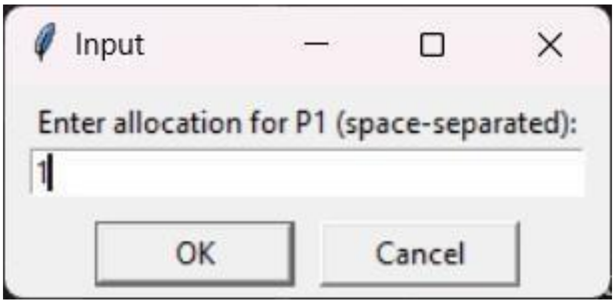


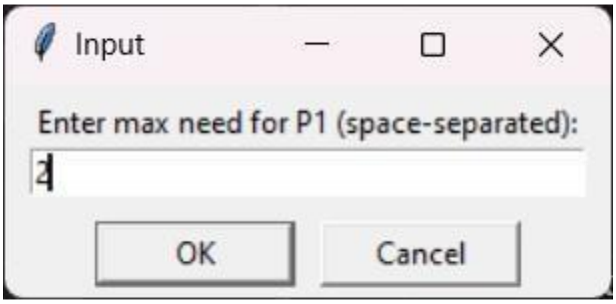

I..
Enter number of resources:
1
OK
Cancel


Input
Enter available resources (space-separated):
2
OK
Cancel


Input
Enter allocation for P0 (space-separated):
1
OK
Cancel


Input
Enter max need for P0 (space-separated):
2
OK
Cancel


Input
Enter allocation for P1 (space-separated):
1
OK
Cancel


Input
Enter max need for P1 (space-separated):
2
OK
Cancel


Safe State
System is in a Safe State!
Safe Sequence: P0 → P1
OK

CHAPTER-6

CONCLUSION

The **Banker’s Algorithm** is a vital deadlock avoidance technique used in operating systems to ensure safe resource allocation while preventing deadlocks. By checking the system's current allocation state and determining whether granting a process's resource request would lead to an unsafe condition, it allows processes to proceed only when the system remains in a safe state. The algorithm relies on three key matrices: **Available**, **Allocation**, and **Need**, which together help determine if a **safe sequence** exists. If a process’s request can be fulfilled without leading to a deadlock, the resources are allocated; otherwise, the request is denied to maintain system stability.