DEADLOCK DETECTION AND AVOIDANCE SYSTEM

A PROJECT REPORT

Submitted to

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

In partial fulfilment of the award of the degree of

BACHELOR OF ENGINEERING IN COMPUTER SCIENCE ENGINEERING

By

S. GEETHA VINAYA SRI (192211458)

M. ARCHANA (192211725)

D. SREE GAYATHRI (192211455)

Supervisor

DR. TERRANCE FREDERICK FERNANDEZ

CSA0494-OPERATING SYSTEMS FOR SECURED INTERPRETER SYSTEM

SAVEETHA SCHOOL OF ENGINEERING

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES, CHENNAI – 602 105.

MARCH 2024

INTRODUCTION

Deadlock occurs when two or more processes are waiting indefinitely for resources that are held by each other, resulting in a situation where none of the processes can proceed. The Banker's algorithm is a well-known method used for deadlock avoidance in systems with multiple resources and processes. In the Banker's algorithm, each process must declare in advance the maximum number of resources of each type that it may need during its execution. The system maintains information about the currently available resources and the maximum resources that can be allocated to each process. Using this information, the algorithm checks whether granting a resource request will lead to a safe state or potentially result in deadlock. By employing the Banker's algorithm, the system can dynamically allocate resources to processes while ensuring that the system's state remains safe and deadlock-free. The algorithm involves a series of checks and calculations to determine if a resource allocation request should be granted or denied based on the potential impact on system stability. This proactive approach to resource management helps improve system efficiency and reliability by preventing the occurrence of deadlock scenarios.

PROBLEM STATEMENT

The problem entails developing a system to manage resource allocation in a multi-process environment, aiming to prevent deadlock using the Banker's algorithm. Deadlock occurs when processes are indefinitely blocked waiting for resources held by other processes, leading to a system-wide standstill. The task involves implementing the Banker's algorithm to ensure that resource allocation decisions are made in a way that guarantees system safety and avoids deadlock. the system will model processes and resource types, maintaining information about each process's maximum resource requirements and currently allocated resources. The Banker's algorithm will be employed to evaluate resource requests, determining whether granting a request will maintain system safety or potentially lead to deadlock. Resource allocation decisions will be made based on this evaluation, ensuring that processes can acquire resources without jeopardizing system stability.

The implementation will include mechanisms for handling resource requests, checking for safe states using the Banker's algorithm, and detecting deadlock situations within the system. A simulation environment will be developed to test the system under varying scenarios, such as different numbers of processes, types of resources, and allocation patterns. The ultimate goal is to demonstrate the effectiveness of the Banker's algorithm in preventing deadlock while optimizing resource utilization in a dynamic and realistic setting. The project will deliver a C code implementation of the system along with documentation detailing the design, implementation, and testing procedures. These endeavours will provide valuable insights into deadlock prevention strategies and resource management techniques in operating systems.

PROPOSED DESIGN WORK

1.KEY COMPONENTS

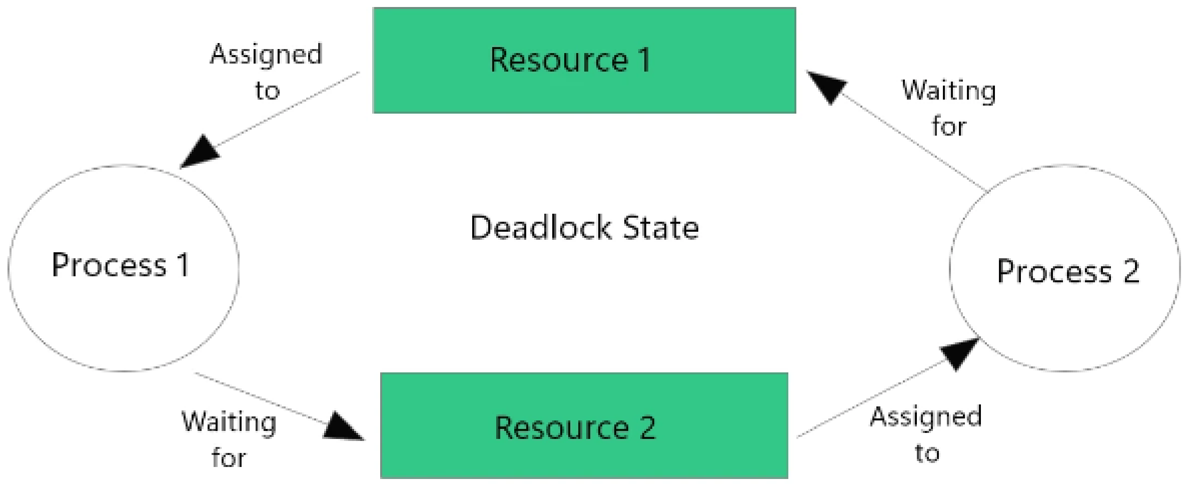
- Process Representation
- Resource Representation
- Data Structures
- Banker's Algorithm Implementation
- Resource Request Handling
- Safety Algorithm
- Deadlock Detection
- Simulation Environment
- Error Handling and Recovery
- Documentation and Testing

2.FUNCTIONALITY

To implement “Deadlock Detection and Avoidance System” using the Banker's algorithm, the system functionality can be summarized into several essential components and steps. Firstly, the system needs to represent processes and resources accurately. Each process should be defined with unique identifiers, maximum resource requirements (in the form of vectors), and currently allocated resources. Similarly, resources should have identifiers, total available instances, and currently allocated instances. The initialization phase sets up the system by defining processes and resources with their respective attributes. Each process declares its maximum resource needs upfront, which the Banker's algorithm will use to make resource allocation decisions during runtime. Resource request handling is a critical function that validates incoming requests against available resources and the process's maximum needs. The Banker's algorithm is then employed to determine if granting the request will maintain system safety, ensuring that allocations do not lead to a potential deadlock.

The core of the implementation revolves around the Banker's algorithm itself. This involves calculating remaining resource needs for each process and employing a safety algorithm to check for safe states. Resources are allocated to processes only when it's safe to do so based on the current system state. To further ensure system stability, a safety algorithm is integrated to assess the safety of resource allocation states. This typically involves simulating resource allocation to predict if processes can complete their tasks without encountering a deadlock. The system should also implement deadlock detection mechanisms to monitor resource requests and identify potential circular wait conditions. Deadlock detection algorithms, such as analysing resource allocation graphs, can be employed to detect cycles indicating a potential deadlock scenario. A simulation environment is crucial for testing the deadlock avoidance system under various scenarios. By varying the number of processes, resource types, and allocation patterns, the effectiveness of the Banker's algorithm in preventing deadlock and ensuring system stability can be evaluated. Lastly, robust error handling mechanisms are implemented to manage unexpected events like insufficient resources or invalid requests. Recovery strategies are defined to restore system stability after encountering deadlock situations. Comprehensive documentation is provided to explain the system's design, implementation details, and usage instructions. Thorough testing results and analysis validate the correctness and efficiency of the deadlock detection and avoidance system. By implementing these functionalities systematically, the system can effectively manage resource allocation to prevent deadlock using the Banker's algorithm, ensuring reliable operation and optimal resource utilization in a multi-process environment.

3.ARCHITECTURAL DESIGN

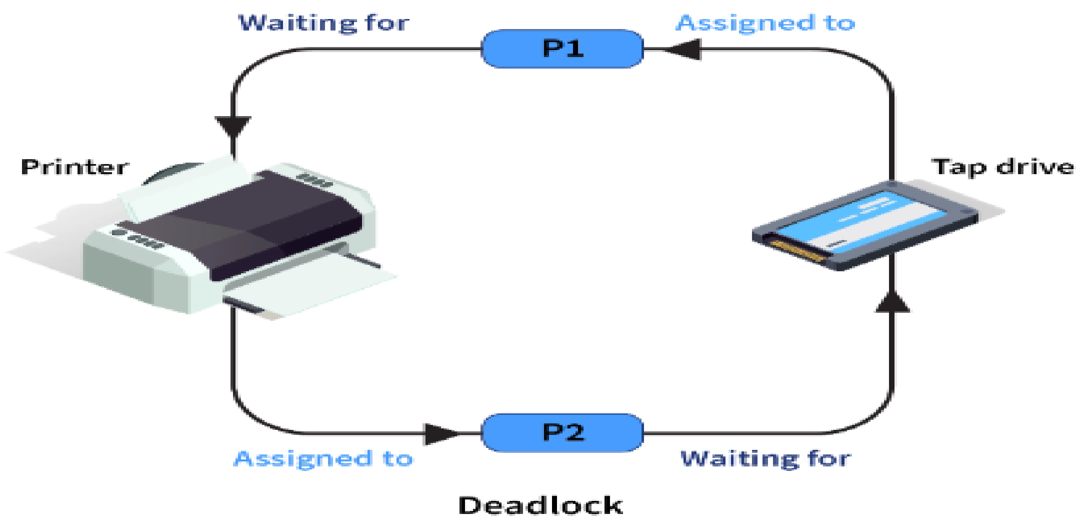


UI DESIGN

LAYOUT DESIGN

- Process Initialization
  - Initialize processes with attributes including process IDs, maximum resource needs (represented as vectors), and currently allocated resources (also represented as vectors).
- Resource Initialization
  - Define and initialize resource types with attributes such as resource IDs, total available instances, and currently allocated instances.
- Request Resource Handling
  - Implement a function to handle resource requests from processes:
  - Validate the request against available resources and the process's maximum resource needs.
- Banker's Algorithm Implementation:
  - Develop the core logic of the Banker's algorithm:
  - Calculate the remaining resource needs for each process (remaining_need = max_need - allocated_resources).

- Safety Algorithm
    - Implement the safety algorithm to determine if a resource allocation request can be granted without leading to an unsafe state (i.e., potential deadlock).
    - Use techniques like simulating resource allocation or constructing a resource allocation graph to evaluate system safety.
- Deadlock Detection
    - Develop mechanisms to detect potential deadlock situations:
    - Monitor resource allocation requests and identify circular wait conditions.
    - Use deadlock detection algorithms such as detecting cycles in resource allocation graphs.
- Resource Allocation and Deallocation
    - Manage resource allocation and deallocation based on process requests and releases:
    - Allocate resources to processes if requests are granted safely.
    - Release resources from processes when they are no longer needed.
- Error Handling and Recovery
    - Implement error handling mechanisms to manage unexpected events:
    - Handle cases of insufficient resources to fulfil requests.
    - Respond to invalid resource requests or process states appropriately.



CODE IMPLEMENTATION

#include <stdio.h> //This line includes the standard input-output library to enable the use of input and output functions. int main () //This is the main function where the execution of the program starts. {

// P0, P1, P2, P3, P4 are the Process names here

```
int n, m, i, j, k; //Declares integer variables n, m, i, j, and k for various counters
and dimensions used in the algorithm. n = 5; // Number of processes //Sets the number of
processes to 5. m = 3; // Number of resources //Sets the number of resources to 3. int alloc[5][3]
= { { 0, 1, 0 }, // P0 // Allocation Matrix //Initializes the allocation matrix, which represents the
number of resources allocated to each process. The matrix is defined
for 5 processes and 3 resources. { 2, 0, 0 }, // P1
{3, 0, 2}, // P2
{2, 1, 1}, // P3
{0, 0, 2} }; // P4
int max [5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix //Initializes the maximum matrix, which
represents the maximum demand of resources for each process. Similar to the allocation
matrix, it's defined for 5 processes and 3 resources. { 3, 2, 2 }, // P1
{9, 0, 2}, // P2
{2, 2, 2}, // P3
{4, 3, 3} }; // P4
int avail [3] = { 3, 3, 2 }; // Available Resources //Initializes the available resources. These
are the resources that are currently available in the system and can be allocated to processes.
int f[n], ans[n], ind = 0; //Declares arrays f, ans, and a variable ind. f is
used to track whether a process has been marked as finished (1) or not (0). ans will store
the safe sequence of processes, and ind is an index for ans. for (k = 0; k < n; k++) {
f[k] = 0;
}
int need[n][m]; //Declares a 2D array need to represent the resource need of
each process. need[i][j] is calculated as max[i][j] - alloc[i][j]. for (i = 0; i < n; i++) {
for (j = 0; j < m; j++)
need[i][j] = max[i][j] - alloc[i][j];
}
int y = 0; //Declares an integer variableyand initializes it to 0. for (k = 0; k < 5; k++) {
for (i = 0; i < n; i++) {
if (f[i] == 0) {
int flag = 0;
for (j = 0; j < m; j++) {
if (need[i][j] > avail[j]) {
flag = 1;
break;
}
}
if (flag == 0) {
ans[ind++] = i;
for (y = 0; y < m; y++)
avail[y] += alloc[i][y];
f[i] = 1;
}
}
}
}
int flag = 1;
for (int i=0;i<n;i++)
{
if(f[i]==0)
{
flag=0;
printf("The following system is not safe");
break;
}
}
if(flag==1)
{
printf("Following is the SAFE Sequence\n");
for (i = 0; i < n - 1; i++)
printf(" P%d ->", ans[i]);
```

printf(" P%d", ans[n - 1]);

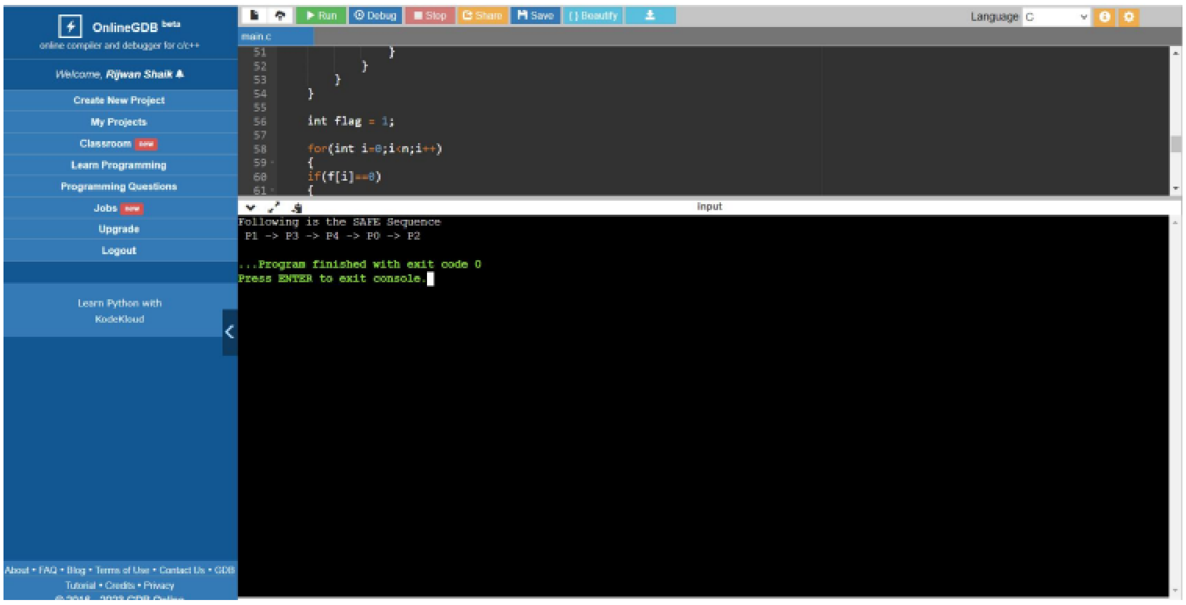
}

return (0); //Indicates the successful execution of the program.

}

The code calculates the safe sequence of processes and checks for the

OUTPUT



```c
                }
            }
        }
    }

    int flag = 1;

    for(int i=0;i<n;i++)
    {
    if(f[i]==0)
    {
```

```
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2

...Program finished with exit code 0
Press ENTER to exit console.
```

ANALYSIS

1.System Behaviour under Different Scenarios

- Conduct simulation and testing to analyze how the system behaves under various scenarios, such as different numbers of processes, resource types, and allocation patterns.
- Evaluate system performance in terms of resource utilization, throughput, and response time.
- Assess system stability and the ability to prevent deadlock situations effectively.

2.Banker's Algorithm Effectiveness

- Analyse the effectiveness of the Banker's algorithm in preventing deadlock.

- Evaluate how well the algorithm manages resource allocation to ensure system safety and avoid potential deadlock scenarios.
- Investigate the scalability of the algorithm with increasing numbers of processes and resource types.

3.Resource Allocation Efficiency

- Measure the efficiency of resource allocation in terms of minimizing resource wastage and maximizing resource utilization.
- Assess the impact of resource allocation strategies on overall system performance and responsiveness.

4.Safety Algorithm Performance

- Evaluate the performance of the safety algorithm used within the Banker's algorithm.
- Analyse the algorithm's ability to determine safe resource allocation states and its computational complexity.

5.Deadlock Detection and Recovery

- Investigate the effectiveness of deadlock detection mechanisms in identifying and resolving potential deadlock situations.
- Assess the system's ability to recover from deadlock scenarios and restore normal operation.

6.Error Handling and System Resilience

- Examine the system's error handling mechanisms and their effectiveness in managing unexpected events, such as resource exhaustion or invalid resource requests.
- Evaluate the system's resilience to failures and its ability to recover from error states.

7.Documentation and Reporting

- Review the comprehensiveness and clarity of system documentation.
- Ensure that documentation provides detailed insights into system design, implementation details, and usage instructions.
- Evaluate the quality of testing reports and analysis presented in the documentation.

8.Practical Considerations and Real-World Application

- Consider practical implications of implementing deadlock detection and avoidance in real-world systems.
- Address challenges related to system scalability, reliability, and maintainability.
- Explore potential extensions or enhancements to the system for more complex environments or requirements.

CONCLUSION

The implementation of deadlock detection and avoidance using the Banker's algorithm represents a critical aspect of resource management in multi-process environments. Through the analysis and utilization of key components and functionalities, we can achieve effective system behaviour and prevent potential deadlock scenarios. By representing processes and resources accurately and initializing the system appropriately, we establish a solid foundation for managing resource allocation. The implementation of resource request handling, coupled with the Banker's algorithm, ensures that resource allocations are made safely, considering each process's maximum needs and the overall system state. The effectiveness of the Banker's algorithm lies in its ability to calculate remaining resource needs, assess system safety using the safety algorithm, and prevent deadlock by granting resource requests only when it's safe to do so. Additionally, deadlock detection mechanisms play a crucial role in identifying and resolving potential deadlock situations, ensuring system stability and resilience.

Through simulation and testing, we can analyse system behaviour under different scenarios, evaluate algorithm performance, and assess resource allocation efficiency. This analysis helps in understanding the system's strengths and areas for improvement, guiding further enhancements and optimizations. In practical applications, considerations such as error handling, system resilience, and documentation play vital roles in ensuring system reliability and maintainability. Addressing these aspects enables the implementation of deadlock detection and avoidance systems that are robust and effective in real-world environments. The implementation of deadlock detection and avoidance using the Banker's algorithm requires a comprehensive approach encompassing system design, algorithm implementation, testing, analysis, and documentation. This approach facilitates the development of reliable and efficient systems that can effectively manage resource allocation, prevent deadlock, and ensure system stability in complex computing environments.