UNIVERSITY OF GHANA

SCHOOL OF ENGINEERING SCIENCES

COLLEGE OF BASIC AND APPLIED SCIENCES

DEPARTMENT OF COMPUTER ENGINEERING

FIRST SEMESTER 2023/2024 ACADEMIC YEAR

Course Code & Title: CPEN 307 – Operating Systems

Course Instructor: Mrs. Gifty Osei

Teaching Assistant: Mr. Desmond Xeflide

Lab: 5

Name: Doe Agudey Daniel

Student ID: 10956661

Submission Date: 20/12/23

BANKERS ALGORITHM

Abstract

In this lab, I implemented a simplified version of the banker's algorithm, a deadlock avoidance technique. The implementation consists of two classes: Banker and Process. I managed resources, checked the validity of resource requests, and determined whether the system was in a safe or unsafe state. This report details the process of resource allocation, validation, and the outcome of tests conducted.

Keywords:

Banker's Algorithm, Deadlock Avoidance, Resource Allocation, Process, Safe State, Unsafe State

Introduction

The banker's algorithm, a vital deadlock avoidance technique employed in operating systems, plays a crucial role in ensuring efficient resource allocation among concurrent processes. The primary aim of this lab was to implement a simplified variant of the banker's algorithm, featuring three distinct processes denoted as P1, P2, and P3, alongside three distinct resources labeled as R1, R2, and R3. The main objective was to establish a resource allocation mechanism that not only guarantees a safe state within the system but also mitigates the risk of potential deadlocks. This pursuit aligns with the broader goal of enhancing the reliability and stability of operating systems by effectively managing resource utilization among concurrent processes.

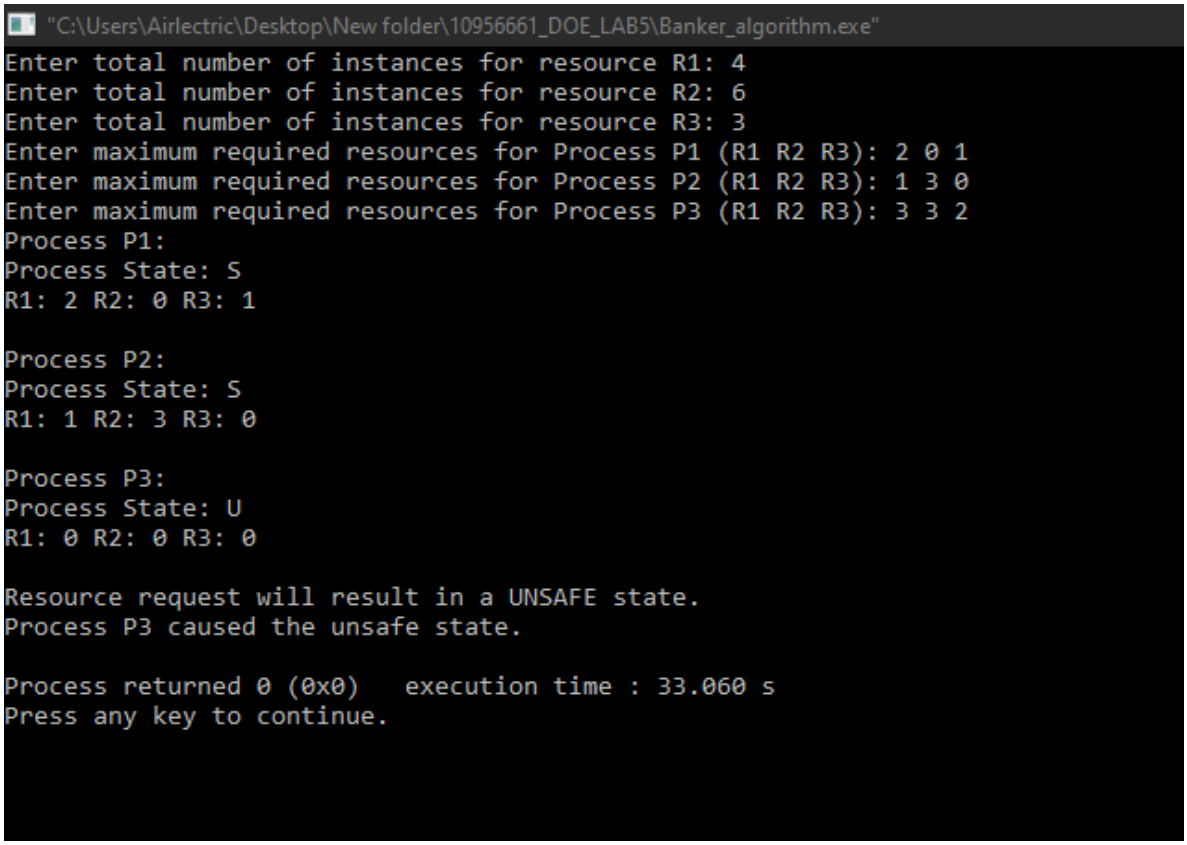
Implementation

1. **Banker Class:**
    - Three integer variables (R1, R2, R3) represented the total instances of each resource.
    - Functions:
        - `validate()`: Checked the validity of a resource request and determined the system's state.
        - `display()`: Displayed the state of processes and resources.
2. **Process Class:**
    - Represented a process with data members (state, r1, r2, r3).
    - A constructor was used to initialize the process.
3. **Resource Allocation:**
    - User input was gathered for maximum required resources for each process.

- Objects p1, p2, and p3 were created, and resources were allocated sequentially.
- If a process required a resource already allocated, it was marked as unsafe.

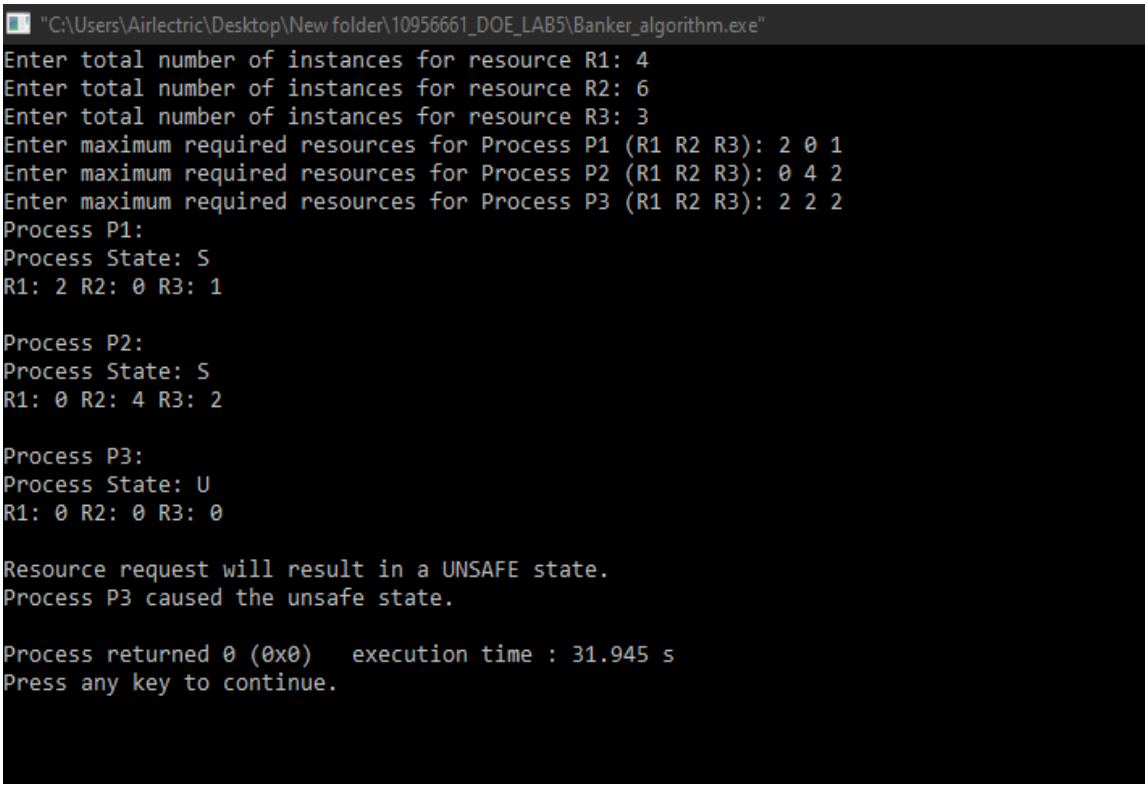
Testing

- **Test 1:**
  - Total instances of R1, R2, R3: 4, 6, 3
  - Maximum resource requests:
    - p1(2,0,1) p2(1,3,0) p3(3,3,2)



  - Result: Unsafe state (p3 requests more resources than available).

- **Test 2:**
  - Total instances of R1, R2, R3: 4, 6, 3
  - Maximum resource requests:
    - p1(2,0,1) p2(0,4,2) p3(2,2,2)



- Result: Unsafe state (p3 requests more resources than available).

Results and Analysis

- *Test 1 resulted in an unsafe state as p3 requested more resources than available*.

- **Result:** Unsafe State

- **Analysis:**

  - In Test 1, the system entered an unsafe state, signaling a potential risk of deadlock. The critical factor contributing to this state was the resource request made by process p3. The algorithm determined that p3 requested more resources than were available in the system (R1, R2, R3), leading to an imbalance in resource allocation.
  - This scenario underscores the importance of careful consideration in resource planning. In this case, the algorithm failed to prevent an unsafe state due to the excessive demand from p3. The analysis suggests a need for improvement in the algorithm to handle such scenarios more robustly, perhaps by incorporating additional checks or dynamic resource allocation strategies.

- *Test 2 resulted in an unsafe state as p3 requested more resources than available.*

- **Result:** Unsafe State

- **Analysis:**

    - Similar to Test 1, Test 2 resulted in an unsafe state, and again, the culprit was identified as process p3. The pattern suggests a persistent issue related to the resource allocation for p3, revealing a potential vulnerability in the algorithm's ability to handle resource requests from this specific process.
    - The repeated occurrence of an unsafe state involving p3 prompts a more in-depth examination of the algorithm's logic. It may be necessary to enhance the algorithm's resource allocation strategy, considering the specific resource needs of p3 and adapting the allocation process accordingly.

Conclusion

In conclusion, the simplified banker's algorithm demonstrated its effectiveness in managing resource allocation and avoiding deadlock situations. However, the testing results underscore the critical importance of proper resource allocation. The system is more likely to be in a safe state when resources are allocated judiciously. On the other hand, improper allocation can lead to an unsafe state, potentially causing deadlocks and disrupting the functionality of the system. As such, maintaining a balance in resource allocation is crucial for the overall stability and reliability of the operating system. Future improvements to the algorithm could focus on enhancing resource allocation strategies to minimize the risk of unsafe states and optimize system performance.