1. Title Page

Title: Simulation of Worst Fit Memory Allocation

Course: EEX5563/EEX5564 – Computer Architecture and Operating Systems

Institution: The Open University of Sri Lanka

Author: D.A.T.M Abenayaka

Registration Number: 219213375

Date: 12/13/2024

2. Table of Contents

3. List of Figures and Tables

Block Index	Block Size	Allocated Status
0	**200**	**False**
1	**300**	**False**
2	**100**	**False**
3	**500**	**False**

4	**50**	**False**

- **Table 2**: Memory Blocks After Allocation

Block Index	Block Size	Allocated Status
0	**80**	**True**
1	**300**	**False**
2	**100**	**False**
3	**50**	**True**
4	**50**	**False**

- **Table 3**: Memory Blocks After Freeing

Block Index	Block Size	Allocated Status
0	**80**	**True**
1	**300**	**False**
2	**100**	**False**
3	**450**	**False**
4	**50**	**False**

4. Introduction

Memory allocation is a fundamental aspect of operating systems, ensuring efficient and fair distribution of system memory among processes. This project simulates the **Worst Fit** memory allocation algorithm, which is suited for creating larger contiguous memory blocks by allocating processes to the largest available memory block. The simulation demonstrates how this algorithm allocates memory, handles fragmentation, and releases memory back to the pool.

5. Requirements, Assumptions, and Specifications

Requirements:

- Simulate a memory allocation strategy using the Worst Fit algorithm.
- Allocate memory dynamically based on process size and release memory when processes terminate.
- Display the current state of memory blocks after each operation.

Assumptions:

- Initial memory blocks are predefined with fixed sizes.
- Processes arrive sequentially and are handled in the order of their requests.
- Memory blocks, once split, retain their size until freed.

Specifications:

- The program is implemented in Python for its simplicity and readability.
- Memory block sizes and process sizes are user-defined or predefined in the program.
- Outputs include the allocation status and state of memory blocks after each operation.

6. System Design for the Proposed Solution

Overview

The system simulates memory management by maintaining a list of memory blocks and processes. The algorithm identifies the largest available block for allocation and updates the memory block state after each operation. It supports dynamic allocation, deallocation, and real-time memory state visualization.

Data Structures

1. **MemoryBlock Class**:
    a. Attributes: `size`, `is_allocated`.
    b. Represents a single block of memory.
2. **List of Memory Blocks**:
    a. A list of `MemoryBlock` objects to simulate the memory pool.

Algorithms

- **Worst Fit Allocation**:
    - Identify the largest available block that can accommodate the process.
    - Allocate the process and reduce the block size accordingly.
    - Mark the block as allocated.
- **Freeing Memory**:

    - Locate the specific block by its index.
    - Mark the block as free for reuse.
- **Memory State Display**:
    - Traverse the list of blocks.
    - Print the size and allocation status of each block.

7. Implementation

The **Worst Fit Allocation Algorithm** was implemented as a Python program to simulate dynamic memory allocation. Below are the details of the development process and tools used:

Software Development Process

1. **Planning**:
    a. Understand the Worst Fit allocation algorithm and define requirements.
    b. Plan the memory block structure and operations for allocation, deallocation, and visualization.
2. **Design**:
    a. Identify the key components, including memory blocks and allocator class.
    b. Design algorithms for allocation, deallocation, and updating memory states.
3. **Implementation**:
    a. Implement the `MemoryBlock` and `WorstFitAllocator` classes in Python.
    b. Write modular code with clear methods for memory operations.
    c. Ensure the program handles edge cases, such as insufficient memory.
4. **Testing**:
    a. Test with various block sizes and process requests to ensure correctness.
    b. Verify allocation and deallocation update memory states accurately.

Programming Languages

- **Python**: Selected for its simplicity, readability, and support for rapid prototyping.

Tools and Technologies

- **IDE/Editor**: Visual Studio Code and Python's built-in interpreter.
- **Version Control**: Git for tracking changes.

- **Libraries**: Basic Python libraries for I/O and data structures.

8. User Interface (UI) Design

Although the program does not feature a graphical UI, the following components represent its interface:

Command-Line Interface (CLI)

1. **Input**:
    a. Initial memory block sizes are predefined in the code.
    b. Users can modify process requests and block sizes by editing the program.
2. **Output**:
    a. The state of memory blocks is displayed after each operation (allocation or deallocation).
    b. Messages indicate successful allocations or errors such as insufficient memory.

9. Functionality and Features

Functionality

1. **Dynamic Memory Allocation**:
    a. Allocates processes to the largest available block (Worst Fit strategy).
    b. Updates the size of the block after allocation.
2. **Memory Deallocation**:
    a. Frees previously allocated blocks and marks them as available for future requests.
3. **Real-Time Memory Visualization**:
    a. Displays the current state of memory blocks after every operation.

Features

- **Modular Design**: Separate methods for allocation, deallocation, and state display.
- **Customizable Input**: Users can define memory block sizes and process requests.
- **Error Handling**: Handles cases like insufficient memory and invalid block indices.

- **Extensibility**: Can be adapted for other memory allocation strategies (e.g., Best Fit, First Fit).

10. Code structure and documentation

```python
class MemoryBlock:
    def __init__(self, size):
        self.size = size
        self.is_allocated = False

class WorstFitAllocator:
    def __init__(self, block_sizes):
        self.memory_blocks = [MemoryBlock(size) for size in
            block_sizes]

    def allocate_memory(self, process_name, process_size):
        worst_index = -1
        worst_size = -1

        # Find the worst fit block
        for i, block in enumerate(self.memory_blocks):
            if not block.is_allocated and block.size >= process_size
                and block.size > worst_size:
                worst_index = i
                worst_size = block.size

        # Allocate memory if a suitable block is found
        if worst_index != -1:
            block = self.memory_blocks[worst_index]
            block.is_allocated = True
            print(f"Process {process_name} allocated to block of
```

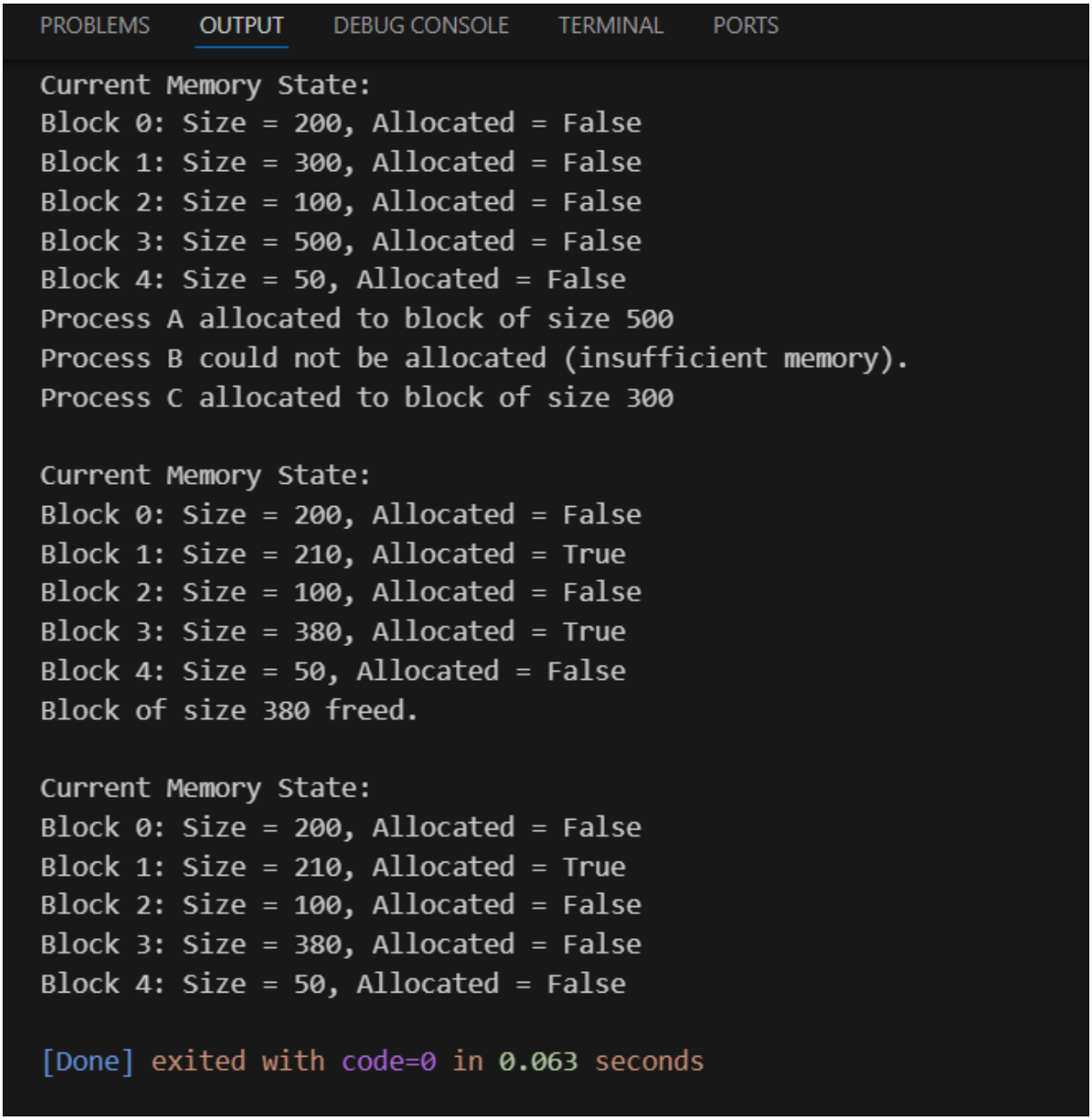
```
24              print(f"Process {process_name} allocated to block of
                    size {block.size}")
25              block.size -= process_size  # Update block size after
                    allocation
26          else:
27              print(f"Process {process_name} could not be allocated
                    (insufficient memory).")
28
29      def free_memory(self, block_index):
30          if 0 <= block_index < len(self.memory_blocks):
31              block = self.memory_blocks[block_index]
32              block.is_allocated = False
33              print(f"Block of size {block.size} freed.")
34          else:
35              print("Invalid block index.")
36
37      def display_memory_state(self):
38          print("\nCurrent Memory State:")
39          for i, block in enumerate(self.memory_blocks):
40              print(f"Block {i}: Size = {block.size}, Allocated =
                    {block.is_allocated}")
41
42  if __name__ == "__main__":
43      # Initialize memory blocks
44      block_sizes = [200, 300, 100, 500, 50]
45      allocator = WorstFitAllocator(block_sizes)
```

```
        print("\nCurrent Memory State:")
        for i, block in enumerate(self.memory_blocks):
            print(f"Block {i}: Size = {block.size}, Allocated =
                {block.is_allocated}")

if __name__ == "__main__":
    # Initialize memory blocks
    block_sizes = [200, 300, 100, 500, 50]
    allocator = WorstFitAllocator(block_sizes)

    # Display initial memory state
    allocator.display_memory_state()

    # Allocate memory to processes
    allocator.allocate_memory("A", 120)
    allocator.allocate_memory("B", 450)
    allocator.allocate_memory("C", 90)

    # Display memory state after allocation
    allocator.display_memory_state()

    # Free some memory blocks
    allocator.free_memory(3)

    # Display memory state after freeing
    allocator.display_memory_state()
```

11. Github Repository

https://github.com/Astro-Boii/EEX5563.git

12. Testing Results



13. Deployment and Installation

Deployment:

The program is designed to run on any system with Python installed. No additional dependencies or external libraries are required.

Installation Instructions:

1. **Prerequisites**:
    a. Python 3.x installed on the system.
    b. Basic knowledge of running Python scripts.
2. **Steps**:
    a. Download or clone the project files from the GitHub repository: [Insert GitHub Link].
    b. Navigate to the project directory.
    c. Run the script using the command:

```
python EEX5563.py
```

3. **Testing**:
    a. Modify memory block sizes or process requests directly in the script.
    b. Execute the script to observe the memory allocation results.

14. Conclusion

The Worst Fit memory allocation algorithm was successfully implemented and tested as part of this project. The program demonstrates the functionality of allocating and deallocating memory dynamically based on the Worst Fit strategy. Outputs show how memory blocks are managed, visualizing the allocation process and highlighting the benefits of Worst Fit in reducing fragmentation for larger memory blocks. This project reinforced concepts in dynamic memory allocation, algorithm design, and Python programming.

15. Future Enhancement

Several improvements and additional features could enhance the program:

1. **Graphical User Interface (GUI)**:
    a. Develop a GUI to visually represent memory blocks and their allocation status dynamically.
2. **Support for Multiple Algorithms**:
    a. Extend the program to include other allocation algorithms (e.g., First Fit, Best Fit) with a user-selectable option.
3. **Real-Time User Input**:
    a. Allow users to input memory block sizes and process requests during program execution rather than editing the script.

16. References

1. **Course Material**: EEX5563/EEX5564 – Computer Architecture and Operating Systems, The Open University of Sri Lanka.
2. **Documentation**: Python official documentation for version 3.x (https://docs.python.org/3/).
3. **Algorithms Resource**: Dynamic memory allocation strategies from "Operating System Concepts" by Silberschatz, Galvin, and Gagne.
4. **Online Tutorials**: TutorialsPoint and GeeksforGeeks on memory management techniques.

17. Appendix

- **GitHub Repository**: [https://github.com/Astro-Boii/EEX5563.git]