**Operating Systems Lab – Project Documentation**

**Project Title:**  
Memory Allocation Simulator Using First Fit, Best Fit, and Worst Fit Strategies in Python

**Group Members**

* Ahmed Jalal (SU92-BSSEM-F23-075)
* Abdullah Asif (SU92-BSSEM-F23-098)
* Abdul Rehman (SU92-BSSEM-F23-094)
* M. Awais Jamil (SU92-BSSEM-F23-064)

**GitHub Repository:**

* https://github.com/Ahmedjalalz/OS-Project

**Scheduling Algorithm Implemented:**

* **First Fit** – Allocates the first block that is large enough for the process.
* **Best Fit** – Allocates the smallest block that fits the process.
* **Worst Fit** – Allocates the largest available block that fits the process.

**Project Description:**

This project is a simulation of dynamic memory allocation strategies typically used in operating systems. It demonstrates how processes are assigned to memory blocks using First Fit, Best Fit, and Worst Fit algorithms. The simulation allows users to enter process sizes interactively and view how each strategy allocates memory over time.

**Inputs Required:**

* Fixed memory block sizes: [100, 500, 200, 300, 600]
* Size of each process entered by the user

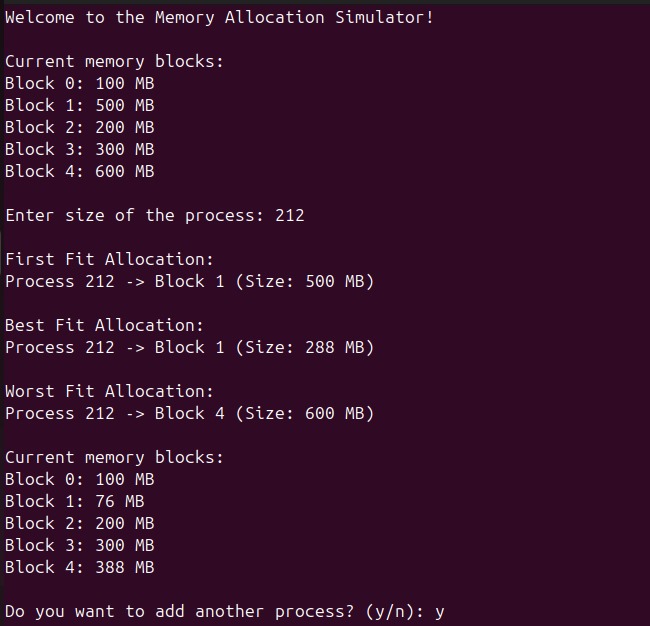
**Outputs Generated:**

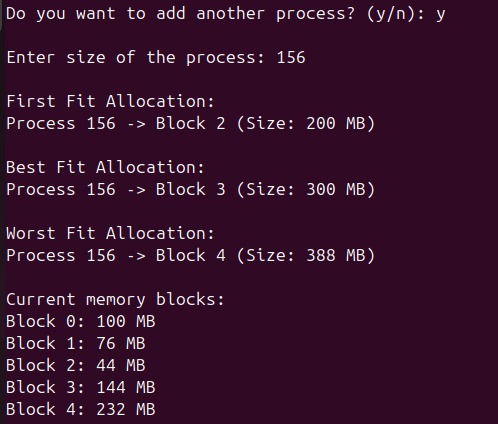
* Allocation details of each process under all three strategies
* Updated memory block status after each allocation

**Implementation:**

* Interactive loop that asks the user for process sizes
* Functions for First Fit, Best Fit, and Worst Fit allocations
* Real-time updating and display of memory block states

**Output Screenshots:**





**Code Structure & Explanation:**

**Functions Used:**

* display\_memory\_blocks() – Displays current memory block sizes
* first\_fit(p) – Allocates process using First Fit
* best\_fit(p) – Allocates process using Best Fit
* worst\_fit(p) – Allocates process using Worst Fit

**Core Logic:**

* The program uses a persistent list to store current memory block sizes
* Each strategy function updates this list as processes are allocated
* Allocation result is printed after every process entry
* The simulation continues interactively until the user chooses to stop

**External Libraries Used:**

* None (uses only standard Python features)

**Performance Metrics:**

|  |  |
| --- | --- |
| **Metric** | **Example Value** |
| Initial Memory Blocks | [100, 500, 200, 300, 600] |
| Total Processes Tested | User-defined |
| Allocation Success Rate | Varies by strategy |

**Challenges Faced:**

1. **Dynamic Allocation Handling**  
   Ensuring memory blocks update correctly after each allocation. Resolved by using a persistent list and deep copying where needed.
2. **User Input Validation**  
   Preventing crashes from invalid user input. Solved by using try-except blocks.
3. **Output Clarity**  
   Making sure the outputs for all three strategies are readable. Enhanced with labeled print statements and spacing.

**Conclusion:**

This simulator offers a clear, interactive demonstration of how memory allocation strategies work in an OS. It helps visualize memory fragmentation and allocation efficiency under different strategies.