CSCE 5640: Operating System Design

Project Report

Name: Kishan Kumar Zalavadia

EUID: 11685261

Title: Contiguous Memory Allocation

Member name: Kishan Kumar Zalavadia

Project GitHub link: https://github.com/Kishan-Kumar-Zalavadia/OS_Contiguous_Memory_Allocation

1. **Introduction**

   Overview

   This project aims to learn and implement different algorithms for contiguous memory allocation. We will use different memory allocation techniques, like first fit, best fit, and worst fit. This project will handle different inputs like request memory, release memory, compact memory and status report.

   Problem Statement

   In modern operating systems, memory management is very important. Allocating memory inefficiently can lead to degrading performance. Hence, it's important to manage memory efficiently among multiple processes. The OS will manage memory requests and memory releases in an effective way using different techniques.

   Importance

   Managing memory in an operating system is very crucial for the performance of any system. It becomes more crucial for the system, which can perform multitasking. If the memory allocation is not executed properly, the performance of the system can rapidly decrease. Hence, by improving and optimizing the memory management system, it can run efficiently and can support more processes along with better usage of the hardware resources available.

2. **Background**

   To understand the project and its importance, below are some basic concepts of memory management.

   Memory allocation: A process can require memory to execute. So, the operating system allocates the memory required by that process to that process so that the process can execute. If there are multiple processes that require memory at the same time, it's important to manage the memory allocation efficiently.

   Below are some memory allocation strategies we will use in this project.

   1. **Best Fit**: In this strategy, we will allocate the first available memory block, which is large enough to fulfill the memory requirements of the process. The strategy is very simple to implement, but it may lead to fragmentation.

2. **Best Fit**: In this strategy, we find the smallest sufficient memory block. Which means we try to have a hole with the least possible size. This strategy tends to minimize wasted space, but it requires more search time.
3. **Worst Fit**: this strategy is the opposite of best fit in this strategy, the OS allocates the largest available. Memory block to the process. The aim of the strategy is to maintain the largest free hole or block. This strategy can be inefficient.

3. Implementation

Solution Approach

To implement memory allocation using first fit best fit, and worst fit we use the following process. First, we stored the inputs in an input.TXT file, which contains instructions in order. The first line of the TXT file is the initial/total memory in a memory block. Then, there are different processes that request memory using one of the above fits, which also specifies the amount of memory. It requires a process that can also request a release of memory when the process is completed. There is also a compact command, which will merge all the holes present in the memory, and the last command is stats, which will display the memory block details, for example, what part of memory is allocated to a process and what part of memory is unused.

The MemoryAllocator.CPP file contains the code to allocate memory based on the Input text file given. This project contains the MemoryAllocatorTestCaseGenerator.java file, which will generate multiple input files of different initial memory as the test cases. MultiFileMemoryAllocator.cpp file will run all the test cases one by one and display the result. It will also store the results in a dot TXT file which will be used to analyze the result. This project contains a Python code that will take the output.TXT file as input and visualize and analyze the results.

Implementation Details:

- **Programming Language:** CPP, Java, and Python
    - CPP: Implement memory allocation
    - Java: Creating test cases
    - Python: Visualization the results.
- **Operating system:** Linux CSE machine.

Test cases:

The test cases are randomly generated using a Java code. The following are some of the test cases which are used.

Below is the average waiting time by different scheduling algorithms.

Test case 1: test_4MB_F.txt

4096

RQ P1 69 F

RQ P2 155 F

RL P1

C

RQ P6 24 F

RL P2

RQ P8 301 F

C

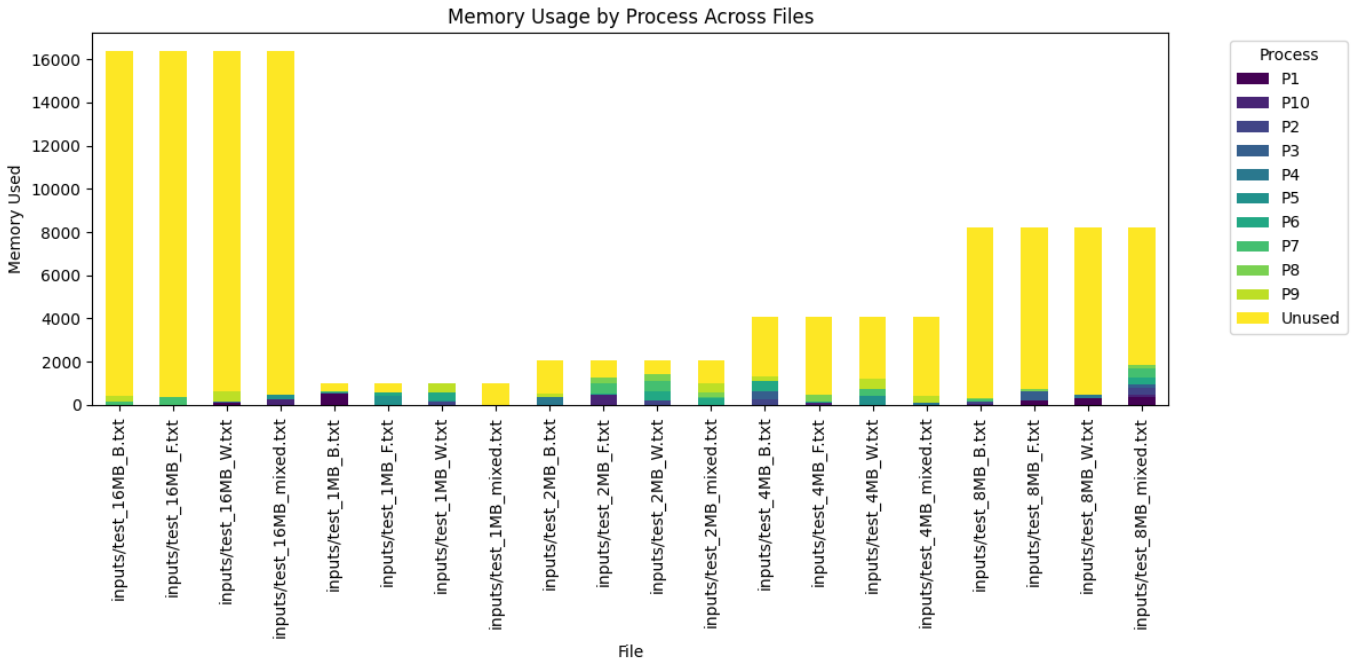
RQ P10 131 F

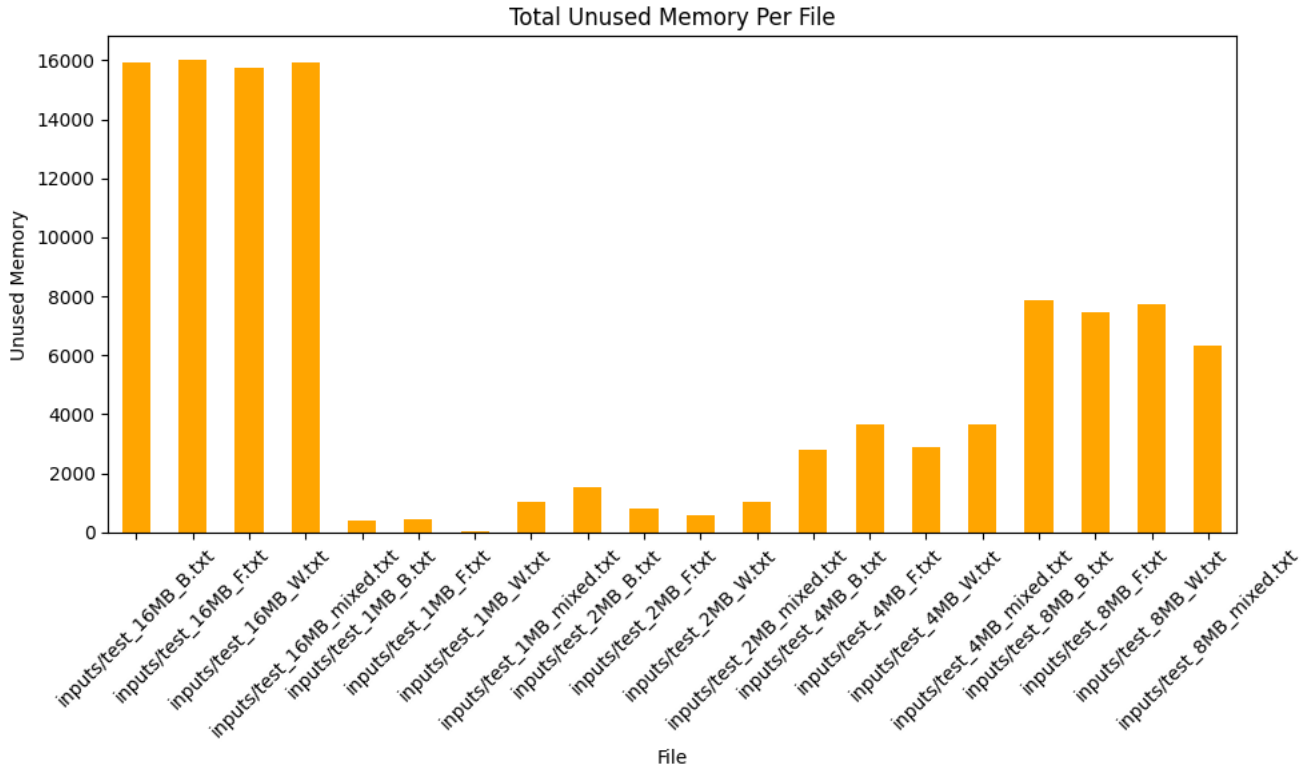
STAT

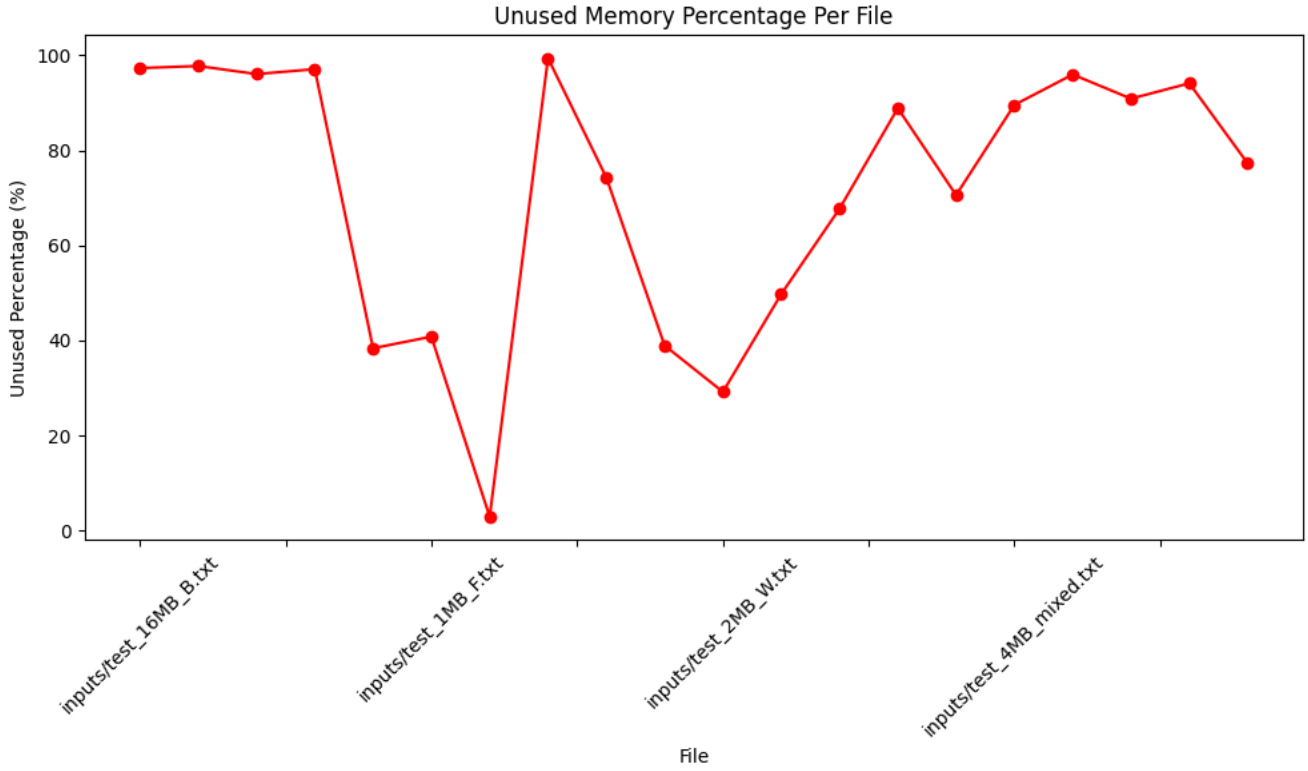
Test case 2: test_8MB_B.txt
8192
C
RQ P2 92 B
RQ P4 75 B
C
RL P4
RQ P7 118 B
RQ P8 60 B
C
RQ P10 60 B
STAT

Test case 3: test_4MB_mixed.txt
4096
C
RQ P4 116 F
RQ P5 439 B
C
RL P5
C
RQ P9 311 F
STAT

4. **Experimental Results:**


Memory Usage by Process Across Files
Memory Used
File
Process
P1
P10
P2
P3
P4
P5
P6
P7
P8
P9
Unused
inputs/test_16MB_B.txt
inputs/test_16MB_F.txt
inputs/test_16MB_W.txt
inputs/test_16MB_mixed.txt
inputs/test_1MB_B.txt
inputs/test_1MB_F.txt
inputs/test_1MB_W.txt
inputs/test_1MB_mixed.txt
inputs/test_2MB_B.txt
inputs/test_2MB_F.txt
inputs/test_2MB_W.txt
inputs/test_2MB_mixed.txt
inputs/test_4MB_B.txt
inputs/test_4MB_F.txt
inputs/test_4MB_W.txt
inputs/test_4MB_mixed.txt
inputs/test_8MB_B.txt
inputs/test_8MB_F.txt
inputs/test_8MB_W.txt
inputs/test_8MB_mixed.txt


Total Unused Memory Per File
Unused Memory
File
inputs/test_16MB_B.txt
inputs/test_16MB_F.txt
inputs/test_16MB_W.txt
inputs/test_16MB_mixed.txt
inputs/test_1MB_B.txt
inputs/test_1MB_F.txt
inputs/test_1MB_W.txt
inputs/test_1MB_mixed.txt
inputs/test_2MB_B.txt
inputs/test_2MB_F.txt
inputs/test_2MB_W.txt
inputs/test_2MB_mixed.txt
inputs/test_4MB_B.txt
inputs/test_4MB_F.txt
inputs/test_4MB_W.txt
inputs/test_4MB_mixed.txt
inputs/test_8MB_B.txt
inputs/test_8MB_F.txt
inputs/test_8MB_W.txt
inputs/test_8MB_mixed.txt



Analysis Summary:

	File	Total Memory	Unused Memory	Unused Percentage
0	inputs/test_16MB_B.txt	16384	15944	97.314453
1	inputs/test_16MB_F.txt	16384	16017	97.760010
2	inputs/test_16MB_W.txt	16384	15738	96.057129
3	inputs/test_16MB_mixed.txt	16384	15908	97.094727
4	inputs/test_1MB_B.txt	1024	393	38.378906
5	inputs/test_1MB_F.txt	1024	418	40.820312
6	inputs/test_1MB_W.txt	1024	30	2.929688
7	inputs/test_1MB_mixed.txt	1024	1018	99.414062
8	inputs/test_2MB_B.txt	2048	1522	74.316406
9	inputs/test_2MB_F.txt	2048	798	38.964844
10	inputs/test_2MB_W.txt	2048	598	29.199219
11	inputs/test_2MB_mixed.txt	2048	1019	49.755859
12	inputs/test_4MB_B.txt	4096	2774	67.724609
13	inputs/test_4MB_F.txt	4096	3640	88.867188
14	inputs/test_4MB_W.txt	4096	2892	70.605469
15	inputs/test_4MB_mixed.txt	4096	3669	89.575195
16	inputs/test_8MB_B.txt	8192	7862	95.971680
17	inputs/test_8MB_F.txt	8192	7445	90.881348
18	inputs/test_8MB_W.txt	8192	7710	94.116211
19	inputs/test_8MB_mixed.txt	8192	6332	77.294922

Based on different test cases, there are different amounts of memory unused. The efficiency depends on which fit is used, but since we are compacting the holes, it improves the performance.

The above graphs show memory usage by a process across files, total unused, memory per file, and unused memory percentage per file, which totally depends on the input.

5. Conclusion

This project accomplishes memory, allocation, simulation, and visualizing the result. This project successfully implements three fit techniques used in memory allocation, which are first fit, best fit, and worst fit. The project also shows the memory allocation for different test cases and visualizes the results.

Future Work

- Implementation of additional memory allocation techniques.
- Optimizing the compaction algorithm.

- Adding priorities to the processes for memory allocation.
- Improving the scalability.

6. References

https://www.geeksforgeeks.org/memory-management-in-operating-system/
https://www.geeksforgeeks.org/first-fit-allocation-in-operating-systems/