

CS342 Operating Systems

Project 3

Experiments Report

Ata Seren Osman Semih Tiryaki

21901575 21801994

Section-2 Section-1

Introduction

Before starting to code our library and its functions, we researched how we can access the data of our allocated segment with mmap() and manipulate that data. We learned that we need to access and manipulate data by using bitwise operators and for initialization, allocation and freeing, we need to use these operators in different and specific methods. After deciding on these methods, we started to code the functions of the library. We double-checked every function after we wrote it and at the end, we tested them with basic and complex test cases, including threads.

Computer Specifications

We performed experiments on a laptop with Windows 10 Home, Intel i7-9750H CPU @ 2.60GHz with 4.2GHz Turbo and 16,0 GB of RAM.

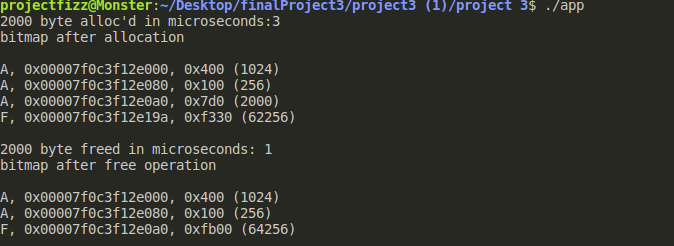
We used Oracle VirtualBox VM to run the code on virtual machine with Ubuntu 20.04. I reserved 5 CPU cores and 10725MB’s of RAM for the virtual machine. In the machine, we used Linux terminal to compile and run the code and VS Code to write the code.

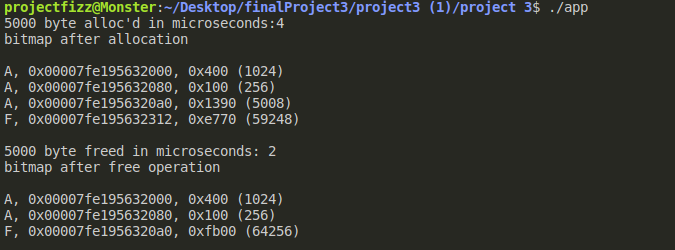
Experiments

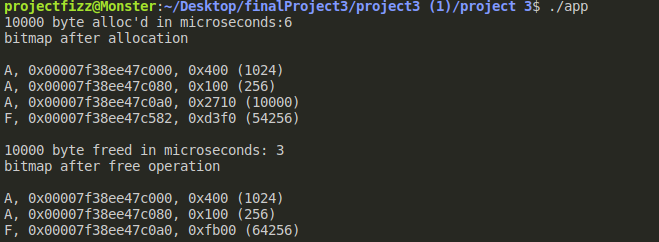
Other than the functionality and thread tests, we conducted 2 types of experiments: Constant allocation sizes and variable segment size, constant segment size and variable allocation sizes. For both allocating and freeing, we measured the time passed. For the latter type, we created random allocation sizes and used them in various test cases.

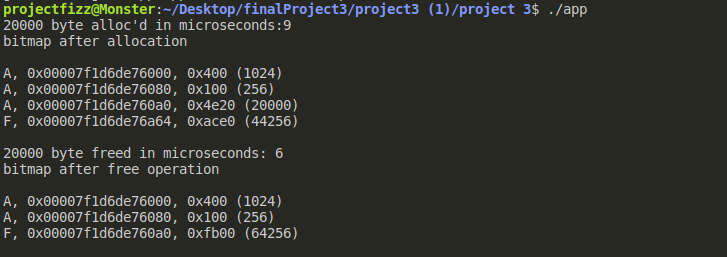
A) Constant allocation size, variable segment size

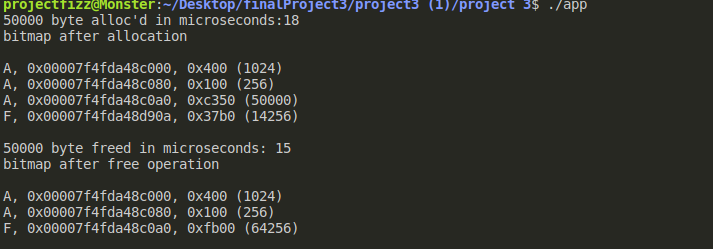
Segment Size: 216, Allocation sizes: 2000, 5000, 10000, 20000, 50000



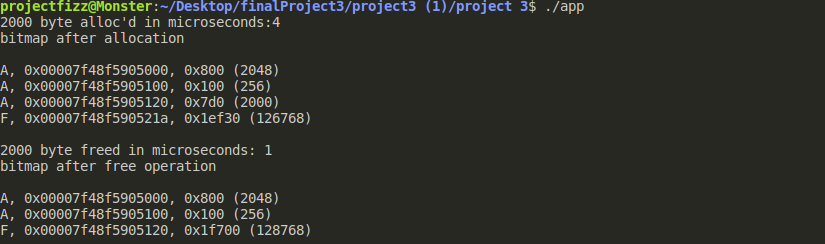


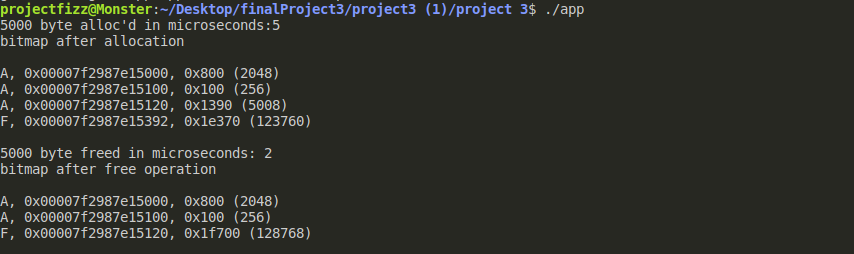


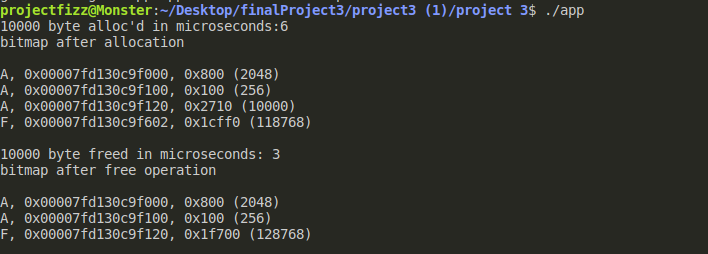


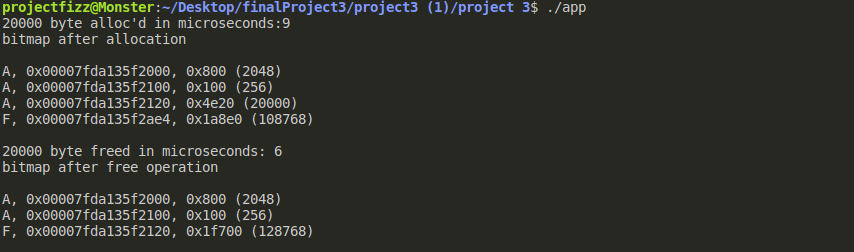


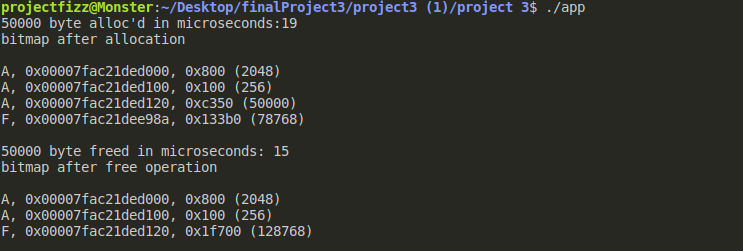
Segment Size: 217, Allocation sizes: 2000, 5000, 10000, 20000, 50000



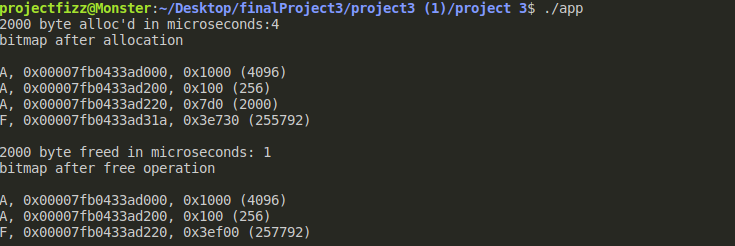


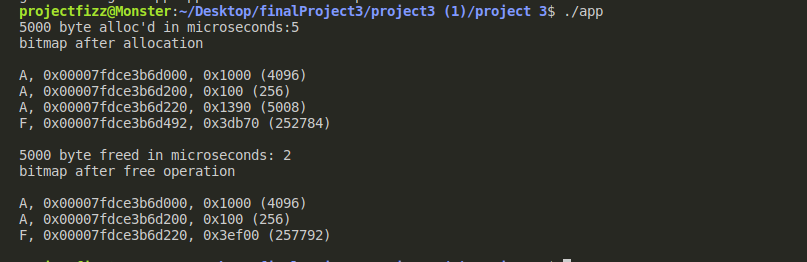


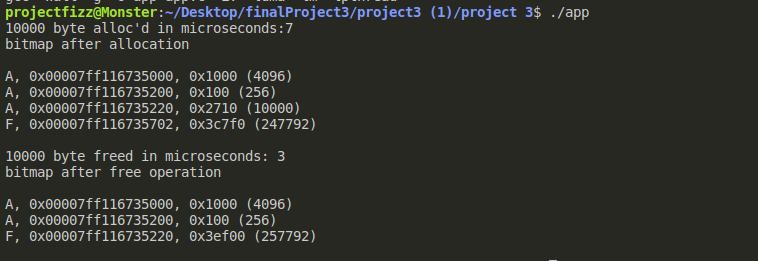


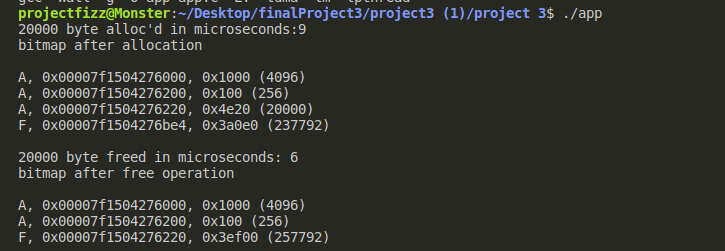


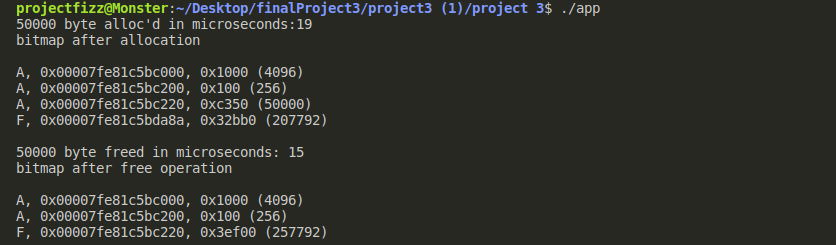
Segment Size: 218, Allocation sizes: 2000, 5000, 10000, 20000, 50000



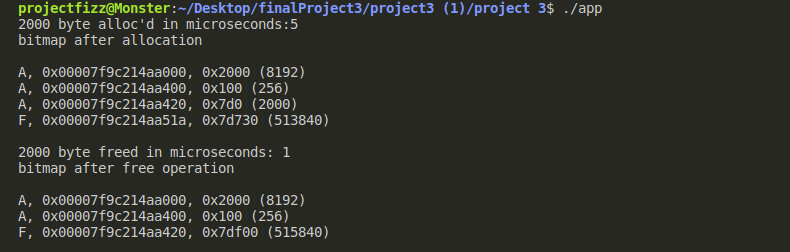


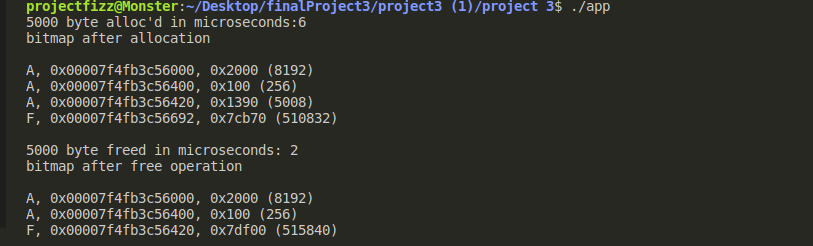


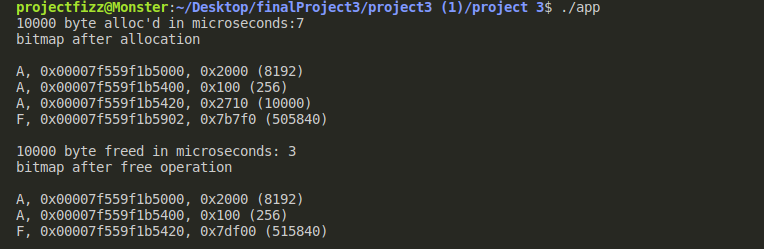


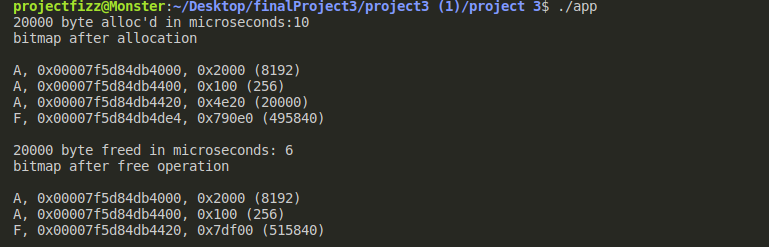


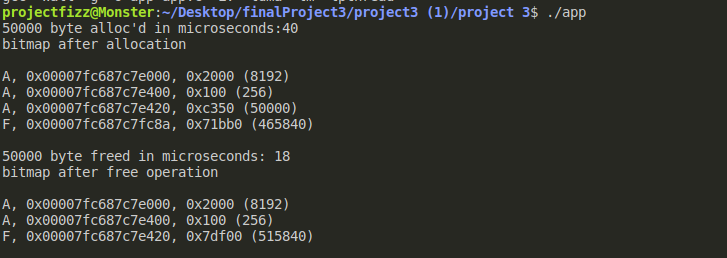
Segment Size: 219, Allocation sizes: 2000, 5000, 10000, 20000, 50000



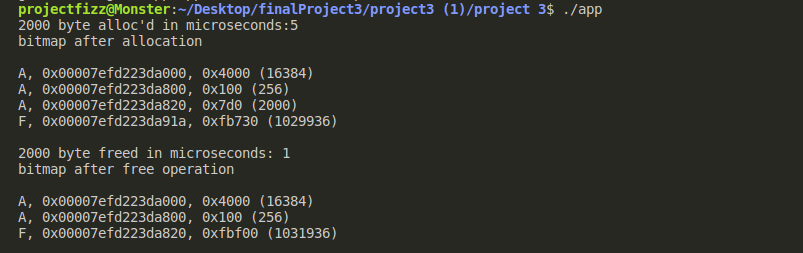


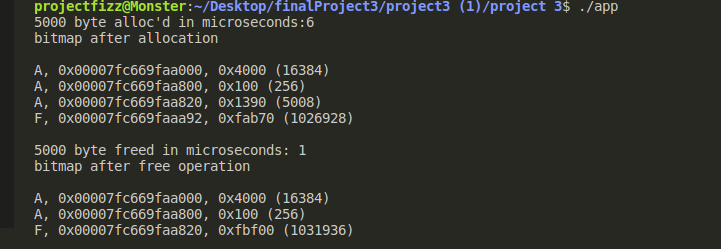


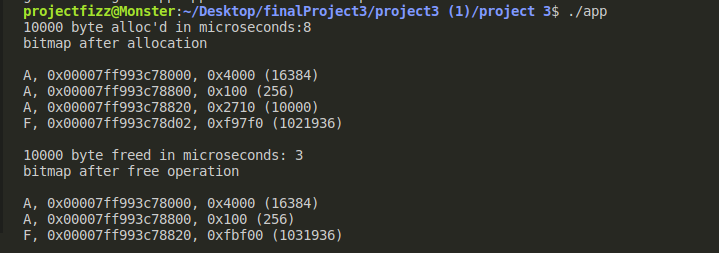


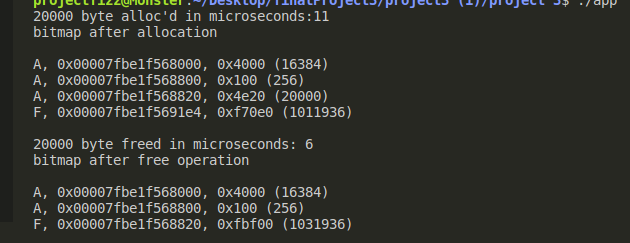


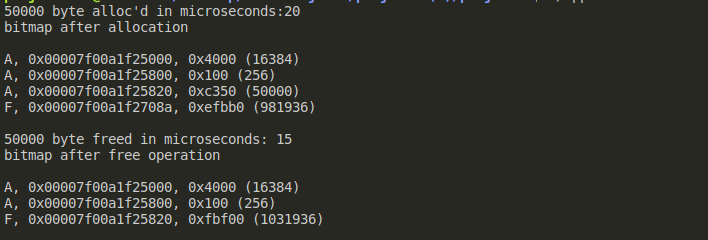
Segment Size: 220, Allocation sizes: 2000, 5000, 10000, 20000, 50000







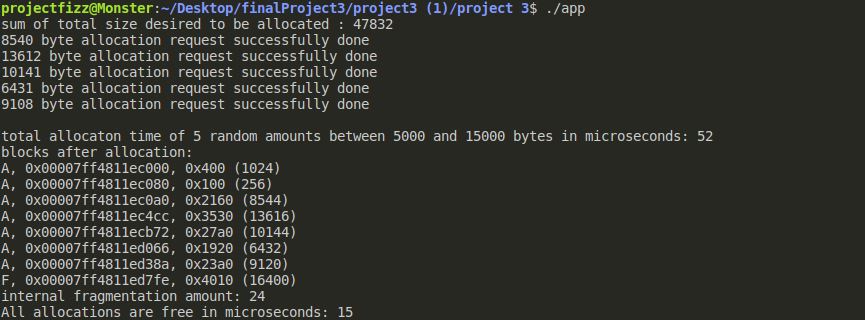


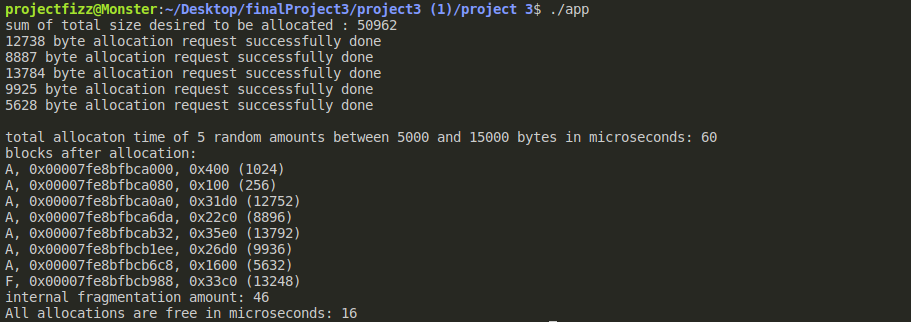


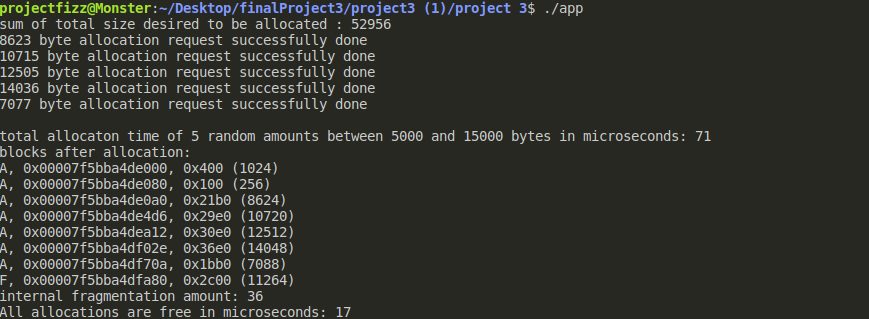
B) Constant segment size, variable allocation size

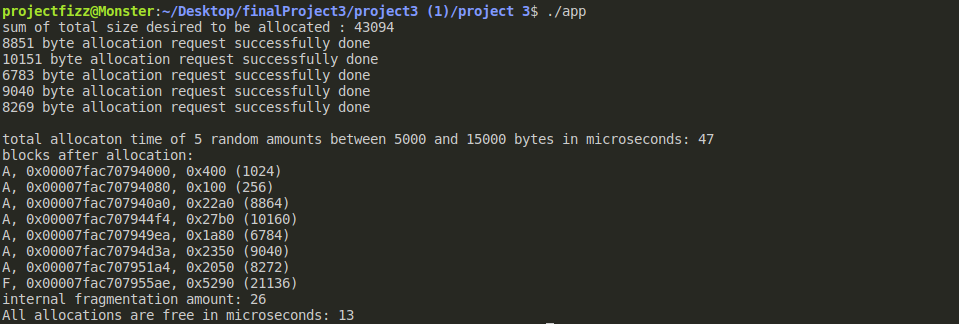
5 random allocation between 5000 and 15000

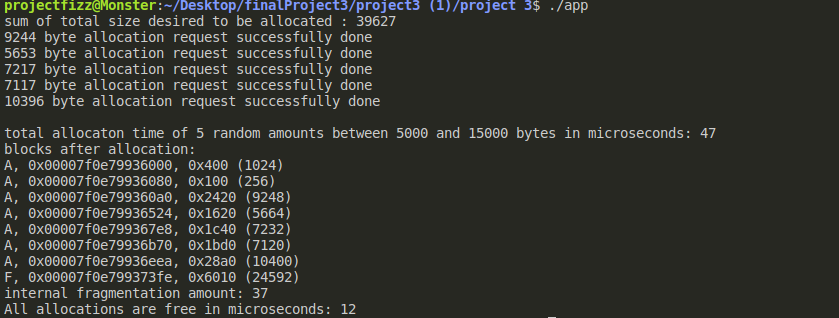
Segment size is 64KB



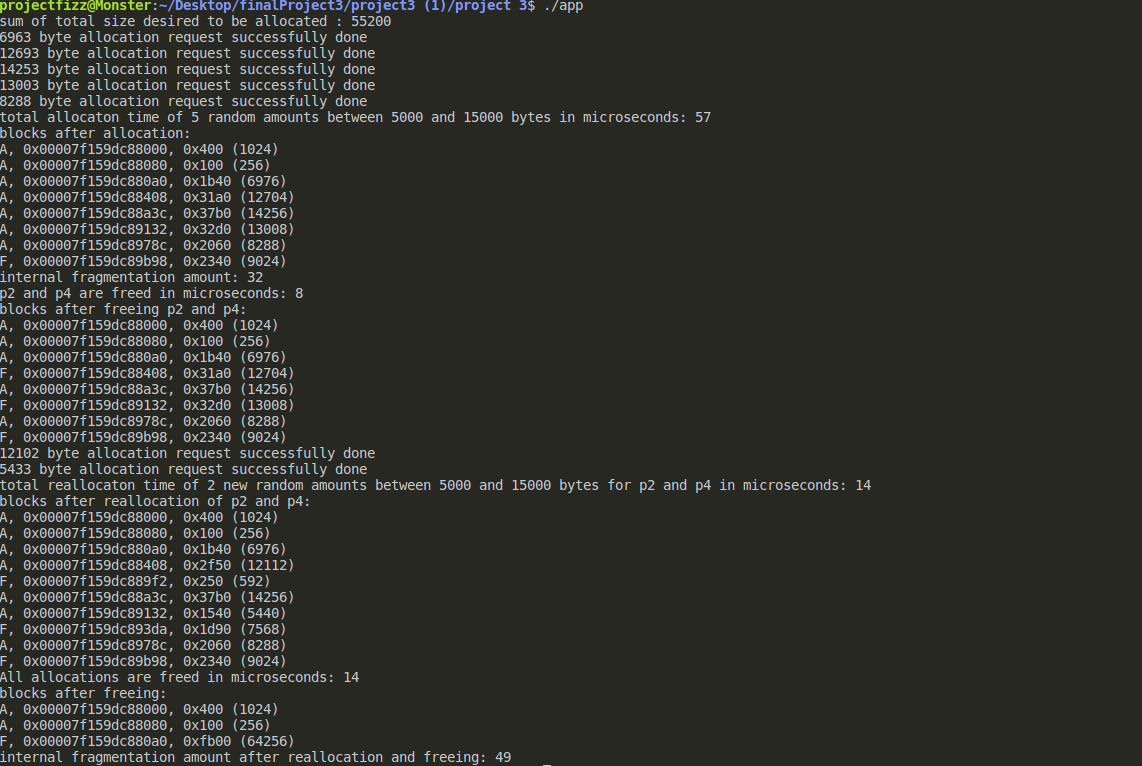


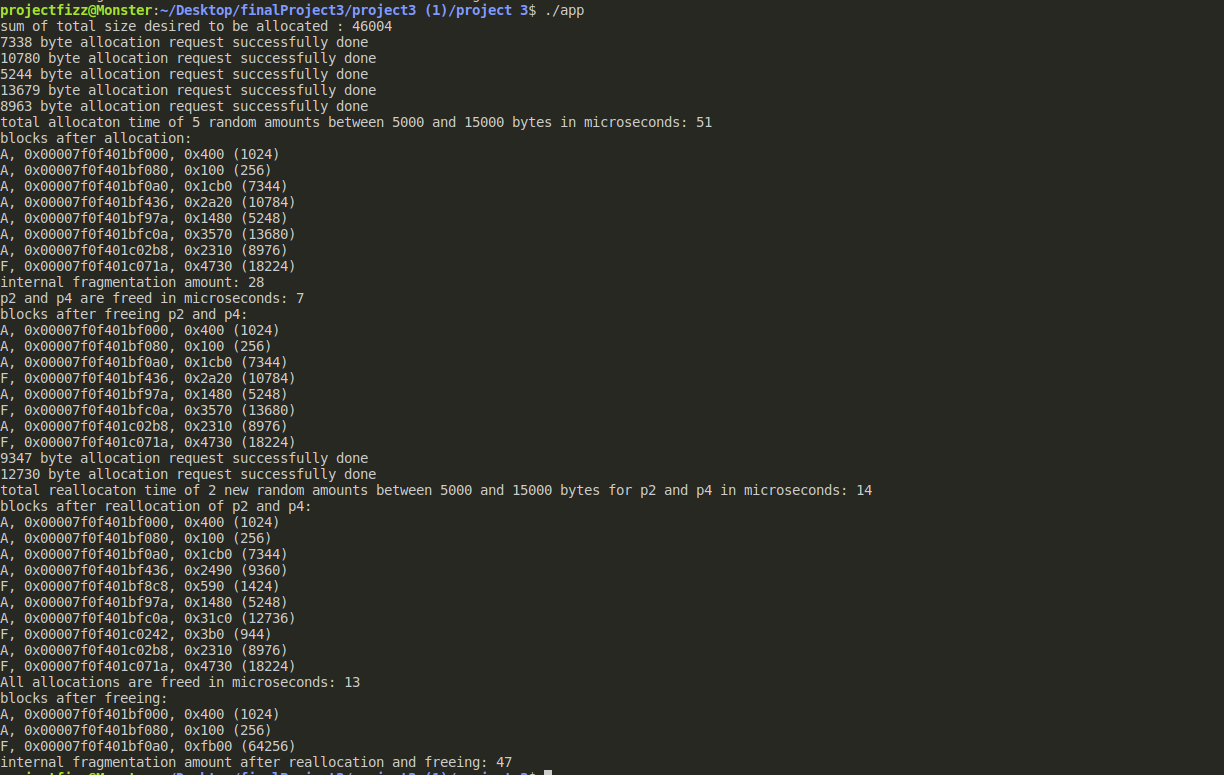


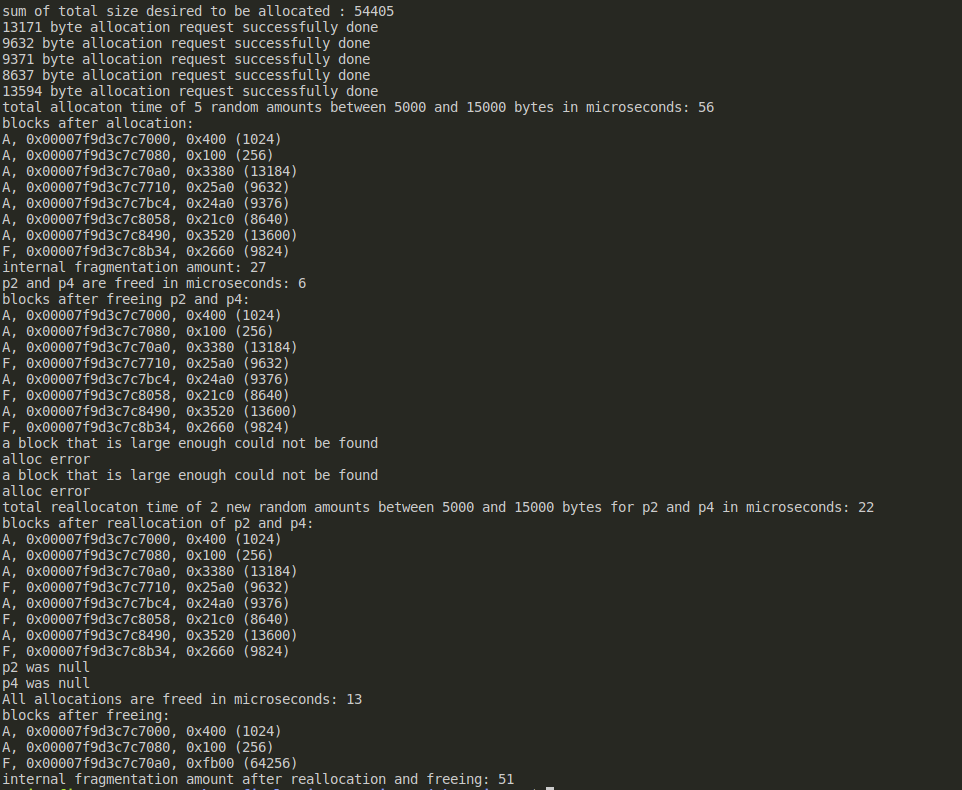


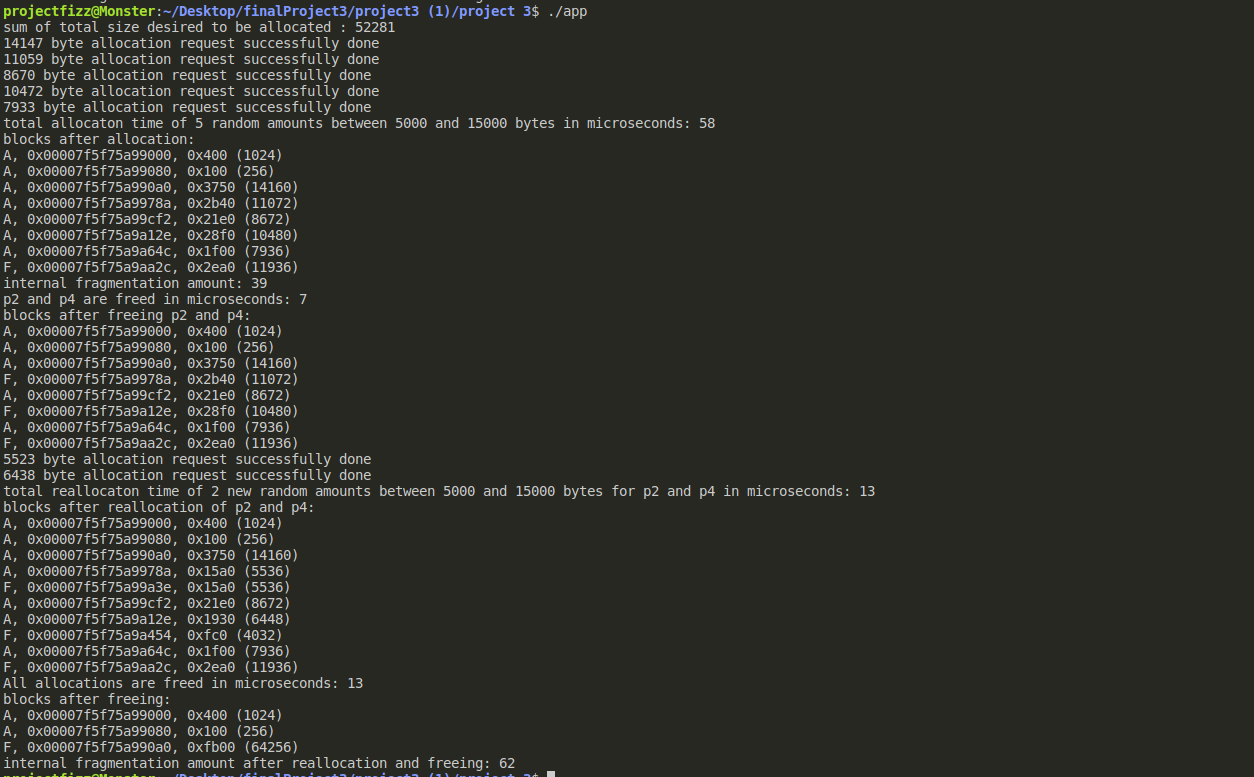


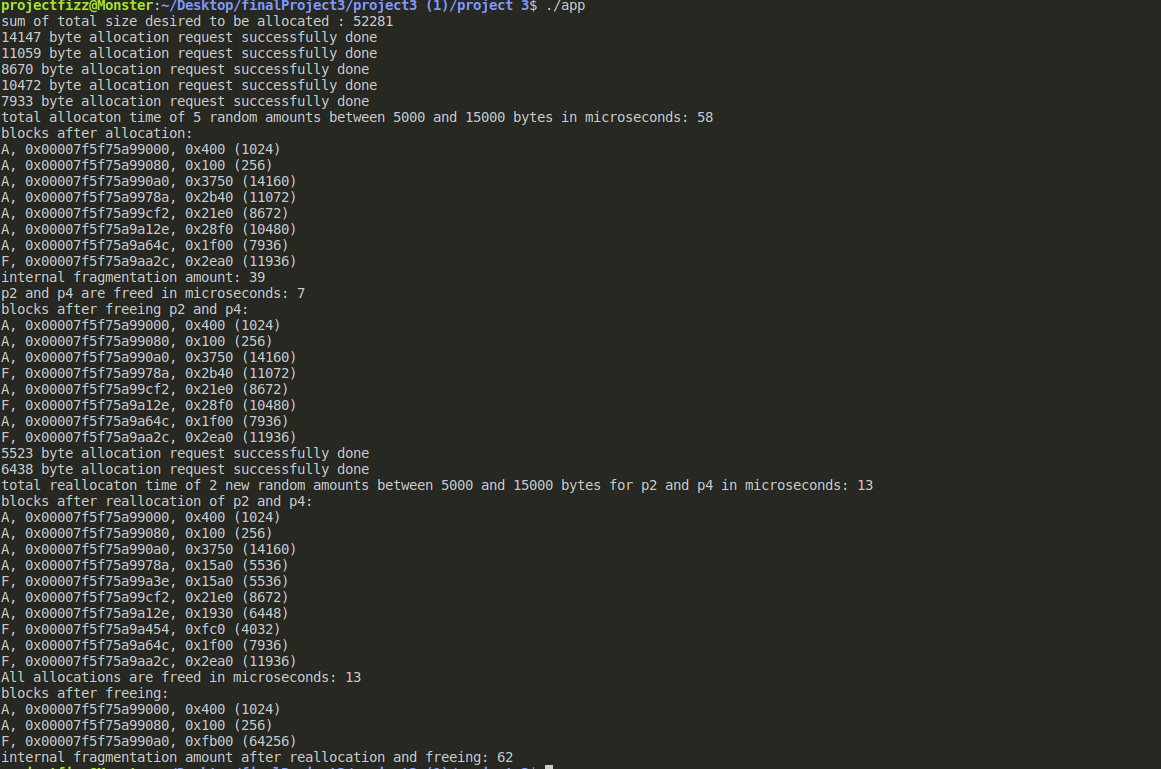
First 5 random allocation performed, 2 of them are freed, 2 new random allocation performed and all of them are freed.











Results

In results of experiment A, we can see that allocation time increases when allocated size increases. It is obvious since allocation is performed by O(n) time complexity and bits are traversed to find a place to allocate according to first fit. Also, we can see that for the same allocation sizes, when we increase the segment size, allocation time increases. This happens because library looks for a free place from the beginning and when we increase segment size, we indirectly increase bitmap size too. Therefore, traverse of a larger bitmap increases the allocated time. Similar results can be seen in graphs for freeing time too. There are fluctuations but they are not frequent to disrupt the result.

In results of experiment B, allocation time increases but not in a specific proportion. Reason of it is the different sizes of different allocations. Because of this difference, program spends various amounts of time for each allocation. In general, allocation size determines the time but time values could be different with same total size but different allocation sizes for each allocation.

The experiment with sequences of allocations and freeing some of these allocations is conducted to test the functionalities of the library. We also conducted similar tests with app.c program and you can see a simple test case in our submitted file.

We added screenshots of our terminal outputs for detailed results.