Date: 04/12/2021

Class: CS5541

Assignment: Memory Allocation Lab

Author: Matt Kennedy

Email: mfj0222@wmich.edu

General info:

For this program, I decided to use Python. My version is 3.8.2, but this should work for most versions of Python 3.

I used the Google Style Python Docstrings, found here: https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_google.html

All algorithms/code are mine.

I used both vim on ubuntu 20.04 LTS, as well as visual studio code on Windows 10 to run this. It ran fine on both.

If the help flag is specified, then the code will exit after printing help. This is done because the example cache simulator provided does the same thing.

Running:

There are 2 different ways this can be run, both of which vary depending on the python command you use on your system.

The first is just like a compiled executable, with the following format:

./mem_allocation_sim.py [-h] -f <filepath> -l <IE> -a <FB>

-h: Optional help flag that prints usage info then exits

-f <input text file>: Path to the input file to read from

-l <IE>: Free list type. Either I is specified, for Implicit, or E is specified, for Explicit

-a <FB>: Allocation type. Either F is specified, for First-fit, or B is specified, for Best-fit

If this method provides an error, this will be due to the shebang, which is the first line of code. The shebang points to your system's python 3 location.

On your system, you may need to remove the very last character and change it to just "python" instead of "python3". Or use the second method of execution.

The other way to run this is using the python command. For example, on my system, it is:

python3 mem_allocation_sim.py [-h] -f <filepath> -l <IE> -a <FB>

If you have the python3 command on your system, you shouldn't have issues. If you instead just use python, use that instead.

Some notes on this program -

The approach I took for my heap was different than what was provided in the output, as I had this done before examples were provided online. The differences are applied with coalescing.

What I did with coalescing was as follows - I organized all free blocks around the used blocks, and if any free blocks were adjacent, they were combined.

After blocks were organized, if the last block was a free block, the I called sbrk to reduce the size of the heap to what was needed so there was none extra.

This is different than the output provided.

Additionally, the format of the output.txt file is different. I stored headers/footers in the same way that was described in the lecture notes video, where the header and footer are written, where the first bit is 0 or 1 for not free or free, respectively, then the rest of the bits are the size of the payload.

This is vastly different than output provided. However, all myalloc, realloc, free, and sbrk calls work appropriately, and provide a returned pointer per the rules provided.

The way the output file is generated is by going through each heap address, 0 to a max of 100,000, and providing relevant header/footer info.

As an example of the way my output file works, I will walk through example 2.in, line by line.

For these examples, I will use the implicit list, with first fit.

a, 1000, 0

The program will allocate a block of size 250 bytes (4 bytes in a word). It will do this with the header at address 1, to follow double-word alignment.

f, 0

This block is free'd. The heap is now empty again.

a, 1, 1

The program will allocate a block of size 1. It will do this with the header at address 1, the payload at word 2, and the footer at word 3.

a, 1, 2

The program will allocate a block of size 1. It will do this with the header at address 5 so the payload follows double word alignment, the payload at word 6, and the footer at word 7.

The output.txt file will look as follows:

0,

1, 0x00000010

2,

3, 0x00000010

4,

5, 0x00000010

6,

7, 0x00000010

Another example I will walk through is 3.in, line by line:

a, 5, 0

The program will allocate a block of size 2. It will do this with the header at address 1, the payload at words 2-3, and the footer at word 4.

a, 5, 1

The program will allocate a block of size 2. It will do this with the header at address 5, the payload at words 6-7, and the footer at word 8.

f, 0

The programm will free block 0. Because coalescing a free block does not require double word alignment anywhere, the free block has header at word 0, and footer at word 4.

f, 1

The programm will free block 1. Since the block directly adjacent is also free, the program will combine them into one free block. Since this is the only used block, the program will not sbrk down. The current only block, this free block, has header at word 0, and footer at word 8

a, 9, 2

The program will allocate a block of size 3. It will do this at the free block we currently have, with header at word 1, payload words 2-4, and footer at word 5. Additionally, the program will sbrk down all remaining free space after this block, as it is not needed currently.

a, 1, 3

The program will allocate a block of size 1. The program will create the space needed with sbrk, and set the header at block 7, the payload at word 8, then the footer at word 9.

f, 2

This program will free block 2, which will leave that block as free space. Because this is a free call, we don't have to follow double world alignment for any payload, so now there is a free block with header at word 0 and footer at word 6.

The output.txt file will look as follows:

0, 0x00000051

1,

2,

3,

4,

5,

6, 0x00000051

7, 0x00000010

8,

9, 0x00000010