CSE 320 Spring 2018 HW #2

February 19, 2018

Deadline: March 4th, 23:59 Stony Brook time (Eastern Time Zone).

1 Introduction

Goal of this homework is to help you better understand how memory allocation works and give you a better understanding of pointers. On top of that, you will also learn various tools such as Makefile and gdb. This homework consists of two parts : In the first part you are required to read the provided material that is crucial for doing the second part. The second part of the homework is about implementation of defragmentation tool but for the main memory instead of hard disk space.

2 Part II

You are required to read and/or walk through the provided material. First, it will help you a lot during implementation of the second part. Second, part of it may appear on the exams.

2.1 Starting point

You need to read the Sections 9.9 through 9.11 of Chapter 9 from the textbook Computer Systems: A programmer's Perspective by Bryant.

2.2 Code hand-out

You are given an object file that contains several functions that you need to use to implement your program (more details in the GitHub instruction).

You will start by calling the function `cse320_init(char* filename)`, in the skeleton code provided by us, that will load test file into the emulated memory and will return you a pointer to the start of it. Then you will call `cse320_tmp_buffer_init()` that will return you a pointer to a temporary buffer that you may want to use as a temporary storage. Size of the emulated memory

is 1024 Bytes and maximum size of the temporary buffer is also 1024 bytes. However, initially the temporary buffer is limited to 128 Bytes. To increase its size you need to call `cse320_sbrk(ssize_t size)` that will increase its size by moving the break point. If the function is called with `size` equals zero, then it returns current break point.

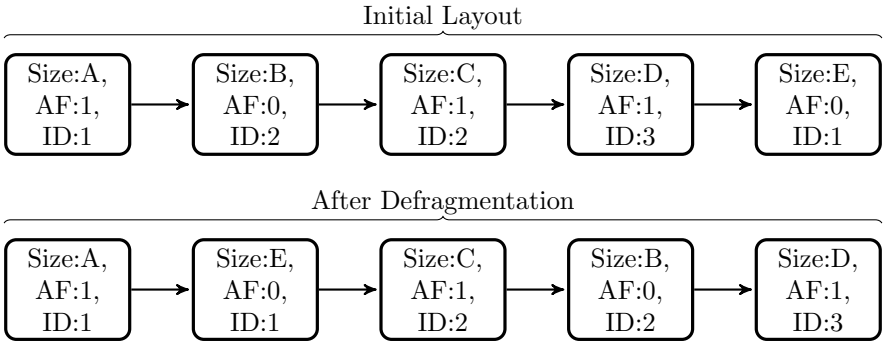
After you have implemented the above, you will call `cse320_check(char* filename)` that will check your solution and then you will call `cse320_free()` that will free temporary buffer and emulated memory.

2.3 Coding task

Function `cse320_init(char* filename)` will load test file into emulated memory. Each memory block consists of header, footer, payload, and possibly padding. Size of the header and footer is one word (8 Bytes). Each block should be double word aligned. You can find structure of header/footer below:

- Lowest bit represents *allocation flag* that is 0 when block is free and 1 when block is allocated
- Next two lowest bits represent *application ID* to distinguish between different applications (more details further)
- Rest of the bits represent size of the block in a similar way as was discussed in lecture (recall, that 8 Bytes always have three lowest bits equal to zero)

Your task is to perform *defragmentation*. Each application may have its memory blocks spread over the memory but we want them to be located together. That said, you need to put blocks of each application next to each other. Look at the example that follows:



Thus, you can see that you need to put memory blocks of each application next to each other.

It will be easier (and better from the learning perspective) if you will treat your temporary buffer as an implicit list that starts with one free block of size 128 Bytes and application ID zero. Which policy to use (first-fit, next-fit, best-fit, etc) is up to you.

2.4 Handling Errors

You should check if `cse320_sbrk(ssize_t size)` returns an null pointer. Then you should print to the console **SBRK_ERROR** and exit your program with return value matching error number returned by the `cse320_sbrk(ssize_t size)` function. Same applies to *cse320_init(char* filename)* function except error text to print is **INIT_ERROR**. In general, for this assignment if there is an error your program should exit and return value matching corresponding error number.

2.5 Constraints and Specification

You should adhere to the following constraints:

- You cannot use `malloc()` and `mmap()` functions.
- Memory blocks corresponding to the application with lowest ID go first (in the ascending order from left to right, where the leftmost node is the beginning of your list)
- Allocated blocks should be located before free blocks
- Blocks with lowest size go first (In the example above, size of A<E<C<B<D)
- If there will be adjacent free blocks corresponding to the same application, they should be coalesced
- You can assume that there are no two blocks with exact same size, application ID, and allocation flag
- You should add block with size of 16 Bytes, application ID equal zero, and allocation flag equal zero as a last block to your list and emulated memory (not shown in example)
- If application ID of a particular block is not zero then it size should be minimum 32 Bytes
- You application should be called **defrag_tool**
- We should be able to compile your code by typing `make`

2.6 Writing your own test cases

Initially you will be given four test cases. It will be easy to see if you handle them correctly. However, you are free to construct your own test files. To do such we provide you a simple test builder called `test_builder` with interactive interface in your repository. After answering few questions it will create a test file for you, which you can use to test your code.

Please note that it is very simple test builder and does not perform checks for incorrect inputs from user.

2.7 Extra Credit

Stay tuned. Will be released soon.

2.8 GitHub Part

To start please follow the *link https://classroom.github.com/a/4aY_yZMP*

I think it is obvious but do not share course materials outside of class.

2.9 Submission

As with homework 1, we will grade your latest push taking into consideration your late days, if any. Please make sure that your code can pass tests in Travis CI (which also means that it should compile there).

2.10 Useful reading

Below you can find some useful readings

http://mrbook.org/blog/tutorials/make/

http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/

https://www.tutorialspoint.com/makefile/index.htm

https://www.gnu.org/software/libc/manual/html_node/Environment-Access.html#Environment-Access

https://askubuntu.com/questions/58814/how-do-i-add-environment-variables

https://en.wikipedia.org/wiki/Include_guard