Mini-Project 2: Memory Allocation

CS 4410: Operating Systems

Fall 2015

1 Introduction

For MP2, you will implement an interface that mimics dynamic memory allocation. Throughout this assignment, you will design your own data structure, implement the given function headers, and use pointer arithmetic to manage memory.

Along with a makefile, you will be given two files: alloc.c and alloc.h. You are required to implement all of the functions specified in alloc.c, and you cannot change the given function signatures. For documentation of the given functions, see alloc.h. However, you are free to add any data structures or helper functions you feel would be helpful.

2 Description

You may notice a common argument in the given function headers: `void *memarea`. In this assignment, you do not need to worry about getting a chunk of memory from somewhere. Instead, `void *memarea` will point to the total "memory area" whose memory you will be allocating.

So, how do we organize this memory and ensure we can re-use any memory that has been freed? You will need some way to keep track of each chunk of memory that has currently been allocated. A common way of doing this is to place a "header" in front of each chunk of allocated memory with important information about that memory block.

A primitive implementation of such a data structure could be as follows:

```
struct m_header {
    unsigned int size;
};
```

Your memory area would then resemble a sequence of alternating headers and blocks. Keep in mind that blocks of memory can be of size 0. The addresses of blocks should be "aligned" at multiples of 8 bytes.

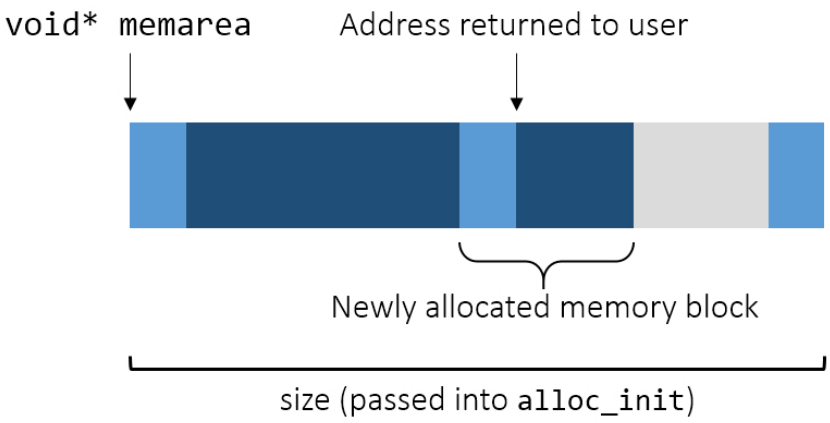

Figure 1: Example of a possible layout of `void *memarea` when allocating a chunk of memory. The light blue sections represent the headers, and the dark blue sections are the allocated chunks of memory. Free memory is colored with gray.

To navigate through these pointers, you will want to use pointer arithmetic. As a basic example, see the following snippet of code:

```
struct m_header *mh = memarea;
return (void*) (mh + 1);
```

As the type of `mh` is `struct m_header*`, adding 1 returns a pointer that is `sizeof(struct m_header)` bytes past the pointer `mh`. You should become familiar with pointer arithmetic if you are not already.

3 Testing

We have provided a few simple test files to find any large bugs in your implementation and help you better understand the functions in this assignment. However, keep in mind that these tests are in no way intensive, and you should try to create additional tests for your code. (You do not have to turn in any tests you make.)

Also, you may find `gdb`, the GNU debugger, helpful while completing this assignment. (Tutorials can be found on `http://www.unknownroad.com/rtfm/gdbtut/gdbtoc.html`)

4 Evaluation

Your implementation will primarily be graded on its correctness. It should be able to allocate memory of a given size, if there exists sufficient *contiguous* memory in `void *memarea` to accommodate the request. Additionally, released memory should be able to be allocated again for reuse.