CSUS, College of Engineering and Computer Science
Department of Computer Science
CSC/CPE 139 – Operating System

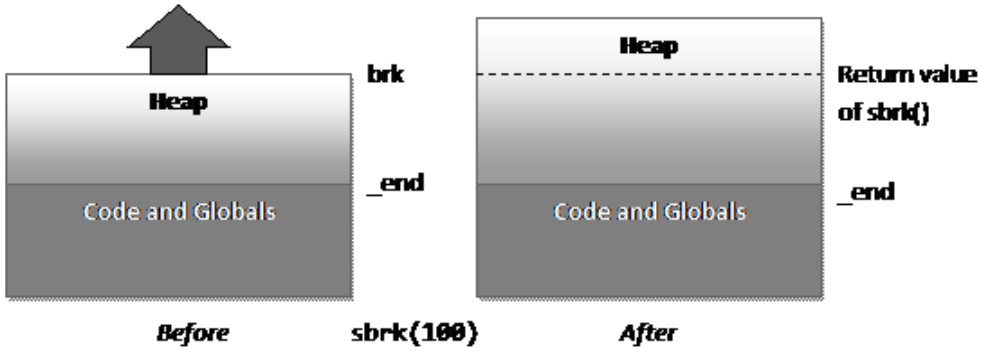
Programming Assignment 4 – Your Implementation of Malloc() and Free()

Goal: In this project, you will implement your own version of the standard C library calls malloc() and free().

Acknowledgements: Significant portions of this project have been adapted from a project given by Prof. Urgaonkar of Penn State University.

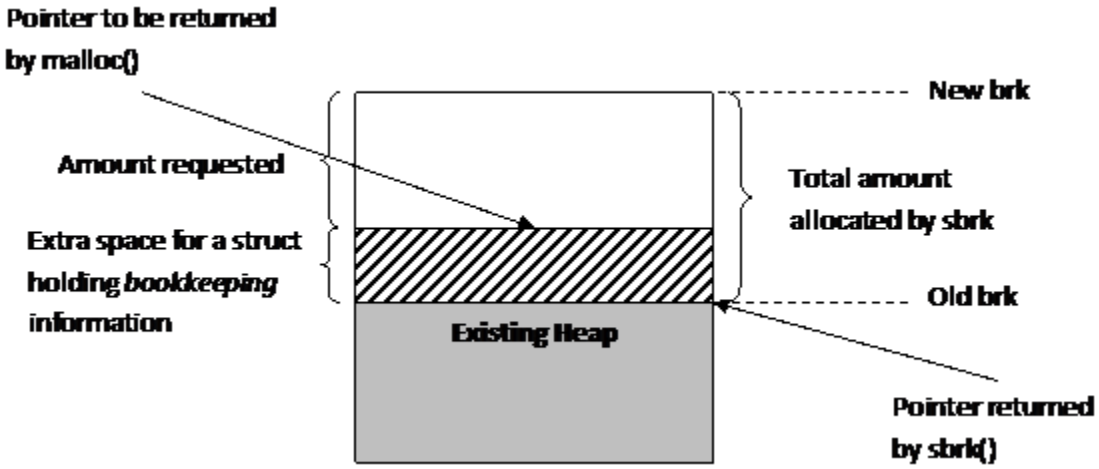
Hardware/Software: You need access to a machine with Intel-based CPU, running UNIX/Linux, and loaded with C-related software (see below). It is tested that virtual machines with Linux environment is OK to run. You can reuse the virtual machine image from the textbook.

Instructions: Although your implementation will be in user-space, it will help you understand some basic concepts about memory management within an operating system. Take a look at the figure below which shows how malloc designates a region of a process's address space from the symbol _end (where the code and global data ends) to brk as the heap. (brk stands for "program break" in the C nomenclature, think of it as the "end" of the heap.)



A process can grow or shrink the size of its heap by setting new values of brk. The function sbrk() handles changing the brk value based on the parameter it is passed. Check out how sbrk() works using its man pages as well as material about in covered in class. The figure above shows the effect of calling sbrk(100), which increments the size of the process's heap by 100 Bytes. (Make sure you also think about how the OS can only add memory to a process's address space in increments of "pages". What do you think happens when a process accesses a portion of a page that it has not malloced but which has been allocated to the process by the OS? This will help refine your understanding of segmentation faults.)

Details: As part of dynamic memory management, we will discuss various algorithms for the management of the empty spaces that may be created after a malloc()-managed heap has had some of its allocations freed. Your version of malloc will employ the best-fit algorithm. It is not enough to write a malloc that simply increments brk by the amount requested and returns the old value of brk as the pointer. This is because when you want to write a free() function, the parameter to free() is just a pointer to the start of the region, so you have no way to determine how much space to deallocate. In order to know what space is free or used, and how big each region is, you will use linked lists. Since we also need some place to store this dynamic list of free and occupied memory regions inside of the heap, just allocating some fixed-size region when doing malloc may not be adequate for how many nodes in the list we would need to create. A better idea is illustrated in the figure below:



We can add some additional space to each update of brk in order to accommodate a structure that is a node in our linked list, and this structure can contain useful things like:

- The size of this chunk of memory
- Whether it is free or empty
- A pointer to the next node
- A pointer to the previous node

We then return back a pointer that is in the middle of the chunk we allocated, and thus the program calling malloc() will never notice the additional structure. However, when we get a pointer back to free, we can simply look at the memory before it for the structure that we wrote there with the information we need.

Requirements: You are to create two functions for this project.

1. A malloc() replacement called void *bestfit_malloc(int size) that allocates memory using the best-fit algorithm. Again, if no empty space is big enough, allocate more via sbrk(). A free() called void bestfit_free(void *ptr) that deallocates a pointer that was originally allocated by the malloc you wrote above.
2. Your free function should coalesce adjacent free blocks. If the block that touches brk is free, you should use sbrk() with a negative offset to reduce the size of the heap.

As you are working on this project, a sample driver program is provided to help you test that your code works or not.

Deliverable: Submit the following through Canvas, in a **tar-ball or zip file**. Please submit only an **electronic version** to **Canvas**.

1. A header file named bestfit_malloc.h with the implementations of your two functions, and the test program you used during your initial testing. All source files including .c, .h and .o files should be all included in the tar-ball.
2. A report containing source code and necessary screenshots.