UNIVERSITY OF LEEDS

XV6 Memory Management Written Report

Aodhan Gallagher
201450577
University of Leeds

Design Decisions

The solution is composed of two files: memory_management.c and memory management.h.

The header file (memory_management.h) is used to contain a number of declarations from function prototypes to macro definitions. It is also used to define our structure / linked list which is used extensively in the source file.

The source file (memory_management.c) houses the main code used for malloc and free to function. Two additional functions have also been defined here. One which finds free space in the heap and one that requests more space from the OS if there is no sufficient space in the heap. These functions are called multiple times in the malloc function and have been implemented to reduce the number of lines of code and thus make it more readable and efficient.

Implementation

When implementing malloc and free, a basic understanding of the heap was used with a head that contained meta information about the block of memory, followed by the actual block of memory used to store data. This is shown in Figure 1 below, but it should be noted that a magic number was not used in this solution.

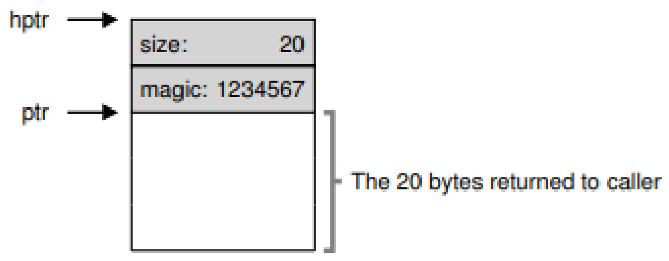

Figure 1, Allocated region of space plus header with specific contents
(Arpaci-Dusseau, 2015, p.157)

The header file contains a link list (struct) that is used to store information about each block of memory including the size of the block, a pointer to the next block of memory and whether the block is free or not.
The size of the structure is also given a definition as it is used repeatedly throughout the source file. Function prototypes are also given here, as is standard in C programming.

The source file begins with a void pointer that will be used to initialise and act as a head for the linked list:

```
void *globalHead = 0;
```

The function ‘getBlock()’ is used to iterate through the linked list to check if there’s a block that is both free and also large enough to allocate the inputted amount of bytes. The first block to satisfy these conditions (if it exists) is returned.

The function ‘getSpace()’ is used to request space from the OS and add that new block of space at the end of the linked list. It achieves this through use of the sbrk() function and when the new block is created, it is initialised within the function.

In my implementation of malloc, if the requested size is 0 (or less), a unique pointer value is returned (so that it can be passed to free).
If globalHead is 0 (and thus malloc has not yet been called), getSpace() is used to request space from the OS and store it in a new block pointed to by memBlock. globalHead is also set to point to this block too.
If globalHead is not null (and thus malloc has been called previously), getBlock() is called to see if there’s any available space that can be used. If there is, then getBlock() successfully re-uses space from the ‘heap’, otherwise getSpace() is called to request space from the OS.
memBlock->free is then set to 0 to indicate that the block is currently not free.
A void pointer that points to the start of the allocated memory is returned.
In the code, ‘memBlock + 1’ is used to point to the region of memory after the head, which is the block itself.

In my implementation of free, if the pointer value is null (0 or less), then no operation is performed and the function returns (with no return value).
Otherwise, the address of the struct is assigned to a pointer. That pointer is then used in error checking to ensure the block is not free, by using ‘blockPtr->free != 0’.
If the block is indeed not free, then it is ‘made free’ by setting ‘blockPtr->free = 1’.

Reflection

In implementing this solution, I learned a lot about how to handle pointers as well as pointer arithmetic in C. Furthermore, in reading and preparing for this coursework, I gained a good understanding of how memory is managed in an operating system and how the heap is used in memory management.

In terms of what went well, I would say that I developed a better understanding of pointers (which has proven to be an integral part of programming in C) and this has allowed me to implement a basic version of malloc and free.

Given more time, I would have liked to have implemented a more efficient way to allocate memory such as going for a method of best fit rather than first fit. I also would like to have implemented a way to coalesce the memory to make it more space efficient. In the future, I would like to manage my time more effectively so that I could add these features and complete the solution to a higher standard.

References

Arpaci-Dusseau, A and Arpaci-Dusseau, R. 2015. *Operating Systems: Three Easy Pieces*. [No place]: Arpaci-Dusseau Books.

Silberschatz, A et al. 2018. *Operating System Concepts*. Tenth Edition. New York: John Wiley & Sons, Inc.

Luu, D. 2014. *Dan Luu's website*. [Online]. [Accessed 25 November 2022]. Available from: https://danluu.com/