Next, previous, before and after

Daniel Gustafsson

2020-01-13

1 Introduction

Everyone who has ever tried programming in C should have heard of "malloc" and "free", these two functions provide a way for the programmer to easily allocate memory on the heap of a process. Modern day memory management is fast and efficient, but what acctually goes on under the hood of the operating system when a process calls "malloc"? In this report I will create my own heap manager similar to "malloc" and explore different aspects approaches, can i make it really space-efficient? Will it have a speed payoff? Continue reading and you will find answers.

2 Background

To create your own heap manager, you first need to understand how a heap manager works. The traditional way is to have a linked list data-structure where you keep all the free blocks from your heap, then when the process wants to allocate something the manager finds a free block with appropriate size and returns the virtual memory address. Later when (hopefully) "free" is called by the process, the manager will put the newly freed block back in the list of free blocks. To make this possible we need to implement a couple things. Firstly we need to get memory from the operating system , we can do this with the "mmap" system call, we should get as much memory as we think the process will use right from the start because calling the system to get more all the time would be very performance costly. Now we need a header with some information to be stored in every block on the heap. To start, the header should contain the following information,

- Free - if the block is free or taken
- Size - the size of the block
- Bfree - if the block before is free or taken
- Bsize - the size of the block before
- Next - pointer to the next block in the list
- Prev - pointer to the previous block in the list

When a block is requested, the manager needs to make sure the block is of appropriatesize, neither too small nor too big. A block that is too small would quickly result in a segmentation fault, if the manager keeps giving out

blocks that are too big the memory will run out (internal fragmentation). To solve this the manager tries always looks for a block that is bigger than the one requested and then if it is too big split it into two smaller blocks. This creates a new problem, after a while all blocks will be really small becuase of the splitting (external fragmentation) to solve this the manager will try to merge neighboring blocks when they are freed. This is the reason for each block to contain information about the block before it, to get the block after the current one in memory, just check the block that is 'sizeof(header) + size' after it. To get the block before, check the address that is '-(sizeof(header) + Bsize)' before it.

3 Method

I focused my improvements on lowering overhead, I did this using 3 different improvement methods.

The first improvement I made on lowering overhead was removing the "Prev"-pointer, this would lower overhead by 8 bytes therefore making it a big first improvement. The tradeof of this is that the Free-list is no longer a double linked list but a single linked list, this has performance drawbacks on "detach" and "insert" operations on the list. When detaching, the manager now have to change just one pointer, this pointer is the "Next"-pointer of the previous item on the free-list, this is problematic because the manager has no easy way of finding the previous item like it did before. The manager now has to go through the whole list to find the item and this can become performance-costly when the list is long.

The second improvement I made was reducing the "Next"-pointer from 8 to 4 bytes. I achieved this by removing the 4 most significant bytes of the pointer, making it a 32bit int instead. This is possible because I know that this heap will never grow beyond a 32bit address-space and therefore will always have teh same 4 most significant bytes. The implementation of this is quite simple, two new functions are introduced, ADRS and NEXT. ADRS returns a unsigned 32bit integer, this integer is calculated using by subtracting the address to the arena (the start of the heap-space) from a block. This will be the 32bit (4 byte) address of the block and is what will be used in another blocks "Next"-pointer. The NEXT function returns a pointer to a head structure, similarly this pointer address is calculated by

taken the unsigned 32bit int from ADRS and adding the address to the arena.

This second improvement can also have significant performance trade-ofs, especially in combination with the first improvement. Now each time the manager want to traverse the free-list, it needs to convert every "pointer"(32bit unsigned int) to the real address with NEXT function. This isn't a big problem when the list is short.

The third and final improvement is the removal of "Free" and "Bfree".

4 Results

this is the results