OS-LAB-2

Erik Rehn

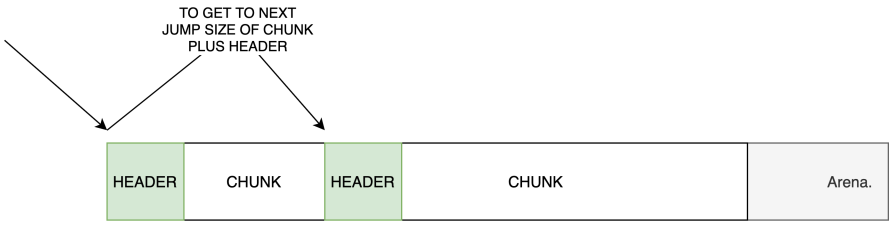
November 2020

1 Introduction

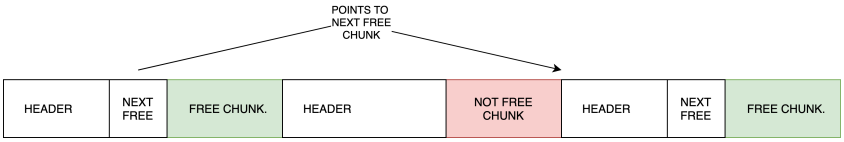
The purpose of the was to create malloc and free in C. Malloc is a utility function which allows you to allocate memory base and Free allows you to clear up the allocated memory.

2 Our implementation

Our implementation of the code consists of a sys call which allocates a arena of memory. In our case 64 * 1024 bytes. The arena is the used to store the allocated memory chunks. Chunks are stored sequentially and are appended after eachohter and a head is used to keep track on the allocated size. The size of the current chunk can be used to get the position of the next chunk. Hence we only need to keep track of where the first chunk in the arena is is.



On each head we also information to make it possible to insert and remove chunks. Such as before free and if the current chunk is free. To keep track on which block are currently free we store that information in a linked list.



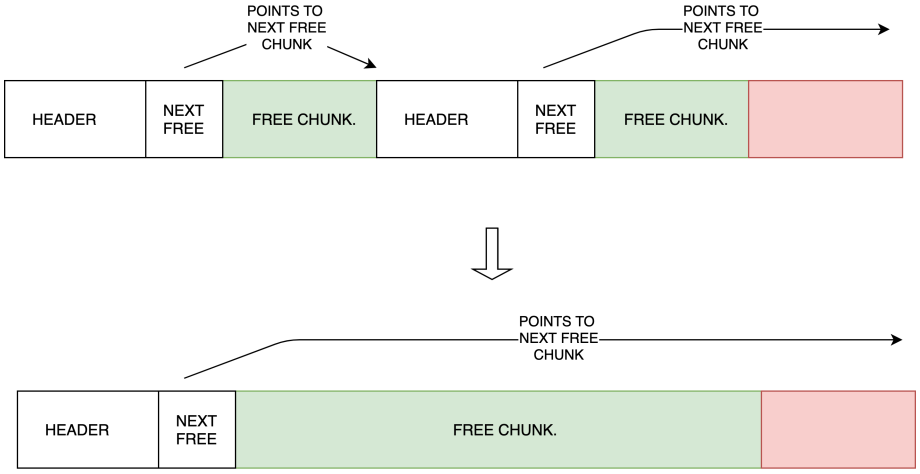
Base on this we are able to both loop threw all the once which are free and we are able to free up allocated using its current size. The issue with this implementation is that this fractures up the memory in the free list. If we take for example the following code.

```
1
2  int main()
3  {
4
5      int i;
6      void *ptrs[ARENA / 16];
7
8      for (i = 0; i < ARENA / 16; i++)
9      {
10         ptrs[i] = malloc(16);
11     }
12
13     for (i = 0; i < ARENA / 16; i++)
14     {
15         free(ptrs[i]);
16     }
17
18     return 0;
19 }
```

Listing 1: Example of code which would fracture the arena

This code would create 16 bytes chunks which would fill up the arena. Even after we have free all of our chunks the sizes would remain the same and a request for a 32 bytes chunk would now not able to be fullfilled even though the arena is compelety free.

To resolve this we implemented a merging mechanism. Which in the free list merges two chunks into one if the are after eachother. This is actually really easy to done with only a couple of lines of code. All we have to do is check when freeing a block if the one before or after is free. Detach the merged block and then point from our current to the one which we have merged with.



Now looking at the code sited in *Listing 1*, we are not able to allocated and free and get back to where we started, which allows us to allocated as much as we have in our free list.

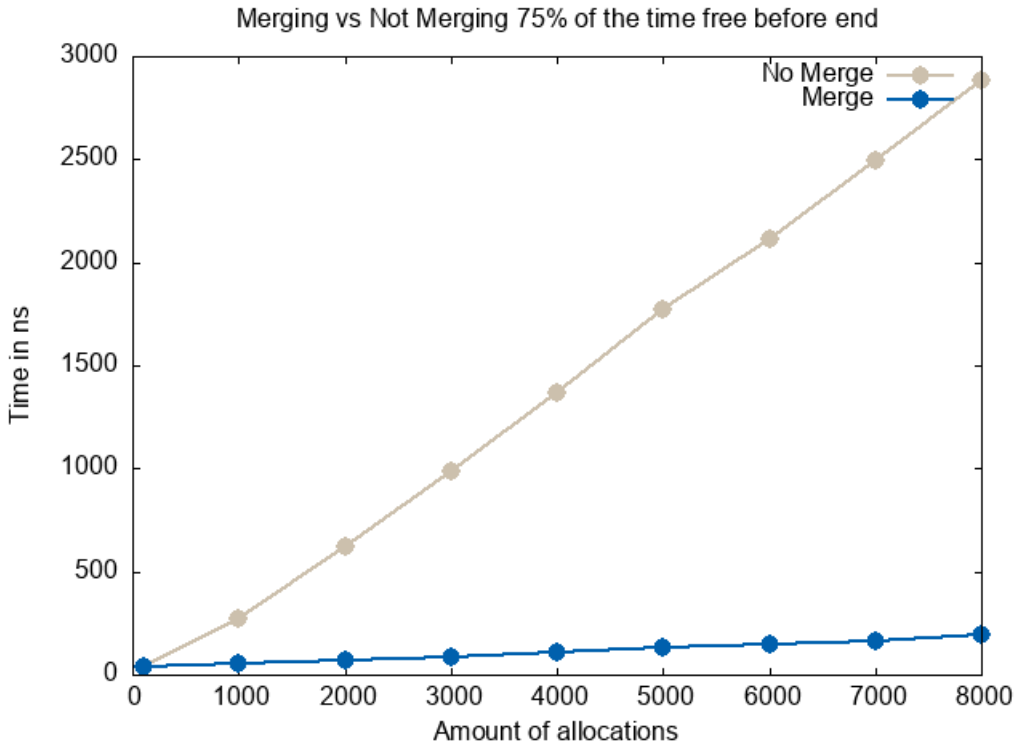
Another aspect is that the time it takes to allocate a block is dependent on how long the free list is. If it is a long free list we are going to have a greater time complexity. Therefor will merging block create a smaller list which in turns makes the programm faster.

3 Some issues we ran into

What we had the most issues with was the initialization of the arena. We wrongly configured the sentinel which gave us undefined behaiviors. Our sentinel at first was not configured to be not free which made the merging logic not work as expected. After that we did not insert the arena into the free section which gave us issues when trying to make any allocations at all. Besides those two mishaps we acutally were in pretty good shape. After the initliatization we were able to get threw the rest of the program without any real issues.

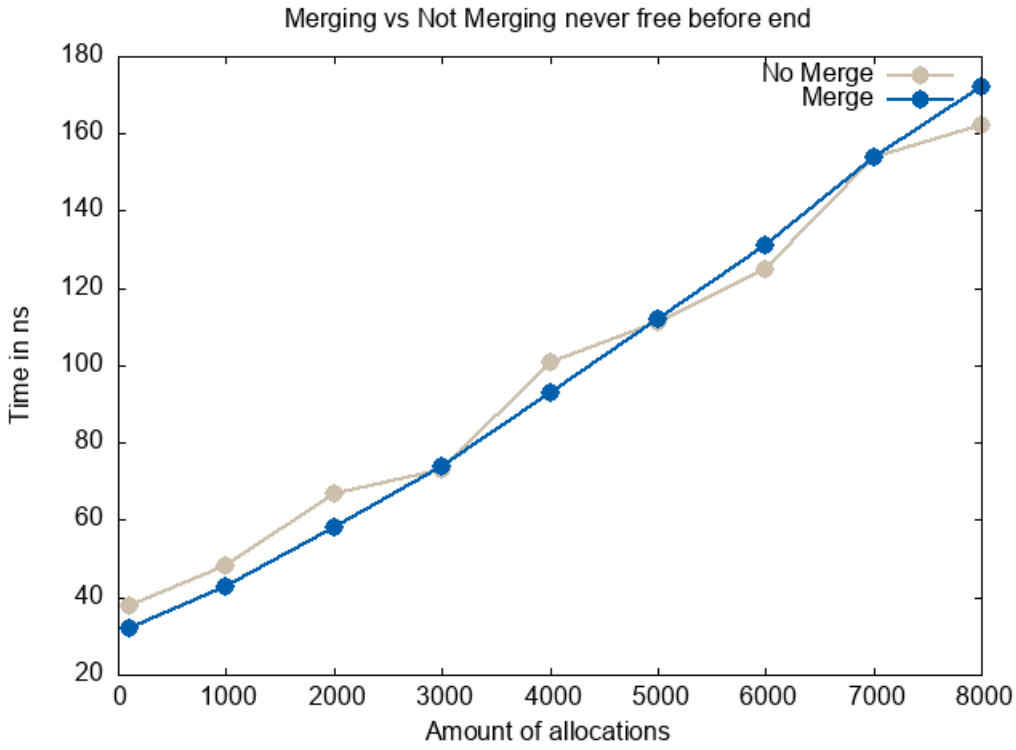
4 Tests

To test to see if our merging had an effect we wrote a small program which allocated chunks between 8 to 4096 bytes and freed them. To simulate the randomness of softare we also at random freed them directly after we had 75% of the requests be reliized directly after and 25% where released after the entire program.



We could explain this graph by thinking about what is going on when we merge two. Since we merge whether we are able to the length of the list will be at least cut in half!

One might wonder what would happen if we only free after we have allocated the entire list and then freed them in order. Then we would not ever be able to preform any merges and the result should look about the same for both when merging and without merging.



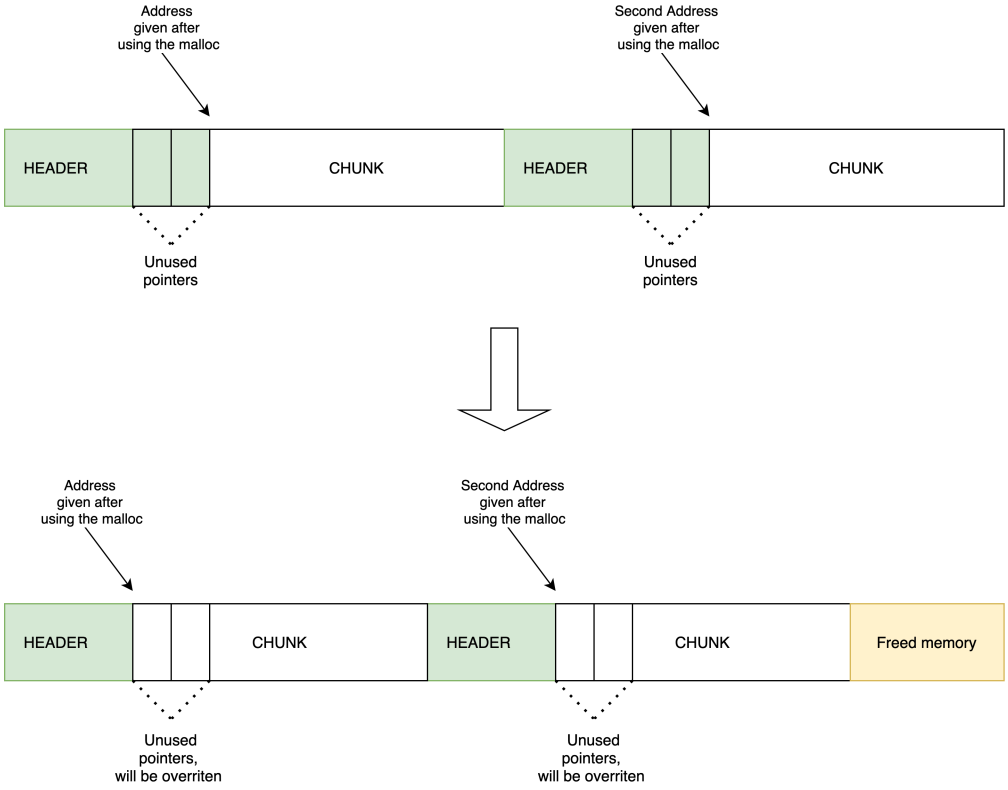
As we can see then the merges have little to not effect!

5 Improvements

For improvements we choose to remove the linked list pointers to the not free data chunks. We did this by making two real changes.

1. When allocating a memory chun return an adress which is offseted back the size of two head pointers. This will make it so that programm will override the pointers with usable data.

2. When using *after* and *before* we need to take into account if it is a free block or a allocated block.



This cut the size of a taken block from 32 bytes to 16 bytes. Since we use merge logic only on the free blocks we are more likely to have more taken blocks than freed block making this improvement very important. However this does not any notable effect on time complexity since this does not affect the length of the freed list and the look up from the previouse is still in *O(1)*.

6 Source code

https://github.com/erikschmutz/ID1206/