# Operating Systems - Seminar 2: Malloc

Oisin Michael Farrell

November 2020

## 1 Introduction

The goal of this seminar was to understand and create our own version of the built in malloc() and free() methods. For simplicity, they were named dalloc() and dfree(), respectively. In this report I will outline the problems that I encountered and the solutions used to overcome these problems.

## 2 Allocating from the end.

The main problem that I faced was when I ran the program and allocated some things to memory it seemed to me that the location that these blocks were stored outside of the arena, which I thought was incorrect at the time. When I ran the program I got the following result.

```
Allocating: Size: (8) Location: (0x7f2c3e9d4fc8)
Allocating: Size: (24) Location: (0x7f2c3e9d4f98)
Allocating: Size: (32) Location: (0x7f2c3e9d4f60)
Sanity 1: Check complete
--------Sanity Check--------
Flist Pointer: 0x7f2c3e9c5000
Prev: (nil) Next: (nil)
Size: 65352
---sanity2 check complete---
```

As you can see, the locations of the three allocated blocks are far greater than the pointer to arena, which led me to believe that they were outside the arena as I assumed that the blocks would be allocated at the start of the arena and grow upward.

Furthermore, the size of the leftover chunk of memory is 65352 bits, which at the time I thought was all the memory available. Then I realised that the initial size of memory was actually 65536 bits. When I discovered that this was the case I realised that they were not being allocated outside of the arena, just at the end.

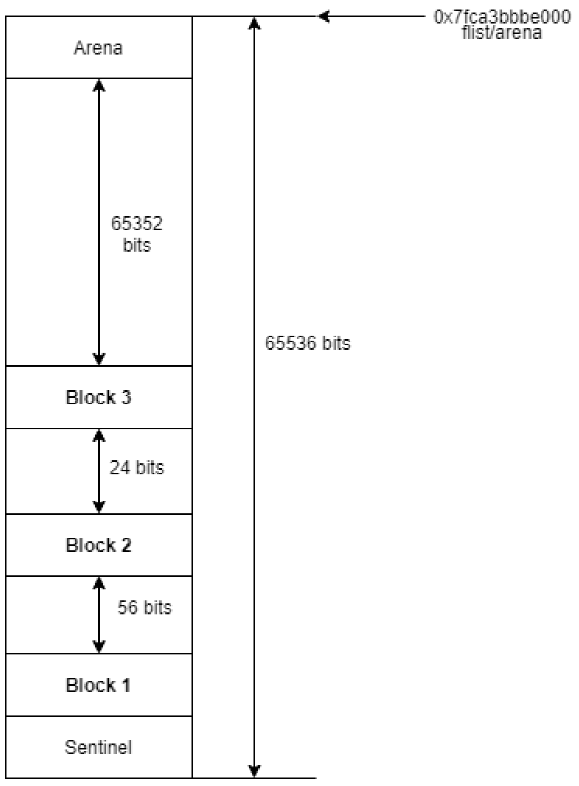

Figure 1: Block Allocation

1
The image above shows a diagram of implementation for the project. The blocks are allocated at the end of the arena and work backwards towards the start.

## 3 The working model.

When I realised that my implementation of malloc was working for a few allocations it was time for me to test the system more vigorously to make sure that it did in fact work and that it would work under all conditions.
As you seen above in 1 the system can allocate three blocks, but now I want to test if my program can free a block. A good way to test this would be to free the block two, as the block before and after are not free which meaning it will not be able to merge, creating two free blocks in flist which will be linked together. Using my sanity method it was easy to see that these two blocks were in fact linked.

```
--------Sanity Check--------
Flist Pointer: 0x7f2c3e9d4f98
Prev: (nil) Next: 0x7f2c3e9c5000
Size: 24
---sanity2 check complete---


--------Sanity Check--------
Flist Pointer: 0x7f2c3e9c5000
Prev: 0x7f2c3e9d4f98 Next: (nil)
Size: 65352
---sanity2 check complete---
```

Above is the result of running the sanity check on the free list. As you can see there are two blocks in the list, one of size 24 bits and the other 65352 bits. You will notice that they are linked. This can be seen as they are both pointing to each other.

Now we will free block 1. This will cause block one and block two to merge as they are both free but block three will prevent any further merging as it is not free. This should hopefully result in flist containing two blocks, the old block of size 65352 bits and the newly merged block.

```
--------Sanity Check--------
Flist Pointer: 0x7f2c3e9d4f98
Prev: (nil) Next: 0x7f2c3e9c5000
Size: 56
---sanity2 check complete---


--------Sanity Check--------
Flist Pointer: 0x7f2c3e9c5000
Prev: 0x7f2c3e9d4f98 Next: (nil)
Size: 65352
---sanity2 check complete---
```

As you can see by freeing block 1 (8 bits + 24 bit header) and adding it to the previous 24 bits we now have one larger 56 bit block which is linked to the large free block.

Now that we know my system can free and allocate memory we can begin to look at some improvements.

## 4 Improvements

The improvement that I will be looking at is in relation to having multiple lists based on free block size.

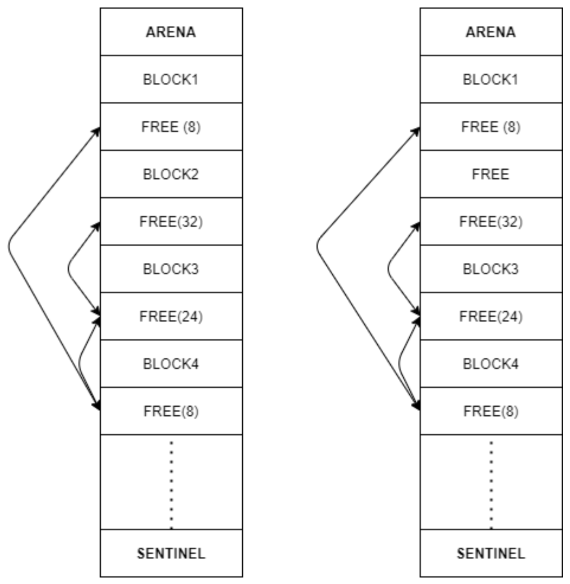

Figure 2: My understanding of ordering flist based on size

Above is a visualisation of what I imagine the system to look like if free blocks are linked by size. On the right, you can see that block two has been removed but the two free blocks either side can not be merged as they are not linked together, as they are different sizes. In this way, ordering the free blocks based on size will have a negative impact on the system.
However, one advantage of this implementation is that the system will rarely have to look through the whole free list to find a suitable sized block, unless it is looking for a large size towards the end of the list.
Unfortunately I was unable to implement this in time to run tests on performance so I could only provide a diagram of what I thought would happen.

## 5 Conclusion

Throughout this seminar I learned a lot. I have a far better understanding of how the malloc and free methods work and I have a greater appreciation for the c programming language.