Seminar II Report
ID1206 Operating Systems

Azer Hojlas

December 2, 2021

1 Introduction

1.1 Disclaimer

My code is partially (read substantially) based on the ID1206 github of Filip Jacobsson

1.2 General description

The purpose of the assignment is to illustrate the inner workings of the malloc() and free() library calls, additionally implementing said calls. As the real malloc() and free() are not allowed to be used, these will per assignment instructions be called dalloc() and dfree(). The following description of both malloc and free is offered by the system manual:

```
The malloc() function allocates size bytes of memory
and returns a pointer to the allocated memory.

The free() function deallocates the memory allocation
pointed to by ptr. If ptr is a NULL-pointer, no operation
is performed.
```

In the not so rare cases when the specifications of programs exceed the conveniences of the stack, the necessity for manual memory allocation arises. The assignment instructions provide a basic skeleton that the student is meant to implement, followed by the creation of a test and benchmark with the sole purpose of measuring program performance. This becomes relevant when the assignment requires that the student improves upon their code in various ways. The student will proceed to quantitatively illustrate the differences in performance (using the benchmark) once the aforementioned improvements have been implemented.

2 Implementation of dalloc() and free()

```
        void *dalloc(size_t request)
{
    if (request <= 0){
        return NULL;
    }
    int size = adjust(request);
    struct head *taken = find(size);
    if (taken == NULL)
    {
        return NULL;
    }
    else
    {
        return HIDE(taken);
    }
}
void dfree(void *memory)
{
    if (memory != NULL)
    {
        struct head *block = (struct head *)MAGIC(memory);
        block= merge(block);
        /*struct head *aft = after(block);
        block->free = TRUE;
        aft->bfree = block->free;
        insert(block);*/
    }
    return;
}
```

3 Implementation of benchmark

I have decided to use a similar benchmark to that of the mylloc assignment. The client has the option to use a block size that distributes evenly between a MAX and a MIN value, or choose a constant block size of 8 bytes. The benchmarks only functionality is its measurement of the length of the free list where available blocks of memory are stored. I found it to problematic to create a list with all individual block sizes for each run of the benchmark. Furthermore, an array of memory pointers is initially provided with capacity BUFFER (defined by client).

4 Effects of merging

Merging will for quite self-explanatory reasons (longer code, more memory accesses) take a tad bit longer to execute. What is more significant however is the improvement of memory management, as the amount of chunks in free-list will decrease substantially. This is due to the merge function combining together free blocks that are available for merging.

4.1 Effects of merging

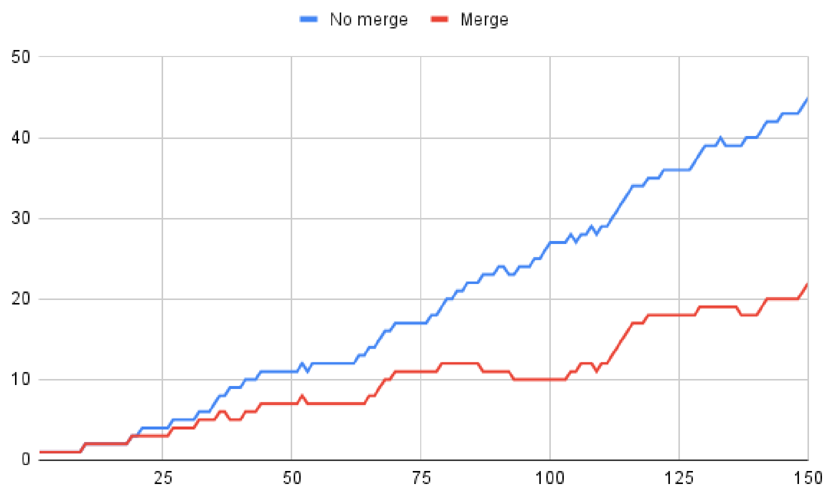

Figure 1: No merge vs merge with 150 runs

5 Improvements and their impact on performance

I have (from the instructions) chosen improvement number 4.1 (the size of the head) as my only implemented improvement. Strangely, as the improvements are meant to decrease the number of blocks in the free list, the smaller HEAD actually increased them. My only possible explanation for this is the decrease of the minimal block size in LIMIT (even if MIN increases). This results in that some blocks that before were not eligible for splitting now are, thus resulting in more splits and more blocks, ergo a longer free list.

5.1 Effects of the improvement

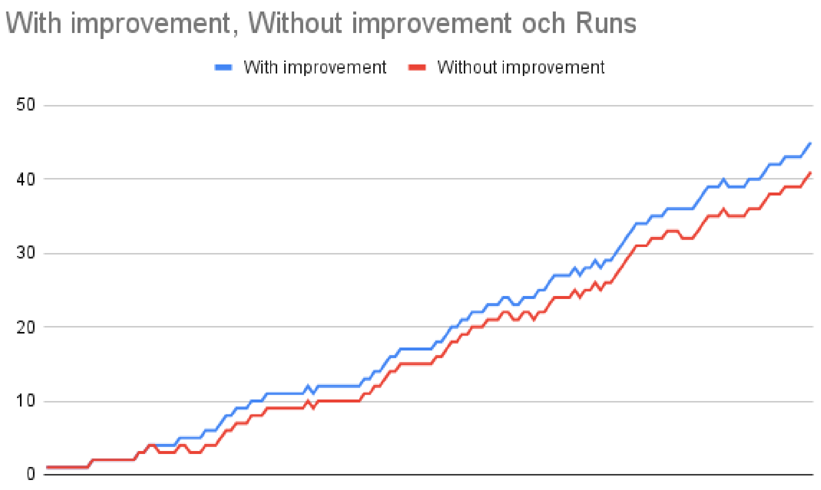

Figure 2: Regular HEAD vs smaller HEAD with 150 runs

6 Conclusion

With regards to presented material, it is safe to assume that merge has somewhat reduced the amount of blocks in the free list. The header improvement on the other hand added a small constant amount, while merge appears to decrease the free list non-linearly. Even though it was not specified, I should have added time-measurements in the benchmark, as it seems that this measurement is assumed for most benchmarks. There are several more improvements and optimizations that exists, which I chose not to implement due to time-constraints.