Ryan Barker
ECE 2230
11/18/2014

# MP4 Test Log

| Performance Analysis Data | | | | | |
|---|---|---|---|---|---|
| Search Policy | Roving Policy | Coalescing | Time (ms) | Size of linked list (chunks) | Total memory (B) |
| First fit | Rove | Off | 4545.55 | 49737 | 10346496 |
| First fit | Rove | On | 237.207 | 1 | 770048 |
| First fit | Head | Off | 1517.61 | 50659 | 13041664 |
| First fit | Head | On | 260.869 | 1 | 741376 |
| Best fit | Rove | Off | 689.929 | 7141 | 933888 |
| Best fit | Rove | On | 264.162 | 1 | 704512 |
| Best fit | Head | Off | 501.292 | 7141 | 933888 |
| Best fit | Head | On | 259.4 | 1 | 708608 |

| | Time (ms) | Size of linked list (chunks) | Total memory (B) |
|---|---|---|---|
| System malloc / free | 191.376 | 1 | 135168 |

**Discussion of Implementation:**

- chunk_t Structure: I opted to use chunk_t's containing one next pointer and one size variable. That is, I opted to make the free list a one way linked list rather than a two way linked list. This helps made updating pointers in mem_alloc and mem_free simpler, since there was only one pointer to null or point to the free list, but also made parts of certain algorithms more complicated, since any time I needed to get the block before rover, I had to make a previous pointer and rover it to the position before rover. In a two way implementation, this is irrelevant. I think the advantages and disadvantages to both approaches weigh equally, so no matter which you go with, you're doing about the same amount of work.
- First Fit versus Best Fit: By itself, first fit is slightly faster than best fit, since it takes less time to complete in average case because its' checks are more lenient. However, best fit makes far more efficient use of the memory currently in the free list, so when run with coalescing, it is the far superior option for system stability.
- Roving Policies: When coalescing or using best fit, the Roving Pointer's initial position is relatively irrelevant toward runtime, since both the best fit search algorithm and the coalescing algorithm work to fight fragmentation in different ways and substantially reduce the amount of memory fragments in the free list. However, when using first fit or not coalescing, the position of the roving pointer makes a huge difference to the efficiency of the list: When it is set to head, it will concentrate memory fragments at the beginning of the list. However, if it is set to rove, fragments will be more randomly disturbed throughout the freed list. The farther fragments spread through the free list, the more times memory has to be requested from the system, and

therefore the slower the algorithms go. Because of this, without coalescing or best fit, a roving policy of head is far more efficient to runtime.

- Coalescing: Coalescing offers great improvement to run time because it fights memory fragmentation by sticking adjacent memory pieces together. This allows for block sizes to increase, when adjacent allocated blocks are freed, meaning that memory does not need to be requested from the system as often. Coalescing is used to its fullest potential when run with best fit, which makes the most efficient use out of the available blocks in the free list.