ECE 2230 MP4
PERFORMANCE ANALYSIS
Cameron L. Burroughs

For this program, I opted to keep the singly-linked list structure of mem_chunk_t. For a singly circular linked list, only one pointer on each effected block needs to be changed, whereas on a doubly-linked list, two pointers would need to be changed. By implementing a singly linked structure, inserting and removing items from the list is must faster, as less pointers need to be changed, and it is simple to understand in terms of a circularly linked list. In a circularly linked list, the rover can transverse forward (or Rover->next) infinitely. With a singly linked list, it is simpler to understand this, as rover can only go in one direction. If a doubly linked structure were used, it may cause more complicated debugging. A downfall of this structure is that often times, when an item needs to be removed, a previous pointer is needed. For this, an additional previous pointer needs to be made that will be updated each run through the loop. This additional step may slow down the removal process.

The First-fit policy is advantageous in the sense that it decreases the amount of time to return a block of memory and splits each block of memory in a way such that the memory is fairly distributed across the list. A draw back, however, is that since each block gets split without regard for the amount of remaining memory, it quickly decreases the number of large blocks, and may cause more memory to be used in the end as new pages need to be added for larger requests. Best-fit takes longer to find the correct block to return to the user, especially if there is not an exact size block available, however, by taking memory from the block with the smallest difference, it allows larger blocks of memory to stay intact and may prevent the addition of another page sooner than first fit would. A disadvantage of this policy is that without perfect matches, the rover will have to transverse the entire list comparing differences and eventually split the blocks leaving small fragments left. This process is a bit slower than just returning the first block with enough available space. In summary, First-fit is strong at reducing time, whereas Best fit is strong in reducing wasted memory.

Coalescing makes the process of freeing memory much slower, however it prevents new pages from being added when allocating memory and may even make allocation faster, as there would be fewer blocks in the free list and they would have more memory available. Coalescing greatly decreases fragmentation, however it is only useful when contiguous blocks are freed. If every other block were to be freed from an allocated set of pointers, coalescence would have no effect in comparison to running the same operation without coalescence.

Table 1: Simulation in Equilibrium (./lab4 -e)

Policy	Time (ms)	Size of Linked List	Total Memory (bytes)
First-fit, no coalescing	3575.72	47,482	8,773,632
Best-fit, no coalescing	813.196	7,732	1,101,824
First-fit, coalescing	233.308	593	913,408
Best-fit, coalescing	256.487	297	827, 392
System Malloc	193.249	280	945,136

Table 2: Simulation in Equilibrium with range = 0 (./lab4 -e -r 0)

Policy	Time (ms)	Size of Linked List	Total Memory (bytes)
First-fit, no coalescing	141.486	415	847,872
Best-fit, no coalescing	144.573	412	847,872
First-fit, coalescing	166.274	212	802,816
Best-fit, coalescing	179.934	99	798,720
System Malloc	207.541	205	809,968

As seen in Table 1, without coalescing, best-fit greatly improved both the time it took to run as well as the amount of memory over first-fit. This is due to the amount of fragmentation first-fit produces. As more fragmentation occurs, more pages need to be added which is a very slow process. Best-fit, however, decreases fragmentation and therefore allows less pages to be added as the memory requested increases. In Table 2, the amount of memory requested always stays the same, therefore fragmentation is relatively equal among first-fit and best-fit without coalescing. As you can see, even though the memory used is the same for the two, best-fit took slightly longer.

Next, looking at the two policies in Table 1, coalescing significantly decreases the amount of time it takes for the program to run, as well as decreases the total memory used. Coalescing allows larger blocks to be made and therefore it is much faster for the policies to fit a suitable fit for the requested block of memory. This, in turn, decreases fragmentation, decreasing memory. Comparing the two policies, best-fit uses slightly less memory, however it increases the runtime. This tradeoff may be useful when memory is quite expensive, however in this case, first-fit with coalescing would be my preferred method. Comparing to Table 2, when the size of blocks requested is constant, coalescing increases the runtime and has little effect on the total memory used.