Nicholas Prezioso

njp107

Michael Serpico

mvs71

In our implementation, we used an implicit list, where each block requires two bytes of metadata. The most significant bit in one of the bytes is used to record allocation information. mymalloc() searches through memory[] in order to find the first unallocated block that is a large enough size (this is the FIRST fit algorithm mentioned in class) and returns NULL if this is unsuccessful. The chosen block is split in two, with the first half being allocated and of the requested size and the second half being free and of the remaining size. If the next block is also free, they are conjoined. myfree() makes sure that the given address is a valid pointer to an allocated block within memory[] and sets the allocation bit to 0. Afterwards, it check if the next block is free and merges the two if so. The same is done with the previous block using an O(n) search, where n is the number of blocks. Rather than printing an error for every error case, we count the number of each specific type of error encountered and print their counts out at the end. This is because printf statements are quite expensive, and doing a huge number of them would require an enormous amount of time to complete.

Results of running memgrind.out

Average time (in microseconds) for workload A: 58773

Average time (in microseconds) for workload B: 125

Average time (in microseconds) for workload C: 3522

Average time (in microseconds) for workload D: 3887

Average time (in microseconds) for workload E: 10211

Average time (in microseconds) for workload F: 315

Average time (in microseconds) for a full A-F workload: 76836

1917700 Mallocs that successfully returned addresses

133300 Mallocs that retuned NULL

1917700 Successful frees

133300 Attempts to free a NULL pointer

0 Attempts to free at an address outside the given memory array

0 Attempts to free already unallocated blocks

Interesting results

Workload A takes the longest of all of them. This makes sense, as it makes the most calls to malloc and free. It makes sense that the last two error types never happened, as none of the workloads should have involved those sorts of calls to free.