Joseph Klaszky & Patrick Hommes

Asst1 February 26, 2017

Mymalloc is a program created to mimic the standard C library functions of malloc and free. Using a linked list, various sizes of 'memory', including the size of the meta-data that would normally be present within a standard memory allocation, are split off from a pre-set block of 5000 'bytes'. Once the pre-defined limit is reached the program will catch the end of the memory and report to the user when this occurred. The blocks can also be freed so that the memory can be used again. Once freed the blocks are merged back with the larger set block of the original 5000. Running the makefile will compile mymalloc and execute the memgrind portion of the program running through several various tests of the malloc and free system, its capabilities, as well as any errors found.

Expected output:

Test A - (malloc * 1000 then free * 1000) * 100...

Not enough memory to allocate 1 bytes in file: memgrind.c, line: 32
A total of 80100 error(s) were collected.
User requested a total of 80100 bytes over 80100 attempts
that could not be allocated in available space.
Final size of data block 4976

Total calls to malloc: 100000
Total valid calls to malloc: 19900
Total calls to free: 19900
Average time of execution: 690.740000 ms.

In Test A the errors encountered were experienced once the allotted memory space of 5000 was reached. Our malloc handled the error and reported it as expected. With the metadata of our malloc structure being 24 bytes, 199 bytes plus their metadata (199 x 24) could be allocated into each run as expected.

The final size of 4976 is expected as the first iteration of the metadata struct is initialized. The average time is very high, possibly because of the many errors and invalid calls for malloc once space had run out.

Test B - (malloc then free * 1000) * 100...

No errors found.
Final size of data block 4976

Total calls to malloc: 100000
Total valid calls to malloc: 100000
Total calls to free: 100000

Test B reports no errors, as expected. With only one byte being allocated and then immediately freed there should never be an issue of running out of space for additional calls to malloc and metadata. Since no errors were found the average runtime was much lower than Test A.

Total calls to free: 100000
Average time of execution: 15.950000 ms.

Test C - (random 1 byte malloc or free * 100) * 100...

No errors found.
Final size of data block 4976

Total calls to malloc: 100000
Total calls to free: 100000
Total calls to free swapped to malloc: 1769
Average time of execution: 178.550000 ms.

Test C has also reposted no errors as expected. With a random 50/50 call of malloc or free and only one byte being allocated there should once again never be an issue with space not being available. No errors were found but the time was greater than Test B assumingly due to the random calls and testing for whether a new malloc is needed or freeing can happen.

Test D - (random *random* byte malloc or free * 100) * 100...

Not enough memory to allocate 46 bytes in file: memgrind.c, line: 142
A total of 126 error(s) were collected.
User requested a total of 5005 bytes over 126 attempts
that could not be allocated in available space.
Final size of data block 4976

Total calls to malloc: 100000
Total bytes requested: 3253983
Total calls to free: 100000
Total calls to free swapped to malloc: 3725
Average time of execution: 171.640000 ms.

Test D reported as expected errors similar to size A, but much less frequently. As with Test C the 50/50 call keeps enough available space free for additional malloc calls with the exception that the added random size of bytes can result in several malloc calls in a row that could eventually exceed the available space. The average time is slightly less than Part C but with more errors and a second random call which is interesting.

Test E - (step bytes then free * 1000) * 100...

User requested 0 bytes in file: memgrind.c, line: 203
A total of 100 error(s) were collected.
User requested 0 bytes 100 times.
Final size of data block 4976

Total calls to malloc: 100000
Total valid calls to malloc: 99900

Test E ran as expected, stepping from 0 to 1000 along with the loop counter. The easily fixed error of starting at 0 with the loop and therefore requesting 0 bytes was left in to show the error being handled and reported. Like part B the average execution time is quite low, possibly only higher to the larger size requests and allocations.

Total bytes requested: 53203983
Total calls to free: 99900
Average time of execution: 17.510000 ms.

Test F - (step byte malloc * 1000 then free * 1000) * 100...

Not enough memory to allocate 79 bytes in file: memgrind.c, line: 235
A total of 92200 error(s) were collected.
User requested a total of 49741900 bytes over 92200 attempts
that could not be allocated in available space.
Final size of data block 4976

Total calls to malloc: 100000
Total valid calls to malloc: 7800
Total calls to free: 7800
Average time of execution: 229.390000 ms.

Test F performed as expected. As with Test E it was stepping with the loop but this time from 1 to 1001, so the requested 0 bytes error is no longer expected. Like in Test A the available memory was quickly filled without anything before it being freed, but since the stepping loop made the memory size increase this happened much faster resulting in lass total valid runs before maxing the memory. The timing of time F is much lower than Part A, but still higher than all other tests. Again, possibly sure to the large amount of invalid malloc requests, possibly being faster from the lower amount of free calls required.

Previous Case for Test F:

Not enough memory to allocate 79 bytes in file: test.c, line: 238
Tried to free NULL in file: test.c, line: 249
A total of 92300 error(s) were collected.
User requested a total of 49741900 bytes over 92200 attempts
that could not be allocated in available space.
User attempted to free NULL 100 times.
Final size of data block 4976

In the previous case the free loop was not changed along with the stepping loop and therefore tried to free down to 0, resulting in trying to free NULL errors, shown found and reported here….