CS518 - A1: UMalloc!

Group Members:

Jay Patil (jsp255)

Aishwarya Harpale (ach149)

OVERVIEW

The aim of this assignment is to create a memory allocator using *malloc()* and *free()* functions that are used to dynamically allocate memory in C. We have implemented the first-fit algorithm wherein the first free large enough partition is considered to accommodate the requested amount of memory (in Bytes). The output of this assignment was a user-level library that logically allocated memory in a 10MB character array that resembled an OS memory and also freed up the memory. The library also returned appropriate errors for incorrect allocation of memory as well as incorrect freeing of memory.

Note: This code was tested on ilab machine Butter in the Meltdown Lab hosted on butter.cs.rutgers.edu.

API

1. init():

   This function is used to initialize the memory if not already done. It checks if the isMemoryIntialized flag is set; if not, the memory is initialized and the flag is set.

2. umalloc(size_t size, char* file, int line):

   This function is used to allocate the requested memory. It finds the first free partition (block) in the memory that is big enough to accommodate the requested amount of memory. It checks for any violating conditions such as trying to allocate memory to a NULL pointer. The user is given a diagnostic error message if any such operations are tried out.

3. ufree(void * ptr, char* file, int line):

   This method deallocates the memory or blocks pointed by the ptr. It also checks if consecutive blocks are free. If that is the case, it merges the free block.

4. getNextBlock(char * ptr)

   This is an auxiliary function used to get the next block in our logical memory so as to find a particular block to either free or to allocate a space to.

5. mergeFreedBocks()

   This method merges all the freed consecutive free blocks into one so as to give the intuition of consecutive free memory.

STRUCTURES

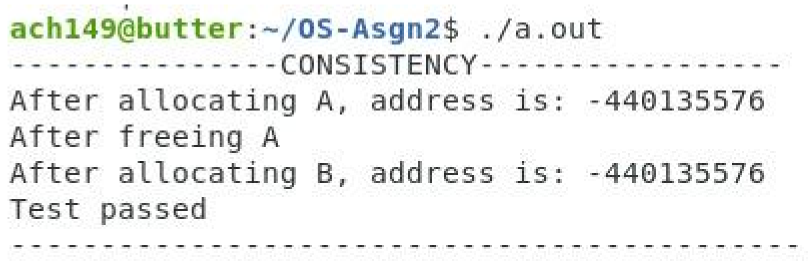
```
typedef struct _metaData {

    bool isFree;

    int blockSize;

}metaData;
```

The above structure is used to maintain the metadata of a memory block in our memory. The 'isFree' variable is used to check if the given memory block is free or not. 'blockSize' is used to maintain the available memory size for usage.

RESULTS

1. Consistency:

This test tries to allocate a small block of memory ranging from 1 to 10 Bytes. Then data is written to it and later the block is freed. Next, another block of the same size is created. This test expects the address of the previously freed pointer and the currently assigned pointer address to be the same and hence checking for consistency.

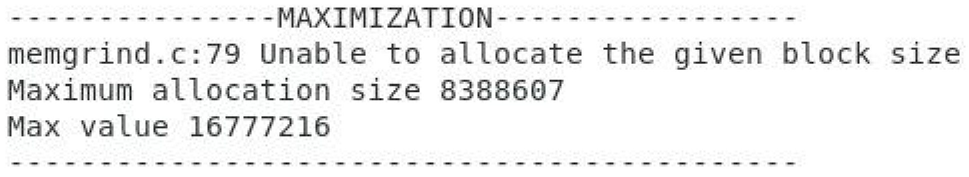


```
ach149@butter:~/OS-Asgn2$ ./a.out
---------------CONSISTENCY-----------------
After allocating A, address is: -440135576
After freeing A
After allocating B, address is: -440135576
Test passed
-------------------------------------------
```

The screenshot above shows the result of the consistency test of our library. We have attempted to allocate a variable of the size of a *char* datatype i.e 1B. It is freed and another variable of the same size is attempted to be allocated. The output is that both of the variables have the same address as expected from the test.

2. Maximization (Simple Coalescence):

This test tries to allocate 1B of memory. If the library does not return NULL, the memory is freed and another block of twice the size is allocated. This process is carried out until the library returns NULL. When NULL is returned, the memory size is halved and again allocation is attempted. On success after NULL, the memory is freed and the test is complete.

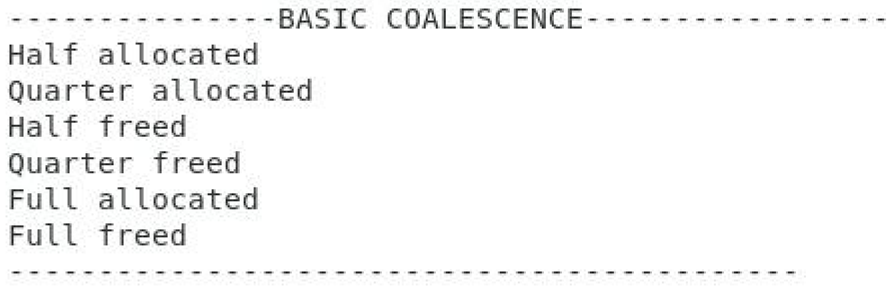


```
---------------MAXIMIZATION----------------
memgrind.c:79 Unable to allocate the given block size
Maximum allocation size 8388607
Max value 16777216
--------------------------------------------
```

The above screenshot shows that the maximum allocation of size 8388607 B can be done. A result error is shown when the library attempts to allocate 16777216 B while trying out the maximization test.

3. Basic Coalescence:

This test allocates half the size of the maximum memory (i.e 10MB) and then allocates one-quarter of the maximum memory. Then the first block is freed as well as the second block. Once this is done, it tries to allocate maximum memory. The library should allow it and then free the memory.

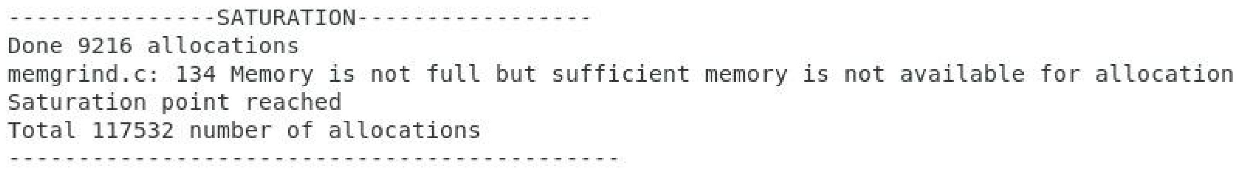


```
---------------BASIC COALESCENCE-----------------
Half allocated
Quarter allocated
Half freed
Quarter freed
Full allocated
Full freed
--------------------------------------------
```

The above screenshot shows the output of the basic coalescence test. Initially, half the maximum possible memory is allocated. Further, one-quarter of the maximum memory is allocated and both allocations are freed. Maximum possible allocation is tried and after successful allocation, the memory is deallocated.

4. Saturation:

This test attempts to allocate 9K of 1KB allocations. After that, it starts allocating 1B until the library (i.e. *umalloc()*) returns NULL and hence saturates the space.

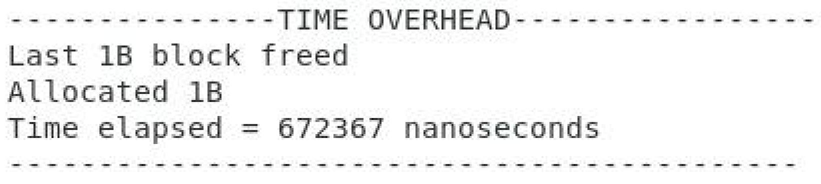


```
---------------SATURATION-----------------
Done 9216 allocations
memgrind.c: 134 Memory is not full but sufficient memory is not available for allocation
Saturation point reached
Total 117532 number of allocations
------------------------------------------
```

The above screenshot shows the successful implementation of the saturation test. After 9216 allocations, and continuous 1B allocations, there is not sufficient free memory to make the next allocation which indicates that our memory is saturated. The same result is reflected in the message provided by the library. Therefore, the total number of allocations 117532.

5. Time Overhead:

Immediately after the previous test, the last 1B block is freed. It then checks the max time overhead by calculating the time elapsed after allocating another 1B.

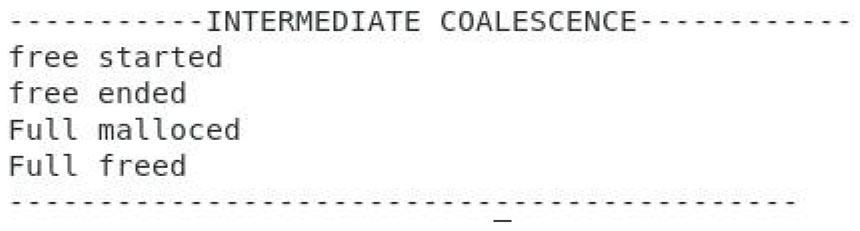


```
--------------TIME OVERHEAD-----------------
Last 1B block freed
Allocated 1B
Time elapsed = 672367 nanoseconds
-------------------------------------------
```

Above screenshot shows the time overhead calculation. The maximum time overhead for our library as seen from the above screenshot is 672367 nanoseconds.

6. Intermediate Coalescence:

After test 5 i.e Time Overhead, free all the memory one by one. After this, the test attempts to allocate the maximum memory possible (i.e 10MB) and expects the library to successfully allocate the memory.



```
-----------INTERMEDIATE COALESCENCE------------
free started
free ended
Full malloced
Full freed
-------------------------------------------
```

As seen from the above screenshot, we have initially freed all the memory one by one that was allocated during saturation. Later maximum possible memory is attempted to be allocated. After successfully allocating, the block is freed.

HOW TO RUN

To make the library, run “make all”.

To run memgrind after this, run “./memgrind”.

To clean all the created files, run “make clean”.

To simply run memgrind.c without creating a library, run “gcc memgrind.c umalloc.c”.

ANALYSIS

While implementing this assignment we came across some of the issues during memory allocation like fragmentation wherein there is a sufficient amount of free memory, however, it is not a contiguous block of memory as it is scattered or fragmented. We were able to create a user-level library that closely resembles the malloc() and free() functions for memory allocation.