Name: Neha Alluri
Net Id: nra48

Implementation Details

Implemented malloc & free functions as below.
The malloc function allocates all memory aligned to 8-byte address. As discussed in alignment section, the static memory buffer is created using double

```
static double memory[MEMSIZE/8];
```

Implementation details are as below

1. malloc

Created memory of size 8192 bytes (defined MEMSIZE in myalloc.h)
The memory block is allocated using double to ensure 8 byte alignment

Each allocated memory has a header with blocksize & prevblock

```
typedef struct _memheader {
   uint32_t blocksize;
   uint32_t prevsize;
} mh;
```

malloc returns the address after the header

blocksize most significant bit is reserved for memory used or free; 1 -> used, 0 -> free
remaining 31 bits are used for size of block;
max memory that can be supported is approximately 2 TB
prevsize is used to keep track of previous block size
when free is called, blocksize and prevsize are used to defragment the block in O(1)

Note: When a block is found, check if the free space can be used for that request and remaining space can be used for another block of at-least 8 bytes. In such case break the block into 2 parts. Also, if needed adjust the previous block size of the next block if it exists.

2. free

When free is called, check if previous block is free.
If prevblock is free, defragment previous block and current block
Next check if next block is free. If next block is free, join current block and next block; update the memheader appropriately. It is possible that 3 blocks are degramented, if the previous block and next block of current block is being freed.

Test Plan Details

Various scripts are develops to test performance (memgrind.c) and test aspects like allocation, free & defragmentation. Details are below

1.1 Memory fragmentation

Developed 2 test cases to check memory fragmentation/defragmentations

1. checkbasic.c

This is basic test function to test malloc and free and also test 8 byte allignment.

Code

```
char *mem1 = malloc (10);
char *mem2 = malloc (1);
char *mem3 = malloc (20);
char *mem4 = malloc(33);
printf ("mem1: %p Is 8 bit aligned: %ld\n" , mem1, (uint64_t)mem1 & 0x7  );
printf ("mem2: %p Is 8 bit aligned: %ld\n" , mem2, (uint64_t)mem2 & 0x7  );
printf ("mem3: %p Is 8 bit aligned: %ld\n" , mem3, (uint64_t)mem3 & 0x7  );
printf ("mem4: %p Is 8 bit aligned: %ld\n" , mem4, (uint64_t)mem4 & 0x7  );
free(mem2);
free(mem1);
free(mem3);
free(mem4);

char *mem5 = malloc(99);
printf ("mem5: %p Is 8 bit aligned: %ld\n" , mem5, (uint64_t)mem5 & 0x7  );

free (mem5);
```

Output

```
nra48@assembly:~/cs214/a1$ ./checkbasic
mem1: 0x55ec42609f08 Is 8 bit aligned: 0
mem2: 0x55ec42609f20 Is 8 bit aligned: 0
mem3: 0x55ec42609f30 Is 8 bit aligned: 0
mem4: 0x55ec42609f50 Is 8 bit aligned: 0
mem5: 0x55ec42609f08 Is 8 bit aligned: 0
```

This tests that memory is always 8 byte aligned and memory is reclaimed when free is called.
By releasing mem2 at start, tested that mem1 & mem3 are defragmented

2. checkfrag.c

This program repeatedly allocates 20 byte blocks till memory is not available. At the end it frees 2 blocks and tries to allocated 40 bytes**. This function also shows that when memory is not available, prints error message with file name and lineno.**

```
nra48@assembly:~/cs214/a1$ ./checkfrag
Defragmnter check
Error: malloc memory not available: called from: checkfrag.c line: 12
Allocated 256 block of 20 bytes each
Allocate 40 bytes malloc(40)
Error: malloc memory not available: called from: checkfrag.c line: 20
40 bytes not available
 free  2 blocks
Address of memory allocated 0x55fd74dff5c8
Success: memory is defragmented
```

1.2 Implementation Details

Tested by allocation following blocks: 1K-8=1016, 2K-8=2040, 4K-8=4088, 1K-8=1016, 1K-8=1016
The malloc fails after 8K is allocated (as expected) and right offsets are given. After this 2k block is released and allocated 2 blocks of 1k-8 each

Code: checkimpl.c	Output:

char *mem1 = malloc (1016);
char *mem2 = malloc (2040);
char *mem3 = malloc (4088);
char *mem4 = malloc (1016);
char *mem5 = malloc (1016);
printf ("Addr of mem1 %p\n", mem1);
printf ("Addr of mem2 %p\n", mem2);
printf ("Addr of mem3 %p\n", mem3);
printf ("Addr of mem4 %p\n", mem4);
printf ("Addr of mem5 %p\n", mem5);
 	
free (mem2);
char *mem6 = malloc (1016);
char *mem7 = malloc (1016);
 	
printf ("Addr of mem6 %p\n", mem6);
printf ("Addr of mem7 %p\n", mem7);
	nra48@assembly:~/cs214/a1$ ./checkimpl
	Error: malloc memory not available: called from: checkimpl.c line: 12
	Addr of mem1 0x5641cc70e0c8
	Addr of mem2 0x5641cc70e4c8
	Addr of mem3 0x5641cc70ecc8
	Addr of mem4 0x5641cc70fcc8
	Addr of mem5 (nil)
	Addr of mem6 0x5641cc70e4c8
	Addr of mem7 0x5641cc70e8c8

2. Detectable errors 5 Performance Testing

Code: err.c

Can be run with argument 1, 2 or 3
This module tests free with wrong address.
My implementation of free provides right error messages

```
nra48@ilab2:~/cs214/a1$ ./err 1
Error: free -> wrong address: called from: /common/home/nra48/cs214/a1/err.c line: 17

nra48@ilab2:~/cs214/a1$ ./err 2
Error: free -> trying to free wrong block: called from: /common/home/nra48/cs214/a1/err.c line: 22
nra48@ilab2:~/cs214/a1$ ./err 3
Error: free -> trying to free wrong block: called from: /common/home/nra48/cs214/a1/err.c line: 28
```

3. Reporting errors

Used preprocessor macros of __FILE__ && __LEN__ to detect file name & line number when errors are detected for malloc & free. Checkimpl.c & Err.c and other programs provide required messages. The following checks are done

Malloc: Shows error when memory block of required size is not available
Free: checks if the memory header (8 bytes before pointer) has right header. Also checks if the memory pointer being freed is with in the memory block start and end and the block is not previously freed. The Detectable Errors is section 2 shows appropriate messages

4 Correctness Testing

The files err.c, checkimpl.c and checkcorrect.c tests various parts of malloc & free buffers. Err.c & checkimpl.c are described 2 and 1.2. The details for checkcorrect is below

Checkcorrect.c creates 4 memory buffers (1k,2k,4k and 1k) and writes pattern of 1,2, 3 & 4 respectively. After this the entire blocks are read to ensure that data is not corrupted.
And then it tries to malloc addition 2k which fails as expected, After this allocated memory block2 (size 2 k) is freed and 2 blocks of 1k is created. The data in the 2 blocks shows that data is not corrupted.

Code: Refer to file: checkcorrect.c
Output:

```
nra48@assembly:~/cs214/a1$ ./checkimpl
Error: malloc memory not available: called from: checkimpl.c line: 12
Addr of mem1 0x5641cc70e0c8
Addr of mem2 0x5641cc70e4c8
Addr of mem3 0x5641cc70ecc8
Addr of mem4 0x5641cc70fcc8
Addr of mem5 (nil)
Addr of mem6 0x5641cc70e4c8
Addr of mem7 0x5641cc70e8c8
```

5 Performance Testing

Performance testing is done by calling the 3 performance tests as described in the project details file. In addition 2 tests are developed to check other performance. Each test is repeated 50 times, and the min, max and average times are captured. Used function pointers to perform each test and not duplicate the code to capture statistics

Code: memgrind.c

Output:

```
nra48@assembly:~/cs214/a1$ ./memgrind
Performance test1: 50 repeats of 1 byte alloc and immediately free 120 times
Results Min: 3 Max: 5 Average: 3
Performance test2: 50 repeats of 1 byte alloc 120 times  and free 1 byte 120 times
Results Min: 40 Max: 50 Average: 42
Performance test3: 50 repeats of random 1 byte alloc or 1 byte free with 120 allocs
After randomly allocating 120 1 byte buffers, free up all allocated remaing buffers
Results Min: 13 Max: 30 Average: 19
Performance test4: 50 repeats of random size  byte alloc ( 1 to 50 bytes) and  immediately free 120
times
This is similar to test1 .. except we allocate between 1 and 50 bytes instead of just 1 byte
Results Min: 5 Max: 10 Average: 6
Performance test5: 50 repeats of random size (1 to 50) byte alloc or  free one of the allocated buffers
with 120 allocs
This is similar to test 3. Instead of allocating 1 byte buffer we allocate randsom size buffer
Results Min: 13 Max: 22 Average: 14
nra48@assembly:~/cs214/a1$
```

Have run the memgrind function multiple times and found that the averages are similar.

Details of tests

Test 1: Allocate 1 byte and immediately free that 1 byte, 120 times. This test is performed 50 times to capture, min, max & average times for allocating and freeing 120 times. The min time is 3 micro seconds, max is 5 microseconds and average is 3 microseconds.

Test 2: Allocate 1 byte 120 times and then free the 120 allocated buffers. Used array to store the pointers. This test is performed 50 times to capture, min, max & average times for allocating and freeing 120 times. The min time is 40 micro seconds, max is 50 microseconds and average is 42 microseconds.

Test 3: Used srand to initialize random variable and rand to get random values. Call rand repeatedly and based on the last of output, allocate ot free a memory of 1 byte, Ensure that rand is called required to allocate 120 bytes, And remaining bytes are freed at the end. This test is performed 50 times to capture, min, max & average times for allocating and freeing 120 times. The min time is 13 micro seconds, max is 30 microseconds and average is 19 microseconds. Test 1: Allocate 1 byte and

Test 4: Test 4 is similar to test except instead of allocating 1 byte buffer allocate buffer of random size (1 to 50 bytes) This test is performed 50 times to capture, min, max & average times for allocating and freeing 120 times. The min time is 5 micro seconds, max is 10 microseconds and average is 6 microseconds.

Test 3: Test 5 is similar to test 3 and instead of allocating 1 byte buffer allocate buffer of random size (1 to 50 bytes).. This test is performed 50 times to capture, min, max & average times for allocating and freeing 120 times. The min time is 13 micro seconds, max is 22 microseconds and average is 14 microseconds.

6. Alignment

As described in section 1, Implementation details memory buffer is allocated using double This ensure original unallocated memory block is assigned to 8 byte boundary. The header used for each allocation is 8 bytes (4 bytes for blocksize & 4 bytes for prevblock). At each malloc if the size requested is not multiplier of 8, round up to 8 byte boundary. In the file checkbasic.c allocated blocks of size 10, 1, 20 & 33 bytes and checked that memory allocated is always aligned to 8 byte boundary. After that the memory blocks are freed and new block of size 99 is created. This also is aligned to 8 bytes

Code: checkbasic.c

Output: ra48@assembly:~/cs214/a1$ ./checkbasic
mem1: 0x55b8c8c81f08 Is 8 bit aligned: 0
mem2: 0x55b8c8c81f20 Is 8 bit aligned: 0
mem3: 0x55b8c8c81f30 Is 8 bit aligned: 0
mem4: 0x55b8c8c81f50 Is 8 bit aligned: 0
mem5: 0x55b8c8c81f08 Is 8 bit aligned: 0

How to Run

The tar file has the required source and header files and Makefile
Run “make clean” to cleanup unwanted files
Run “make all” to create required binaries.
The binaries include following. The functionality is specified in the sections above

1. checkbasic
2. checkcorrect
3. checkfrag
4. memgrind
5. err
6. checkimpl