BLG 312E Computer Operating Systems Homework 2 Report

Mustafa Kırcı
Faculty of Computer and
Informatics Engineering
Istanbul Technical University
Email: kirci20@itu.edu.tr
Student No: 150200096

***Abstract*—This homework involves implementing a thread safe memory management system on a shared memory block that multiple processes can access. This memory management system has corressponding functions to classical memory management provided by OS's such as MyMalloc, MyFree. On top of these, there are InitMyMalloc and DumpFreeList functions providing debugging tools and initialization options. MyMalloc function provides the requested size with given allocation strategy, MyFree releases the memory occupied by given pointer, InitMyMalloc initializes the heap block to be used and DumpFreeList prints which blocks of heap are occupied and which of them are free.**

I. Answers to Questions

What are the main differences between "malloc" and "mmap"? If you had to make a decision between these two, which would you choose? Why?

The main differences between malloc and mmap are about how they allocate a memory. Malloc allocates memory and returns a pointer to that memory. In the other hand, mmap maps files or devices into memory. Malloc is a simple function and widely supported. Mmap has more specific use cases and it can be used to share memory between different processes unlike malloc.

Which one i would prefer would depend on my use case. If i want to use the allocated memory between different processes, i have to use mmap. On the other hand, if i only need a dynamically allocated memory block that will be used only in the current process then malloc is much simpler choice.

What was the method/call you used in MyFree() instead of free()? Do you think it is as successful as free() at correctly and safely releasing the memory back? Why?

In the MyFree function, the block start of given pointer is calculated. And then this block is inserted into the free list. The previous block, next block and the created block are then checked for coalescing. To do this operation, coalesceFreeList method is called.

In extensive testing of my memory management system, there was no bug found. So in theory, it is as safe as real free function. But there can be always some edge cases and it is almost certaion that original free function is more robust on safe. The original implementation is battle tested through years unlike MyFree function.

II. Implementation Details

A. InitMyMalloc

This function initializes the heap block to be used. It takes the size of the heap block that will be mapped. If the size of the heap block is not a multiple of the page size, the function rounds it up to the nearest multiple of the page size.

The function first initializes the heap block with the given size. Then it allocates some space for the free list and the mutex lock. The free list is a linked list that holds the free blocks in the heap. The mutex lock is used to synchronize the access to the heap block. Figure 1 shows the representation of heap block.

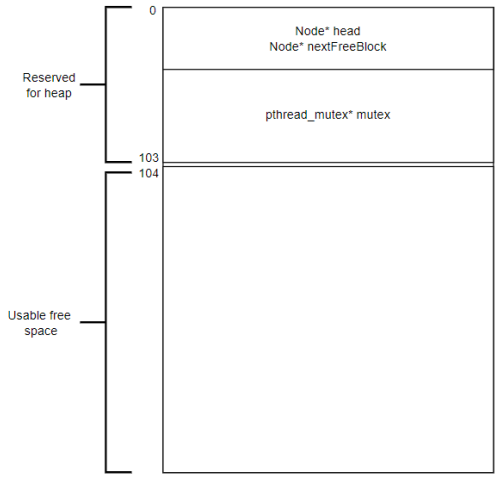

Fig. 1. Allocated Heap Representation

The function returns 0 if the initialization is successful, -1 otherwise.

B. MyMalloc

This function allocates a memory block of the given size. It takes the size of the memory block as a parameter and strategy as a parameter. The strategy can be either FIRST_FIT, BEST_FIT, WORST_FIT or NEXT_FIT.

Function finds the free block according to the strategy and returns the pointer to the start of the block. During this operation, the free block is splitted if the remaining is big enough to hold another data. Header information such as magic number and is freed are also set. Function updates head and next free block pointers accordingly.

The function returns the pointer to the start of the allocated block if the allocation is successful, NULL otherwise.

C. MyFree

This function releases the memory block occupied by the given pointer. It takes the pointer to the start of the block as a parameter. The function first checks if the given pointer is valid. To determine whether the pointer is valid or not, the function checks the magic number and is freed fields of the block. It also checks whether given pointer is in boundaries of the heap.

If pointer is valid, function inserts the freed block into the free list. The previous block, next block and the created block are then checked for coalescing. To do this operation, coalesceFreeList method is called. Function updates head and next free block pointers accordingly.

The function returns 0 if the free operation is successful, -1 otherwise.

D. DumpFreeList

This function prints the free list. It prints the start addresses and sizes of each block in the free list. Occupied blocks are also printed in this function. Those blocks are determined by calculating the distances between ends and starts of each free block in free list.

E. unInitMyMalloc

This function uninitializes the heap block. It frees the heap block and the mutex lock.

III. Testing

A. Strategies Test

This test creates 3 pointers from the heap using MyMalloc. Frees the middle variable and then allocates the 4th variable. Dumps the free list after each step to be observed. Before quitting, frees every allocated memory. This is repeated for all of the allocation strategies.

FIRST_FIT

Figure 2 shows the results of the FIRST_FIT tests. As it is expected, the fourth variable is given the second variable's space because it was the first free memory block in the heap.

```
Initializing MyMalloc
InitMyMalloc result: 0

Testing strategies alone
Testing FIRST_FIT strategy
------------------------------------------------------------
ptr: 0x7fdb41b2e080
Address: 0, Size: 138, Status: Full
Address: 138, Size: 3958, Status: Free

ptr2: 0x7fdb41b2e0a2
Address: 0, Size: 172, Status: Full
Address: 172, Size: 3924, Status: Free

ptr3: 0x7fdb41b2e0c4
Address: 0, Size: 206, Status: Full
Address: 206, Size: 3890, Status: Free

Freed ptr2
Address: 0, Size: 138, Status: Full
Address: 138, Size: 34, Status: Free
Address: 172, Size: 34, Status: Full
Address: 206, Size: 3890, Status: Free

ptr4: 0x7fdb41b2e0a2
Address: 0, Size: 206, Status: Full
Address: 206, Size: 3890, Status: Free

Freeing ptr
Address: 0, Size: 104, Status: Full
Address: 104, Size: 34, Status: Free
Address: 138, Size: 68, Status: Full
Address: 206, Size: 3890, Status: Free

Freeing ptr3
Address: 0, Size: 104, Status: Full
Address: 104, Size: 34, Status: Free
Address: 138, Size: 34, Status: Full
Address: 172, Size: 3924, Status: Free

Freeing ptr4
Address: 0, Size: 104, Status: Full
Address: 104, Size: 3992, Status: Free
```

Fig. 2. First Fit Test Results

NEXT_FIT

```
Testing NEXT_FIT strategy
------------------------------------------------------------
ptr: 0x7fdb41b2e080
Address: 0, Size: 138, Status: Full
Address: 138, Size: 3958, Status: Free

ptr2: 0x7fdb41b2e0a2
Address: 0, Size: 172, Status: Full
Address: 172, Size: 3924, Status: Free

ptr3: 0x7fdb41b2e0c4
Address: 0, Size: 206, Status: Full
Address: 206, Size: 3890, Status: Free

Freed ptr2
Address: 0, Size: 138, Status: Full
Address: 138, Size: 34, Status: Free
Address: 172, Size: 34, Status: Full
Address: 206, Size: 3890, Status: Free

ptr4: 0x7fdb41b2e0e6
Address: 0, Size: 138, Status: Full
Address: 138, Size: 34, Status: Free
Address: 172, Size: 68, Status: Full
Address: 240, Size: 3856, Status: Free

Freeing ptr
Address: 0, Size: 104, Status: Full
Address: 104, Size: 68, Status: Free
Address: 172, Size: 68, Status: Full
Address: 240, Size: 3856, Status: Free

Freeing ptr3
Address: 0, Size: 104, Status: Full
Address: 104, Size: 102, Status: Free
Address: 206, Size: 34, Status: Full
Address: 240, Size: 3856, Status: Free

Freeing ptr4
Address: 0, Size: 104, Status: Full
Address: 104, Size: 3992, Status: Free
```

Fig. 3. Next Fit Test Results

Figure 3 shows the results of the NEXT_FIT tests. As it is expected, the fourth variable is given the space after third variable because that free block was the biggest free block in the heap.

BEST_FIT

Figure 4 shows the results of the BEST_FIT tests. As it is expected, the fourth variable is given the second variable's space because it was the best free memory block to the requested size in the heap.

```
Testing BEST_FIT strategy
------------------------------------------------------------
ptr: 0x7fdb41b2e080
Address: 0, Size: 138, Status: Full
Address: 138, Size: 3958, Status: Free

ptr2: 0x7fdb41b2e0a2
Address: 0, Size: 172, Status: Full
Address: 172, Size: 3924, Status: Free

ptr3: 0x7fdb41b2e0c4
Address: 0, Size: 206, Status: Full
Address: 206, Size: 3890, Status: Free

Freed ptr2
Address: 0, Size: 138, Status: Full
Address: 138, Size: 34, Status: Free
Address: 172, Size: 34, Status: Full
Address: 206, Size: 3890, Status: Free

ptr4: 0x7fdb41b2e0a2
Address: 0, Size: 206, Status: Full
Address: 206, Size: 3890, Status: Free

Freeing ptr
Address: 0, Size: 104, Status: Full
Address: 104, Size: 34, Status: Free
Address: 138, Size: 68, Status: Full
Address: 206, Size: 3890, Status: Free

Freeing ptr3
Address: 0, Size: 104, Status: Full
Address: 104, Size: 34, Status: Free
Address: 138, Size: 34, Status: Full
Address: 172, Size: 3924, Status: Free

Freeing ptr4
Address: 0, Size: 104, Status: Full
Address: 104, Size: 3992, Status: Free
```

Fig. 4. Best Fit Test Results

WORST_FIT

```
Testing WORST_FIT strategy
------------------------------------------------------------
ptr: 0x7fdb41b2e080
Address: 0, Size: 138, Status: Full
Address: 138, Size: 3958, Status: Free

ptr2: 0x7fdb41b2e0a2
Address: 0, Size: 172, Status: Full
Address: 172, Size: 3924, Status: Free

ptr3: 0x7fdb41b2e0c4
Address: 0, Size: 206, Status: Full
Address: 206, Size: 3890, Status: Free

Freed ptr2
Address: 0, Size: 138, Status: Full
Address: 138, Size: 34, Status: Free
Address: 172, Size: 34, Status: Full
Address: 206, Size: 3890, Status: Free

ptr4: 0x7fdb41b2e0e6
Address: 0, Size: 138, Status: Full
Address: 138, Size: 34, Status: Free
Address: 172, Size: 68, Status: Full
Address: 240, Size: 3856, Status: Free

Freeing ptr
Address: 0, Size: 104, Status: Full
Address: 104, Size: 68, Status: Free
Address: 172, Size: 68, Status: Full
Address: 240, Size: 3856, Status: Free

Freeing ptr3
Address: 0, Size: 104, Status: Full
Address: 104, Size: 102, Status: Free
Address: 206, Size: 34, Status: Full
Address: 240, Size: 3856, Status: Free

Freeing ptr4
Address: 0, Size: 104, Status: Full
Address: 104, Size: 3992, Status: Free
```

Fig. 5. Worst Fit Test Results

Figure 5 shows the results of the WORST_FIT tests. As it is expected, the fourth variable is given the space after third variable because that free block was worst free block for the requested size in the heap.

B. Utilization Test

This test aims to observe the utilization of the heap block. For this purpose, heap memory is allocated at 4096 bytes. Since 104 bytes are reserved for the heap, the remaining space is 3992 bytes.

Then the test tries to allocate 3992 - sizeof(struct block) bytes of memory. After that, it frees the given pointer to make up space for the next iteration.

This process is conducted for requst sizes of 3992 - sizeof(struct block) -1, 3992 - sizeof(struct block), 3992 - sizeof(struct block) +1, 3992 - 2*sizeof(struct block) -1, 3992 - 2*sizeof(struct block), 3992 - 2*sizeof(struct block) +1 bytes.

```
Testing full utilization
Address: 0, Size: 104, Status: Full
Address: 104, Size: 3992, Status: Free

Requested 3967 bytes, returned pointer: 0x7fdb41b2e080 with size 3967
Address: 0, Size: 4096, Status: Full
Freeing ptr
Address: 0, Size: 104, Status: Full
Address: 104, Size: 3992, Status: Free

Requested 3968 bytes, returned pointer: 0x7fdb41b2e080 with size 3968
Address: 0, Size: 4096, Status: Full
Freeing ptr
Address: 0, Size: 104, Status: Full
Address: 104, Size: 3992, Status: Free

Requested 3969 bytes, returned pointer: (nil) with size 0
Address: 0, Size: 104, Status: Full
Address: 104, Size: 3992, Status: Free
Freeing ptr
Address: 0, Size: 104, Status: Full
Address: 104, Size: 3992, Status: Free

Requested 3943 bytes, returned pointer: 0x7fdb41b2e080 with size 3943
Address: 0, Size: 4071, Status: Full
Address: 4071, Size: 25, Status: Free
Freeing ptr
Address: 0, Size: 104, Status: Full
Address: 104, Size: 3992, Status: Free

Requested 3944 bytes, returned pointer: 0x7fdb41b2e080 with size 3944
Address: 0, Size: 4096, Status: Full
Freeing ptr
Address: 0, Size: 104, Status: Full
Address: 104, Size: 3992, Status: Free

Requested 3945 bytes, returned pointer: 0x7fdb41b2e080 with size 3945
Address: 0, Size: 4096, Status: Full
Freeing ptr
Address: 0, Size: 104, Status: Full
Address: 104, Size: 3992, Status: Free
```

Fig. 6. Utilization Test Results

The size of struct block is 24 bytes for this test environment.

As it can be seen from Figure 6, allocating 3967 bytes is successful but remaining 1 bytes are given to the pointer. This is because the remaining 1 byte is not enough to hold all the data for a node in the free list.

Requesting 3968 bytes is also successful because the free space is just enough for the requested size.

However requesting 3969 bytes is not successful because 3969 + 24 is equal to 3993. But there is no large enough block to

accommodate this request.
Requesting 3943 bytes is fine because remaining 25 bytes are enough to be allocated for possible upcoming request.
Requesting 3944 bytes is also fine but this time, the remaining 24 bytes are also allocated to the pointer. This is because 24 bytes cannot hold any data for upcoming request. That pointer's header size is already 24 bytes. Requesting 3945 bytes is also fine but remaining 23 bytes are allocated to the pointer from the same reason as previous case.

C. Fork Test

This tests aims to observe the threaded usage of the memory management system. Different processes will request memory from the same heap in the physical memory. MyMalloc, MyFree and DumpFreeLists must be thread safe to successfully accomplish memory allocations.

```
Testing shared Memory usage between processes
Malloc strategies
0: BEST_FIT
1: WORST_FIT
2: FIRST_FIT
3: NEXT_FIT
Enter the fork number 0 's malloc strategy
0
Enter the fork number 0 's malloc size
500
Enter the fork number 1 's malloc strategy
1
Enter the fork number 1 's malloc size
1000
Enter the fork number 2 's malloc strategy
2
Enter the fork number 2 's malloc size
1500
Enter the fork number 3 's malloc strategy
3
Enter the fork number 3 's malloc size
2000
Heap before other processes
Address: 0, Size: 104, Status: Full
Address: 104, Size: 3992, Status: Free

Allocating memory from different processes in same memory block
P1 Child process
Child PID: 10709
Child strategy: BEST_FIT
Malloc succeded in process PID 10709 with a pointer 500 byte at address 184 in heap.

P4 Child process
Child PID: 10712
Child strategy: NEXT_FIT
Malloc succeded in process PID 10712 with a pointer 2000 byte at address 708 in heap.

P2 Child process
Child PID: 10710
Child strategy: WORST_FIT
Malloc succeded in process PID 10710 with a pointer 1000 byte at address 2732 in heap.

P3 Child process
Child PID: 10711
Child strategy: FIRST_FIT
Malloc failed in process PID 10711 for request size of 1500

Address: 0, Size: 3732, Status: Full
Address: 3732, Size: 364, Status: Free
Freeing Pointers
Address: 0, Size: 104, Status: Full
Address: 104, Size: 3992, Status: Free
```

Fig. 7. Fork Test Results

As it can be seen from Figure 7, process P1 will allocate 500 bytes with strategy of best fit, process P2 will allocate 1000 bytes with strategy of worst fit, process P3 will allocate 1500 bytes with strategy of first fit and process P4 will allocate 2000 bytes with strategy of next fit.
Order of processes are random because they are running concurrently.
As it can be seen, memory is allocated for processes P1, P2 and P4. The reason it fails for the process P3 is that there is no enough space. The free space is only 364 bytes and 364-24(header size) which is 340 of them can be requested. So MyMalloc function returns NULL pointer.
After child processes are terminated, the allocated variables still live and have to be freed. Those pointers were stored in an array created before, and via accessing that array, those allocated memories are freed and memory returns to its initial position.

D. Invalid Pointer Test

This test aims to observe the behaviour of MyFree function to given various pointers, such as a valid pointer, recently freed pointer and a random pointer.

```
Testing freeing invalid pointer
ptr: 0x7fdb41b2e080
Address: 0, Size: 138, Status: Full
Address: 138, Size: 3958, Status: Free

Freeing ptr
MyFree result: 0
Address: 0, Size: 104, Status: Full
Address: 104, Size: 3992, Status: Free

Freeing ptr again
MyFree result: -1
Address: 0, Size: 104, Status: Full
Address: 104, Size: 3992, Status: Free

Freeing unvalid pointer
MyFree result: -1
Address: 0, Size: 104, Status: Full
Address: 104, Size: 3992, Status: Free
```

Fig. 8. Free Invalid Pointer Test Results

Firstly, a memory is allocated from the heap and then it is freed. This free operation is expected to be successful. Then the same pointer is attempted to be freed again. As it can be seen from figure 8, MyFree function returns an error as expected.
Then random pointer inside heap is attempted to be freed. But MyFree function again returns an error.

E. Uninitialize Heap Test

This test aims to observe behaviour when the mapped heap block is unmapped.

```
Uninitializing MyMalloc
ptr is NULL, MyMalloc has failed as expected (unitialized heap)
```

Fig. 9. Uninitialize Heap Test Results

As it can be seen from Figure 9, MyMalloc call after uninitializing heap fails because there is not any heap to allocate memory from.

IV. Compiling The Source Code

There is a Makefile in the source code. To use this Makefile, you need to have make in the system. Most Linux distribution have make installed as default.

To create an executable and object files, type ”make” in the shell. Of course the shell needs to be in the same directory as the Makefile. Object files will be created in the obj directory and executable will be created in the bin folder.

To run the executable, type ”make run” without quotes.

To clear object files and executable, type ”make clean”.