ITP 30002-02 Operating System, 2023 Spring

Homework 3

Wooseob Chung, 21700664, 21700664@handong.ac.kr

Gwangjin Jeong, 21900651, 21900651@handong.ac.kr

1. Introduction

In this homework, in accordance with the given baseline code, we tried to implement a buddy memory allocation system. In the buddy memory allocation system, when there is a request for memory allocation, the algorithm allocates the memory which is currently available and has a fitting size for allocation request, which is not too big and not too small. And every memory block should be connected to each other then we can find them easily.

2. Approach

Implementing buddy memory allocation system, we use given struct form of bm_header is concluded with three variables, used, size, and pointer variable *next. Since all memory blocked is connected, so we must be able to access other memory block by referring bm_header address in *next. With function fitting(), we find optimal memory size for memory allocation requests.

For implementing bmalloc(), we should check if there is empty memory space available for request. If empty memory space exists, we divide the memory space into fitting size and allocate that memory. If there is no empty memory space available, we map a new page size 4096 with mmap() function. To implement bfree(), we erase the information of memory, and coalesce that empty memory to another sibling memory block if possible. With function bmprint(), we iterate through the whole memory block and print the information of them, and print total memory status.

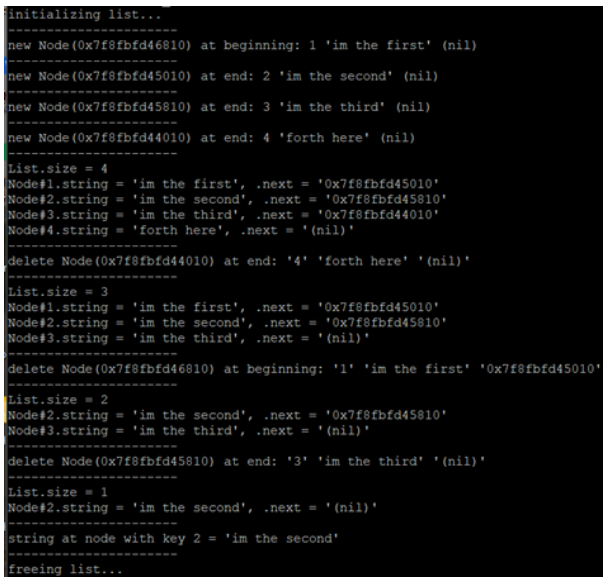
3. Evaluation

```
s21900651@peace:~/OS_HW3$ ./test1
==================== bm_list ====================
=================================================
Total Memory: 0
Used Memory: 0
Available Memory: 0
Internal fragmentation: 0
bmalloc(2000):0x7f74b9efa010
==================== bm_list ====================
  0:0x7f74b9efa010:1       11:00 00 00 00 00 00 00 00
  1:0x7f74b9efa810:0       11:00 00 00 00 00 00 00 00
=================================================
Total Memory: 4064
Used Memory: 2032
Available Memory: 2032
Internal fragmentation: 16
```

```
bmalloc(2500):0x7f74b9ef9010
==================== bm_list ====================
  0:0x7f74b9efa010:1       11:00 00 00 00 00 00 00 00
  1:0x7f74b9efa810:0       11:00 00 00 00 00 00 00 00
  2:0x7f74b9ef9010:1       12:00 00 00 00 00 00 00 00
=================================================
Total Memory: 8144
Used Memory: 6112
Available Memory: 2032
Internal fragmentation: 16
bfree(0x7f74b9efa010)
==================== bm_list ====================
  0:0x7f74b9ef9010:1       12:00 00 00 00 00 00 00 00
=================================================
Total Memory: 4080
Used Memory: 4080
Available Memory: 0
Internal fragmentation: 0
```

```
bmalloc(1000):0x7f74b9efa010
==================== bm_list ====================
  0:0x7f74b9ef9010:1       12:00 00 00 00 00 00 00 00
  1:0x7f74b9efa010:1       10:00 00 00 00 00 00 00 00
  2:0x7f74b9efa410:0       10:00 00 00 00 00 00 00 00
  3:0x7f74b9efa810:0       11:00 00 00 00 00 00 00 00
=================================================
Total Memory: 8128
Used Memory: 5088
Available Memory: 3040
Internal fragmentation: 32
bmalloc(1000):0x7f74b9efa410
==================== bm_list ====================
  0:0x7f74b9ef9010:1       12:00 00 00 00 00 00 00 00
  1:0x7f74b9efa010:1       10:00 00 00 00 00 00 00 00
  2:0x7f74b9efa410:1       10:00 00 00 00 00 00 00 00
  3:0x7f74b9efa810:0       11:00 00 00 00 00 00 00 00
=================================================
Total Memory: 8128
Used Memory: 6096
Available Memory: 2032
Internal fragmentation: 16
```

The provided images are the result of executing the 'test1.c' code that uses 'bmalloc.h'. In the initial state, everything is initialized to 0. When a size is inputted using 'bmalloc', the code divides the existing block into smaller blocks to match the requested size and allocates memory accordingly. In this case, an input of 2000 bytes divided the block into smaller blocks with a size of 11. It then allocated the requested memory within one of these blocks. Every time a new allocation is made, the code divides the block and allocates memory according to the size of the block, while also changing the used value to 1. When allocating memory, if the combined remaining memory space is smaller than the requested size in bytes, a new block with a size of 12 is created and allocated.



The image above shows the result of executing the 'test3.c' code. This code creates a linked list and inserts or removes nodes at the beginning or end while allocating and freeing memory using ‘bmalloc()’. The program verifies that nodes are correctly allocated and freed through the use of bmalloc.h.

After implementing bmalloc.c and running the test cases, it was observed that while some tests produced the expected results, there were also tests that did not execute correctly.

4. Discussion

While implementing the buddy memory allocation algorithm, we could consider its advantages compared to malloc() allocation method.

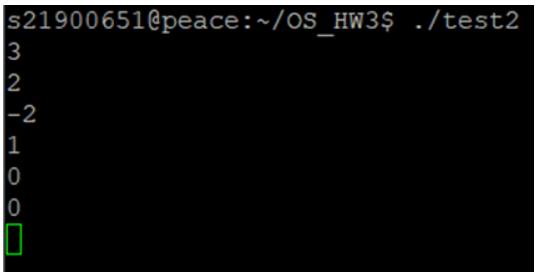
First, we can reduce the time complexity while adding or removing a memory block. Since it uses the binary tree method, it requires less time complexity, O(log 2 n), compared to the existing method, O(log n).

Secondly, by minimizing internal fragmentation, we can increase the efficiency of memory utilization. In the case of bmalloc, which we implemented in this assignment, it differs from the traditional memory allocation method by dividing a single page into smaller units for allocation. As a result, it reduces internal fragmentation. For example, let's consider allocating 1000 bytes of memory from a 4096-byte page. In the traditional memory allocation method, we would allocate one full page (4096 bytes), resulting in 3096 bytes of internal fragmentation. However, with bmalloc, we can divide the page into smaller units and allocate a fitting size, resulting in only 24 bytes of internal fragmentation. This allows for more efficient space utilization.

By minimizing internal fragmentation, bmalloc improves the efficiency of memory utilization. It allocates memory in smaller units, reducing wasted space and ensuring that memory blocks are utilized optimally. This leads to better overall memory management and improved performance in memory allocation and deallocation operations.

However, when a value such as 2050 bytes is allocated, an internal fragmentation of 2046 bytes occurs because it must be allocated to a page of 4096 bytes. The method of allocating memory by dividing it in this way has the advantage of reducing internal fragmentation, but it is also necessary to find a way to reduce memory waste as much as possible because of the disadvantage of wasting memory depending on the size of input.

While we were implementing the functions in the given baseline code, we reached some limits below.



The provided image shows the result of executing the 'test2.c' code. This code continuously prompts the user for input until 0 is entered, and it allocates memory only for positive numbers. However, in the written code, even when 0 is entered, the loop does not terminate, causing an issue. Despite efforts to find the cause of the problem, the exact reason could not be identified.

5. Conclusion

In this homework, we tried to implement a buddy memory allocation system. In accordance with the given baseline code, we tried to complete given functions. By mapping a fitted size memory, we can reduce memory waste in the process of memory allocation.

References

https://www.kuniga.me/blog/2020/07/31/buddy-memory-allocation.html

https://ko.wikipedia.org/wiki/%EB%B2%84%EB%94%94_%EB%A9%94%EB%AA%A8%EB%A6%AC_%ED%95%A0%EB%8B%B9