CS 503 Lab 4

Memory Management
Aligned getmem

Shuheng Guo

1. Introduction

In this lab, we are going to create a new system call to allocate memory from the heap. However, different from XINU system call getmem, this new system call meet the requirement that the address of the allocated memory be *aligned* to a multiple of a particular number.

```
char    *aligned_getmem(uint32 bytes, uint32 alignment);
```

which allocates *bytes* bytes of memory at an address that is a multiple of *alignment*.

2. Implementation

To implement this function, there are several additional requirements.

- Requirement: The value of *alignment* must be a power of 2 and the value of *bytes* must be a multiple of alignment.
- Implementation:

```
if (nbytes % alignment != 0) {
    restore(mask);
    return (char *)SYSERR;
}

if (alignment & (alignment - 1)){
    restore(mask);
    return (char *)SYSERR;
}
```

- Requirement: If the value of *alignment* or *bytes* is less than 8, the values must be rounded up to 8.
- Implementation:

```
if (nbytes < 8) {
    nbytes = (uint32) roundmb(nbytes);
}
if (alignment < 8){
    alignment = (uint32) roundmb(alignment);
}
```

- Requirement: For values of less than 8 the behavior should be:
    - 0 - return SYSERR
    - 1 - round up to 8
    - 2 - round up to 8
    - 3 - return SYSERR
    - 4 - round up to 8
    - 5 - return SYSERR
    - 6 - return SYSERR
    - 7 - return SYSERR

- Implementation:

```
int badalignments[5] = {0, 3, 5, 6 ,7};
int i;
for (i = 0; i < 5; i++) {
    if (alignment == badalignments[i]) {
        restore(mask);
        return (char *)SYSERR;
    }
}
```

- Requirement: The number of bytes and the value for alignment must be greater than 0. Return SYSERR if either is 0.
- Implementation:

```
if (nbytes <= 0 || alignment <= 0) {
    restore(mask);
    return (char *)SYSERR;
}
```

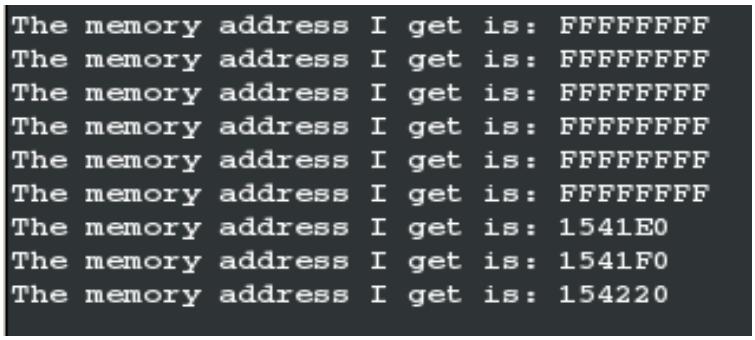
At this time, all of the requirements are meet. And then I go through the memory free list to find a memory address which is multiple of alignment and the memory block length is nbytes at the same time.

3. Results and Analysis

To test the functionality of my new created system call aligned_getmem, I write several test case to test each possible situation.

```
process main(void)
{
    char * yihan;
    yihan = aligned_getmem(0,2);
    kprintf("The memory address I get is: %X\n", (uint32)yihan);
    yihan = aligned_getmem(8,-2);
    kprintf("The memory address I get is: %X\n", (uint32)yihan);
    yihan = aligned_getmem(3,3);
    kprintf("The memory address I get is: %X\n", (uint32)yihan);
    yihan = aligned_getmem(20,2);
    kprintf("The memory address I get is: %X\n", (uint32)yihan);
    yihan = aligned_getmem(31,8);
    kprintf("The memory address I get is: %X\n", (uint32)yihan);
    yihan = aligned_getmem(18,9);
    kprintf("The memory address I get is: %X\n", (uint32)yihan);
    yihan = aligned_getmem(8,4);
    kprintf("The memory address I get is: %X\n", (uint32)yihan);
    yihan = aligned_getmem(32,16);
    kprintf("The memory address I get is: %X\n", (uint32)yihan);
    yihan = aligned_getmem(64,32);
    kprintf("The memory address I get is: %X\n", (uint32)yihan);
    return OK;
}
```

The results of test case above is shown in the picture below:



1. nbytes = 0, alignment = 2.

   Result: SYSERR

   nbytes equals 0. This case doesn't meet the requirement.

2. nbytes = 8, alignment = -2.

   Alignment is less than zero. This case doesn't meet the requirement.

3. nbytes = 3, alignment = 3.

   Alignment is one of th bad alignments. This case doesn't meet the requirement.

4. nbytes = 20, alignment = 2.

   Alignment is round to 8. 20 is not multiples of 8. This case doesn't meet the requirement.

5. nbytes = 31, alignment = 8.

   nbytes is not multiples of alignment. This case doesn't meet the requirement.

6. nbytes = 18, alignment = 9.

   Alignment is not power of 2. This case doesn't meet the requirement.

7. nbytes = 8, alignment = 4.

   Alignment is round to 8. This case meets the requirement. And it returns an address.

   The address returned is 1541E0, which is 1393120 in decimal. We notice that 1393120 is a multiple of alignment 8.

8. nbytes = 32, alignment = 16.

   Both nbytes and alignment are larger than 8. nbytes is multiple of alignment. Alignment is power of two. This case meets the requirement.The address returned is 1541F0, which is 1393136 in decimal. We notice that 1393136 is a multiple of alignment 16.

9. nbytes = 64, alignment = 32.

   Both nbytes and alignment are larger than 8. nbytes is multiple of alignment. Alignment is power of two. This case meets the requirement.The address returned is 154220, which is 1393184 in decimal. We notice that 1393184 is a multiple of alignment 32.

4. Discussion

- Q: How does your solution ensure allocation of memory in an aligned manner?
- A: Each time when I am going to allocate memory for user I will examine the memory address. If the address is not a multiple of alignment, I program to move the address to a multiple of alignment and remain the enough memory length at the same time.
- Q: Describe your test cases. How do they ensure that your code correctly meets the requirements?
- A: I had 9 test cases here. These test cases include all the error and correct representatives. As discussed above, they will ensure that my code meets the requirement.
- Q : Does it make sense to allow an alignment value that is not a power of 2, of so why if not why not?
- A: Since all the memory addresses are power of 2. It doesn't make any sense to find an address that is a multiple of alignment which is not a power of 2.
- Q: The getmem system call is very similar to the POSIX malloc system call. In POSIX, there is a corresponding free system call which performs a very similar operation to the freemem system call in XINU, however the freemem system call in XINU requires the

caller to specify not only the memory address to free, but also the size of memory to free (take a look at system/freemem.c). The POSIX free does not require the caller to specify the size (only the address). Why does XINU require the size to be specified when freemem is called? What modifications would need to be made to XINU to not require the caller to specify a size when the call freemem?

- A: Because XINU doesn't have any mechanism to record the memory size each time it malloc for users. It require the size of memory to free. If we add a new variable in struct memblk to record the memory size each time the system execute getmem system call, XINU will be able to free the memory when necessary with out requirement of memory size. According to different computer architecture, to ensure the new variable we add is able to record 64bit computer memory size, we'd better use a 8 bytes variable to do this job in memblk.