CoE 135: Lab 7 Documentation (Memory)

Salmon, Paulino III I.
2015-11557
paulino.salmon@eee.upd.edu.ph

I. Properties of kmalloc() (50 pts)

1) Does kmalloc() always allocate memory sizes in powers of two?

To answer this, I created a for loop in my kernel module that would continuously just allocate and free memory used by kmalloc() at a maximum value of 255. This counter is to be plugged in the code line: **ptr = kmalloc(i, GFP_KERNEL);**

```
// #1
unsigned int i;
for(i = 0; i < 255; i++) {
        ptr = kmalloc(i, GFP_KERNEL);
        printk(KERN_INFO "I got: %zu
            bytes of memory. I is: %d\n",
            ksize(ptr), i);
        kfree(ptr);
}
```

Running this kernel module in *dmesg*, it shows that the allocated bytes of memory are **not always** in powers of two (See 96 bytes and 192 bytes below).

```
paulinosalmon@paulinosalmon-VirtualBox: ~/Desktop/GitClones/...
File  Edit  View  Search  Terminal  Help
[ 1314.366148] I got: 96 bytes of memory. I is: 89
[ 1314.366149] I got: 96 bytes of memory. I is: 90
[ 1314.366150] I got: 96 bytes of memory. I is: 91
[ 1314.366151] I got: 96 bytes of memory. I is: 92
[ 1314.366151] I got: 96 bytes of memory. I is: 93
[ 1314.366152] I got: 96 bytes of memory. I is: 94
[ 1314.366153] I got: 96 bytes of memory. I is: 95
[ 1314.366154] I got: 96 bytes of memory. I is: 96
[ 1314.366155] I got: 128 bytes of memory. I is: 97
[ 1314.366156] I got: 128 bytes of memory. I is: 98
[ 1314.366157] I got: 128 bytes of memory. I is: 99
[ 1314.366158] I got: 128 bytes of memory. I is: 100
[ 1314.366159] I got: 128 bytes of memory. I is: 101
```

```
paulinosalmon@paulinosalmon-VirtualBox: ~/Desktop/GitClones/...
File  Edit  View  Search  Terminal  Help
[ 1314.366228] I got: 192 bytes of memory. I is: 188
[ 1314.366229] I got: 192 bytes of memory. I is: 189
[ 1314.366229] I got: 192 bytes of memory. I is: 190
[ 1314.366230] I got: 192 bytes of memory. I is: 191
[ 1314.366231] I got: 192 bytes of memory. I is: 192
[ 1314.366232] I got: 256 bytes of memory. I is: 193
[ 1314.366233] I got: 256 bytes of memory. I is: 194
[ 1314.366234] I got: 256 bytes of memory. I is: 195
[ 1314.366235] I got: 256 bytes of memory. I is: 196
[ 1314.366235] I got: 256 bytes of memory. I is: 197
[ 1314.366236] I got: 256 bytes of memory. I is: 198
[ 1314.366237] I got: 256 bytes of memory. I is: 199
[ 1314.366238] I got: 256 bytes of memory. I is: 200
```

Page sizes are normally in powers of two, but the system allocates usually a little bit more than what is asked for as according to **TEXTCITE []**.

2) What is the minimum amount of memory kmalloc() can provide?

Using the code snippet below and printing the kernel module in *dmesg* afterwards:

```
// #2
printk(KERN_INFO "Minimum size is: %d
    bytes.\n", KMALLOC_MIN_SIZE);
```

```
[ 1314.366282] Minimum size is: 8 bytes.
```

Terminal output shows that the minimum amount of memory kmalloc() can provide is **8 bytes**. This KMALLOC_MIN_SIZE constant is defined in the slab.h header.

3) What is the maximum amount of memory kmalloc() can provide?

Using the code snippet below and printing the kernel module in *dmesg* afterwards:

```
// #3
printk(KERN_INFO "Maximum size is: %ld
    bytes.\n", KMALLOC_MAX_SIZE);
```

```
[ 1314.366283] Maximum size is: 4194304 bytes.
```

Terminal output shows that the minimum amount of memory kmalloc() can provide is **4194304 bytes**. This KMALLOC_MAX_SIZE constant is defined in the slab.h header.

4) Is the memory given to you by kmalloc() physically contiguous?

As per stack overflow users number1 and number2, the kmalloc() function guarantees that the pages are both physically and virtually contiguous, although, contiguity for kmalloc() may fail if the order of allocation is already very high.

II. Memory and Paging (50 pts)

1) Determine the value of the Page Size in the system you are currently using.

Using the code snippet below and printing the kernel module in *dmesg* afterwards:

```
// #1
printk(KERN_INFO "Page size is: %ld
    bytes.\n", PAGE_SIZE);
```

```
[ 3928.472193] Page size is: 4096 bytes.
```

Terminal output shows that the value of the page size is **4096 bytes**. This PAGE_SIZE constant is also already defined in one of the linux headers.

2) Determine the start of the HIGHMEM region of your memory. Clearly state the architecture used by your machine when answering this problem.

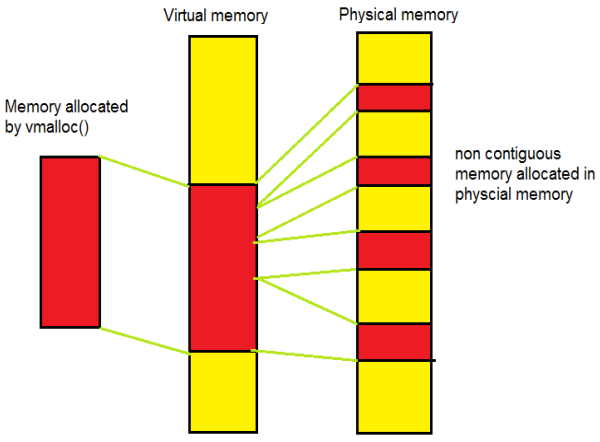
Invoking the *hostnamectl* terminal command, it shows that my system is of the 64-bit architecture.

```
paulinosalmon@paulinosalmon-VirtualBox:~/Desktop/GitClones/C
/Lab/ME7$ hostnamectl
   Static hostname: paulinosalmon-VirtualBox
         Icon name: computer-vm
           Chassis: vm
        Machine ID: 9aa1d72f8e5f401fa8f9cf678e1fbde8
           Boot ID: bcf22be84cd342689190b318db62c746
    Virtualization: oracle
  Operating System: Ubuntu 18.04.3 LTS
            Kernel: Linux 5.0.0-29-generic
      Architecture: x86-64
```

64-bit processors can directly access 2^{64} bytes of memory. As per user Josh Kelly, the linux kernel splits these into the high memory (user space) and the low memory (kernel space). Compared to 32-bit machines in which the high memory address range starts at 0x00000000, high memory does not exist for 64-bit machines as it can access a huge memory of 16 EB, theoretically giving anyone more RAM than they ever need in the entire history of computing.

3) Determine if the memory given by vmalloc() is physically contiguous per page, and if the memory given by vmalloc() is physically contiguous overall.

As per Aliaksei Ramanau, vmalloc() allocates memory that is virtually contiguous but not necessarily physically contiguous overall.



4) Determine if the memory given by vmalloc() in the HIGHMEM region corresponds to a physical memory address.

A memory allocated by vmalloc() has a virtual address, but this may not be necessarily mapped directly to a physical address. The vmalloc() function only allocates contiguous virtual memory. This entire chuck may be fragmented when it comes to physical memory address translation, as show in the previous image.