REPORT: MEMORY ALLOCATOR

BY: - Nishi Jain (121032)

Raj Jhala (121042)

Shailee Mehta (121048)

Shreeya Vachhani (121052)

Table of Contents

Problem Statement:

The Linux kernel is equipped with three memory allocators: SLAB and SLUB, and SLOB. The SLOB (Simple List of Blocks) allocator, located in the Linux kernel tree at mm/slob.c, is a piece of the kernel that, unlike the process scheduler, can easily be extended by students.Write system calls to compute the total amount of memory on the free list as well as the total amount claimed by the SLOB allocator for allocations less than one page (i.e. memory that is either on the free list or has been allocated off the free list and not released). The values returned by these functions can be used to compute a rough measure of internal fragmentation, and students can use these values to compare the amount of fragmentation that results from using different allocation strategies in the SLOB allocator (typically, the best-fit and worst-fit strategies will suffer less from fragmentation than does the first-fit strategy).

Brief Introduction:

The memory management layer is the part of the kernel that services all memory allocation requests. To handle smaller memory requests (less than a whole page, e.g. through malloc()), the kernel currently gives a choice of three different allocators: the SLAB allocator, the SLUB allocator, and the SLOB allocator. SLUB (the most recent of these) and SLAB are complex allocation frameworks for use in resource-rich systems such as desktop computers. They are designed to reduce internal fragmentation of memory, and to permit efficient reuse of freed memory. The SLOB (Simple List Of Blocks) allocator, on the other hand, is designed to be a small and efficient allocation framework for use in small systems such as embedded systems. Unfortunately, a major limitation of the SLOB allocator is the high degree to which it suffers from internal fragmentation. One likely cause for SLOB's high fragmentation rate is the fact that it uses a simple first-fit algorithm for memory allocation. In this project, we have tried to investigate this issue by modifying the SLOB allocator to use the best-fit allocation algorithm, and by writing system calls to provide a measure of the degree of internal fragmentation within the SLOB allocator at a specific point in time.

Understanding:

The Linux kernel is equipped with three memory allocators: SLAB and SLUB, and SLOB. These allocators are on a memory management layer that is logically on top of the system's low level page allocator and are mutually exclusive (i.e. you can only have one of them compiled in your kernel). They are used when a kernel developer calls malloc() or a similar function. They can all be found in the mm directory. All of them follow, to various extends and by extending or simplifying, the traditional slab allocator design.

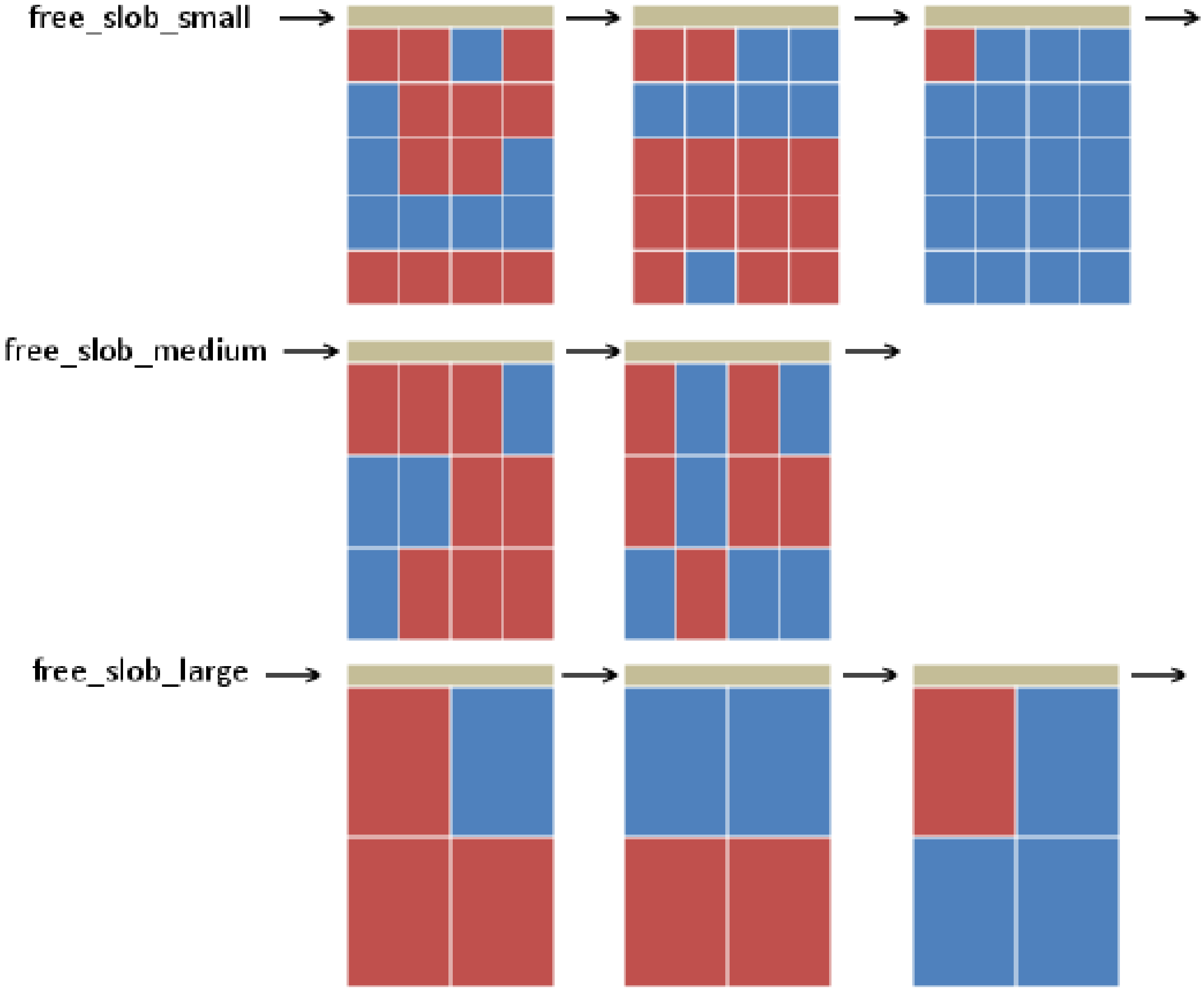
SLOB:

SLOB is a stripped down kernel allocator implementation designed for systems with limited amounts of memory, for example embedded versions/ distributions of the Linux. In fact its design is closer to traditional user land memory allocators rather than the slab allocators SLAB and SLUB. SLOB places all objects/structures on pages arranged in three linked lists, for small, medium and large allocations.

The SLOB (Simple List of Blocks) allocator, located in the Linux kernel tree at mm/slob.c, is a piece of the kernel that, unlike the process scheduler, that can easily be extended.

SLOB organizes memory into pages. Initially, a SLOB page contains a single free block, which is fragmented as necessary to service smaller request sizes. SLOB pages contain blocks of varying sizes, which differentiates SLOB from a classic slab allocator. The units' field for each slob page is checked against the requested allocation size. If the total amount of space available in the page is sufficient, the allocation is attempted. If the page that the block belongs to is full (completely allocated), then the free-list pointer for that page is updated to point to the block and the page is reinserted into the appropriate linked list of partially full pages. These type of allocations (greater than a page size) are passed to the page frame allocator directly, via a call to alloc_pages(). Each SLOB page is broken into individual chunks, which are referred to as blocks (as mentioned in mm/slob.c). The blocks are referenced from singly linked list within each page. For smaller allocations, SLOB maintains three singly-linked lists of partially allocated pages, each of which services requests for allocations of different sizes: less than 256 bytes, less than 1024 bytes, and all other objects less than a page size 4096 bytes:

```
#define SLOB_BREAK1 256
#define SLOB_BREAK2 1024
static LIST_HEAD(free_slob_small);
static LIST_HEAD(free_slob_medium);
static LIST_HEAD(free_slob_large);
```



SLOB basically uses 3 types of memory allocation techniques namely best-fit, first-fit and worst-fit techniques. Internal fragmentation occurs when a memory block of size which is larger than required is allocated. The extra memory space gets wasted as it cannot be utilized in allocation of memory for some other block.

The best-fit technique finds the best block which is best suited according to the requirement and also where the internal fragmentation is minimum.

Whereas the first-fit technique finds the first memory block that can satisfy the requirements. It does not take into consideration the internal fragmentation and hence maximum wastage of memory takes place if this technique is used.

The worst-fit method, it is just the reverse of the best-fit method. It allocates the largest block available in storage list. The idea is to reduce the rate of production of small blocks.

Objectives:

1. Write system calls to compute the total amount of memory on the free list as well as the total amount claimed by the SLOB allocator for allocations less than one page.
2. Attempt to improve the fragmentation rate of the SLOB allocator by modifying it to use a different memory allocation algorithm.
3. Understand and modify an existing component of the Linux kernel tree.

Implementation:

We have read the file header at the top of mm/slob.c to familiarize ourself with the memory allocation schemes presently used in the kernel.
We browsed through mm/slob.c in order to understand how the SLOB allocator works.

The slob_alloc() function implements a next-fit scheme to find a page that has enough space for an incoming request. The slob page_alloc() function implements a first-fit scheme to allocate space within a candidate page. Then we understood that we had to change the first-fit scheme of slob page_alloc() to a best-fit scheme.

Our first objective was to find out the total amount of memory on the free list as well as the total amount claimed by the SLOB allocator using System call. So firstly we understood how system call works and how to implement system call.(started from the basics)

Difficulties faced

We have made a simple “Hello” system call. For that what we have done is:
We have made one makefile and hello.c code. Then hello was included in the kernel's makefile. We declared systemcall in the systemcall_64.tbl and then added systemcall in the header file of the linux. Then we compiled the kernel.

To see the output of this system call, we run a command called dmesg:
It gives output on the command prompt.
Then we have tried to trace the kernel code “slob.c” by writing printk sentences in the given code. After modified slob.c, we tried to see the output of the printk statements. For that we have compiled the kernel. It was successfully compiled. Then we wrote dmesg command to see the output but didn’t succeed. We also tried to see it in the var/log/syslog file though didn’t get it.
Then we were asked to modify page_alloc.c code. The next approach we tried was to calculate the number of free pages using the get_free_pages() function.
From this we thought of getting the number of free and fragmented pages and for that the free cache memory. But this function does not return the free memory of the cache.
We wrote following printk in the page-alloc.c.
printk("The address of a process %lu and the PF_number is %lu",(unsigned long)page_address(page),(unsigned long)virt_to_phys(page));
Return (unsigned long) page_address(page);
We successfully got the output of above statement by using dmesg. Then we came to know about

the mode of the memory allocator which was set as SLUB by default. We didn't change it initially.

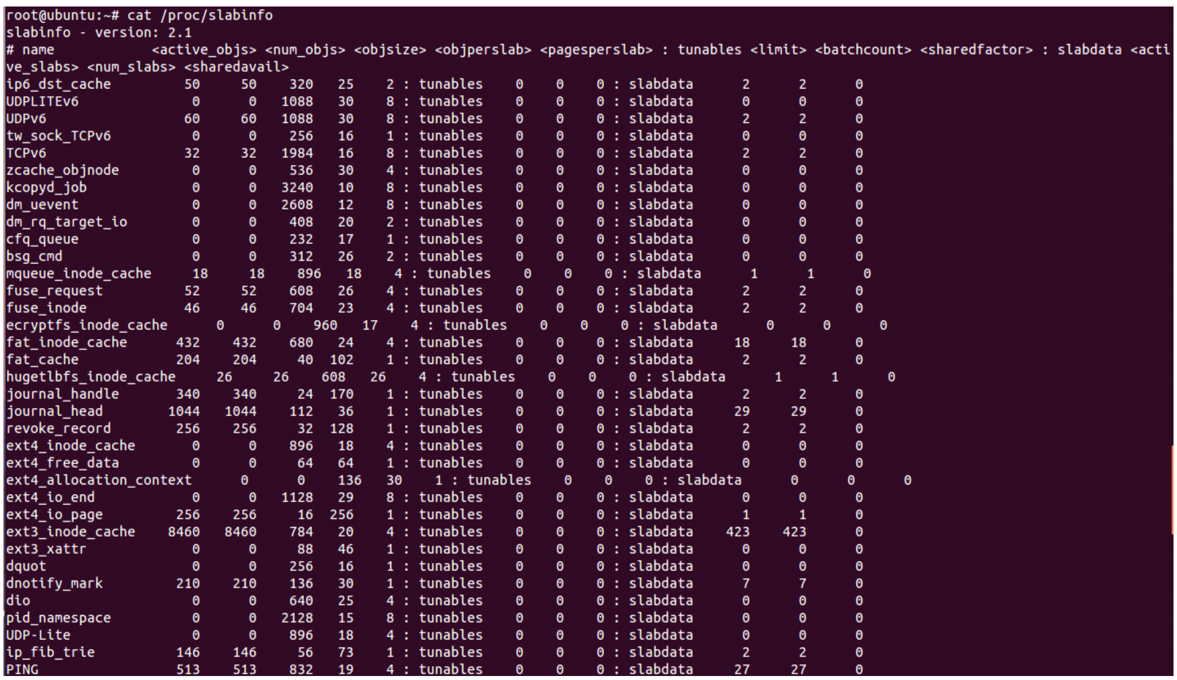
Till now we were working on the kernel version 3.36 but then we started working on version 2. In this version we tried to implement the slob code but every time we try to compile the kernel we get an error stating that cache not found and the kernel crashes. One of our group member's laptop is still facing problems.

Then we have changed it to "SlOB" and then tried again. We successfully got printk output on one of our group member's PC. Then we have modified slob code to convert it in the best-fit. (It gives some errors, we are working on it)

We have developed two system call functions which will give us the total amount of memory on the free list as well as the total amount claimed by the SLOB allocator. We are working on it.

asmlinkage long sys_get_slob_amt_claimed();
asmlinkage long sys_get_slob_amt_free();

Test Results:



```
root@ubuntu:~# cat /proc/slabinfo
slabinfo - version: 2.1
# name            <active_objs> <num_objs> <objsize> <objperslab> <pagesperslab> : tunables <limit> <batchcount> <sharedfactor> : slabdata <acti
ve_slabs> <num_slabs> <sharedavail>
ip6_dst_cache         50     50    320   25    2 : tunables    0    0    0 : slabdata      2      2      0
UDPLITEv6              0      0   1088   30    8 : tunables    0    0    0 : slabdata      0      0      0
UDPv6                 60     60   1088   30    8 : tunables    0    0    0 : slabdata      2      2      0
tw_sock_TCPv6          0      0    256   16    1 : tunables    0    0    0 : slabdata      0      0      0
TCPv6                 32     32   1984   16    8 : tunables    0    0    0 : slabdata      2      2      0
zcache_objnode         0      0    536   30    4 : tunables    0    0    0 : slabdata      0      0      0
kcopyd_job             0      0   3240   10    8 : tunables    0    0    0 : slabdata      0      0      0
dm_uevent              0      0   2608   12    8 : tunables    0    0    0 : slabdata      0      0      0
dm_rq_target_io        0      0    408   20    2 : tunables    0    0    0 : slabdata      0      0      0
cfq_queue              0      0    232   17    1 : tunables    0    0    0 : slabdata      0      0      0
bsg_cmd                0      0    312   26    2 : tunables    0    0    0 : slabdata      0      0      0
mqueue_inode_cache     18     18    896   18    4 : tunables    0    0    0 : slabdata      1      1      0
fuse_request          52     52    608   26    4 : tunables    0    0    0 : slabdata      2      2      0
fuse_inode            46     46    704   23    4 : tunables    0    0    0 : slabdata      2      2      0
ecryptfs_inode_cache      0      0    960   17    4 : tunables    0    0    0 : slabdata      0      0      0
fat_inode_cache      432    432    680   24    4 : tunables    0    0    0 : slabdata     18     18      0
fat_cache            204    204     40  102    1 : tunables    0    0    0 : slabdata      2      2      0
hugetlbfs_inode_cache     26     26    608   26    4 : tunables    0    0    0 : slabdata      1      1      0
journal_handle       340    340     24  170    1 : tunables    0    0    0 : slabdata      2      2      0
journal_head        1044   1044    112   36    1 : tunables    0    0    0 : slabdata     29     29      0
revoke_record        256    256     32  128    1 : tunables    0    0    0 : slabdata      2      2      0
ext4_inode_cache       0      0    896   18    4 : tunables    0    0    0 : slabdata      0      0      0
ext4_free_data         0      0     64   64    1 : tunables    0    0    0 : slabdata      0      0      0
ext4_allocation_context      0      0    136   30    1 : tunables    0    0    0 : slabdata      0      0      0
ext4_io_end            0      0   1128   29    8 : tunables    0    0    0 : slabdata      0      0      0
ext4_io_page         256    256     16  256    1 : tunables    0    0    0 : slabdata      1      1      0
ext3_inode_cache    8460   8460    784   20    4 : tunables    0    0    0 : slabdata    423    423      0
ext3_xattr             0      0     88   46    1 : tunables    0    0    0 : slabdata      0      0      0
dquot                  0      0    256   16    1 : tunables    0    0    0 : slabdata      0      0      0
dnotify_mark         210    210    136   30    1 : tunables    0    0    0 : slabdata      7      7      0
dio                    0      0    640   25    4 : tunables    0    0    0 : slabdata      0      0      0
pid_namespace          0      0   2128   15    8 : tunables    0    0    0 : slabdata      0      0      0
UDP-Lite               0      0    896   18    4 : tunables    0    0    0 : slabdata      0      0      0
ip_fib_trie          146    146     56   73    1 : tunables    0    0    0 : slabdata      2      2      0
PING                 513    513    832   19    4 : tunables    0    0    0 : slabdata     27     27      0
```

FIGURE 1: OUTPUT OF COMMAND CAT /PROC/SLABINFO

```
syslog_old
107752278492160The address of a process 18446612141208305664 and the PF_number is 107752278492160The address of a process 18446612141208305664
and the PF_number is 107752278492160
Apr 20 03:10:56 iict-HP-EliteDesk-800-G1-SFF kernel: [  127.934872] The address of a process 18446612141215907840 and the PF_number is
107752278610944The address of a process 18446612141208305664 and the PF_number is 107752278492160
Apr 20 03:10:56 iict-HP-EliteDesk-800-G1-SFF kernel: [  127.947039] The address of a process 18446612141198880768 and the PF_number is
107752278344896The address of a process 18446612141208305664 and the PF_number is 107752278492160The address of a process 18446612141208305664
and the PF_number is 107752278492160The address of a process 18446612141208305664 and the PF_number is 107752278492160The address of a process
18446612141208305664 and the PF_number is 107752278492160
Apr 20 03:10:56 iict-HP-EliteDesk-800-G1-SFF kernel: [  127.951254] The address of a process 18446612141194854400 and the PF_number is
107752278281984The address of a process 18446612141194854400 and the PF_number is 107752278281984The address of a process 18446612141194854400
and the PF_number is 107752278281984The address of a process 18446612141194854400 and the PF_number is 107752278281984
Apr 20 03:10:56 iict-HP-EliteDesk-800-G1-SFF kernel: [  127.951946] The address of a process 18446612141208305664 and the PF_number is
107752278492160The address of a process 18446612141208305664 and the PF_number is 107752278492160
Apr 20 03:10:56 iict-HP-EliteDesk-800-G1-SFF kernel: [  127.962321] The address of a process 18446612141198880768 and the PF_number is
107752278344896The address of a process 18446612141208305664 and the PF_number is 107752278492160
Apr 20 03:10:56 iict-HP-EliteDesk-800-G1-SFF kernel: [  127.970924] The address of a process 18446612141215907840 and the PF_number is
107752278610944The address of a process 18446612141208305664 and the PF_number is 107752278492160The address of a process 18446612141208305664
and the PF_number is 107752278492160The address of a process 18446612141208305664 and the PF_number is 107752278492160The address of a process
18446612141208305664 and the PF_number is 107752278492160
Apr 20 03:10:56 iict-HP-EliteDesk-800-G1-SFF kernel: [  127.982150] The address of a process 18446612141194854400 and the PF_number is
107752278281984The address of a process 18446612141194854400 and the PF_number is 107752278281984The address of a process 18446612141194854400
and the PF_number is 107752278281984The address of a process 18446612141194854400 and the PF_number is 107752278281984
Apr 20 03:10:56 iict-HP-EliteDesk-800-G1-SFF kernel: [  127.982818] The address of a process 18446612141208305664 and the PF_number is
107752278492160The address of a process 18446612141215907840 and the PF_number is 107752278610944
Apr 20 03:10:56 iict-HP-EliteDesk-800-G1-SFF kernel: [  127.983405] The address of a process 18446612141208305664 and the PF_number is
107752278492160The address of a process 18446612141194854400 and the PF_number is 107752278281984The address of a process 18446612141194854400
and the PF_number is 107752278281984The address of a process 18446612141194854400 and the PF_number is 107752278281984The address of a process
18446612141194854400 and the PF_number is 107752278281984
Apr 20 03:10:56 iict-HP-EliteDesk-800-G1-SFF kernel: [  127.984498] The address of a process 18446612141208305664 and the PF_number is
107752278492160The address of a process 18446612141208305664 and the PF_number is 107752278492160
```

FIGURE 2: OUTPUT OF PRINTK STATEMENT WRITTEN IN PAGE_ALLOC.C

```
[   17.296531] (unsigned long)b: 18446612135826853888 & order: 0(unsigned long)b: 18446612135880818688 & order: 0
[   17.296569] (unsigned long)b: 18446612135826853888 & order: 0(unsigned long)b: 18446612135880818688 & order: 0
[   17.296584] (unsigned long)b: 18446612135826853888 & order: 0(unsigned long)b: 18446612135826853888 & order: 0(unsigned long)b:
18446612135826853888 & order: 0(unsigned long)b: 18446612135826853888 & order: 0(unsigned long)b: 18446612135826853888 & order: 0(unsigned
long)b: 18446612135826853888 & order: 0(unsigned long)b: 18446612135826853888 & order: 0(unsigned long)b: 18446612135826853888 & order: 0
(unsigned long)b: 18446612135826853888 & order: 0(unsigned long)b: 18446612135867559936 & order: 0(unsigned long)b: 18446612135867559936 &
order: 0(unsigned long)b: 18446612135867559936 & order: 0(unsigned long)b: 18446612135867559936 & order: 0(unsigned long)b:
18446612135867559936 & order: 0(unsigned long)b: 18446612135871873024 & order: 0(unsigned long)b: 18446612135871873024 & order: 0
[   17.297783] (unsigned long)b: 18446612133091586048 & order: 0(unsigned long)b: 18446612135926579200 & order: 0
[   17.298475] (unsigned long)b: 18446612135855505408 & order: 0(unsigned long)b: 18446612135855505408 & order: 0(unsigned long)b:
18446612135855505408 & order: 0(unsigned long)b: 18446612135855505408 & order: 0(unsigned long)b: 18446612135855505408 & order: 0(unsigned
long)b: 18446612135855505408 & order: 0(unsigned long)b: 18446612135855505408 & order: 0(unsigned long)b: 18446612135855505408 & order: 0
(unsigned long)b: 18446612135855505408 & order: 0(unsigned long)b: 18446612135855505408 & order: 0(unsigned long)b: 18446612135855505408 &
order: 0(unsigned long)b: 18446612135855505408 & order: 0(unsigned long)b: 18446612135855505408 & order: 0(unsigned long)b:
18446612135855505408 & order: 0(unsigned long)b: 18446612135855505408 & order: 0(unsigned long)b: 18446612135886573568 & order: 0(unsigned
long)b: 18446612135886573568 & order: 0(unsigned long)b: 18446612135886573568 & order: 0(unsigned long)b: 18446612135886573568 & order: 0
(unsigned long)b: 18446612135886573568 & order: 0(unsigned long)b: 18446612135883919360 & order: 0(unsigned long)b: 18446612135883919360 &
order: 0
[   17.300348] (unsigned long)b: 18446612133097684992 & order: 0(unsigned long)b: 18446612135858749440 & order: 0
[   17.301273] (unsigned long)b: 18446612135832801280 & order: 0(unsigned long)b: 18446612135832801280 & order: 0(unsigned long)b:
18446612135832801280 & order: 0(unsigned long)b: 18446612135832801280 & order: 0(unsigned long)b: 18446612135832801280 & order: 0(unsigned
long)b: 18446612135832801280 & order: 0(unsigned long)b: 18446612135832801280 & order: 0(unsigned long)b: 18446612135832801280 & order: 0
(unsigned long)b: 18446612135832801280 & order: 0(unsigned long)b: 18446612135832801280 & order: 0(unsigned long)b: 18446612135832801280 &
order: 0(unsigned long)b: 18446612135826796544 & order: 0(unsigned long)b: 18446612135826796544 & order: 0(unsigned long)b:
18446612135826796544 & order: 0(unsigned long)b: 18446612135826796544 & order: 0(unsigned long)b: 18446612135826796544 & order: 0(unsigned
long)b: 18446612135870783488 & order: 0(unsigned long)b: 18446612135870783488 & order: 0(unsigned long)b: 18446612135870783488 & order: 0
(unsigned long)b: 18446612135870783488 & order: 0(unsigned long)b: 18446612135870783488 & order: 0(unsigned long)b: 18446612135870783488 &
order: 0
[   17.302262] (unsigned long)b: 18446612135870784384 & order: 0(unsigned long)b: 18446612135867817984 & order: 0(unsigned long)b:
18446612135867817984 & order: 0(unsigned long)b: 18446612135867817984 & order: 0(unsigned long)b: 18446612135867817984 & order: 0(unsigned
long)b: 18446612135863717888 & order: 0(unsigned long)b: 18446612135863717888 & order: 0(unsigned long)b: 18446612135832731648 & order: 0
(unsigned long)b: 18446612135832731648 & order: 0(unsigned long)b: 18446612135832731648 & order: 0(unsigned long)b: 18446612135928832000 &
order: 0(unsigned long)b: 18446612135928832000 & order: 0(unsigned long)b: 18446612133096124416 & order: 0(unsigned long)b:
18446612133096124416 & order: 0(unsigned long)b: 18446612133096124416 & order: 0(unsigned long)b: 18446612133096124416 & order: 0(unsigned
long)b: 18446612133096124416 & order: 0(unsigned long)b: 18446612133096124416 & order: 0(unsigned long)b: 18446612135826825216 & order: 0
(unsigned long)b: 18446612135826825216 & order: 0(unsigned long)b: 18446612135826825216 & order: 0(unsigned long)b: 18446612135826825216 &
```

FIGURE 3: OUTPUT OF THE PRINTK STATEMENT WRITTEN IN THE SLOB.C

References:

[1] http://census-labs.com/news/2012/01/03/linux-kernel-heap-exploitation/

[2] http://en.wikipedia.org/wiki/SLOB

[3] http://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=4&cad=rja&uact=8&ved=0CDEQFjAD&url=http%3A%2F%2Fwww.vsecurity.com%2Fdownload%2Fpapers%2Fslobexploitation.pdf&ei=IXdCVaWJPOarmAWt94GADw&usg=AFQjCNH9S0rq2HnSxJ8M

gSp2Evb_1SMtcw&sig2=rAIE1DVORrNYMgYEeQoCPw&bvm=bv.92189499,d.dGY