

OPERATING SYSTEMS

CS342

Project 3B

Ramazan Mert Cinar – Yaman Yagiz Tasbag
21601985 - 21601639
Section 2 – Section 2

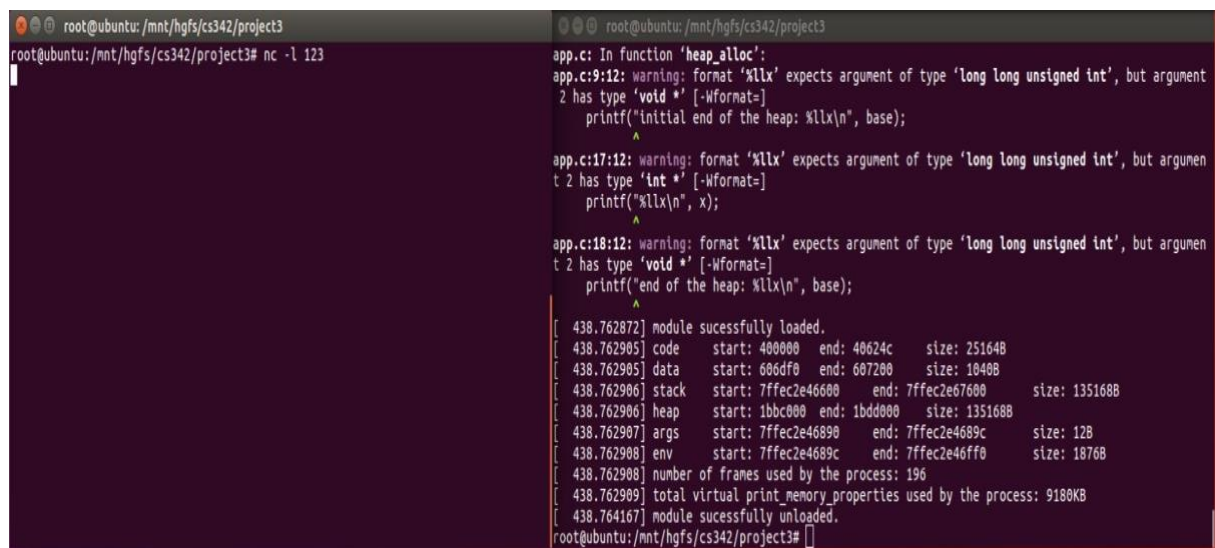
Part B)

Step 1

We have named the first module as “print_memory_properties”. In this module, we print the information about a process containing stack, heap, data, code, arguments, environment variables and rss. We have passed the process id as a parameter from the command line and defined the parameter in the code as follows:

```
module_param(processid, int, 0);
```

To test the program we have run a netcat program (using nc -l command) and found its process id using “ps aux | grep netcat”. Finally, we loaded the module using “insmod” command and see the output using “dmesg”. A sample output for a netcat process is shown in Figure 1.



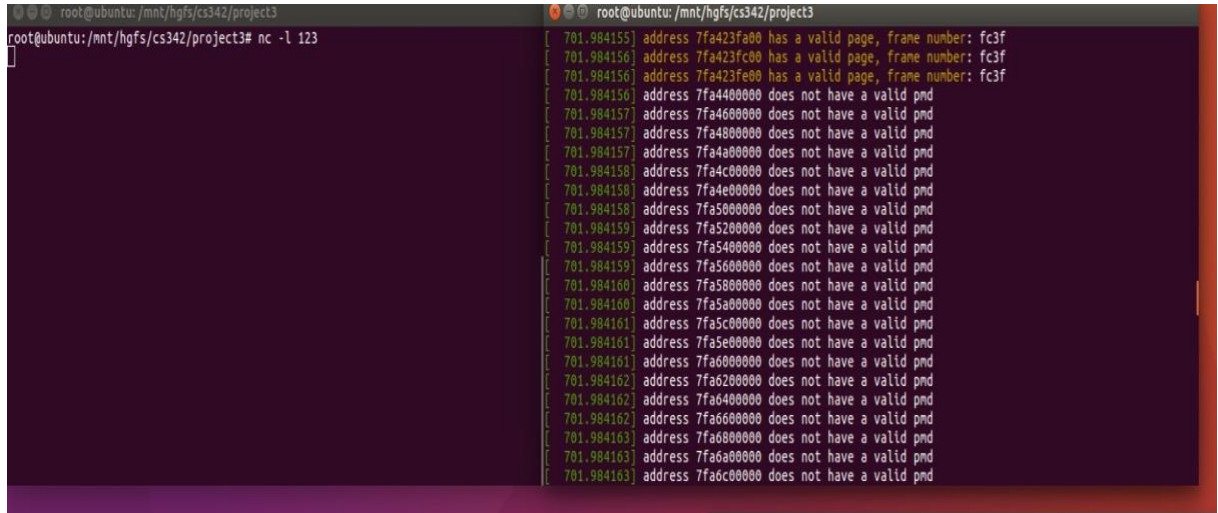
```
root@ubuntu:/mnt/hgfs/cs342/project3# nc -l 123
app.c: In function 'heap_alloc':
app.c:9:12: warning: format '%llx' expects argument of type 'long long unsigned int', but argument 2 has type 'void *' [-Wformat=]
    printf("initial end of the heap: %llx\n", base);
    ^
app.c:17:12: warning: format '%llx' expects argument of type 'long long unsigned int', but argument 2 has type 'int *' [-Wformat=]
    printf("%llx\n", x);
    ^
app.c:18:12: warning: format '%llx' expects argument of type 'long long unsigned int', but argument 2 has type 'void *' [-Wformat=]
    printf("end of the heap: %llx\n", base);
    ^
[ 438.762872] module successfully loaded.
[ 438.762905] code start: 400000 end: 40624c size: 25164B
[ 438.762905] data start: 606df0 end: 607200 size: 1040B
[ 438.762906] stack start: 7ffec2e46000 end: 7ffec2e67600 size: 135168B
[ 438.762906] heap start: 1bbc000 end: 1bdd000 size: 135168B
[ 438.762907] args start: 7ffec2e46890 end: 7ffec2e4689c size: 12B
[ 438.762908] env start: 7ffec2e4689c end: 7ffec2e46ff0 size: 1876B
[ 438.762908] number of frames used by the process: 196
[ 438.762909] total virtual print_memory_properties used by the process: 9180KB
[ 438.764107] module successfully unloaded.
root@ubuntu:/mnt/hgfs/cs342/project3#
```

Figure 1

Step 2

We have written another module to traverse the page tables. Similar to the first one, it takes an argument indicating the process id. Using the api of linux kernel it traverses the all levels of page tables (pgd, p4d, pud, pmd, pte). To traverse the page tables, we are basically looping all possible indexes (using library parameter, e.g. PTRS_PER_P4D). If an address is not valid, the module logs it to logfile and continues to the next iteration.

To test the second module, we have followed the same path and ran a netcat process. We again observed the output of the module using “dmesg”. An output snippet of the module can be seen in Figure 2.



```
root@ubuntu: /mnt/hgfs/cs342/project3
root@ubuntu: /mnt/hgfs/cs342/project3# nc -l 123
[ 701.984155] address 7fa423fa00 has a valid page, frame number: fc3f
[ 701.984156] address 7fa423fc00 has a valid page, frame number: fc3f
[ 701.984156] address 7fa423fe00 has a valid page, frame number: fc3f
[ 701.984156] address 7fa4400000 does not have a valid pnd
[ 701.984157] address 7fa4600000 does not have a valid pnd
[ 701.984157] address 7fa4800000 does not have a valid pnd
[ 701.984157] address 7fa4a00000 does not have a valid pnd
[ 701.984158] address 7fa4c00000 does not have a valid pnd
[ 701.984158] address 7fa4e00000 does not have a valid pnd
[ 701.984158] address 7fa5000000 does not have a valid pnd
[ 701.984159] address 7fa5200000 does not have a valid pnd
[ 701.984159] address 7fa5400000 does not have a valid pnd
[ 701.984159] address 7fa5600000 does not have a valid pnd
[ 701.984160] address 7fa5800000 does not have a valid pnd
[ 701.984160] address 7fa5a00000 does not have a valid pnd
[ 701.984161] address 7fa5c00000 does not have a valid pnd
[ 701.984161] address 7fa5e00000 does not have a valid pnd
[ 701.984161] address 7fa6000000 does not have a valid pnd
[ 701.984162] address 7fa6200000 does not have a valid pnd
[ 701.984162] address 7fa6400000 does not have a valid pnd
[ 701.984162] address 7fa6600000 does not have a valid pnd
[ 701.984163] address 7fa6800000 does not have a valid pnd
[ 701.984163] address 7fa6a00000 does not have a valid pnd
[ 701.984163] address 7fa6c00000 does not have a valid pnd
```

Figure 2

Step 3

We have implemented a menu system for this part. It is up to the user to allocate memory from heap or stack and the size of the allocation is again up to the user. The process id of the program is taken using the function “getpid()”. The system works as follows:

- Press 1 to allocate from heap
 - o Enter the amount of allocation in terms of bytes
 - o Print heap information
 - o Go to top menu
- Press 2 to allocate from stack
 - o Enter the number of times to call the recursive function
 - o Print the stack information before starting to pop functions from stack
 - o Go to top menu
- Press 0 to exit

We have observed the output using the second module that we have written and an example of it can be found in Figure 3.

```

root@ubuntu: /mnt/hgfs/cs342/project3
yytasbag@ubuntu:~$ cd /mnt/hgfs/cs342/project3
yytasbag@ubuntu: /mnt/hgfs/cs342/project3$ sudo su
[sudo] password for yytasbag:
Sorry, try again.
[sudo] password for yytasbag:
root@ubuntu: /mnt/hgfs/cs342/project3# ./app
process id: 2056
press 1 to use heap (malloc)
press 2 to use stack (recursive)
press 0 to exit
1
Initial end of the heap: 24f9000
enter amount of bytes to allocate
10000
24daba0
end of the heap: 24f9000
press 1 to use heap (malloc)
press 2 to use stack (recursive)
press 0 to exit
0

[ 15.589899] audit: type=1400 audit(1544646495.412:10): apparmor="STATUS" operation="profile_
load" profile="unconfined" name="/usr/bin/evince" pid=717 comm="apparmor_parser"
[ 15.589902] audit: type=1400 audit(1544646495.412:11): apparmor="STATUS" operation="profile_
load" profile="unconfined" name="/usr/bin/evince//sanitized_helper" pid=717 comm="apparmor_pars
er"
[ 17.155443] Adding 998396k swap on /dev/sda5. Priority:-2 extents:1 across:998396k FS
[ 26.177748] IPv6: ADDRCONF(NETDEV_UP): ens33: link is not ready
[ 26.183359] IPv6: ADDRCONF(NETDEV_UP): ens33: link is not ready
[ 26.185108] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: None
[ 26.186125] IPv6: ADDRCONF(NETDEV_CHANGE): ens33: link becomes ready
[ 113.622684] print_memory_properties: loading out-of-tree module taints kernel.
[ 113.622729] print_memory_properties: module verification failed: signature and/or required k
ey missing - tainting kernel
[ 113.623126] module successfully loaded.
[ 113.623159] code start: 400000 end: 400d0c size: 33408
[ 113.623160] data start: 600e10 end: 601068 size: 6008
[ 113.623161] stack start: 7ffc06049420 end: 7ffc0606a420 size: 1351688
[ 113.623161] heap start: 24d8000 end: 24f9000 size: 1351688
[ 113.623162] args start: 7ffc0604a89a end: 7ffc0604a8a0 size: 68
[ 113.623162] env start: 7ffc0604a8a0 end: 7ffc0604aff2 size: 18748
[ 113.623163] number of frames used by the process: 196
[ 113.623163] total virtual print_memory_properties used by the process: 4352KB
[ 113.624963] module successfully unloaded.
root@ubuntu: /mnt/hgfs/cs342/project3#

```

Figure 3

Analysis

We have made several experiments to obtain various results. The plot and table below (shown in Figure 4 and 5) demonstrate the increase in stack size considering the number of times that the recursive function called.

# Recursion	Stack Size
0	135168
100000	204800
200000	299008
300000	393216
400000	491520
500000	585728
600000	684032
700000	778240
800000	876544
900000	970752

Figure 4

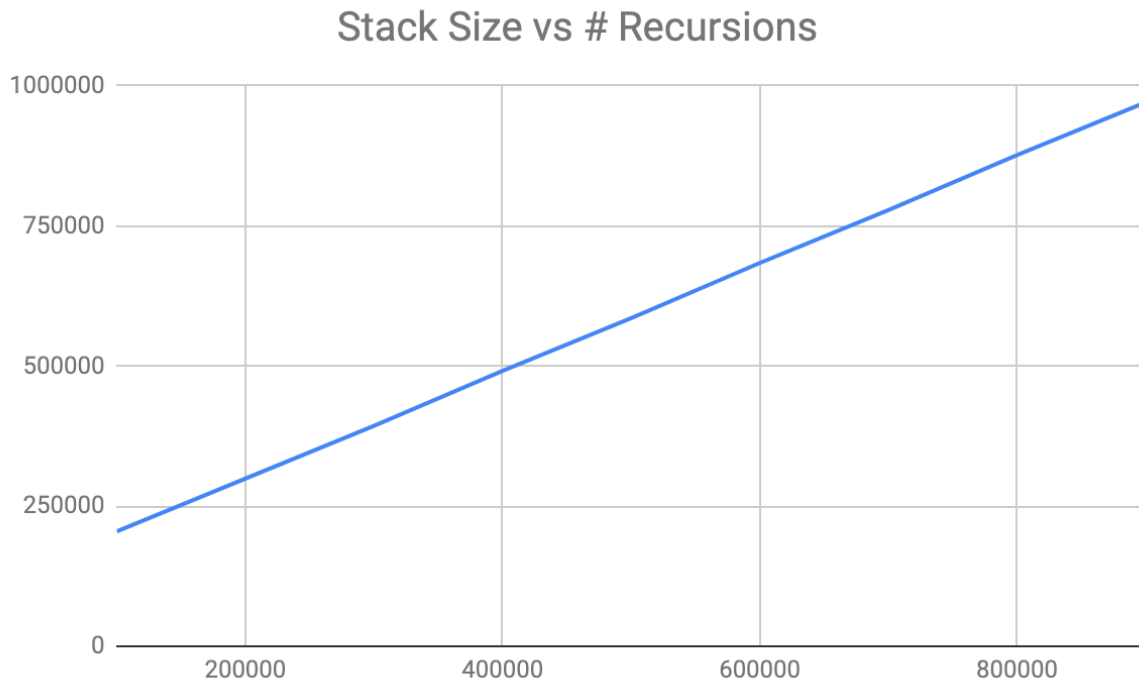


Figure 5

Another plot that we have produced is comparing the heap size with the amount of allocation in terms of bytes. The results can be seen in the plot and table below (Figure 6 and 7).

# Bytes Allocated	Heap Size
100000	135168
200000	135168
300000	270336
400000	405504
500000	405504
600000	540672

700000	675840
800000	811008
900000	811008
1000000	946176

Figure 6

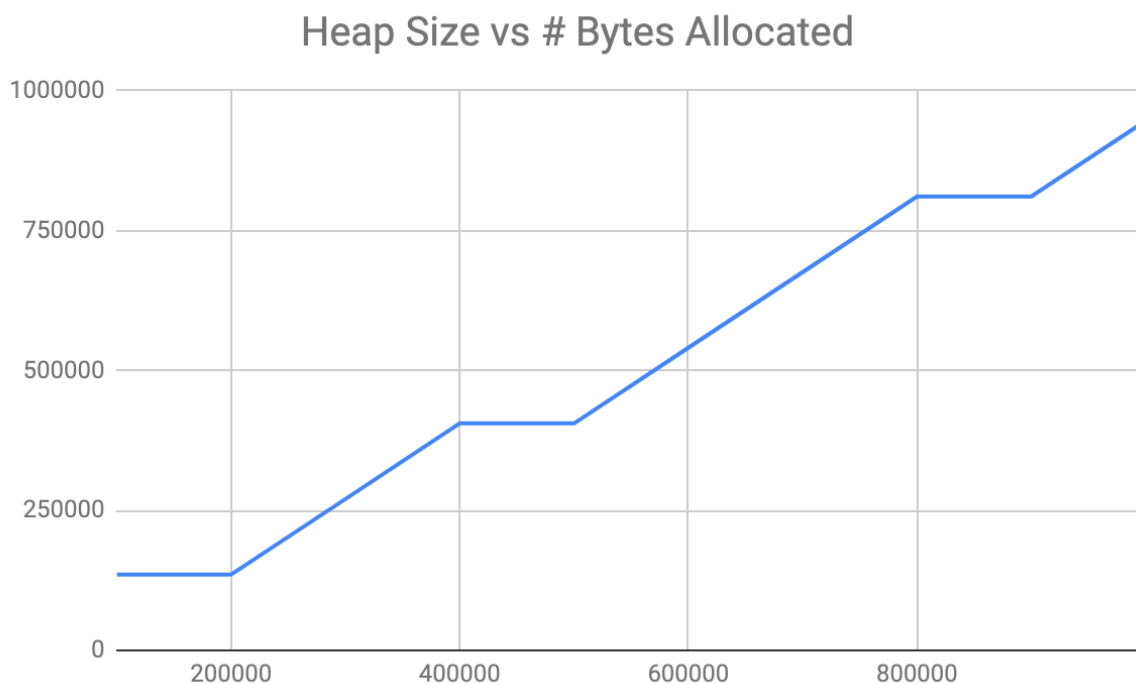


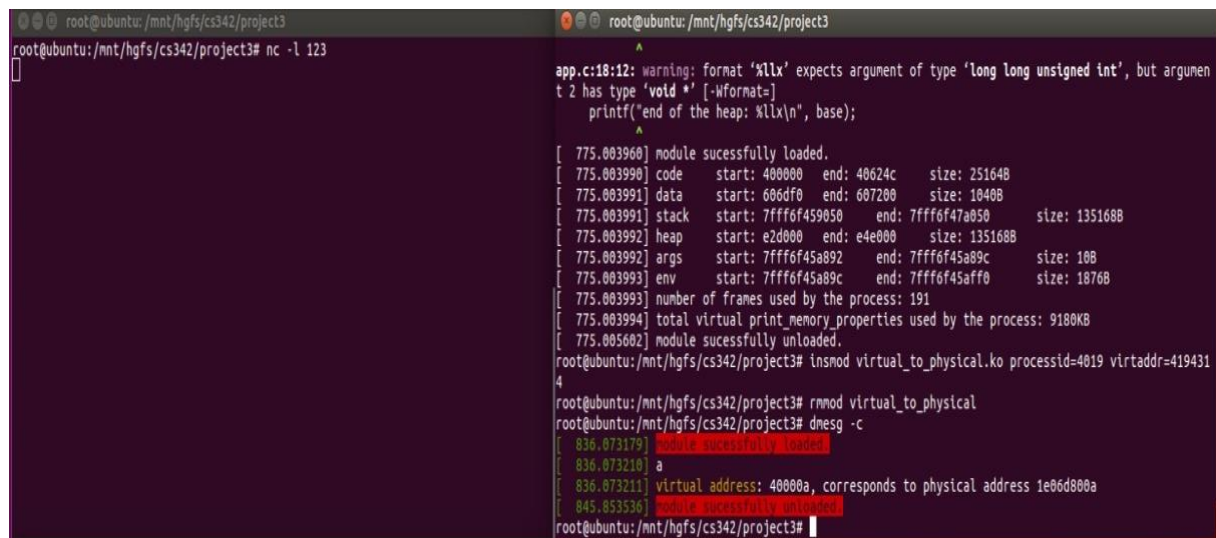
Figure 7

We observe that by allocating memory, the stack size is always increased, however, the heap size might remain the same. This is due to the implementation of malloc() function and also if there is enough space the size can remain the same.

Step 4

We have written another named “virtual_to_physical”. It takes two arguments: First one indicates the process id and the second one is a virtual address. Using the techniques that we have used for traversing the page tables, we have walk them using the virtual address as the input. If the program encounters an error, it is printed to the kernel log, otherwise the physical

address corresponding to the given virtual one, is printed to the kernel log. A sample output can be seen in Figure 8.



```
root@ubuntu: /mnt/hgfs/cs342/project3
root@ubuntu: /mnt/hgfs/cs342/project3# nc -l 123
[
app.c:18:12: warning: format '%llx' expects argument of type 'long long unsigned int', but argumen
t 2 has type 'void *' [-Wformat=]
    printf("end of the heap: %llx\n", base);
    ^
[
[ 775.003960] module successfully loaded.
[ 775.003990] code    start: 400000    end: 40624c    size: 25164B
[ 775.003991] data    start: 606df0    end: 607200    size: 1040B
[ 775.003991] stack   start: 7fff6f459050    end: 7fff6f47a050    size: 135168B
[ 775.003992] heap    start: e2d000    end: e4e000    size: 135168B
[ 775.003992] args    start: 7fff6f45a892    end: 7fff6f45a89c    size: 10B
[ 775.003993] env     start: 7fff6f45a89c    end: 7fff6f45aff0    size: 1876B
[ 775.003993] number of frames used by the process: 191
[ 775.003994] total virtual print memory properties used by the process: 9180KB
[ 775.005602] module successfully unloaded.
root@ubuntu: /mnt/hgfs/cs342/project3# insmod virtual_to_physical.ko processid=4019 virtaddr=419431
4
root@ubuntu: /mnt/hgfs/cs342/project3# rmmod virtual_to_physical
root@ubuntu: /mnt/hgfs/cs342/project3# dmesg -c
[ 836.073179] module successfully loaded
[ 836.073210] a
[ 836.073211] virtual address: 40000a, corresponds to physical address 1e06d800a
[ 845.853536] module successfully unloaded
root@ubuntu: /mnt/hgfs/cs342/project3#
```

Figure 8