CS342 –Project 3

Report

Arif Can Terzioğlu

21302061

Environment

My main system works on Windows 10 OS. On the hardware side, computer processer is 4 cores Intel i7 4700HQ 2.Ghz series. Ram amount is 16GB.
I installed Ubuntu 16.04 64bit via virtual machine which course assistants provided us. I used kernel version: 4.10.0-28-generic.

Step 1

As project assignment stated, step 1 was a great opportunity get into module concept. It has some obvious differences from regular C program. Modules have a strong connection with kernel version. While I was writing them, I got a lot of problem due to kernel versions. All references and code examples were based on different version of kernel. As kernel version changes, structure of libraries and functions may vary as I understand. For example, at the beginning I tried to use end_of_stack() function in kernel 4.13.x. However, its' make file did not complete compilation. However, when I tried it on 4.10, it was successfully done. Also, place of functions can be changed between library files. Therefore; while developing a module, keeping touch with changes on Linux kernel is really necessary. I handled this situation by working on kernel 4.10.0-28 for modules. If we want to select different linux kernel version, we can select another version at the booting screen pressing shift button. Of course, build and run a new kernel version is another option.

Since modules do not permit to use regular C libraries, usage of printf function is not possible. I recorded all outputs with printk function and display them on the terminal with dmesg function. Printk function takes different priority parameters such as KERN_INFO, KERN_ALERT and etc. Also, different than regular C programs main, modules takes init_module and cleanup_module for initialization and termination respectively.

At the end of step 1, I learnt that modules are used to add new functionality directly into the kernel.

Step 2

The basic usage of modules were learnt in the step 1 such as insmode, object make file, rmmod, dmesg, dmesg –c and etc. This step was more about the memory content and management of the process

Part A

In this part, supplementary notes and fourth reference of the project assignment helped me a lot. Basically what I did is that I create a task_struct then equalize the current task of module. Then starting from that process I iterate over the PCB list until I find the desired “pid” value.

Finding the desired process was very straightforward and fourth reference of the project assignment enabled me to get into memory management part and mm_struct structure.

Part B

Theory

As we learnt in the process chapter of the book, each process divided into some segments such as stack, code, heap, memory mapping segment and bss segment. The virtual memory layout of a process is a list of virtual memory regions. These regions constitute the virtual address space of the process.

As I stated, every process has an associated struct task_struct makes the processes unique in terms of identity and memory management. I want to mention two of member of task_struct that are vm_area_struct and mm_struct.

Each vm_area_struct contain information about a single region in a contigious range of virtual addresses. An instance of it describes a memory area such as start and end addresses. I used vm_area_struct to find the start and end addresses.

“mm_struct” contains information about general memory layout. In other word, mm_struct provides us to kind of executive summary of memory parts.

Implementation

Mm_struct has a list of vm_area_struct that belongs to the process. I saw the example that iterates through the list and prints the start and end addresses of each in reference four. I used that sample for my code. To find their size, I basically subtract start from end. I asked Ibrahim Hoca, he said probably correct but you should do a more research to be sure. I divided the sizes with page size and I saw they are multiple of page size (4096). In that way I was able to be sure.

When I looked at the detailed structure of the mm_struct on linux kernel v4.10, I realized some unsigned long start addresses name start_code, start_data and etc. I printed the code, stack and all

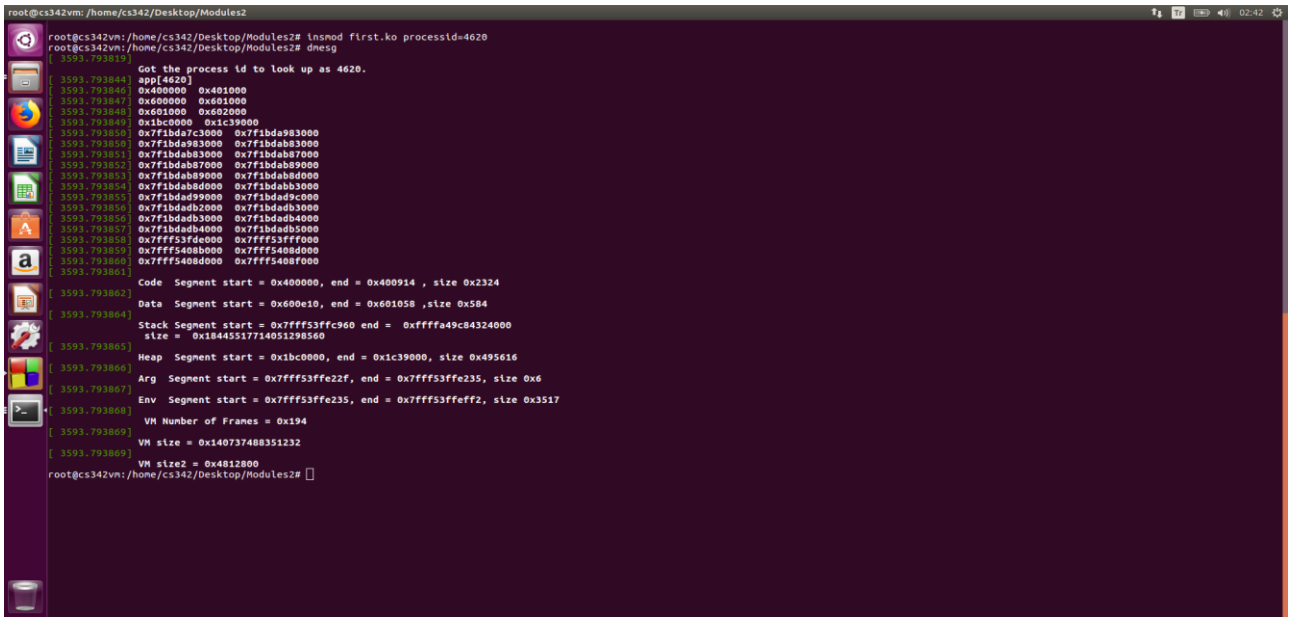
heap segments via these addresses. However, when I checked the results from cat /proc/pid/maps, some of values matched and some of them did not match. More interestingly, addresses are very close to each other but not exactly the same. To match the exact answer, I decided to print each data, stack and other segments through their vm_areas. I again iterated over the vm_area_struct and compare their start and end address with unsigned long addresses in the mm_struct. When I found the correct interval, print them.

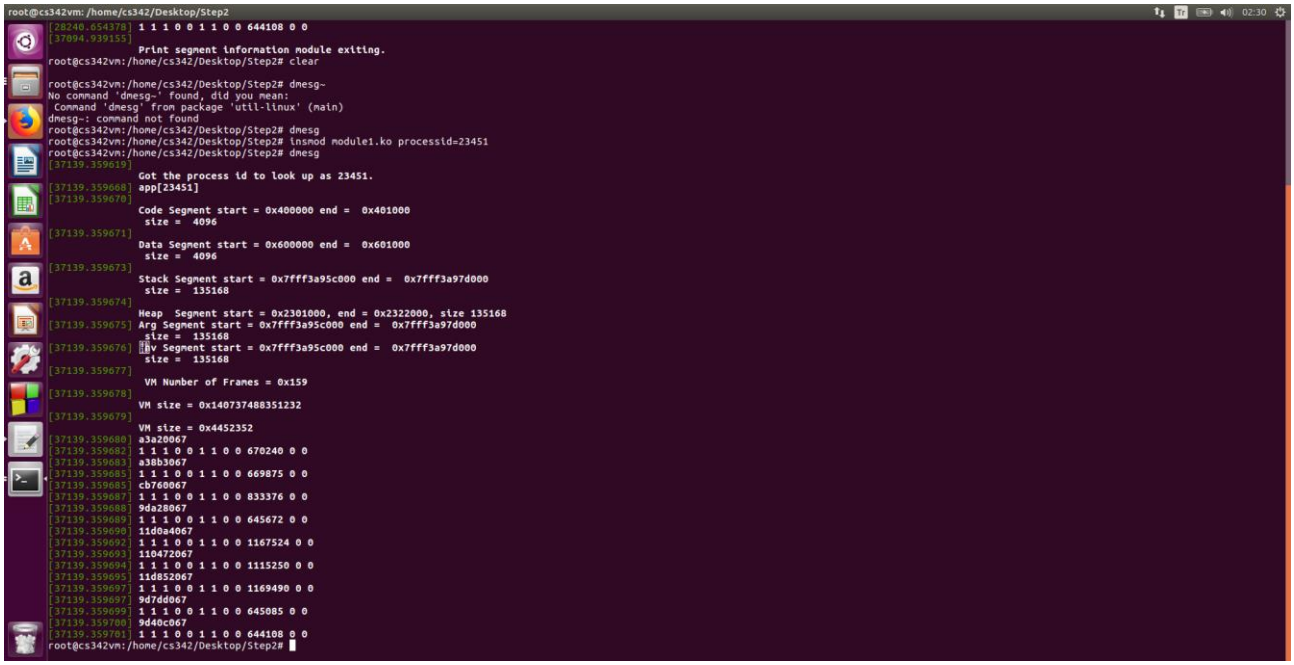
Stack was the problematic part in terms of finding its end. Again like other segment, I found it end segment by iterating over the vm_areas. Before iteration method, I used method called end_of_stack(task_struct). It did not give the matching result with the cat /proc/pid/maps. One confusing point is still each thread's stack regions.

I found the number of frames by using get_mm_rss that I saw on the piazza.

At the beginning I supposed that total_vm gives the total memory used. However, when I read the piazza I learnt I returns the total number of pages used. Then, I multiply it with page size to get total virtual memory used. However, another parameter of mm_struct attracts my attention that is task_size. In the source code, with comments it says size of task vm space. So,I gave it a shot to try. However; task_size gave really strange numbers that I could not figure out. I am still confused about that parameter. I decided to leave multiplication with total_vm. Because at the step 3, increase in heap size was equal to increase in multiplication of total_vm and page size.

Some Outputs



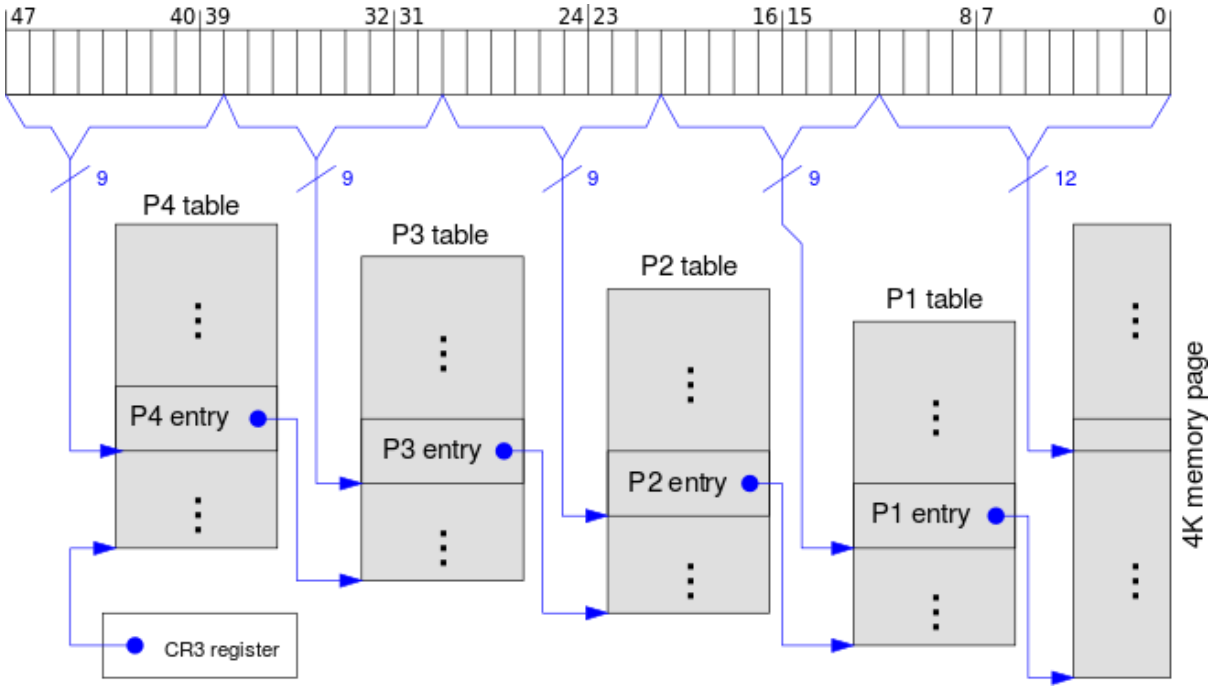


The first VM size is with task_size.

Part C

Theory and Implementation

I started to this part by observing the four level paging structures. Since we are only interested in only top level page table, first 9 bit of our 48 bit virtual address is our concern. While I was thinking about how can I parse these 9-bits and reach the top level page table. I learnt from reference 7 that mm_struct has a pointer pgd that points to the top level page table. In this table, each entry corresponds to second level page table.



This image is taken from the page table that some students share on the piazza. That links helped me a lot in terms of understanding. I investigated some approaches on the piazza about this part. Some methods use pgd_offset function to get the table entries. As far as I understand, we need to increase the address by 2^{39} to increase PGD address (2^{48} unique address/ 2^{9} top level pages). From this point, we can understand, we can iterate over pgd_t instance to access the each top level page. Due to offset we have 512 entries. For getting the each page entry I used pgd_val() function which is easier than offset function. I saw that function in supplementary notes. I printed each of entry that's valid bit is not 0. Basically, I iterated over all pdg entries until 511 as an index, get their page table entry value then print it.

Next stage was parsing. To understand what I need to parse, I had to talk and read a lot. This discussion was an ambiguity among our friends also. Although Intel document seems very

straightforward, to understand what these bits or bit was hard stage. I shifted and masked the entries to get the desired bits.

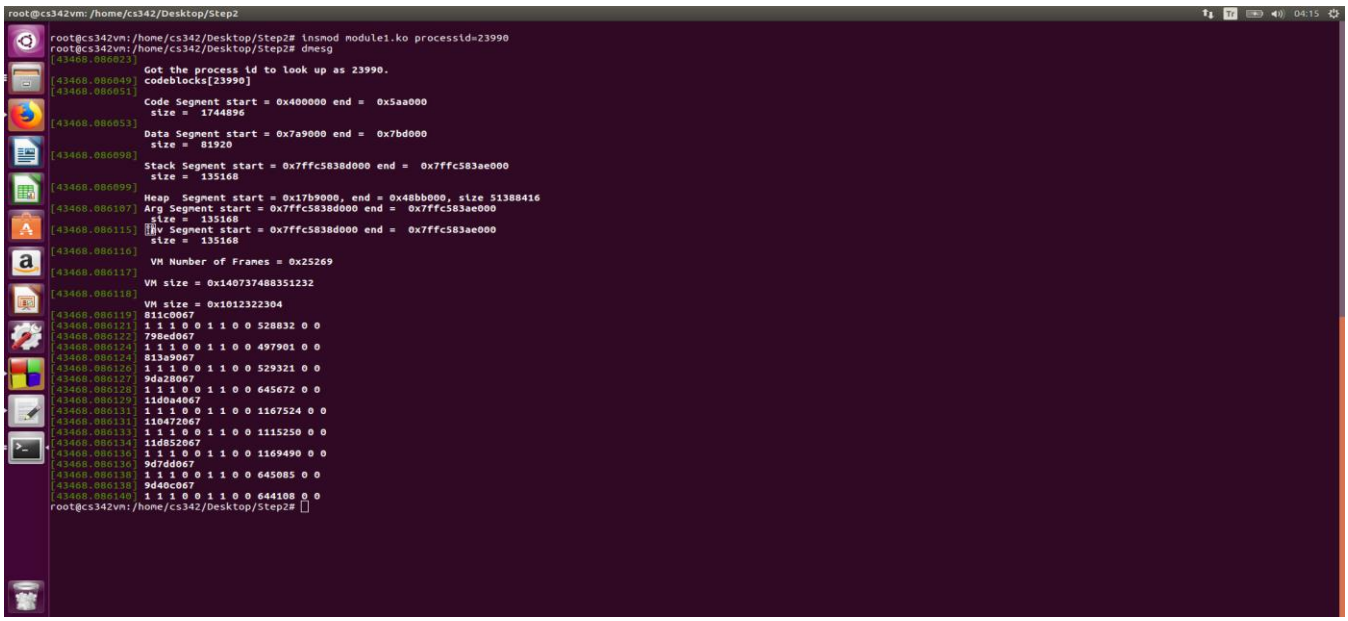
As a result, I am able to print these bits on the Intel document Table 4-14.

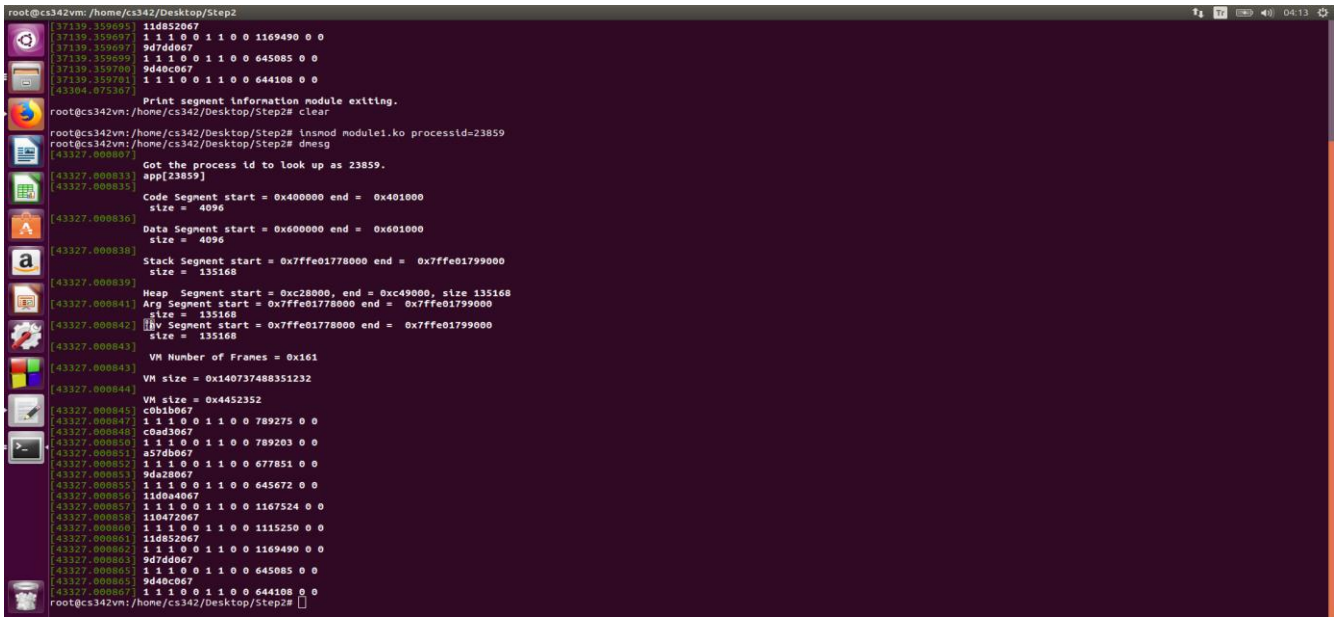
Present bit 0: If entry is used it must be 1.

Read/Write 1: If 0, writes may not be allowed and etc.

I learnt each of these bits and bit meaning from the document.

Some Outputs





Step 3

Heap Allocation and Deallocation

Input Allocation sizes: 120KB, 180KB, 300KB, 360KB and 420KB

Note: For each allocation, program is started newly. I mean increases are calculated based on initial size.

It needs to point out that we need to allocate memory from heap part by part. I mean as stated in the piazza, “if you call malloc() with a request higher than 135KB, then it will call mmap() systemcall and satisfy your request by mapping ANOTHER memory region outside of heap. Thus, heap will not enlarge.”

Thus, to enlarge the heap, I used sequence of 60KB allocations.

Results

I allocated memory with “malloc” function with given sizes above. Our initial heap size is around 135KB.

Exact number is 135168 byte. Important point is that this size is multiple of our page size that is 135168 / 4096 = 33. It means we have 33 pages initially.

Our initial vm space size is 4,452,352 that is 1087 pages total.

When we allocate 120KB initially, since current heap is enough, heap was not enlarged.

When we allocate 180KB, our heap size becomes 315392 byte. Increase is 180224 byte 44 pages in both total size and heap size. We can see that increase equals to allocation amount.

If we allocate 300KB, sizes does not change.

When we allocate 360KB, our heap size becomes 495616 byte. Increase is 360448 byte 88 pages pages in both total size and heap size.

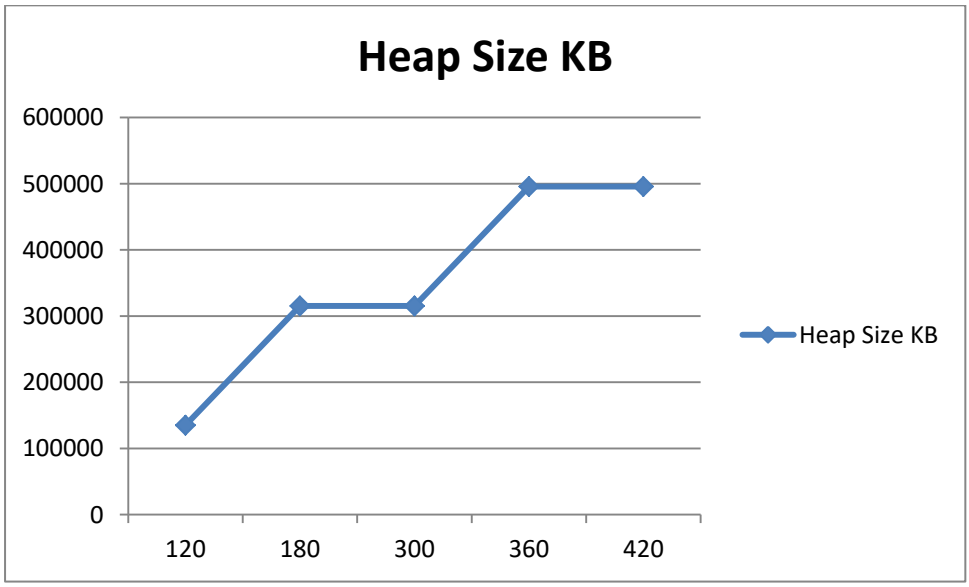
If we allocate 420KB, sizes are the same with 360KB. Changes did not occur my guess is our current heap size in 360 KB 495616 which is also enough for 420KB.

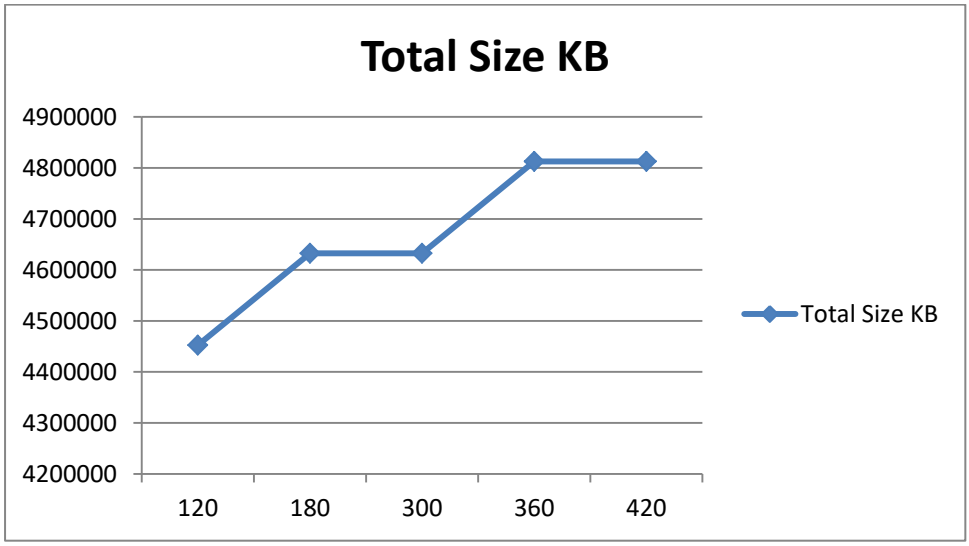
Number of page count increase equals to number of page count increase in total size.

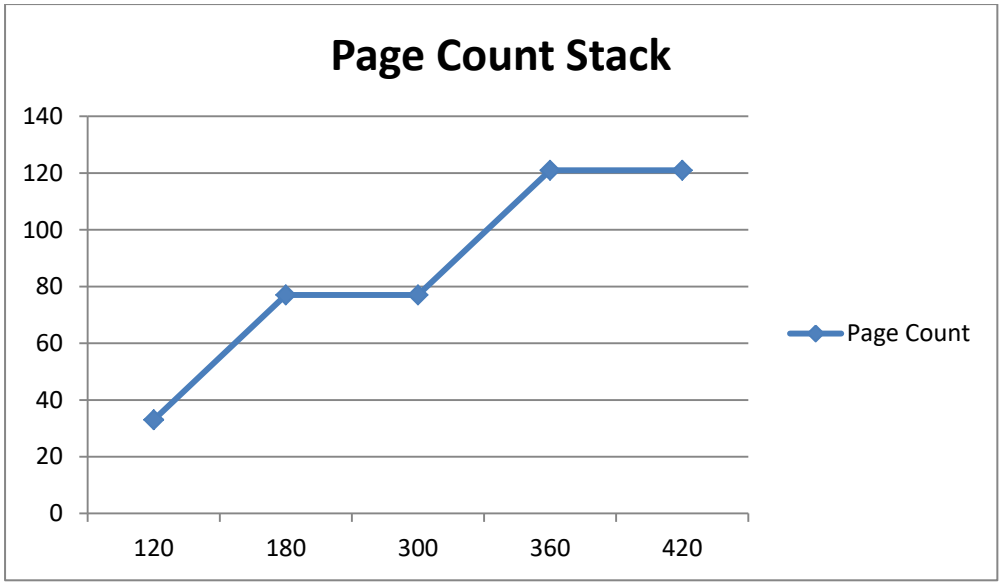
Deallocation

On the deallocation part, when we deallocate the memory heap turned to its initial size that is 135KB. In the experiment, when I deallocate all of the heap areas i have allocated, at the end the heap decreases back to its default size.

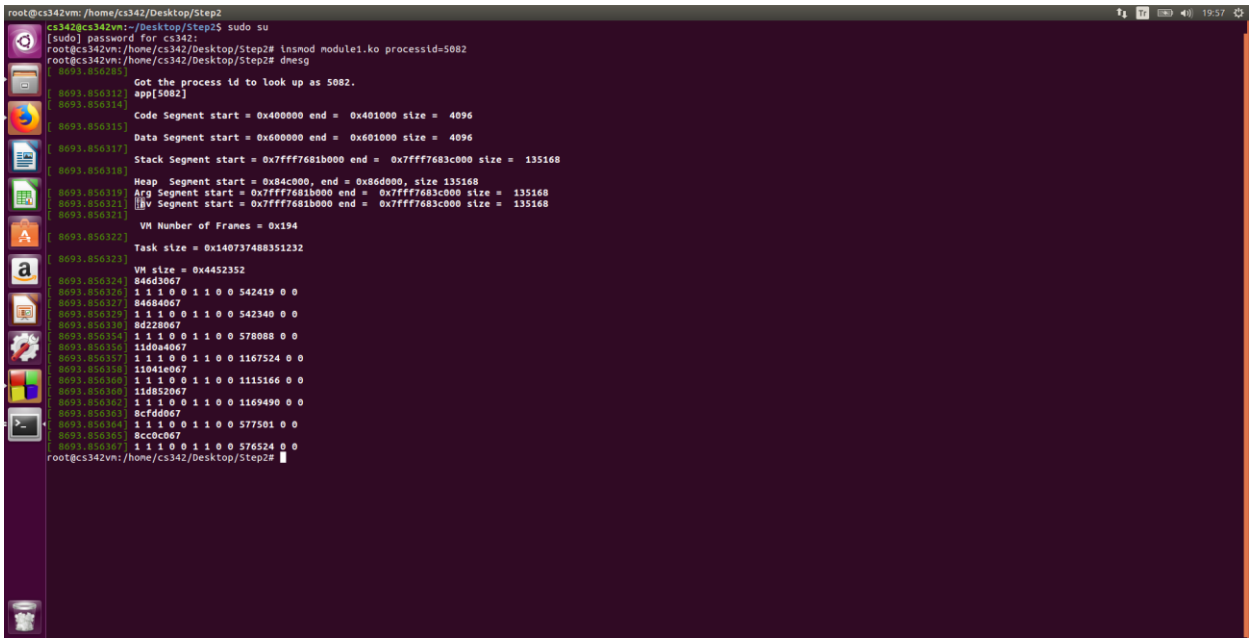
Graphics

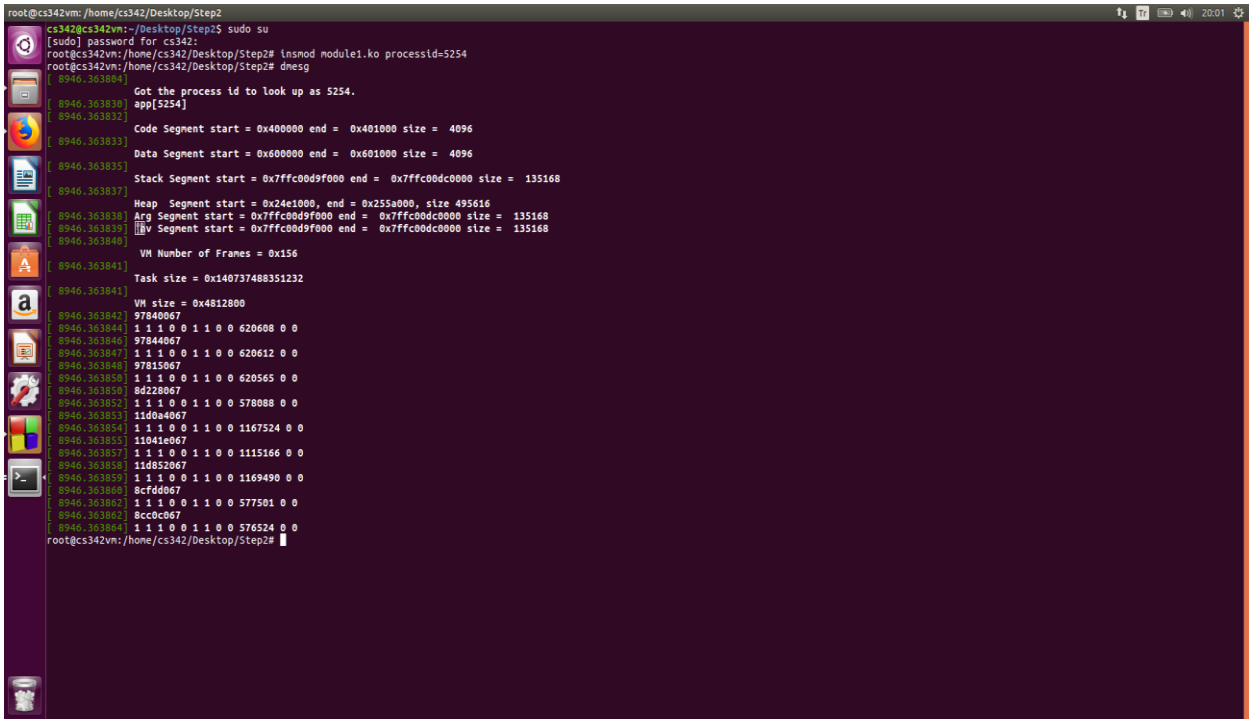


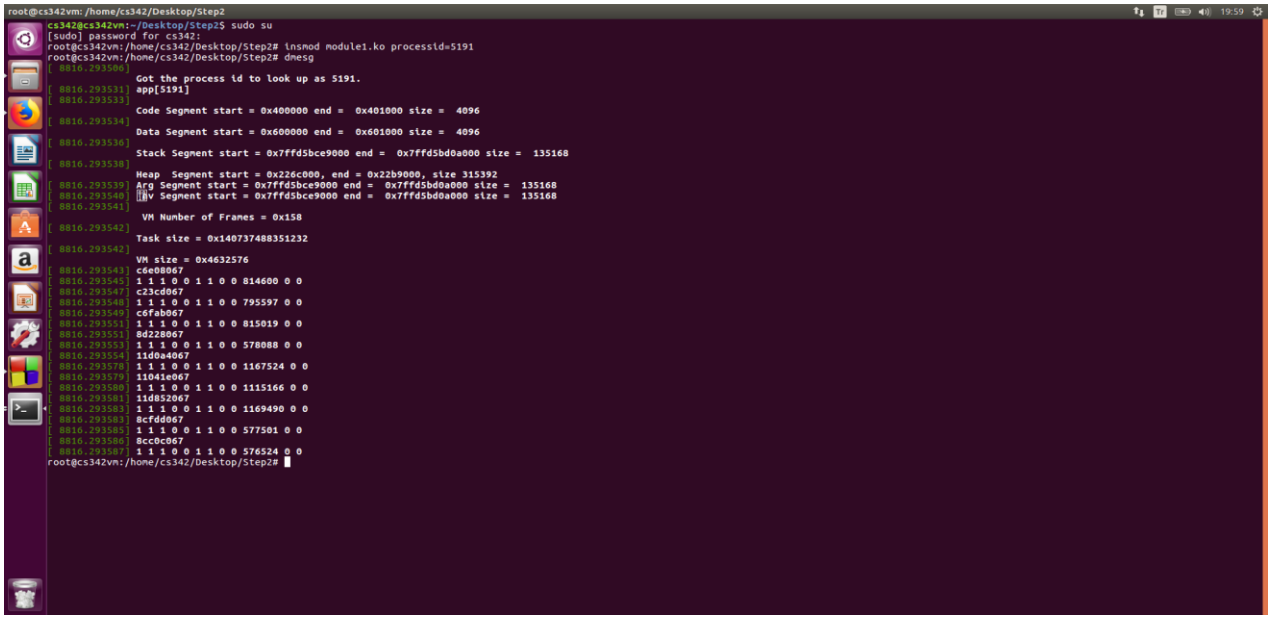




Some Outputs




root@cs342vm: /home/cs342/Desktop/Step2
20:01
cs342@cs342vm:~/Desktop/Step2$ sudo su
[sudo] password for cs342:
root@cs342vm:/home/cs342/Desktop/Step2# insmod module1.ko processId=5254
root@cs342vm:/home/cs342/Desktop/Step2# dmesg
[8946.363804]
Got the process id to look up as 5254.
[8946.363830] app[5254]
[8946.363832]
Code Segment start = 0x400000 end = 0x401000 size = 4096
[8946.363833]
Data Segment start = 0x600000 end = 0x601000 size = 4096
[8946.363835]
Stack Segment start = 0x7ffc00d9f000 end = 0x7ffc00dc0000 size = 135168
[8946.363837]
Heap Segment start = 0x24e1000, end = 0x255a000, size 495616
[8946.363838] Arg Segment start = 0x7ffc00d9f000 end = 0x7ffc00dc0000 size = 135168
[8946.363839] Env Segment start = 0x7ffc00d9f000 end = 0x7ffc00dc0000 size = 135168
[8946.363840]
VM Number of Frames = 0x156
[8946.363841]
Task size = 0x140737488351232
[8946.363841]
VM size = 0x4812800
[8946.363842] 97840067
[8946.363844] 1 1 1 0 0 1 1 0 0 620608 0 0
[8946.363846] 97844067
[8946.363847] 1 1 1 0 0 1 1 0 0 620612 0 0
[8946.363848] 97815067
[8946.363850] 1 1 1 0 0 1 1 0 0 620565 0 0
[8946.363850] 8d228067
[8946.363852] 1 1 1 0 0 1 1 0 0 578088 0 0
[8946.363853] 11d0a4067
[8946.363854] 1 1 1 0 0 1 1 0 0 1167524 0 0
[8946.363855] 11041e067
[8946.363857] 1 1 1 0 0 1 1 0 0 1115166 0 0
[8946.363858] 11d852067
[8946.363859] 1 1 1 0 0 1 1 0 0 1169490 0 0
[8946.363860] 8cfdd067
[8946.363862] 1 1 1 0 0 1 1 0 0 577501 0 0
[8946.363862] 8cc0c067
[8946.363864] 1 1 1 0 0 1 1 0 0 576524 0 0
root@cs342vm:/home/cs342/Desktop/Step2#


root@cs342vm: /home/cs342/Desktop/Step2
19:59
cs342@cs342vm:~/Desktop/Step2$ sudo su
[sudo] password for cs342:
root@cs342vm:/home/cs342/Desktop/Step2# insmod module1.ko processId=5191
root@cs342vm:/home/cs342/Desktop/Step2# dmesg
[8816.293500]
Got the process id to look up as 5191.
[8816.293531] app[5191]
[8816.293533]
Code Segment start = 0x400000 end = 0x401000 size = 4096
[8816.293534]
Data Segment start = 0x600000 end = 0x601000 size = 4096
[8816.293536]
Stack Segment start = 0x7ffd5bce9000 end = 0x7ffd5bd0a000 size = 135168
[8816.293538]
Heap Segment start = 0x226c000, end = 0x22b9000, size 315392
[8816.293539] Arg Segment start = 0x7ffd5bce9000 end = 0x7ffd5bd0a000 size = 135168
[8816.293540] Env Segment start = 0x7ffd5bce9000 end = 0x7ffd5bd0a000 size = 135168
[8816.293541]
VM Number of Frames = 0x158
[8816.293542]
Task size = 0x140737488351232
[8816.293542]
VM size = 0x4632576
[8816.293543] c6e08067
[8816.293545] 1 1 1 0 0 1 1 0 0 814600 0 0
[8816.293547] c23cd067
[8816.293548] 1 1 1 0 0 1 1 0 0 795597 0 0
[8816.293549] c6fab067
[8816.293551] 1 1 1 0 0 1 1 0 0 815019 0 0
[8816.293551] 8d228067
[8816.293553] 1 1 1 0 0 1 1 0 0 578088 0 0
[8816.293554] 11d0a4067
[8816.293578] 1 1 1 0 0 1 1 0 0 1167524 0 0
[8816.293579] 11041e067
[8816.293580] 1 1 1 0 0 1 1 0 0 1115166 0 0
[8816.293581] 11d852067
[8816.293583] 1 1 1 0 0 1 1 0 0 1169490 0 0
[8816.293583] 8cfdd067
[8816.293585] 1 1 1 0 0 1 1 0 0 577501 0 0
[8816.293586] 8cc0c067
[8816.293587] 1 1 1 0 0 1 1 0 0 576524 0 0
root@cs342vm:/home/cs342/Desktop/Step2#

Stack Allocation and Deallocation

During the trial of stack allocation I mean recursive function, I tried Fibonacci but I could not increase the heap size. Then, I turned to factorial.

Note: I could not decrease the stack after recursion. As it stated in piazza, maybe the compiler predetermined at compile time for optimization purposes.

Results

Observation: In heap allocation, only total size and heap size was increased. However, in stack allocation, with stack size also number of frames, environment variable size and main argument sizes are also increases parallel to stack size. It can be occurred that since recursive function uses new parameters and arguments in each recursion, these parameters are also extended. Therefore, total size in stack increases more than heap increases.

As in heap size, stack initial size is also 135KB.

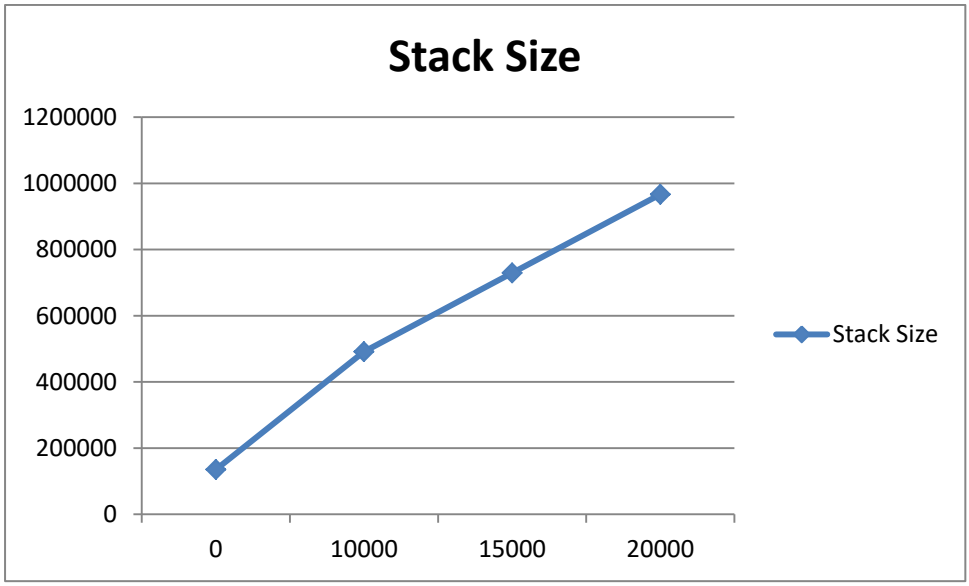
Exact number is 135168 byte. Important point is again that this size is multiple of our page size that is 135168 / 4096 = 33. It means we have 33 pages initially.

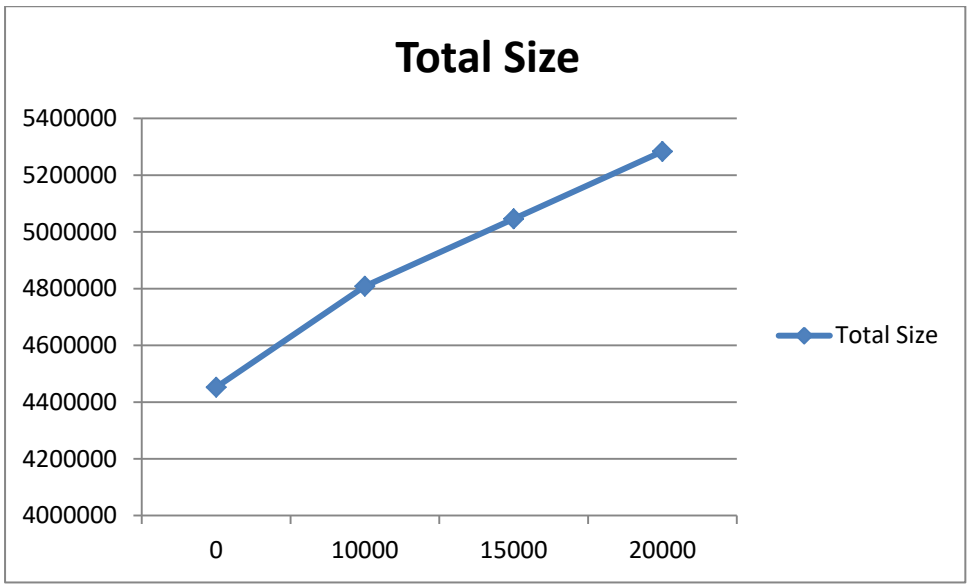
When we calculate 10000!, our stack size becomes 491520 that is 120 pages. In total we have 4452352 bytes with 1087 pages. One point is that increase in sizes is kind of larger than increase in heap. As I stated, stack is not only parameter that increases. Variable, argument and frames are also increasing.

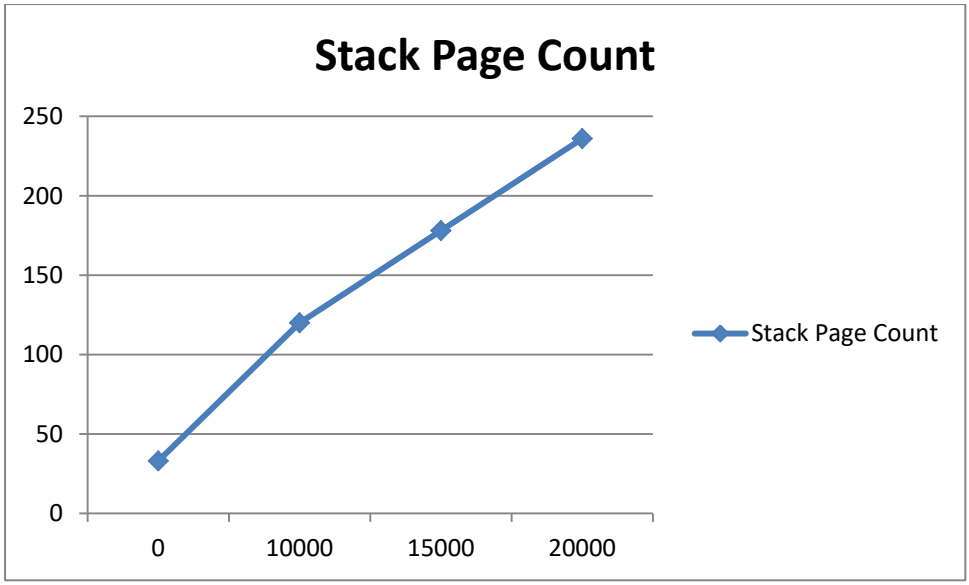
When we calculate 15000!, our stack size becomes 729088 that is 178 pages. In total we have 5046272 bytes with 1174 pages.

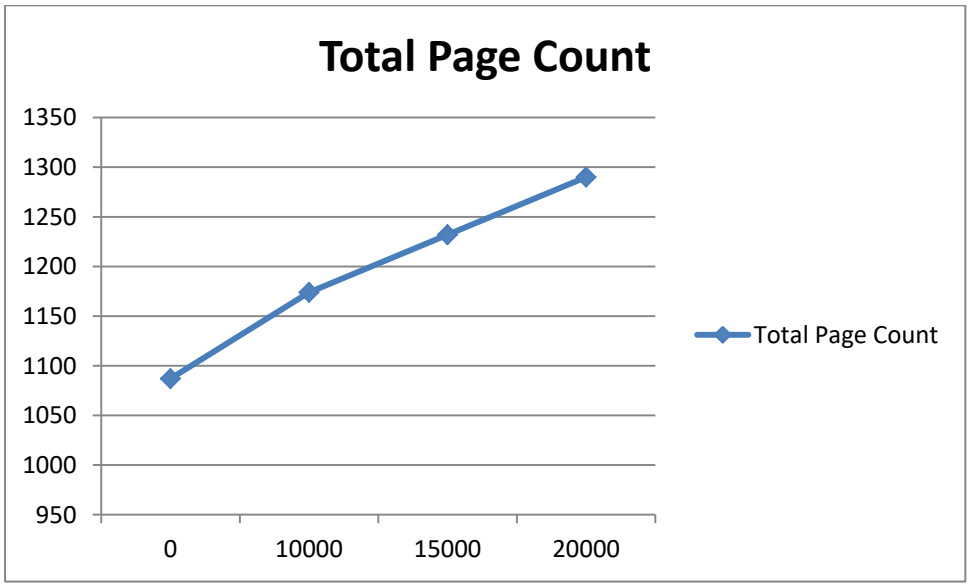
When we calculate 20000!, our stack size becomes 966656 that is 236 pages. In total we have 5283840 bytes with 1290 pages. We can see that in heap increase in input may not be resulted in increases in heap size. However, in here I get always increase in size. It may be occurred from that I kept interval between two inputs very large.

Graphics

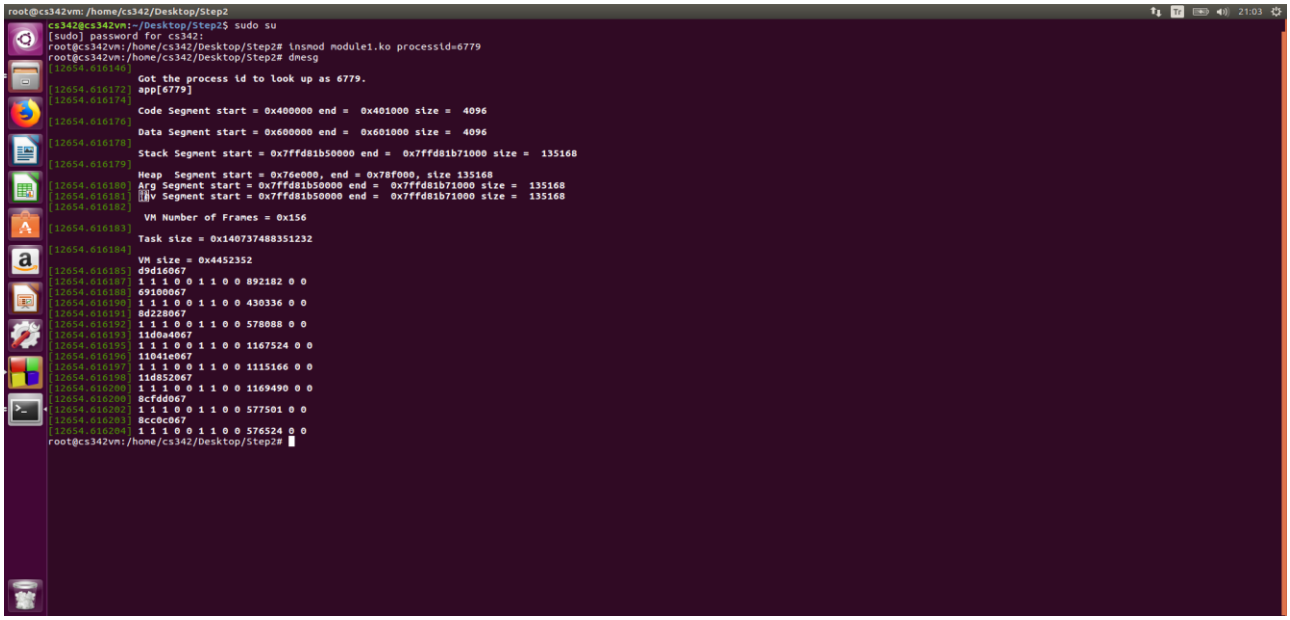

Stack Size
1200000
1000000
800000
600000
400000
200000
0
0
10000
15000
20000
Stack Size

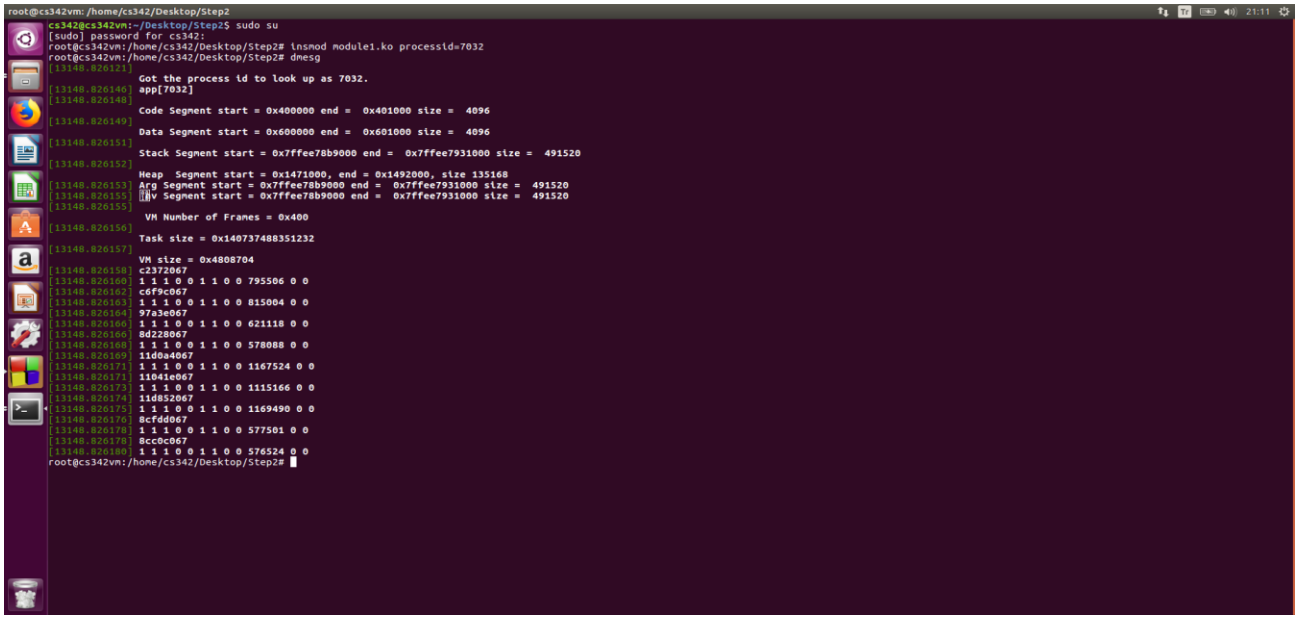

Total Size
5400000
5200000
5000000
4800000
4600000
4400000
4200000
4000000
0
10000
15000
20000
Total Size

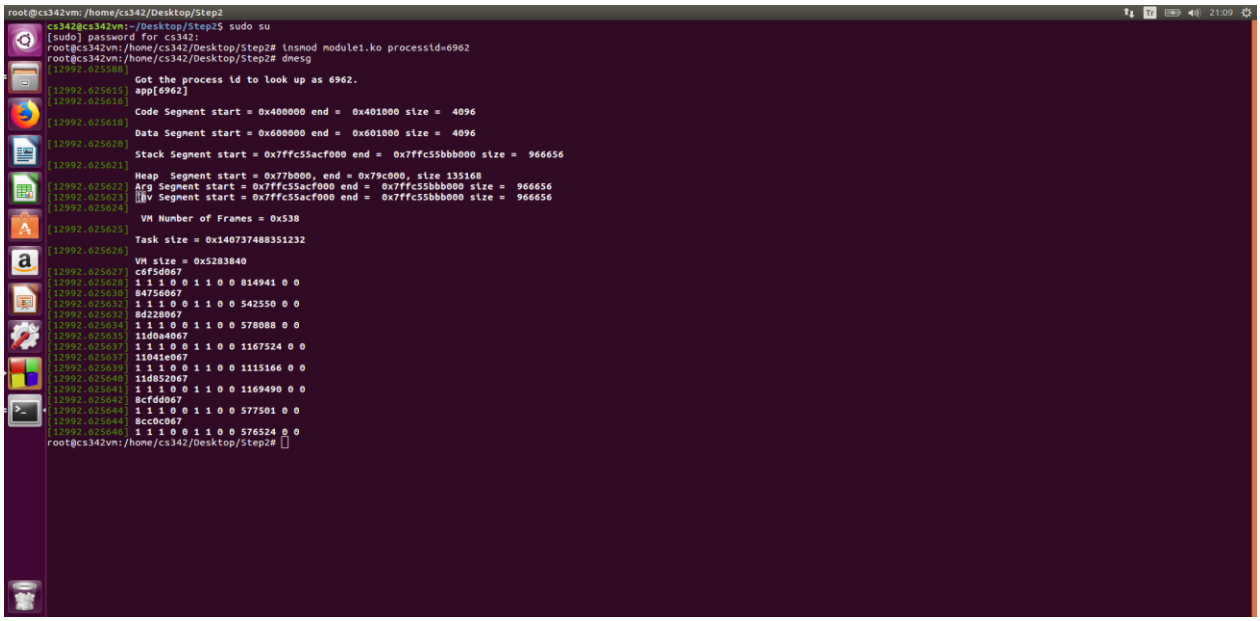

Stack Page Count
250
200
150
100
50
0
0
10000
15000
20000
Stack Page Count

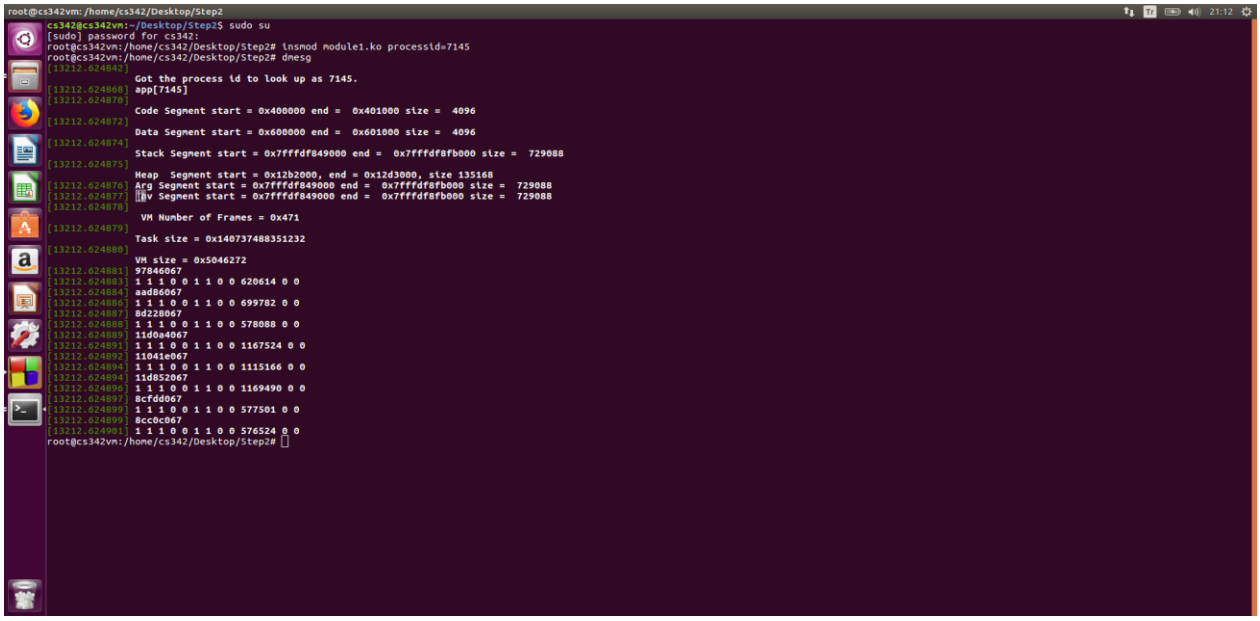

Total Page Count
1350
1300
1250
1200
1150
1100
1050
1000
950
0
10000
15000
20000
Total Page Count

Some Outputs






root@cs342vm: /home/cs342/Desktop/Step2
21:09
cs342@cs342vm:~/Desktop/Step2$ sudo su
[sudo] password for cs342:
root@cs342vm:/home/cs342/Desktop/Step2# insmod module1.ko processid=6962
root@cs342vm:/home/cs342/Desktop/Step2# dmesg
[12992.625588]
Got the process id to look up as 6962.
[12992.625615] app[6962]
[12992.625616]
Code Segment start = 0x400000 end = 0x401000 size = 4096
[12992.625618]
Data Segment start = 0x600000 end = 0x601000 size = 4096
[12992.625620]
Stack Segment start = 0x7ffc55acf000 end = 0x7ffc55bbb000 size = 966656
[12992.625621]
Heap Segment start = 0x77b000, end = 0x79c000, size 135168
[12992.625622] Arg Segment start = 0x7ffc55acf000 end = 0x7ffc55bbb000 size = 966656
[12992.625623] Env Segment start = 0x7ffc55acf000 end = 0x7ffc55bbb000 size = 966656
[12992.625624]
VM Number of Frames = 0x538
[12992.625625]
Task size = 0x140737488351232
[12992.625626]
VM size = 0x5283840
[12992.625627] c6f5d067
[12992.625628] 1 1 1 0 0 1 1 0 0 814941 0 0
[12992.625630] 84756067
[12992.625632] 1 1 1 0 0 1 1 0 0 542550 0 0
[12992.625632] 8d228067
[12992.625634] 1 1 1 0 0 1 1 0 0 578088 0 0
[12992.625635] 11d0a4067
[12992.625637] 1 1 1 0 0 1 1 0 0 1167524 0 0
[12992.625637] 11041e067
[12992.625639] 1 1 1 0 0 1 1 0 0 1115166 0 0
[12992.625640] 11d852067
[12992.625641] 1 1 1 0 0 1 1 0 0 1169490 0 0
[12992.625642] 8cfdd067
[12992.625644] 1 1 1 0 0 1 1 0 0 577501 0 0
[12992.625644] 8cc0c067
[12992.625646] 1 1 1 0 0 1 1 0 0 576524 0 0
root@cs342vm:/home/cs342/Desktop/Step2#


root@cs342vm: /home/cs342/Desktop/Step2
21:12
cs342@cs342vm:~/Desktop/Step2$ sudo su
[sudo] password for cs342:
root@cs342vm:/home/cs342/Desktop/Step2# insmod module1.ko processid=7145
root@cs342vm:/home/cs342/Desktop/Step2# dmesg
[13212.624842]
Got the process id to look up as 7145.
[13212.624868] app[7145]
[13212.624870]
Code Segment start = 0x400000 end = 0x401000 size = 4096
[13212.624872]
Data Segment start = 0x600000 end = 0x601000 size = 4096
[13212.624874]
Stack Segment start = 0x7fffdf849000 end = 0x7fffdf8fb000 size = 729088
[13212.624875]
Heap Segment start = 0x12b2000, end = 0x12d3000, size 135168
[13212.624876] Arg Segment start = 0x7fffdf849000 end = 0x7fffdf8fb000 size = 729088
[13212.624877] Env Segment start = 0x7fffdf849000 end = 0x7fffdf8fb000 size = 729088
[13212.624878]
VM Number of Frames = 0x471
[13212.624879]
Task size = 0x140737488351232
[13212.624880]
VM size = 0x5046272
[13212.624881] 97846067
[13212.624883] 1 1 1 0 0 1 1 0 0 620614 0 0
[13212.624884] aad86067
[13212.624886] 1 1 1 0 0 1 1 0 0 699782 0 0
[13212.624887] 8d228067
[13212.624888] 1 1 1 0 0 1 1 0 0 578088 0 0
[13212.624889] 11d0a4067
[13212.624891] 1 1 1 0 0 1 1 0 0 1167524 0 0
[13212.624892] 11041e067
[13212.624894] 1 1 1 0 0 1 1 0 0 1115166 0 0
[13212.624894] 11d852067
[13212.624896] 1 1 1 0 0 1 1 0 0 1169490 0 0
[13212.624897] 8cfdd067
[13212.624899] 1 1 1 0 0 1 1 0 0 577501 0 0
[13212.624899] 8cc0c067
[13212.624901] 1 1 1 0 0 1 1 0 0 576524 0 0
root@cs342vm:/home/cs342/Desktop/Step2#