CS695-A2

How to compile

- Use the Makefile and Kbuild file as given in this tutorial. Change obj-m variable in the Kbuild file to point to the lkm<i>.c file being compiled.
- Run `make && sudo insmod lkm<i>.ko && sudo dmesg | tail -100` to see the output after inserting the module
- Run `sudo rmmod lkm<i>` to remove the module
- The code runs perfectly on `Linux 5.15.0-57-generic` version
- For programs that take input, the values need to be specified through arguments while inserting the module

2a - Listing RUNNING and RUNNABLE Processes

- The output obtained after inserting the module :

```
[182697.083944] RUNNING and RUNNABLE processes
[182697.084366] PID:[1169]  CMD: Xorg
[182697.084516] PID:[1470]  CMD: gnome-shell
[182697.084740] PID:[1793]  CMD: gnome-terminal-
[182697.084987] PID:[65023] CMD: kworker/u8:2
[182697.085152] PID:[65035] CMD: kworker/u8:4
[182697.085633] PID:[65792] CMD: python3
[182697.085816] PID:[67351] CMD: insmod
```

- Output of `ps -aux | grep R`

```
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
ubuntu      1169  0.1  3.4 393652 137716 ?       Rl   Jan25   5:55 /usr/lib/xorg/Xorg :10
                                                                    -auth .Xauthority
                                                                    -config xrdp/xorg.conf
                                                                    -noreset -nolisten tcp
                                                                    -logfile .xorgxrdp.
                                                                    %s.log
ubuntu      1470  0.9  8.7 5045444 349148 ?      Rsl  Jan25  29:44 /usr/bin/gnome-shell
ubuntu      1793 30.8  1.3 895596 55656 ?        Rsl  Jan25 939:07 /usr/libexec/gnome-
                                                                    terminal-server
root       64947  1.9  0.0      0     0 ?        R    12:01   0:22 [kworker/u8:0-events_
                                                                     unbound]
root       65034  5.8  0.0      0     0 ?        R    12:12   0:26 [kworker/u8:3-events_
                                                                     unbound]
ubuntu     65792 76.0  0.6  39300 25008 pts/1    R+   12:16   2:50 python3
ubuntu     67102  0.0  0.0  10852  3216 pts/2    R+   12:20   0:00 ps -aux
ubuntu     67103  0.0  0.0   6416  1864 pts/2    S+   12:20   0:00 grep --color=auto R
```

- STAT being R in the ps command output stands for that the process is in Running or Runnable mode

- As can be observed the PIDs 1169, 1470, 1793, 65792 are listed in the both places. insmod, ps and grep are the processes of the command itself. The kworkers listed are different as different kernel threads perform processing in both the cases.

2b - Printing the Process PID that uses max heap

- I created a cpp program that allocated an integer array of size 10, 10000000 times on the heap and also accesses each of those arrays. This much of heap was not allocated to any other process.
- The output obtained after inserting the module

```
[187600.064988] PROCESS USING MAXIMUM HEAP
[187600.067394] PID:[75407] CMD: a.out  MAX HEAP MEM USAGE(in Bytes):480116736
```

Which is close to the expected 400000000 Bytes of expected allocation.

2c - Printing kernel stack ptr of PID=1 process

- The output obtained is :

```
[189882.786654] Kstack ptr of PID = 1 process
[189882.788016] PID:[1] Kstack ptr: ffff800008028000
```

- task_struct->stack stores the kernel stack pointer of the process. The address is also in the range of kernel space addresses.

2d - Virtual to Physical Address calculation

- Run the program like this : `sudo insmod lkm4.ko virtual_address=<virtual_address_in_hex> pid=<pid> && sudo dmesg | tail -5`
- The output for input virtual address `0xffff800000000008` (for any pid) is

```
[268350.959084] Virtual to Physical address
[268350.959937] Virtual address : 0xffff800000000008
[268350.960513] Physical Address: 0xecc00008
```

The above address belongs to kernel space and the mapping is independent of the pid of the process. - I wrote a C++ program and allocated an integer pointer (also assigned a value to it). I printed the virtual address of the location (pointer value). Then I checked the physical address mapping corresponding to that virtual address, and obtained the following output:

```
[262259.480009] Virtual to Physical address
[262259.480209] Virtual address : 0xaaaafa0aeeb0
[262259.480465] Physical Address: 0x123254eb0
```

- For the cases when the mapping doesn't exist or the PID doesn't exist the program prints appropriate message.

2e - Size of allocated virtual address space and size of physical address space

- Run the program like this : `sudo insmod lkm4.ko pid=<pid> && sudo dmesg | tail -5`
- The output obtained for some of the PIDs are:

```
[267828.588876] Virtual Address Space and Physical Address Space size
[267828.589802] PID:[147647]    total virtual memory (in Bytes): 82243584,
total physical memory(in Bytes) 1032192
[267837.043382] Virtual Address Space and Physical Address Space size
[267837.043714] PID:[147648]    total virtual memory (in Bytes): 82243584,
total physical memory(in Bytes) 81354752
```

- Program with PID 147547 is

```
#include <stdlib.h>

int main(int argc, char **argv) {

    int *big = (int*)malloc(sizeof(int)*20000000); // allocate 80 million bytes

    while(1) {

    }
}
```

- Program with PID 147548 is

```
#include <stdlib.h>
int main(int argc, char **argv) {

int *big = (int*)malloc(sizeof(int)*20000000); // allocate 80 million bytes
    for ( int* end = big + 20000000; big < end; big+=1024 ) {
        *big = 0xDEADBEEF ;
    }
    while(1) {

    }
}
```

- As can be seen in the output of the lkm5, same amount of virtual memory is allocated to the two processes but the physical memory allocated to the 2nd process is much more than the first one. This is because each page allocated in 2nd program was accessed and it ends up being alloted a physical page by the OS. This proves that linux uses lazy page allocation.
- References :

- https://www.kernel.org/doc/gorman/html/understand/understand006.html
- http://embeddedguruji.blogspot.com/2018/12/linux-kernel-driver-to-print-all.html
- http://www.science.smith.edu/~nhowe/262/oldlabs/sched.html
- https://stackoverflow.com/questions/8980193/walking-page-tables-of-a-process-in-linux
- Pointers given on moodle discussion forum and some stackoverflow threads pertaining to coding specific to kernel modules.