

CS695-A2

How to compile

- Use the Makefile and Kbuild file as given in [this tutorial]{https://linux-kernel-labs.github.io/refs/heads/master/labs/kernel_modules.html}. 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 set at the top of the program before compiling

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

- The output for input virtual address 0xffff800000000008 (for any pid) is

```
[194138.248162] Virtual to Physical address
[194138.248267] Virtual address : 0xffff800000000008
[194138.248377] 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:

```
[194379.877137] Virtual to Physical address
[194379.877838] Virtual address : 0xaaaaecc36eb0
[194379.878253] Physical Address: 0xaa0
```

- 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

- The output obtained for some of the PIDs are:

```
[195844.701846] PID:[106538]    total virtual memory (in Bytes): 24363008, physical_vm (in B
[195844.702080] PID:[106539]    total virtual memory (in Bytes): 14401536, physical_vm (in B
[195844.702303] PID:[106540]    total virtual memory (in Bytes): 14401536, physical_vm (in B
[195844.702518] PID:[106541]    total virtual memory (in Bytes): 7823360, physical_vm (in B
```

As we can see that virtual memory allocated \geq physical memory allocated.
This means that there exist some virtual pages with no physical address mapping
by the os. This shows that linux uses lazy allocation.