CS 477
Advanced Operating System

Tutorial 03: kernel module

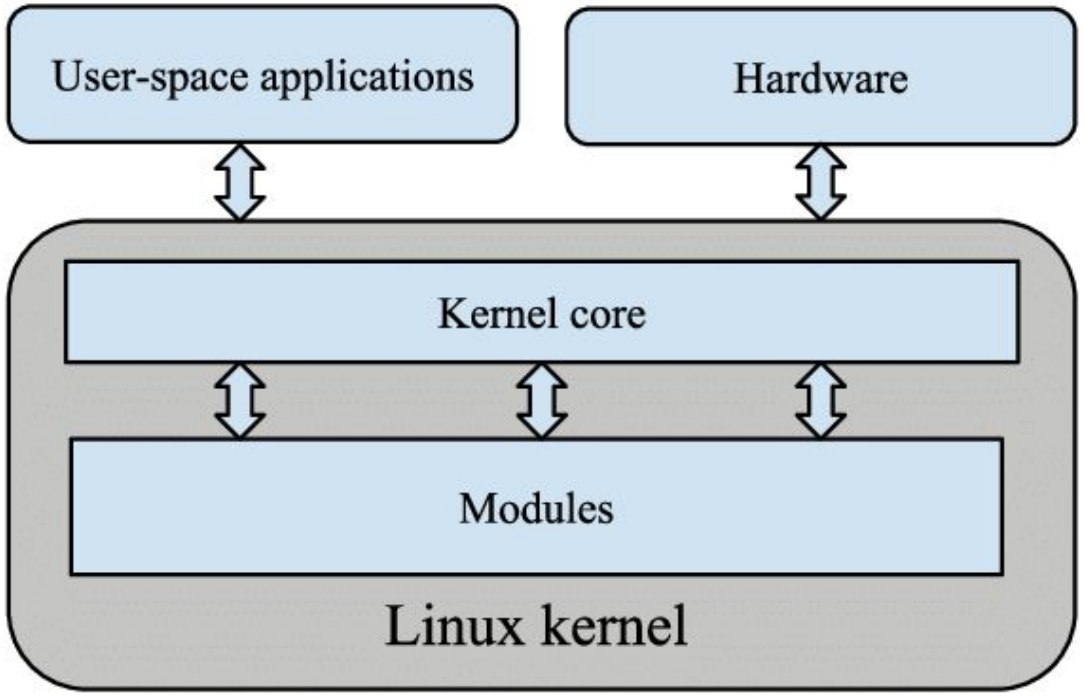
Today's tutorial

- Kernel Modules
  - What is kernel module?
    - Where is it used?
  - How to write/compile a kernel module?
  - What if you do something bad?
- eBPF (briefly)
  - What is eBPF?
    - Where is it used?
  - How to write eBPF code?
  - Some widely used eBPF framework

What is kernel module?

A Linux kernel module is some code that can be dynamically (un)loaded into the kernel without rebooting to extend the functionality at runtime.

- Device drivers (e.g. NIC)
- Filesystem (e.g. btrfs)
- Custom system calls
- Research / experimental features



How to write a kernel module?

Start with a very simple example

- A kernel module which
- Prints hello world while being loaded
- Prints goodbye while being unloaded

Demo code:
https://github.com/PanJason/module_and_ebpf/tree/main/kernel_module_examples/hello_world

```
$ make
$ sudo insmod hello_world.ko
$ sudo rmmod hello_world
$ modinfo hello_world.ko
$ sudo dmesg | tail -n 2
```

How to add some control over your kernel module?

Again a very simple example:
We create a slab cache when initialization:

- slab/slub: kernel cache that efficiently handles fixed size allocation
- Handy tools: slabtop, /proc/slabinfo
- Expose a debugfs file which allows user to change the number of allocated objs
- `/sys/kernel/debug/slab_allocation/object_count`

Note:

- Concurrency!
- Slab can be merged if some

Demo code:

- https://github.com/PanJason/module_and_ebpf/tree/main/kernel_module_examples/slab_allocation

How to add a new syscall?

We want to replace an existing syscall/ add a new syscall:

- Need to check `Documentation/process/adding-syscall.rst`
- Previously replace an entry in sys_call_table
- Now we can no long do it due to safety reasons:
  - https://stackoverflow.com/questions/78599971/hooking-syscall-by-modifying-sys-call-table-does-not-work/78607015#78607015

Kprobe:

- a dynamic instrumentation mechanism in Linux that allows you to attach your handler to almost any kernel functions.

Demo code:

- https://github.com/PanJason/module_and_ebpf/tree/main/kernel_module_examples/new_syscall

How to do multi-threading inside kernel?

We want to create some kthreads to help us do some work concurrently (possibly on different CPUs)

- Use kthread_* APIs to create thread
- Use completion to synchronization across threads (similar to condvar)
- Use mutex (sleepable) and spinlock (non sleepable)

Very simple example, two threads, one thread waits for another
Demo code:
https://github.com/PanJason/module_and_ebpf/tree/main/kernel_module_examples/multi_threading

How to put requesters to sleep?

We create a procfs file under /proc:

- Only allow one process to open the file at a time
- Put all the other processes on the waitq
- Wakes one up after close
- If the process open the file with O_NONBLOCK, return

Note:

- Use wake_up_interruptible
- If using wake_up, ctrl-c will have no effect

Demo code:
https://github.com/PanJason/module_and_ebpf/tree/main/kernel_module_examples/wake_up

What if we do something bad?

We create a kernel module which

- Deref a nullptr when loaded
- Usually if there is context to kill, the kernel won't "die"
- But if panic is called, then the kernel dies

Demo code:
https://github.com/PanJason/module_and_ebpf/tree/main/kernel_module_examples/bad_kernel_module

So you can see, there is very little protection from what you can do inside kernel
Kernel module is flexible but error-prone

A useful book to learn about

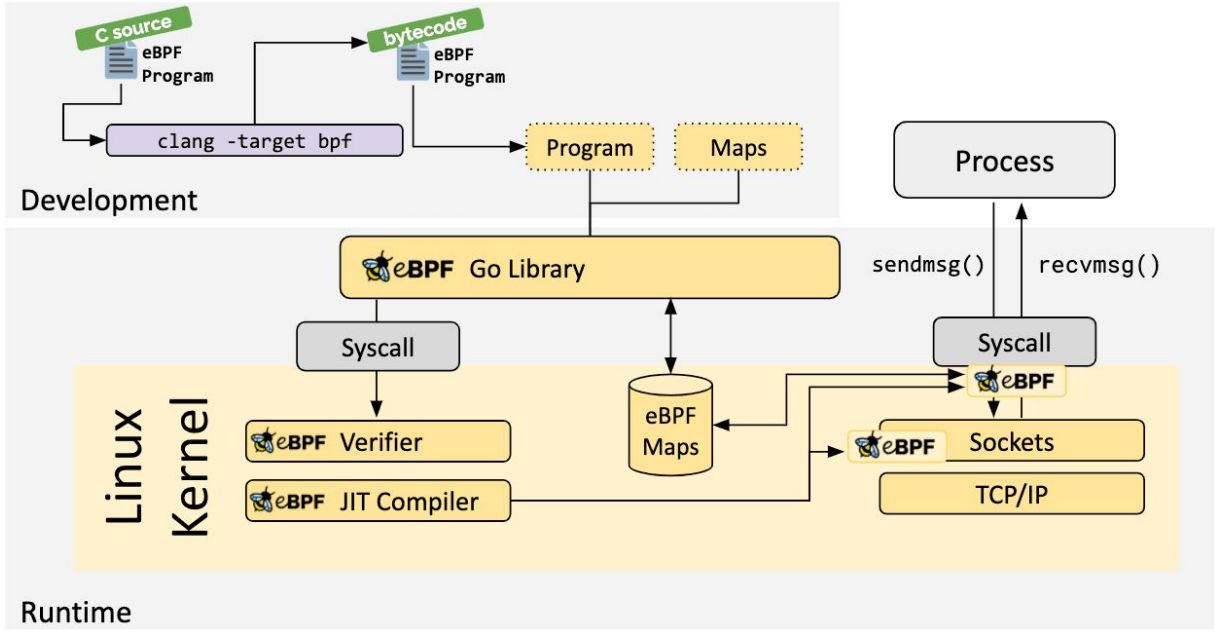
Some of the examples are taken from here:
https://sysprog21.github.io/lkmpg/

You can read the book for more interesting examples how to use kernel module to do thing you want

Is there a safer way to inject code into kernel?

eBPF (extended Berkeley Packet Filter) is a powerful Linux kernel technology that lets you run custom, safe, sandboxed programs inside the kernel — without changing kernel code or rebooting

- Verifier to guarantee safety
- Attached to hooks
- Executed in kernel with low overhead using JIT compiler
  - Which translates the eBPF bytecode to native machine code



A very simple eBPF example:

Our program use BPF_map to count the number of times execve syscall is called
Bpf provides a bunch of:

- Data structures
  - https://docs.kernel.org/bpf/maps.html
- Helper functions that invokes the kernel functionalities safely:
  - https://man7.org/linux/man-pages/man7/bpf-helpers.7.html

Demo code:
https://github.com/PanJason/module_and_ebpf/tree/main/bpf_examples/simple_map

Some framework which uses eBPF

Observability & Tracing tools:

- bcc https://github.com/iovisor/bcc/tree/master
  - Python c++ toolkit to do a lot of things, like observe I/O network
- bpftrace https://github.com/bpftrace/bpftrace
  - High level tracing language like awk
  - ```
    bpftrace -e 'tracepoint:syscalls:sys_enter_openat {
    printf("%s opened %s\n", comm,
    str(args->filename)); }'
    ```

Networking:

- katran: https://github.com/facebookincubator/katran
  - Layer 4 load balancer
- XDP tools: https://github.com/xdp-project/xdp-tools
  - A bunch of tools works at XDP layer
    ```