Introduction to Linux Kernel Modules

Luca Abeni

luca.abeni@santannapisa.it

Linux Kernel Modules

- Kernel module: code that can be dynamically loaded/unloaded into the kernel at runtime
- Change the kernel code without needing to reboot the system
- More technically: the modules' object code is dynamically linked to the running kernel code
  - Form of dynamic linking!
- This mechanism can be used for some simple experiments on Linux kernel programming!

Using Kernel Modules

- Kernel Module: *kernel object* $\rightarrow$ `.ko` file
- Inserted with `modprobe <module name>`
- Can be removed with `rmmod <module name>`
- When inserted, a kernel module can:
  - Register some services
  - Start some tasks (kernel threads)
- A kernel module can use some *exported kernel functions*

Kernel Programming - 1

- No single entry point (no "`main()` function)
- No memory protection
  - Kernel Memory Address Space: all the memory can be accessed
  - Kernel-space tasks can easily corrupt important data structures!
- Not linked to standard libraries
  - Cannot include `<stdio.h>` and friends...
  - No standard C library!

Kernel Programming - 2

- The kernel (or nanokernel, or ...) provides some functions we can use
  - Example, no `printf()`, but `printk()`...
- Errors do not result in segmentation faults...
- ...But can cause system crashes!
- Other weird details
  - No floating point (do not use `float` or `double`)
  - Small stack ($4KB$ or $8KB$)
  - Atomic contexts, ...

Kernel Programming Language

- OS kernels are generally coded in C or C++
  - The Linux kernel uses C
  - Subset of C99 + some extensions (`likely()` / `unlikely()` annotations, etc...)
- As said, no access to standard libraries
  - Different set of header files and utility functions
- Some Assembly is used (for entry points, etc...)
- Example: Linked Lists (`include/linux/list.h`)

Writing Linux Kernel Modules

- Written in C99 + extensions (see previous slide)
- Must include some headers:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
```

- Must define two entry points: *init* and *cleanup*
  - Init entry point: called when the module is inserted
  - Cleanup entry point: called when the module is removed

The Init Entry Point

```
static int __init my_init(void)
{
  ...
  return 0;
}

module_init(my_init);
```

- `static`: not used outside this compilation unit
- `__init`: annotation for the kernel (not used after `insmod`)
- `return 0;`: module initialised without errors
- `module_init(my_init);`: mark `my_init` as the init entry point

The Exit Entry Point

```
static void __exit my_cleanup(void)
{
   ...
}

module_exit(my_cleanup);
```

- `__exit`: annotation for the kernel (used only in `rmmod`)
- `module_exit(my_cleanup);`: mark `my_cleanup` as the cleanup entry point
- Responsible for undoing things done by init
- If not defined, the module cannot be unloaded

Compiling Linux Kernel Modules

- Compiling user-space code is simple
  - `gcc` without additional parameters works
  - Makefiles and similar for more complex programs
- But compiling kernel code is more difficult!
  - “Freestanding” programming environment $\rightarrow$ special compiler options are needed
  - The compiler defaults might change from version to version
  - ...
- Fortunately, Linux developers already did the dirty work for us!
  - KBuild system

KBuild

- Set of Makefiles, programs and scripts used to build the Linux kernel
- Already knows which compiler options to use
- Simpler to use than “regular” Makefiles
  - We just need to tell kbuild the name of the module we want to build
- Supports the compilation of kernel modules
  - Even external (out-of-tree) modules!

Using KBuild

- Based on Makefiles
- Important line: "`obj-m = modulename.o`"
  - This assumes modules composed by one single compilation unit (`.c` file)
  - In case of multiple compilation units, use "`modulename-objs = ...`" (list of `.o` files)
- To use it, we must tell `make` where KBuild is
  - `make -C PathToLinuxSources M=$(pwd)`
  - Where "`PathToLinuxSources`" is the pathname of a *compiled* Linux kernel
- The "`-C ...`" complication can be embedded in a Makefile rule (see example)

Applications as Kernel Modules

- The init entry point must return quickly
  - `modprobe` does not terminate until init returns
- It can create some threads, or register some device, and return
  - After loading the module, the application is started!
- The cleanup entry point stops the threads / unregister the device
- See example