Linux Kernel Device Driver Development – Part 1

Course Content

- ❖ Installing the Linux on Virtual machine.
- ❖ Introduction to Linux Kernel
- ❖ Code compilation
- ❖ Makefile creation
- ❖ **Hello world Module**
- ❖ Module utilities
- ❖ Character Driver
- ❖ Creating /sys and /proc entries using kernel module
- ❖ Submitting your First patch to Linux kernel source.

Hello world Module

Kernel Module

- Runtime loadable modules.
- Built against the kernel source version.
- Modules built have extensions ***.ko**
- Can be built static inside the kernel image.
- Easy to handle source code.

Hello world Module

module_init

- Entry function is registered by module_init.
- Entry function is called while loading the module.

module_exit

- Exit function is registered by module_exit.
- Exit function is called while loading the module.

Function macros can be found under include/linux/module.h

Hello world Module

Module information

- MODULE_AUTHOR – Developer of the module
- MODULE_LICENSE – Proprietary or open source
- MODULE_DESCRIPTION – Module functionality

Hello world Module

Module Compilation

- Makefile for module

  Obj-m:= helloworld.o

- Build the kernel module using

  make –C <path of kernel source> M=${pwd}

- By executing this, the Makefile dependencies of the kernel source are resolved and .ko files are created in the driver source directories.

Hello world Module

Complete Makefile

```
obj-m:= helloworld.o

default:
	make –C /usr/src/linux-headers-`uname –r`/   M=`pwd` modules
clean:
	make –C /usr/src/linux-headers-`uname -r`/    M=`pwd`  clean
```

Hello world Module

Insmod

- use to load the module.
- Need the sudo permission
- Execute with the .ko extension

Lsmod

- Used to get the information of active modules in kernel
- No need of sudo permission

Rmmod

- used to remove the module from kernel
- Sudo permission is required
- Execute without .ko extension

printk

printk log level.

- Every **printk** message with log level
- Console driver reads the ring buffer continuously.
- Console log level uses default console log level as reference.
- Default log level can be seen in **/proc/sys/kernel/**printk
- **printk** messages above console log level are printed.
- **printk** works in critical area
- **printk** writes directly to memcpy.

More information about log levels and printk can be found under include/linux/kern_levels.h and include/linux/printk.h respectively.

Hello world Module

Runtime status of the module

- Modules which are active have the directory created under /sys/modules.
- Inside the respective kernel module directory, the status of module can be known from userspace.

Module stages (cat /proc/module)

- Coming : While module is being loaded
- Live : when module is live in the memory
- Going: While module is unloaded.

We often see the status as Live since the module has been successfully loaded in the memory. However, if module is crashed in the init phase, then status can be seen as coming.

Hello world Module

Symbol table of the module

- Each kernel module will have symbol table entries.
- Can be accessed in **/proc/kallsyms.**
- Gives the virtual address of the functions in the kernel memory.
- Useful for debugging.

Hello world Module

Use of **__init**

- Without the use of init, the entry function will be present in kernel memory.
- Entry function will occupy few bytes of memory.
- In resource constrained environment, this will cause a bottleneck.
- Once the initialization function is successfully retuned, the function can be removed from the memory using **__init** keyword.
- This feature is widely used in modern day drivers, where, after probing of the drivers, the init functions are removed from memory.