Linux Kernel Workbook

Release 1.0

Rishi Agrawal

Aug 06, 2017

Contents

CHAPTER 1

About the Book

Introduction

Linux Kernel Programming is one of the most exciting thing for the beginners in System Programming. With time a lot of excellent books were published on it but the books were written to cover everything about the Linux Kernel and thus had a widespread scope. These books enabled to a lot of people to take kernel programming as their career chocices or as a hobby.

I have been seeing a lot of queries on the `kernelnewbies` mailing list where people are requesting for the documents to start with. Most of them want to go through a guided course or something.

People (even I) generally suggest them the following books:

1. The Design Of Unix Operating System
2. Linux Kernel Development
3. Understanding The Unix Kernel

For the learners who can't wait to get into code - I have written this small book.

This book uses Ubuntu Server LTS releases.

This book will deal with the folowing topics.

- Compilation of Kernel
- Code Walkthrough using `cscope` and `ctags`
- Kernel Modules
- Kernel Module Programming
- Proc Interface
- Data Strucutures available in the Kernel
- Character Device Drivers

The section has many `Hands-On` exercises which will enable you to learn how to write code for kernel modules, and make changes to the linux kernel. This section is very essential for getting a good feel of kernel programming.

Note that the book is entirely not my work, I have taken references from a lot of documents and articles from the open source community.

Happy Hacking !!

CHAPTER 2

Setting Up

Introduction

We need to setup our machine properly so that we do not face issues at a later stage.

This chapter is the pre-requisite for the whole document to be completed successfully.

Steps

- Download and install `Ubuntu Server 16.04.2 LTS` on a Virtual Machine.
- Here are the output of the commands `uname -a` and `cat /etc/lsb-release` file.

```
rishi@rishi-VirtualBox:~$ uname -a
Linux rishi-VirtualBox 4.4.0-24-generic #43-Ubuntu SMP Wed Jun 8 19:27:37 UTC 2016␣
↪x86_64 x86_64 x86_64 GNU/Linux
```

```
rishi@rishi-VirtualBox:~$ cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=16.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu 16.04 LTS"
```

Why a Virtual Machine

- The machine where you will play should be a virtual machine so that your invalid setting to the kernel or any other configuration should not effect your base host machine. During this whole book you will write code which CAN end up shutting down the system or corrupting the system. Doing all this stuff on a virtual machine will help us in running the code on a different machine, thus saving our machines from the catastrophic failures.

- The virtual machine should have at least 2 cores and 512 MB of RAM. This will make the kernel compilation fast.
- Take a snapshot/clone of the machine once you are done with the installation. This will enable you to revert your machine to the original state in case you mess up with it too badly.

Install Packages

Install the following packages.

```
vim
gcc
openssh-server
gdb
tree
cscope
ctags
```

The script below will help you in this.

```
#! /usr/bin/bash

PACKAGES="vim gcc openssh-server gdb tree ctags cscope make"

for package in ${PACKAGES}; do
    sudo apt-get install ${package}
done
```

CHAPTER 3

Kernel Compilation

Introduction

At first we will learn how to compile a kernel. This is just a series of steps we need to do in order to compile and boot to a new kernel.

Why this chapter

This chapter will give you your first hands on with Linux Kernel - thus enabling you understand some fundamental concepts around it.

You compile the kernel - make changes to the kernel right in the first chapter. So your exciting journey with Linux kernel begins.

What will you learn

You will learn

- How to download a Linux kernel.
- How to configure the Linux kernel in different ways.
- How to see the effects of the configuration changes you did.
- Compile a Linux kernel.
- Package a compiled kernel.
- Install a Linux kernel.
- Boot from a new Linux kernel.
- Making only a module in the kernel.

Prerequisites

- We expect that you have already installed a Linux system and have some basic understanding of Linux terminal.

Linux Kernel

- This is simple code written in `C`.
- This is a **LARGE** `C` program.
- Code looks difficult to understand because of the LARGENESS of the system and lack of understanding of the operating system concepts.
- You have to be little more careful while coding as small mistakes can cause the whole system to crash. Debugging these can be difficult at times.

Kernel Compilation

- We may need to compile our own kernel to add/remove some features present in the system.
- The kernel distributed with **general settings** setting which should run on all the possible installations.
- Thus they need to support a wide range of hardware.
- Some of the features may be built in the kernel while some of them may be built as modules.
- It's alright if they are built as module as they don't increase the size of the kernel.
- Built-in features will increase the size of kernel, thus effecting the system's performance. (not too heavily)
- Making our own kernel will ensure the kernel is having appropriate set of features.
- Double check before you **remove** any feature, your freshly compiled kernel may not boot :-).
- Read `Linux Kernel In A Nutshell` for further understanding.

Compiling a Kernel - Steps

- www.kernel.org - download the new kernel from this website.
- **`tar -xzf/xjf downloaded kernel`**
- **`make oldconfig`** - makes the old configuration present in the system, if new features are present asks about them.
- **`make defconfig`** - makes the default configuration for your architecture, the configuration file is present in `arch/ARCHITECTURE/configs/`.
- **`make gconfig`** - gives a GTK based configuration menu. (We will not use this in this book.)
- **`make menuconfig`** - gives a ncurses based configuration menu.
- **`make modules`** - makes the modules.
- **`make modules_install`** - installs the modules to the required location usually `/lib/modules/KERNEL-VERSION/kernel`.
- **`make install`** - installs the kernel in the required location usually `/boot/`, makes the initial ramfs sets up the boot loader, you are now ready to reboot your system.

Hands-On Compling a Kernel

- Let us see the current kernel version on the system

```
rishi@rishi-VirtualBox:~$ uname -a
Linux rishi-VirtualBox 4.4.0-24-generic #43-Ubuntu SMP Wed Jun 8 19:27:37 UTC 2016␣
↪x86_64 x86_64 x86_64 GNU/Linux
```

- First download the kernel. We will use the 4.7 kernel for it.

```
wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.7.tar.xz
--2016-07-28 08:38:14--  https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.7.tar.xz
Resolving cdn.kernel.org (cdn.kernel.org)... 151.101.8.69
Connecting to cdn.kernel.org (cdn.kernel.org)|151.101.8.69|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 90412100 (86M) [application/x-xz]
Saving to: 'linux-4.7.tar.xz'

100%[=======================================================================================>
↪]
9,04,12,100  225KB/s   in 4m 23s

2016-07-28 08:42:39 (336 KB/s) - 'linux-4.7.tar.xz' saved [90412100/90412100]
```

- Copy the kernel to your virtual machine. Here replace the IP with your machine's IP. You can directly download the kernel in your virtual machine as well.

```
[08:42 - ] -------- /home/rishi/code
[rishi-office 7] > scp linux-4.7.tar.xz rishi@192.168.0.106:
linux-4.7.tar.xz
100%   86MB  86.2MB/s   00:01
[08:47 - ] -------- /home/rishi/code
[rishi-office 8] >
```

- Untar the kernel

```
$ tar -xf linux-4.7.tar.xz
```

- This will give you a folder.

```
$ ls
linux-4.7  linux-4.7.tar.xz
```

- This folder has thousands of files. Lets do a `find` and count the number of files.

```
rishi@rishi-VirtualBox:~/lkw$ find linux-4.7/| wc -l
58057
```

- This folder has a lot of folders. See the following tree command.

```
$ tree linux-4.7  -L 1 -f
linux-4.7
|__ linux-4.7/arch
|__ linux-4.7/block
|__ linux-4.7/certs
|__ linux-4.7/COPYING
|__ linux-4.7/CREDITS
|__ linux-4.7/crypto
```

```
|__ linux-4.7/Documentation
|__ linux-4.7/drivers
|__ linux-4.7/firmware
|__ linux-4.7/fs
|__ linux-4.7/include
|__ linux-4.7/init
|__ linux-4.7/ipc
|__ linux-4.7/Kbuild
|__ linux-4.7/Kconfig
|__ linux-4.7/kernel
|__ linux-4.7/lib
|__ linux-4.7/MAINTAINERS
|__ linux-4.7/Makefile
|__ linux-4.7/mm
|__ linux-4.7/net
|__ linux-4.7/README
|__ linux-4.7/REPORTING-BUGS
|__ linux-4.7/samples
|__ linux-4.7/scripts
|__ linux-4.7/security
|__ linux-4.7/sound
|__ linux-4.7/tools
|__ linux-4.7/usr
|__ linux-4.7/virt
```

- This folder has a `Makefile` which is the `Makefile` to compile the kernel. The file is very long, but you need not bother about it. We are interested only in few targets.

```
$ wc -l  Makefile
1669 Makefile
```

- We will now do the steps mentioned above. Right now the folder has no `.config` file. When we configure the kernel for compilation the file get created. The `.config` file keep the configuration for the kernel to be built. Whatever configuration changes you do while configuring the kernel, it gets saved in this file.

```
$ ls .config
ls: cannot access '.config': No such file or directory
```

- Here is a small snippet of the configuration file which we will generate in sometime.

```
$ tail -f .config
CONFIG_UCS2_STRING=y
CONFIG_FONT_SUPPORT=y
# CONFIG_FONTS is not set
CONFIG_FONT_8x8=y
CONFIG_FONT_8x16=y
# CONFIG_SG_SPLIT is not set
CONFIG_SG_POOL=y
CONFIG_ARCH_HAS_SG_CHAIN=y
CONFIG_ARCH_HAS_PMEM_API=y
CONFIG_ARCH_HAS_MMIO_FLUSH=y
```

- `make defconfig` - makes the default configuration for your architecture, the configuration file is present in `arch/ARCHITECTURE/configs/`.

```
$ make defconfig
*** Default configuration is based on 'x86_64_defconfig'
#
```

```
# configuration written to .config
#

rishi@rishi-VirtualBox:~/lkw/linux-4.7$ ls .config
.config
```

- The `.config` file is very long. See this.

```
$ wc -l .config
4044 .config
```

- `make oldconfig` will read the config file `(/boot/config-4.4.0-24-generic)` in your machine and try to use that configuration file. There might be some features in the new kernel which is not available in the older/default kernel you have. For this the command takes input from you. Based on the features you want to enable and disable - you can give the inputs.

```
$ make oldconfig
scripts/kconfig/conf  --oldconfig Kconfig
#
# using defaults found in /boot/config-4.4.0-24-generic
#
/boot/config-4.4.0-24-generic:1631:warning: symbol value 'm' invalid
    for RXKAD

/boot/config-4.4.0-24-generic:3589:warning: symbol value 'm' invalid
    for SERIAL_8250_FINTEK
*
* Restart config...
*
*
* General setup
*
*
* Timers subsystem
*
Timer tick handling
  1. Periodic timer ticks (constant rate, no dynticks) (HZ_PERIODIC)
> 2. Idle dynticks system (tickless idle) (NO_HZ_IDLE)
  3. Full dynticks system (tickless) (NO_HZ_FULL)
choice[1-3]: 2
Old Idle dynticks config (NO_HZ) [Y/n/?] y
High Resolution Timer Support (HIGH_RES_TIMERS) [Y/n/?] y
*
* CPU/Task time and stats accounting
*
Cputime accounting
> 1. Simple tick based cputime accounting (TICK_CPU_ACCOUNTING)
  2. Full dynticks CPU time accounting (VIRT_CPU_ACCOUNTING_GEN)
  3. Fine granularity task level IRQ time accounting (IRQ_TIME_ACCOUNTING)
choice[1-3]: 1
BSD Process Accounting (BSD_PROCESS_ACCT) [Y/n/?] y
  BSD Process Accounting version 3 file format (BSD_PROCESS_ACCT_V3) [Y/n/?] y
Export task/process statistics through netlink (TASKSTATS) [Y/?] y
  Enable per-task delay accounting (TASK_DELAY_ACCT) [Y/?] y
  Enable extended accounting over taskstats (TASK_XACCT) [Y/n/?] y
    Enable per-task storage I/O accounting (TASK_IO_ACCOUNTING) [Y/n/?] y
*
* RCU Subsystem
```

```
*
Make expert-level adjustments to RCU configuration (RCU_EXPERT) [N/y/?] n
Kernel .config support (IKCONFIG) [N/m/y/?] n
Kernel log buffer size (16 => 64KB, 17 => 128KB) (LOG_BUF_SHIFT) [18] 18
CPU kernel log buffer size contribution (13 => 8 KB, 17 => 128KB) (LOG_CPU_MAX_BUF_
↪SHIFT) [12] 12
Temporary per-CPU NMI log buffer size (12 => 4KB, 13 => 8KB) (NMI_LOG_BUF_SHIFT) [13]␣
↪(NEW)

-----------SNIPPED------------
#
# configuration written to .config
#
```

- You can see the difference in the default config file and the currently generated `.config` file by the `diff` command.

```
$ diff /boot/config-4.4.0-24-generic .config  | more
3c3
< # Linux/x86_64 4.4.0-24-generic Kernel Configuration
---
> # Linux/x86 4.7.0 Kernel Configuration
9d8
< CONFIG_PERF_EVENTS_INTEL_UNCORE=y
14d12
< CONFIG_HAVE_LATENCYTOP_SUPPORT=y
15a14,17
> CONFIG_ARCH_MMAP_RND_BITS_MIN=28
> CONFIG_ARCH_MMAP_RND_BITS_MAX=32
> CONFIG_ARCH_MMAP_RND_COMPAT_BITS_MIN=8
> CONFIG_ARCH_MMAP_RND_COMPAT_BITS_MAX=16
42a45
> CONFIG_DEBUG_RODATA=y
69d71
< CONFIG_VERSION_SIGNATURE="Ubuntu 4.4.0-24.43-generic 4.4.10"
97d98
< # CONFIG_IRQ_FORCED_THREADING_DEFAULT is not set
145a147
> CONFIG_NMI_LOG_BUF_SHIFT=13
153,159d154
< # CONFIG_CGROUP_DEBUG is not set
< CONFIG_CGROUP_FREEZER=y
< CONFIG_CGROUP_PIDS=y
< CONFIG_CGROUP_DEVICE=y
< CONFIG_CPUSETS=y
< CONFIG_PROC_PID_CPUSET=y
< CONFIG_CGROUP_CPUACCT=y
164,166c159,161
< CONFIG_MEMCG_KMEM=y
< CONFIG_CGROUP_HUGETLB=y
< CONFIG_CGROUP_PERF=y
---
```

- `make gconfig` - gives a GTK based configuration menu. In my system there is no gtk based libraries available hence the window did not start.

```
rishi@rishi-VirtualBox:~/lkw/linux-4.7$ make gconfig
*
```

```
* Unable to find the GTK+ installation. Please make sure that
* the GTK+ 2.0 development package is correctly installed...
* You need gtk+-2.0, glib-2.0 and libglade-2.0.
*
make[1]: *** No rule to make target 'scripts/kconfig/.tmp_gtkcheck', needed by
'scripts/kconfig/gconf.o'.  Stop.
Makefile:544: recipe for target 'gconfig' failed
make: *** [gconfig] Error 2
```

- `make menuconfig` - gives a ncurses based configuration menu. We will use this for configuring our new kernel. **This will initially fail as there is not ncurses installed.**

```
$ make menuconfig
  HOSTCC  scripts/kconfig/mconf.o
  In file included from scripts/kconfig/mconf.c:23:0:
  scripts/kconfig/lxdialog/dialog.h:38:20: fatal error: curses.h: No such file or␣
↪directory
  compilation terminated.
  scripts/Makefile.host:108: recipe for target 'scripts/kconfig/mconf.o' failed
  make[1]: *** [scripts/kconfig/mconf.o] Error 1
  Makefile:544: recipe for target 'menuconfig' failed
  make: *** [menuconfig] Error 2
```

- We will now install ncurses.

```
$ sudo apt-get install ncurses-dev
[sudo] password for rishi:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'libncurses5-dev' instead of 'ncurses-dev'
The following additional packages will be installed:
  libtinfo-dev
Suggested packages:
  ncurses-doc
The following NEW packages will be installed:
  libncurses5-dev libtinfo-dev
0 upgraded, 2 newly installed, 0 to remove and 306 not upgraded.
Need to get 252 kB of archives.
After this operation, 1,461 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://in.archive.ubuntu.com/ubuntu xenial/main amd64 libtinfo-dev amd64 6.
↪0+20160213-1ubuntu1 [77.4 kB]
Get:2 http://in.archive.ubuntu.com/ubuntu xenial/main amd64 libncurses5-dev amd64 6.
↪0+20160213-1ubuntu1 [175 kB]
Fetched 252 kB in 0s (255 kB/s)
Selecting previously unselected package libtinfo-dev:amd64.
(Reading database ... 209625 files and directories currently installed.)
Preparing to unpack .../libtinfo-dev_6.0+20160213-1ubuntu1_amd64.deb ...
Unpacking libtinfo-dev:amd64 (6.0+20160213-1ubuntu1) ...
Selecting previously unselected package libncurses5-dev:amd64.
Preparing to unpack .../libncurses5-dev_6.0+20160213-1ubuntu1_amd64.deb ...
Unpacking libncurses5-dev:amd64 (6.0+20160213-1ubuntu1) ...
Processing triggers for man-db (2.7.5-1) ...
Setting up libtinfo-dev:amd64 (6.0+20160213-1ubuntu1) ...
Setting up libncurses5-dev:amd64 (6.0+20160213-1ubuntu1) ...
rishi@rishi-VirtualBox:~/lkw/linux-4.7$
```

- Now when we run `make menuconfig` we will get the following on terminal and a ncurses based screen will open.

```
$ make menuconfig
HOSTCC  scripts/kconfig/mconf.o
HOSTCC  scripts/kconfig/zconf.tab.o
HOSTCC  scripts/kconfig/lxdialog/checklist.o
HOSTCC  scripts/kconfig/lxdialog/util.o
HOSTCC  scripts/kconfig/lxdialog/inputbox.o
HOSTCC  scripts/kconfig/lxdialog/textbox.o
HOSTCC  scripts/kconfig/lxdialog/yesno.o
HOSTCC  scripts/kconfig/lxdialog/menubox.o
HOSTLD  scripts/kconfig/mconf
scripts/kconfig/mconf  Kconfig
```

- We will now configure our kernel.
- We will add `EXT2` and `EXT3` as kernel modules. We will then compare the default config file and the currently generated config file to see the effect of the changes. We will also remove the VFAT support and add the NTFS support to the kernel image directly. There is no particular reason for doing all this. All this is intended to teach you how the configuration of kernel works and what are the effect of it. Once the kernel boots we will see how these changes effect the booted kernel.

We will now do some configuration changes to the new kernel which we will just compile and configure.

- **Goto - `File systems` ->** mark `Ext2` as module i.e. `M` use spacebar to toggle between the possible values mark `Ext3` as a built into images i.e. `*`

- **Goto - `File Systems` -> `DOS/NT Filesystem`** remove `VFAT` support i.e. `BLANK` add `NTFS` module support i.e. `M`

- Go back using `<esc> <esc>`
- Save the configuration, you will get a `.config` file in your directory.
- There a ton of features which are configurable. You should just go inside some of them and see what is available and what are the configuration option. Do it !!

- Let us see the difference in the current config of the system and the config file which is generated by us. We are insterested in seeing the entries which must have been modified because of us, hence we are `grep-ing` those.

```
$ diff -y /boot/config-4.4.0-24-generic .config | grep -a "EXT2\|EXT3\|VFAT\|NTFS"

# CONFIG_EXT2_FS is not set                             | CONFIG_EXT2_FS=m
# CONFIG_EXT3_FS is not set                             | # CONFIG_EXT2_FS_XATTR is␣
↪not set
                                                        > CONFIG_EXT3_FS=m
                                                        > # CONFIG_EXT3_FS_POSIX_
↪ACL is not set
                                                        > # CONFIG_EXT3_FS_SECURITY␣
↪is not set
CONFIG_EXT4_USE_FOR_EXT2=y                              <
CONFIG_MSDOS_FS=m                                       | # CONFIG_VFAT_FS is not␣
↪set
CONFIG_VFAT_FS=y                                        | CONFIG_NTFS_FS=y
CONFIG_FAT_DEFAULT_CODEPAGE=437                                 | CONFIG_NTFS_
↪DEBUG=y
CONFIG_FAT_DEFAULT_IOCHARSET="iso8859-1"                | CONFIG_NTFS_RW=y
CONFIG_NTFS_FS=m                                        <
# CONFIG_NTFS_DEBUG is not set                          <
# CONFIG_NTFS_RW is not set                             <
```

- Let us see the difference between the default configuration file of the kernel and our configuration file.

```
$ diff -y .config ./arch/x86/configs/x86_64_defconfig | grep -i
↪"EXT2\|EXT3\|NTFS\|VFAT"
# CONFIG_EXT2_FS is not set                   <
# CONFIG_EXT3_FS is not set                   <
CONFIG_EXT4_USE_FOR_EXT2=y                    <
CONFIG_VFAT_FS=y                            CONFIG_VFAT_FS=y
# CONFIG_NTFS_FS is not set
```

- So we have now configured the kernel. Mostly we have changed some of the file system related settings and not made much changes. We will now start with the compilation.
- `make -j 4` - this will start the compilation of the linux kernel. `-j` option runs the `make` in a multithreaded fashion. `4` here stands for the number of threads. For selecting the number of threads you can see the number of cores in your virtual machine. The file `/proc/cpuinfo` has the information about cpus. Generally its a good idea to have 2 threads per cpu. i.e for a 2 cpu machine have 4 threads. You can keep more threads per cpu.
- Errors I faced - this was due to the openssl library missing from the system. It can be installed by the command `sudo apt-get install libssl-dev`. Install the package and restart the compilation process.

```
$ make -j 4

scripts/kconfig/conf  --silentoldconfig Kconfig
  SYSHDR  arch/x86/entry/syscalls/../../include/generated/asm/unistd_32_ia32.h
  SYSTBL  arch/x86/entry/syscalls/../../include/generated/asm/syscalls_32.h
  CHK     include/config/kernel.release
  SYSHDR  arch/x86/entry/syscalls/../../include/generated/asm/unistd_64_x32.h

  >>>>>>>> SNIPPED <<<<<<<<<<<<

  HOSTCC  scripts/selinux/genheaders/genheaders
  HOSTCC  scripts/selinux/mdp/mdp
  HOSTCC  scripts/kallsyms
  HOSTLD  scripts/mod/modpost
```

```
  HOSTCC  scripts/conmakehash
  HOSTCC  scripts/recordmcount
  HOSTCC  scripts/sortextable
  HOSTCC  scripts/asn1_compiler
  HOSTCC  scripts/sign-file


scripts/sign-file.c:25:30: fatal error: openssl/opensslv.h: No such file or directory



compilation terminated.
scripts/Makefile.host:91: recipe for target 'scripts/sign-file' failed
make[1]: *** [scripts/sign-file] Error 1
make[1]: *** Waiting for unfinished jobs....
Makefile:558: recipe for target 'scripts' failed
make: *** [scripts] Error 2
make: *** Waiting for unfinished jobs....
make: *** wait: No child processes.  Stop.
```

- I started with `make -j 4` and saw that the processor is still underutilised. Hence I started the 16 threads with `time` command to see the time taken.

```
$ time make -j 16
```

- `make modules` - compiles the modules - this step is not required.
- `make modules_installs` - installs (copies) the modules to the required location usually `/lib/modules/KERNEL-VERSION/kernel`.
- `make install` - installs the kernel in the required location usually `/boot/`, makes the initial ramfs sets up the boot loader, you are now ready to reboot your system.
- If everything goes fine then your kernel will install properly.

BootLoader

- Let us do some settings in grub so that we can see some of the changes.
- Open the file `/etc/default/grub` and change the `GRUB_TIMEOUT` to 60

```
$ head /etc/default/grub
# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.
# For full documentation of the options in this file, see:
#   info -f grub -n 'Simple configuration'

GRUB_DEFAULT=-1
GRUB_HIDDEN_TIMEOUT=
GRUB_HIDDEN_TIMEOUT_QUIET=true
GRUB_TIMEOUT=60
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
```

- This will ensure that the boot menu waits for 60 seconds before it goes to the default selection.
- Let us check the entries using the `grub-customizer` tool. Run the following command and then start the `grub-customizer`

```
sudo add-apt-repository ppa:danielrichter2007/grub-customizer
sudo apt-get update
sudo apt-get install grub-customizer
```

Reference for this http://askubuntu.com/questions/532238/how-do-i-customize-the-grub-2-menu

- When I run the tool I can see the following

- Do not make any changes, just observe that there are new entries for the 4.7 kernel.
- Now let us reboot the system. Following screen will come.

- When you go to the `Advanced options for Ubuntu` you can see the following screen. Here you can choose which kernel to boot manually. There are settings in `grub` which can enable you in making the default kernel as you want.

- When your new kernel boots up run the command `uname -a` to see the current kernel version.

```
$ uname -a
Linux rishi-VirtualBox 4.7.0 #1 SMP Sat Aug 20 09:41:02 IST 2016 x86_64 x86_64 x86_64␣
↪GNU/Linux
```

- It shows that we are into our new kernel. Congratulations !!!

Files Of The New Kernel

- Let us see some of the important files of the newly installed kernel.
- /boot/initrd.img-4.7.0
- /boot/System.map-4.7.0
- /boot/vmlinuz-4.7.0
- /boot/config-4.7.0

```
diff /boot/config-4.7.0 ~/lkw/linux-4.7/.config
```

- `/lib/modules/4.7.0/`
- Symlink

```
ls /lib/modules/4.7.0/build -l
lrwxrwxrwx 1 root root 25 Aug 20 09:43 /lib/modules/4.7.0/build -> /home/rishi/lkw/
↪linux-4.7
```

- /lib/modules/4.7.0/modules.dep
- /lib/modules/4.7.0/modules.order
- /lib/modules/4.7.0/source

Module Loading and Unloading

- We had configured the system to have ext2 file system as a module. So the linux system should not show that it supports the file system unless the module is loaded. Right?
- Let us check this fact by listing the supported file systems. `cat` the file `/proc/filesystems`

```
rishi@rishi-VirtualBox:~$ cat /proc/filesystems
nodev   sysfs
nodev   rootfs
nodev   ramfs
nodev   bdev
nodev   proc
nodev   cpuset
nodev   cgroup
nodev   cgroup2
nodev   tmpfs
nodev   devtmpfs
nodev   binfmt_misc
nodev   debugfs
nodev   tracefs
nodev   sockfs
nodev   pipefs
nodev   hugetlbfs
nodev   rpc_pipefs
nodev   devpts
        ext3
        ext4
        iso9660
nodev   nfs
nodev   nfs4
nodev   autofs
nodev   mqueue
nodev   selinuxfs
```

Note: We will use some commands like `modprobe insmod lsmod rmmod`. Do not worry if you are unable to understand these. In the next chapters I will detail them.

- Let us load the `ext2` file system in the kernel and see what happens. We can do this by the `modprobe` command. `modprobe` is an intelligent tool which knows the exact locations of the modules and it can load them from there. We should do it as `sudo` as we need the `root` privileges. The other tool to insert modules is `insmod`.

```
rishi@rishi-VirtualBox:~$ sudo modprobe ext2
[sudo] password for rishi:
```

- We can check if the module got loaded by running the `lsmod` command. The second column of the `lsmod` command is the usage count. Right now there is no ext2 filesystem which is mounted hence the usage count is `0`.

```
rishi@rishi-VirtualBox:~$ lsmod | grep ext2
ext2                   50017  0
```

- We should now be supporting the `ext2` file system as well. See the last line. You can see the `ext2` file system being supported.

```
rishi@rishi-VirtualBox:~$ cat /proc/filesystems
nodev   sysfs
nodev   rootfs
nodev   ramfs
nodev   bdev
nodev   proc
nodev   cpuset
nodev   cgroup
nodev   cgroup2
nodev   tmpfs
nodev   devtmpfs
nodev   binfmt_misc
nodev   debugfs
nodev   tracefs
nodev   sockfs
nodev   pipefs
nodev   hugetlbfs
nodev   rpc_pipefs
nodev   devpts
        ext3
        ext4
        iso9660
nodev   nfs
nodev   nfs4
nodev   autofs
nodev   mqueue
nodev   selinuxfs
        ext2
```

- Let us now remove the `ext2` module. Use the command `rmmod`.

```
rishi@rishi-VirtualBox:~$ sudo rmmod ext2
```

- Now there will be no entry in `lsmod` for the `ext2` file system.

Automatic Loading of modules when required.

- We will now `mount` a file system of type `ext2` and we will see that the module gets loaded automaticllay.

- First let us see the currently mounted file systems.

```
rishi@rishi-VirtualBox:~$ mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,relatime,size=1042368k,nr_inodes=260592,
↪mode=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,noexec,relatime,size=209632k,mode=755)
/dev/sda1 on / type ext4 (rw,relatime,errors=remount-ro,data=ordered)
selinuxfs on /sys/fs/selinux type selinuxfs (rw,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k)
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,mode=755)
cgroup on /sys/fs/cgroup/systemd type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,
↪release_agent=/lib/systemd/systemd-cgroups-agent,name=systemd)
cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,
↪cpuacct)
cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
systemd-1 on /proc/sys/fs/binfmt_misc type autofs (rw,relatime,fd=34,pgrp=1,timeout=0,
↪minproto=5,maxproto=5,direct)
mqueue on /dev/mqueue type mqueue (rw,relatime)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
tmpfs on /run/user/108 type tmpfs (rw,nosuid,nodev,relatime,size=209632k,mode=700,
↪uid=108,gid=114)
tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,relatime,size=209632k,mode=700,
↪uid=1000,gid=1000)
```

- Now let us create a file which we will use as a disk for making an `ext2` partition.
- `dd` command will help us in this. We will make a 100MB file. Here `if` is the input file name. `/dev/zero` is a device which gives zeros when it is read. Read more about it. `of` is the output filename. `bs` is the blocksize and `count` is the number of blocks we want it to write. `bs*count` is the size of the file made.

```
rishi@rishi-VirtualBox:~$ dd if=/dev/zero of=100mb bs=1M count=100
100+0 records in
100+0 records out
104857600 bytes (105 MB, 100 MiB) copied, 0.0596335 s, 1.8 GB/s
```

- Let us make a file system now.

```
rishi@rishi-VirtualBox:~$ mkfs.ext2 100mb
mke2fs 1.42.13 (17-May-2015)
Discarding device blocks: done
Creating filesystem with 102400 1k blocks and 25688 inodes
Filesystem UUID: acc67a5c-572f-4e83-b0cc-f9a53cbb9f0f
Superblock backups stored on blocks:
    8193, 24577, 40961, 57345, 73729

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
```

- Associate it with a `loop` device. Loop devices are fake devices which allow regular files to be used as block devices. Read about them in `man losetup`.

```
rishi@rishi-VirtualBox:~$ sudo losetup /dev/loop0 100mb
```

- Current status of `ext2` module
- Mounting the `ext2` filesystem we just made.

```
rishi@rishi-VirtualBox:~$ sudo mount /dev/loop0 /mnt
```

- Now the state of `ext2` module. The usage count is 1 here.

```
rishi@rishi-VirtualBox:~$ lsmod | grep ext2
ext2                   50017  1
```

- Status of `mount` command.

```
rishi@rishi-VirtualBox:~$ mount | grep ext2
/dev/loop0 on /mnt type ext2 (rw,relatime,errors=continue)
```

- Status of `/proc/filesystems`

```
rishi@rishi-VirtualBox:~$ cat /proc/filesystems
nodev   sysfs
nodev   rootfs
nodev   ramfs
nodev   bdev
nodev   proc
nodev   cpuset
nodev   cgroup
nodev   cgroup2
nodev   tmpfs
nodev   devtmpfs
nodev   binfmt_misc
nodev   debugfs
nodev   tracefs
nodev   sockfs
nodev   pipefs
nodev   hugetlbfs
nodev   rpc_pipefs
nodev   devpts
        ext3
        ext4
        iso9660
nodev   nfs
nodev   nfs4
nodev   autofs
nodev   mqueue
nodev   selinuxfs
        ext2
```

- Unmounting the file system.

```
rishi@rishi-VirtualBox:~$ sudo umount /mnt
```

- The module does not get unloaded by itself. Its usage count gets to 0 though.

```
rishi@rishi-VirtualBox:~$ lsmod | grep ext2
ext2                   50017  0
```

- We can now remove it ourselves.

```
rishi@rishi-VirtualBox:~$ sudo rmmod  ext2
```

- We can check the supported file systems again. There is not `ext2`

```
rishi@rishi-VirtualBox:~$ cat /proc/filesystems
nodev   sysfs
nodev   rootfs
nodev   ramfs
nodev   bdev
nodev   proc
nodev   cpuset
nodev   cgroup
nodev   cgroup2
nodev   tmpfs
nodev   devtmpfs
nodev   binfmt_misc
nodev   debugfs
nodev   tracefs
nodev   sockfs
nodev   pipefs
nodev   hugetlbfs
nodev   rpc_pipefs
nodev   devpts
        ext3
        ext4
        iso9660
nodev   nfs
nodev   nfs4
nodev   autofs
nodev   mqueue
nodev   selinuxfs
```

Exercises

- Build the `XFS` file system in the kernel. See the effect in the file `/proc/filesystems`
- Build the `reiserfs` file system as module. See the effect in the file `/proc/filesystems`. See the new `.config` file which got generated.
- Make some files and associate them with different loop devices. Mount them, do some operations.

References

- The Design Of Unix Operating System.
- `man` pages are the best references you will find for Linux. Read the `man` pages for `dd`, `mkfs.ext2`, `mount`, `modprobe`. Do not worry about understanding them end to end, just read it. We will detail them in coming chapters or documents.

CHAPTER 4

Kernel Modules

Introduction

We generally do not make changes directly to the core kernel code. We mostly write small modules to be added to the kernel. This way we can separate out our changes from the mainline kernel.

If you see - the file system are thus modules only, as not all the Linux installations do not need all the file systems and the user can load them on demand.

In this chapter we will see how to write kernel modules.

Why this chapter

This chapter is the most important building block of the whole document. So spend some time with it. Do not just copy paste the code.

Type the code and then run it. Read the comments and type them as well so as to understand what the comments mean.

The more time you spend here - the lesser time you will spend later.

What will you learn

We will learn

- How to write a basic kernel module.
- How to write functions and make it available to other modules. Sort of library.
- How to pass parameters to the modules which is getting loaded.

Concepts and Keywords involved

Prerequisites

C Programming.

What are Kernel Modules

- Pieces of code that can be loaded and unloaded into the kernel upon demand.
- Extends the functionality of the kernel without the need to reboot the system.
- Device drivers which allows the kernel to access hardware connected to the system. If we plug any new device the kernel cannot use it properly unless we have the right drivers in place. The kernel detects the device type and loads the respective modules from the code base. Thus device drivers are generally modules (apart from the basic / legacy drivers.)

Kernel Module Advantages

- Without modules, we would have to build monolithic kernels.
- Adding new functionality would require adding the code in the kernel image which will need recompilation.
- Monolithic kernel leads to larger kernel images resulting in loading code which is not required.
- Rebooting machine whenever new functionality is added.
- Example: till the time you don't have to mount a Ext2 file system why will you load code for reading ext2 file system.

Tools for Kernel Modules

Before getting into writing a module we must first see the tools/commands which we will use to get the modules working.

`insmod`

- `insmod` - insert module.

`modprobe`

- `modprobe -r` - remove the module present in the kernel.

`rmmod`

- `rmmod` - remove the module present in the kernel.

lsmod

- `lsmod` - list the modules inserted in the kernel.

modinfo

- `modinfo` - prints the information of the module.

dmesg

- `dmesg`

syslog

- `syslog`

Some System Calls

- `init_module(), query_module(), create_module(), delete_module()` - system calls called by the various commands to insert/delete module to the kernel space.

New Functions

There are few new things which you will see in the code.

printk()

- `printk()` - this is the kernel counter part of the user space `printf()` function. Its syntax is `printk(LEVEL "FORMAT_SPECIFIER", parameters)`. The `LEVEL` is used to specify the severity of the message you wish to print. See different_log_levels_printk-label.

module_init()

- `module_init()` - `module_init()` is the first function which will be called when the module is loaded. This gives us a entry point to the kernel module. Here we can initialize the global variables, setup functions, setup enviornment and other stuff. The use of this will be clear in the next chapters.

module_exit()

- `module_exit()` - `module_exit()` is the function which is called when the module is being unloaded from the system. Here we can free the allocated memory, free the locks and do other cleanup actions. This will be clear in the coming chapters.

Hello World Kernel Module

Introduction

We will write our first kernel module. We will code it, compile it and then `insert` the module in the kernel. When the module gets inserted the kernel gets the functionality provided by the module. In this `Hello World` module, their is no functionality provided to the kernel. The module just gets inserted and it prints `Hello World`. You can see the message printed by the kernel module by running the `dmesg` tool. `dmesg` shows you the buffer of the kernel.

Code

FILE: mymodule.c

```
1  /*
2   * <PD> Hello World Module </PD>
3   */
4
5  #include <linux/module.h>
6  #include <linux/init.h>
7
8  /*
9   * This is the starting point of the kernel module's code execution.
10  * Right now we will just print a hello and return from here.
11  */
12
13 static int __init my_init(void)
14 {
15         printk(KERN_INFO "Hello from Hello Module");
16         return 0;
17 }
18
19 /*
20  * This is the exit point of the kernel module. When the module is removed
21  * this function is called to do any sort of cleanup activities and other such
22  *  stuff.
23  *
24  * For example you have made a tree where you keep someinformation - you would
25  * like to place the code for removing the nodes of the tree here.
26  */
27
28 static void __exit my_exit(void)
29 {
30         printk(KERN_INFO "Bye from Hello Module");
31 }
32
33 module_init(my_init);
34 module_exit(my_exit);
35
36 MODULE_DESCRIPTION("Sample Hello World Module");
37 MODULE_AUTHOR("Rishi Agrawal <rishi.b.agrawal@gmail.com>");
38 MODULE_LICENSE("GPL");
```

FILE: Makefile

```
MODULE_FILENAME=mymodule

obj-m +=  $(MODULE_FILENAME).o
KO_FILE=$(MODULE_FILENAME).ko

export KROOT=/lib/modules/$(shell uname -r)/build

modules:
        @$(MAKE) -C $(KROOT) M=$(PWD) modules

modules_install:
        @$(MAKE) -C $(KROOT) M=$(PWD) modules_install

clean:
        @$(MAKE) -C $(KROOT) M=$(PWD) clean
        rm -rf   Module.symvers modules.order

insert: modules
        sudo insmod $(KO_FILE)

remove:
        sudo rmmod $(MODULE_FILENAME)

printlog:
        sudo dmesg -c
        sudo insmod $(KO_FILE)
        dmesg
```

Common Questions

How to run the code

To insert the module we need to use the `insmod` command with the module name. We need to be `root` while doing it.

In the `Makefile` we have a target `insert`. This target calls the `insmod` command to insert the module into the kernel.

How the code gets executed

On inserting the module the function registered as `module_init()` gets called. In our case it is `my_init()`.

This function just prints a message to the logs and returns.

How to remove the module

To remove the module we need to use the `rmmod` command with the module name. We need to be `root` while doing it.

```
rmmod mymodule
```

How the module gets removed/unloaded

When the `rmmod` command is invoked the function registered with `module_exit()` is called. In our case it is `my_exit()`. The function just prints the messages in the logs and then returns.

Module Which Does Some Calculations

Introduction

The above module did not do anything and just printed some messages. We will now write a module which will do some calculations.

The intention of this excercise it to show that its plain C code and you can do the regular stuff here as well.

Code

FILE: mycalc.c

```
/*
 * <PD> Module to demostrate how to do some calculations. </PD>
 */
#include <linux/module.h>
#include <linux/init.h>

struct sample {
    int x, y;
};

void print_sample_struct(struct sample temp) {
    printk (KERN_INFO "\nx is %d y is %d", temp.x, temp.y);
}

void calculator(void) {
    int a = 15, b = 3;
    int *ptra = &a;
    int i;

    struct sample temp;
    temp.x = 10;
    temp.y = 20;

    printk (KERN_INFO "\nSum is           %d", a+b);
    printk (KERN_INFO "\nDifference is    %d", a-b);
    printk (KERN_INFO "\nProduct is       %d", a*b);
    printk (KERN_INFO "\nDivison is       %d", a/b);
    printk (KERN_INFO "\nMod is           %d", a%b);
    printk (KERN_INFO "\nBitwise NOT of %d Is   %d", a, ~a);
    printk (KERN_INFO "\nBitwise OR is    %d", a|b);
    printk (KERN_INFO "\nBitwise AND is   %d", a&b);
    printk (KERN_INFO "\nBitwise XOR Is   %d", a^b);
    printk (KERN_INFO "\nLogical OR Is    %d", a||b);
    printk (KERN_INFO "\nLogical AND Is   %d", a&&b);

    if (a>b) {
```

```
        printk (KERN_INFO "\n%d is greater than %d", a, b);
    } else if (b>a) {
        printk (KERN_INFO "\n%d is greater than %d", b, a);
    } else {
        printk (KERN_INFO "\n%d is equal to %d", b, a);
    }

    printk (KERN_INFO "\nAddress of a is %p", ptra);
    printk (KERN_INFO "\nValue of ptra is %d", *ptra);

    /*
     * You can have loops as well.
     */

    for (i = b; i <=a; i++) {
        printk (KERN_INFO "\nPrinting i %d", i);
    }

    /*
     * You can have structures as well.
     */

    print_sample_struct(temp);

}

static int __init my_init(void)
{
        printk(KERN_INFO "Hello from Hello Module");
    calculator();
        return 0;
}

static void __exit my_exit(void)
{
        printk(KERN_INFO "Bye from Hello Module");
}

module_init(my_init);
module_exit(my_exit);

MODULE_DESCRIPTION("Sample Hello World Module");
MODULE_AUTHOR("Rishi Agrawal <rishi.b.agrawal@gmail.com>");
MODULE_LICENSE("GPL");
```

FILE: Makefile

```
MODULE_FILENAME=mycalc

obj-m +=  $(MODULE_FILENAME).o
KO_FILE=$(MODULE_FILENAME).ko

export KROOT=/lib/modules/$(shell uname -r)/build

compile:
        @$(MAKE) -C $(KROOT) M=$(PWD) modules
```

```

modules_install:
	@$(MAKE) -C $(KROOT) M=$(PWD) modules_install

clean:
	@$(MAKE) -C $(KROOT) M=$(PWD) clean
	rm -rf   Module.symvers modules.order

insert: compile
	sudo insmod $(KO_FILE)

remove:
	sudo rmmod $(MODULE_FILENAME)

printlog:
	sudo dmesg -c
	sudo insmod $(KO_FILE)
	dmesg
```

Module with Parameters

Introduction

In the above calculator if we want to change the variables, or we want to do calculation of different integers we will have to change the code and hardcode the value.

We do not want that, we want that the values should be taken from the user. For this we can pass the paramters to the kernel module to do it.

First we will see a small module which does it and then we will see it in the calculator program.

Code

FILE: mymodule_with_parameters.c

```c
/*
 * <PD> Program to add the passed parameters to a kernel module </PD>
 */
#include <linux/module.h>
#include <linux/init.h>

#define DEFAULT_PARAM1 100
#define DEFAULT_PARAM2 200

/*
 * Variable for integer parameter
 */
int param1 = DEFAULT_PARAM1;
int param2 = DEFAULT_PARAM2;

/*
 * Get the parameters.
 */
```

```
module_param(param2, int, 0);
module_param(param1, int, 0);

static int __init my_init(void)
{
   printk(KERN_INFO "\nHello !! from Paramter Passing Demo Module\n");

   /*
    * Print the parameters passed
    */

   printk(KERN_INFO "\nThe sum of the parameters are :%d:", param1 + param2);

   printk(KERN_INFO "\nPassed Parameters\n");

   if (param1 == DEFAULT_PARAM1) {
      printk(KERN_INFO "\nNothing Passed OR Default Value :%d: for param1 is Passed\n
↪", DEFAULT_PARAM1);
   } else {
      printk(KERN_INFO "\nparam1 passed is :%d:", param1);
   }

   if (param1 == DEFAULT_PARAM2) {
      printk(KERN_INFO "\nNothing Passed OR Default Value :%d: for param1 Passed\n",␣
↪DEFAULT_PARAM2);
   } else {
      printk(KERN_INFO "\nparam2 passed is :%d:", param2);
   }

   return 0;
}

static void __exit my_exit(void)
{
   printk(KERN_INFO "\nBye from Parameter Passing Demo Module");
}

module_init(my_init);
module_exit(my_exit);

MODULE_DESCRIPTION("Module To Demonstrate Module Parameters");
MODULE_AUTHOR("rishi.b.agrawal@gmail.com");
MODULE_LICENSE("GPL v2");
MODULE_VERSION("1.0");
```

FILE: Makefile

```
MODULE_FILENAME=mymodule

obj-m +=  $(MODULE_FILENAME).o
KO_FILE=$(MODULE_FILENAME).ko

export KROOT=/lib/modules/$(shell uname -r)/build

compile: clean
        @$(MAKE) -C $(KROOT) M=$(PWD) modules
```

```
10
11 modules_install:
12         @$(MAKE) -C $(KROOT) M=$(PWD) modules_install
13
14 clean:
15         @$(MAKE) -C $(KROOT) M=$(PWD) clean
16         rm -rf   Module.symvers modules.order
17
18 insert: compile
19         sudo insmod $(KO_FILE) param1=10 param2=20
20
21 remove:
22         sudo rmmod $(MODULE_FILENAME)
23
24 printlog:
25         sudo dmesg -c
26         sudo insmod $(KO_FILE)
27         dmesg
28
29 output:
30         sudo rmmod $(MODULE_FILENAME)
31         sudo dmesg -c
32         sudo insmod $(KO_FILE) param1=10 param2=20
33         sudo dmesg -c
34
35 testsanity: clean compile insert remove
```

Module with Character Array and Arrays as parameters

Introduction

Passing paramters is not limited to integers only, we can pass characters and arrays as well. See the following example to understand how to do it.

Questions

How to pass floats?

Code

FILE: mymodule_with_parameters.c

```
1 /*
2  * <PD> Program to demonstrate arrays and strings for module paramters. </PD>
3  */
4 #include <linux/module.h>
5 #include <linux/init.h>
6
7 #define DEFAULT_PARAM1 100
8 #define ARRAY_LEN 5
9 #define STRING_LEN 10
```

```
10
11  /*
12   * Variable for integer parameter
13   */
14  int param1 = DEFAULT_PARAM1;
15  module_param(param1, int, S_IRUGO | S_IWUSR);
16
17  /*
18   * Variable for named parameters
19   */
20  static int for_myself = 42;
21  module_param_named(for_world, for_myself, int, 0444);
22  MODULE_PARM_DESC(for_world, "For the world");
23
24  /*
25   * Variable for integer array
26   */
27  static int int_array[ARRAY_LEN];
28  int array_len;
29  module_param_array(int_array, int, &array_len, S_IRUGO | S_IWUSR);
30  MODULE_PARM_DESC(int_array, "Integer array for doing nothing");
31
32  /*
33   * Variable for strings
34   */
35  char test[STRING_LEN];
36  module_param_string(test_string, test, STRING_LEN, S_IRUGO | S_IWUSR);
37
38  static int __init my_init(void)
39  {
40     int i = 0;
41     printk(KERN_INFO "\nHello from Hello Module\n");
42     printk(KERN_INFO "\nPassed Parameters\n");
43
44     /*
45      * Print the parameters passed
46      */
47     if (param1 == DEFAULT_PARAM1) {
48        printk(KERN_INFO
49               "\nNothing Passed or Default Value %d for param1 \
50                                 passed\n",
51               DEFAULT_PARAM1);
52     } else {
53        printk(KERN_INFO "\nParam1 passed is %d", param1);
54     }
55
56     /*
57      * Module Parameter named - see the file
58      * /sys/module/-module-name-/parameters
59      */
60     printk(KERN_INFO "\nValue of for_myself is %d", for_myself);
61
62     /*
63      * Integer array as a parameter
64      */
65     for (i = 0; i < array_len; i++) {
66        printk(KERN_INFO "Interger Array element %d is %d", i,
67               int_array[i]);
```

```
    }

    /*
       Print the Character array
     */
    printk(KERN_INFO "\nThe character array passed %s", test);
    return 0;
}

static void __exit my_exit(void)
{
    printk(KERN_INFO "Bye from Hello Module");
}

module_init(my_init);
module_exit(my_exit);

MODULE_DESCRIPTION("module to demonstrate module parameters");
MODULE_AUTHOR("abr");
MODULE_LICENSE("GPL v2");
MODULE_VERSION("1.0");
```

FILE: Makefile

```
MODULE_FILENAME=mymodule_with_parameters

obj-m +=  $(MODULE_FILENAME).o
KO_FILE=$(MODULE_FILENAME).ko

export KROOT=/lib/modules/$(shell uname -r)/build

compile:
        @$(MAKE) -C $(KROOT) M=$(PWD) modules

modules_install:
        @$(MAKE) -C $(KROOT) M=$(PWD) modules_install

clean:
        @$(MAKE) -C $(KROOT) M=$(PWD) clean
        rm -rf   Module.symvers modules.order

insert: compile
        sudo insmod $(KO_FILE)

remove:
        sudo rmmod $(MODULE_FILENAME)

printlog:
        sudo dmesg -c
        sudo insmod $(KO_FILE)
        dmesg
```

Calculator with parameters

Introduction

In this module, we have mostly changed the calculator's code to suite the parameter passing.

Code

FILE: mycalc_with_parameters.c

```
/*
 * <PD> Calculator  with parameters </PD>
 */
#include <linux/module.h>
#include <linux/init.h>

struct sample {
    int x, y;
};

void print_sample_struct(struct sample temp) {
    printk (KERN_INFO "\nx is %d y is %d", temp.x, temp.y);
}


/*
 * Variable for integer parameter
 */
int param1;
module_param(param1, int, 0);

int param2;
module_param(param2, int, 0);

void calculator(int a, int b) {
    int *ptra = &a;
    int i;

    struct sample temp;
    temp.x = 10;
    temp.y = 20;

    printk (KERN_INFO "\nSum is           %d", a+b);
    printk (KERN_INFO "\nDifference is    %d", a-b);
    printk (KERN_INFO "\nProduct is       %d", a*b);
    printk (KERN_INFO "\nDivison is       %d", a/b);
    printk (KERN_INFO "\nMod is           %d", a%b);
    printk (KERN_INFO "\nBitwise NOT of %d Is   %d", a, ~a);
    printk (KERN_INFO "\nBitwise OR is    %d", a|b);
    printk (KERN_INFO "\nBitwise AND is   %d", a&b);
    printk (KERN_INFO "\nBitwise XOR Is   %d", a^b);
    printk (KERN_INFO "\nLogical OR Is    %d", a||b);
    printk (KERN_INFO "\nLogical AND Is   %d", a&&b);


```

```
    if (a>b) {
        printk (KERN_INFO "\n%d is greater than %d", a, b);
    } else if (b>a) {
        printk (KERN_INFO "\n%d is greater than %d", b, a);
    } else {
        printk (KERN_INFO "\n%d is equal to %d", b, a);
    }


    printk (KERN_INFO "\nAddress of a is %p", ptra);
    printk (KERN_INFO "\nValue of ptra is %d", *ptra);

    /*
     * You can have loops as well.
     */

    for (i = b; i <=a; i++) {
        printk (KERN_INFO "\nPrinting i %d", i);
    }

    /*
     * You can have structures as well.
     */

    print_sample_struct(temp);

}

/*
 * This is the starting point of the kernel module's code execution.
 * Right now we will just print a hello and return from here.
 */

static int __init my_init(void)
{
    /*
     * Variable for integer parameter
     */
        printk(KERN_INFO "Hello from Hello Module");

    calculator(param1, param2);
        return 0;
}

/*
 * This is the exit point of the kernel module. When the module is removed
 * this function is called to do any sort of cleanup activities and other such
 *  stuff.
 *
 * For example you have made a tree where you keep someinformation - you would
 * like to place the code for removing the nodes of the tree here.
 */

static void __exit my_exit(void)
{
        printk(KERN_INFO "Bye from Hello Module");
}

```

```
104 module_init(my_init);
105 module_exit(my_exit);
106
107 MODULE_DESCRIPTION("Calculator with parameters.");
108 MODULE_AUTHOR("Rishi Agrawal <rishi.b.agrawal@gmail.com>");
109 MODULE_LICENSE("GPL");
```

FILE: Makefile

```
1  MODULE_FILENAME=mycalc_with_parameters
2
3  obj-m +=  $(MODULE_FILENAME).o
4  KO_FILE=$(MODULE_FILENAME).ko
5
6  export KROOT=/lib/modules/$(shell uname -r)/build
7
8  modules:
9          @$(MAKE) -C $(KROOT) M=$(PWD) modules
10
11 modules_install:
12         @$(MAKE) -C $(KROOT) M=$(PWD) modules_install
13
14 clean:
15         @$(MAKE) -C $(KROOT) M=$(PWD) clean
16         rm -rf   Module.symvers modules.order
17
18 insert:
19         sudo insmod $(KO_FILE)
20
21 remove:
22         sudo rmmod $(MODULE_FILENAME)
23
24 printlog:
25         sudo dmesg -c
26         sudo insmod $(KO_FILE)
27         dmesg
```

Conclusion

In this chapter we mostly learnt about the very basics of kernel module programming. We have a lot of ground to cover. Let's get into other concepts of modules.

References

CHAPTER 5

EXPORT_SYMBOL

Introduction

Generally the modules will never live alone. We need to divide the code into multiple modules for better organization and readability as well as we need to use the APIs or functionality which is available in other modules or the functions which are made available to us by the Linux Kernel.

Here in this chapter we will see how to make our functions available to other modules. In latter chapters we will use some functionality given by the other modules or the kernel code.

Why this chapter

We need to understand how to

- For making a function available for others to use.
- How to use functions given by other modules.

What will you learn

- Exporting an API to an external module.
- Using API given by the other module.
- Using library functions available in kernel.

Prerequisites

Previous chapters.

Export Symbol

- `EXPORT_SYMBOL()` helps you provide APIs to other modules/code.
- The functions which you EXPORT are available to the other modules/code.
- Your module will not load if the its expecting a symbol(variable/function) and its not present in the kernel.
- `modprobe` helps here and loads the modules which is needed by your module.
- What if there is circular dependency between the modules?

Module Exporting Some Functions and Variables

Introduction

- Here we will write two modules. In one module we will have the functions which will be exported using the `EXPORT_SYMBOL()` whereas the other module will just call the functions and use the variables which are exported.
- We will then see the details of the module by seeing the `modinfo command`. See the `depends` field of the output. In `mymodule1.ko` you will see that it depends on `mymodule1`.

FILE: mymodule1.c

```
/*
 * <PD> Module 1 for demonstration of circular dependancy </PD>
 */
#include <linux/module.h>
#include <linux/init.h>

int GLOBAL_VARIABLE = 1000;

EXPORT_SYMBOL(GLOBAL_VARIABLE);

/*
 * Function to print hello for num times.
 */
void print_hello(int num)
{
        while (num--) {
                printk(KERN_INFO "Hello Friend!!!\n");
        }
}
EXPORT_SYMBOL(print_hello);

/*
 * Function to add two passed number.
 */
void add_two_numbers(int a, int b)
{
        printk(KERN_INFO "Sum of the numbers %d", a + b);
}

EXPORT_SYMBOL(add_two_numbers);
```

```

static int __init my_init(void)
{
        printk(KERN_INFO "Hello from Export Symbol 1 module.");
        return 0;
}

static void __exit my_exit(void)
{
        printk(KERN_INFO "Bye from Export Symbol 1 module.");
}

module_init(my_init);
module_exit(my_exit);

MODULE_DESCRIPTION("Module to demonstrate the EXPORT_SYMBOL functionality");
MODULE_AUTHOR("Rishi Agrawal <rishi.b.agrawal@gmail.com");
MODULE_LICENSE("GPL v2");
```

FILE: mymodule2.c

```
/*
 * <PD> Module 2 for exporting symbol demostration </PD>
 */
#include <linux/module.h>
#include <linux/init.h>

extern void print_hello(int);
extern void add_two_numbers(int, int);
extern int GLOBAL_VARIABLE;

/*
 * The function has been written just to call the functions which are in other module.
↪ This way you can also write modules which does provide some functionality to the␣
↪other modules.
 */
static int __init my_init(void)
{
    printk(KERN_INFO "Hello from Hello Module");
    print_hello(10);
    add_two_numbers(5, 6);
    printk(KERN_INFO "Value of GLOBAL_VARIABLE %d", GLOBAL_VARIABLE);
    return 0;
}

static void __exit my_exit(void)
{
    printk(KERN_INFO "Bye from Hello Module");
}

module_init(my_init);
module_exit(my_exit);

MODULE_DESCRIPTION("Module to demonstrate the EXPORT_SYMBOL functionality");
MODULE_AUTHOR("Rishi Agrawal <rishi.b.agrawal@gmail.com>");
MODULE_LICENSE("GPL v2");
```

FILE: Makefile

```
obj-m +=  mymodule1.o
obj-m +=  mymodule2.o

export KROOT=/lib/modules/$(shell uname -r)/build

allofit:  modules
modules:
        @$(MAKE) -C $(KROOT) M=$(PWD) modules
modules_install:
        @$(MAKE) -C $(KROOT) M=$(PWD) modules_install
kernel_clean:
        @$(MAKE) -C $(KROOT) M=$(PWD) clean

clean: kernel_clean
        rm -rf   Module.symvers modules.order
```

- Let us see what `modinfo` tells about our modules. Before this compile the modules.

```
$ modinfo mymodule1.ko
filename:       /home/rishi/mydev/publications/lkw/doc/code/04_exporting_symbols/
↪exporting_symbols/mymodule1.ko
license:        GPL v2
author:         Rishi Agrawal <rishi.b.agrawal@gmail.com
description:    Module to demonstrate the EXPORT_SYMBOL functionality
depends:
vermagic:       4.7.0 SMP mod_unload

$ modinfo mymodule2.ko
filename:       /home/rishi/mydev/publications/lkw/doc/code/04_exporting_symbols/
↪exporting_symbols/mymodule2.ko
license:        GPL v2
author:         Rishi Agrawal <rishi.b.agrawal@gmail.com>
description:    Module to demonstrate the EXPORT_SYMBOL functionality
depends:        mymodule1    <<<<<<<<<<<<<<<
vermagic:       4.7.0 SMP mod_unload
```

- Let us try to insert the `mymodule2.ko` before the `mymodule1.ko`. It will give errors.

```
$ sudo insmod mymodule2.ko
insmod: ERROR: could not insert module mymodule2.ko: Unknown symbol in module

rishi@rishi-VirtualBox:~/mydev/publications/lkw/doc/code/04_exporting_symbols/
↪exporting_symbols$ dmesg
[15588.009164] mymodule2: Unknown symbol add_two_numbers (err 0)
[15588.009171] mymodule2: Unknown symbol GLOBAL_VARIABLE (err 0)
[15588.009176] mymodule2: Unknown symbol print_hello (err 0)
```

- Now insert the `mymodule1.ko`

```
$ sudo insmod mymodule1.ko
```

- Now insert the `mymodule2.ko`

```
$ sudo insmod mymodule2.ko
[15606.692155] Hello from Export Symbol 1 module.
[15612.175760] Hello from Hello Module
```

```
[15612.175764] Hello Friend!!!
[15612.175766] Hello Friend!!!
[15612.175767] Hello Friend!!!
[15612.175769] Hello Friend!!!
[15612.175770] Hello Friend!!!
[15612.175772] Hello Friend!!!
[15612.175773] Hello Friend!!!
[15612.175775] Hello Friend!!!
[15612.175776] Hello Friend!!!
[15612.175778] Hello Friend!!!
[15612.175780] Sum of the numbers 11
[15612.175782] Value of GLOBAL_VARIABLE 1000
```

- `SUCESSS !!` You have successfully inserted a module which uses functions from another module.

Removing the modules

- You cannot remove a module which is in use.

```
$ sudo rmmod mymodule1
rmmod: ERROR: Module mymodule1 is in use by: mymodule2
```

- Check who is using the `mymodule1`. See the `Used by` column in the `lsmod` output.

```
$ lsmod
Module                  Size  Used by
mymodule2               1056  0
mymodule1               1324  1 mymodule2
```

- We will have to remove the `mymodule2` first and `mymodule1`.

```
$ sudo rmmod mymodule2
$ sudo rmmod mymodule1
```

Other files

- See the `Module.order` file. It has the order in which the modules should be loaded.

```
$ cat modules.order
kernel//home/rishi/mydev/publications/lkw/doc/code/04_exporting_symbols/exporting_
↪symbols/mymodule1.ko
kernel//home/rishi/mydev/publications/lkw/doc/code/04_exporting_symbols/exporting_
↪symbols/mymodule2.ko
```

- See the `Module.symvers` file. It shows the symbols which are exported.

```
$ cat Module.symvers
0x00000000  print_hello /home/rishi/mydev/publications/lkw/doc/code/04_exporting_
↪symbols/exporting_symbols/mymodule1    EXPORT_SYMBOL
0x00000000  add_two_numbers /home/rishi/mydev/publications/lkw/doc/code/04_exporting_
↪symbols/exporting_symbols/mymodule1    EXPORT_SYMBOL
0x00000000  GLOBAL_VARIABLE /home/rishi/mydev/publications/lkw/doc/code/04_exporting_
↪symbols/exporting_symbols/mymodule1    EXPORT_SYMBOL
```

See the exported symbols

- Module1 exports the symbols.
- The exported symbols and other functions in the kernel can be seen in the `/proc/kallsyms` file. Its is a huge file.
- Let us see the difference in the file after inserting the `mymodule1.ko`.
- That file clearly that the functions `print_hello()` and others are from `mymodule1`.
- The UpperCase `T` says that the functions are exported (available for others to use) while a lowercase says its not `exported`.
- Run the following commands to make two files with the list of symbols.

  ```
  cat /proc/kallsyms > /tmp/1
  ```

- Insert our module.

  ```
  insmod mymodule1.ko
  ```

- Save the symbols in another file.

  ```
  cat /proc/kallsyms > /tmp/2
  ```

- See the difference.

```
diff /tmp/1 /tmp/2
41353a41354,41357
> 0000000000000000 t my_exit    [mymodule1]
> 0000000000000000 t cleanup_module [mymodule1]
> 0000000000000000 T add_two_numbers    [mymodule1]
> 0000000000000000 T print_hello    [mymodule1]
 $ ~/mydev/publications/lkw/doc/code/04_exporting_symbols/exporting_symbols
```

Tool modprobe

- `modprobe` understands in which order the modules are to be loaded.
- First remove the modules.
- Run the command `modprobe module2` loads both the module.

```
$ sudo modprobe mymodule2
modprobe: FATAL: Module mymodule2 not found in directory /lib/modules/4.7.0

$ sudo modprobe mymodule2.ko
modprobe: FATAL: Module mymodule2.ko not found in directory /lib/modules/4.7.0

$ sudo modprobe -C . mymodule2.ko
modprobe: FATAL: Module mymodule2.ko not found in directory /lib/modules/4.7.0

$ man modprobe

$ sudo modprobe -d . mymodule2.ko
modprobe: ERROR: ../libkmod/libkmod.c:586 kmod_search_moddep() could not open moddep␣
↪file '/home/rishi/mydev/publications/lkw/doc/code/04_exporting_symbols/exporting_
↪symbols/./lib/modules/4.7.0/modules.dep.bin'
modprobe: FATAL: Module mymodule2.ko not found in directory /home/rishi/mydev/
↪publications/lkw/doc/code/04_exporting_symbols/exporting_symbols/./lib/modules/4.7.0
```

Tool - depmod

- **DONT RUN IT `depmod` is smart enough to find the dependencies and write to a file - don't run it as it will overwrite the c**
  `depmod ABSOLUTE_PATH_OF_THE_MODULE1 ABSOLUTE_PATH_OF_THE_MODULE2` see the file `/modules/3.2.0-23-generic/modules.dep`

One module dependent on several modules

`modprobe` automatically loading all the modules.

String related functions available in kernel

- There are a lot of function related to string operations available in the Linux Kernel for you to use. They all are exported.
- See the output of `/proc/kallsyms`.

```
$ cat /proc/kallsyms | grep "T str"
0000000000000000 T strndup_user
0000000000000000 T string_to_security_class
0000000000000000 T string_to_av_perm
0000000000000000 T strcasecmp
0000000000000000 T strcpy
0000000000000000 T strncpy
0000000000000000 T strcat
0000000000000000 T strcmp
0000000000000000 T strncmp
0000000000000000 T strchr
0000000000000000 T strchrnul
0000000000000000 T strrchr
0000000000000000 T strnchr
0000000000000000 T strlen
0000000000000000 T strnlen
0000000000000000 T strspn
0000000000000000 T strcspn
0000000000000000 T strpbrk
0000000000000000 T strsep
0000000000000000 T strtobool
0000000000000000 T strstr
0000000000000000 T strnstr
0000000000000000 T strlcpy
0000000000000000 T strncasecmp
0000000000000000 T strnicmp
0000000000000000 T strncat
0000000000000000 T strim
0000000000000000 T strlcat
0000000000000000 T string_get_size
0000000000000000 T string_unescape
0000000000000000 T string_escape_mem
0000000000000000 T strncpy_from_user
0000000000000000 T strnlen_user
```

```
0000000000000000 T strlen_user
0000000000000000 T strict_strtoul_scaled
```

- The count of exported functions is

```
cat /proc/kallsyms | grep "T " | wc -l
18388
```

- Let us now see how can we make use of one of the string functions that is `strcat()`. In the following module we will just concatenate two strings and will print the output.

FILE: mystring.c

```
/*
 * <PD> String functions available in the kernel. </PD>
 */
#include <linux/module.h>
#include <linux/init.h>
#include <string.h>

static int __init my_init(void)
{
    char str1[20] = "Hello";
    char str2[20] = "World";
    char *dest  = NULL;

    dest = strcat(str1, str2);
    printk(KERN_INFO "Concated String is %s", dest);
    return 0;
}

static void __exit my_exit(void)
{
        printk(KERN_INFO "Bye from Hello Module");
}

module_init(my_init);
module_exit(my_exit);

MODULE_DESCRIPTION("Sample Module using string functions.");
MODULE_AUTHOR("Rishi Agrawal <rishi.b.agrawal@gmail.com>");
MODULE_LICENSE("Apache");
```

FILE: Makefile

```
MODULE_FILENAME=mystring

obj-m +=  $(MODULE_FILENAME).o
KO_FILE=$(MODULE_FILENAME).ko

export KROOT=/lib/modules/$(shell uname -r)/build

modules:
        @$(MAKE) -C $(KROOT) M=$(PWD) modules
```

```
10
11 modules_install:
12         @$(MAKE) -C $(KROOT) M=$(PWD) modules_install
13
14 clean:
15         @$(MAKE) -C $(KROOT) M=$(PWD) clean
16         rm -rf   Module.symvers modules.order
17
18 insert:
19         sudo insmod $(KO_FILE)
20
21 remove:
22         sudo rmmod $(MODULE_FILENAME)
23
24 printlog:
25         sudo dmesg -c
26         sudo insmod $(KO_FILE)
27         dmesg
```

Exercises

The exercises here will generally not make much sense with respect to kernel development. You will not be writing a feature for the kernel but you will be learning how to do the basics. So you MUST do it.

1. Write a kernel module to which we can pass a string and it does the following. It must have the functions exported so that another kernel module can use the functions. #. Find the length of the string. `mystring_find_length()` #. Returns the reverse of the string. `mystring_get_reverse()` #. Returns the rotation of the string by 3 places. `char *mystring_get_rotated(char *srcstr, char *deststr, int rotations, int direction)` #. Returns if the string is palindrome or not. `int mystring_is_palindrome(char *str)` #. Returns a character array where you have saved only the characters which are present in the even indexes. #. Returns a string which has all the letter capitalized. #. Returns a string which has all the letter converted to lowercase.
2. For the above kernel module write a testcase module which will call the functions and test if the functions are working correctly.

CHAPTER 6

Proc Interface

Introduction

We have inserted modules to the kernel and can get them do something useful. But how to get some information from the kernel.

We need to be able to get some information from the kernel. There are multiple ways of doing it.

- System Calls
- Proc File System and Sysfs file system
- Netlink Kernel Sockets

In this chapter we will see the Proc Interface. Others are not in the scope of this book.

Why this chapter

This chapter will enable you to understand and write your own proc interface. Proc interface is being used by kernel to communicate information to the userspace and take input from the userspace. For example: file `/proc/mounts` is a file which lists the file systems which are mounted on the system.

Proc insterface is useful to communicate some information to the kernel space as well. For example the file `/proc/sysrq-trigger` gives you ability to ask the kernel to behave differently. For example

`echo l > /proc/sysrq-trigger` will print the backtraces of all cpus. The file `linux-4.7/Documentation/sysrq.txt` describes this in more detail.

For more details on proc interface read `https://www.kernel.org/doc/Documentation/filesystems/proc.txt`.

Thus - we can also use the proc interface to talk to the user space. In the next chapter on BUGS, we will see how a write to a proc interface can trigger different function calls in the kernel space. In the next chapter on linked list, we will see how we can get some work done using the proc interface. Understanding the code of this chapter is thus important as it becomes the ground work of next chapters.

What will you learn

- Writing a proc interface.

Concepts and Keywords involved

Prerequisites

None

New Functions

`kmalloc()`

- kmalloc() is the kernel space version of malloc()
- call it **kmalloc(bytes, GFP_KERNEL)**
- Read more about it in `Linux Kernel Develpement, Robert Love`

Proc entry to read and write data to it

```
/*
 * Module to create a proc entry. The user can read and write to the proc entry.
 */

#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <linux/sched.h>
#include <linux/mm.h>
#include <linux/slab.h>
#include <asm/uaccess.h>
#include <asm/types.h>

#define DATA_SIZE 1024 // We can keep 1024 bytes of data with us.
#define MY_PROC_ENTRY "my_proc_entry_write"
#define PROC_FULL_PATH "/proc/my_proc_entry_write"

struct proc_dir_entry *proc;
int len;
char *msg = NULL;

/*
 * Function to write to the proc. Here we free get the new value from buffer,
 * count from the buffer and then overwrite the data in our file.
 *
 * Note that - you can have any other implementation as well for this, all you have to
 * ensure that you comply with the expectations of the write() system calls
 * like filling in the buffer, and returning the numbers of character written.
 */

```

```
31 static ssize_t my_proc_write(struct file *filp, const char __user * buffer, size_t␣
   ↪count, loff_t *pos)
32 {
33     int i;
34     char *data = PDE_DATA(file_inode(filp));
35 
36     if (count > DATA_SIZE) {
37         return -EFAULT;
38     }
39 
40     printk(KERN_INFO "Printing the data passed. Count is %lu", (size_t) count);
41     for (i=0; i < count; i++) {
42         printk(KERN_INFO "Index: %d . Character: %c Ascii: %d", i, buffer[i],␣
   ↪buffer[i]);
43     }
44 
45     printk(KERN_INFO "Writing to proc");
46     if (copy_from_user(data, buffer, count)) {
47         return -EFAULT;
48     }
49 
50     data[count-1] = '\0';
51 
52     printk(KERN_INFO "msg has been set to %s", msg);
53     printk(KERN_INFO "Message is: ");
54     for (i=0; i < count; i++) {
55         printk(KERN_INFO "\n Index: %d . Character: %c", i, msg[i]);
56     }
57 
58     *pos = (int) count;
59     len = count-1;
60 
61     return count;
62 }
63 
64 /*
65  * Function to read the proc entry, here we copy the data from our proc entry
66  * to the buffer passed.
67  */
68 
69 ssize_t my_proc_read(struct file *filp,char *buf, size_t count, loff_t *offp )
70 {
71     int err;
72     char *data = PDE_DATA(file_inode(filp));
73 
74     if ((int) (*offp) > len) {
75         return 0;
76     }
77 
78     printk(KERN_INFO "Reading the proc entry, len of the file is %d", len);
79     if(!(data)) {
80         printk(KERN_INFO "NULL DATA");
81         return 0;
82     }
83 
84     if (count == 0) {
85         printk(KERN_INFO "Read of size zero, doing nothing.");
86         return count;
```

```
87      } else {
88          printk(KERN_INFO "Read of size %d", (int) count);
89      }
90
91      count = len + 1; // +1 to read the \0
92      err = copy_to_user(buf, data, count); // +1 for \0
93      printk(KERN_INFO "Read data : %s", buf);
94      *offp = count;
95
96      if (err) {
97          printk(KERN_INFO "Error in copying data.");
98      } else {
99          printk(KERN_INFO "Successfully copied data.");
100     }
101
102     return count;
103 }
104
105
106 /*
107  * The file_operations structure. This is the glue layer which associates the
108  * proc entry to the read and write operations.
109  */
110 struct file_operations proc_fops = {
111     .read = my_proc_read,
112     .write = my_proc_write,
113 };
114
115
116 /*
117  * This function will create the proc entry. This function will allocate some
118  * data where the data will be written incase of a write to the proc entry. The
119  * same memory will be used to serve the reads.  * Initially the function fills
120  * the data with DATA which has "Hello People".
121
122  * The important function to see here is the proc_create_data, this function
123  * will take the proc entry name and create it with the given permissions
124  * (0666). We also need to pass the file_operations structure which has the
125  * function pointers to the functions which needs to be called when read or
126  * write is called on the file. The last argument has the pointer to the data
127  * associated with the file.
128  */
129
130 int create_new_proc_entry(void) {
131     int i;
132     char *DATA = "Hello People";
133     len = strlen(DATA);
134     msg = kmalloc((size_t) DATA_SIZE, GFP_KERNEL); // +1 for \0
135
136     if (msg != NULL) {
137         printk(KERN_INFO "Allocated memory for msg");
138     } else {
139         return -1;
140     }
141
142     strncpy(msg, DATA, len+1);
143     for (i=0; i < len +1 ; i++) {
144         printk(KERN_INFO "%c", msg[i]);
```

```
        if (msg[i] == '\0') {
            printk(KERN_INFO "YES");
        }
    }
    proc = proc_create_data(MY_PROC_ENTRY, 0666, NULL, &proc_fops, msg);
    if (proc) {
        return 0;
    }
    return -1;
}


/* The init function of the module. Does nothing other than calling the
 * create_new_proc_entry. */

int proc_init (void) {
    if (create_new_proc_entry()) {
        return -1;
    }
    return 0;
}

/* Function to remove the proc entry.  Call this when the module unloads. */
void proc_cleanup(void) {
    remove_proc_entry(MY_PROC_ENTRY, NULL);
}

MODULE_LICENSE("GPL");
module_init(proc_init);
module_exit(proc_cleanup);
```

```
MYPROC=myproc
obj-m += $(MYPROC).o

export KROOT=/lib/modules/$(shell uname -r)/build
#export KROOT=/lib/modules/$(uname)3.2.0-23-generic/build

allofit:  modules

modules: clean

        @$(MAKE) -C $(KROOT) M=$(PWD) modules

modules_install:
        @$(MAKE) -C $(KROOT) M=$(PWD) modules_install

kernel_clean:
        @$(MAKE) -C $(KROOT) M=$(PWD) clean

clean: kernel_clean
        rm -rf   Module.symvers modules.order

insert: modules
        sudo dmesg -c
        sudo insmod myproc.ko

remove: clean
        sudo rmmod myproc
```

```
28
29
```

- Let us first insert the module by running `make insert`
- Let us print the original value.

```
cat /proc/my_proc_entry_write
Hello People$
```

- Let us now write on the original value.

```
# echo "I am writing this" > /proc/my_proc_entry_write
# cat /proc/my_proc_entry_write
I am writing this
```

CHAPTER 7

Kernel's Bug Reporting

- Kernel has built-in functions/macros for BUGS
- BUG(), BUG_ON(), dump_stack() and panic() can be used in your code to report error conditions.
- For more details on these function read the chapter `Debugging` in the book `Linux Kernel Development, 3rd Edition, Robert love`.
- This chapter will give you example with the proc interface on how to use the debugging facilities given in the kernel.

BUG(), BUG_ON(), dump_stack(), panic() example

```
// Module to make a read entry in the proc file system.
// Module to write a command line calculator
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <linux/sched.h>
#include <linux/mm.h>
#include <linux/slab.h>
#include <asm/uaccess.h>
#include <asm/types.h>

#define MY_PROC_ENTRY "my_bugon_driver"

// #define DATA "Hello World People !!!"

struct proc_dir_entry *proc;
int len;
char *msg = NULL;
#define DATA_SIZE 1024 // We can keep 1024 bytes of data with us.

/*
```

```
 * Function to write to the proc. Here we free the old data, and allocate new space and copy the data to
 * that newly allocated area.
 */

#define MY_BUG_ON 1
#define MY_BUG 2
#define MY_DUMPSTACK 3
#define MY_PANIC 4
static int param = 100;
static ssize_t my_proc_write(struct file *filp, const char __user * buffer, size_t count, loff_t *pos)
{
        char *str;
        str = kmalloc((size_t) count, GFP_KERNEL);
        if (copy_from_user(str, buffer, count)) {
                kfree(str);
                return -EFAULT;
        }
        sscanf(str, "%d", &param);
        pr_info("param has been set to %d\n", param);
        kfree(str);

        switch (param) {
        case MY_BUG_ON:
                BUG_ON(param);
                break;
        case MY_BUG:
                BUG();
                break;
        case MY_DUMPSTACK:
                dump_stack();
                break;
        case MY_PANIC:
                panic("I am panicking, Why? -- you told so");
                break;
        }
        return count;
}

ssize_t my_proc_read(struct file *filp,char *buf,size_t count, loff_t *offp )
{
    int err;
    char *data = PDE_DATA(file_inode(filp));

    if ((int) (*offp) > len) {
        return 0;
    }
    printk(KERN_INFO "Reading the proc entry, len of the file is %d", len);

    if(!(data)) {
        printk(KERN_INFO "NULL DATA");
        return 0;
    }

    if (count == 0) {
        printk(KERN_INFO "Read of size zero, doing nothing.");
        return count;
```

```
78      } else {
79          printk(KERN_INFO "Read of size %d", (int) count);
80      }
81
82      count = len + 1; // +1 to read the \0
83      err = copy_to_user(buf, data, count); // +1 for \0
84      printk(KERN_INFO "Read data : %s", buf);
85      *offp = count;
86
87      if (err) {
88          printk(KERN_INFO "Error in copying data.");
89      } else {
90          printk(KERN_INFO "Successfully copied data.");
91      }
92
93      return count;
94  }
95
96  struct file_operations proc_fops = {
97      .read = my_proc_read,
98      .write = my_proc_write,
99  };
100
101 int create_new_proc_entry(void) {
102     int i;
103     char *DATA = "Hello People";
104     len = strlen(DATA);
105     msg = kmalloc((size_t) DATA_SIZE, GFP_KERNEL); // +1 for \0
106
107     if (msg != NULL) {
108         printk(KERN_INFO "Allocated memory for msg");
109     } else {
110         return -1;
111     }
112
113     strncpy(msg, DATA, len+1);
114     for (i=0; i < len +1 ; i++) {
115         printk(KERN_INFO "%c", msg[i]);
116         if (msg[i] == '\0') {
117             printk(KERN_INFO "YES");
118         }
119     }
120     proc = proc_create_data(MY_PROC_ENTRY, 0666, NULL, &proc_fops, msg);
121     if (proc) {
122         return 0;
123     }
124     return -1;
125 }
126
127 int proc_init (void) {
128     if (create_new_proc_entry()) {
129         return -1;
130     }
131     return 0;
132 }
133
134 void proc_cleanup(void) {
135     remove_proc_entry(MY_PROC_ENTRY, NULL);
```

```
}

MODULE_LICENSE("GPL");
module_init(proc_init);
module_exit(proc_cleanup);

```

Makefile

```
MYPROC=mybugondriver
obj-m += $(MYPROC).o

export KROOT=/lib/modules/$(shell uname -r)/build
#export KROOT=/lib/modules/$(uname)3.2.0-23-generic/build

allofit:  modules

modules: clean

        @$(MAKE) -C $(KROOT) M=$(PWD) modules

modules_install:
        @$(MAKE) -C $(KROOT) M=$(PWD) modules_install

kernel_clean:
        @$(MAKE) -C $(KROOT) M=$(PWD) clean

clean: kernel_clean
        rm -rf   Module.symvers modules.order

insert: modules
        sudo dmesg -c
        sudo insmod mybugondriver.ko

remove: clean
        sudo rmmod mybugondriver


```

Running the code

To run the code you will have to write to the `proc` entry. Based on the value written the system will behave differently.

You can see the output in the `dmesg` output.

```
$ make insert
make[1]: Entering directory '/usr/src/linux-headers-4.4.0-62-generic'
CLEAN   /home/rishi/doc_linux_kernel_workbook/doc/code/05_bug_reporting/01_bugon/.tmp_
↪versions
CLEAN   /home/rishi/doc_linux_kernel_workbook/doc/code/05_bug_reporting/01_bugon/
↪Module.symvers
make[1]: Leaving directory '/usr/src/linux-headers-4.4.0-62-generic'
rm -rf   Module.symvers modules.order
```

```
make[1]: Entering directory '/usr/src/linux-headers-4.4.0-62-generic'
CC [M]  /home/rishi/doc_linux_kernel_workbook/doc/code/05_bug_reporting/01_bugon/
↪mybugondriver.o
Building modules, stage 2.
MODPOST 1 modules
CC      /home/rishi/doc_linux_kernel_workbook/doc/code/05_bug_reporting/01_bugon/
↪mybugondriver.mod.o
LD [M]  /home/rishi/doc_linux_kernel_workbook/doc/code/05_bug_reporting/01_bugon/
↪mybugondriver.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.4.0-62-generic'
sudo dmesg -c
sudo insmod mybugondriver.ko
```

We will now write 1 to the proc entry.

```
$ echo 1 > /proc/my_bugon_driver
$ dmesg


------------[ cut here ]------------
[ 2936.363067] kernel BUG at /home/rishi/doc_linux_kernel_workbook/doc/code/05_bug_
↪reporting/01_bugon/mybugondriver.c:45!
[ 2936.363101] invalid opcode: 0000 [#1] SMP
[ 2936.363118] Modules linked in: mybugondriver(OE) ppdev vboxvideo ttm drm_kms_
↪helper drm snd_intel8x0 fb_sys_fops snd_ac97_codec syscopyarea ac97_bus sysfillrect␣
↪snd_pcm joydev sysimgblt input_leds snd_timer snd soundcore serio_raw i2c_piix4␣
↪parport_pc vboxguest 8250_fintek parport mac_hid ib_iser rdma_cm iw_cm ib_cm ib_sa␣
↪ib_mad ib_core ib_addr iscsi_tcp libiscsi_tcp libiscsi scsi_transport_iscsi autofs4␣
↪btrfs raid10 raid456 async_raid6_recov async_memcpy async_pq async_xor async_tx xor␣
↪raid6_pq libcrc32c raid1 raid0 multipath linear hid_generic usbhid hid crct10dif_
↪pclmul crc32_pclmul ghash_clmulni_intel aesni_intel aes_x86_64 lrw gf128mul glue_
↪helper ablk_helper cryptd ahci psmouse libahci e1000 pata_acpi fjes video
[ 2936.363407] CPU: 0 PID: 1766 Comm: bash Tainted: G           OE   4.4.0-62-generic
↪#83-Ubuntu
[ 2936.363429] Hardware name: innotek GmbH VirtualBox/VirtualBox, BIOS VirtualBox 12/
↪01/2006
[ 2936.363451] task: ffff8800d8deb800 ti: ffff880035734000 task.ti: ffff880035734000
[ 2936.363471] RIP: 0010:[<ffffffffc03e618e>]  [<ffffffffc03e618e>] my_proc_
↪write+0xae/0xc0 [mybugondriver]
[ 2936.363501] RSP: 0018:ffff880035737e78  EFLAGS: 00010246
[ 2936.363516] RAX: 0000000000000000 RBX: 0000000000000002 RCX: 00000000000032a5
[ 2936.363535] RDX: 00000000000032a4 RSI: ffff88011b41a020 RDI: ffff880116801e00
[ 2936.363553] RBP: ffff880035737e90 R08: 000000000001a020 R09: ffffffffc03e615c
[ 2936.363572] R10: ffffea0004572b40 R11: 0000000000000239 R12: ffff880115cad950
[ 2936.363592] R13: 0000000001189408 R14: 0000000000000002 R15: 0000000000000000
[ 2936.363611] FS:  00007f4b6fc64700(0000) GS:ffff88011b400000(0000)␣
↪knlGS:0000000000000000
[ 2936.363633] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[ 2936.363648] CR2: 00007ffedb58b4b8 CR3: 00000000d597d000 CR4: 00000000000406f0
[ 2936.363669] Stack:
[ 2936.363676]  ffff8800344a4a80 fffffffffffffffb ffff880035737f18 ffff880035737eb0
[ 2936.363699]  ffffffff8127bee2 ffff8800da934400 0000000001189408 ffff880035737ec0
[ 2936.363723]  ffffffff8120e168 ffff880035737f00 ffffffff8120eaf9 ffffffff810caeb1
[ 2936.363746] Call Trace:
[ 2936.363758]  [<ffffffff8127bee2>] proc_reg_write+0x42/0x70
[ 2936.363784]  [<ffffffff8120e168>] __vfs_write+0x18/0x40
[ 2936.363799]  [<ffffffff8120eaf9>] vfs_write+0xa9/0x1a0
[ 2936.364337]  [<ffffffff810caeb1>] ? __raw_callee_save___pv_queued_spin_unlock+0x11/
↪0x20
```

```
[ 2936.365035]  [<ffffffff8120f7b5>] SyS_write+0x55/0xc0
[ 2936.365697]  [<ffffffff818385f2>] entry_SYSCALL_64_fastpath+0x16/0x71
[ 2936.366236] Code: f8 02 74 29 7e 20 83 f8 03 74 11 83 f8 04 75 11 48 c7 c7 e0 70␣
↪3e c0 e8 c5 6f da c0 e8 7c 1a 01 c1 48 89 d8 eb 9c 83 e8 01 75 f6 <0f> 0b 0f 0b 0f␣
↪1f 40 00 66 2e 0f 1f 84 00 00 00 00 00 0f 1f 44
[ 2936.367912] RIP  [<ffffffffc03e618e>] my_proc_write+0xae/0xc0 [mybugondriver]
[ 2936.368464]  RSP <ffff880035737e78>
[ 2936.369000] fbcon_switch: detected unhandled fb_set_par error, error code -16
[ 2936.370289] fbcon_switch: detected unhandled fb_set_par error, error code -16
[ 2936.371596] ---[ end trace 064cb0dbcc2892d3 ]---
```

```
$ echo 2 > /proc/mybugondriver
$ dmesg

------------[ cut here ]------------
[ 3035.436271] kernel BUG at /home/rishi/doc_linux_kernel_workbook/doc/code/05_bug_
↪reporting/01_bugon/mybugondriver.c:48!
[ 3035.437611] invalid opcode: 0000 [#3] SMP
[ 3035.438117] Modules linked in: mybugondriver(OE) ppdev vboxvideo ttm drm_kms_
↪helper drm snd_intel8x0 fb_sys_fops snd_ac97_codec syscopyarea ac97_bus sysfillrect␣
↪snd_pcm joydev sysimgblt input_leds snd_timer snd soundcore serio_raw i2c_piix4␣
↪parport_pc vboxguest 8250_fintek parport mac_hid ib_iser rdma_cm iw_cm ib_cm ib_sa␣
↪ib_mad ib_core ib_addr iscsi_tcp libiscsi_tcp libiscsi scsi_transport_iscsi autofs4␣
↪btrfs raid10 raid456 async_raid6_recov async_memcpy async_pq async_xor async_tx xor␣
↪raid6_pq libcrc32c raid1 raid0 multipath linear hid_generic usbhid hid crct10dif_
↪pclmul crc32_pclmul ghash_clmulni_intel aesni_intel aes_x86_64 lrw gf128mul glue_
↪helper ablk_helper cryptd ahci psmouse libahci e1000 pata_acpi fjes video
[ 3035.442196] CPU: 1 PID: 4793 Comm: bash Tainted: G      D    OE   4.4.0-62-generic
↪#83-Ubuntu
[ 3035.442719] Hardware name: innotek GmbH VirtualBox/VirtualBox, BIOS VirtualBox 12/
↪01/2006
[ 3035.443321] task: ffff8800da7bc600 ti: ffff8800db9bc000 task.ti: ffff8800db9bc000
[ 3035.443858] RIP: 0010:[<ffffffffc03e6190>]  [<ffffffffc03e6190>] my_proc_
↪write+0xb0/0xc0 [mybugondriver]
[ 3035.444982] RSP: 0018:ffff8800db9bfe78  EFLAGS: 00010246
[ 3035.445590] RAX: 0000000000000002 RBX: 0000000000000002 RCX: 0000000000001063
[ 3035.446204] RDX: 0000000000001062 RSI: ffff88011b51a020 RDI: ffff880116801e00
[ 3035.446741] RBP: ffff8800db9bfe90 R08: 000000000001a020 R09: ffffffffc03e615c
[ 3035.447305] R10: ffffea0003653180 R11: 00000000000027d R12: ffff8800d94c6400
[ 3035.447842] R13: 0000000001e05408 R14: 0000000000000002 R15: 0000000000000000
[ 3035.448374] FS:  00007f2dc4769700(0000) GS:ffff88011b500000(0000)␣
↪knlGS:0000000000000000
[ 3035.448945] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[ 3035.449484] CR2: 00007ffed043f1b0 CR3: 00000000d5f6f000 CR4: 00000000000406e0
[ 3035.450035] Stack:
[ 3035.450573]  ffff8800344a4a80 fffffffffffffffb ffff8800db9bff18 ffff8800db9bfeb0
[ 3035.451152]  ffffffff8127bee2 ffff8800da949b00 0000000001e05408 ffff8800db9bfec0
[ 3035.451730]  ffffffff8120e168 ffff8800db9bff00 ffffffff8120eaf9 ffffffff810caeb1
[ 3035.452368] Call Trace:
[ 3035.452911]  [<ffffffff8127bee2>] proc_reg_write+0x42/0x70
[ 3035.453545]  [<ffffffff8120e168>] __vfs_write+0x18/0x40
[ 3035.454400]  [<ffffffff8120eaf9>] vfs_write+0xa9/0x1a0
[ 3035.455002]  [<ffffffff810caeb1>] ? __raw_callee_save___pv_queued_spin_unlock+0x11/
↪0x20
[ 3035.455525]  [<ffffffff8120f7b5>] SyS_write+0x55/0xc0
[ 3035.456044]  [<ffffffff818385f2>] entry_SYSCALL_64_fastpath+0x16/0x71
[ 3035.456587] Code: 74 29 7e 20 83 f8 03 74 11 83 f8 04 75 11 48 c7 c7 e0 70 3e c0␣
↪e8 c5 6f da c0 e8 7c 1a 01 c1 48 89 d8 eb 9c 83 e8 01 75 f6 0f 0b <0f> 0b 0f 1f 40␣
↪00 66 2e 0f 1f 84 00 00 00 00 00 0f 1f 44 00 00
```

```
[ 3035.458285] RIP  [<ffffffffc03e6190>] my_proc_write+0xb0/0xc0 [mybugondriver]
[ 3035.458821]  RSP <ffff8800db9bfe78>
[ 3035.459413] ---[ end trace 064cb0dbcc2892d5 ]---
```

```
$ echo 3 > /proc/my_bugon_driver
$ dmesg
 param has been set to 3
[ 3097.627769] CPU: 1 PID: 4846 Comm: bash Tainted: G      D    OE   4.4.0-62-generic
↪#83-Ubuntu
[ 3097.628428] Hardware name: innotek GmbH VirtualBox/VirtualBox, BIOS VirtualBox 12/
↪01/2006
[ 3097.629052]  0000000000000286 00000000f2a77aaf ffff880035737e68 ffffffff813f7c63
[ 3097.629676]  0000000000000002 ffff8800d94c6780 ffff880035737e90 ffffffffc03e6184
[ 3097.630283]  ffff8800344a4a80 fffffffffffffffb ffff880035737f18 ffff880035737eb0
[ 3097.630882] Call Trace:
[ 3097.631464]  [<ffffffff813f7c63>] dump_stack+0x63/0x90
[ 3097.632046]  [<ffffffffc03e6184>] my_proc_write+0xa4/0xc0 [mybugondriver]
[ 3097.632587]  [<ffffffff8127bee2>] proc_reg_write+0x42/0x70
[ 3097.633036]  [<ffffffff8120e168>] __vfs_write+0x18/0x40
[ 3097.633480]  [<ffffffff8120eaf9>] vfs_write+0xa9/0x1a0
[ 3097.633917]  [<ffffffff810caeb1>] ? __raw_callee_save___pv_queued_spin_unlock+0x11/
↪0x20
[ 3097.634411]  [<ffffffff8120f7b5>] SyS_write+0x55/0xc0
[ 3097.634854]  [<ffffffff818385f2>] entry_SYSCALL_64_fastpath+0x16/0x71
```

The following command will panic the machine `echo 4 > /proc/my_bugon_driver`

CHAPTER 8

Kernel Data Structures

- Kernel gives you linked list and red black tree implementations.
- You need not code your own linked list for your code.
- The linked list is extensively used by the kernel.
- Red Black tree is used in the Completely Fair Schedular.

Using Kernel's linked list for your data structure

Let us write a small application.

The application will show the kernel's linked list in a graphical way. How can we do it.

What we will do

1. We will implement the kernel linked list.
2. We will give a proc interface to user. When the user write's `add [int]` to the file we will add it to the linked list.
3. Writing `delete [int]` will delete the node with the provided value from the linked list.
4. Writing `print` will print the whole linked list. The list can be viewed through the `dmesg` command.

```
1  /*
2   * <PD> Program for linked list manipulation using proc </PD>
3   */
4  #include "ll.h"
5
6  struct list_head todo_list;
7
8  int todo_add_entry(int number)
9  {
10    struct todo_struct *new_node = kmalloc((size_t) (sizeof(struct todo_struct)), GFP_
   ↪KERNEL);
```

```
   if (!new_node) {
      printk(KERN_INFO
            "Memory allocation failed, this should never fail due to GFP_KERNEL␣
↪flag\n");
      return 1;
   }
   new_node->priority = number;
   list_add_tail(&(new_node->list), &todo_list);
   return 0;
}

void show_list(void)
{
   struct todo_struct *entry = NULL;

   list_for_each_entry(entry, &todo_list, list) {
      printk(KERN_INFO "Node is %d\n", entry->priority);
   }
}

int delete_node(int number)
{
   struct todo_struct *entry = NULL;

   list_for_each_entry(entry, &todo_list, list) {
      if (entry->priority == number) {
         printk(KERN_INFO "Found the element %d\n",
                entry->priority);
         list_del(&(entry->list));
         kfree(entry);
         return 0;
      }
   }
   printk(KERN_INFO "Could not find the element %d\n", number);
   return 1;
}
```

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/list.h>
#include <linux/proc_fs.h>
#include <linux/uaccess.h>
#include <linux/slab.h>

#ifndef LL_H
#define LL_H
#define NODE "driver/mmmod"
/*
 * Linked List Node
 */
struct todo_struct {
                struct list_head list;
                int priority;
};

/*
 * Linked list related functions
 */
```

```
void show_list(void);
int todo_add_entry(int);

/*
 * Proc Interface related function
 */

void proc_cleanup(void);
int ll_proc_init (void);
int configure_proc_entry(void);


ssize_t my_proc_read (struct file *filp,char *buf,size_t count,loff_t *offp );
ssize_t my_proc_write(struct file *filp, const char __user * buffer, size_t count,
↪loff_t *pos);

int configure_proc_entry(void);
int delete_node (int);
void destroy(void);
#endif
```

```
/*
 * Module to show how to use the kernel linked list for managing data
 * 1. Add node to linked list
 * 2. Print the list
 * 3. Delete nodes from the list
 * The modules exposes a /proc interface on which if you cat
 * 1. add 5 -- it adds five
 * 2. print -- it prints the linked list
 * 3. delete 5 -- it deletes the node in the linked list
 * 4. destroy -- destroys the whole linked list
 */

#include "ll.h"
extern struct proc_dir_entry *proc;
extern struct list_head todo_list;
static int __init my_init(void)
{

   printk(KERN_INFO "Hello from Linked List Module");

   if (ll_proc_init()) {
      printk(KERN_INFO "Falied to create the proc interface");
      return -EFAULT;
   }

   INIT_LIST_HEAD(&todo_list);
   return 0;
}

static void __exit my_exit(void)
{
   if (proc) {
      proc_cleanup();
      pr_info("Removed the entry");
   }
   printk(KERN_INFO "Bye from Hello Module");
}
```

```

module_init(my_init);
module_exit(my_exit);
MODULE_LICENSE("GPL v2");
MODULE_AUTHOR("abr");
```

```
// Module to make a read entry in the proc file system.
// Module to write a command line calculator

#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <linux/sched.h>
#include <linux/mm.h>
#include <linux/slab.h>
#include <asm/uaccess.h>
#include <asm/types.h>


#include "ll.h"
struct proc_dir_entry *proc;
int len,temp;
char *msg;

#define PROC_NAME "linked_list"

ssize_t my_proc_write(struct file *filp, const char __user * buffer, size_t count,␣
↪loff_t *pos)
{
   char *str;
   char command[20];
   int val = 0;

   printk("Calling Proc Write");
   str = kmalloc((size_t) count, GFP_KERNEL);
   if (copy_from_user(str, buffer, count)) {
      kfree(str);
      return -EFAULT;
   }

   sscanf(str, "%s %d", command, &val);
   printk("First Arguement %s\n", command);
   printk("Second Argument %d\n", val);

   if (!strcmp(command, "add")) {
      /* Add node */
      printk(KERN_INFO "Adding data to linked list %d\n", val);
      todo_add_entry(val);
   }

   if (!strcmp(command, "delete")) {
      /* Delete Node */
      printk(KERN_INFO "Deleting node from linked list %d\n", val);
      if (delete_node(val)) {
         printk(KERN_INFO "Delete failed \n");
      }
   }

```

```
52      if (!strcmp(command, "print")) {
53          /* Print the linked list */
54          printk(KERN_INFO "Printing the linked list\n");
55          show_list();
56      }
57
58      kfree(str);
59      return count;
60  }
61
62  ssize_t my_proc_read (struct file *filp,char *buf,size_t count,loff_t *offp )
63  {
64          char *data;
65          int err;
66          data=PDE_DATA(file_inode(filp));
67          if(!(data)){
68                  printk(KERN_INFO "Null data");
69                  return 0;
70          }
71
72          if(count>temp) {
73                  count=temp;
74          }
75
76          temp=temp-count;
77
78          err = copy_to_user(buf,data, count);
79          if (err) {
80                  printk(KERN_INFO "Error in copying data.");
81          }
82
83          if(count==0) {
84                  temp=len;
85          }
86
87          return count;
88  }
89
90  struct file_operations proc_fops = {
91      .read = my_proc_read,
92      .write = my_proc_write,
93  };
94
95  int create_new_proc_entry(void) {
96          msg="Hello World";
97          proc=proc_create_data(PROC_NAME, 0666, NULL, &proc_fops, msg);
98          len=strlen(msg);
99          temp=len;
100         if (proc) {
101             return 0;
102         } else {
103             return -1;
104         }
105 }
106
107 int ll_proc_init (void) {
108         create_new_proc_entry();
109         return 0;
```

```
}

void proc_cleanup(void) {
        remove_proc_entry("hello", NULL);
}

```

```

obj-m +=  ll_module.o
obj-m +=  ll_proc.o
obj-m +=  ll.o
obj-m += ll_driver.o
ll_driver-objs := ll_module.o ll_proc.o ll.o

export KROOT=/lib/modules/$(shell uname -r)/build
#export KROOT=/lib/modules/$(uname)3.2.0-23-generic/build

allofit:  modules

modules: clean

        @$(MAKE) -C $(KROOT) M=$(PWD) modules

modules_install:
        @$(MAKE) -C $(KROOT) M=$(PWD) modules_install

kernel_clean:
        @$(MAKE) -C $(KROOT) M=$(PWD) clean

clean: kernel_clean
        rm -rf   Module.symvers modules.order

insert: modules
        sudo dmesg -c
        sudo insmod ll_driver.ko

remove: clean
        sudo rmmod ll_driver.ko


```

Examples

- Lets just print the empty list.

```
echo "print" > /proc/linked_list
root@lkw:~/doc_linux_kernel_workbook/doc/code/07_data_structures/01_ll# dmesg
[  578.782533] ll_driver: module verification failed: signature and/or required key␣
↪missing - tainting kernel
[  578.783117] Hello from Linked List Module
[  630.344370] Calling Proc Write
[  630.344376] First Arguement print
[  630.344378] Second Argument 0
[  630.344380] Printing the linked list
```

- Now let us add some nodes.

```
root@lkw:~/doc_linux_kernel_workbook/doc/code/07_data_structures/01_ll# echo "add 10"␣
↪> /proc/linked_list
root@lkw:~/doc_linux_kernel_workbook/doc/code/07_data_structures/01_ll# echo "add 20"␣
↪> /proc/linked_list
root@lkw:~/doc_linux_kernel_workbook/doc/code/07_data_structures/01_ll# echo "add 30"␣
↪> /proc/linked_list
root@lkw:~/doc_linux_kernel_workbook/doc/code/07_data_structures/01_ll# dmesg
[  578.782533] ll_driver: module verification failed: signature and/or required key␣
↪missing - tainting kernel
[  578.783117] Hello from Linked List Module
[  630.344370] Calling Proc Write
[  630.344376] First Arguement print
[  630.344378] Second Argument 0
[  630.344380] Printing the linked list
[  642.095630] Calling Proc WriteFirst Arguement add
[  642.095638] Second Argument 10
[  642.095640] Adding data to linked list 10
[  645.399251] Calling Proc WriteFirst Arguement add
[  645.399256] Second Argument 20
[  645.399257] Adding data to linked list 20
[  647.542762] Calling Proc WriteFirst Arguement add
[  647.542767] Second Argument 30
[  647.542768] Adding data to linked list 30
```

- Let us print again.

```
root@lkw:~/doc_linux_kernel_workbook/doc/code/07_data_structures/01_ll# echo "print" >
↪ /proc/linked_list
```

- Let us delete some nodes.

```
root@lkw:~/doc_linux_kernel_workbook/doc/code/07_data_structures/01_ll# echo "delete␣
↪30" > /proc/linked_list
```

- Let us delete a non existent node.

```
root@lkw:~/doc_linux_kernel_workbook/doc/code/07_data_structures/01_ll# echo "delete 0
↪" > /proc/linked_list
```

- Delete more node.

```
root@lkw:~/doc_linux_kernel_workbook/doc/code/07_data_structures/01_ll# echo "delete␣
↪10" > /proc/linked_list
```

- Print.

```
root@lkw:~/doc_linux_kernel_workbook/doc/code/07_data_structures/01_ll# echo "print" >
↪ /proc/linked_list
```

- Check the `dmesg` commands output.

```
root@lkw:~/doc_linux_kernel_workbook/doc/code/07_data_structures/01_ll# dmesg
[  578.782533] ll_driver: module verification failed: signature and/or required key␣
↪missing - tainting kernel
[  578.783117] Hello from Linked List Module
[  630.344370] Calling Proc Write
```

```
[  630.344376] First Arguement print
[  630.344378] Second Argument 0
[  630.344380] Printing the linked list
[  642.095630] Calling Proc WriteFirst Arguement add
[  642.095638] Second Argument 10
[  642.095640] Adding data to linked list 10
[  645.399251] Calling Proc WriteFirst Arguement add
[  645.399256] Second Argument 20
[  645.399257] Adding data to linked list 20
[  647.542762] Calling Proc WriteFirst Arguement add
[  647.542767] Second Argument 30
[  647.542768] Adding data to linked list 30
[  653.222910] Calling Proc WriteFirst Arguement print
[  653.222916] Second Argument 0
[  653.222918] Printing the linked list
[  653.222919] Node is 10
[  653.222920] Node is 20
[  653.222921] Node is 30
[  659.727435] Calling Proc WriteFirst Arguement delete
[  659.727440] Second Argument 30
[  659.727442] Deleting node from linked list 30
[  659.727443] Found the element 30
[  665.175263] Calling Proc WriteFirst Arguement delete
[  665.175274] Second Argument 0
[  665.175277] Deleting node from linked list 0
[  665.175281] Could not find the element 0
[  665.175282] Delete failed
[  667.134783] Calling Proc WriteFirst Arguement delete
[  667.134789] Second Argument 10
[  667.134791] Deleting node from linked list 10
[  667.134792] Found the element 10
[  668.942912] Calling Proc WriteFirst Arguement print
[  668.942917] Second Argument 0
[  668.942919] Printing the linked list
[  668.942920] Node is 20
```

CHAPTER 9

Assignments

Following are few examples which will enable you in understanding the kernel modules and the flow better. The examples in the booklet can be used to understand the overall flow of the code. Try to code the solutions without any help.

In case you want to add more assignments, please feel free to send a merge request along with the solved assignment.

1. Calculator in kernel space using parameters passed
2. Pass the file name, print the inode details
3. Use proc interface to call different functions.
4. Use proc interface to modify the behaviour of module.
5. Make a linked list in kernel space – pass the values using ioctls
6. Using kernel's red-black tree for your data structures
7. Using proc interface pass the name of the mount point and print the details.
8. Print the superblock information if the mount point is passed
9. Print the super block information of the "/" mounted file system

CHAPTER 10

Making Changes to Kernel Code

We will have to make changes in the kernel in order to add features to it and get the features working. There are two steps to it. Making the changes and then compiling them.

The steps are quite simple

1. Make the changes
2. Compile the kernel.

Compiling the whole kernel is one way to make the changes you did into the kernel. If your changes are in a small module of the code you need not compile the whole code and you can just make that module of the code.

In this section we will make two changes to the kernel.

1. Effecting only a module.
2. Effecting the whole kernel.

- Make the changes in the required file, re-check it.
- `make SUBDIRS="path"` - to compile the particular module.
- `make modules_install` - to install the module, you can manually copy the module to the `/lib/` directory.
- `make install` - update the whole system if required, not required generally.
- Check the working of the code.

Hands-On - Making changes to a small module

- Make changes to the ext2 code. In the `ext2_fsync` function, add a `printk` and compile the code. Copy the modules to the `/lib/XX` directory and test by mounting a ext2 file system. Check `dmesg` for the printed message. Open/Write a file using `vim` to see the effects.
- The diff is based of `linux-3.4.6 Kernel` - use the same kernel version to avoid unneccessary problems at this stage.

Hands-On - Making changes to a code which effects the whole kernel

CHAPTER 11

Device Drivers

- A device driver or software driver is a computer program allowing higher- level computer programs to interact with a hardware device.
- Translator between a hardware device and the applications or operating systems
- Drivers are hardware-dependent and operating-system-specific.
- They usually provide the interrupt handling required for any necessary asynchronous time-dependent

Device Drivers Types

- Character Device Drivers
- Operate on characters as basic unit of input and output
- Accessed in sequential and non-random manner
- Block Device Driver
- Serves blocks of data
- Random access as required by file systems

copy_to_user(), copy_from_user()

- Macros to copy data from user space and copy data to kernel space.

Character Device driver and ioctls

file: chardev.c

```
/*
 *  <PD> Creates a read-only char device that says how many times
 *  you've read from the dev file </PD>
 */

#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <asm/uaccess.h>   /* for put_user */

struct param {
   int number;
   char *ptr[];
};
/*
 *  Prototypes - this would normally go in a .h file
 */
int init_module(void);
void cleanup_module(void);
static int device_open(struct inode *, struct file *);
static int device_release(struct inode *, struct file *);
static ssize_t device_read(struct file *, char *, size_t, loff_t *);
static ssize_t device_write(struct file *, const char *, size_t,\
                                                    loff_t *);
static long device_ioctl(struct file *, unsigned int,
         unsigned long);

static long unlocked_device_ioctl(struct file *, unsigned int,
         unsigned long);
#define SUCCESS 0
#define DEVICE_NAME "sav_dev" /* Dev name as it appears in
                                 /proc/devices*/
#define BUF_LEN 80   /* Max length of the message from the device */

/*
 * Global variables are declared as static,
 * so are global within the file.
 */

static int Major;    /* Major number assigned to our device driver */
static int Device_Open = 0;   /* Is device open?
             * Used to prevent multiple access to device */
static char msg[BUF_LEN]; /* The msg the device will give when asked*/
static char *msg_Ptr;

static struct file_operations fops = {
   .read = device_read,
   .write = device_write,
   .open = device_open,
   .release = device_release,
   .compat_ioctl = device_ioctl,
   .unlocked_ioctl = unlocked_device_ioctl

};

/*
```

```
 * This function is called when the module is loaded
 */
int init_module(void)
{
   Major = register_chrdev(0, DEVICE_NAME, &fops);

   if (Major < 0) {
      printk(KERN_ALERT "Registering char device failed with %d\n",\
             Major);
      return Major;
   }

   printk(KERN_INFO "I got major number %d. To talk to\n", Major);
   printk(KERN_INFO "the driver, create a dev file with\n");
   printk(KERN_INFO "'mknod /dev/%s c %d 0'.\n", DEVICE_NAME, Major);
   printk(KERN_INFO "Try various minor numbers, Try to cat and \
                         echo to the device file\n");
   printk(KERN_INFO "Remove the device file and module when done.\n");

   return SUCCESS;
}

/*
 * This function is called when the module is unloaded
 */
void cleanup_module(void)
{
   /*
    * Unregister the device
    */

   printk(KERN_INFO "\n\nUnregistering the device file.\n");
   unregister_chrdev(Major, DEVICE_NAME);
}

/*
 * Methods
 */

/*
 * Called when a process tries to open the device file, like
 * "cat /dev/mycharfile"
 */
static int device_open(struct inode *inode, struct file *file)
{
   static int counter = 0;

   if (Device_Open)
      return -EBUSY;

   Device_Open++;
   sprintf(msg, "I already told you %d times Hello world!\n",
                                    counter++);
   msg_Ptr = msg;
   try_module_get(THIS_MODULE);

   return SUCCESS;
}
```

```
115
116 /*
117  * Called when a process closes the device file.
118  */
119 static int device_release(struct inode *inode, struct file *file)
120 {
121    Device_Open--;    /* We're now ready for our next caller */
122
123    /*
124     * Decrement the usage count, or else once you opened the file,
125     * you'll
126     * never get get rid of the module.
127     */
128    module_put(THIS_MODULE);
129
130    return 0;
131 }
132
133 /*
134  * Called when a process, which already opened the dev file, attempts
135    to read from it.  */
136 static ssize_t device_read(struct file *filp, /* see include/linux/fs.h   */
137             char *buffer,  /* buffer to fill with data */
138             size_t length, /* length of the buffer     */
139             loff_t * offset)
140 {
141    /*
142     * Number of bytes actually written to the buffer
143     */
144    int bytes_read = 0;
145
146    /*
147     * If we're at the end of the message,
148     * return 0 signifying end of file
149     */
150    if (*msg_Ptr == 0)
151       return 0;
152
153    /*
154     * Actually put the data into the buffer
155     */
156    while (length && *msg_Ptr) {
157
158       /*
159        * The buffer is in the user data segment, not the kernel
160        * segment so "*" assignment won't work.  We have to use
161        * put_user which copies data from the kernel data segment to
162        * the user data segment.
163        */
164       put_user(*(msg_Ptr++), buffer++);
165
166       length--;
167       bytes_read++;
168    }
169
170    /*
171     * Most read functions return the number of bytes put into the buffer
172     */
```

```
   return bytes_read;
}

/*
 * Called when a process writes to dev file: echo "hi" > /dev/hello
 */
static ssize_t
device_write(struct file *filp, const char *buff, size_t len, loff_t * off)
{
   printk(KERN_ALERT "Sorry, this operation isn't supported.\n");
   return -EINVAL;
}

static long device_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
{
        printk(KERN_INFO "In unlocked ioctl\n");
        return 0;
}

static long unlocked_device_ioctl( struct file *file, unsigned int cmd, unsigned long
↪arg)
{
   int numbers, i;
   printk(KERN_INFO "Command Passed is %d", cmd);

   switch (cmd) {
   case 1:
      printk(KERN_INFO "I AM IN COMMAND NUMBER 1");
      numbers = ((struct param *)arg)->number;
      printk(KERN_INFO "Number Passed %d", ((struct param *)arg)->number);
      for (i = 0; i < numbers; i++) {
         printk(KERN_INFO "ARG 1 %s", ((struct param *)arg)->ptr[i]);
      }
      printk(KERN_INFO "I will try to print the values passed to me");
      return 0;
   case 2:
      printk(KERN_INFO "I AM IN COMMAND NUMBER 1");
      printk(KERN_INFO " HEY ITS DONE");
      return 0;

   default:
      printk(KERN_INFO "Unsupported command. Command is %d", cmd);
      return -1;
   }
   return 0;
}
```

`file`: Makefile

```
# Makefile to compile chardev.c
# Also this Makefile is used to learn writing Makefiles

CC = gcc

# We have two types of kernel code.
# 1. Which are staticly compiled in the kernel.
```

```
#         For this type we use the string "obj-y". Note the "y" used here.
# 2. Which gets compiled as a module.
#         For this type we use the string "obj-m". Note the "m" used here.
# So we basically use variables here to specify the value "y" or "m".
# I am using variacle CONFIG_CHARDEV to specify it.
#
CONFIG_CHARDEV = m
MODULE_NAME = chardev
obj-m += chardev.o
#obj-m += chardev.o
#obj-$(CONFIG_CHARDEV) += chardev.o

# obj-m += chardev.o


all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
        ctags -R

clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
        rm -rf tags

```

file: ioctl.c

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <string.h>

#define DEV_NAME_LENGTH 20

struct param {
    int number;
    char **ptr;
};

int main(int argc, char *argv[])
{
    int i;
    char dev_name[DEV_NAME_LENGTH];
    struct param ioctl_params;
    int fd, retval;
    printf("\n\nNumber of arguements passed %d", argc);

    for (i = 0; i < argc; i++) {
        printf("\nARG %d is %s", i, argv[i]);
    }

    if (argc < 4) {
        printf("\n\n Number of arguements is very less");
        printf("\n\n argc 1 should be device name");
        printf("\n\n argc 2 should be the ioctl number");
        printf("\n\n argc 3 onwards should be all the arguements");
        goto exit;
```

```
    } else {
        printf("\n\n Number of arguements is fine .... proceeding");
    }
    /* argc 1 should be device name */
    /* argc 2 should be the ioctl number */
    /* argc 3 onwards should be all the arguements */
    /* Asumming that the parameters are correct */
    ioctl_params.number = argc - 3;
    ioctl_params.ptr = argv[3];

    printf("\nXXXX ARG %d is %s", i, ioctl_params.ptr);
    /* Substract from a single digit number to get the correct digit */
    int ioctl_number = argv[2][0] - 48;

    printf("\n\nioctl number is %d", ioctl_number);

    strcpy(dev_name, argv[1]);
    fd = open(dev_name, O_RDWR);

    /* complain if the open failed */
    if (fd == -1) {
        printf("Error in opening the device %s", dev_name);
        return 1;
    } else {
        printf("\n\nOpened the device file .... proceeding");
    }

    /* Call the ioctl */
    printf("\n\nCalling the ioctl, lets see what happens");
    retval = ioctl(fd, ioctl_number, &ioctl_params);
    if (retval == -1) {
        perror("ioctl : ");
        printf("\n\nIOCTL Failed boss");
    } else {
        printf("\n\nWorks fine");
    }

 exit:
    return 0;
}
```

Steps:

- `insmod chardev.ko`
- `cat /proc/devices | grep sav_dev` - to get the major number of the device, for example 250.
- `mknod /dev/mydev c 250 0`
- `ls -l /dev/mydev`

CHAPTER 12

Indices and tables

- genindex
- modindex
- search