Contents

- Linux Source
- Kernel Configuration and Compile
- Module Programming
Linux source tree

- Architecture
- Drivers
- Filesystem
- Init
- Interprocess Communication
- Kernel
- Memory Management
- Networking
Architecture

- Linux supports several different architectures.
- The `arch` directory contains all the platform specific code necessary to implement low-level system interface.
  - Arm
  - Alpha
  - Athlon
  - i386
  - MIPS
  - SPARC
  - ...

Drivers

- This interaction and controlling of hardware is a small piece of the kernel called **drivers**.
  - cdrom
  - scsi
  - usb
  - char
  - block
  - sound
  - ...

In order for the kernel to know how to interact with the filesystem, it must know the structure of it.

- CramFS
- DevFS
- ext2
- ext3
- FAT
- ...

Filesystem
Init

- Init is the initial process of the Linux kernel.
- All initialization of the kernel in handled in this area.
  - Defining all devices
  - Parsing parameters
  - ...
- It is the main process on any UNIX systems.
Interprocess control

- IPC is a method for the kernel to manage processes and allow them to communicate with each other.
  - Message queue
  - Shared memory
  - Semaphore
Kernel

- It contains the code to provide the other areas of the kernel with ways to communicate.

- Several functions that are used through other subsystem of the kernel.
  - panic
  - printk
  - softirq
  - ...

Memory management

- It is responsible for keeping track of all system memory and its usage by the kernel.
Networking

- Linux supports several protocol suites in the kernel.
  - 802
  - Appletalk
  - ATM
  - BGP
  - Bluetooth
  - Ethernet
  - IPV4
  - ...

Contents

- Linux Source
- Kernel Configuration and Compile
- Module Programming
Kernel configuration

- make oldconfig
- make menuconfig
- make xconfig
Kernel configuration

Linux Kernel Configuration

Arrow keys navigate the menu. <Enter> selects submenus -->. Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help. Legend: [*] built-in [ ] excluded <M> module < > module capable

- Code maturity level options -->
  - General setup -->
  - Loadable module support -->
  - Processor type and features -->
  - Power management options (ACPI, APW) -->
  - Bus options (PCI, PCMCIA, EISA, MCA, ISA) -->
  - Executable file formats -->
  - Device Drivers -->
  - File systems -->
  - Profiling support -->
  - Kernel hacking -->
  - Security options -->
  - Cryptographic options -->
  - Library routines -->
  -->
  - Load an Alternate Configuration File
  - Save Configuration to an Alternate File
Kernel compile

- Kernel 2.4.x and older
  - make menuconfig
  - make dep
  - make bzImage
  - make modules
  - make modules_install
  - make install
Kernel compile

- Kernel 2.6.x
  - make
  - make install

- Clean Kernel Source
  - make clean
  - make mrproper
Kernel version can be broken down into four sections:

- Major version
- Minor version
- Sublevel number
- Extraversions level
Kernel version

- **Even** numbered minor version are stable kernels.
- **Odd** numbered version are development release.
Contents

- Linux Source
- Kernel Configuration and Compile
- Module Programming
Module vs. Application

- **Application**
  - An application performs a single task from beginning to end.
  - An application runs in user space.

- **Module**
  - A module registers itself in order to serve future requests.
  - A module runs in kernel space.
  - The role of a module is to extend kernel functionality.
#define MODULE
#include <linux/module.h>
//------------------------------------------------------
int init_module()
{
    printk("<1>Hello World ...
    return 0;
}
//------------------------------------------------------
void cleanup_module()
{
    printk("<1>Goodbye ...
}

Compiling “Hello World”

root# gcc –c hello.c
root# insmod ./hello.o
Hello World …
root# rmmod hello
Goodbye …
root#
Linking a module to the kernel
Module header files

- Because no library is linked to modules, source files should never include the usual header files.
  - include/asm
  - include/linux
Module compile notes

- `#define __KERNEL__`
  - Much of the kernel specific content in the kernel headers is unavailable without this symbol.

- `#define MODULE`
  - It must define before `#include <linux/module>`

- `-O`
  - gcc doesn’t expand `inline` functions unless optimization is enabled.

- `-Wall`
  - In order to prevent unpleasant errors.
KERNELDIR = /usr/src/include
CFLAGS = -D__KERNEL__ -DMODULE
          -l$(KERNELDIR)/include
-0 -Wall
Version dependency

- Module’s code has to be recomplied for each version of the kernel that it will be linked to.
Version dependency

- Each module defines a symbol called \_module-kernel-version
  - Insmod matches it against the version number of current kernel.
  - The compiler will define this symbol whenever the module’s code includes `<linux/module.h>`
  - This placed in the .modinfo ELF section.
    - ELF (Executable Linking and Format)
Version dependency

- When asked to load a module, `insmod` follows its own search path to look for the object file (/lib/module).

- In case of version mismatch
  - Force to install (`insmod -f`).
  - This operation is not safe.
Version macros

- **UTS_RELEASE**
  - Kernel version : 2.3.48

- **LINUX_VERSION_CODE**
  - Kernel binary version : 2.3.48 → 131888

- **KERNEL_VERSION(major, minor, release)**
  - KERNEL_VERSION(2.3.48) = 131888

- They are defined in <linux/version.h>
Kernel symbol table

- It contains the addresses of global kernel items (**functions and variables**).
  - They are needed to implement modularized drivers.
  - `/proc/ksyms`

- When a module is loaded, symbol exported by the module becomes part of the kernel symbol table.
Symbol table macros

- EXPORT_NO_SYMBOLS
- EXPORT_SYMBOL(name)
- EXPORT_SYMBOL_NOVERS(name)
Error handling

- If any errors occur when you register utilities, you must undo any registration before the failure.
- Error recovery is sometimes best handled with the `goto` statement.
Error handling

```c
int init_module()
{
    int err;
    err = register_this(ptr1, "skull");
    if (err)
        goto fail_this;
    ...
    return 0;
    fail_this: return err;
}
```
Usage count

- The system keeps usage count for every module in order to determine whether the module can be safely removed.
  - /proc/modules
Usage count macros

- **MOD_INC_USE_COUNT**
  - Increment the count.

- **MOD_DEC_USE_COUNT**
  - Decrements the count.

- **MOD_IN_USE**
  - Evaluate to true if the count is not zero
Module configuration

- Parameter values can be assigned at load time by `insmod`.
  - `insmod skull.o ival=666 sval="hello"

- Supported types:
  - Integer : i
  - Long : l
  - String : s
  - Byte : b
  - Shot (two bytes) : h
Module configuration

```c
int num = 0;
MODULE_PARM (num, "i");

char *str;
MODULE_PARM (str, "s");

int array[4];
MODULE_PARM (array, "2-4i");
```
Question?