Introduction to Linux Kernel Development

Luis R. Rodriguez <mcgrof@gmail.com>

Doc last updated: June 17, 2008

Goal of presentation:

Bring people with no experience into the realm of Linux kernel development and its community. Towards the end you will be expected to be ready to submit a patch upstream for inclusion on the next release of the stock kernel.

ubuntu

History, license, politics and chain of command

- Early history – the GNU Project missing link
- Kernel: C
- Userspace: C, C++(xconfig, QT), perl, python
- Covered under GPL License v2
- Non GPL-compatible drivers cannot go upstream
- Support helpful vendors, ignore those who ignore us... or reverse engineer

Chain of command example:

Wireless driver maintainer → Happy Hacker, Jr.
Wireless networking maintainer → John W. Linville
Networking device driver maintainer → David Miller
Linux development kernel maintainer → Linus Torvalds

2.6-Development - Linus Torvalds) || 2.6-Stable - Andrew Morton

Why Linux? FOSS, new freedom and technology enlightenment

Free Software:

- Freedom (yes, a “freedom”) to go or remain public (GPL) or private (BSD)
- Stays within the community, forever
- Available to less fortunate, to the masses, “free as in free beer”
- Freedom to redistribute

Open Source:

- Potential: Work of the masses, not the work of a small army
- Public scrutiny: destroys childish and narcissistic “security by obscurity” philosophy, the community knows better now
- Superior: Yields enduring, long lasting, reliable technology

Conclusion: *obvious benefits to FOSS, people are just used to and keep choosing “proprietary software” despite the incredible benefits of FOSS.* ***Will it catch on to the masses ?*** *... or* ***will the masses catch on to it****? GNU/Xorg/GNOME/KDE/Linux,* ***the community operating system*** *needs public* ***commitment****, slowly private sector is* ***accepting*** *it.* ***Last few milestones....***

ubuntu

Linux development metrics - 2.6.21

- New release every 2 ½ months
  - 2.89 changes per hour
  - 8.2 million lines of code
- 2000 lines added every day
- 2800 lines modified every day
- 24x7!

Hello world Linux module
hello_world.c

```
#include <linux/module.h>
MODULE_AUTHOR("Luis R. Rodriguez");
MODULE_LICENSE("GPL");
static int hello_init(void)
{
        printk("I am a module, cheers!\n");
        return 0;
}
static void goodbye_exit(void)
{
        printk("Goodbye cruel world!\n");
}
module_init(hello_init);
module_exit(goodbye_exit);
```

Hello world Makefile

Command line:

make -C /lib/modules/`uname -r`/build/ M=`pwd`

Works with minimal Makefile:

obj-m += hello_world.o

Or

-C <directory> change directory
M=<directory> build external module

Decent module Makefile:

```
obj-m += hello_world.o
all:
	make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules
clean:
	make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) clean
clean-files := Module.symvers
```

ubuntu

Building hello_world

```
$ uname -r
2.6.22-14-generic
$ pwd
/home/mcgrof/devel/modules/hello_world
$ ls
hello_world.c   Makefile
$ make
```

```
make -C /lib/modules/2.6.22-14-generic/build/ M=/home/mcgrof/devel/modules/hello_world modules
make[1]: Entering directory `/usr/src/linux-headers-2.6.22-14-generic'
  CC [M]  /home/mcgrof/devel/modules/hello_world/hello_world.o
  Building modules, stage 2.
  MODPOST
  CC      /home/mcgrof/devel/modules/hello_world/hello_world.mod.o
  LD [M]  /home/mcgrof/devel/modules/hello_world/hello_world.ko
make[1]: Leaving directory `/usr/src/linux-headers-2.6.22-14-generic'
```

ubuntu

dmesg, insmod, rmmod

$ sudo dmesg -c > /dev/null klogctl() --> sys_syslog()

$ sudo insmod hello_world.ko sys_init_module()

$ sudo dmesg -c sys_syslog()

I am a module, cheers!

$ sudo rmmod hello_world sys_delete_module()

$ sudo dmesg -c sys_syslog()

Goodbye cruel world!

Hello world -Huh???

What the heck just happened?

ubuntu

Linux Kernel build process

The kernel **build** process has five parts:

- Makefile
- .config (Kconfigs read in for options)
- arch/$(ARCH)/Makefile
- scripts/Makefile.*
- Kbuild Makefiles

Knowledge requirements

Users:

- Mom & Dad: GUI interfaces to upgrade, dpkg, rpm
- College kids: make menuconfig, make, make install, update-grub

Normal Developers:

- Kconfig, Makefiles, source code

Architecture Developers:

- Kconfig, Makefiles, source code, assembly, Kbuild hacks

Kbuild Developers:

- Kconfig, Makefiles, Kbuild hacks, gcc compatibility

Linux Kernel Top Makefile

Top Makefile:

The top Makefile is responsible for producing vmlinux and modules. This file is updated **on every kernel release** but usually only for versioning:

```
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 22
EXTRAVERSION = .9
NAME = Holy Dancing Manatees, Batman!
```

Kernel Configuration

The .config file: (use menuconfig)

config - Update current config utilising a line-oriented program
menuconfig - Update current config utilising a menu based program
xconfig - Update current config utilising a QT based front-end
gconfig - Update current config utilising a GTK based front-end
oldconfig - Update current config utilising a provided .config as base
randconfig - New config with random answer to all options
defconfig - New config with default answer to all options
allmodconfig - New config selecting modules when possible
allyesconfig - New config where all options are accepted with yes
allnoconfig - New config where all options are answered with no

.config and Kconfig

drivers/net/wireless/Kconfig

```
config IPW2200
    tristate "Intel PRO/Wireless 2200BG and 2915ABG Network Connection"
    depends on NET_RADIO && PCI
    select FW_LOADER
    select IEEE80211
    ---help---
        A driver for the Intel PRO/Wireless 2200BG and 2915ABG
        Network Connection adapters.

        See <file:Documentation/networking/README.ipw2200> for
        information on the capabilities currently enabled in this
        driver and for tips for debugging issues and problems.
                                    -- etc --
```

Yields

<M> Intel PRO/Wireless 2200BG and 2915ABG Network Connection

arch/$(ARCH)/Makefile

An arch Makefile cooperates with the top Makefile to define variables which specify how to build the vmlinux file. Note that there is no corresponding arch-specific section for modules; *the module-building machinery is all architecture-independent*.

head-y, init-y, core-y, libs-y, drivers-y, net-y

$(head-y) list objects to be linked first in vmlinux.
$(libs-y) list directories where a lib.a archive can be located.
The rest list directories where a built-in.o object file can be located.

$(init-y) objects will be located after $(head-y).
Then the rest follows in this order:
$(core-y), $(libs-y), $(drivers-y) and $(net-y).

scripts/Makefile.*, scripts/

scripts/ directory contains userspace and build time utilities

- bin2c – bin2c < binary --> yields header file to regenerate binary
- Lindent – calls indent with kernel specific parameters

scripts/kconfig/Makefile

make xconfig

scripts/Makefile
```
PHONY += oldconfig xconfig gconfig menuconfig config silentoldconfig update-po-config

xconfig: $(obj)/qconf
	$< arch/$(ARCH)/Kconfg
```

scripts/Makefile.*

scripts/Makefile.modpost
<module>.mod
<module>.mod.c
<module.ko>

Kbuild Makefiles

Where you will start off

drivers/net/wireless/Makefile

```
obj-$(CONFIG_IPW2200) += ipw2200.o
```

drivers/net/Makefile

```
obj-$(CONFIG_NET_RADIO) += wireless/
```

drivers/Makefile

```
obj-y += base/ block/ misc/ mfd/ net/ media/
```

Coding Style 101 Part I

Documentation/CodingStyle

Tab	8 characters
Columns	80 lines (ehh)

```
int fun(int a)
{
        int result = 0;
        char *buffer = kmalloc(SIZE);
        if (buffer == NULL)
                return -ENOMEM;
        if (condition1) {
                ...
        } else if (a > BIG) {
                while (loop1) {
                        ...
                }
                result = 1;
                goto out;
        }
        ...
out:
        kfree(buffer);
        return result;
}
```

- Functions should fit two ISO/ANSI screens (80x24)
- Don't use typdefs
- Wrong: ThisVariableIsATemporaryCounter, cntusr()
- OK: tmp, count_active_users()
- CAPITALIZE macros, but functional macros can be lower
- Wrong: // do not use this type of commenting
- OK: /* These are more welcomed */

ubuntu

Coding Style 101
Part II

Macros with multiple statements
should be enclosed in a do - while block:

```
#define macrofun(a, b, c)              \
        do {                           \
                if (a == 5)             \
                        do_this(b, c);  \
        } while (0)
```

Trivia: Why?

“First off, I'd suggest printing out a copy of the GNU coding standards, and NOT read it. Burn them, it's a great symbolic gesture.”

“some people will claim that having 8-character indentations makes the code move too far to the right, and makes it hard to read on a 80-character terminal screen. The answer to that is that if you need more than 3 levels of indentation, you're screwed anyway, and should fix your program.”

“you want your comments to tell WHAT your code does, not HOW.”

Understanding diff
Requirement, no buts!

```
$ cp \
    linux-2.6.22/drivers/net/wireless/Kconfig
    linux-2.6.22/drivers/net/wireless/Kconfig.orig
$ vim linux-2.6.22/drivers/net/wireless/Kconfig
$ diff -u \
    linux-2.6.22/drivers/net/wireless/Kconfig.orig \
    linux-2.6.22/drivers/net/wireless/Kconfig > wireless-kconfig.diff
$ cat wireless-kconfig.diff

--- drivers/net/wireless/Kconfig.orig   2006-09-07 11:57:08.000000000 -0400
+++ drivers/net/wireless/Kconfig        2006-09-07 11:59:02.000000000 -0400
@@ -2,7 +2,7 @@
 # Wireless LAN device configuration
 #

-menu "Wireless LAN (non-hamradio)"
+menu "Wireless LAN - IEEE-802.11"
        depends on NETDEVICES

 config NET_RADIO
@@ -203,6 +203,7 @@
        depends on NET_RADIO && PCI
        select FW_LOADER
        select IEEE80211
+       select CRYPTO
        ---help---
          A driver for the Intel PRO/Wireless 2200BG and 2915ABG Network
          Connection adapters.
```

ubuntu

No libc – The Kernel API

The kernel *does not use* ***libc*** and it shouldn't. The kernel implements its own library.

- lib-y objects
- part of final vmlinux
- Written in C and assembly

lib/Makefile lib/string.c

lib/Makefile

```
#
# Makefile for some libs needed in the kernel.
#

lib-y := errno.o ctype.o string.o vsprintf.o cmdline.o \
         bust_spinlocks.o rbtree.o radix-tree.o dump_stack.o \
         idr.o div64.o int_sqrt.o bitmap.o extable.o prio_tree.o \
         sha1.o
```

lib/string.c --- Triva: what happens to count on the caller?

```
#ifndef __HAVE_ARCH_MEMSET
/**
 * memset - Fill a region of memory with the given value
 * @s: Pointer to the start of the area.
 * @c: The byte to fill the area with
 * @count: The size of the area.
 *
 * Do not use memset() to access IO space, use memset_io() instead.
 */
void *memset(void *s, int c, size_t count)
{
        char *xs = s;

        while (count--)
                *xs++ = c;
        return s;
}
EXPORT_SYMBOL(memset);
#endif
```

ubuntu

GCC extensions and Kernel API hacks

GNU C provides several language features not found in ANSI standard C. (The `-pedantic' option directs GNU CC to print a warning message if any of these features are used)

`packed'
The `packed' attribute specifies that a variable or structure field should have the smallest possible alignment

`unused'
This attribute, attached to a function, means that the function is meant to be possibly unused. GNU CC will not produce a warning for this function.

```
#define offsetof(TYPE, MEMBER) ((size_t) &((TYPE *)0)->MEMBER)

#define container_of(ptr, type, member)({                       \
        const typeof( ((type *)0)->member ) *__mptr = (ptr); \
        (type *)( (char *)__mptr - offsetof(type,member) );})
```

ubuntu

offsetof() and __attribute__ ((packed))

```c
#include <stdio.h>

#define offsetof(TYPE, MEMBER) ((size_t) &((TYPE *)0)->MEMBER)

struct foo {
        int a; /* offset: 0 */
        char s[5]; /* offset: 4  */
        int b; /* What is my offset ? */
        int c;
} __attribute__ ((packed)) ;

int main() {
        int r;
        r = (size_t) & ((struct foo *) 0)->b;
//      r = offsetof(struct foo, b); /* The same */
        printf("offest: %d\n", r);
        return r;
}
```

What is b's offset if:

- s[4] was used? 8 (not packed)
- s[5] was used? 12 (not packed)
- s[5] with __attribute__ ((packed))? 9

ubuntu

Internal kernel data types

Portability is not only important in the linux kernel, its a requirement but...

linux/types.h

Bits	Kernel-unsigned	Kernel-signed	Userspace-unsigned	Userspace-signed
8	u8	s8	__u8	__s8
16	u16	s16	__u16	__s16
32	u32	s32	__u32	__s32
64	u64	s64	__u64	__s64

Endianness

```
Big endian:    most    significant ("biggest")   byte (also known as MSB) first
Little endian: least   significant ("littlest")  byte (also known as LSB) first
```

A processor's native format may be BE or LE, since Linux driver development needs to be CPU-agnostic, we provide interfaces for conversion to/from our CPU.

return_type	CPU-->LE	LE-->CPU	CPU-->BE	BE-->CPU
u16	cpu_to_le16(u16)	le16_to_cpu(u16)	cpu_to_be16(u16)	be16_to_cpu(u16)
u32	cpu_to_le32(u32)	le32_to_cpu(u32)	cpu_to_be32(u32)	be32_to_cpu(u32)
u64	cpu_to_le64(u64)	le64_to_cpu(u64)	cpu_to_be64(u64)	be64_to_cpu(u64)

kmalloc() and kfree()

What does mallloc() use?

kmalloc() is just like libc malloc() but requires a new flag passed -- we are now dealing with memory directly though so we need to get specific about our requirements in memory allocation. This is what the flag provides, an interface to specifying details of the requested memory.

```
struct *foo = kmalloc(sizeof(struct foo), GFP_KERNEL);
if (foo == NULL)
    return -ENOMEM;
```

GFP_ATOMIC means roughly "make the allocation operation atomic". This means that the kernel will try to find the memory using a pile of free memory set aside for urgent allocation. If that pile doesn't have enough free pages, the operation will fail. This flag is useful for allocation within interrupt handlers.

GFP_KERNEL will try a little harder to find memory. There's a possibility that the call to kmalloc() will sleep while the kernel is trying to find memory (thus making it unsuitable for interrupt handlers). It's much more rare for an allocation with GFP_KERNEL to fail than with GFP_ATOMIC.

GFP_BUFFER has the lowest priority, and doesn't try to free other pages if the requested memory isn't available.

In all cases, kmalloc() should only be used allocating small amounts of memory (a few kb). vmalloc() is better for larger amounts.

Linux kernel linked list implementation

include/linux/list.h

```
struct list_head {
        struct list_head *next, *prev;
};

#define LIST_HEAD_INIT(name) { &(name), &(name) }

#define LIST_HEAD(name) \
        struct list_head name = LIST_HEAD_INIT(name)

static inline void __list_add(struct list_head *new,
                              struct list_head *prev,
                              struct list_head *next)
{
        next->prev = new;
        new->next = next;
        new->prev = prev;
        prev->next = new;
}

static inline void list_add(struct list_head *new, struct list_head *head)
{
        __list_add(new, head, head->next);
}
```

Available linked lists:

- Kernel modules
- netdevices
- list of INET protocols
- list of packet handlers
- *etc...*

One of the most popular kernel data structures

Iterating over the lists

```
#define list_entry(ptr, type, member) \
        container_of(ptr, type, member)
/**
 * list_for_each_entry  -       iterate over list of given type
 * @pos:        the type * to use as a loop cursor.
 * @head:       the head for your list.
 * @member:     the name of the list_struct within the struct.
 */
#define list_for_each_entry(pos, head, member)                          \
        for (pos = list_entry((head)->next, typeof(*pos), member);      \
             prefetch(pos->member.next), &pos->member != (head);        \
             pos = list_entry(pos->member.next, typeof(*pos), member))
/**
 * list_for_each_entry_safe - iterate over list
 *   of given type safe against removal of list entry
 * @pos:        the type * to use as a loop cursor.
 * @n:          another type * to use as temporary storage
 * @head:       the head for your list.
 * @member:     the name of the list_struct within the struct.
 */
#define list_for_each_entry_safe(pos, n, head, member)                  \
        for (pos = list_entry((head)->next, typeof(*pos), member),      \
                n = list_entry(pos->member.next, typeof(*pos), member); \
             &pos->member != (head);                                    \
             pos = n, n = list_entry(n->member.next, typeof(*n), member))
```

Interrupts

Force a context switch to a new procedure or task. Used for hardware or software interrupts of the CPU and exceptions.

Interrupt types (origins):

- Synchronous – executing code
- Asynchronous – response to hardware or exception detected

Categories (classification):

- Maskable Interrupts
- Nonmaskable Interrupts

Standard interrupts

Intel 8259 – there since the beginning, part of IBM's PC/XT (1983, processor 8088) and PC/AT (1984, processor 80286). IRQ vectors (aka IRQ lines, IRQs) fit into an unsigned one-byte int (range 0-255). 0-31 unmaskable; 32-47 maskable, IRQ lines 1-15; 48-255 software interrupts.

```
0     divide error
1     debug exception
2     NMI interrupt
3     Breakpoint
4     INTO-detected Overflow
5     BOUND range exceeded
6     Invalid opcode
7     coprocessor not available
8     double fault
9     coprocessor segment overrun
10    invalid task state segment
11    segment not present
12    stack fault
13    general protection
14    page fault
15    reserved
16    coprocessor error
17-31      reserved
32-255     maskable interrupts

*128 (0x80) - Special Software Interrupt
```

Today we move away from Programmable Interrupt Controllers (PIC, a multiplexor) like the x86 popular Intel 8259 to APICs, introduced for SMP support, to Message Signaled Interrupts (MSI). MSI is part of PCI 2.2 and later PCI Express.

Hierarchical privilege levels

The idea is to use hardware to enforce protection and access to system resources. Introduced by the Multics operating system (UNIX pun father) as “hardware supported rings”. Multics introduced 8 rings, today's Unices use only 2 rings.

- Supervisor mode - Kernel space
- Protected mode – Userspace

Need a way to let userspace talk to kernel space...

Interrupt 0x80 – System Calls

We use interrupt 128 (0x80) for System Calls. This software interrupt, initialized at system startup (like system clock vector), is used to transfer control to the kernel. Userspace uses system calls as means to access protected system resources. The kernel is in charge of the protected resources, the kernel listens to userspace through system calls.

From wikipedia on IA-32:

“The 386 and all other IA-32 processors have eight 32-bit general purpose registers for application use. AMD64 processors double this to 16. There are 8 floating point stack registers (also doubled in AMD64 processors). Later processors added new registers with their various SIMD instruction sets too, such as MMX, 3DNow!, SSE, SSE2, SSE3, and SSSE3.”

General purpose registers

General data registers

All of the four following registers may be used as general purpose registers. However each has some specialized purpose as well. Each of these registers also have 16-bit or 8-bit subset names.

* EAX (At 000) Dedicated accumulator which is used for all major calculations.
* ECX (At 001) The universal loop counter which has a special interpretation for loops.
* EDX (At 010) The data register, which is an extension to the accumulator, stores data relevant to the operation applied to the accumulator.
* EBX (At 011) Currently used for free storage but was originally used as a pointer in 16-bit mode

General address registers

Used only for address pointing. They have 16-bit subset names, but no 8-bit subsets.

* ESP (At 100) Stack pointer. Is used to hold the top address of the stack.
* EBP (At 101) Base pointer. Is used to hold the address of the current stack frame. It is also sometimes used as free storage.
* ESI (At 110) Source index. Commonly used for string operations. It has a one-byte opcode for loading data from memory to the accumulator.
* EDI (At 111) Destination index. Commonly used for string operations. Has a one-byte STOS instruction to write data out of the accumulator.
* EIP Instruction pointer. Holds the current instruction address.

System call example

kernel/sys.c

```
asmlinkage long sys_newuname(struct new_utsname __user * name)
{
        int errno = 0;

        down_read(&uts_sem);
        if (copy_to_user(name,&system_utsname,sizeof *name))
                errno = -EFAULT;
        up_read(&uts_sem);
        return errno;
}
```

include/asm-i386/linkage.h

```
#define asmlinkage CPP_ASMLINKAGE __attribute__((regparm(0)))
```

include/asm-i386/unistd.h

```
#define __NR_uname              122
```

arch/i386/kernel/syscall_table.S

```
        .long sys_newuname
```

asmlinkage optimization: tells the compiler that the arguments of a function are on the CPU registers, and not on the stack.

Adding a system call

Add a system call to Linux:

- For each arch add entry to the end of system call table (entry.S)
- For each arch add entry number to asm/unistd.h
- Needs to be part of vmlinux – any part of kernel/ as kernel/sys.c