Compiling Linux Kernel
Tutorial

Content

- Environment
- Brief Linux History
- Linux Kernel Structure
- Configuring, Compiling and Installing Kernel
- Implementing System Call

Environment

- Not mandatory (but strongly suggested): downloand and install some sort of virtualization product:
  - VMware Server, VirtualBox
  - KVM, Xen
  - …
- Install a linux distribution
- Download and install a kernel source tree from ftp.kernel.org
- Suggestion: back up often

GNU: Open Source before Linux

- The concept of "free" software
  - Through 70s: Richard Stallman advocates "free" software
  - "free" as in freedom (not zero cost):
    - free to use, distribute (for a profit), and modify
- 1984: Richard Stallman founded GNU (Gnu's Not Unix)
  - Goal: to produce free software.
  - GPL: ensure software freedom by copyright terms
  - GNU software: Unix-like programs (no kernel)

Linux Right on the Start

- 1991: Finnish student Linus Torvalds started working on update of Minix
- From early start, Linus asked for volunteers on the Internet to help him develop Linux
  - People started using and publicizing Linux
  - A number of programmers joined the project
- From the start, the source code has been freely available on the Internet
  - GNU has lots of user-space programs but no kernel

Linux Kernel Evolution

- 0 (Apr 1991): First e-mail from Linus
  - 0.01 (Sept 1991)
  - 1.0 (March 94)
    - 1.2 (March 95)
  - 2.0 (June 96)
  - 2.2 (January 99)
  - 2.4 (January 2001)
  - 2.6 (May 2004)

Getting Started

- **Important Link!!!**
  - **http://lxr.linux.no/**

    Cross-Referencing Linux
  - **google://lxr**

Linux Kernel Code Structure

- ..../include : Header files (.h)
- ..../lib : Common functions
- ..../init : Kernel initialization code
- ..../kernel : Kernel core
- ..../mm : Memory management
- ..../ipc : Inter-process Communication
- ..../net : Networking code
- ..../scripts: Helping tools for building kernel
- ..../Documentation

Linux Kernel Code Structure

- ..../fs : Filesystems
  - ..../fs/ext2: Most popular Linux filesystem
  - ..../fs/msdos : MSDOS file system (C> drive)
  - ..../fs/vfat : MS Windows (VFAT)
  - ..../fs/proc : virtual file system (process info)
- ..../drivers : Device drivers
  - ..../drivers/block : hard drives
  - ..../drivers/scsi : SCSI device
  - ..../drivers/char : character-stream device
  - ..../drivers/net : Network cards

Linux Kernel Code Structure

- ..../arch : Platform-dependent code
  - ..../arch/i386 : Intel 386 (IBM PC architecture)
  - ..../arch/alpha : Compaq's Alpha architecture
  - ..../arch/sparc : Sun's SPARC architecture

Getting and Configuring the Kernel

- 1. Getting the kernel sources
  - ftp.kernel.org
- 2. Unpack it
  - tar xzvf
- 3. Configuring for a build
  - make oldconfig
  - make menuconfig
  - make xconfig
  - make defconfig

Compiling the Kernel

- 4. Builds the tree of interdependencies
  - make dep (not needed with 2.6)
- 5. Cleanup the source (optional)
  - make clean
- 6. Build the kernel
  - make bzImage
- 7. Build modules
  - make modules

Installing the Kernel

- 8. Copy bzImage to /boot/vmlinuz-X.Y.Z
- 9. Copy System.map to /boot and create a symbolic link from /boot/System.map
- 10. Edit /boot/grub/grub.conf file
  - title: XXX
  - root (hdX,Y)
  - kernel /vmlinuz-X.Y.Z ro root=<path> (e.g. /dev/hda1)
- 11. Install modules to /lib/modules
  - make modules_install

System Call

- Process in *User Mode* make a system call (a special function call) to obtain kernel service
  - E.g. getpid()
- Some takes one or more arguments (words)
- Returns an integer value
  - If failed: return -1 and set errno (see include/asm-i386/errno.h)
- In *User Space*, a system call is implemented as a wrapper function in libc.

System Call Wrapper Function

- Define system calls using macros: _syscall0 ... _syscall5 (see include/asm-i386/unistd.h)
- For example:
  - pid_t getpid()
    - _syscall0(int,getpid)
  - unsigned int alarm(unsigned int seconds)
    - _syscall1(unsigned int,alarm,unsigned int,seconds)
  - int write(int fd,const char * buf,unsigned int count)
    - _syscall3(int,write,int,fd,const char *,buf,unsigned int, count)

After the Macro Expansion

```
unsigned int alarm(unsigned int seconds)
{ long __res;
  __asm__ volatile ("int $0x80"
          : "=a" (__res)
          : "" (__NR_alarm), "b“ ((long)(seconds)));

      if ((unsigned long)(__res) >= (unsigned long)(-125)) {
          errno = -(__res);
          __res = -1;
       }
       return (unsigned int) (__res);
}
```

Return Value

- Long integer (in 32-bit architecture)
- (unsigned long)0
  - System call succeeded
- (unsigned long)(-125) ... (unsigned long)(-1)
  - System call failed
  - Set errno.

Transfer of Control

- By special programmed exception (int $0x80)
- Exception vector 128 defined as system call
- The system call handler in kernel
- system_call(): written in assembly code
- Dispatch to the corresponding system call service routine (e.g., sys_alarm()), according to the system call number (e.g, __NR_alarm)
- System call dispatch table

System Call Dispatch Table

See arch/i386/kernel/entry.S:

```
.data
ENTRY(sys_call_table)
    .long SYMBOL_NAME(sys_ni_syscall) /* 0 */
    .long SYMBOL_NAME(sys_exit)
    .long SYMBOL_NAME(sys_fork)
    .long SYMBOL_NAME(sys_read)
    .long SYMBOL_NAME(sys_write)
    .long SYMBOL_NAME(sys_open) /* 5 */
    .long SYMBOL_NAME(sys_close)
    ...
```

System Call Service Routine

Corresponding to each system call, e.g.:

```
asmlinkage unsigned long sys_alarm(unsigned int seconds)
{
    struct itimerval it_new, it_old;
    unsigned int oldalarm;

     it_new.it_interval.tv_sec = it_new.it_interval.tv_usec = 0;
     it_new.it_value.tv_sec = seconds;
     it_new.it_value.tv_usec = 0;
     do_setitimer(ITIMER_REAL, &it_new, &it_old);
     oldalarm = it_old.it_value.tv_sec;
     if (it_old.it_value.tv_usec)
              oldalarm++;
     return oldalarm;
}
```

Parameter Passing

- Through CPU Registers
  - Each parameter is a word (32bit)
  - Parameters limited by register number
- What about complex data (e.g., pointer to a buffer, such as in write() system call)
  - Parameter: memory address
  - Kernel will copy data from/to process' user-mode memory space
- Verifying the system call parameters

User Space Memory Access

- Two service routines
  - bytes_left = copy_from_user(void*to, const void *from, unsigned long n);
  - bytes_left = copy_to_user(void*to, const void *from, unsigned long n);

Steps for Adding New System Calls

- Add a wrapper function in user-space code
  - Include <linux/unistd.h>
  - Use _syscall0() ... _syscall5()
- Include a macro definition for NR_XYZ
  - In include/asm-i386/unistd.h
- Write the corresponding service routine in kernel source tree (e.g., sys_XYZ())
- Add an entry to system call dispatch table
  - In arch/i386/kernel/entry.S