

# CS 378 (Spring 2003)

## Linux Kernel Programming

**Yongguang Zhang**

**([ygz@cs.utexas.edu](mailto:ygz@cs.utexas.edu))**

# Questions

---

- Have everybody turned in their Assignments?
- Any problem with the Assignments?
- Any questions on Kernel Modules?
- Any other questions?

# Getting to Know Each Other

---

- Introduce Ourselves
  - Your name
  - Your school year (under, grad?)
  - Special interests in Linux? (Anything that you want us to know?)
- I welcome anyone to audit this course, to participant in the discussions (in & out of class), and to do the projects
  - You don't need to say so if you don't like people to know that you are auditing

# More UML Hints

---

- Increasing memory used by UML
  - `./linux ubd0=..... mem=128M`
- Modify files in the UML source code tree
  - All are read-only links
    - `linux/arch/i386/kernel/entry.S -> /projects/cs378.ygz/src/linux/arch/i386/kernel/entry.S`
  - You need to make your own copy in place of the link
    - `cd linux/arch/i386/kernel`
    - `mv entry.S entry.S.link`
    - `cp -p entry.S.link entry.S`

# How to Debug a Kernel?

---

- Add instrument print statements
  - NO: `printf()`
  - Use `printk()`
    - Include `include/linux/kernel.h`
    - Example: `printk(KERN_DEBUG "i = %u\n", i);`
    - Note: no “,” between `KERN_DEBUG` and the format string
    - See `include/linux/kernel.h`; for other `KERN_` values
    - `printk()` internally uses a 1K buffer and does not check for buffer overflow, so be careful!
- Debugging UML with `gdb`
  - More control, but takes up lots of space

# A Helpful Website

---

- <http://kernelnewbies.org/>
  - #kernelnewbies is an IRC network dedicated to the "newbie" kernel hacker. The audience mostly consists of people who are learning about the kernel, working on kernel projects or professional kernel hackers that want to help less seasoned kernel people.
  - (Hacking does NOT mean breaking into other people's system!)

# Enhance Our Website

---

- Set up a discussion web for this class
  - To replace the newsgroup or cross-post on it
  - Like slashdot type
- Set up an lxr website for 2.4.19-46um
  - Like <http://www.iglu.org.il/lxr>
  - Source code: [lxr.sourceforge.net](http://lxr.sourceforge.net)
- I am looking for help to do these
  - Will get community building credit

# This Lecture

---

- Kernel Source Code Structure
- Kernel/User Space Interactions
- Transfer of Control
  - System calls
- Transfer of Data
  - /proc file system

# Linux Kernel Code Structure

---

- Core
  - Initialization part
  - Memory management part
  - Inter-process communication part
  - Networking part
  - File Systems (Many many choices)
  - Device Drivers (Many many choices)
  - Platform-dependent Part (PC, Alpha, Sparc, ...)

# Kernel Source 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

# Kernel Source 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

# Kernel Source 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
  - `... ./arch/um` UML's virtual machine architecture
    - `....`/arch/um/fs
      - `... ./arch/um/fs/hostfs` UML's host file system
    - `....`/arch/um/drivers
      - `... ./arch/um/drivers` UML's ubd device

# Kernel and User Modes

---

- Modern CPU architecture:
  - Supervisor mode: execute any instructions
  - User mode: execute non-privileged instructions, in bounded address space
- Modern operating systems: kernel & user modes
  - Strict separation of responsibility
  - Strictly-defined interface between kernel and user space (e.g., the POSIX standard)

# Kernel Responsibility

---

- Managing machine's resources
  - Hardware (control, I/O)
  - Software (software abstraction of hardware, data structures)
- Examples:
  - Process (CPU)
  - Memory
  - File (storage abstraction)
  - Devices (hardware abstraction)

# User/Kernel Interactions

---

- Transfer of Control between User and Kernel Mode
  - System Calls
  - Interrupts (timer or device)
- Transfer of Data between User and Kernel Space
  - Memory copy
  - /proc/ file system

# System Call Implementation

---

- 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)
: "0" (__NR_alarm),"b" ((long)(seconds)));
do {
if ((unsigned long)(__res) >= (unsigned long)(-125)) {
errno = -(__res);
__res = -1;
}
return (unsigned int) (__res);
} while (0);
}
```

# Return Value

---

- Long integer (in 32-bit architecture)
  - (unsigned long)0 ... (unsigned long)(-126)
    - 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;
    /* eh.. We can't return 0 if we have an alarm pending.. */
    /* And we'd better return too much than too little anyway */
    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's user-mode memory space
- Verifying the system call parameters

# User Space Memory Access

---

- Service routines: `get_user()`, `put_user()`, `copy_from_user()`, `copy_to_user()`, ...
  - They are macros, see `include/asm-i386/uaccess.h`
  - They in turn call `__get_user_x()`, `__put_user_x()` and other assembly codes.
- Example
  - System call: `int stime(time_t *t);`
  - Purpose: set the system time to value pointed by `t`
  - Parameter: is a memory address

# Service Routine for stime()

---

```
asmlinkage long sys_stime(int * tptr)
{
    int value;

    if (!capable(CAP_SYS_TIME))
        return -EPERM;
    if (get_user(value, tptr))
        return -EFAULT;
    write_lock_irq(&xtime_lock);
    xtime.tv_sec = value;
    xtime.tv_usec = 0;
    time_adjst = 0; /* stop active adjtime() */
    time_status |= STA_UNSYNC;
    time_maxerror = NTP_PHASE_LIMIT;
    time_esterror = NTP_PHASE_LIMIT;
    write_unlock_irq(&xtime_lock);
    return 0;
}
```

# Parameter Checking

---

- Check if each parameter is valid
  - Otherwise return `-ERRNO`.
- Check address
  - Illegitimate address? (e.g., reversed to kernel only)
- Page fault and Fixup Code
  - (We will wait until we cover the memory access)

# 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`
  - Update total syscall number: `.rept NR_syscalls-222`

# Additional Steps for UML

---

- In `arch/um/kernel/sys_call_table.c`
  - Add it to the list of externs as
    - `extern syscall_handler_t sys_XYZ;`
  - Add it to the call table (`sys_call_table[]`) as
    - `[ __NR_XYZ ] = sys_XYZ,`
- If it is the last system call,
  - In `arch/um/include/sysdep-i386/syscalls.h`,
    - `#define LAST_SYSCALL __NR_XYZ`
  - In `arch/um/include/sysdep/syscalls.h`, add
    - `#define LAST_SYSCALL __NR_XYZ`

# System Calls vs /proc File System

---

- Both used to extend the user-kernel interface
- Limitation of system calls:
  - Not scalable: each new service needs a new system call
  - Not extensible: how to manage the system call numbers, confirmation to POSIX standard?
- /proc file system is considered a better way
  - Still, standardization is always a problem

# Summary

---

- System Calls
  - Textbook: LKP §3.3 or ULK §9
- /proc file system
  - Textbook: LKP Appendix C or LDD2 §4
  - Documentation and example:
    - `/projects/cs378.ygz/src/linux-2.4.19/Documentation/DocBook/procfs_example.c`

# Assignment 3

---

- Doing System Calls
  - Will post in class web site
- Next assignment (#4) will be on /proc file system

# Next Lecture

---

- Linux Memory Management
  - Memory Model: Demand Paged Virtual Memory
- Time to review related topics in CS372!
  - What is virtual memory? What is physical memory?
  - What is address translation?
  - What is primary memory? What is secondary memory?
  - What is paging? What is a page?
  - What is page replacement policy?
  - What is a “dirty” page?
  - What is page fault?