

CS 378 (Spring 2003)

Linux Kernel Programming

Yongguang Zhang

(ygz@cs.utexas.edu)

This Lecture

- Any Questions?
- /proc file system
- Memory Management

/proc File System

- Pseudo-filesystem as an interface to kernel data structures
 - Each file correspond to a data structure in kernel
 - Purpose of /proc filesystem: an extensible mechanism to support all types of data transfer in a common interface
 - Most files in /proc are read-only, but some allow kernel variables to be changed
- Top-level subdirectories
 - Numerical: for processes
 - Non-numerical

Numerical Subdirectory

- Kernel data structures for the running processes
 - One subdirectory per process
 - Named by process ID
- Example files:
 - cmdline the complete command line for the process.
 - cpu CPU usage by this process
 - cwd symbolic link to the \$CWD of the process.
 - environ contains the environment for the process.
 - exe symbolic link to the running binary.
 - fd/ a subdirectory of open files by the process.
 - ...

Non-Numerical Files/Directories

cpuinfo information about the CPUs

...

net/ many files/subdir for many parts of Linux networking

 net/arp

 net/dev

 net/route

...

sys/ sysctl files (dynamic configurable kernel parameters)

...

Too many of them!

Read/Write at User Space

- Just like a normal file
 - open, read, write, close
- Examples:
 - To read in shell:
 - `cat /proc/cpuinfo`
 - To write in shell:
 - `echo cool-name > /proc/sys/kernel/hostname`
- For configurable kernel parameters (`/proc/sys/`)
 - Also with `sysctl` command (read/write)
 - `/sbin/sysctl -w kernel/hostname= "cool-name"`

Creating New /proc File in Kernel

- Understand the struct `proc_dir_entry` data object
- Write callback functions
- Create struct `proc_dir_entry` data object

/proc File Data Object

- include/linux/proc_fs.h

```
struct proc_dir_entry {
    unsigned short low_ino;
    unsigned short namelen;
    const char *name;
    mode_t mode;
    nlink_t nlink;
    uid_t uid;
    gid_t gid;
    unsigned long size;
    struct inode_operations * proc_iops;
    struct file_operations * proc_fops;
    get_info_t *get_info;
    struct module *owner;
    struct proc_dir_entry *next, *parent, *subdir;
    void *data;
    read_proc_t *read_proc;
    write_proc_t *write_proc;
    atomic_t count;      /* use count */
    int deleted;        /* delete flag */
    kdev_t rdev;
};
```

Callbacks

- Function types (include/linux/proc_fs.h)
 - typedef int (read_proc_t)(char *page, char **start, off_t off, int count, int *eof, void *data);
 - typedef int (write_proc_t)(struct file *file, const char *buffer, unsigned long count, void *data);

- Example use:

```
struct proc_dir_entry *e = create_proc_entrance(...);
```

```
...
```

```
int my_read_proc(...) { ... }
```

```
E->read_proc = my_read_proc;
```

Communicate with User Space

- User space to read data from kernel
 - User Space: to make `read()` system call
 - Kernel: will call the corresponding `e.read_proc()`
 - `sys_read() ⇒ ... ⇒ e.read_proc()`
- User space to write data to kernel
 - User Space: to make `write()` system call
 - Kernel: will call the corresponding `e.write_proc()`
 - `sys_write() ⇒ ... ⇒ e.write_proc()`
- Q: how/where is the transfer of data/control really taken place?

Adding Entries to /proc File System

- To add “regular” file:
 - `struct proc_dir_entry *create_proc_entry(const char *name, mode_t mode, struct proc_dir_entry *parent);`
- To make symbolic link:
 - `struct proc_dir_entry *proc_symlink(const char *, struct proc_dir_entry *, const char *);`
- To create **sysctl** file
- To make subdirectory

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

Lectures From Now On

- How to deal with Linux kernel in general
 - Basics data structures, utility routines
 - Learning and problem solving skills
- Topics about different parts of the kernel
 - I will go very fast (high level)
 - Details you can learn by doing the projects

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?

Where to Look?

- Source code
 - ./mm: architecture-independent linux memory model
 - ./arch/xxx/mm : architecture dependent
 - ./include/asm-xxx/:
 - page.h, pg*.h, mmu*.h, ...
- Textbook
 - LKP: §4
 - ULK: §2, §7

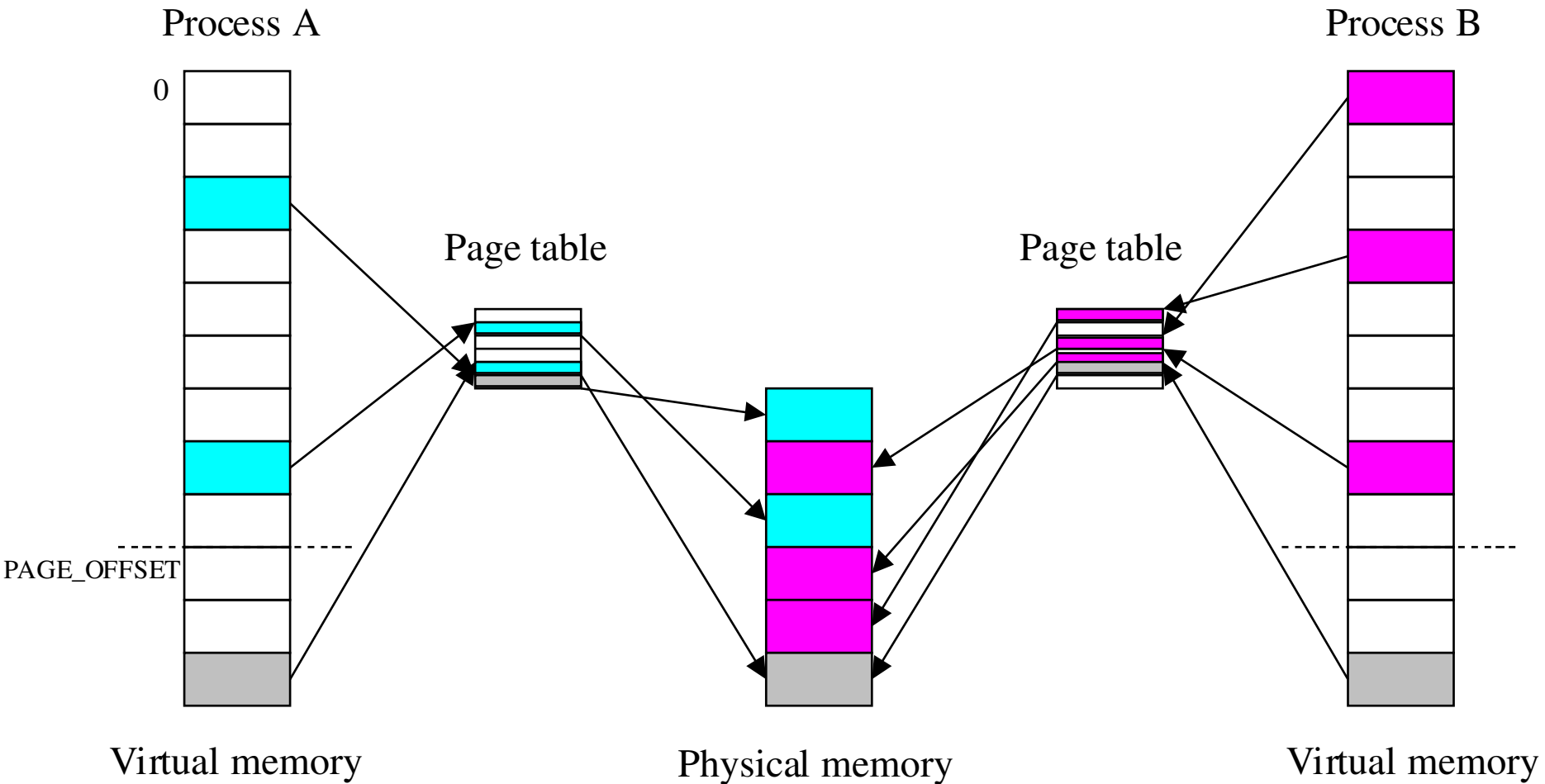
Memory Model

- Linear Address Space
 - Each process has its own memory address space
 - Mapping to Physical Memory by kernel MM
 - X86 architecture: 32bit (4GB)
- Actual Physical Memory
 - Can range from KB to TB
 - Larger than 4GB in i386 architecture? Need PAE
- Linux Memory Model
 - Architecture-independent (mm/)
 - Must be mapped to architecture specific (arch/xxx/mm)

Process Memory Layout

- User Space
 - Code segment, data segment (include/asm/page.h)
 - Addressable: 0 to PAGE_OFFSET-1
- Kernel Space
 - Code segment, data segment
 - Shared by all processes
 - Addressable: PAGE_OFFSET-1 to (unsigned)(-1)
- In i386 architecture
 - PAGE_OFFSET=0xc0000000
 - This means: 1GB reserved for kernel, 3GB for user

A Very Abstract Model



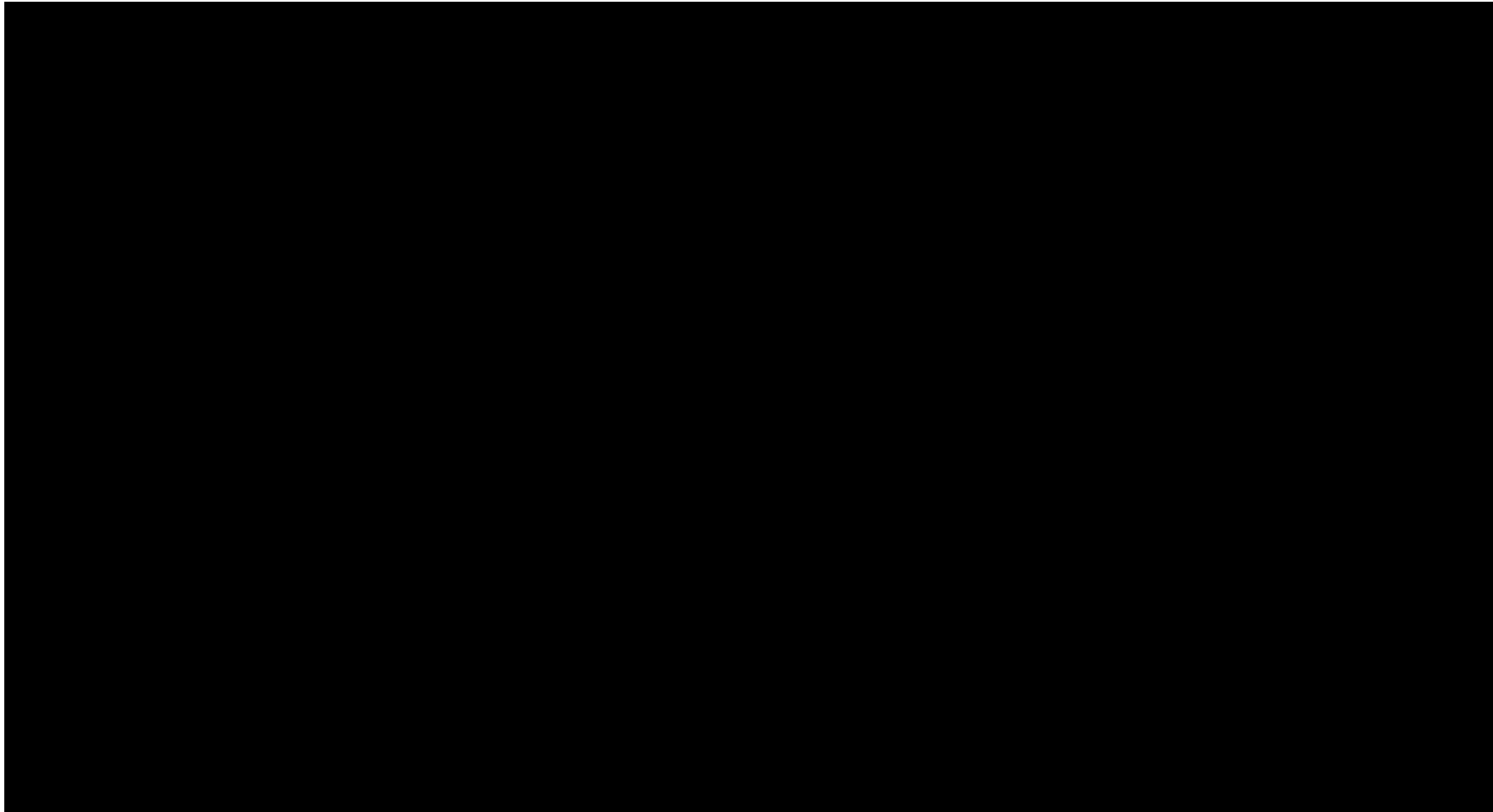
Architecture Specific

- Page: Basic unit of memory management
 - `PAGE_SIZE` (include/asm/page.h)
 - i386 architecture: page size is 4KB (2^{12})
 - Alpha architecture: page size is 8KB (2^{13})
- Addressable memory space
 - 32-bit iX86 architectures: 4GB (2^{32})
 - 64-bit Alpha architecture: 8TB (2^{43})
- Memory management highly related to hardware
 - Lots of routines implemented in assembly code

Page Translation Tables

- Linear virtual address
 - Low bits: offset in the page (e.g., 12 bits in 32b arch)
 - High bits: to identify the page unit (e.g., 20 bits)
- Page translation
 - Mapping the high bits to the base address of the physical page
 - OS kernel prepare the tables (pointed by CR3)
 - Actual translation done by hardware
- Multi-level page tables
 - Single-level is not efficient for large address space

3-Level Page Table Illustration



3-Level Page Table

- 3-level of indirect table access to address a page
 - Page Global Directory -> Page Middle Directory -> Page Table -> Page
 - Look for `pgd_t`, `pmd_t`, `pte_t`
 - In `include/asm-xxx/`: `page.h`, `pgalloc.h`, `pgtable.h`
- x86 memory address:
 - 10-bit (table), 0-bit (table), 10-bit (table), 12-bit (page)
 - `include/asm-i386/pgtable-2level.h`
- Alpha memory address:
 - 10-bit, 10-bit, 10-bit, 13-bit (page)

Page Table Entries

- Page Table `pte_t`
 - Each entry: 32-bit integer
 - Fields: the high bits, flags
 - Flags: present, accessed, dirty, R/W, user, ...
 - Look for `_PAGE_*` macros
- Functions
 - Look for `pte_*`() macros
- Page Directories Functions
 - Look for `pgd_*`() and `pmd_*`() macros

Virutal Memory

- Linux “Virtual” Memory Management
 - Mapping, allocating, and managing physical pages
 - Managing secondary memory: swapping
- Linux terminology
 - Swapping = demand-paging
- Architecture independent
 - A clean interface to support different memory mapping used under different architecture
 - To start: `include/linux/mm.h`, `mm/`

Using Memory in Kernel

- Kernel space is permanent mapped
 - No virtual or secondary memory
- Be careful when you write code in kernel space
 - Don't use too much memory
 - Never use recursive call (kernel stack is limited too)
- Routines
 - `kmalloc()`, `kfree()`
 - `vmalloc()`, `vfree()`
 - Macros to allocate pages

Summary

- /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
- Memory Management: LKP §4 or ULK §2
- Assignment 4: writing /proc file system module
 - Due next Friday (02/07)
 - Handout post in class web site
 - Use kernel modules!