CSE 451: Operating Systems

Section 2
Shells and System Calls

OSDI 2010
Finding a Needle in a Haystack: Facebook’s Photo Storage

Results: storage costs reduced 28%; 4x greater read throughput

Project 0
How is it going?
Project 1

ён Two main parts:
* Write a simple shell in C
* Add a simple system call to Linux kernel

You’ll learn:
* How system calls work in Linux
* How to work inside the Linux kernel

Due: Monday Oct 18, 11:59pm
* Electronic turnin: code + writeup

Project 1

E-mail groups to Abdul & Pete by Monday October 11
* After that, assigned to random groups

One turnin per group
* Use CVS for version control

System calls

Mechanism for applications to request service from the OS
* How do library calls differ from system calls?
System calls vs. library calls

Good example: `exec` family of calls
- Library calls: `execl, execlp, execle, execv, execvp`
- All map to one system call: `execve`

Useful tools
- `ltrace`
- `strace`

Project 1: adding a system call

- Add `execcounts` system call to Linux
  - Purpose: count the number of times that some other system calls are called
- Steps:
  - Modify kernel to track these counts
  - Implement `execcounts` function in kernel
  - Add it to the system call table
  - Call it from your shell, print results

Shells
**Shells**

*Primary responsibilities:*
- Parse user commands
- Execute commands / programs
- Manage input and output streams
- Job control

*Examples:*
- UNIX: bash, csh, ...
- Windows: Command Prompt, PowerShell

---

**The UNIX shell**

*Internal (built-in) commands*
- Execute routines embedded in the shell
- Manage state of the shell (e.g., current working directory, environment variables)

*External programs*
- How can you tell external from internal?

---

**Other UNIX shell capabilities**

*Redirect standard input / output / error streams*

```bash
# ./parser < logfile > outfile 2> errfile
```

*Command pipelines*

```bash
# ps -ef | grep java | awk '{print $2}'
```

*Background execution of process*

```bash
# time make > make.out &
# jobs
1+ Running time make > make.out &
```

---

**The CSE451 shell**

CSE451Shell% /bin/date
Fri Jan 16 00:05:39 PST 2004
CSE451Shell% pwd
/root
CSE451Shell% cd /
CSE451Shell% pwd
/
CSE451Shell% exit
The CSE451 shell

* Repeat these steps:
  * Print out prompt
  * Read and parse input
  * If built-in command:
    * Do it directly
  * Else (external program):
    * Launch specified program in new process
    * Wait for it to finish

Shell system calls

* `fork`
  * Create a child process

* `execve`
  * Execute a specified program

* `wait`
  * Wait until child process terminates

Project 1: final product

```
CSE451Shell% execcounts clear
CSE451Shell% cd /
CSE451Shell% pwd
/ 
CSE451Shell% date
Wed Sep 29 16:52:41 PDT 2004
CSE451Shell% time
CSE451Shell% execcounts
Statistics:
Fork: 3 27%
VFork: 0 0%
Clone: 0 0%
Exec: 8 72%
CSE451Shell% exit
```

Project 1: shell hints

* Useful library functions (see man pages):
  * Strings: `strcmp`, `strncpy`, `strtok`, `atoi`
  * I/O: `fgets`
  * Error reporting: `perror`
  * Environment variables: `getenv`
Programming in kernel mode

* Your shell will operate in user mode
* Your system call code will be in the Linux kernel, which operates in kernel mode
* Be careful - different programming rules, conventions, etc.

Programming in kernel mode

* Can’t use application libraries (i.e. libc); can only use functions defined by kernel
  * Can’t use `printf`, for example: use `printk` instead
* Different set of header files
* Can’t trust user space
  * Unsafe to directly access a pointer passed from user space
  * More details later

Linux development hints

* No man pages for Linux kernel functions!
  * Documentation/ directory in kernel source code
  * “The code is the documentation”

Navigating Linux code

* Invaluable tools:
  * `ctags`
  * `cscope`
* Linux Cross Reference (LXR)
  * [http://lxr.linux.no/linux+v2.6.13/](http://lxr.linux.no/linux+v2.6.13/)
Computing resources

☆ Develop your code on forkbomb
☆ Test your code on VMware PCs in 006

☆ Do not use attu

VMware

☆ Software simulation of x86 architecture
☆ Run an OS in a sandbox
☆ Easily reset to known good state

Using VMware

☆ Go through project setup first

Using VMware

Power on / off, reset

VMWare config: don't change!
**Using VMware**

* All disks are non-persistent
  * Powering off loses your changes!
  * To reboot:
    * `shutdown -r now`

**Using VMware**

* There is only one user: root (password: rootpassword)

* You will need to:
  * Build a Linux kernel image on forkbomb
  * Run VMware Player on Windows or Linux desktop
  * Transfer kernel image to VM (use scp)
  * Boot your kernel in VM

**Legends of computer science**

* Andrew Tanenbaum: in class tomorrow!

* Come early to get a seat