CSE 451 Section 2:
Processes, the shell, and system calls

Homework 1 highlights
- (1.7.) Availability is typically ensured through replication
- (1.11.) Memory bus contention can occur
- (2.14.) User: Your arguments + VMs protect from compromised software
- (2.15.) Asynchronous communication: harder to program with, but does not interrupt.

Project 1
- Teaches you:
  - how to handle processes
  - how to build & run Linux in VMware
- Two main parts:
  - Write a simple shell in C
  - Add a simple system call to Linux kernel
- Due: Wed., Oct 15, 11:59pm
  - Electronic turnin: code + writeup

Project groups
- Please form groups of 3
  - Send me emails with groups by Tues., Oct. 7
  - After that, I will assign you to random groups
  - You can change groups for next projects
- Only one person / group submits
- You can use CVS
  - Instructions of how to set it up on forkbomb and use it are on website

The shell
- What is it?

"A program that works with the operating system as a command processor, used to enter commands and initiate their execution."

--- American Heritage ® Dictionary of the English Language

Examples of shells:
- UNIX: bash, csh, ...
- Windows: command prompt
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)
  - Examples?
- External commands
  - Examples?
- How can you tell external from internal?

Other UNIX shell capabilities

- Redirect stdin / stdout / stderr
  
  ```
  # ./my_parser < logfile > outfile 2> errfile
  ```

- Background execution of process
  
  ```
  # ./my_parser < logfile > outfile 2> errfile &
  ```

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

The CSE451 shell

- Print out prompt
- Accept input
- Parse input
- If built-in command
  - do it directly
- Else spawn new process
  - Launch specified program
  - Wait for it to finish
- Repeat

CSE451 Shell Hints

- In your shell:
  - Use `fork` to create a child process
  - Use `execvp` to execute a specified program
  - Use `wait` to wait until child process terminates
- Useful library functions (see man pages):
  - Strings: `strcmp`, `strncpy`, `strtok`, `atoi`
  - I/O: `fgets`
  - Error report: `perror`
  - Environment variables: `getenv`

System Calls

- What’s a system call?
- Examples?
- How do system calls compare to library calls?

System calls & library calls

- System call
  - Using some OS service
  - Process/Signal/File/Network/IPC/…
- Library call
  - Not using any OS service
  - Provide a high level interface for OS service
- What happens when we call
  - `strncpy(3)`?
  - `fgets(3)`?
Project 1: Adding a System Call

- Add `execcounts` system call to Linux:
  - Purpose: collect statistics
  - Count number of times you call `fork`, `vfork`, `clone`, and `exec` system calls.

- Steps:
  - Modify kernel to keep track of this information
  - Add `execcounts` to return the counts to the user
  - Use `execcounts` in your shell to get this data from kernel and print it out.

Example of `execcounts`

```
CSE451Shell% execcounts clear
CSE451Shell% execcounts
Statistics:
Fork:                   3      27%
Clone:                  0      0%
VFork:                  0      0%
Exec:                   8      72%
```

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 (e.g. libc)
  - E.g. can’t use printf
- Use only functions defined by the kernel
  - E.g. use printk instead
- Include files are different in the kernel
- Don’t forget you’re in kernel space
  - *You cannot trust user space*
  - E.g. unsafe to access a pointer from user space directly

Kernel development hints

- Best way to learn: read existing code
- Use `grep -r search_string *`
- Use LXR (Linux Cross Reference): `http://lxr.linux.no/`

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
- All disks are nonpersistent
  - Powering off loses your changes! Use “shutdown -r now” instead

Linux && VMware
- There is only one user: root (password: rootpassword)
- You will need to:
  - Build a kernel image on forkbomb
  - Transfer it to Linux running inside VMware
    (you can use scp from the hosting OS)
  - Boot your new Linux kernel in VMware
- Instructions at: