

Alex Moshchuk anm@cs Office hours: Tue 1-2, Thu 4:30-5:30 Allen 218 (or lab)

1

Reminders

- n Sign up for mailing list
- n Read the web site
 - " Work through <u>lab information</u>
- _n Start reading the book
- Do the first homework due tomorrow!
- n Check spinlock/coredump access
- n Read & start project 1

2



451 Projects

- _n 4 projects
- First one individual, others groups of 3
- n Need basic C and UNIX skills
- n Check links if you need help with this
- _n Challenging
 - n Don't leave until last minute
- n Learn a lot of cool stuff

First Project

- n Introduces C and Unix skills you'll need
- Teaches how to build and run Linux in VMWare
- _n Two main parts:
 - " Write a simple shell in C
 - Add a simple system call to Linux kernel
- n Due: Friday, Jan 14, before lecture (2:30)
 - Electronic turnin: code + writeup

4



The shell

- _n Print out prompt
- _n Accept input
- _n Parse input
- $_{\rm n}\,$ If built-in command
 - n do it directly
- _n Else create new process
 - Launch specified program there
 - Wait for it to finish
- n Repeat

CSE451Shell% /bin/date Fri Jan 16 00:05:39 PST 2004 CSE451Shell% pwd

/root CSE451Shell% cd / CSE451Shell% pwd

CSE451Shell% exit

-

System Calls

- $_{\scriptscriptstyle \rm n}$ What's a system call?
- _n Examples?
- _n In your shell:
 - Use fork to create a child process
 - Use execvp to execute a specified program
 - _n Use *wait* to wait until child process terminates

6

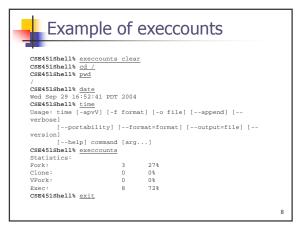
5



Project 1: Adding a System Call

- Add execcounts system call to Linux:
 - _n Purpose: collect statistics
 - ¹² Count number of times you call *fork, vfork, clone,* and *exec* system calls.
- Steps:
 - Modify kernel to keep track of this informationWe give you the kernel code
 - n Add *execcounts* to return the counts to the user
 - Use execcounts in your shell to get this data from kernel and print it out.

7





Programming in kernel mode

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

9



Programming in kernel mode

- ⁿ Can't use application libraries (e.g. libc)
 - _n E.g. can't use printf
- _n Use only functions defined by the kernel
 - _n E.g. use printk instead
- n Include files are different in the kernel
- Don't forget you're in kernel space
 - E.g. unsafe to access a pointer from user space directly, use fn's that perform checks
- Best way to learn look at existing code

10



Computing Resources

- Develop your code on dedicated 451 Linux hosts:
 - " spinlock, coredump
- n Test your code on VMWare PCs in 006
- _n Do not use attu



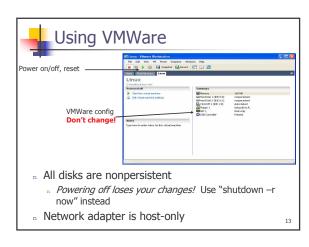
VMWare

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

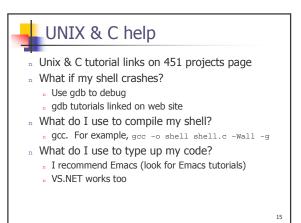


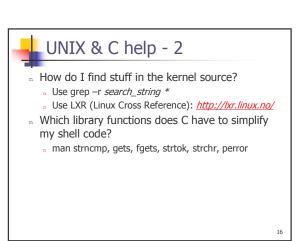
12

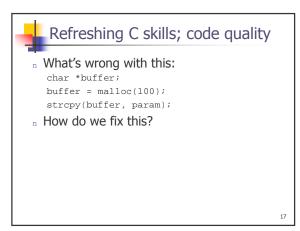
11











```
#define MYDEBUG // comment out to disable debugging

#ifdef MYDEBUG
#define DEBUG(x) x
#else
#define DEBUG(x)
#endif

...
int main() {
    ...
    printf("normal output");
    DEBUG(printf("debug output"));
    ...
}
```

3

More debugging

```
n Just for printing:
#ifdef MYDEBUG
# ifdef __KERNEL__
/* This one if debugging is on, and kernel space */
# define PDEBUG(fmt, args...) printk(*myprg: " fmt, ## args)
# else
/* This one for user space */
# define PDEBUG(fmt, args...) fprintf(stderr, fmt, ## args)
# endif
# else
          # else
# define PDEBUG(fmt, args...) /* not debugging: nothing */
#endif
```

n works for both for kernel and userspace

n To use:

PDEBUG("Testing two numbers: %d and %d\n", num, num2);