PURDUE UNIVERSITY®

CS 25200: Systems Programming

Lecture 11: File System Wrap-up, System Calls and Shell Introduction

Prof. Turkstra

Lecture 11

- File system wrap-up
- System calls
- Shell introduction

Directories

- Are files that simply contain a (file name, inode number) mapping
- inodes may appear in multiple directories
- Reference count tracks the number of directories in which it appears
- When refcnt == 0, the file is removed
  - Almost. Also cannot have open file descriptors

Links

- Hard
  - Files only (no cycles)
  - Directory entry + duplicate inode
  - $ ln target linkname
- Soft (Symbolic, symlink)
  - File itself (has an inode)
  - Holds a path
  - Cycles permitted
  - Can cross file systems
  - $ ln -s target linkname

Extended attributes

- Extension to the normal attributes associated with every inode in the system
- name:value pairs associated with files
- Eg, setfacl, getfacl
- -rwxr-xr-x+
- setfacl -x to remove
- getfattr

Examples

- setfacl -m u:apache:r /some/path
  getfacl /some/path
  ls -l

Discretionary Access Control

- User dictated
- Eg, classic file permissions
- POSIX Access Control Lists (ACLs)

Mandatory Access Control

- ...or MAC.
- Policy-based access control

SELinux

- Security-Enhanced Linux
- Implements MAC
- Set of kernel modifications and userland tools
  - Originally from the NSA
- Added to mainline kernel as of 2.6
- Originally included in RedHat
  - CentOS and Scientific Linux
  - Fedora *by default*
- Now Debian, Ubuntu, openSUSE, etc optionally

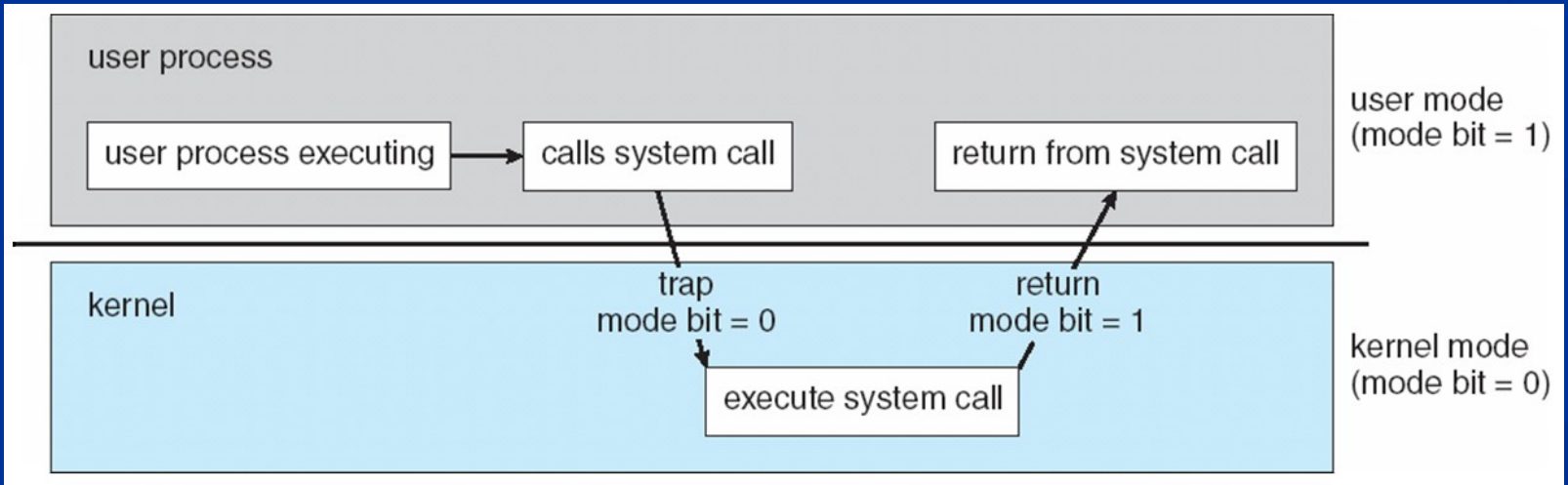
How?

- ls -Z
- chcon
- restorecon
- Etc

  chcon -R -t httpd_user_content_t
  setsebool -P httpd_can_network_connect on
  setsebool -P httpd_can_sendmail on

System calls

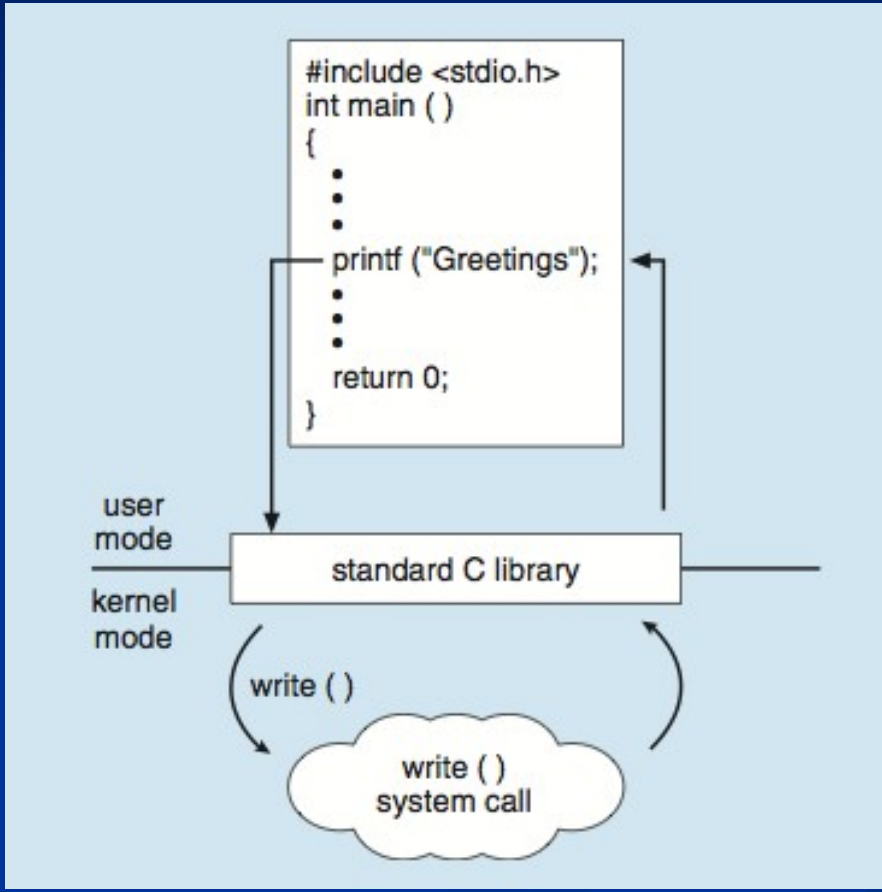
- System calls are the interface between processes and the OS kernel



System call types

- Process management
  - Create, terminate, execute, wait, etc
- File management
  - Create file, delete, open, close, read write, getattr, setattr, etc
- Device management
  - ioctl, read, write, etc
- Information management
  - getpid, alarm, sleep, etc
- Communication – between processes
  - pipe, shmget, mmap, etc

Standard C library



Why not function calls?

- System calls deal with privileged operations
  - Need privileged instructions
  - Access to devices and kernel data structures
  - Mechanism by which the kernel's security policy is enforced

Libraries and wrappers

- Most code does not invoke system calls directly
- Done through libraries and wrapper functions
  - E.g., printf() and malloc()
- Try to do as much work in user mode as possible
  - Context switches are expensive

Purdue "All-American" Marching Band

- Will be performing tomorrow starting at 6:00 PM, Slayter Center

UNIX Organization

- *nix has multiple components…
  - Scheduler – decides when and for how long a process should run
  - File system – provides a persistent storage mechanism
  - Virtual memory – address space isolation
  - Networking
  - Windowing system
  - Shells and applications

Shells

- A shell is basically a command interpreter. It provides an interface between the user and the computer (operating system).
- Shells may be graphical (explorer.exe, for instance) or text-based - often times called a CLI or command line interface

Shells cont...

- When we write a shell script, the first line of the file tells the operating system which shell to use.
- Some common *NIX shells include:
  - `#! /bin/sh` Bourne shell
  - `#! /bin/csh` C-Shell
  - `#! /bin/ksh` KornShell (more powerful)
  - `#! /bin/bash` Bourne-Again SHell

Shells

- A shell is a program. We interact with an instance of it – a process.
- Just like malloc() or anything other software, we can write our own
- Basic shells are made up of a parser and an executor
  - Most shells have other subsystems as well

Parser

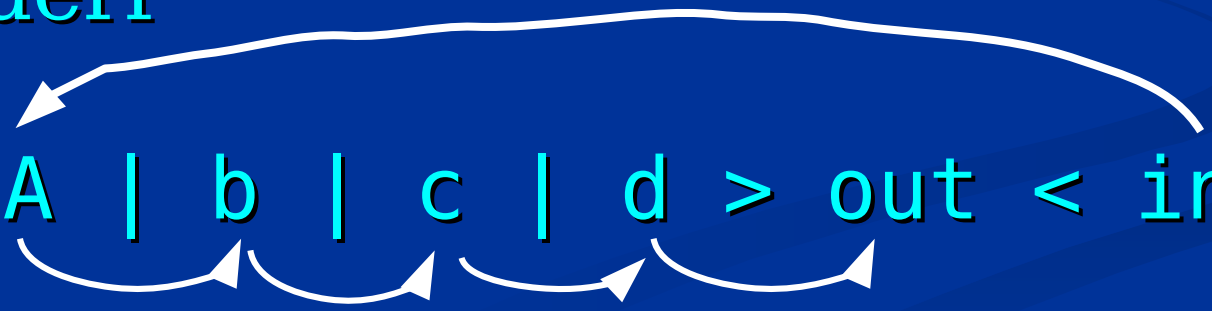
- The process of reading a string of symbols conforming to a grammar
- For our shell, it results in a command table:

```
$ ls -al | grep me > file1
```

ls	-al	
grep	me	
In:dflt	Out:file1	Err:dflt

Executor

- Creates processes for the command table entries
- Juggles file descriptors to create pipes between process inputs and outputs
- Also redirects stdin, stdout, and stderr



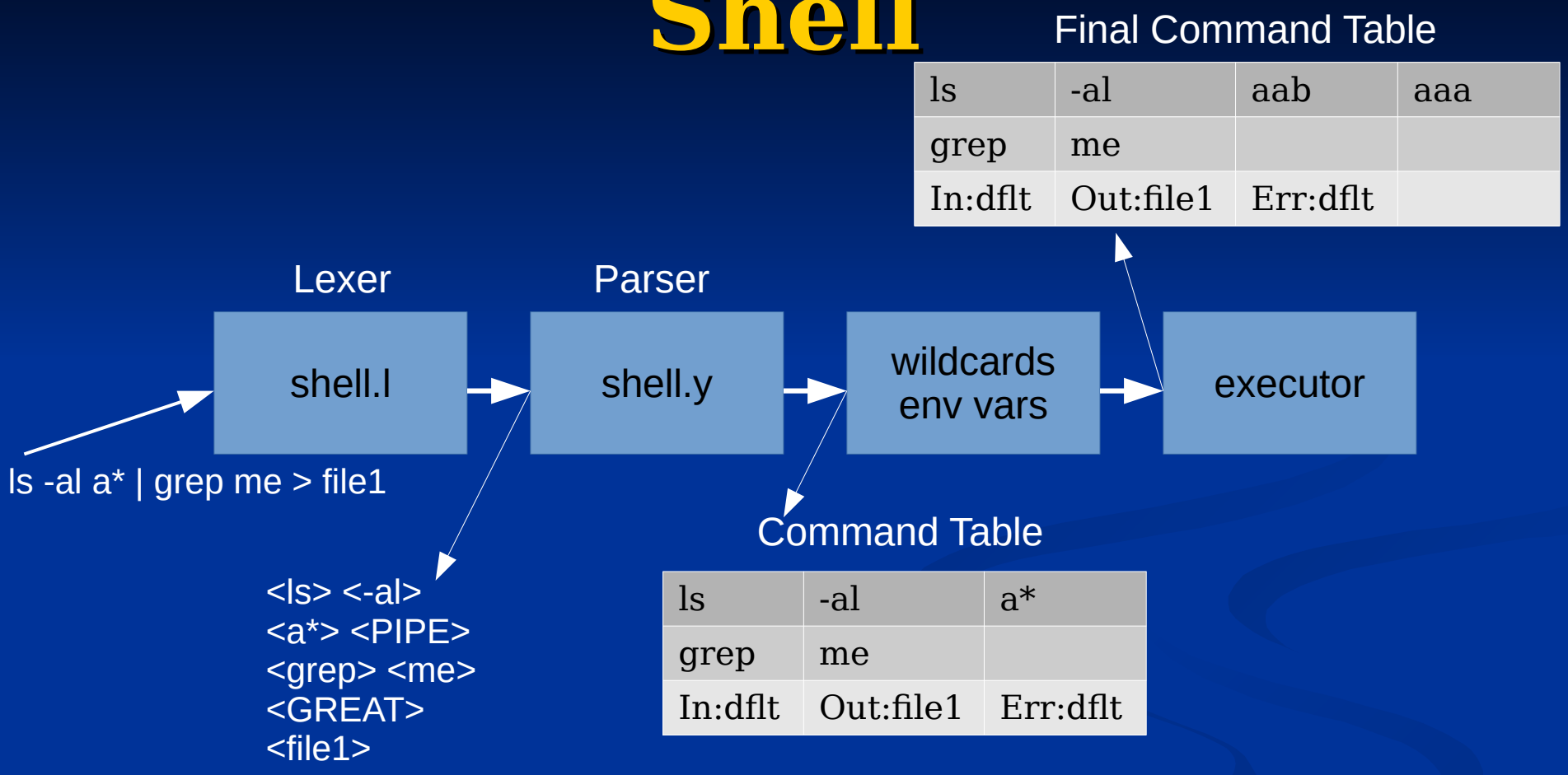
Other subsystems

- Environment variables
  - Setting, printing, expanding
- Wildcards
  - E.g., expanding a*a
- Subshells
  - Backticks ` ` and $( )

Shell project

- Three parts
  - Parsing, execution, and pipes/redirection
    - Read command line
    - Generate command table
    - Create processes, connect them with pipes
      - Also handle I/O redirection
  - Wildcarding and zombies
  - Subshell, env vars, quotes, line editing, etc

Shell


Final Command Table
ls -al aab aaa
grep me
In:dflt Out:file1 Err:dflt
Lexer
Parser
shell.l
shell.y
wildcards
env vars
executor
ls -al a* | grep me > file1
<ls> <-al>
<a*> <PIPE>
<grep> <me>
<GREAT>
<file1>
Command Table
ls -al a*
grep me
In:dflt Out:file1 Err:dflt

Questions?