Operating Systems
CMPSC 473
Operating Systems Structure
January 20, 2009 - Lecture 3
Instructor: Trent Jaeger
• Last class:
  – Computer architecture support for systems

• Today:
  – Operating Systems Structures and Basics
Interlude

• Recap of OS goals
  – Resource management
    • Memory, Devices
  – Scheduling
  – Security
  – Services to programs/applications
Functionality Expected from a Modern OS
Libertarian View

• Everyone should get to do whatever they want
  – As long as they let others live

• Processes should feel they have the entire computer
  – Infinite CPU, RAM, …
  – No threat of someone harming them
Socialistic View

• To each according to his needs
  – Co-operative existence enforced by govt/OS
  – Fair allocation of resources
OS as a Communist Govt.

• Centralized control and monitoring
• Allocate resources efficiently
• Misbehavior $\Rightarrow$ Termination
Theory vs. Practice

• The Theory
  – Many OS problems are NP-complete
    • What’s the best schedule for all possible processes?
  – So, optimal solutions are not possible
    • What do we do?

• Evaluate (preferably, using appropriate workload)
Software Architecture
Operating System Layers

- User application programs
- Network services
- Print queue
- Shells
- Electronic mail
- Device drivers
- System accessories
- File systems
- System call interface
- Timed execution service (cron)
- Other utility programs
- UNIX kernel

Database systems
System Layers

- Application
- Libraries (in application process)
- System Services
- OS API
- Operating system kernel
- Hardware
Applications to Libraries

• Application Programming Interface
  – Library functions (e.g., libc)

• Examples
  – printf of stdio.h

• All within the process’s address space
  – Static and Dynamic linking
Applications to Services

• Provide syntactic sugar for using resources
  – Printing, program mgmt, network mgmt, file mgmt, etc.
  – E.g., chmod

• Provide special functions beyond OS
  – E.g., cron

• UNIX man pages, sections 1 and 8
Libraries to System

• System call interface
  – UNIX man pages, section 2
  – Examples
    • open, read, write – defined in unistd.h
  – Call these via libraries? fopen vs. open

• Special files
  – Drivers, /proc, sysfs
System to Hardware

- Software-hardware interface
- OS kernel functions
  - Concepts == Managers -- Hardware
  - Files == drivers -- devices
  - Address space == virtual memory -- memory
  - Instruction Set == process model -- CPU
- OS provides abstractions of devices and hardware objects (files)
System Call Overview
System Call Handling

Figure 3-7
System service exceptions
System Call Handling

- Procedure call in user process
- Initial work in user mode (libc)
- Trap instruction to invoke kernel (int 0x80)
- Preparation (e.g., sys_read, mmap2)
- I/O command (read from disk)
- Wait (disk is slow)
- Completion (interrupt handling)
- Return-from-interrupt instruction
- Final work in user mode (libc)
- Ordinary return instruction (libc)
File Interface

• Goal: Provide a uniform abstraction for accessing the OS and its resources

• Abstraction: File
  – Use file system calls to access OS services
  – Devices, sockets, pipes, etc.
  – And OS in general
Regular File

- File has a pathname: /tmp/foo
- Can open the file
  - `int fd = open( "/tmp/foo", O_RDWR );`
  - For reading and writing
- Can read from and write to the file
  - `bytes = read( fd, buf, max ); /* buf get output */`
  - `bytes = write( fd, buf, len ); /* buf has input */`
Socket File

- File has a pathname: `/tmp/bar`
  - Files provide a persistence for a communication channel
  - Usually used for local communication (UNIX domain sockets)

- Open, read, and write via socket operations
  - `sockfd = socket(AF_UNIX, TCP_STREAM, 0);`
  - local.path is set to `/tmp/bar`
  - `bind(&local, sockfd, len)`
  - Use sock operations to read and write
Device File

- Files for interacting with physical devices
  - /dev/null (do nothing)
  - /dev/cdrom (CD-drive)
- Use file system operations, but are handled in device-specific ways
  - Open, read, write correspond to device-specific functions
    - Function pointers!
  - Also, use ioctl (I/O control) to interact (later)
Sysfs File and /proc Files

• These files enable reading from and writing to kernel

• /proc files
  – enable reading of kernel state for a process

• Sysfs files
  – Provide functions that update kernel data
    • File’s write function updates kernel based on input data
Summary

• Operating systems must balance many needs
  – Impression that each process has individual use of system
  – Comprehensive management of system resources
• Operating system structures try to make use of system resources straightforward
  – Libraries
  – System services
  – System calls and other interfaces
• Next time: Processes