• 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 => Termination
Theory vs. Practice

• Performance
  – Efficient and fair resource allocation, illusion of unlimited resources

• Isolation
  – Protect everyone from each other and from the OS

• How to do this efficiently?
  – Hardware support
Operating System Approaches

- Monolithic
  - Everything in one program
  - Lacks structure
- Kernel
  - Core services
    - Management of Abstractions
- Microkernel
  - Core concepts
    - Fundamental objects; Management in servers
- Exokernel
  - No abstractions; interface to hardware
Monolithic OS

- Management in kernel
Microkernel OS

• All management is outside kernel
  – Servers
Monolithic Systems
Operating System Layers
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

- Maintain the system software configuration
  - E.g., cron
- Provide syntactic sugar for using resources
  - E.g., printing, network, files, etc.
- UNIX man pages, sections 1 and 8
Libraries to System

• System call interface
  – UNIX man pages, section 2
  – Examples
    • open, read, write, unistd.h

• 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
Resource Manager

- OS is a resource manager
  - Devices, Time, Space, Power, etc.
- OS provides abstractions for resources
  - Files, sockets, etc.
- Resource allocation
- Resource management
Two Processes and an OS

• Access to persistent data
  – File on a disk

• Allocation
  – Each allocates an abstract resource (file)
  – E.g., OS allocates physical resource on disk

• Management
  – Processes may access own files
  – E.g., OS protects access to disk via protecting access to file abstraction
Two Processes and an OS

• OS competes with the processes for resources
  – E.g., OS uses disk to swap

• Allocation
  – Kernel maps memory to disk
  – How does the kernel know disk blocks to use?

• Management
  – System memory is managed in cache, virtually, physically, on disk, …
  – Kernel must keep all of these straight
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
• Resources are allocated and managed by kernel
  – May require mapping between different representations
• Next time: Processes