Review

- Why study OS?
- What is an OS?
- History: Changes driven by technology
  - Phase 1: HW expensive, humans cheap
    - OS optimized for HW (overlap I/O, batch systems…)
  - Phase 2: HW cheap, humans expensive
    - OS optimized for humans (window systems, interactivity, PCs…)
  - Phase 3: HW "free", networks everywhere
    - distributed systems, multimedia, communication…

Outline

- Hardware Support for OS
- What happens on
  - an interrupt / an exception/ a system call
- Operating System Architectures
  - monolithic
  - microkernel
  - virtual machine
- An OS in action
- Processes and Programs

Preview

- Thursday: Process management and scheduling
- Tuesday: Implementing Threads

What's an OS?

- It's a program that mediates between application programs and the hardware
- It provides abstractions to simplify building applications:
  - files instead of "bytes on a disk"
  - contiguous memory regions instead of "bits in a RAM chip"
  - processes to run applications
- It allows application programs to co-exist peacefully
  - enforces security policies
  - enforces safety measures
- Allows effective usage of hardware resources

Hardware Support

- Operating system needs to:
  - control I/O devices
  - control access to the hardware

all while denying these privileges to user programs:
  - for protection
  - for abstraction/ease of use

- Hardware supports two modes of operation (or more):
  - access to hardware & I/O devices is done through privileged instructions, these are only available in "supervisor" mode
  - privileged instructions cannot be executed in "user" mode

Implementation

- Using a bit in the processor
- Using protection rings (Intel Pentium uses 4 protection rings)
- Operating system code runs in supervisor mode, while user program code runs in user mode
- Switching from user to supervisor mode occurs on:
  - interrupts: hardware devices needing service
  - exceptions: user program acts silly (divide by 0, bus error, etc)
  - trap instructions: user program requires OS service (system call)
- Switching back occurs by an RTI instruction

On Interrupts

- Hardware saves state of user programs
- Hardware calls the operating system at a pre-specified location
- Operating system identifies the device and cause of interrupt
- Responds to the interrupt (possibly killing program, <CTRL-C>)
- Operating system restores state of the user program (if applicable) or some other user program
- Execute an RTI instruction to return to the user program
- User program continues exactly at the same point it was interrupted.

Key Fact: None of this is visible to the user program

On Exceptions

- Hardware saves state of user programs
- Hardware calls the operating system at a pre-specified location
- Operating system identifies the cause of the exception (e.g. / 0)
- If user program has exception handling specified, then OS adjusts the user program state so that it calls its handler
- Execute an RTI instruction to return to the user program
- If user program did not have a specified handler, then OS kills it and runs some other user program, as available

Key Fact: Effects of exceptions are visible to user programs and cause abnormal execution flow

On System Calls

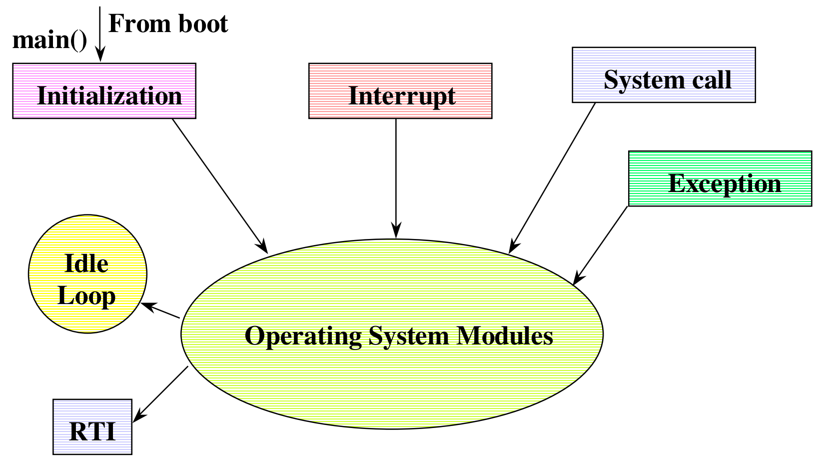
- User program executes a trap instruction (system call)
- Hardware saves state of user programs
- Hardware calls the operating system at a pre-specified location
- Operating system identifies the required service and parameters (e.g. open(filename, O_RDONLY);
- Operating system executes the required service
- Operating system sets a register to contain the result of call
- Execute an RTI instruction to return to the user program
- User program receives the result and continues

Key Fact: To the user program, it appears as a function call executed under program control

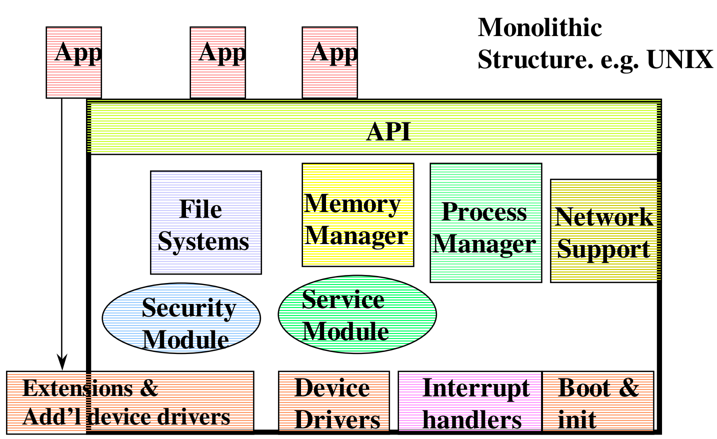
Implications on OS Structure

- An OS is just a program:
  - It has a main() function, which gets called only once (during boot)
  - Like any program, it consumes resources (such as memory), can do silly things (like generating an exception), etc.
- But it is a very strange program:
  - It is “entered” from different locations in response to external events
  - It does not have a single thread of control, it can be invoked simultaneously by two different events (e.g. sys call & an interrupt)
  - It is not supposed to terminate
  - It can execute any instruction in the machine

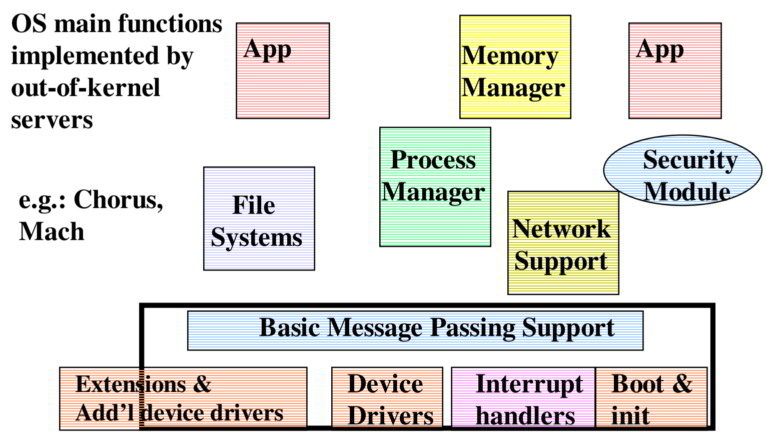
Control Flow in an OS



General OS Structure



Micro-Kernel Organization



Virtual Machine Organization