Base Hardware and OS Support

Operating Systems
Computer Science 354
Dickinson College
Spring 2008
slides courtesy of Professor Grant Braught

Outline

- Review of basic hardware capabilities
- The Bootstrap process
- Hardware support for OS
- Interfacing with the OS
- OS Architectures

Basic Hardware Assumptions

- Single CPU Machine
  - ✓Executes 1 instruction at a time
    - Fetch / Decode / Execute
      - Program Counter (PC): holds memory address for next fetch.
      - Instruction Register (IR): holds instruction for decode/execute.
  - ✓Instruction execution is "atomic".
  - ✓Programs store operands and results in general purpose registers.
    - Register contents are part of a process' "context".

Basic Hardware Assumptions

- Hard Disks:
  - ✓ Basic hard disk controller can:
    - Read a sector (or block)
    - Write a sector (or block)
  - ✓ Sector to read/write is specified by a cylinder:head:sector (CHS) address.
    - Some disk controllers also use linear block addressing (LBA).

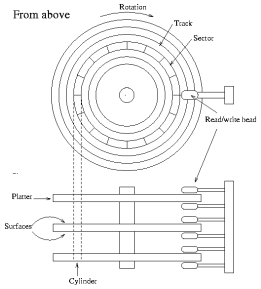


Image from: Linux System Administrators Guide
http://www.tldp.org/LDP/sag/html/hard-disk.html

Basic Hardware Assumptions

- Basic Input/Output System (BIOS)
  - ✓ Contains a number of small programs and subroutines:
    - Power on self test (POST)
    - System configuration utility
      - Settings stored in small amount of battery backed CMOS memory.
    - A set of routines for performing basic operations on common input/output devices.
      - Read / write a specified C:H:S from disk.
      - Read character from keyboard.
      - Display character on the screen.
    - OS bootstrap program
  - ✓ Stored on a Flash ROM that is part of the computer's address space.
    - CMOS for configuration is also in address space

The Boot Sequence

- In the beginning… there is only the BIOS.
  - ✓ The PC is initialized to the address of the POST program contained in the BIOS
  - ✓ The last instruction of the POST jumps to the address of the *bootstrap program*, also contained in the BIOS.
  - ✓ The bootstrap program uses the BIOS routines to load the program contained in the *boot sector* of the *boot disk* into memory at a known address.
    - Boot sector = first sector on the disk (512 bytes).
    - Boot disk is identified by data stored in the configuration CMOS.
  - ✓ The last instruction in the bootstrap program jumps to the address at which the *boot sector program* was loaded.

The Boot Sequence

- The boot sector program proceeds to load the operating system… but usually not directly…
  - ✓ The boot sector program typically loads a *second stage boot loader* from disk.
    - ❖ The second stage boot loader is stored in a known set of contiguous sectors on the disk.
    - ❖ The second stage boot loader knows how to read the file system.
  - ✓ The second stage boot loader finds the C:H:S address of the *OS kernel* using its filename and then uses the BIOS routines to load it into a known location in memory.
  - ✓ The final instruction of the second stage boot loader jumps to the *initialization routine* within the OS kernel.

The Boot Sequence

- The initialization routine within the OS kernel:
  - ✓ Initializes internal OS data structures
  - ✓ Loads device drivers and initializes devices
  - ✓ Starts any services provided by the system
    - ❖ FTP / HTTP / SSH / SMTP etc…
  - ✓ Starts the user interface
    - ❖ Command prompt / GUI / Login screen
      - From there user commands generate new processes.

Boot Sequence Variants

- There are a number of twists on the boot sequence depending on the particulars of the system.
  - ✓ Multiple bootable partitions (I.e. dual boot)
    - ❖ Boot sector program presents a menu.
    - ❖ User picks a boot partition.
    - ❖ A new boot sector program is loaded from the first sector of that partition and the process picks up from there.
  - ✓ Shortcuts
    - ❖ Some systems use larger BIOS bootstrap programs and omit the boot sector program.
  - ✓ Portable devices and small operating systems
    - ❖ Entire OS can be stored in Flash ROM.

Random OS Quote

- Saying that XP is the most stable MS OS is like saying that asparagus is the most articulate vegetable.

  Dave Barry

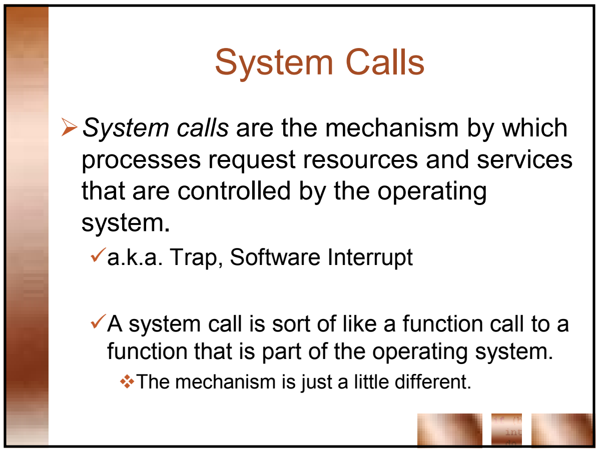
Hardware Support for OS

- All use of shared system resources must be controlled by the operating system if it is to provide:
  - ✓ Protection
  - ✓ Multiprogramming
  - ✓ Timesharing

- Additional hardware is required to ensure that the operating system can control all use of shared resources.

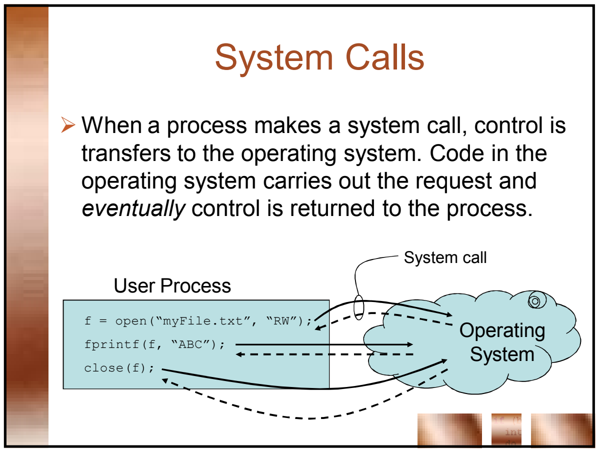
Hardware Support for OS

- The hardware support that is required is provided by:
  - ✓ A mechanism for making *system calls*
  - ✓ A mechanism for handling *Interrupts*
  - ✓ *Dual mode* processor operation
  - ✓ *Base and limit registers* for protecting memory.
  - ✓ A *Timer* hardware device



System Calls

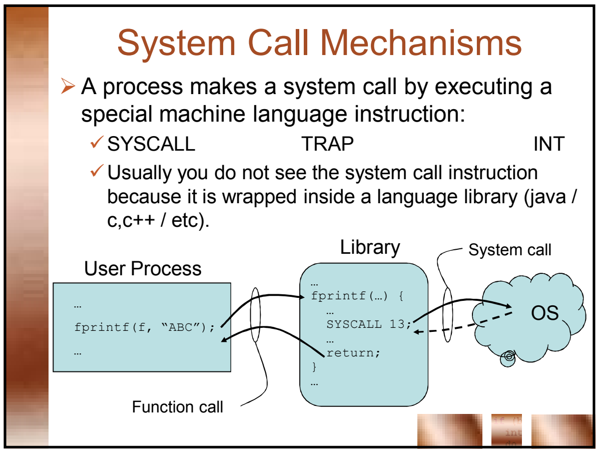
- *System calls* are the mechanism by which processes request resources and services that are controlled by the operating system.
  - ✓a.k.a. Trap, Software Interrupt
  - ✓A system call is sort of like a function call to a function that is part of the operating system.
    - ❖The mechanism is just a little different.



System Calls

- When a process makes a system call, control is transfers to the operating system. Code in the operating system carries out the request and *eventually* control is returned to the process.

```
f = open("myFile.txt", "RW");
fprintf(f, "ABC");
close(f);
```

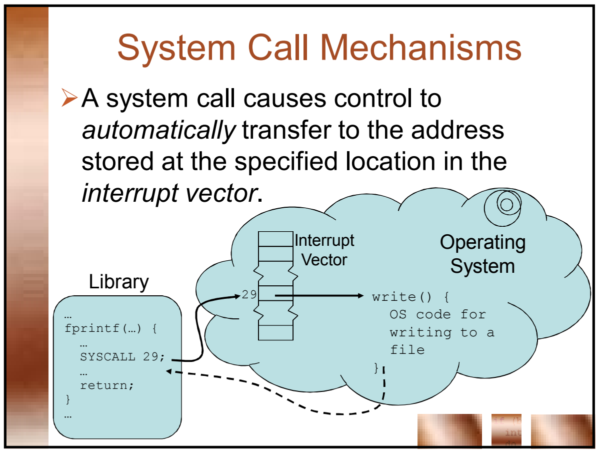


System Call Mechanisms

- A process makes a system call by executing a special machine language instruction:
  - ✓SYSCALL TRAP INT
  - ✓Usually you do not see the system call instruction because it is wrapped inside a language library (java / c,c++ / etc).

```
…
fprintf(f, "ABC");
…
```

```
…
fprintf(…) {
  …
  SYSCALL 13;
  …
  return;
}
…
```

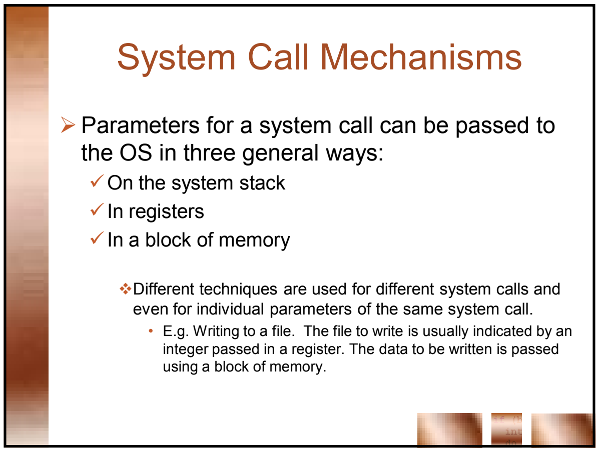


System Call Mechanisms

- A system call causes control to *automatically* transfer to the address stored at the specified location in the *interrupt vector*.

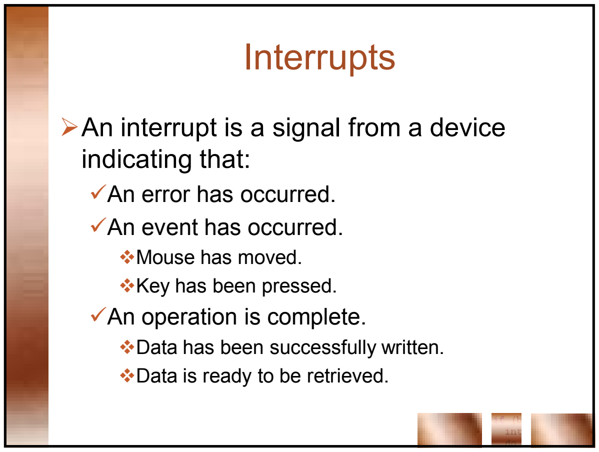
```
…
fprintf(…) {
  …
  SYSCALL 29;
  …
  return;
}
…
```

```
write() {
  OS code for
  writing to a
  file
}
```



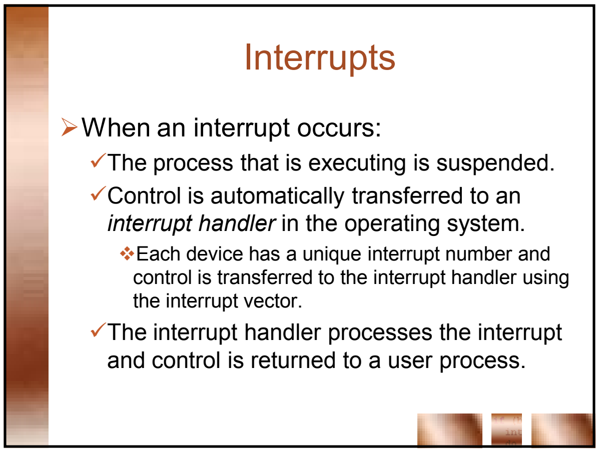
System Call Mechanisms

- Parameters for a system call can be passed to the OS in three general ways:
  - ✓On the system stack
  - ✓In registers
  - ✓In a block of memory
    - ❖Different techniques are used for different system calls and even for individual parameters of the same system call.
      - E.g. Writing to a file. The file to write is usually indicated by an integer passed in a register. The data to be written is passed using a block of memory.



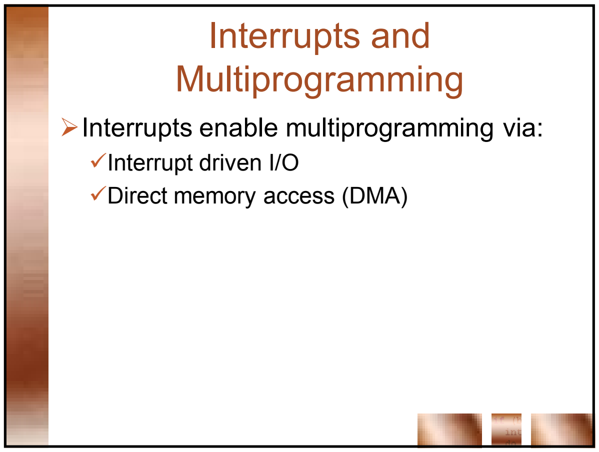
Interrupts

- An interrupt is a signal from a device indicating that:
  - ✓An error has occurred.
  - ✓An event has occurred.
    - ❖Mouse has moved.
    - ❖Key has been pressed.
  - ✓An operation is complete.
    - ❖Data has been successfully written.
    - ❖Data is ready to be retrieved.



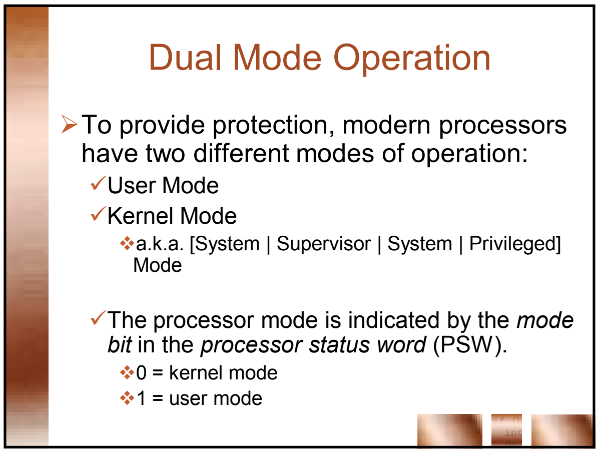
Interrupts

- When an interrupt occurs:
  - ✓The process that is executing is suspended.
  - ✓Control is automatically transferred to an *interrupt handler* in the operating system.
    - Each device has a unique interrupt number and control is transferred to the interrupt handler using the interrupt vector.
  - ✓The interrupt handler processes the interrupt and control is returned to a user process.



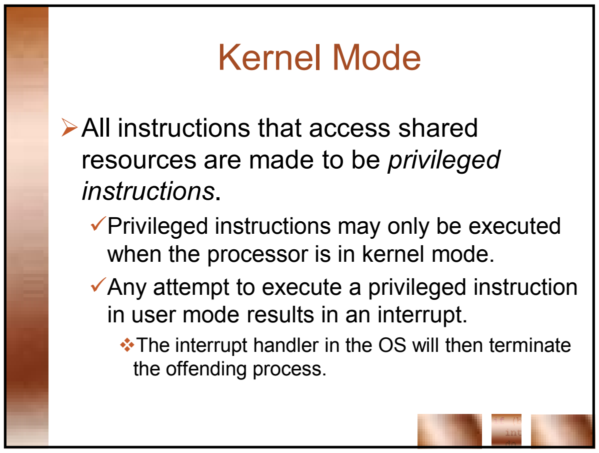
Interrupts and Multiprogramming

- Interrupts enable multiprogramming via:
  - ✓Interrupt driven I/O
  - ✓Direct memory access (DMA)



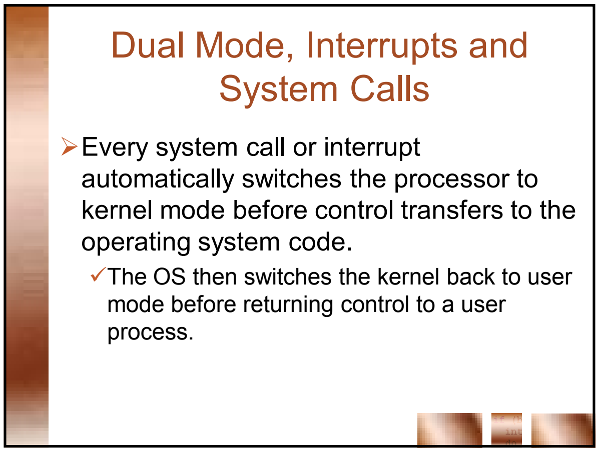
Dual Mode Operation

- To provide protection, modern processors have two different modes of operation:
  - ✓User Mode
  - ✓Kernel Mode
    - a.k.a. [System | Supervisor | System | Privileged] Mode
  - ✓The processor mode is indicated by the *mode bit* in the *processor status word* (PSW).
    - 0 = kernel mode
    - 1 = user mode



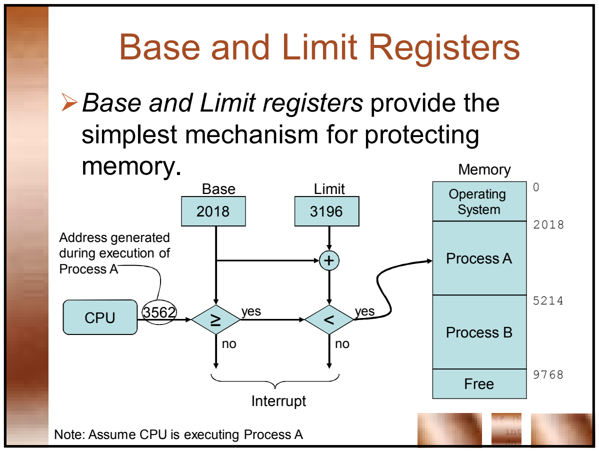
Kernel Mode

- All instructions that access shared resources are made to be *privileged instructions*.
  - ✓Privileged instructions may only be executed when the processor is in kernel mode.
  - ✓Any attempt to execute a privileged instruction in user mode results in an interrupt.
    - The interrupt handler in the OS will then terminate the offending process.



Dual Mode, Interrupts and System Calls

- Every system call or interrupt automatically switches the processor to kernel mode before control transfers to the operating system code.
  - ✓The OS then switches the kernel back to user mode before returning control to a user process.



Base and Limit Registers

- *Base and Limit registers* provide the simplest mechanism for protecting memory.

Note: Assume CPU is executing Process A

Timer Device

- Time sharing is enabled by the *timer device.*
  - The timer is usually implemented using a fixed rate clock and a counter.
    - The counter is set to a positive value.
    - The value of the counter is then decremented on each tick of the clock.
    - When the counter reaches 0 an interrupt is generated.

Random OS Quote

- One of the main advantages of Unix over, say, MVS, is the tremendous number of features Unix lacks.
  Chris Torek

OS Implementation

Implementing Operating Systems

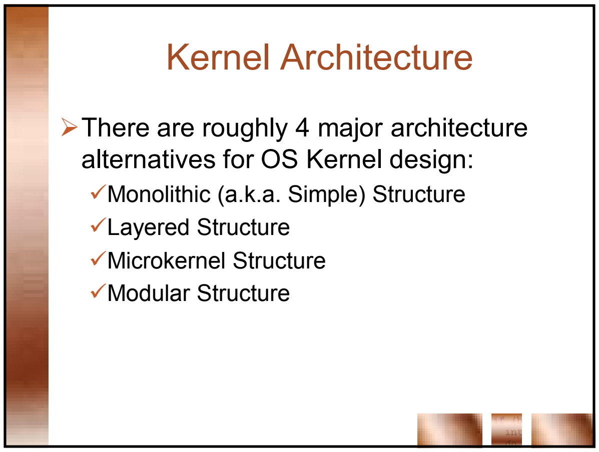
- Some of the design decisions faced in implementing an operating system include:
  - System software vs. OS kernel
  - Separation of mechanism and policy
  - Kernel architecture

System Software vs. Kernel

- Many services can be implemented either in the OS kernel or as a processes that can be run in user mode.

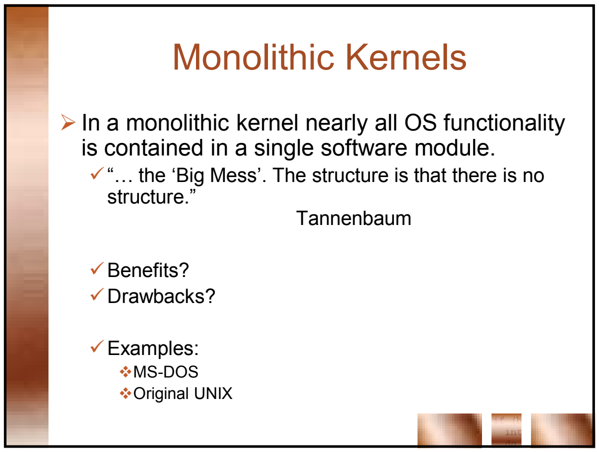
Mechanism and Policy

- *Policies* are likely to change over time and thus should be separate from the *mechanisms* used to enforce them.
  - An ideal mechanism is general enough to support a wide range of policies.



Kernel Architecture

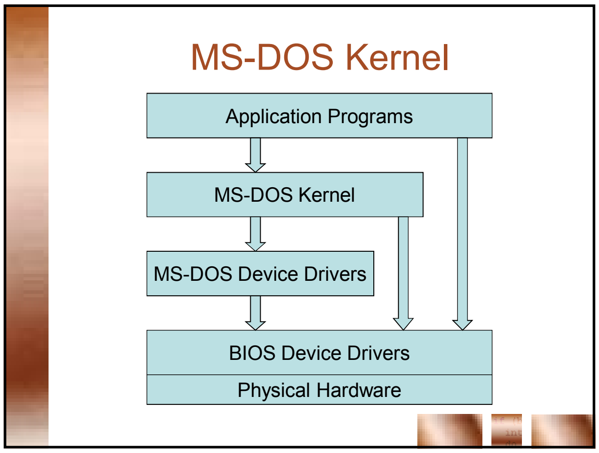
- There are roughly 4 major architecture alternatives for OS Kernel design:
  - ✓Monolithic (a.k.a. Simple) Structure
  - ✓Layered Structure
  - ✓Microkernel Structure
  - ✓Modular Structure



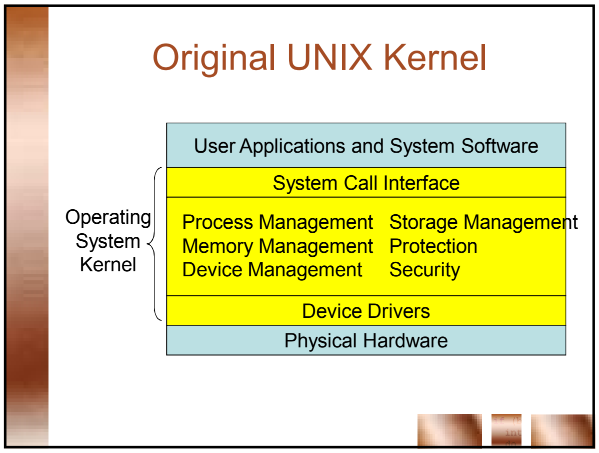
Monolithic Kernels

- In a monolithic kernel nearly all OS functionality is contained in a single software module.
  - ✓"… the 'Big Mess'. The structure is that there is no structure."

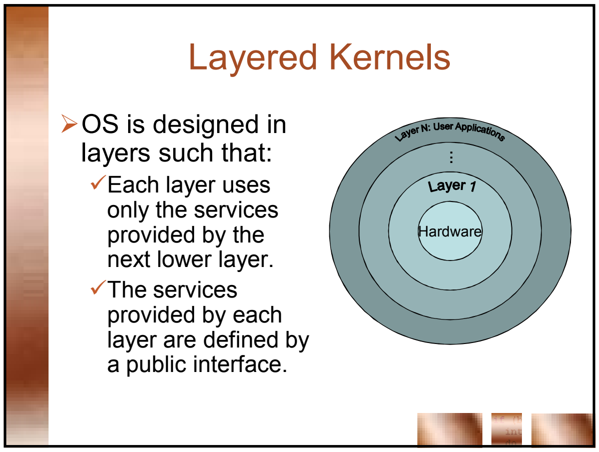
    Tannenbaum
  - ✓Benefits?
  - ✓Drawbacks?
  - ✓Examples:
    - MS-DOS
    - Original UNIX



MS-DOS Kernel

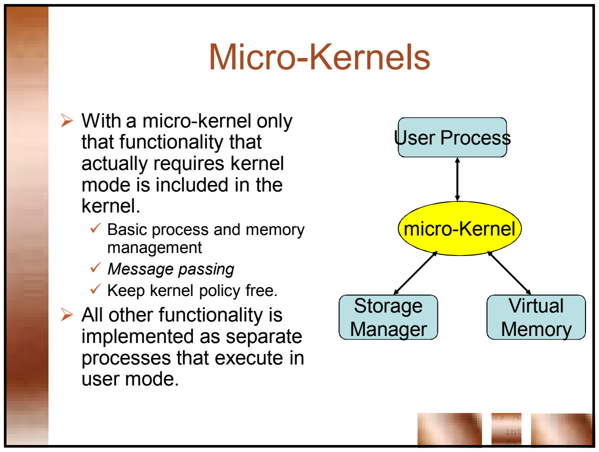


Original UNIX Kernel



Layered Kernels

- OS is designed in layers such that:
  - ✓Each layer uses only the services provided by the next lower layer.
  - ✓The services provided by each layer are defined by a public interface.



Micro-Kernels

- With a micro-kernel only that functionality that actually requires kernel mode is included in the kernel.
  - ✓Basic process and memory management
  - ✓*Message passing*
  - ✓Keep kernel policy free.
- All other functionality is implemented as separate processes that execute in user mode.

Modular Kernels

- Modular kernels have a core set of capabilities (almost a micro-kernel) but then also allow other modules to be dynamically added to the kernel during boot or during execution.

Virtual Machines

- Virtual machines provide a mechanism for hosting multiple independent operating systems on a single machine.

VMWare

- The VMWare virtualization later runs as an application on a host operating system.
- This application appears to the guest operating system as if it is a complete machine with its own CPU, memory and I/O devices.

Image from VMWare Whitepaper.

Random OS Quote

- We just don't think a Linux partition on a mainframe makes a lot of sense. It's kind of like having a trailer park in the back of your estate.

  Scott McNealy

Road Map

- This topic:
  - ✓ Base hardware and support for operating systems.
  - ✓ OS designs and implementation
- Next topic:
  - ✓ Process management
- Later:
  - ✓ Concurrent programming
  - ✓ Memory management
  - ✓ Storage management
  - ✓ Protection and Security