CSC 4103 - Operating Systems
Fall 2009

LECTURE - II

OS STRUCTURES

Tevfik Koşar

Louisiana State University
August 27th, 2009

Announcements

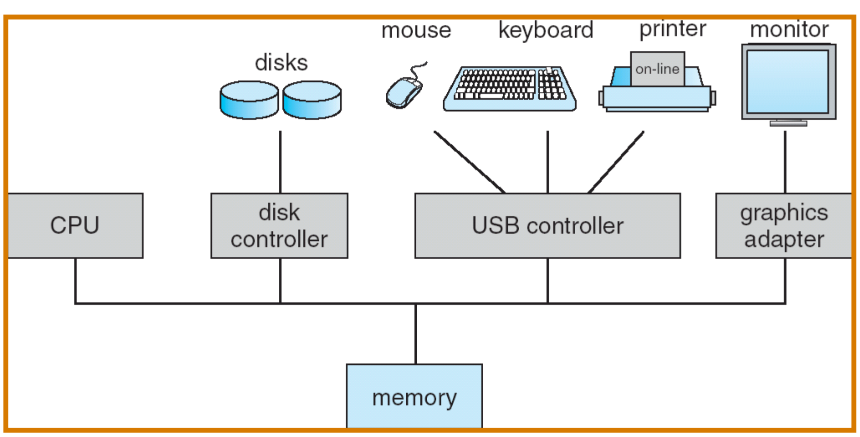
- TA Changed. New TA:
  - Praveenkumar Kondikoppa
  - Email: pkondi1@lsu.edu
- All of you should be now in the class mailing list.
  - Let me know if you haven't received any messages yet.
- Lecture notes are available on the course web site.

Roadmap

- Major OS Components
  - Corresponding OS Responsibilities
- OS Design and Implementation
  - Different Design Approaches
- OS API
  - System Calls
  - Dual Mode of Operation

Computer System Organization

- Computer-system operation
  - One or more CPUs, device controllers connect through common bus providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory cycles



Major OS Components

- Processes
- Memory management
- CPU Scheduling
- I/O Management

Processes

➢ A process is the activity of executing a program

Pasta for six

- boil 1 quart salty water
- stir in the pasta
- cook on medium until "al dente"
- serve

thread of execution

Program

CPU

input data

Process

Processes

- It can be interrupted to let the CPU execute a higher-priority process

Pa…
- boil… wat…
- stir…
- coo… unt…
- serv…

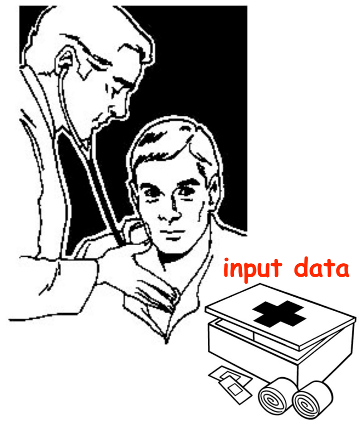
First aid

- Get the first aid kit
- Check pulse
- Clean wound with alcohol
- Apply band aid

thread of execution

Program

CPU (changes hat to "doctor")



input data

Process

Processes

- . . . and then resumed exactly where the CPU left off

Pasta for six

- boil 1 quart salty water
- stir in the pasta
- cook on medium until "al dente"
- serve

thread of execution

Program

hmm... now where was I?

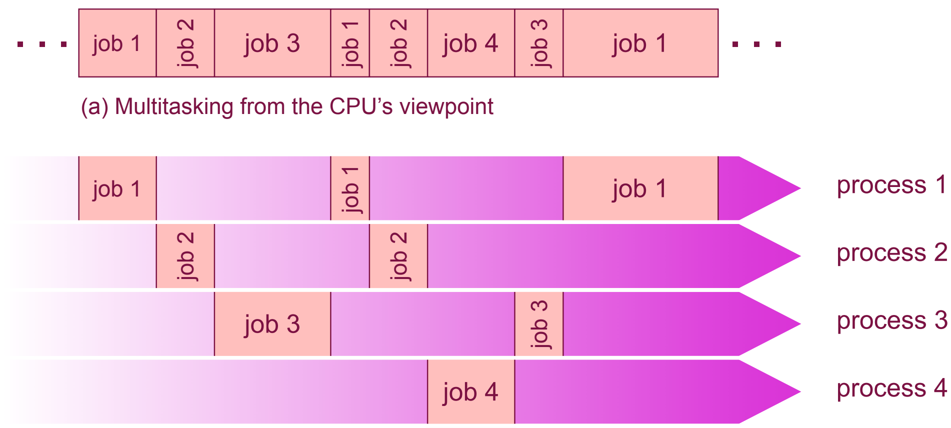
CPU (back to "chef")

input data

Process

Processes

➢ **Multitasking gives the illusion of parallel processing (independent virtual program counters) on one CPU**



(a) Multitasking from the CPU's viewpoint

(b) Multitasking from the processes' viewpoint = 4 virtual program counters

Pseudoparallelism in multitasking

Processes

- **Timesharing** is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - **Response time** should be < 1 second
  - Each user has at least one program loaded in memory and executing ⇨ **process**

Processes

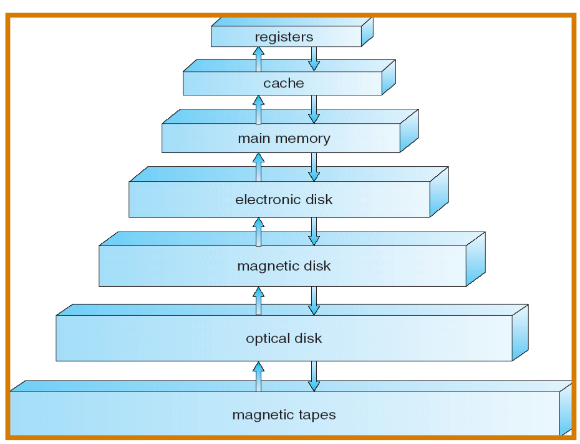
- <u>Operating System Responsibilities:</u>

<u>The O/S is responsible for managing processes</u>

- ✓ the O/S creates & deletes processes
- ✓ the O/S suspends & resumes processes
- ✓ the O/S provides mechanisms for process synchronization
- ✓ the O/S provides mechanisms for interprocess communication
- ✓ the O/S provides mechanisms for deadlock handling

Memory Management

- <u>Main memory</u>
  - ✓ large array of words or bytes, each with its own address
  - ✓ repository of quickly accessible data shared by the CPU and I/O devices
  - ✓ volatile storage that loses its contents in case of system failure



The storage hierarchy

Performance of Various Levels of Storage

- Movement between levels of storage hierarchy can be explicit or implicit

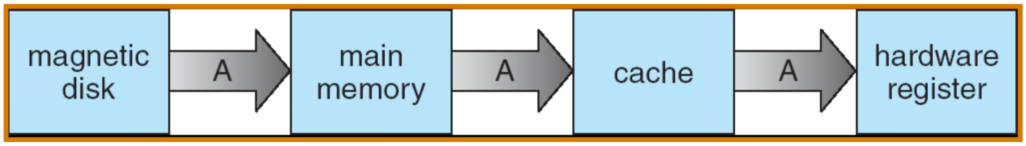
Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

Caching

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
  - If it is, information used directly from the cache (fast)
  - If not, data copied to cache and used there
- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy

Migration of Integer A from Disk to Register

- Multitasking environments must be careful to use most recent value, not matter where it is stored in the storage hierarchy



- Multiprocessor environment must provide cache coherency in hardware such that all CPUs have the most recent value in their cache
- Distributed environment situation even more complex
  - Several copies of a datum can exist

Memory Management

➢ <u>Operating System Responsibilities:</u>

<u>The O/S is responsible for an efficient and orderly control of storage allocation</u>

- ✓ ensures process isolation: it keeps track of which parts of memory are currently being used and by whom
- ✓ allocates and deallocates memory space as needed: it decides which processes to load or swap out
- ✓ regulates how different processes and users can sometimes share the same portions of memory
- ✓ transfers data between main memory and disk and ensures long-term storage

CPU Scheduling

- **Long-term scheduling**
  - ✓ the decision to add a program to the pool of processes to be executed (job scheduling)
- **Medium-term scheduling**
  - ✓ the decision to add to the number of processes that are partially or fully in main memory ("swapping")
- **Short-term scheduling = CPU scheduling**
  - ✓ the decision as to which available processes in memory are to be executed by the processor ("dispatching")
- **I/O scheduling**
  - ✓ the decision to handle a process's pending I/O request

fine- to coarse-grain level

frequency of intervention

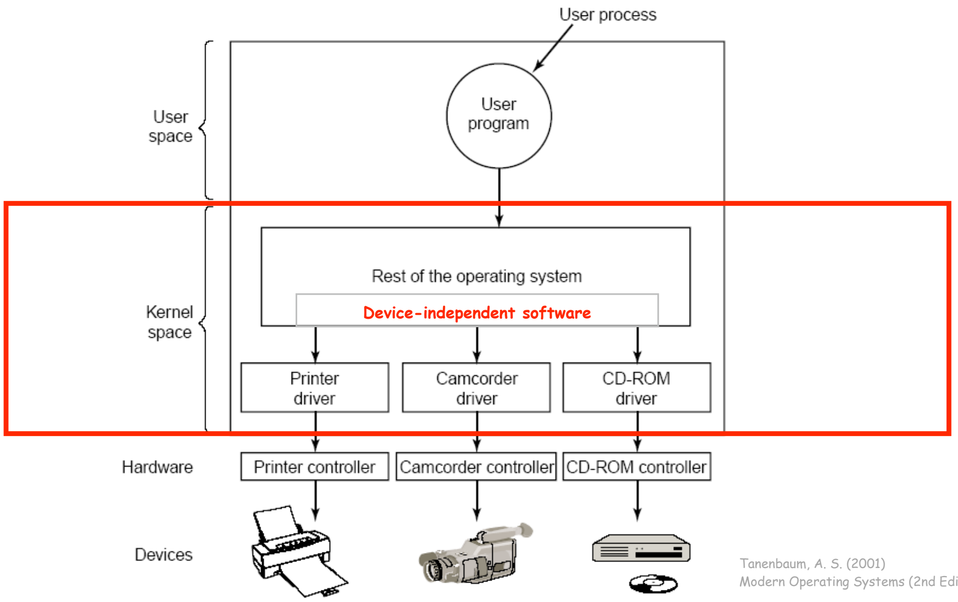
CPU Scheduling

- Operating System Responsibilities:

The O/S is responsible for efficiently using the CPU and providing the user with short response times

- ✓ decides which available processes in memory are to be executed by the processor
- ✓ decides what process is executed when and for how long, also reacting to external events such as I/O interrupts
- ✓ relies on a scheduling algorithm that attempts to optimize CPU utilization, throughput, latency, and/or response time, depending on the system requirements

I/O Management



Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

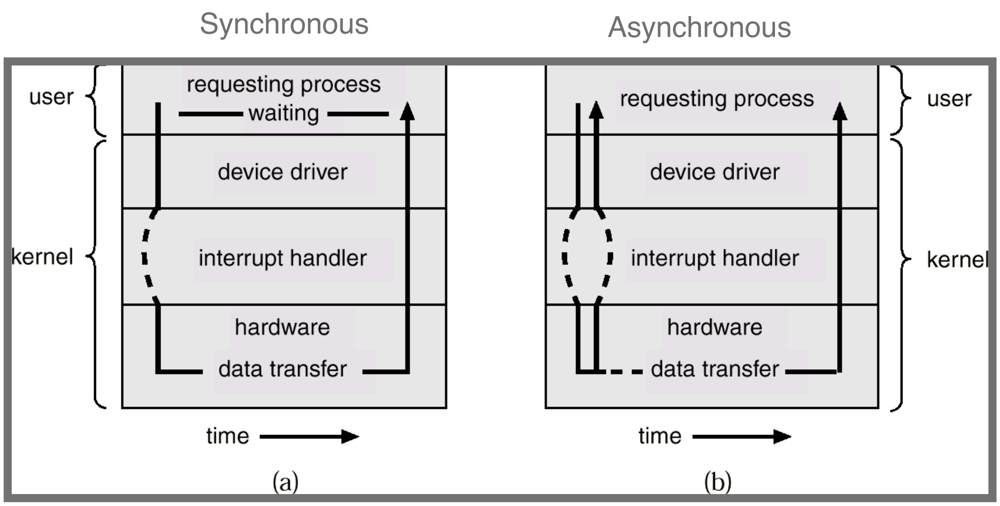
Layers of the I/O subsystem

Two I/O Methods

- After I/O starts, control returns to user program only upon I/O completion ➔ synchronous
  - Wait instruction idles the CPU until the next interrupt
  - Wait loop (contention for memory access).
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing.

- After I/O starts, control returns to user program without waiting for I/O completion ➔asynchronous
  - *System call* – request to the operating system to allow user to wait for I/O completion.
  - *Device-status table* contains entry for each I/O device

Two I/O Methods



I/O Management

- <u>Operating System Responsibilities:</u>

<u>The O/S is responsible for controlling access to all the I/O devices</u>

- ✓ hides the peculiarities of specific hardware devices from the user
- ✓ issues the low-level commands to the devices, catches interrupts and handles errors
- ✓ relies on software modules called "device drivers"
- ✓ provides a device-independent API to the user programs, which includes buffering

OS Design Approaches

Operating System Design and Implementation

- Start by defining goals and specifications
- Affected by choice of hardware, type of system
  - Batch, time shared, single user, multi user, distributed
- *User* goals and *System* goals
  - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient
- No unique solution for defining the requirements of an OS

  → Large variety of solutions

  → Large variety of OS

Operating System Design and Implementation (Cont.)

- Important principle: to separate policies and mechanisms

  Policy: What will be done?
  Mechanism: How to do something?
- Eg. to ensure CPU protection
  - Use Timer construct (mechanism)
  - How long to set the timer (policy)
- The separation of policy from mechanism allows maximum flexibility if policy decisions are to be changed later

OS Design Approaches

- Simple Structure (Monolithic)
- Layered Approach
- Microkernels
- Modules

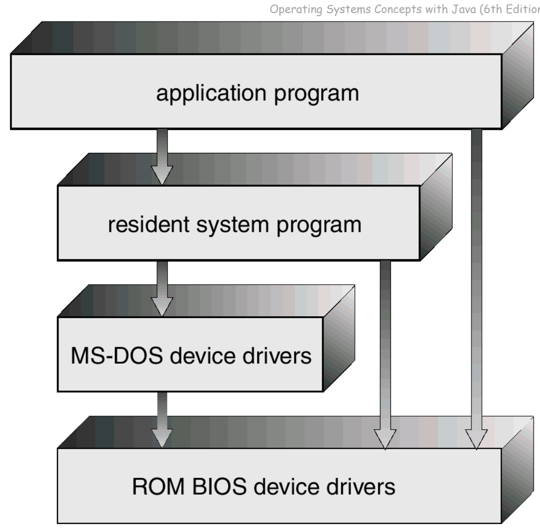
Simple Structure

- Monolithic
- No well defined structure
- Start as small, simple, limited systems, and then grow
- No Layers, not divided into modules

Simple Structure

- Example: MS-DOS
  - ✓ initially written to provide the most functionality in the least space
  - ✓ started small and grew beyond its original scope
  - ✓ levels not well separated: programs could access I/O devices directly
  - ✓ excuse: the hardware of that time was limited (no dual user/kernel mode)

Silberschatz, A., Galvin, P. B. and Gagne. G. (2003) Operating Systems Concepts with Java (6th Edition).

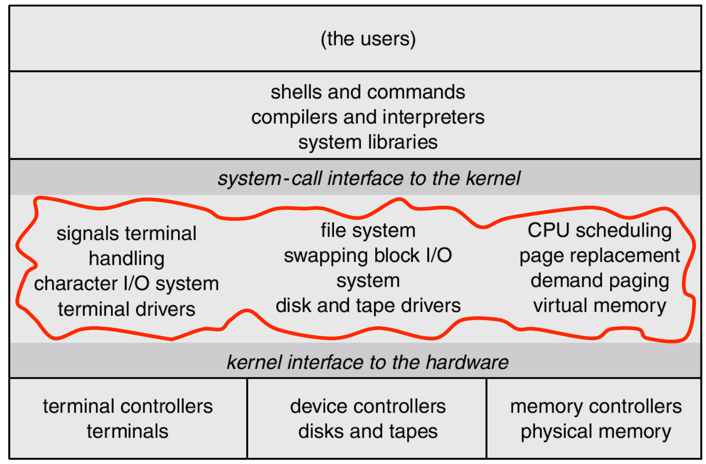


MS-DOS pseudolayer structure

Simple Structure

- ➢ Another example: the original UNIX
  - ✓ enormous amount of functionality crammed into the kernel - everything below system call interface
  - ✓ "The Big Mess": a collection of procedures that can call any of the other procedures whenever they need to
  - ✓ no encapsulation, total visibility across the system
  - ✓ very minimal layering made of thick, monolithic layers

Silberschatz, A., Galvin, P. B. and Gagne. G. (2003) *Operating Systems Concepts with Java (6th Edition).*



UNIX system structure

Layered Approach

- ➢ Monolithic operating systems
  - ✓ no one had experience in building truly large software systems
  - ✓ the problems caused by mutual dependence and interaction were grossly underestimated
  - ✓ such lack of structure became unsustainable as O/S grew
- ➢ Enter hierarchical layers and information abstraction
  - ✓ each layer is implemented exclusively using operations provided by lower layers
  - ✓ it does not need to know how they are implemented
  - ✓ hence, lower layers hide the existence of certain data structures, private operations and hardware from upper layers

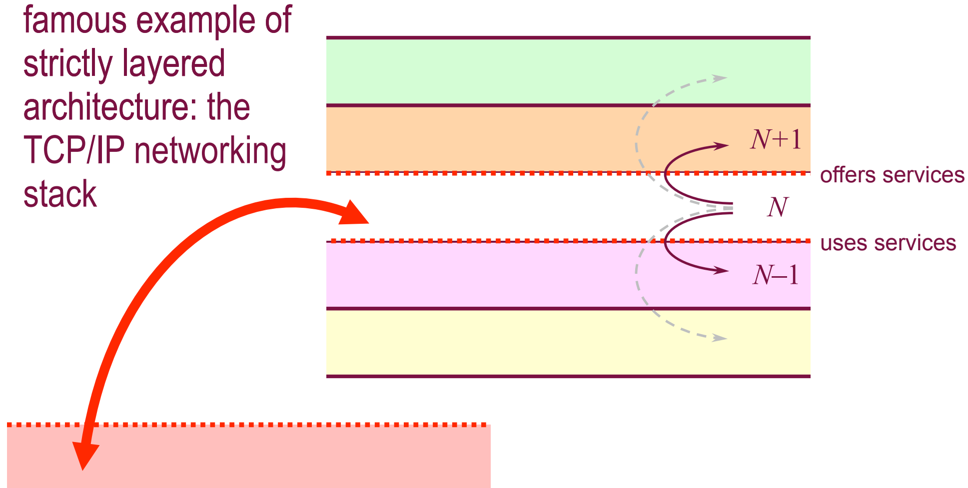
Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers.
  - The bottom layer (layer 0), is the hardware;
  - The highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers
  - GLUnix: Global Layered Unix

Layered Approach

➢ <u>Layers can be debugged and replaced independently without bothering the other layers above and below</u>

✓ famous example of strictly layered architecture: the TCP/IP networking stack



Layered Approach

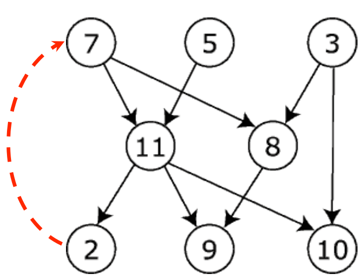
Theoretical model of operating system design hierarchy

	Level	Name	Objects	Example Operations
shell	13	Shell	User programming environment	Statements in shell language
O/S	12	User processes	User processes	Quit, kill, suspend, resume
O/S	11	Directories	Directories	Create, destroy, attach, detach, search, list
O/S	10	Devices	External devices, such as printers, displays, and keyboards	Open, close, read, write
O/S	9	File system	Files	Create, destroy, open, close, read, write
O/S	8	Communications	Pipes	Create, destroy, open, close, read, write
O/S	7	Virtual memory	Segments, pages	Read, write, fetch
O/S	6	Local secondary store	Blocks of data, device channels	Read, write, allocate, free
O/S	5	Primitive processes	Primitive processes, semaphores, ready list	Suspend, resume, wait, signal
hardware	4	Interrupts	Interrupt-handling programs	Invoke, mask, unmask, retry
hardware	3	Procedures	Procedures, call stack, display	Mark stack, call, return
hardware	2	Instruction set	Evaluation stack, microprogram interpreter, scalar and array data	Load, store, add, subtract, branch
hardware	1	Electronic circuits	Registers, gates, buses, etc.	Clear, transfer, activate, complement

Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

Layered Approach

- **Major difficulty with layering**
  - ✓ . . . appropriately defining the various layers!
  - ✓ layering is only possible if all function dependencies can be sorted out into a Directed Acyclic Graph (DAG)
  - ✓ however there might be conflicts in the form of circular dependencies ("cycles")



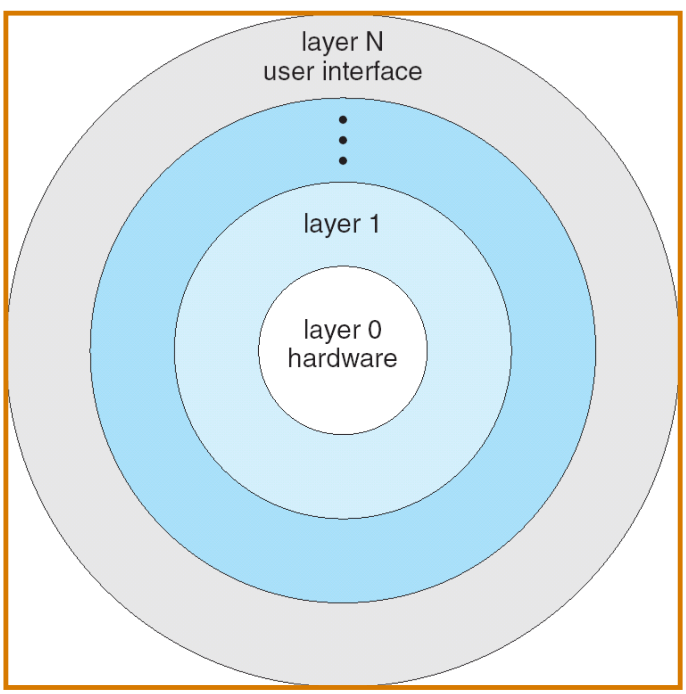
Circular dependency on top of a DAG

Layered Approach

- <u>Circular dependencies in an O/S organization</u>
  - ✓ example: disk driver routines vs. CPU scheduler routines
    - the device driver for the backing store (disk space used by virtual memory) may need to wait for I/O, thus invoke the CPU-scheduling layer
    - the CPU scheduler may need the backing store driver for swapping in and out parts of the table of active processes

- <u>Other difficulty: efficiency</u>
  - ✓ the more layers, the more indirections from function to function and the bigger the overhead in function calls
  - ® backlash against strict layering: return to fewer layers with more functionality

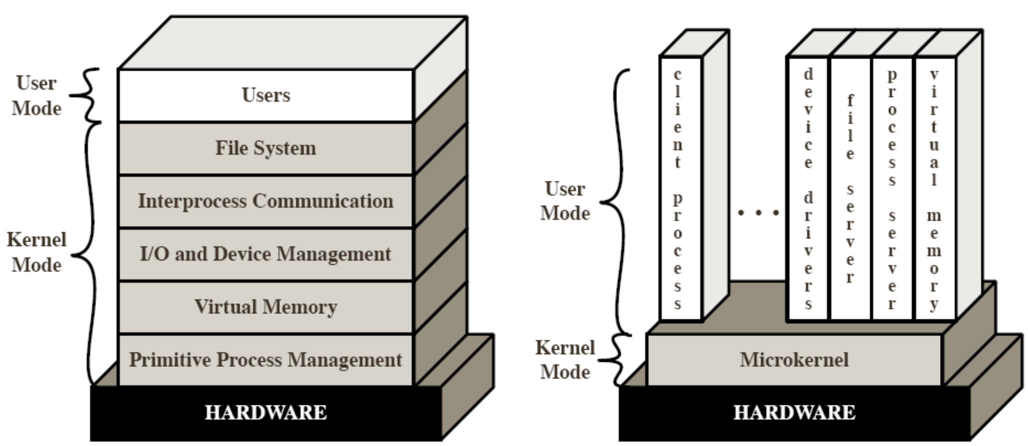
Layered Operating System - Circular View



Microkernel System Structure

- The microkernel approach
  - ✓ a microkernel is a reduced operating system core that contains only essential O/S functions
  - ✓ the idea is to minimize the kernel by moving up as much functionality as possible from the kernel into user space
  - ✓ many services traditionally included in the O/S are now external subsystems running as user processes
    - device drivers
    - file systems
    - virtual memory manager
    - windowing system
    - security services, etc.
- Examples: QNX, Tru64 UNIX

Layered OS vs Microkernel



(a) Layered kernel (b) Microkernel

Microkernel System Structure

- Benefits of the microkernel approach
  - **extensibility** — it is easier to extend a microkernel-based O/S as new services are added in user space, not in the kernel
  - **portability** — it is easier to port to a new CPU, as changes are needed only in the microkernel, not in the other services
  - **reliability & security** — much less code is running in kernel mode; failures in user-space services don't affect kernel space

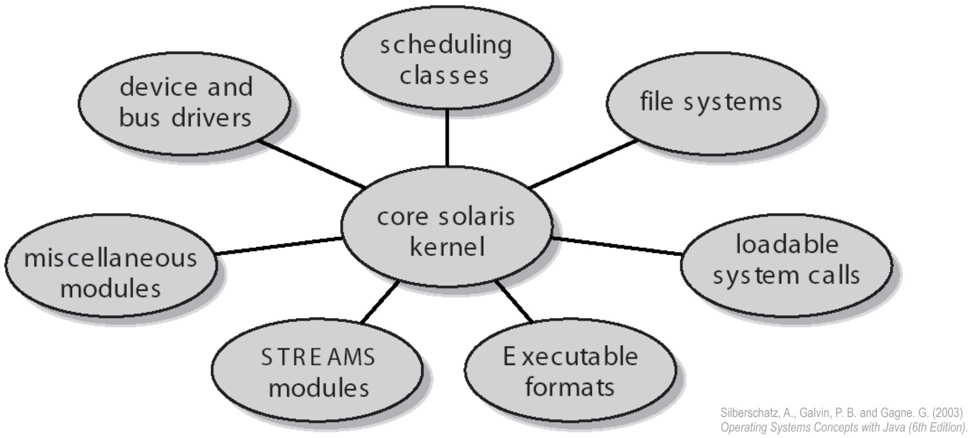
- Detriments of the microkernel approach
  - again, performance overhead due to communication from user space to kernel space
  - not always realistic: some functions (I/O) must remain in kernel space, forcing a separation between "policy" and "mechanism"

Modular Approach

- The modular approach
  - most modern operating systems implement kernel **modules**
  - this is similar to the object-oriented approach:
    - each core component is separate
    - each talks to the others over known interfaces
    - each is loadable as needed within the kernel
  - overall, modules are similar to layers but with more flexibility
  - modules are also similar to the microkernel approach, except they are inside the kernel and don't need message passing

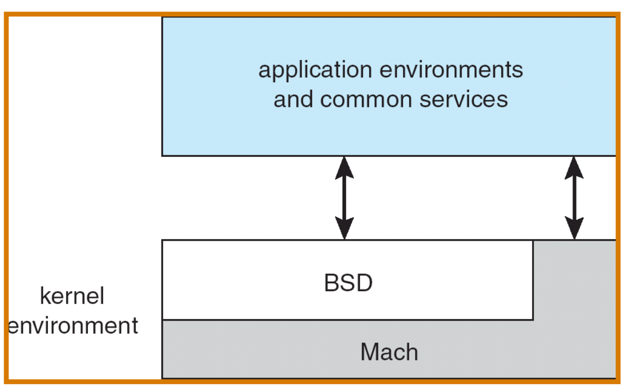
Modular Approach

- <u>Modules are used in Solaris, Linux and Mac OS X</u>



The Solaris loadable modules

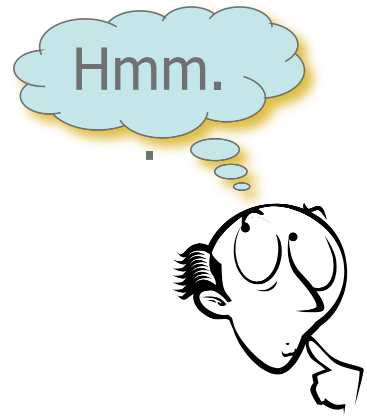
Mac OS X Structure - Hybrid



- BSD: provides support for command line interface, networking, file system, POSIX API and threads
- Mach: memory management, RPC, IPC, message passing

Summary

- Major OS Components
  - Processes, Memory Management, CPU Scheduling, I/O Management
  - Corresponding OS Responsibilities
- OS Design and Implementation
  - Monolithic Systems, Layered Approach, Microkernels, Modules



- Next Lecture: Processes
- Reading Assignment: Chapter 2 from Silberschatz.

Acknowledgements

- “Operating Systems Concepts” book and supplementary material by A. Silberschatz, P. Galvin and G. Gagne
- “Operating Systems: Internals and Design Principles” book and supplementary material by W. Stallings
- “Modern Operating Systems” book and supplementary material by A. Tanenbaum
- R. Doursat and M. Yuksel from UNR