澳 門 理 工 學 院
Instituto Politécnico de Macau
Macao Polytechnic Institute

School of Applied Sciences (BSc. in Computing)

Notes #2: Overview of Operating Systems

COMP 213
(21121/21221)
Operating Systems

2019-2020 1st Semester

Review of Last Lecture

- Computing systems
  - CPU
  - Bus
  - Memory system, memory cache and buffer cache
  - DMA, programmed I/O, interrupts

Eddie Law 2

Topics Today

- Introduction of operating systems
- What is an operating system?
- Some history
- Basic concepts
- Textbook: Chapter 2.1, 2.2, 2.4, 2.8, 3.5, 4.3

Commercial Operating System Products

- Commercial brands
  - Microsoft's Windows 10
  - Red Hat Enterprise (Linux underneath)
  - Apple's OSX (BSD underneath) and iOS
  - Blackberry's QNX
  - Google's Android (Linux)
- Open source and free operating systems
  - Linux
  - FreeBSD and other BSD variants (BSD: Berkeley Software Distribution)
  - OpenSolaris

From Machines to Computers

- Marked the arrival of the operating systems
  - History of OS
  - Generations of OS
  - Classifications of OS
  - Change of cost models

Generations of Computing Systems

1945	1st	1955	2nd	1965	3rd	1980	4th	2010	5th	
	Vacuum tubes, plug boards		Transistors, batch systems (multi-programmed batch systems)		ICs and multi-programming (time-sharing)		Embedded systems; parallel, distributed real-time computing		Cloud Computing, IoT	…

All hardware; no OS concept initially; ran with plunge cards

Computers disappeared from eyesight; sophisticated OS; intelligence created in software

Design Motivation - Cost Models

- Costs of hardware and human resources

Time

Cost model 1
- Hardware expensive?
- Low labour cost?
- Should make hardware more efficient

Cost model 2
- In transition

Cost model 3
- Hardware cheap?
- People cost is expensive?
- Should make hardware easier to use!

Evolution of Operating Systems

- A major OS will evolve over time for a number of reasons:

Hardware upgrades

New types of hardware

New services

Fixes

Similar to What Watches Today

- Many hundreds years ago, all were mechanicals
- Low-budgeted electronic watches arose
  - Basic microcontroller
  - Assembly language could work
  - → Trivial embedded system

Eddie Law 9

Similar to What Watches Today (cont'd)

- Nowadays, aside from keeping time, some watches have other applications, e.g., personal healthcare information
  - E.g., step count per day, heart beat rate

Apple watch
– Multi-core system & iOS

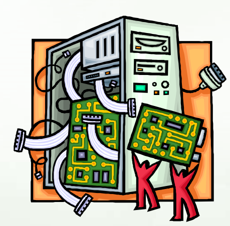
Eddie Law 10

Early Development Stages for OS

Serial Processing → Simple Batch Systems → Multi-programmed Batch Systems → Time Sharing Systems

The Evolution on OS

- **Early generation hardware numeric machine**
  - No OS
  - Vacuum tubes, plug boards, toggle switches, etc.
  - Users accessed it in “series” or “queue”
  - **Scenario 1**: expensive hardware, low labor cost
    $\Rightarrow$ so maximize hardware utilization



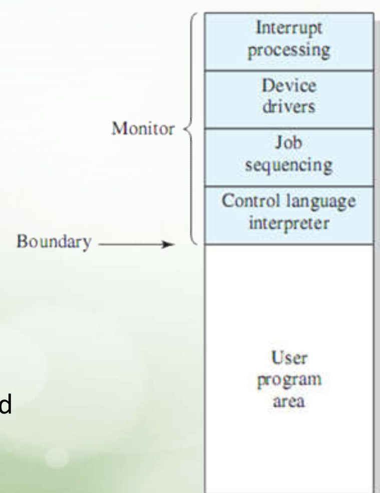
- **Problems**
- Scheduling
  - Most installations used a hardcopy **sign-up sheet** to reserve computer time
  - Time allocations could run short or long, resulting in wasted computer time
- Setup time
  - A considerable amount of time was spent just on setting up the program to run

Simple Batch Systems

- User no longer has direct access to processor
- A **user** submitted job to an operator (on plunge cards or tapes)
- For jobs submitted, the **operator** batched them together and placed a **batch of jobs** on an input device
- A resident program "***monitor***" in main memory managed the executions of the jobs
- Program branches back to the **monitor** when finished

Simple Batch Systems

- Users no longer had direct access to processor
- A **user** submitted job to an operator (on plunge cards or tapes)
- For jobs submitted, the **operator** batched them together and placed a **batch of jobs** on an input device
- A resident program "***monitor***" in main memory
  - Managed executions of the jobs
  - Improve overall system performance
  - Controls sequence of events: by replacing an executed job in user program area with a new job
- Program branches back to the **monitor** when finished

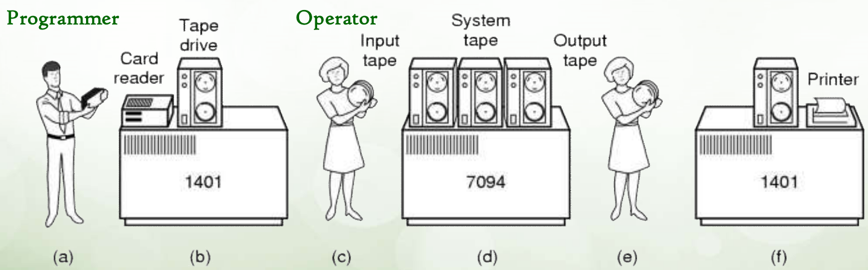


Memory Layout for a Resident Monitor

Simple Batch Systems: Operations

1. **Bring cards to 1401**
2. **Read cards to tape**
3. **Put tape on 7094 which does computing**
4. **Put tape on 1401 which prints output**

Programming language on plunge cards for simple operations and computations



15

A Start : Use Some Special Language

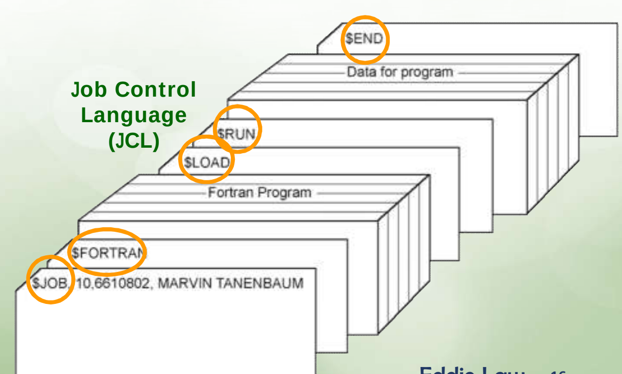
- E.g.: Job Control Language (JCL)

Special type of language used to provide instructions to the "*monitor*"

What compiler to use?

What data to use?

- E.g.: Structure of FMS (Fortran Monitor System)



Eddie Law 16

Processor's Point of View

- Processor executed instructions from the memory containing the monitor
- Executed the instructions in user program until encountering an ending or error condition
- "**Control is passed to a job**" meant processor fetched and executed instructions in a user program
- "**Control is returned to the monitor**" meant that the processor fetched and executed instructions from the monitor program

Eddie Law 17

Time to Advance Technology Again

- The next generation (3rd)
  - Introductions of ICs and multiprogramming
  - Cost model was still in transition
- Improvements made on
  - Batch operating system
  - Alternated executions among user programs and monitor program
- Introduction of the ***multi-programming*** system concept

Eddie Law 18

Multi-Programmed Batch Systems

- For low CPU utilization, e.g.
- **Issues were…**
- I/O operations were slow
  - Poor CPU utilization if only one program is running
- Memory may hold several programs
  - CPU executes another program when a program is waiting for an I/O completion
- Need good supports on hardware and software
  - E.g., interrupts, timers, memory protection

Read one record from file	15 μs
Execute 100 instructions	1 μs
Write one record to file	15 μs
TOTAL	31 μs

$$\text{Percent CPU Utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

To Deal with the Issues…

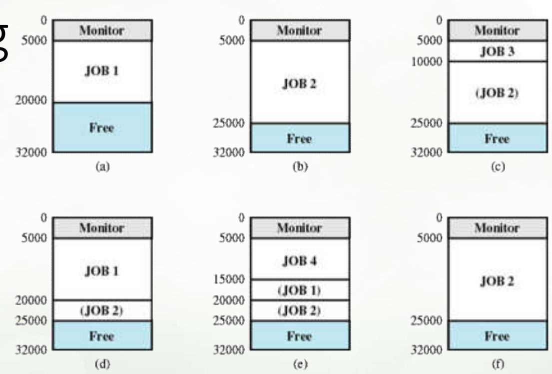
- How to split jobs?
- How to put multiple jobs in memory?
- In other words, how to share hardware resources?
  - Concept of virtual machines

The Time Sharing Systems (TSS)

- Problem with batch systems
  - No user interactions
- Reaction time of human beings is pretty slow
  - Typically, a user needs 2 sec/min processing time
- Time-sharing could then be a possible choice
  - Need better supports on hardware and software for this feature
  - E.g., file system protections, scheduling, concept of ***processes***

Compatible Time-Sharing System

- CTSS at MIT
  - Circa 1963
  - ~32,000 36-bit words storage
  - Multiprogramming: handled multiple interactive jobs
  - Users shared processor's time
  - Users simultaneously accessed the system through terminals



Case 1: Operations

Case 2: Job Nature

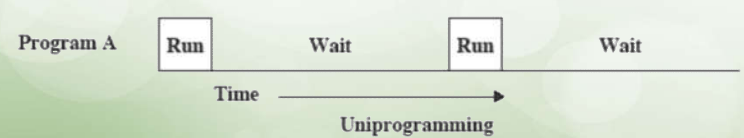
	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

Batch Multiprogramming vs. Time Sharing

	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

Concept of Multiprogramming

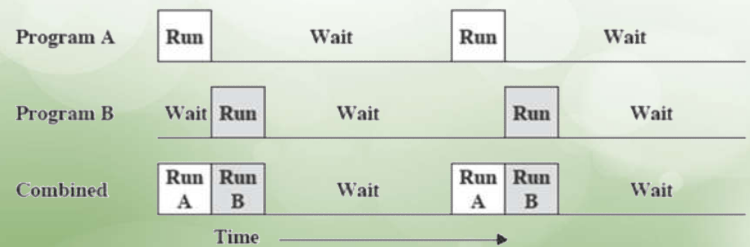
- Uniprogramming
  - Only one process occupies all resources
  - If there is I/O access, CPU must wait for I/O instruction to complete before preceding
  - i.e., resources not well used



Multiprogramming

- Multitasking??
- Running more than one process at the same time
- When one job needs to wait for I/O, the processor can switch to the other job

IS IT REALLY TIME SHARING BASED ON THIS STATEMENT??



Multiprogramming with two programs

Eddie Law 25

The Modern Computing Systems

- Since the 4th generation
- **Scenario 2**: cheap hardware, high labour cost ⇒ make hardware easier to use
- “**Operating Systems**” facilitate resource management and allocation

Eddie Law 26

Operating Systems

- The first piece of software to run upon booting up a computer
- Coordinate executions of all other software, mainly user applications
- Provide various common services needed by users and applications
- An interface between applications and hardware

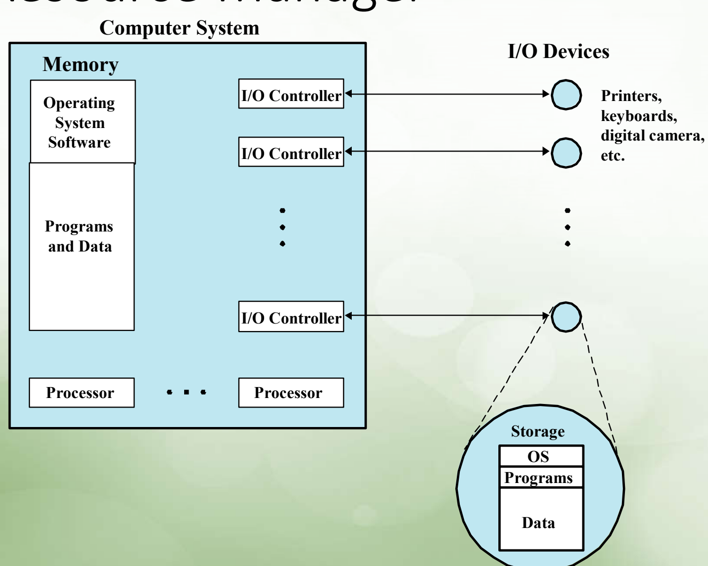
The Role of an OS

- A computer is a set of resources for the movement, storage, and processing of data
- The OS is responsible for managing these resources

Operating System as a Software

- OS functions in the same way as an ordinary computer software
- A software program executed by the CPU
- Application accesses CPU through allocation by the OS; it relinquishes control of the processor

OS as a Resource Manager



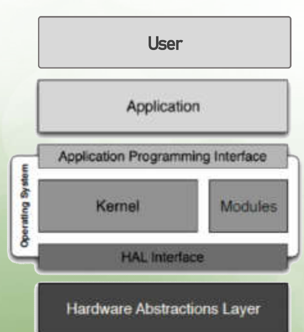
Modes of Operations

- User Mode
  - User program executes in user mode
  - Certain areas of memory protected from user access
  - Certain instructions may not be executed
- Kernel Mode
  - Privileged instructions (e.g. I/O instructions) may be executed, all memory accessible

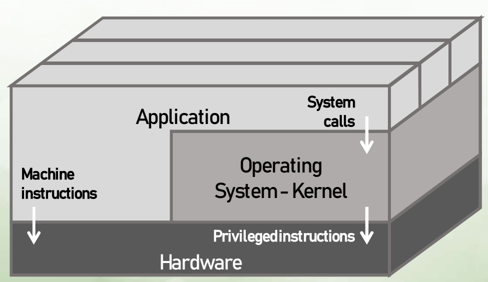
Eddie Law 31

Visualize an OS: Controlling a Machine

- A 2D picture



- A 3D perspective

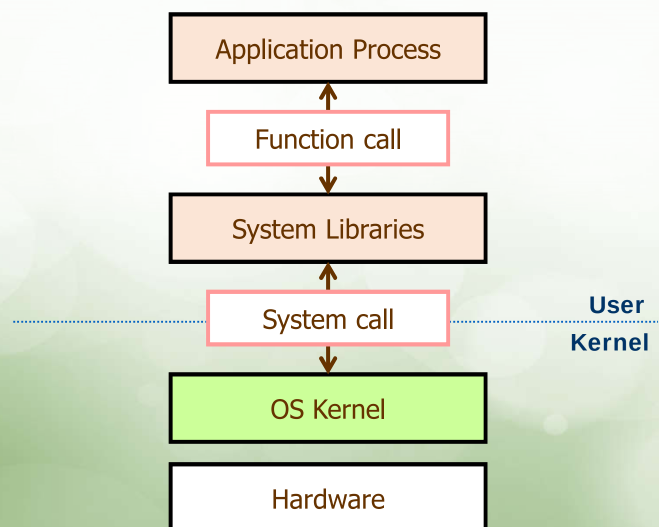


Eddie Law 32

Functions

- From the 3D picture
  - OS schedules applications (i.e., processes) to run
  - When a process runs, OS is not involved, e.g.,
    - 3 + 6 = 9
  - Processes sharing hardware, e.g.,
    - Clock interrupts
    - Context switching
  - A process may run on hardware, and
  - A process may need the OS to do some special things through, e.g., make a software trap for I/O operations

About the System Calls: A Typical View



On Interacting with Hardware

- OS enjoys a privileged execution (kernel) mode
  - Indicated by **a bit** in the special register, the ***processor status word*** (PSW)
- Which are of the followings should be in the privileged mode?
  - Change the program counter
  - Halt the machine
  - Divide by zero
  - Change the execution mode

Eddie Law 35

Hardware Support: Examples

- System calls explicitly ask for the operating system
  - Only predefined operations can run in kernel mode
- Interrupts cause a switch to kernel mode
  - Asynchronous interrupts
    - Clock and I/O
  - Internal (synchronous) interrupts
    - Error condition, and temporary problem (page fault)
  - Software traps
- Hardware has special features to help the operating system
  - Test-and-set
  - Access/used bit on memory page

More on these later

Eddie Law 36

Services Provided by the OS

- Program development
  - Editors and debuggers
- Program execution
  - OS handles scheduling of numerous tasks required to execute a program
- Access I/O devices
  - Each device will have unique interface
  - OS presents standard interface to users

Services Provided by the OS (cont’d)

- Controlled access to files
  - Accessing different media but presenting a common interface to users
  - Provides protection in multi-access systems
- System access
  - Controls access to the system and its resources

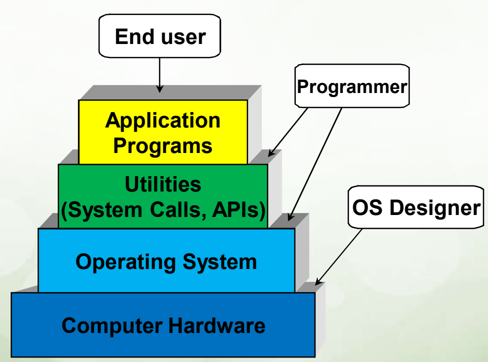
Services Provided by the OS (cont'd)

- Error detection and response
  - Internal and external hardware errors
  - Software errors
  - Operating system cannot grant request of application
- Accounting
  - Collect usage statistics
  - Monitor performance

Eddie Law 39

Three Objectives on Designing OS

- Convenient user/computer interface
  - Provide services to various users
  - Mask details of the hardware
- Efficient resource management
  - Resources: CPU, memory, I/O devices ...
- Ability to evolve



Eddie Law 40

Introduction: Modern OS Designs

- Microkernel architecture
  - Assigns only a few essential functions to the kernel
    - Address spaces
    - Inter-process communication (IPC)
    - Basic scheduling
- Object-oriented design
  - By adding modular extensions to a small kernel
  - Enables programmers to customize an operating system without disrupting system integrity

Eddie Law 41

Introduction: Modern OS Designs

- Multithreading
  - Process is divided into threads that can run concurrently
    - Thread
      - A dispatchable unit of work
      - Executes sequentially and is interruptible
    - Process is a collection of one or more threads

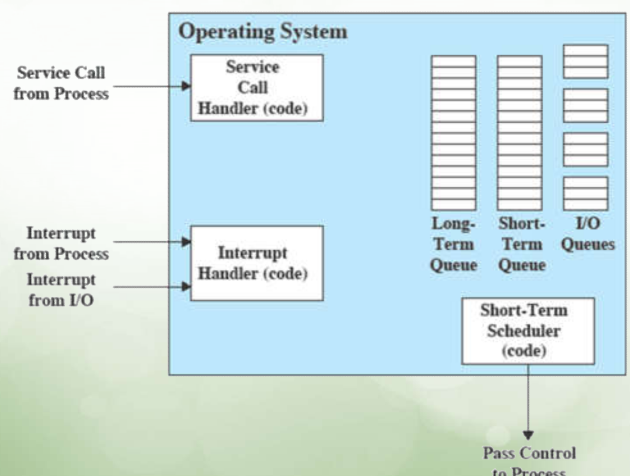
Eddie Law

Modern OS Designs

- Multiprocessing (multi-core system) and multiprogramming

Eddie Law 43

Operating System: Typical Key Elements

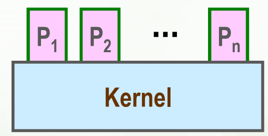


Key Elements of an Operating System for Multiprogramming

44

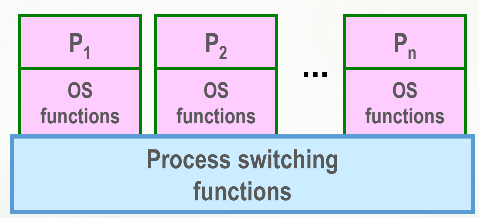
Designs of Kernels

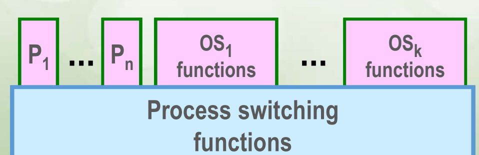
- Chapter 3.5
- Based on where to run operating system functions
- The trivial design
  - Monolithic kernel
    - Also known as non-process kernel
    - Kernel does everything …, let processes run independently on hardware
    - Kernel code is not in any processes
    - Code is executed as a separate entity in privileged mode



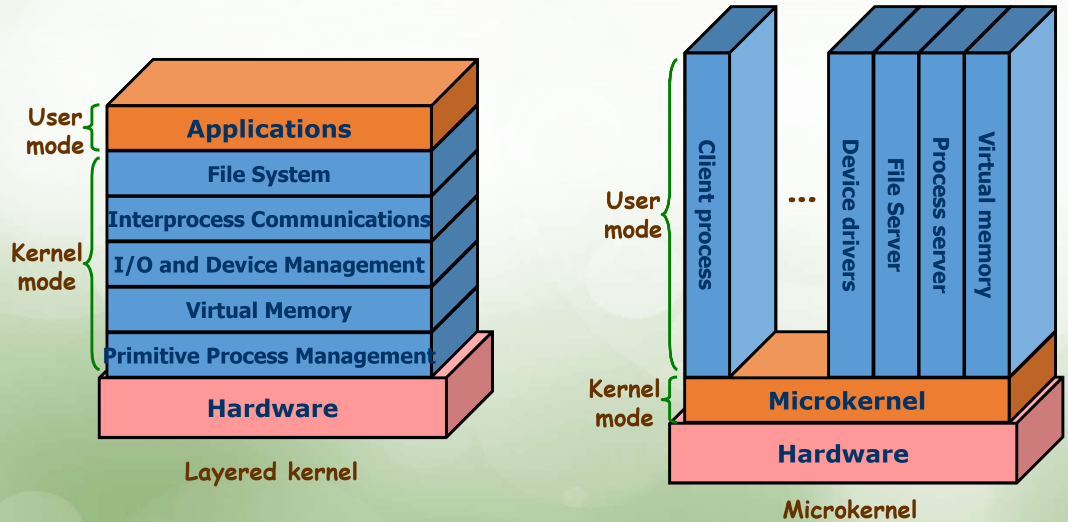
Designs of Kernels (cont'd)

- OS functions execute as user processes
  - OS within context of a user process
  - Process executes in privileged mode when executing operating system code
- OS functions execute as separate processes
  - Implement OS as a collection of system processes
  - Useful in multi-processor or multi-computer environment





Kernel Architectures: Examples



47

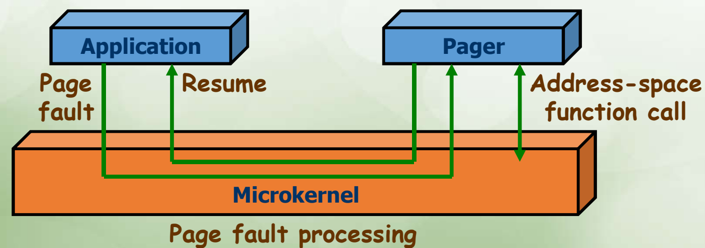
Case Study: Microkernels

- Chapter 4.3
- Small operating system core
- Contains only essential core operating systems functions
- Many services traditionally included in the operating system are now external subsystems
  - Device drivers
  - File systems
  - Virtual memory manager
  - Windowing system
  - Security services

Eddie Law 48

Microkernel Design

- Example: Low-level memory management
  - Mapping each virtual page to a physical page frame
- Inter-process communication
- I/O and interrupt management



Benefits of a Microkernel

- Uniform interface on request made by a process
  - Don't distinguish between kernel-level and user-level services
  - All services are provided by means of message passing
- Extensibility
  - Allows the addition of new services
- Flexibility
  - New features added
  - Existing features can be subtracted
- Portability
  - Changes needed to port to a new processor is only in the microkernel - not in the other services

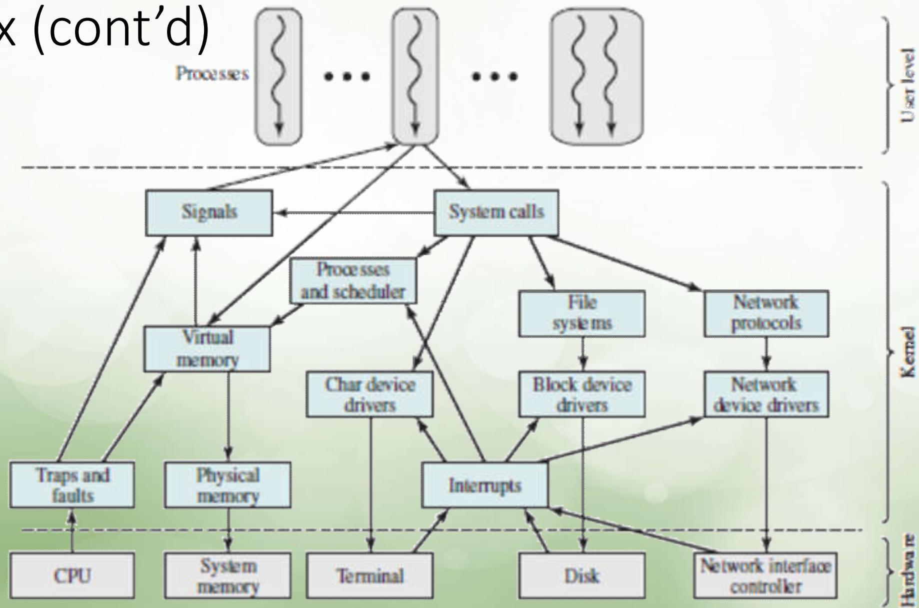
Benefits of a Microkernel (cont'd)

- Reliability
  - Modular design
  - Small microkernel can be rigorously tested
- Distributed system support
  - Message are sent without knowing what the target machine is
- Object-oriented operating system
  - Components are objects with clearly defined interfaces that can be interconnected to form software

Case Study: Linux

- UNIX-variant operating system
- Open source on GNU Public License (GPL)
  - www.gnu.org
- ***Monolithic kernel***, non-modular architecture design
  - Device drivers can be loaded with modules though
- But has concept of loadable modules to test, or load
  - File system, device driver, etc
  - Dynamic linking
  - Stackable modules

Linux (cont'd)



Conclusion

- Operating systems
  - Provide a virtual machine abstraction to handle diverse hardware
  - Coordinate resources and protect users from each other
  - Simplify application development by providing standard services
  - Can provide an array of fault containment, fault tolerance, and fault recovery
- Operating systems associate
  - Programming languages, data structures, hardware, and algorithms

Next Topic

- Process Description and Control
- Read Ch. 3