CMPS 405 (Operating Systems)

Week 1

Evolution Operating Systems, OS internals, Operating System Structures, System Commands, System Calls, and I/O systems & Interrupts handling

CSE-CENG-QU

**This material is based on the operating system books by Silberschatz and Stallings*

CMPS 405 (Operating Systems)

What are you going to learn in this course?

Topic
✓ Operating Systems theory and related algorithms
✓ Mastering Linux Operating System
✓ Batch scripting for Linux
✓ System programming with system calls and the C Library functions
✓ Simulations of CPU scheduling algorithms and other OS related algorithms
✓ Synchronization and Concurrency programming with applications in C and Java
✓ Network programming and application protocols development with Sockets in C and Java
✓ Undertake a project/homework devoted for advanced topics

Contents:

- Motivations
- OS Services
- OS SW Design Structures
- OS Components
- The Command Line Interpreter
- System Commands
- Batch/Bash Scripts
- System Calls
- Process Concepts

What is an Operating System?

- Conclude it from these points:
  - A program that acts as an intermediary between a user of a computer and the computer hardware.
  - Resource allocator: manages and allocates resources.
  - Control program: controls the execution of user programs and operations of I/O devices .
  - Kernel: the one program running at all times (all else being application programs).

- Operating system goals:
  - Execute user programs and make solving user problems easier.
  - Make the computer system convenient to use.
  - Use the computer hardware in an efficient manner.

Operating Systems' Services

- User Interface
- Program Execution
- I/O Operations
- File-System Manipulation
- Communications
- Error Detection
- Resource Allocation
- Accounting
- Protection and Security

Implementations of OS

- Old OS implemented in assembly while currently implemented in higher-level languages.
  - Example: Linux and Win XP are written mostly in C with some inserted assembly code for device drivers, registers and context switching.
  - Advantages:
    - Fast implementation; compact; easier to understand and debug; easier to port (move to other hardware).
    - Better data structures and algorithms resulted in improving the performance of OSs.
  - Disadvantages: reduce speed and increase storage requirements.

- System performance must be monitored by:
  - Producing trace listing of system behavior (a file) to be used by analysis program to determine performance and find bottleneck routines and faults.
    - The tracing file can also be used by a simulation of a suggested improved system.
  - Compute and display the performance measures in real time.

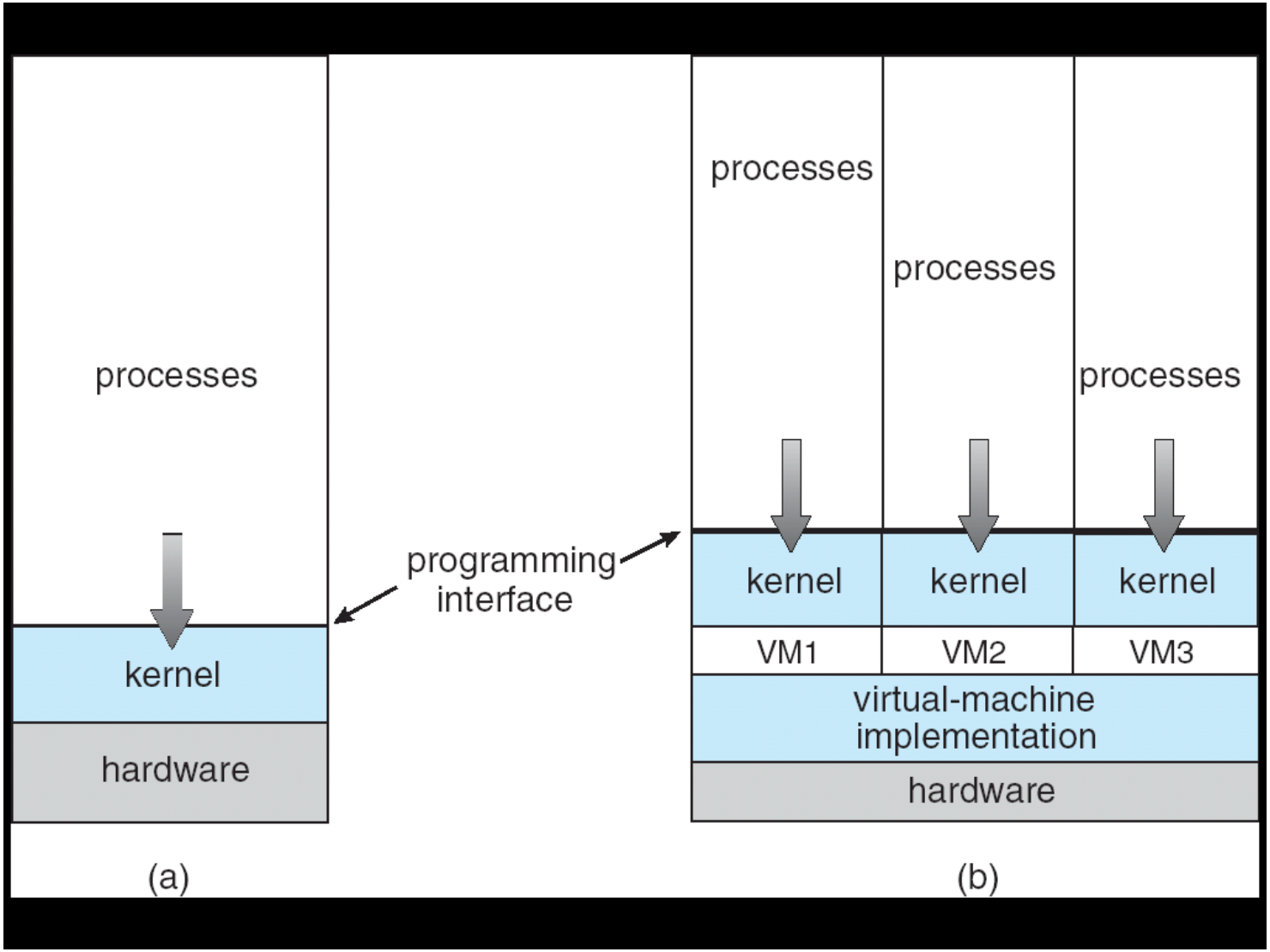
OS System Generation (SYSGEN)

- It is the process of configuring and generating operating system to run on a specific machine. Implemented by a SYSGEN program that:
  - reads from a given file, or
  - asks the operator of the system for information concerning the specific configuration of the hardware system, or
  - probes the hardware directly to determine what components are there including information on CPU, memory, available devices and OS desired options.

Virtual Machines

- A VM is an implementation of an exact (relatively) duplicate of the underlying machine creating an illusion that every process has its own processor and memory. This done by using CPU scheduling and virtual memory techniques.
  - Each process is provided with a virtual copy of the underlying computer.
  - In a VM, disk systems are provided as virtual disks (minidisks that are identical in all respects except size)
- Security and protection:
  - VM provides virtual user mode and virtual monitor mode both running in physical user mode.
  - VMs are completely isolated form each other.

Virtual Machines



Virtual Machines

- Difficulties: Providing an exact duplicate of the underlying machine is a difficult task.
- Benefits:
  - Perfect vehicle for OS research and Development (R&D): System development is done on VMs instead of real machines. As a result eliminating System-development time problem.
  - Brings different machines closer:
    - Running Intel MS windows applications on Sun Microsystems processors (e.g. WABI program).
    - Running Windows applications on Linux-based computers using a VM.
- Example: JVM implementing specifications of an abstract computer. It consists of a class loader and Java interpreter to execute architecture-neutral bytecodes.

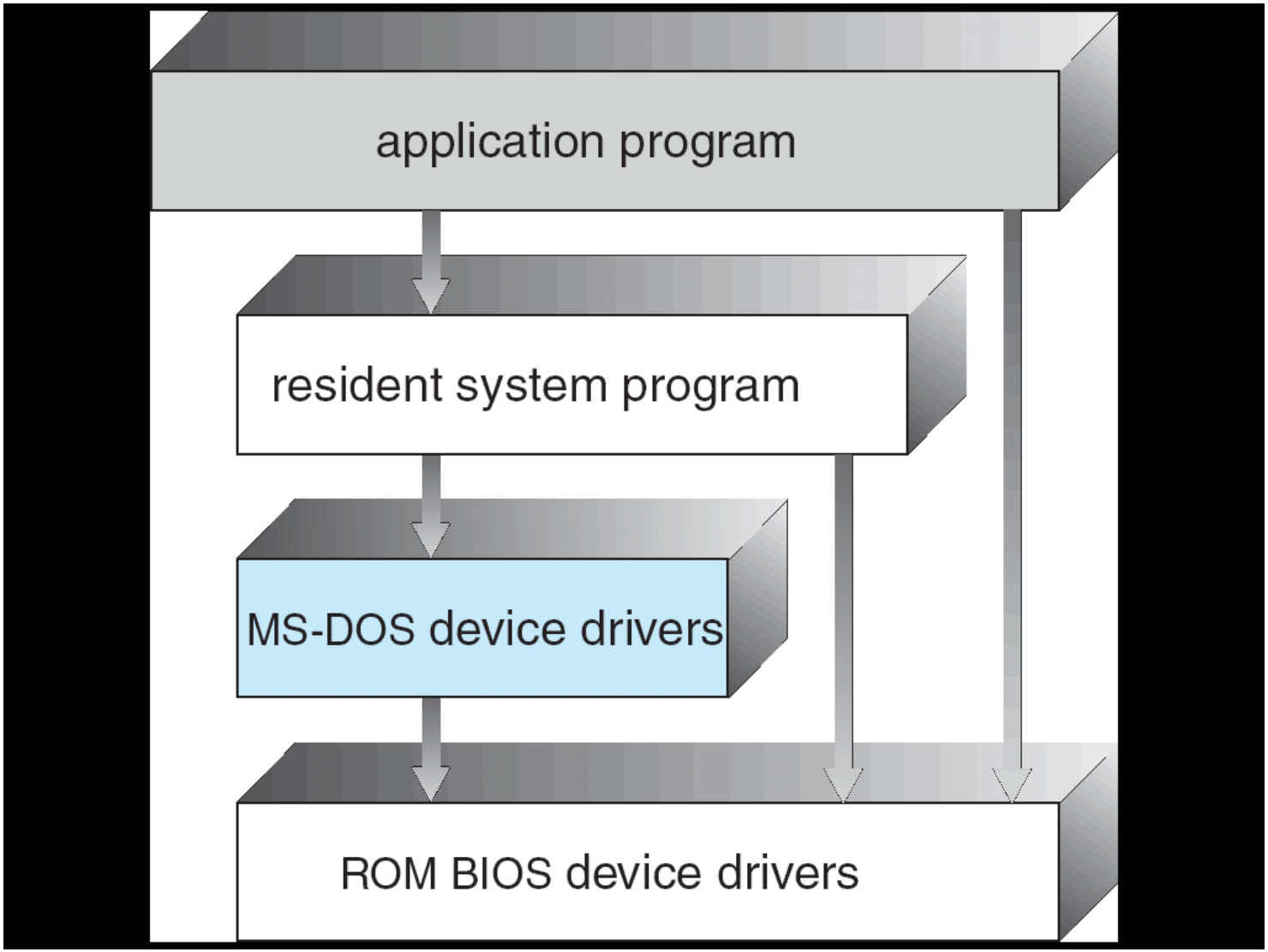
OS Structures

- Simple Structure
- Layered Approach
- Microkernel
- Modules

OS Structures: Simple Structure

- Provides the most functionality in the least space: limited hardware it runs on.
- Does not have well-defined structure: interfaces and levels are not well separated
- The base hardware is accessible by user because the initial hardware does not provide dual mode or hardware protection.
  - The entire system crashes when user programs fail.
  - Such systems are difficult to implement and maintain.

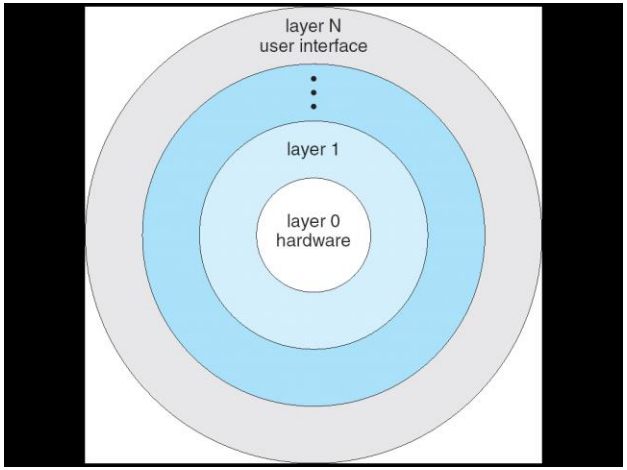
Simple Structure

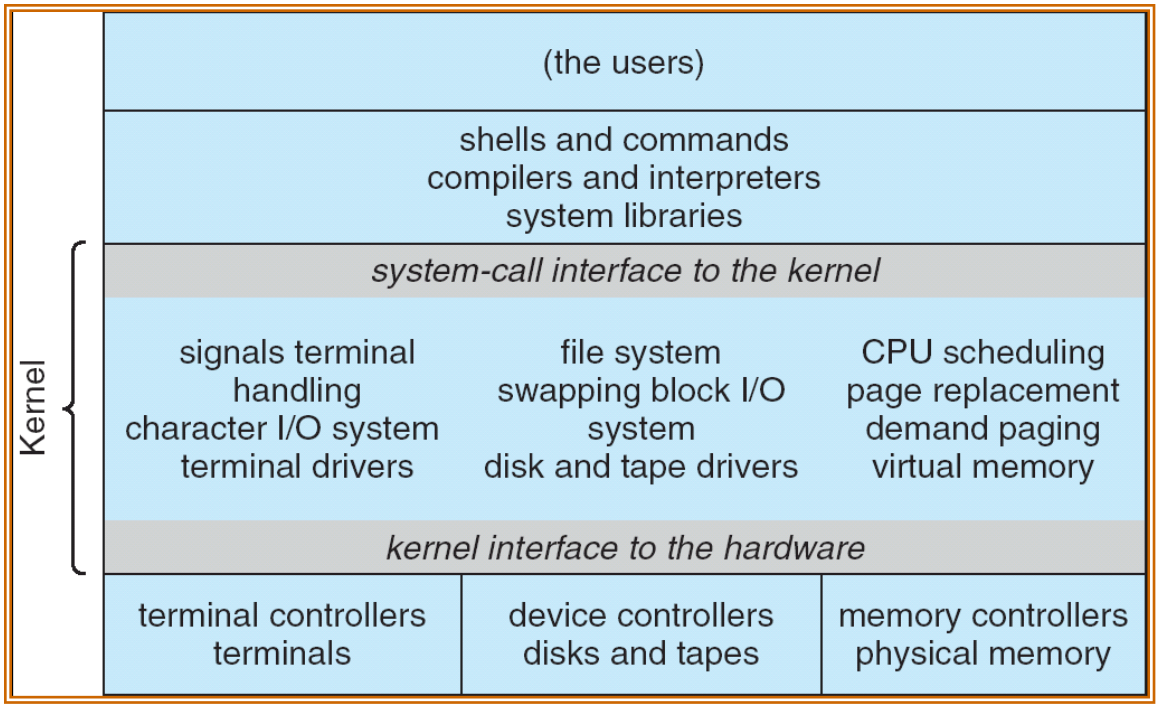


OS Structures: Layered Approach

- An OS layer is an implementation of an abstract object made up of data and their operations.
- In a layered system, layer m data and routines can be invoked by layer m+1 on the other hand layer m can invoke layer m-1 data and routines
- Example: OS/2 which is a descendant of MS-DOS that adds multitasking and dual-mode operations as well as other features.

OS Structures: Layered Approach





OS Structures: Layered Approach

- Advantages:
  - Modularity: resulted from system divided into layered with controlled interaction.
  - Simplification of debugging and system verification.
  - Direct user access to low-level facilities is not allowed

- Disadvantages:
  - Defining appropriate layers data and operations.
  - Less efficient than other types because of the nature of layered system requiring requests to be mapped into other requests down into layers.

OS Structures: Microkernel

- This method structures the operating system by moving all nonessential components from the kernel and implementing them as system and user-level programs.

- The kernel is relatively small (microkernel) providing minimal process and memory management as well as a communication facility using message passing.

OS Structures: Microkernel

- Advantages:
  - Ease of extending the operating system without modifications to the kernel.
  - Easier to port from one hardware design to another.
  - More secured and reliable: services run in user mode, any failure will not disturb the kernel.

- Disadvantages:
  - Decrease of performance because of increased system overheads resulted from mapping system calls into messages to the appropriate user-level services for example.

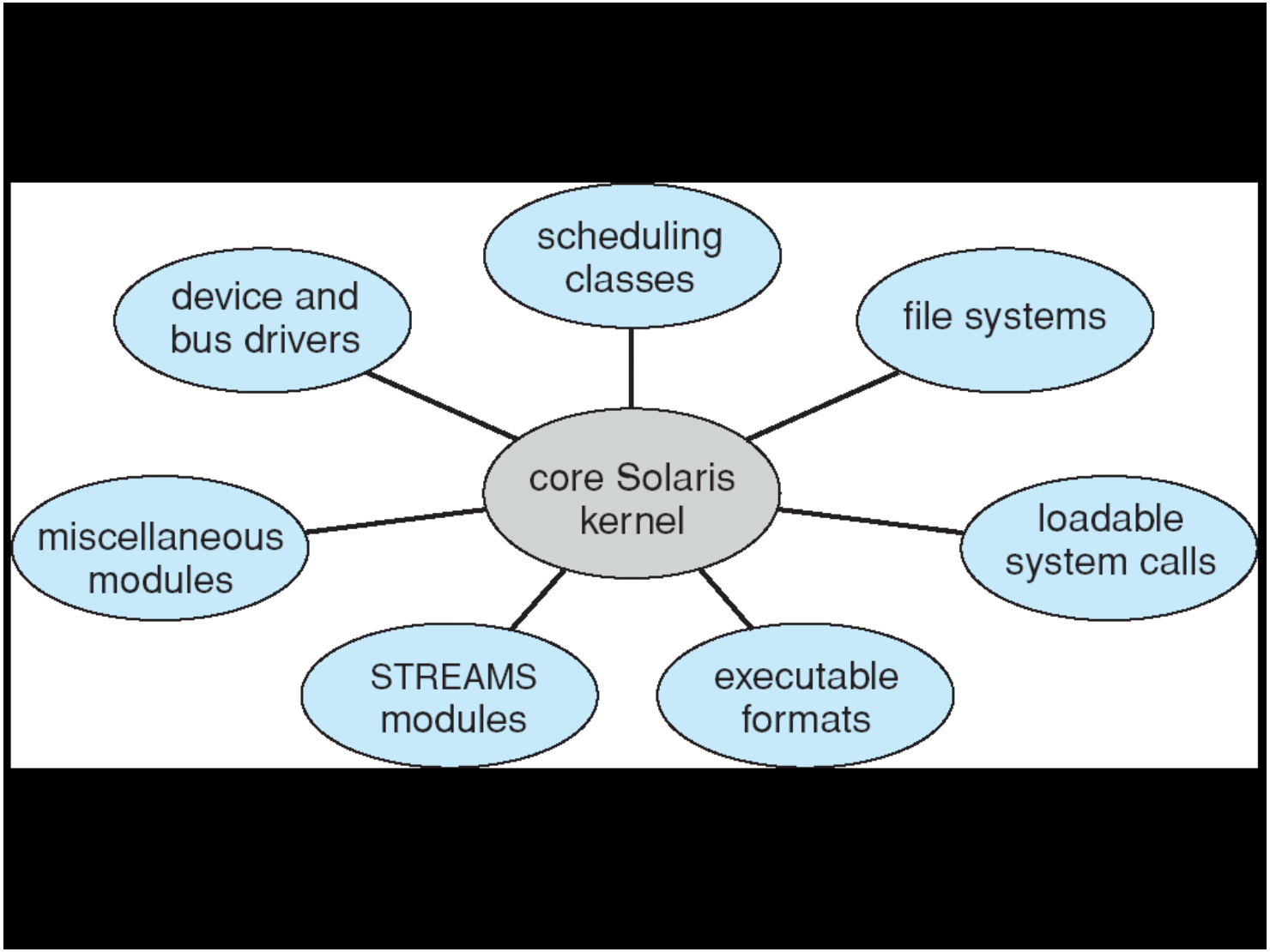
OS Structures: Modules

- Modules that are dynamically linked each implements certain features of the system.

- Advantages include: allowing certain features to be implemented dynamically; modules have protected interfaces; more efficient and required no message passing (i.e. modules have direct communications).

- Examples: UNIX, Solaris, Linux and Mac OS X.

Modules Structure

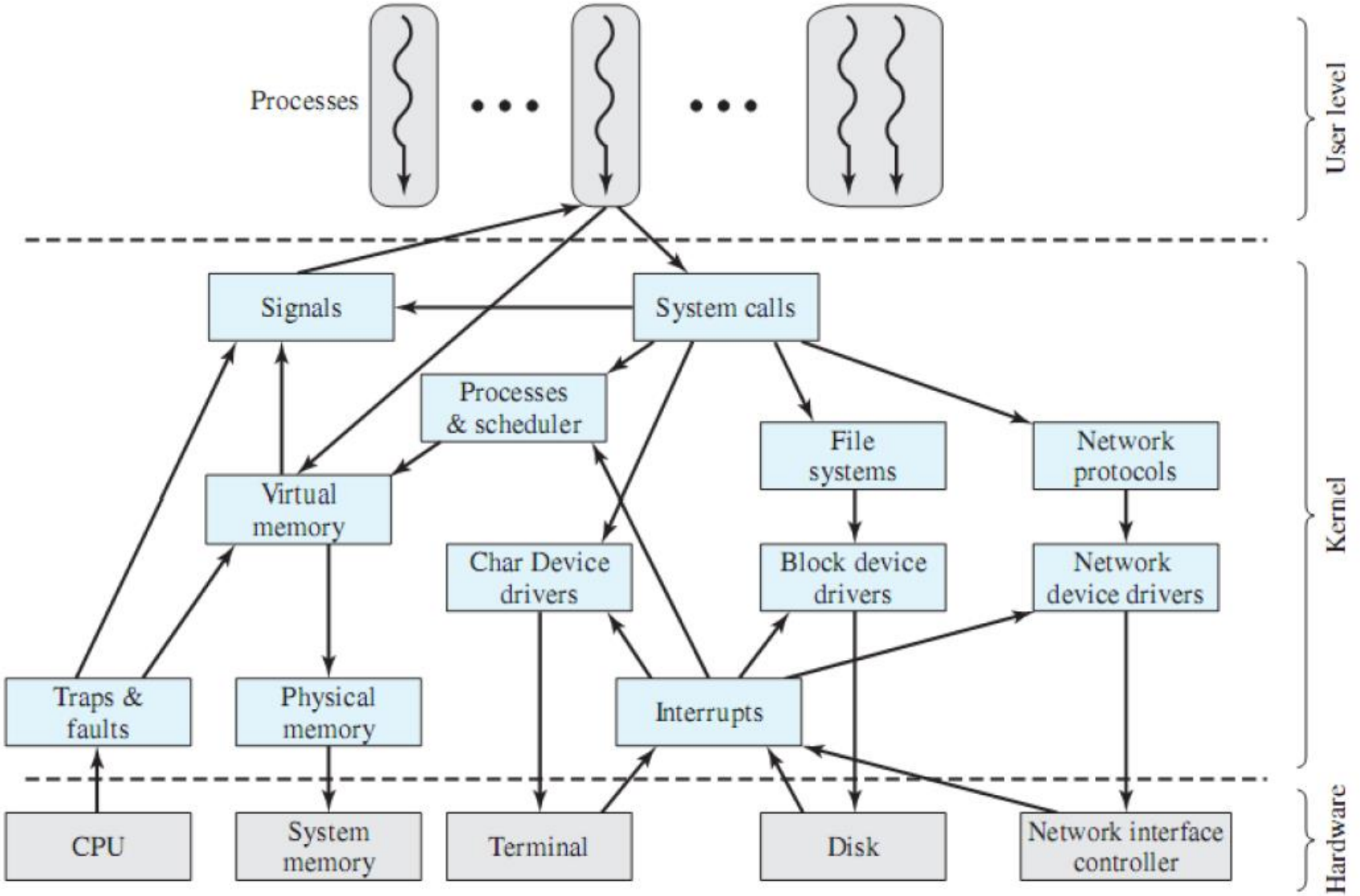


Operating systems design hierarchy

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

Gray shaded area represents hardware.

Linux Kernel Components



System calls

- System calls allow user-level processes to request services of the operating system.
- Generally available as assembly-language instructions.
- Some Languages defined to replace assembly language for systems programming allow system calls to be made directly (e.g., C, C++).
  - high-level languages support system calls by hidden run-time support system (the set of functions built into libraries with a compiler) provided by the operating system.
- System programs can be thought of as bundles of useful system calls. They provide basic functionality to users and so users do not need to write their own programs to solve common problems.

System Calls

- System call parameters are passed to the operating system in three ways:
  - Pass parameters to registers. Only works when number of parameters is less or equal to number of available registers.
  - Parameters are stored in a block or table in memory then the address of that block is passed to a register. (Linux)
  - Parameters are pushed onto the stack and popped by the operating system.

Categories of system calls

- Process control:
  - create, load, execute, abort, wait, get and set attributes and terminate.
- File management:
  - create, open, read, write, get and set attributes, close and delete.
- Device management:
  - request, read, write, get and set attributes and release.
- Communications:
  - create, send, receive, status and delete.
- Information maintenance:
  - get and set time and date, get and set system info, get and set process, file and device info.

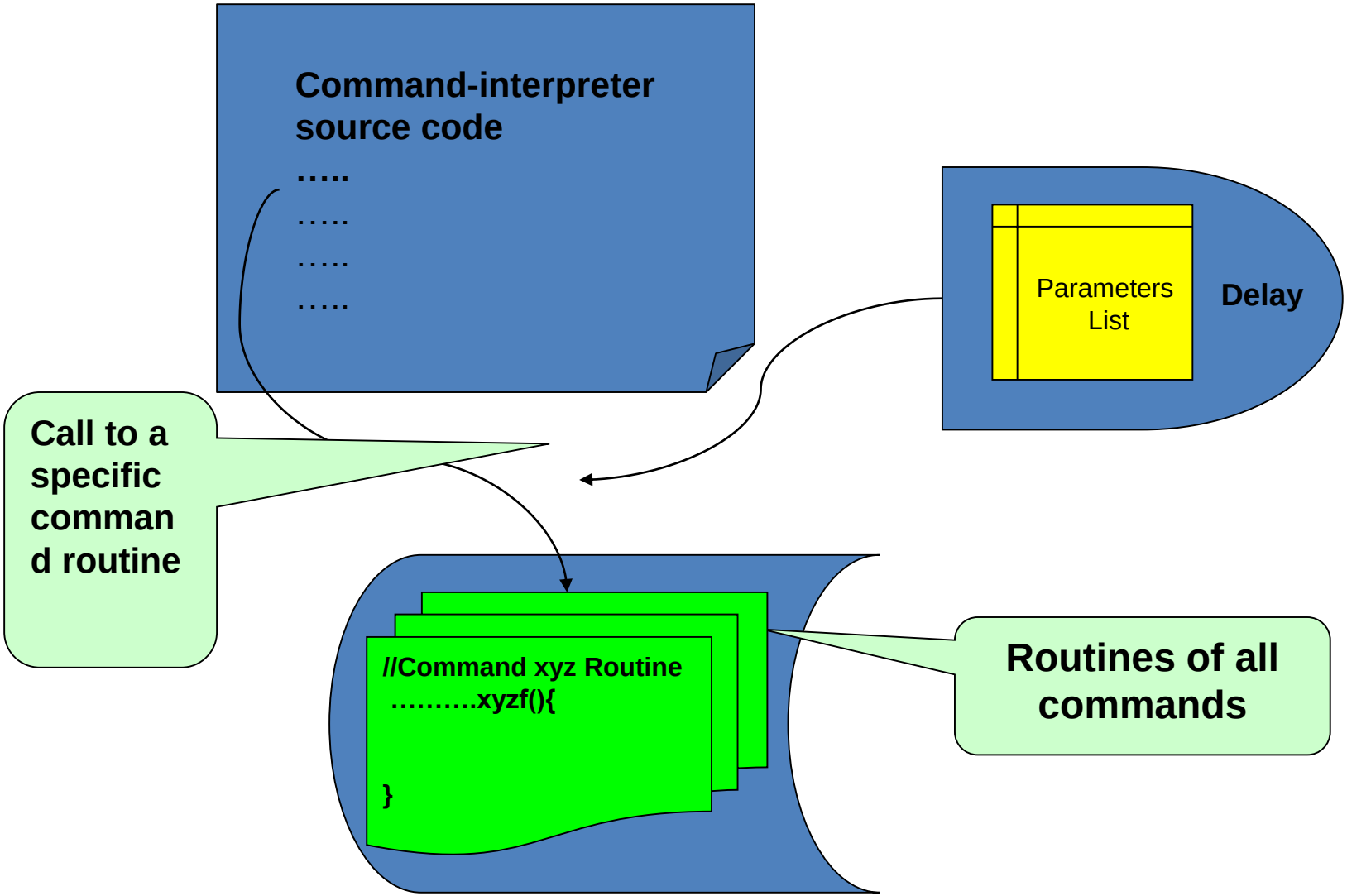
Sample of System call

GENERAL CLASS	SPECIFIC CLASS	SYSTEM CALL
Process Related	Process Creation & Termination	fork()
		exec()
		wait()
		exit()
	Process Identity	getpid()
		getppid()
	Process Control	signal()
		kill()
		alarm()
Interprocess Communication	Pipelines	pipe()
	Messages	msgget()
		msgsnd()
		msgrcv()
		msgctl()
	Semaphores	semget()
		semop()
	Shared Memory	shmget()
		shmat()
		shmdt()

Command-interpreter

- The Command-interpreter reads commands from the user or from a file of commands and executes them, usually by turning them into one or more system calls.
- Most commands are implemented through system programs (UNIX). The command interpreter uses the command to identify a file to be loaded into memory and execute it.
- It is usually not part of the kernel since the command interpreter is subject to changes.

Implementations of Command-interpreter



Windows batch file programming

- Batch file programming is used to perform tasks from the command line. The script is run as an alternative to running each command by hand.
- How to:
  - A batch file is actually a text file with DOS commands in it.
  - Each DOS command will be on a separate line.
  - Then, instead of a .txt extension, it has the extension of .bat.
  - You can double click the batch file or type its name at the DOS prompt to execute the commands.
  - To create a batch file in Windows, you need to know the DOS commands.
  - Scripts can control most programs and system which is important for corporate admins who usually use a GUI tool and scripting.

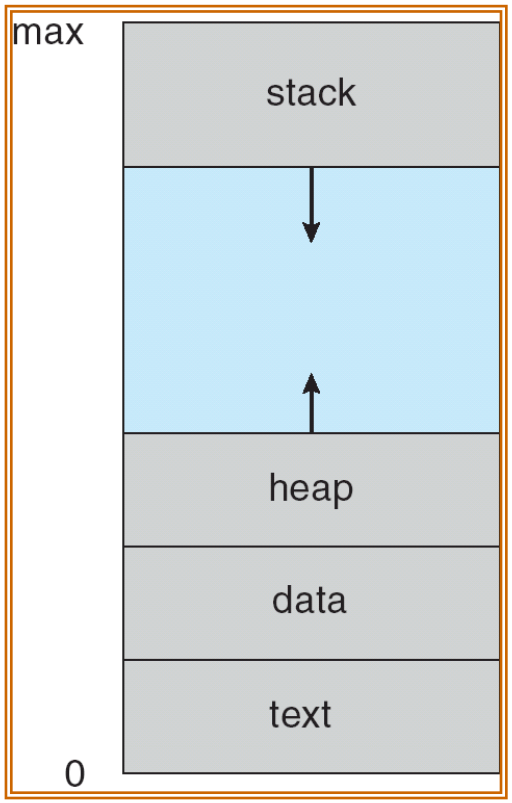
Linux Bash Shell Script programming

- How to:
  - A batch/pash file is actually a text file with commands in it.
  - Each command will be on a separate line or separated by ;
  - All bash scripts must tell the OS what to use as the interpreter. The first line of any script should be #!/bin/bash
  - Save it with the extension of .sh
  - Change the mode of the file to be executable chmod +x mybash.sh
  - You can double click the batch file or execute it at the command prompt by typing ./testbash.sh
  - The script file also can not to have the extension .sh

Process Concept

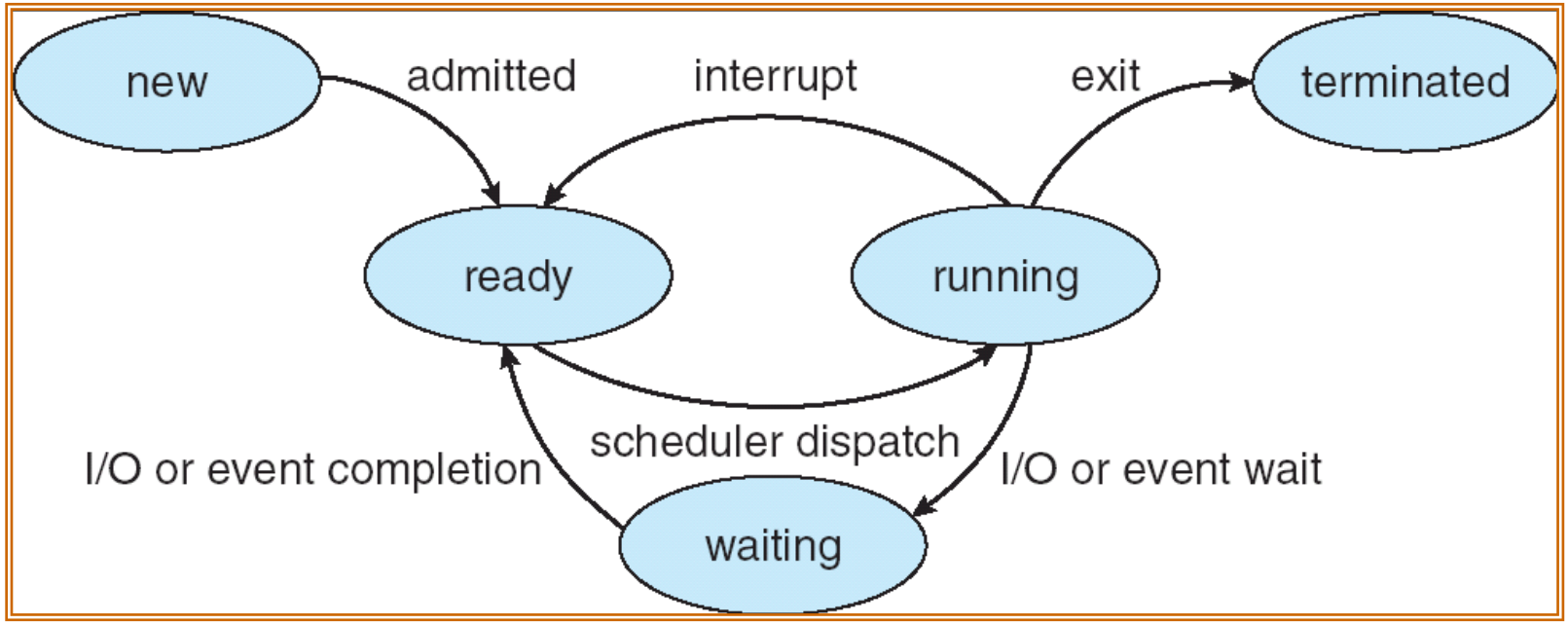
- **Process** – a program in execution; process execution must progress in sequential fashion

- A process is represented by:
  - Text section (program code)
  - Current activity represented program counter and the contents of the CPU registers
  - Stack containing temporary data such as local variables and methods parameters.
  - Data section containing global variables
  - heap for dynamic memory allocation



Process State

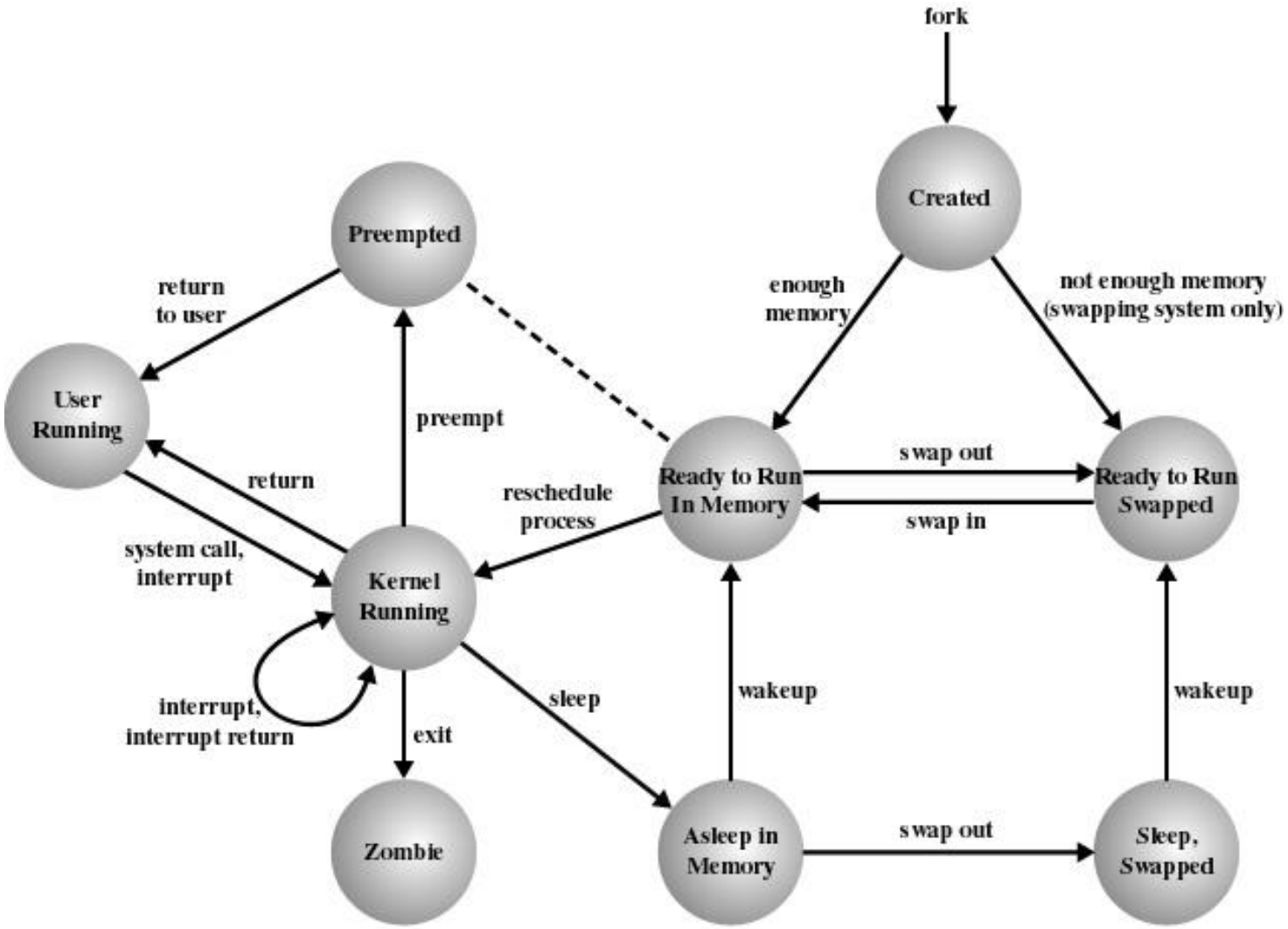
- As a process executes, it changes *state*
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a process
  - **terminated**: The process has finished execution



Process states in UNIX

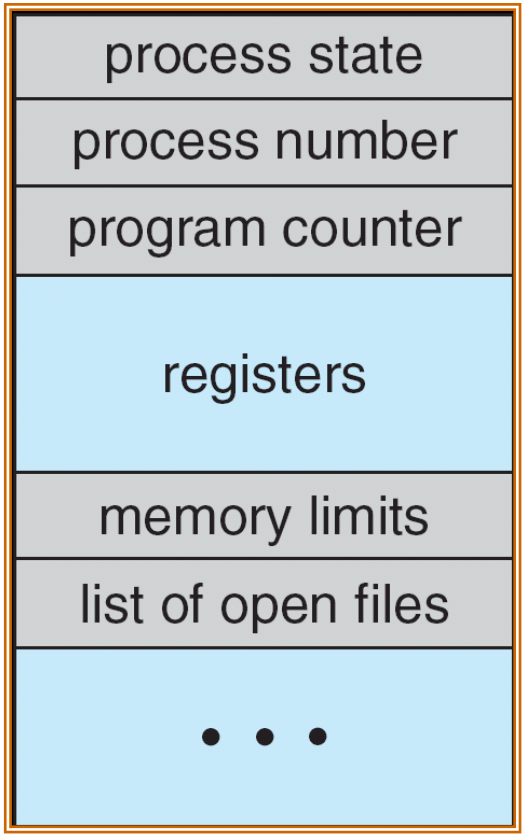
User Running	Executing in user mode.
Kernel Running	Executing in kernel mode.
Ready to Run, in Memory	Ready to run as soon as the kernel schedules it.
Asleep in Memory	Unable to execute until an event occurs; process is in main memory (a blocked state).
Ready to Run, Swapped	Process is ready to run, but the swapper must swap the process into main memory before the kernel can schedule it to execute.
Sleeping, Swapped	The process is awaiting an event and has been swapped to secondary storage (a blocked state).
Preempted	Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.
Created	Process is newly created and not yet ready to run.
Zombie	Process no longer exists, but it leaves a record for its parent process to collect.

Process states in UNIX: Transition Diagram

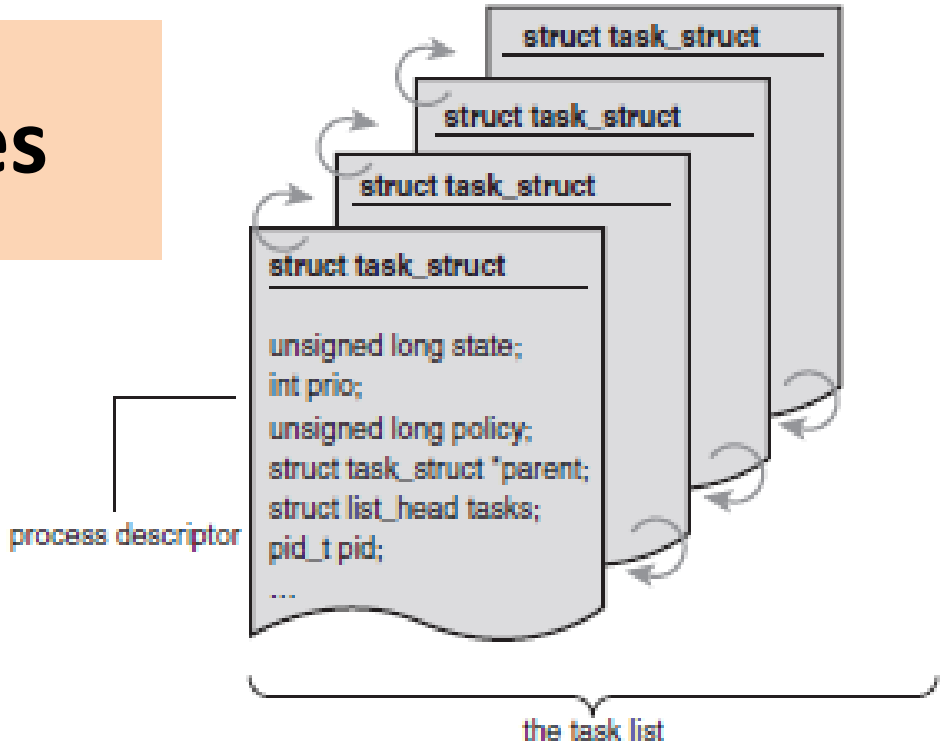


Process Control Block (PCB)

- A data structure containing information associated with each process
  - Process state
  - Program counter
  - CPU registers
  - CPU scheduling information
  - Memory-management information
  - Accounting information
  - I/O status information

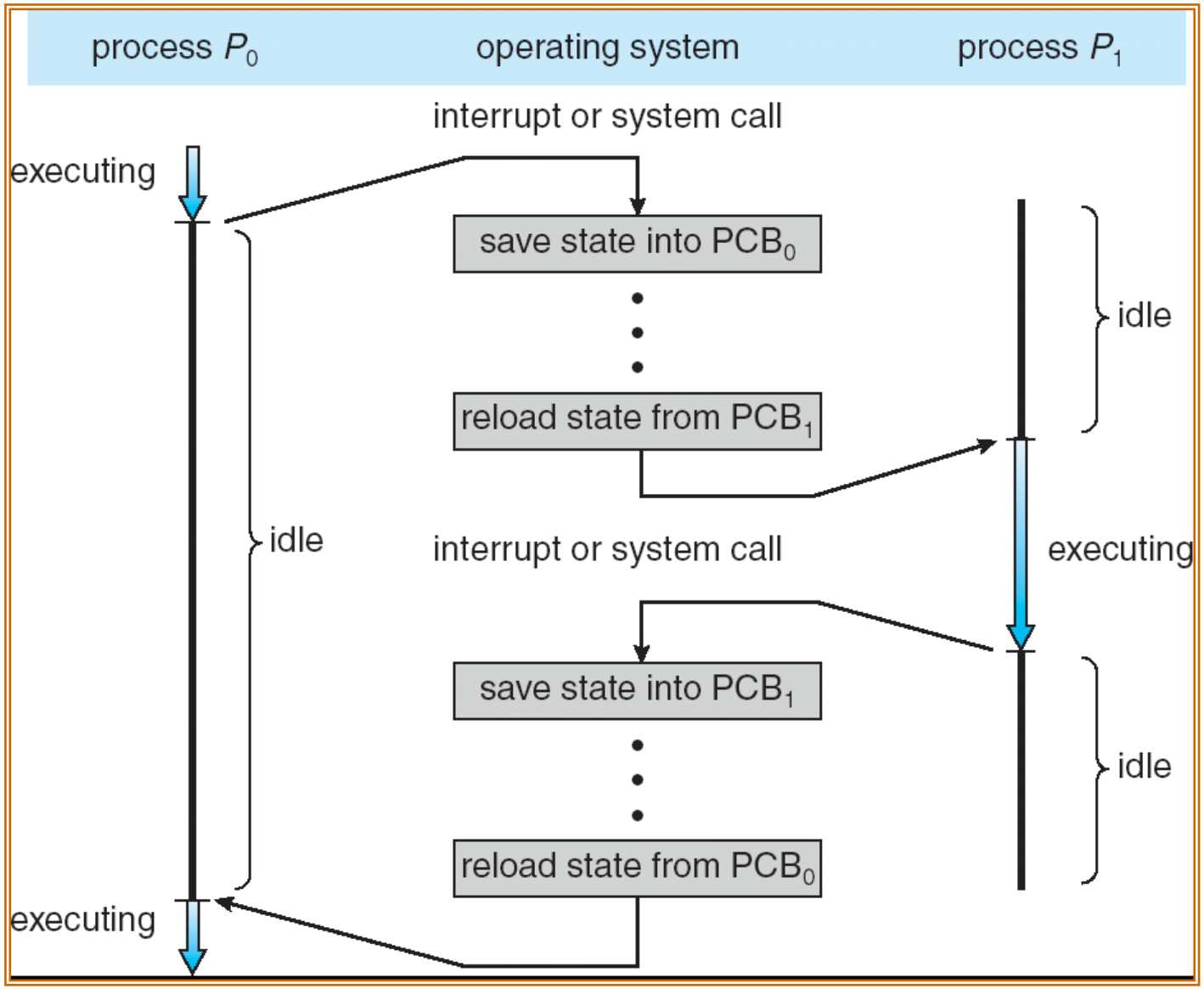


Processes



- The kernel stores the list of processes in a circular doubly linked list called the *task list*.
- Each element in the task list is a process descriptor of the type **struct *task_struct***, which is defined in *<linux/sched.h>*.
- The process descriptor contains all the information about a specific process (1.7 kilobytes in size).

CPU Switching From Process to Process



Context Switch

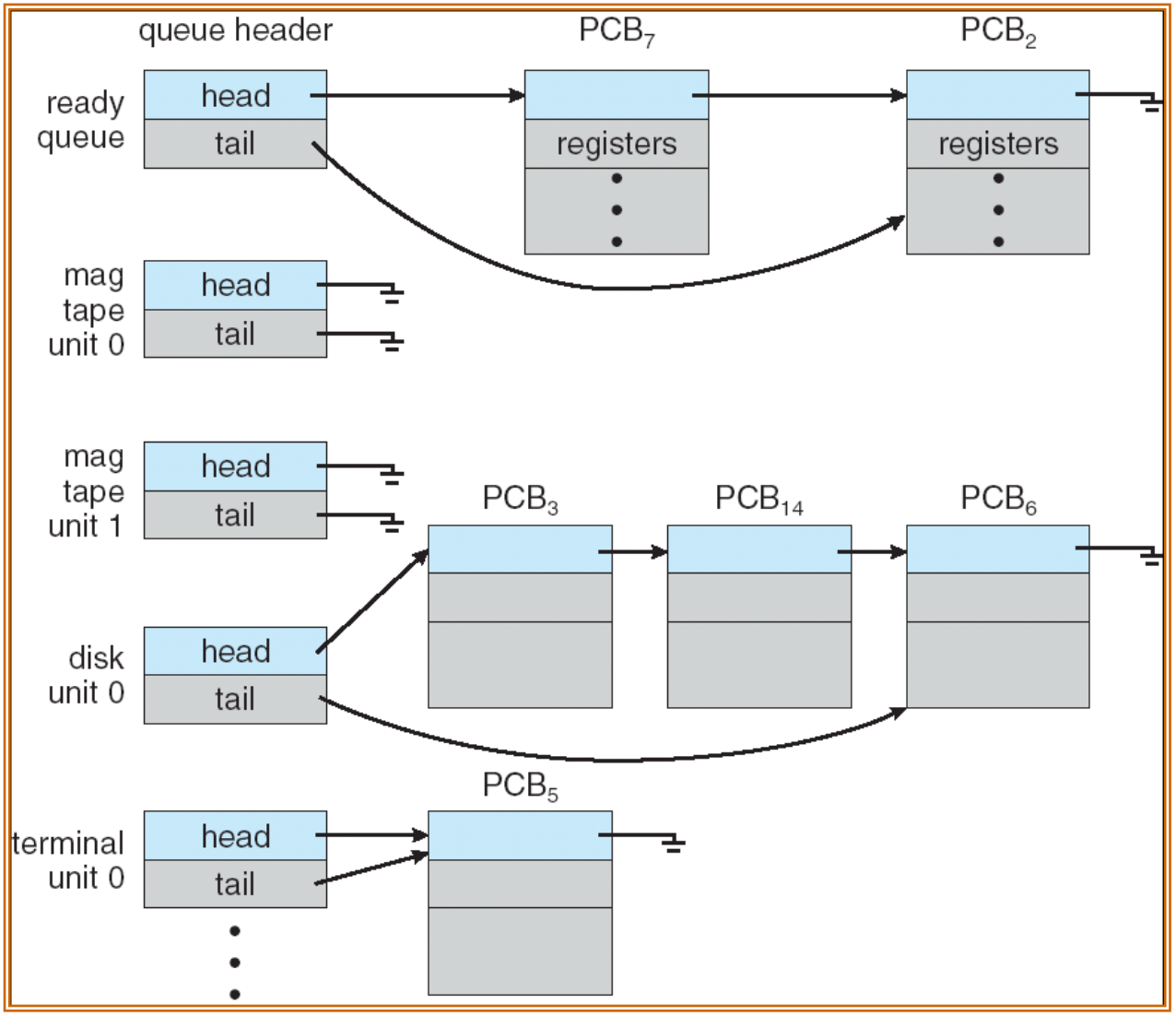
- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process
- Context-switch time is overhead; the system does no useful work while switching
- Time dependent on hardware support
  - Memory speed
  - Number of registers to be copied
  - The existence of special instructions (such as one instruction to load/store all registers)
  - Hardware support: UltraSPARC provide multiple sets of registers. ***Explain!***

Process Scheduling Queues

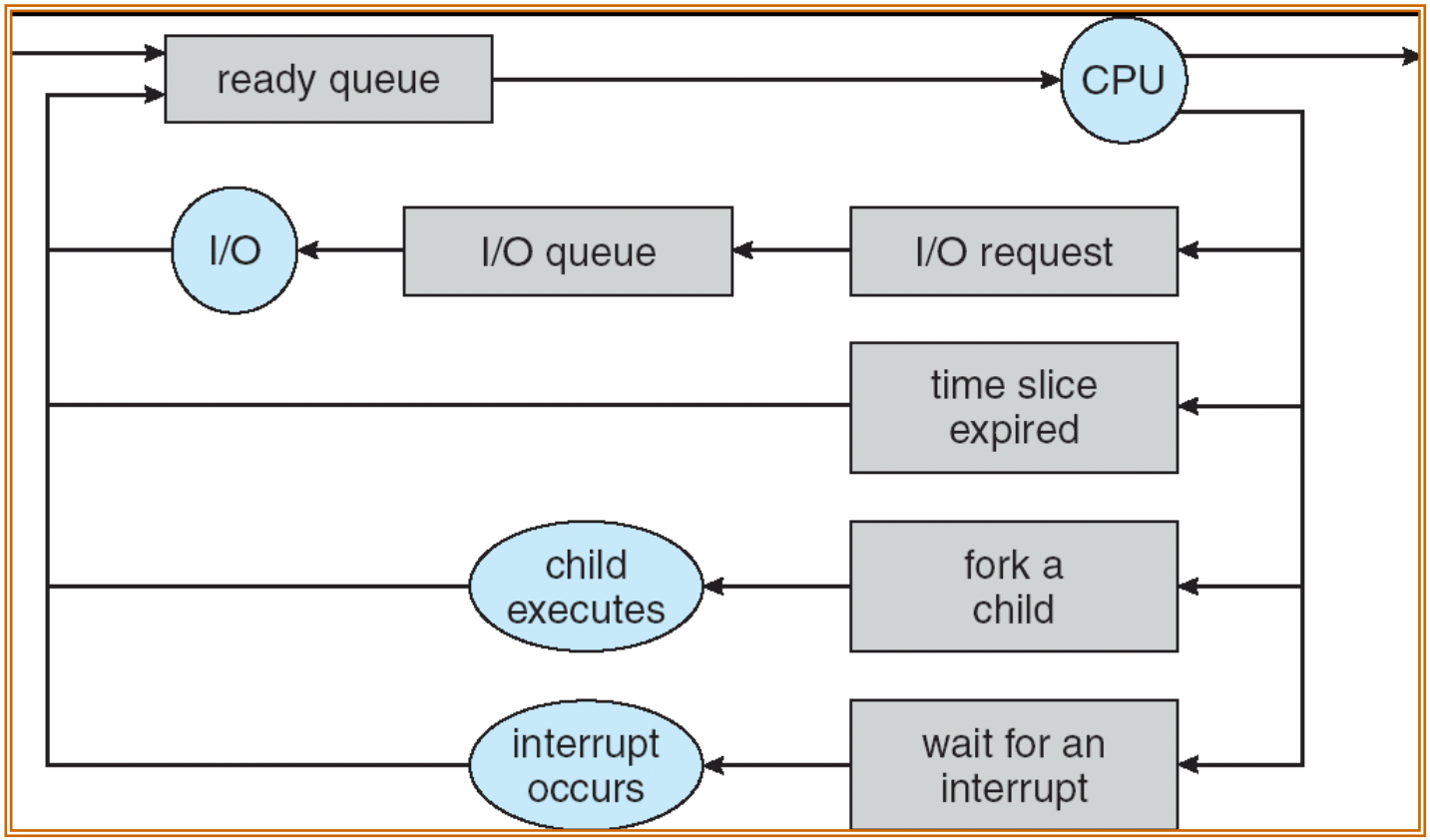
- **Job queue** – set of all processes in the system
- **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
- **Device queues** – set of processes waiting for an I/O device

- Processes migrate among the various queues

Ready Queue And Various I/O Device Queues



Representation of Process Scheduling



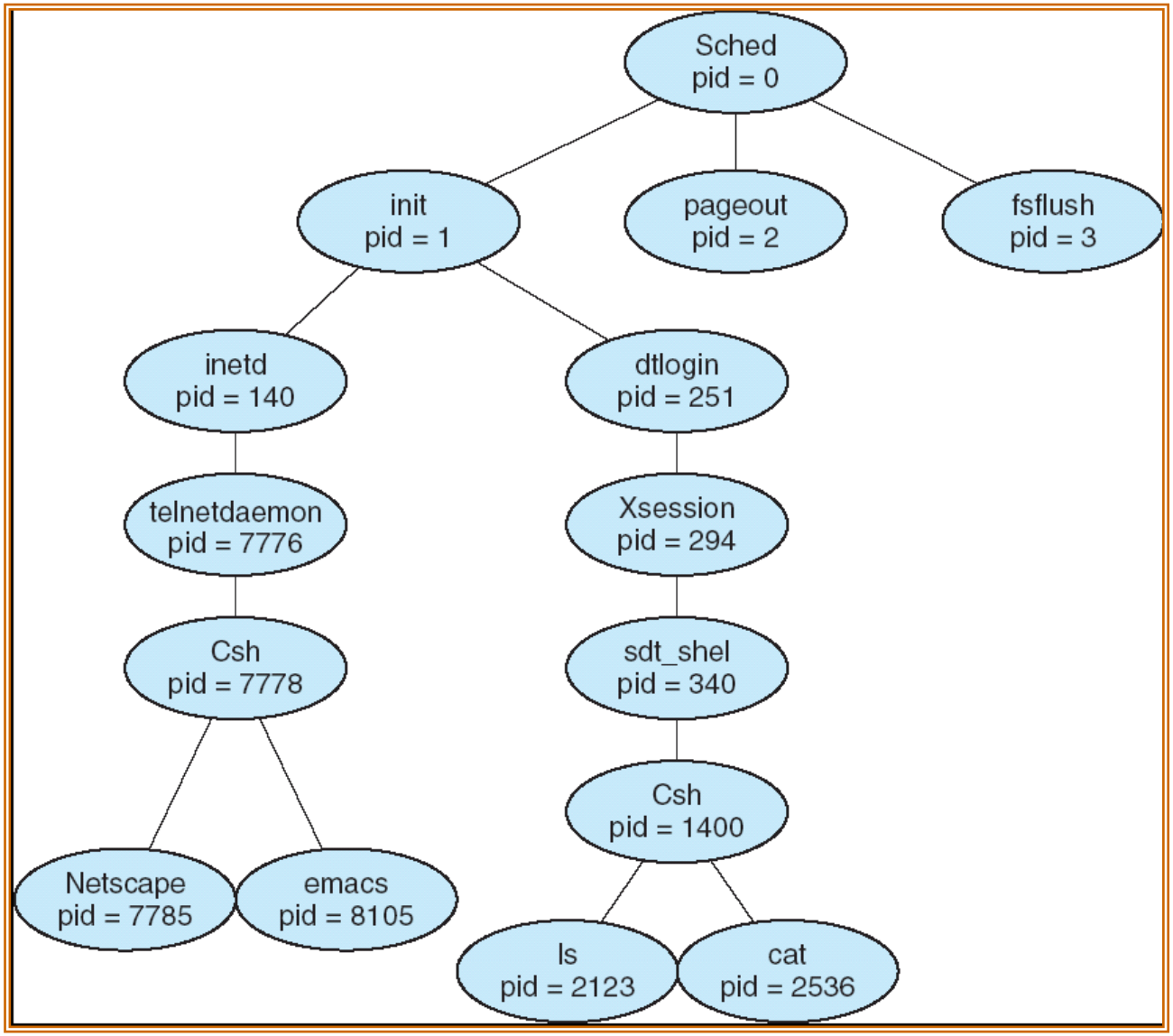
Schedulers

- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU

Schedulers

- Short-term scheduler is invoked very frequently (milliseconds) $\Rightarrow$ (must be fast)
- Long-term scheduler is invoked very infrequently (seconds, minutes) $\Rightarrow$ (may be slow)
- The long-term scheduler controls the *degree of multiprogramming*
- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts

A tree of processes on a typical Solaris



Viewing processes in Linux

- Can view running processes by various options:

1. Any of the commands:
   - ps aux
   - pstree
   - top
   - htop
   - atop
2. System > Administration > System Monitor