Operating Systems
(操作系统 2024秋冬)

教七-306（周3第7-8节，周5第1-2节）
实验课：曹西-503（周4第11-12节）

Lecture 1: Overview

Shou Lidan
should @ zju.edu.cn

Outlines of the overview

- Previously split in two modules, known as “Principles of Operating System” and “Operating System Practices”
- Background
  - The main purpose of this course
  - The contents and schedule of the course
  - The pre-requisite knowledge for the course
- Chapter 1 Introduction
- Chapter 2 Operating-System Structures

Main objectives

An introductory course for Operating System

- To learn the basics and internal design of operating systems
- To look at both the history and the state-of-the-art techniques used in operating systems
- To study the design methodologies applied in OS
- To prepare to practice the techniques in your future work

and

- To enjoy hacking around!

Main Contents

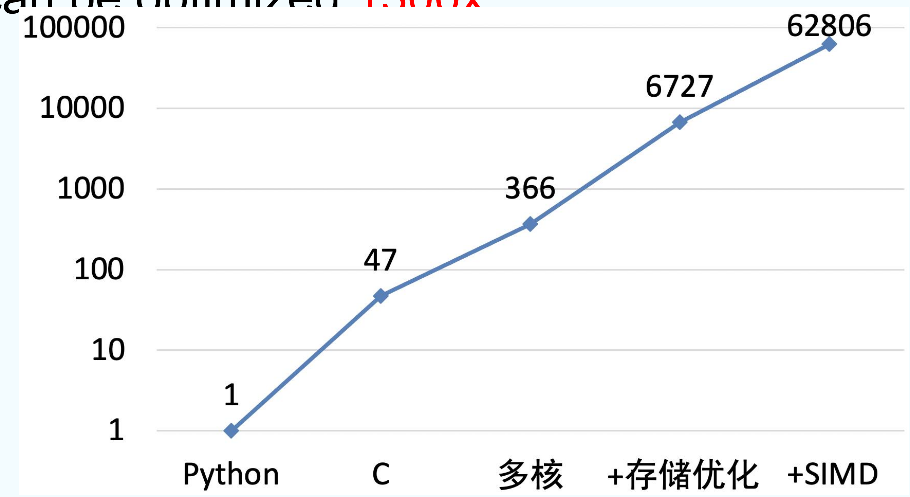
- Overview
  - Intro
  - OS structure
- Process Management
  - Processes
  - Threads
  - CPU scheduling
  - Process Synchronization
  - Deadlocks
- Memory Management
  - Main memory
  - Virtual memory
- Storage Management
  - File-system interface
  - File-system implementation
  - Mass-storage structure
  - I/O systems

Why is OS still important in ChatGPT-era?

- **There's plenty of room at the Top: What will drive computer performance after Moore's law?** https://www.science.org/doi/10.1126/science.aam9744
  - Performance improved 63000x
  - C program can be optimized 1300x

multiplying two 4096-by-4096 matrices

```
for i in xrange(4096):
 for j in xrange(4096):
  for k in xrange(4096):
   C[i][j] += A[i][k] * B[k][j]
```



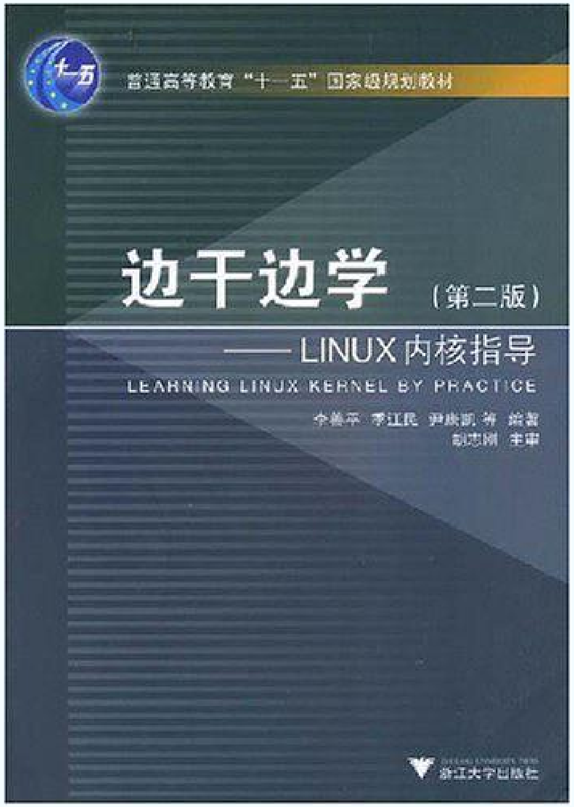
Software performance engineering, development of algorithms, and hardware streamlining at the Top can continue to make computer applications faster in the post-Moore era.

Textbook & Resources

- **Main Textbook: Operating System Concepts 7th, Abraham Silberschatz, Peter Galvin, Greg Gagne，2005.1**
- 高等教育出版社（第7版影印版）
- **10th edition can be found on Amazon.**
- **Course webpage: http://courses.zju.edu.cn/**
- **Lecture slides: http://courses.zju.edu.cn/**

Reference Books

- 边干边学Linux内核指导



- xv6: a simple, Unix-like teaching operating system, Russ Cox, Frans Kaashoek, Robert Morris, August 31, 2020
- Latest version: https://pdos.csail.mit.edu/6.1810/2022/xv6.html

Online Assignments

http://courses.zju.edu.cn/

Teaching Assistants: 刘得志、石劲、包牧天

Please submit your assignment IN TIME!

Pre-requisite Knowledge

- Computer system architecture/organization
- Programming language C
- Data structures

Lab Projects

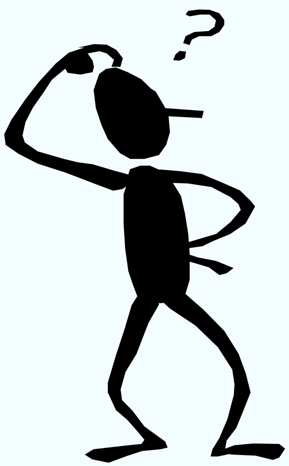
- 每个同学独立完成
  - 实验0 RISC-V 64 内核调试 5%
  - 实验1 内核引导；时钟和中断 15%
- 分组完成5个实验，每组2人
  - 实验2 线程调度,上下文切换 15%
  - 实验3 虚拟内存管理 15%
  - 实验4 用户模式（shell） 20%
  - 实验5 Page fault 20%
  - 实验6 Fork 10%
- 实验7 File System Bonus 10%

Lab Projects

- 实验0、1：每个同学提交一份实验报告；
- 实验2-6：每个同学提交一份实验报告（侧重自己完成的那部分内容）；
  - 自己写的代码必须有详细的注释，每5行代码有注释
  - 每个实验必须写“讨论心得” (实验过程中遇到的问题及解决方法)这部分内容占本实验报告20%分数
- 每个实验完成后，以个人（lab0-1）/小组（lab2-3-4-5-6）为单位向老师/助教演示，以完成验收。
- 实验说明文档在 https://zju-sec.github.io/os24fall-stu/
- 实验所需代码 https://github.com/ZJU-SEC/os24fall-stu

Final Grade

● Final exam	(50%)
● Assignments/homework	(5%)
● In-class Quiz	(5%)
● Lab Reports 实验报告	(20%)
● Lab Demos 实验验收	(20%)



My suggestions on learning OS

- Do a lot of readings before or after the lecture sessions, especially from the English textbook.
- Do NOT refer to the so-called “standard” answers to the exercises, they contain mistakes. Once you are caught using the wrong “standard” answers, penalty might be applicable.
- Hopefully more interactions during the lectures.

怎样学好系统课程

- 系统、全局的观察角度 vs 微观、细致的想象能力

王阳明《蔽月山房》

山近月远觉月小，便道此山大于月。

若有人眼大如天，当见山高月更阔。

- 理论钻研 vs 动手实践

丁肇中：在环境激变的今天，我们应该重新体会到几千年前经书里说的格物致知真正的意义。这意义有两个方面：第一，寻求真理的唯一途径是对事物客观的探索，而不是仅从内心就能领悟；第二，探索的过程不是消极的袖手旁观，而是深度思考勇于探索。

希望我们这一代对于格物和致知有新的认识和思考，使得实干精神真正地变成中国文化的一部分。

- 寻找模式
  - 螺旋式上升；上下文；封装；抽象；过程式和声明式
- 寻找兴奋点

Lab0: GDB + QEMU 调试 LINUX

- 请先将os24fall-stu目录 clone下来
- 实验说明文档在 os24fall-stu/docs/lab0.md
  - https://zju-sec.github.io/os24fall-stu/ 可浏览器阅读文档

- 实验重点：掌握实验环境、尝试简单调试
- gdb有一个text UI可以尝试使用，按 Ctrl-x Ctrl-a 进入或者“gdb –tui”

Chapter 1 Introduction

Outlines for Chapter ONE

- What Operating Systems Do
- Computer-System Organization
- Computer-System Architecture

Overview

- Operating-System Structure
- Operating-System Operations

Structure

- Process Management
- Memory Management
- Storage Management

Main parts

- Protection and Security
- Distributed Systems
- Special-Purpose Systems
- Computing Environments

Objectives

- To provide a grand tour of the major operating systems components
- To provide coverage of basic computer system organization

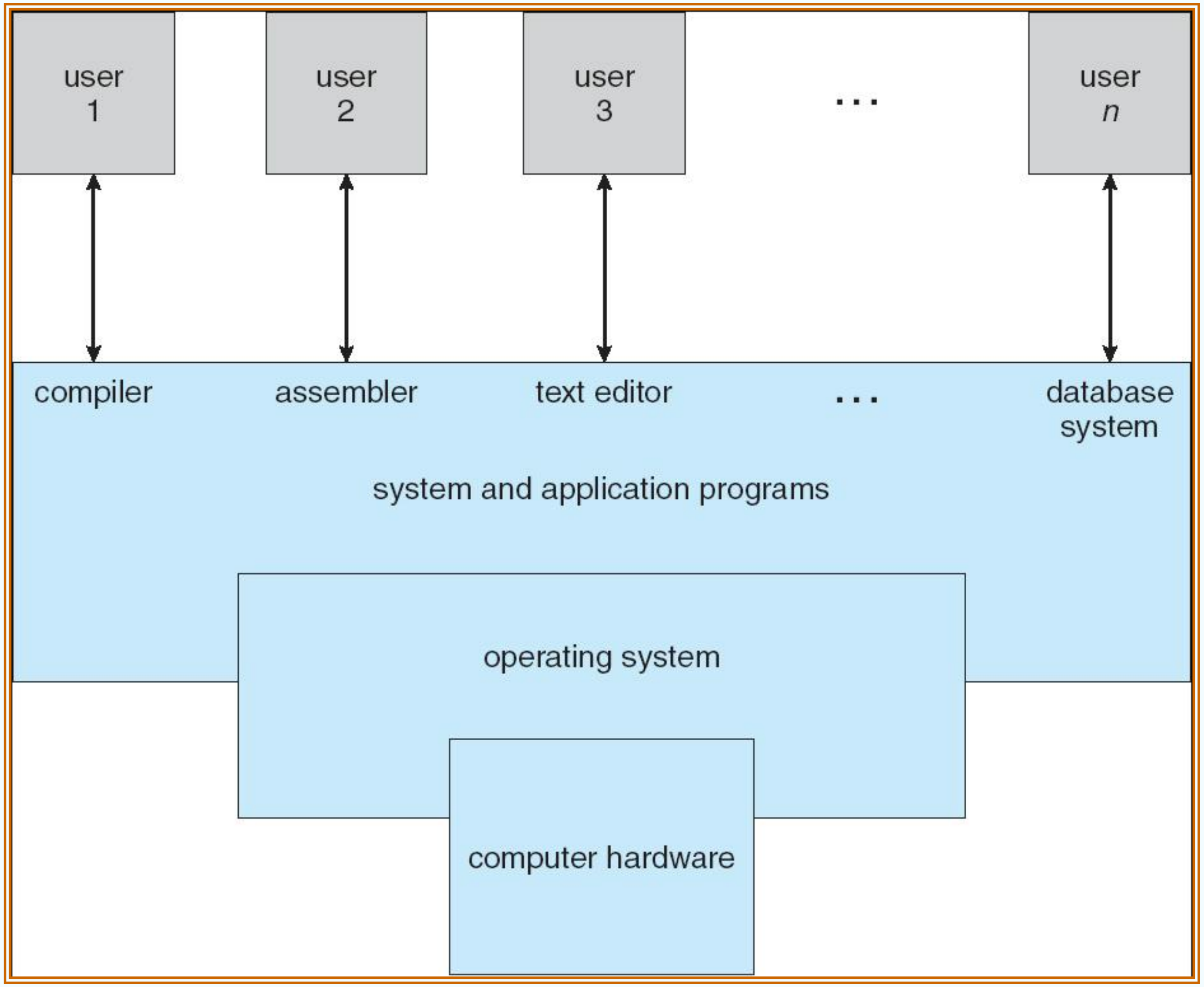
What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware.
- 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.

Computer System Structure

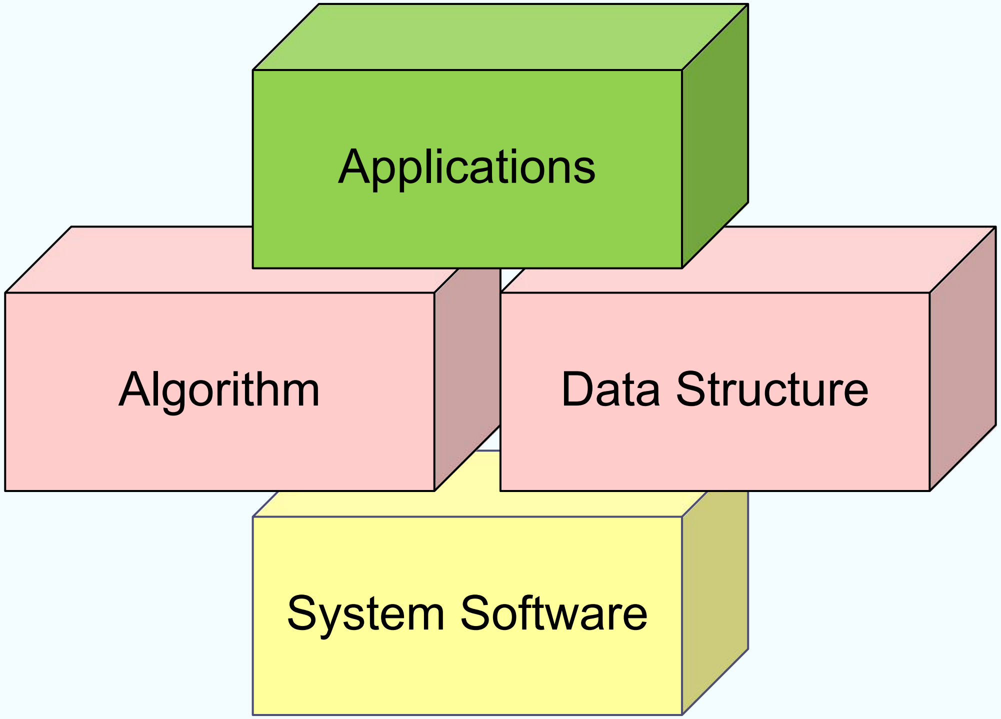
- Computer system can be divided into four components
  - Hardware – provides basic computing resources
    - CPU, memory, I/O devices
  - Operating system
    - Controls and coordinates use of hardware among various applications and users
  - System & application programs – define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, compilers, web browsers, database systems, video games
  - Users
    - People, machines, other computers

Four Components of a Computer System



Why To Learn Operating Systems?

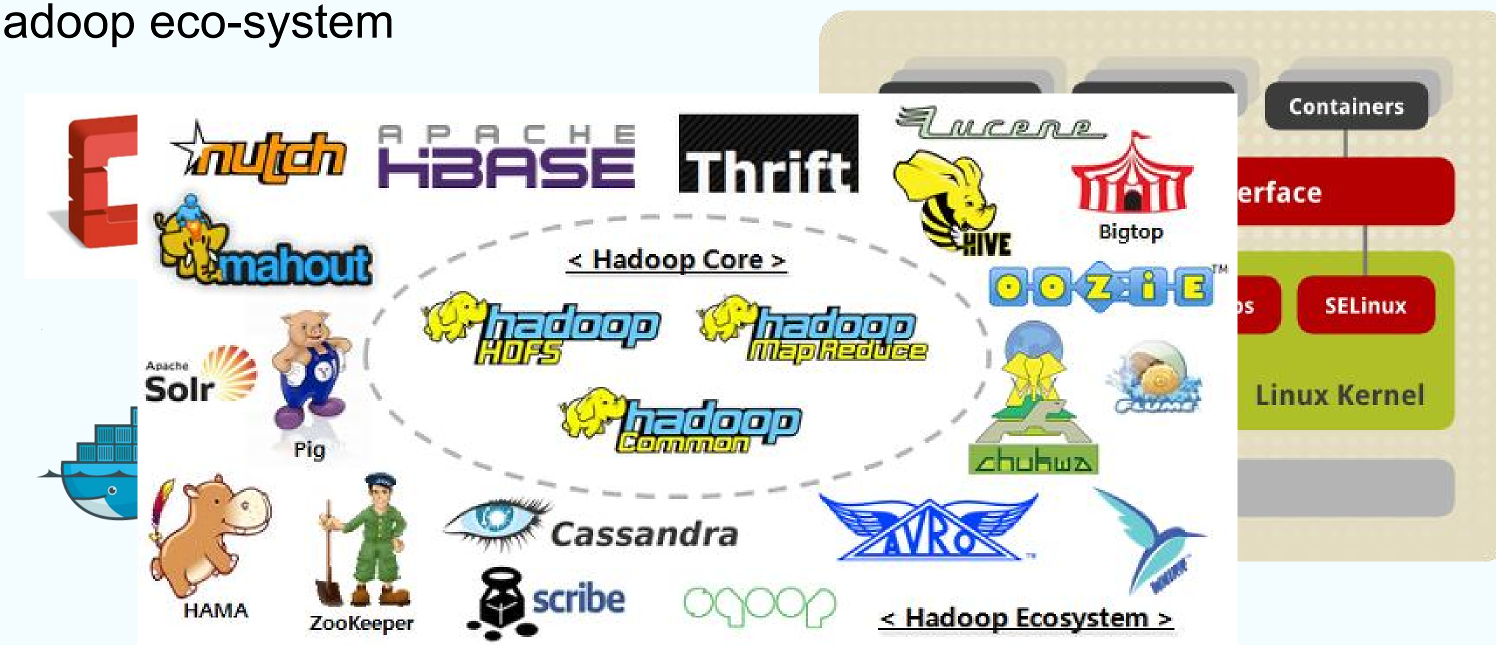
- Building blocks of modern computer software



Why To Learn Operating Systems?

New system technologies involving OS:

- Cloud Computing becomes the major computing model
- Virtualization, Software-Defined Storage/Network
- Containers
- Hadoop eco-system



Operating System Definition

- OS is a **resource allocator**
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer

Operating System Definition (Cont.)

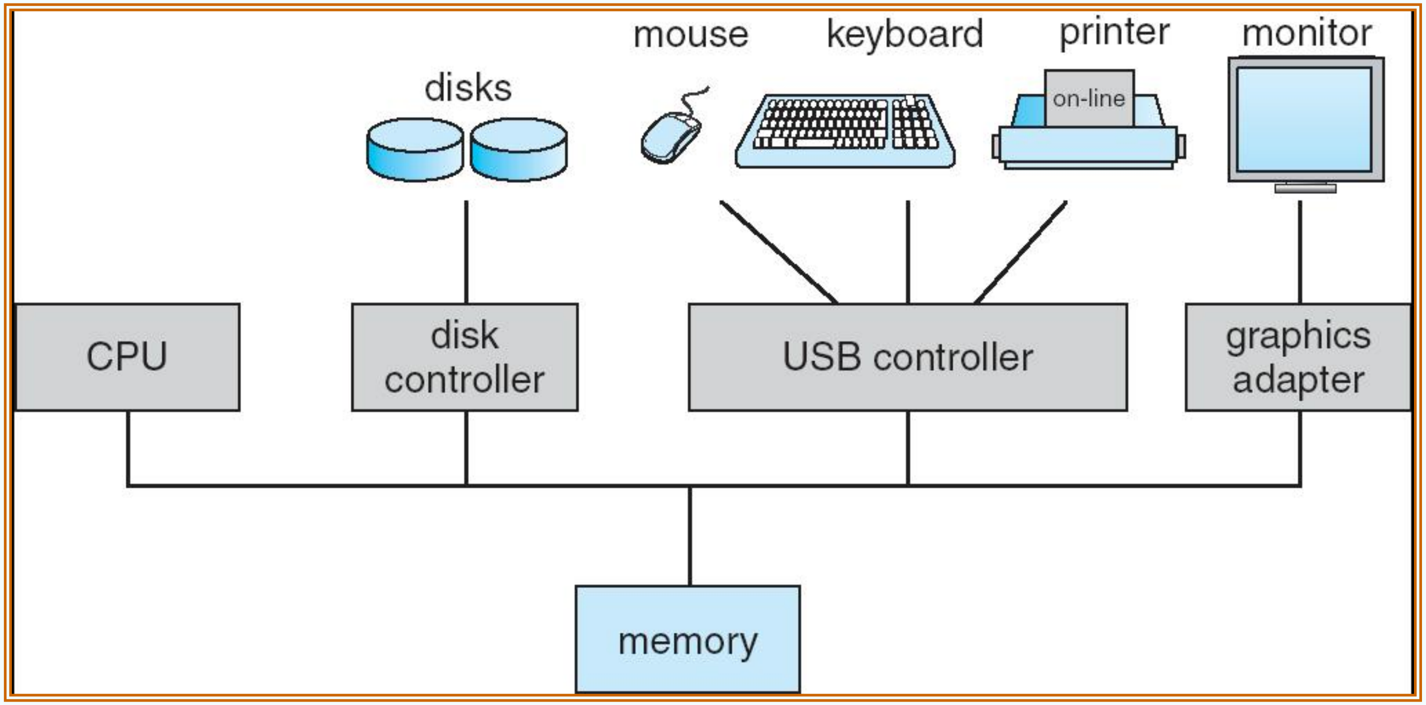
- No universally accepted definition
- "Everything a vendor ships when you order an operating system" is good approximation
  - But varies wildly
- "The one program running at all times on the computer" is the **kernel.** Everything else is either a system program (ships with the operating system) or an application program

Computer Startup

- **bootstrap program** is loaded at power-up or reboot
  - Typically stored in ROM or EPROM, generally known as **firmware**
  - Initializes all aspects of system
  - Loads operating system kernel and starts execution

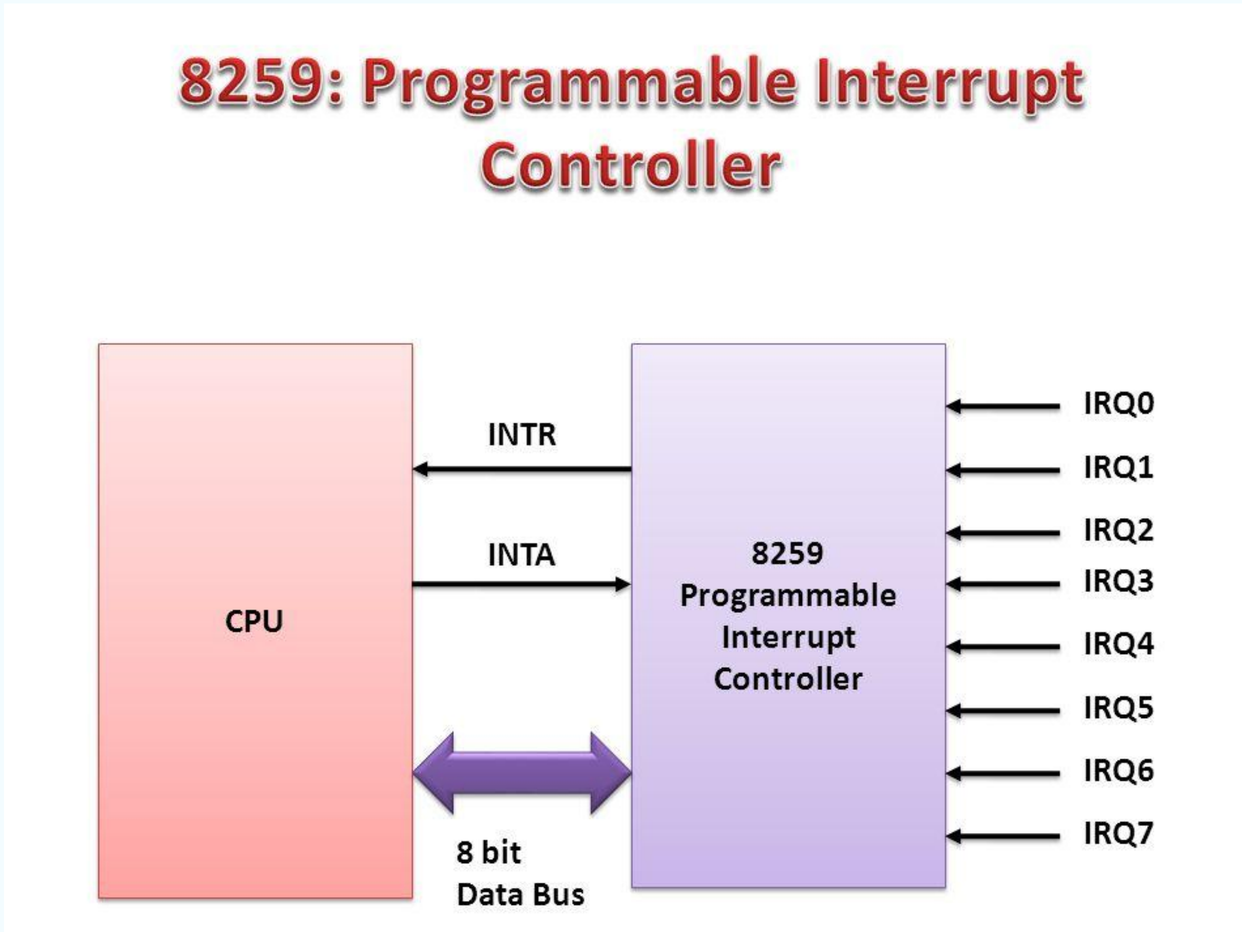
Computer System Organization

- 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



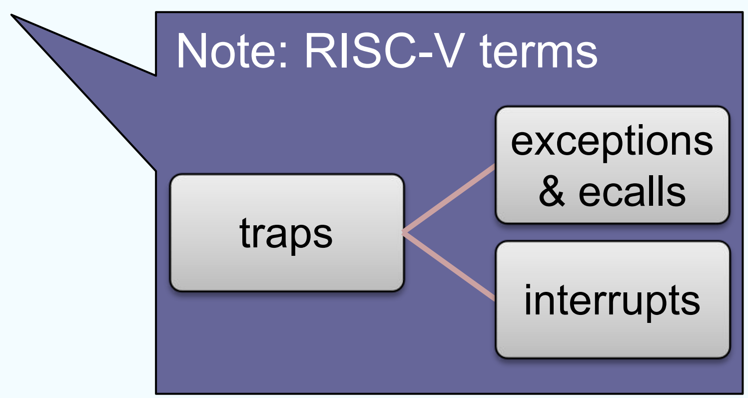
Computer-System Operation

- I/O devices and the CPU can execute concurrently.
- Each device controller is in charge of a particular device type.
- Each device controller has a local buffer.
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller.
- Device controller informs CPU that it has finished its operation by causing an *interrupt* (via system bus).


8259: Programmable Interrupt Controller
CPU
INTR
INTA
8 bit
Data Bus
8259
Programmable
Interrupt
Controller
IRQ0
IRQ1
IRQ2
IRQ3
IRQ4
IRQ5
IRQ6
IRQ7

Common Functions of Interrupts

- Interrupt transfers control to the interrupt service routine generally, through the *interrupt vector*, which contains the addresses of all the service routines.
- Interrupt architecture must save the address of the interrupted instruction.
- Incoming interrupts are *disabled* while another interrupt is being processed to prevent a *lost interrupt*.
- A *trap* is a software-generated interrupt caused either by an error or a user request (the latter is often referred to as a *system call*). Note: names may vary across different architectures.
- An operating system is *interrupt* driven.



Interrupt Handling

- The operating system preserves the state of the CPU by storing registers and the program counter.
- Determines which type of interrupt has occurred:
  - a generic routine examines the interrupt info
  - *vectored* interrupt system, indexed by a unique number
- Separate segments of code determine what action should be taken for each type of interrupt

Interrupt Timeline

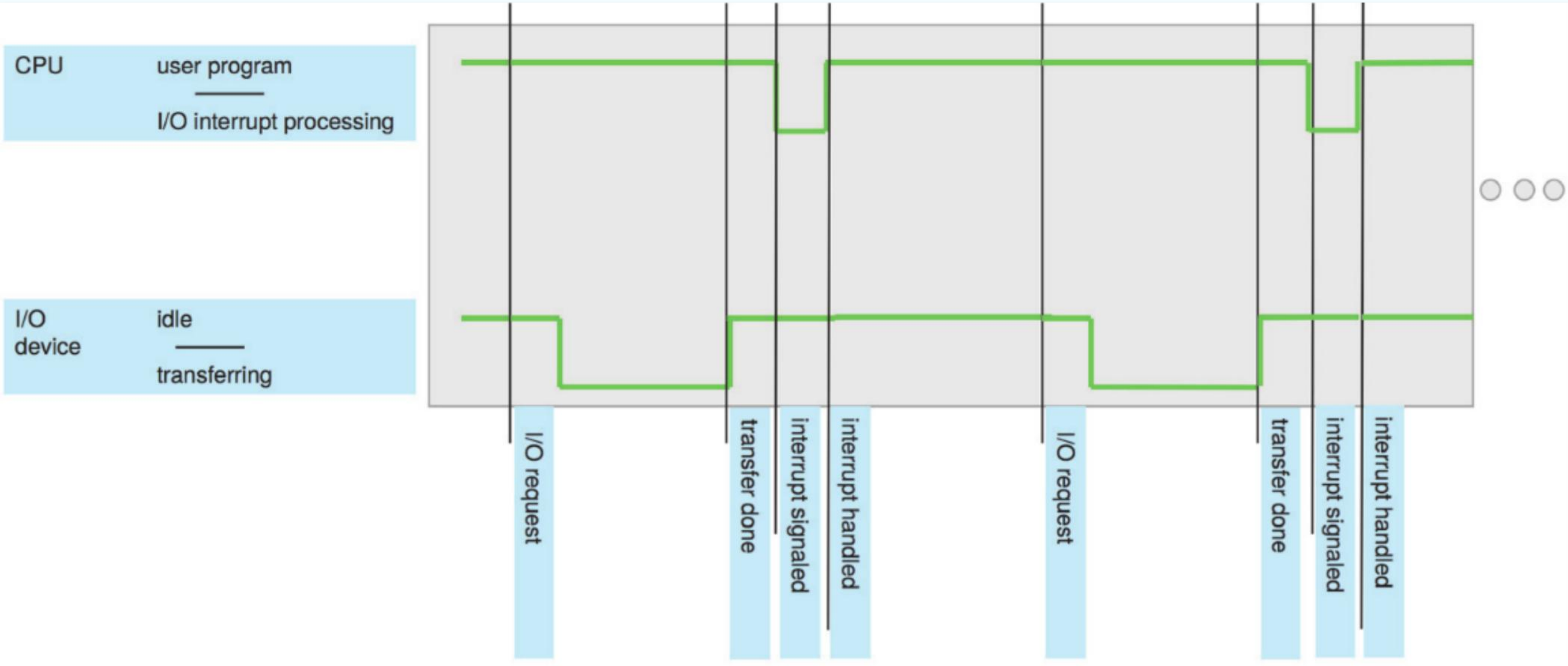
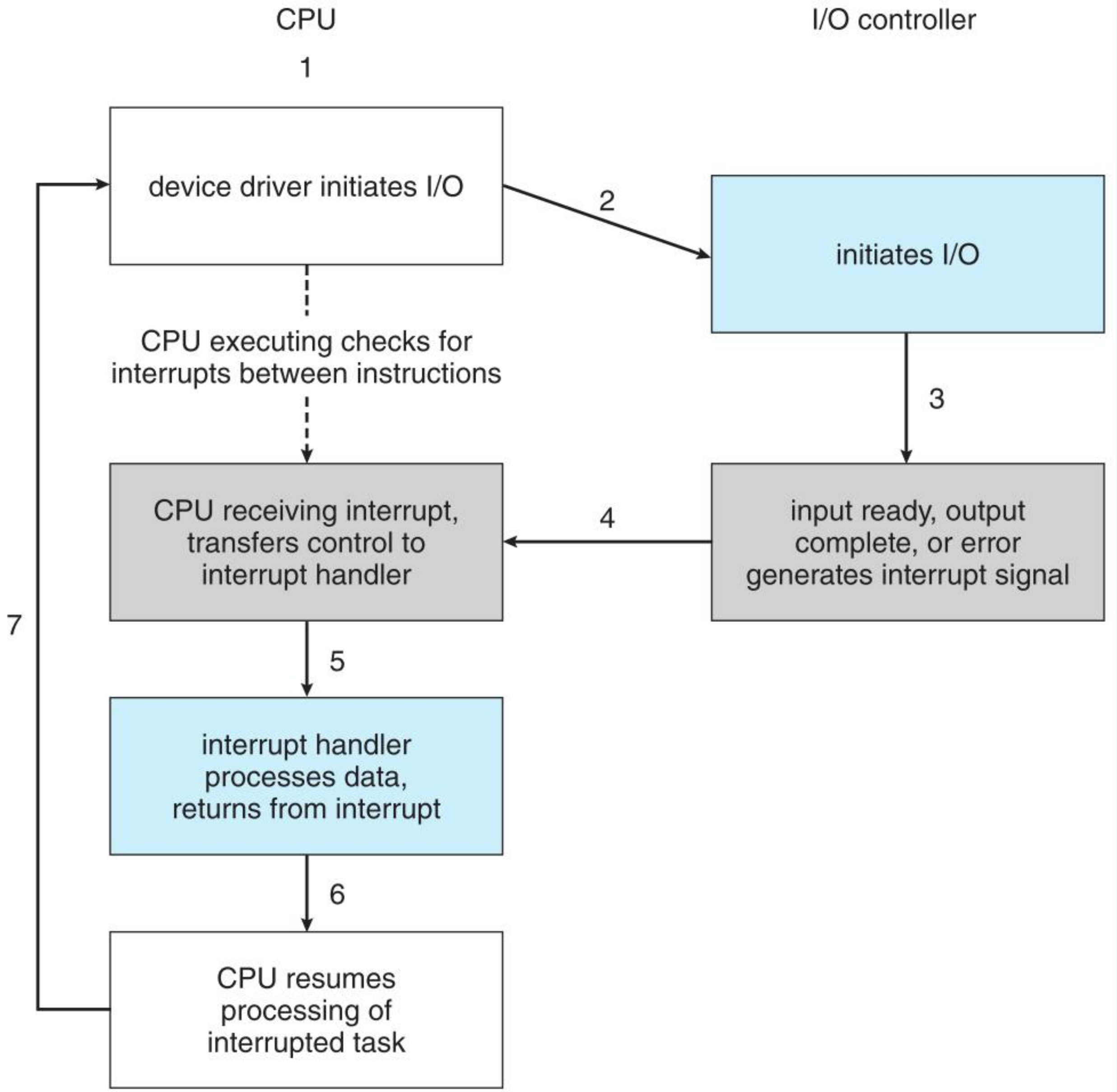


Figure 1.3 Interrupt timeline for a single program doing output.

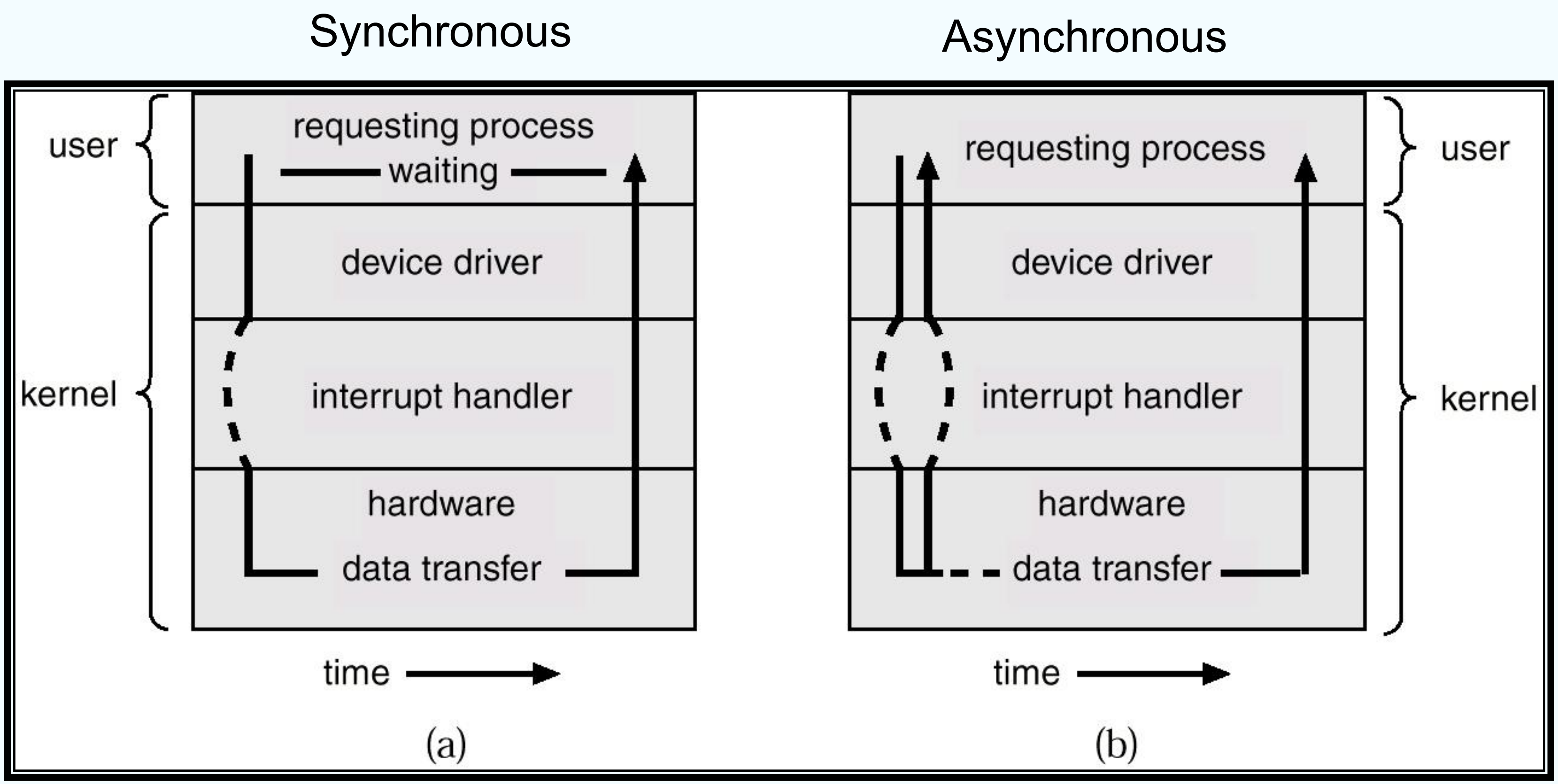
Interrupt-Driven I/O Cycle



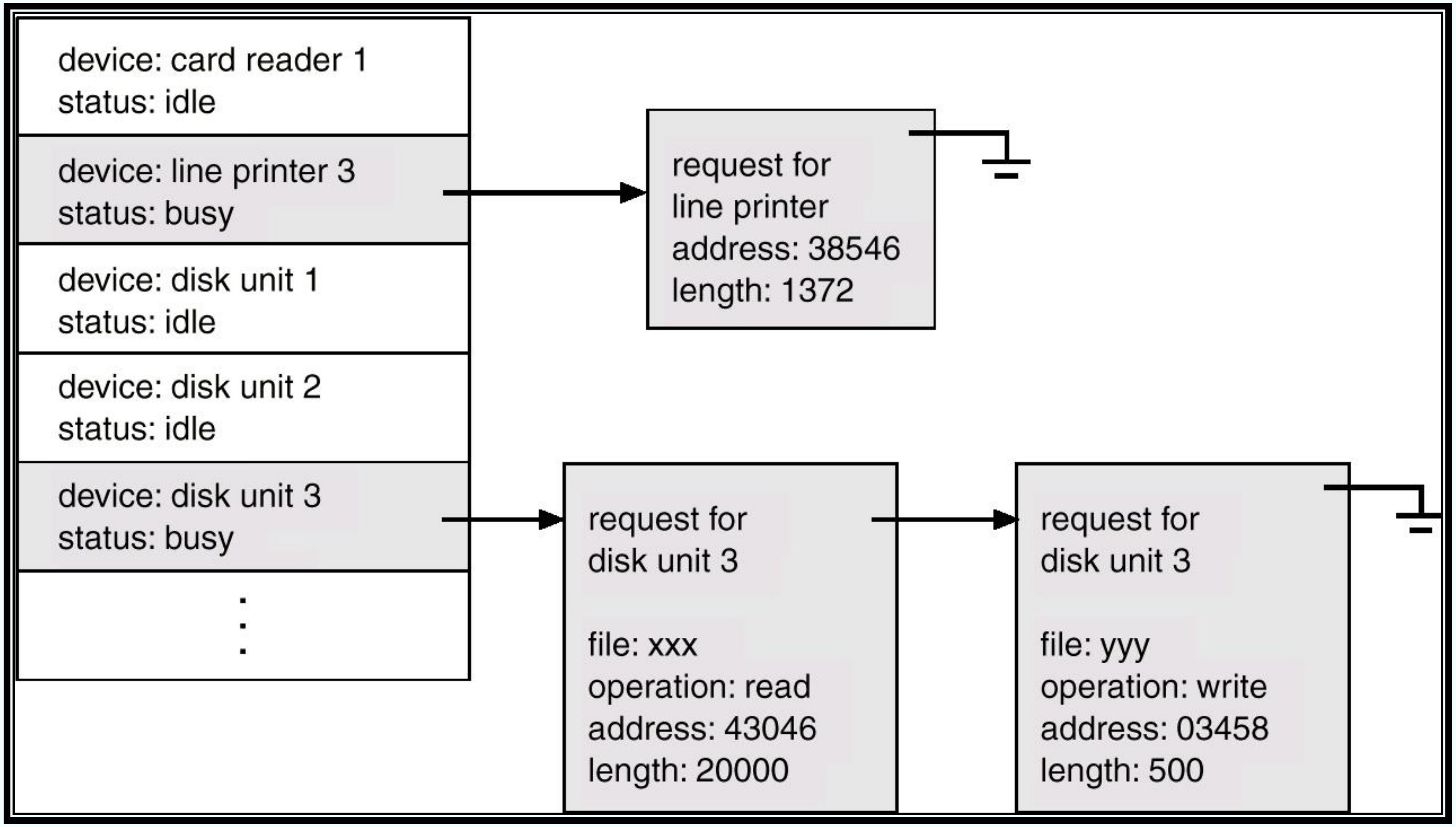
I/O Structure – Two I/O Methods

- After I/O starts, control returns to user program only upon I/O completion.
  - 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.
  - *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 indicating its type, address, and state.
  - Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt.

Two I/O Methods



Device-Status Table



Direct Memory Access Structure

- Used for high-speed I/O devices able to transmit information at close to memory speeds.
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
- Only one interrupt is generated per block, rather than the one interrupt per byte.

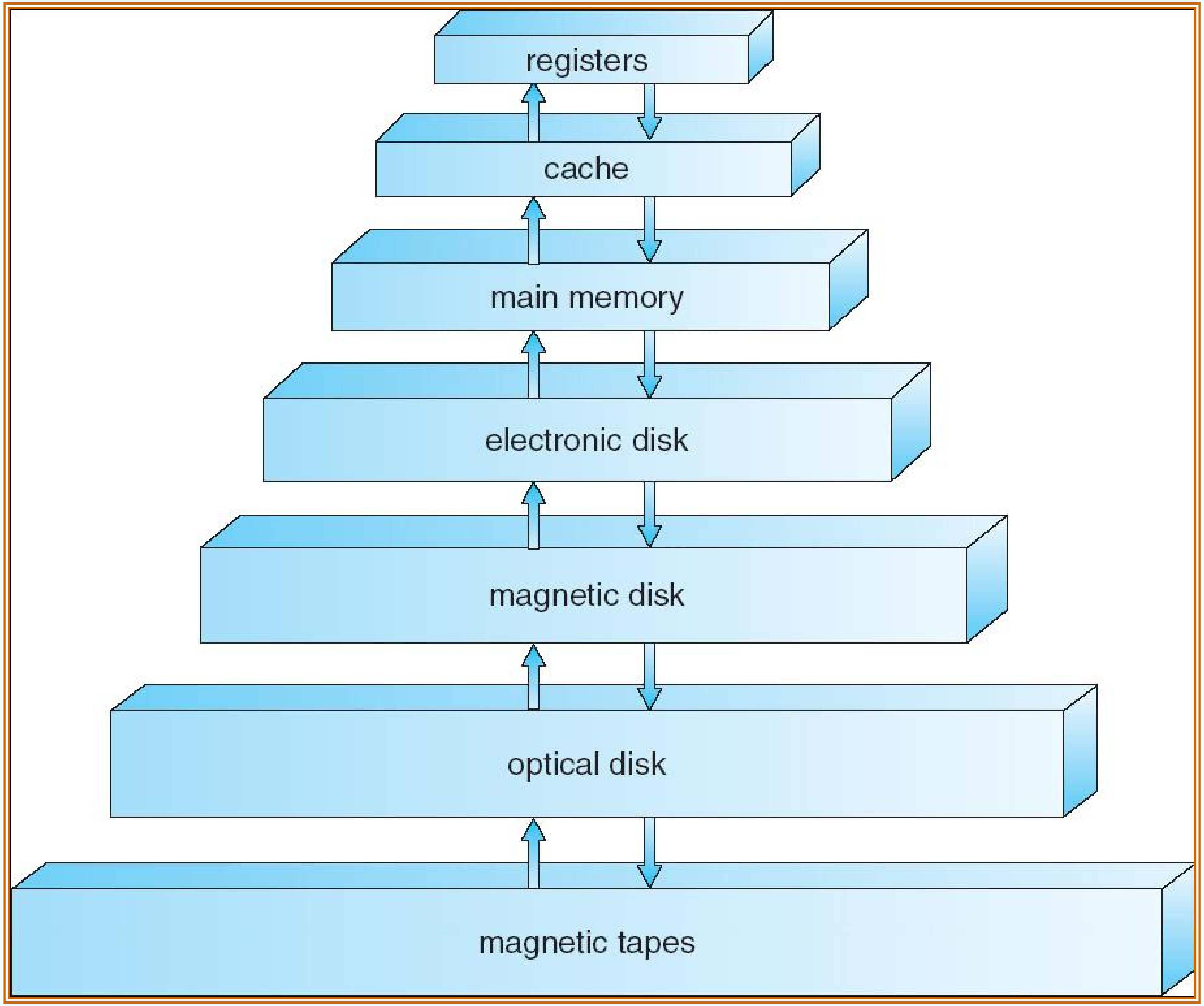
Storage Structure

- Main memory – only large storage media that the CPU can access directly.
- Secondary storage – extension of main memory that provides large nonvolatile storage capacity.
- Magnetic disks – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into *tracks*, which are subdivided into *sectors*.
  - The *disk controller* determines the logical interaction between the device and the computer.

Storage Hierarchy

- Storage systems organized in hierarchy.
  - Speed
  - Cost
  - Volatility
- *Caching* – copying information into faster storage system; main memory can be viewed as a last *cache* for secondary storage.

Storage-Device Hierarchy



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 – Speed mismatch
- 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

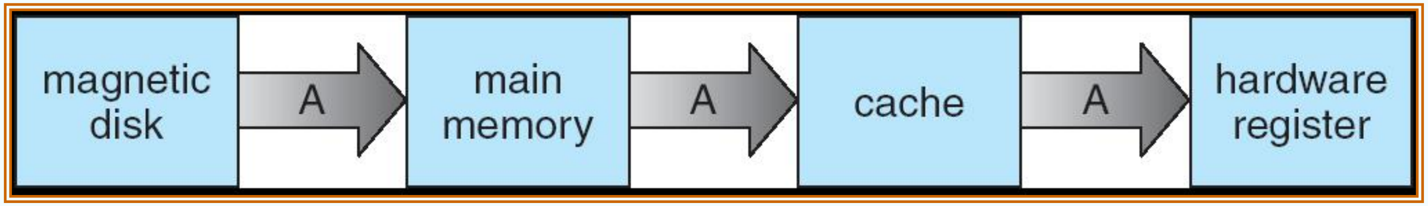
Performance of Various Levels of Storage

- 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

Migration of Integer A from Disk to Register

- Multitasking environments must be careful to use most recent value, no 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

Multiprocessor Systems

- SMP architecture
  - Each CPU processor has its own set of registers
  - All processors share physical memory over the system bus

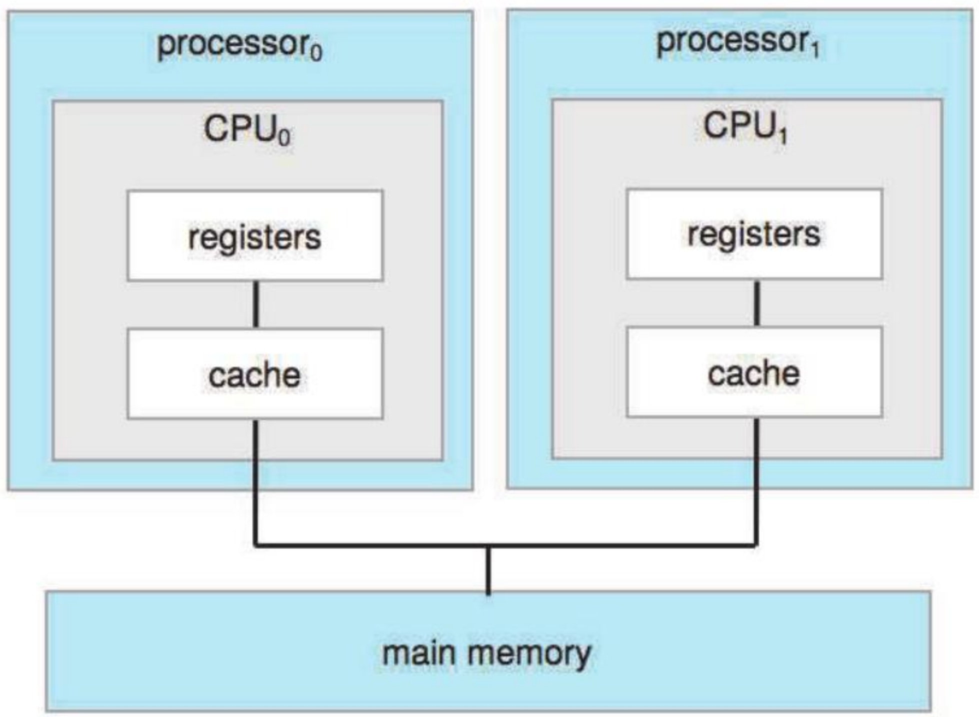
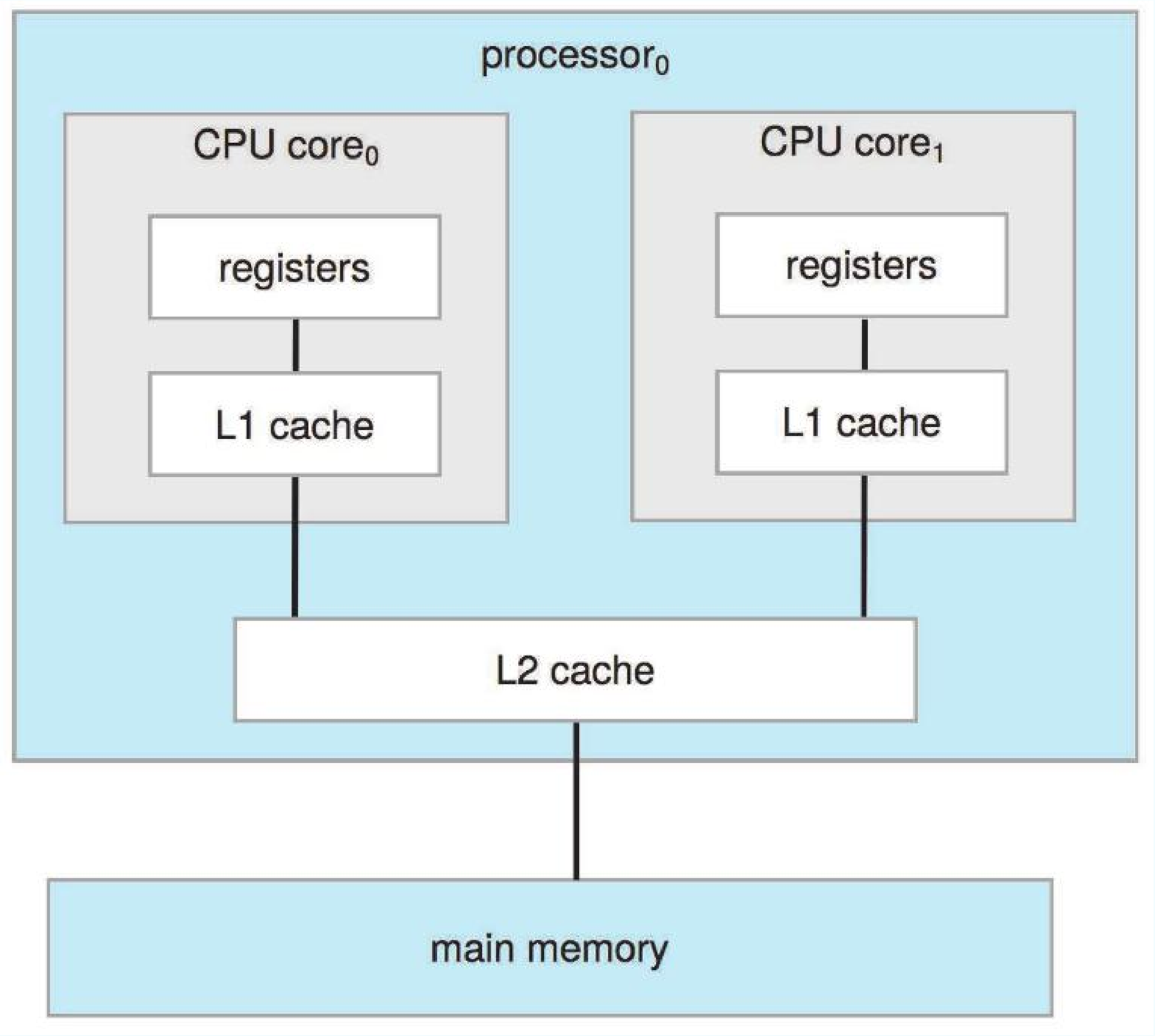


Figure 1.8 Symmetric multiprocessing architecture.

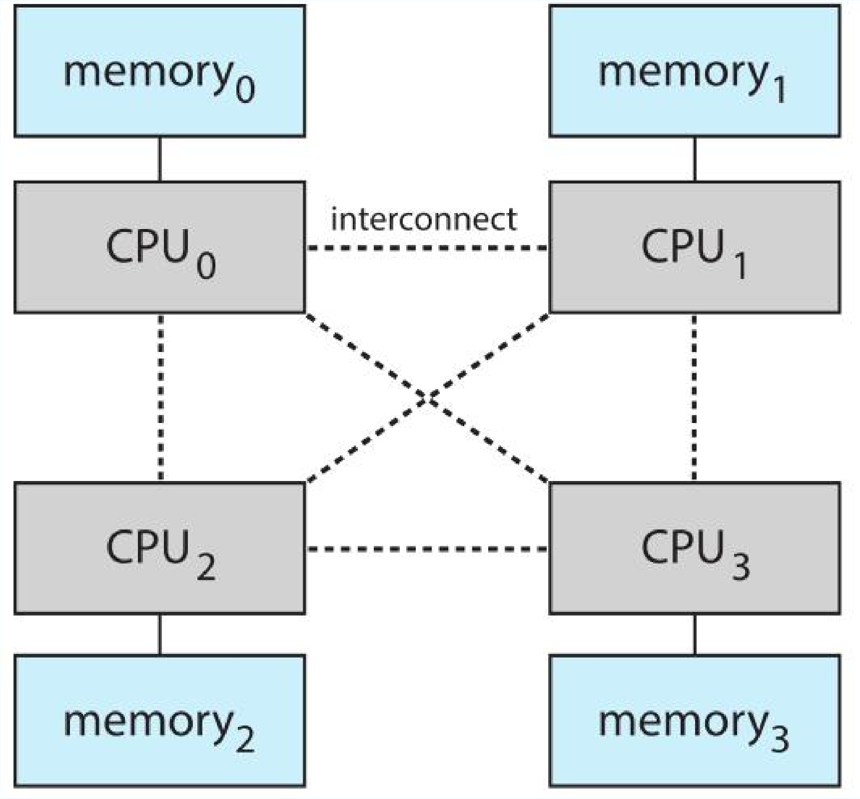
Multicore Systems

- On-chip communication is faster than between-chip communication
- Less power (good for mobile devices)



The NUMA Architecture

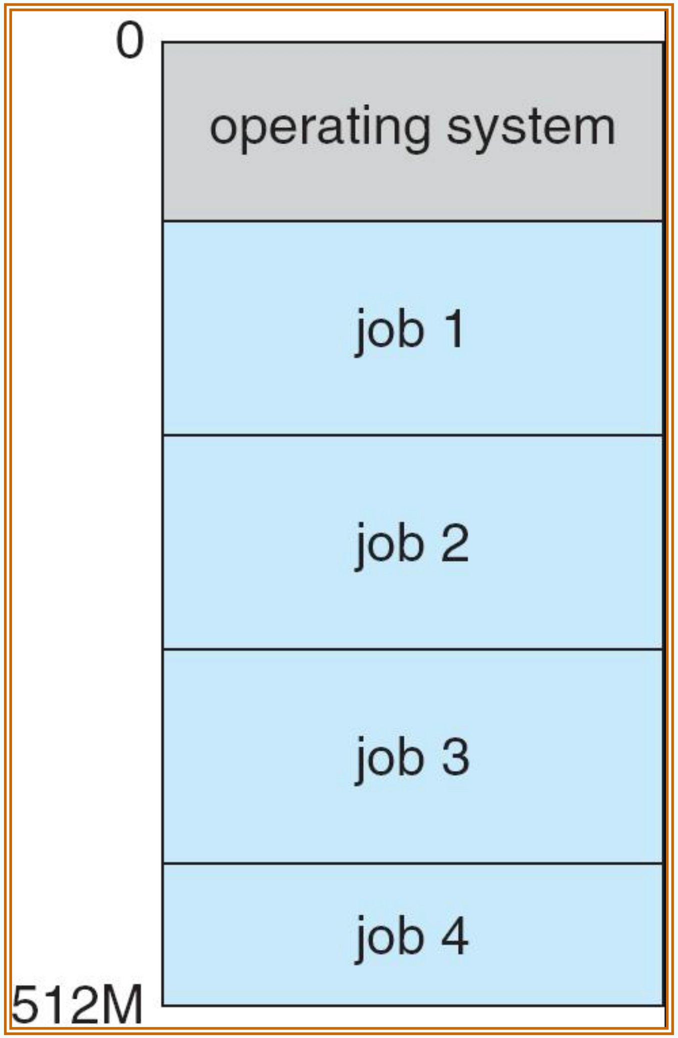
- The CPUs are connected by a shared system interconnect
- Scales more effectively as more processors are added
- Remote memory across the interconnect is slow
- Operating systems need careful CPU scheduling and memory management



Operating System Structure

- **Multiprogramming** needed for efficiency **(CPU utilization)**
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job
- **Timesharing (multitasking)** is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing **(interactivity)**
  - **Response time** should be < 1 second
  - Each user has at least one program executing in memory [**process**
  - If several jobs ready to run at the same time [**CPU scheduling**
  - If processes don't fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in memory

Memory Layout for Multiprogrammed System



Operating-System Operations

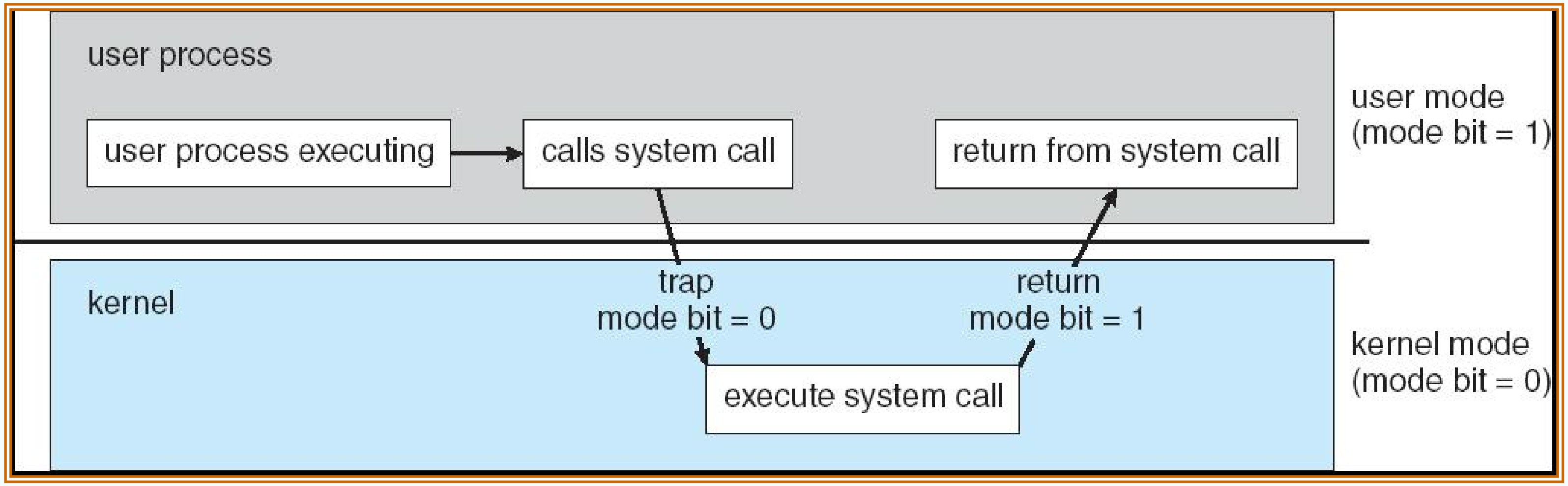
- Interrupt driven by hardware
- Software error or request creates **exception** or **trap**
  - Division by zero, request for operating system service
- Other process problems include infinite loop, processes modifying each other or the operating system

Thus we need protection:

- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions designated as **privileged**, only executable in kernel mode
    - System call changes mode to kernel, return from call resets it to user

Transition from User to Kernel Mode

- Timer to prevent infinite loop / process hogging resources
  - Set interrupt after specific period
  - Operating system decrements counter
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time



Process Management

- A process is a program in execution. It is a unit of work within the system. Program is a *passive entity*, process is an *active entity*.
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one **program counter** specifying location of next instruction to execute
  - Process executes instructions sequentially, one at a time, until completion
- Multi-threaded process has one program counter per thread
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
  - Concurrency by multiplexing the CPUs among the processes / threads

Process Management Activities

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

Memory Management

- All data must be in memory before and after processing
- All instructions must be in memory in order to execute
- Memory management determines what is in memory when
  - Optimizing CPU utilization and computer response to users
- Memory management activities
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed

Storage Management

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - **file**
  - Each medium is controlled by device (i.e., disk drive, tape drive)
    - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
- File-System management
  - Files usually organized into directories
  - Access control on most systems to determine who can access what
  - OS activities include
    - Creating and deleting files and directories
    - Primitives to manipulate files and dirs
    - Mapping files onto secondary storage
    - Backup files onto stable (non-volatile) storage media

Mass-Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a "long" period of time.
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
  - Free-space management
  - Storage allocation
  - Disk scheduling
- Some storage needs not be fast
  - Tertiary storage includes optical storage, magnetic tape
  - Still must be managed
  - Varies between WORM (write-once, read-many-times) and RW (read-write)

I/O Subsystem

- One purpose of OS is to hide peculiarities of hardware devices from the user – *ease of usage & programming*
- I/O subsystem responsible for
  - Memory management of I/O including buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for performance), spooling (the overlapping of output of one job with input of other jobs)
  - General device-driver interface
  - Drivers for specific hardware devices

OS Purposes

- Abstraction
- Multiplex
- Isolation
- Sharing
- Security
- Performance
- Range of uses

End of Chapter 1

RISC-V Registers

- Register file as follows:

- **RV32I/64I have 32 Integer Registers**
  - Optional 32 FP registers with the F and D extensions
  - RV32E reduces the register file to 16 integer registers for area constrained embedded devices
- **Width of Registers is determined by ISA**
- **RISC-V Application Binary Interface (ABI) defines standard functions for registers**
  - Allows for software interoperability
- **Development tools usually use ABI names for simplicity**

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	-
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	-
x4	tp	Thread pointer	-
x5-7	t0-2	Temporaries	Caller
x8	s0/fp	Saved register/Frame pointer	Callee
x9	s1	Saved register	Callee
x10-11	a0-1	Function Arguments/return values	Caller
x12-17	a2-7	Function arguments	Caller
x18-27	s2-11	Saved registers	Callee
x28-31	t3-6	Temporaries	Caller

- In addition, RISC-V has a set of Control & Status Registers (CSRs)