Introduction

Week 01 – Lecture 2

Team

Instructors

Giancarlo Succi

Joseph Brown

Teaching Assistants

Vladimir Ivanov

Stanislav Litvinov

Alexey Reznik

Munir Makhmutov

Hamna Aslam

Sources

- These slides have been adapted from the original slides of the adopted book:
  - Tanenbaum & Bo, Modern Operating Systems: 4th edition, 2013
    Prentice-Hall, Inc.

  and customised for the needs of this course.
- Additional input for the slides are detailed later

Components of a Modern Computer (1)

- One or more processors
- Main memory
- Disks
- Printers
- Keyboard
- Mouse
- Display
- Network interfaces
- I/O devices

Components of a Modern Computer (2)

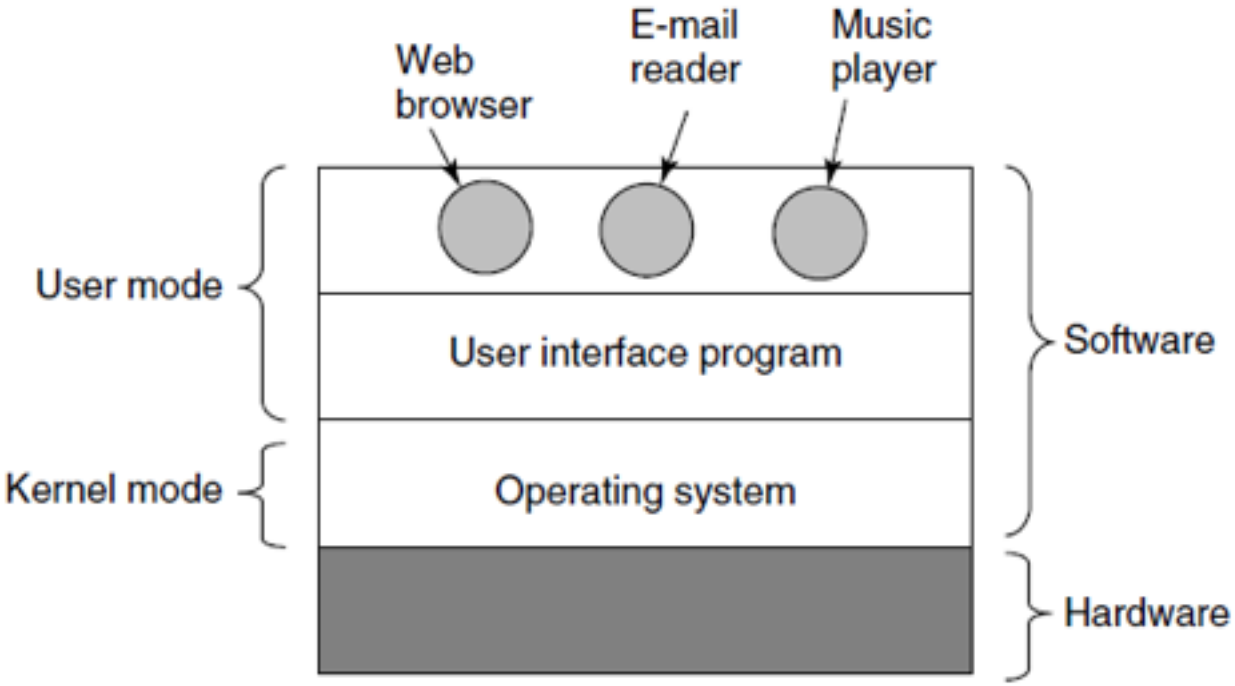

Figure 1-1. Where the operating system fits in.

Operating System's Function

- Managing all the components and using them optimally is an exceedingly challenging job
- For this reason, computers are equipped with a layer of software called the **operating system**
- Operating system's job is to provide user programs with a better, simpler, cleaner, model of the computer and to handle managing all the resources mentioned earlier

Kernel Mode & User Mode (1)

- Most computers have two modes of operation: **kernel mode** and **user mode**
- The operating system runs in kernel mode (also called **supervisor mode**):
  - has complete access to all the hardware
  - can execute any instruction the machine is capable of executing

Kernel Mode & User Mode (2)

- The rest of the software runs in user mode:
  - only a subset of the machine instructions is available
  - those instructions that affect control of the machine or do I/O (Input/Output) are forbidden to user-mode programs

The Operating System as an Extended Machine (1)

- Problem: hardware is very complicated and present difficult and inconsistent interfaces to the software developers
- Solution: use **abstractions**, for instance, a disk driver, that deals with the hardware and provides an interface to read and write disk blocks
- Another example: files. Using this abstraction, programs can create, write, and read files, without having to deal with hard disk interface

The Operating System as an Extended Machine (2)

- By different reasons the hardware designers often do not realize (or care) how much trouble they are causing for the software
- One of the major tasks of the operating system is to hide the hardware and present programs (and their programmers) with nice, clean, elegant, consistent, abstractions to work with instead

The Operating System as an Extended Machine (3)

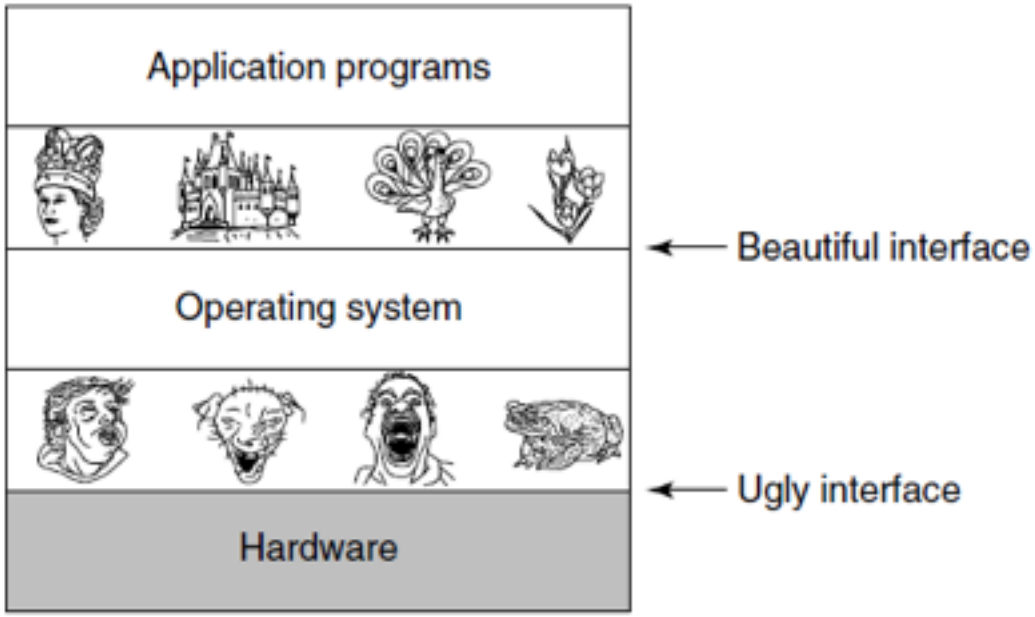

Figure 1-2. Operating systems turn ugly hardware into beautiful abstractions.

The Operating System as a Resource Manager (1)

- Top down view
  - Provide abstractions to application programs which are the ones who deal directly with the operating system and its abstractions
  - Consider the normal Windows desktop and the line-oriented command prompt. Both are programs running on the Windows OS and use the abstractions Windows provides
  - However, the underlying operating system abstractions are the same in both cases

The Operating System as a Resource Manager (2)

- Bottom up view
  - Manage pieces of complex system
  - The job of the operating system is to provide for an orderly and controlled allocation of the processors, memories, and I/O devices among the various programs wanting them
  - This is called **resource management**

The Operating System as a Resource Manager (3)

- Resource management includes **multiplexing** (sharing) resources in two different ways: in time and in space
  - When a resource is **time multiplexed**, different programs or users take turns using it (example: CPU)
  - **Space multiplexing** means that instead of the customers taking turns, each one gets part of the resource (example: memory)

History of Operating Systems

- The first generation (1945–55) vacuum tubes
- The second generation (1955–65) transistors and batch systems
- The third generation (1965–1980) ICs and multiprogramming
- The fourth generation (1980–present) personal computers
- The fifth generation (1990–present) mobile computers

One of the first programs for a loom

Moore’s Law

Apple Macintosh Computer

Computer Hardware (1)

- An operating system is tied to the hardware of the computer it runs on.
- It extends the computer's instruction set and manages its resources
- Conceptually, a simple personal computer can be abstracted to the next model (Fig. 1-6):
  - the CPU, memory, and I/O devices are all connected by a system bus and communicate with one another over it
- More details on modern personal computers that have a more complicated structure, involving multiple buses will be discussed later

Computer Hardware (2)

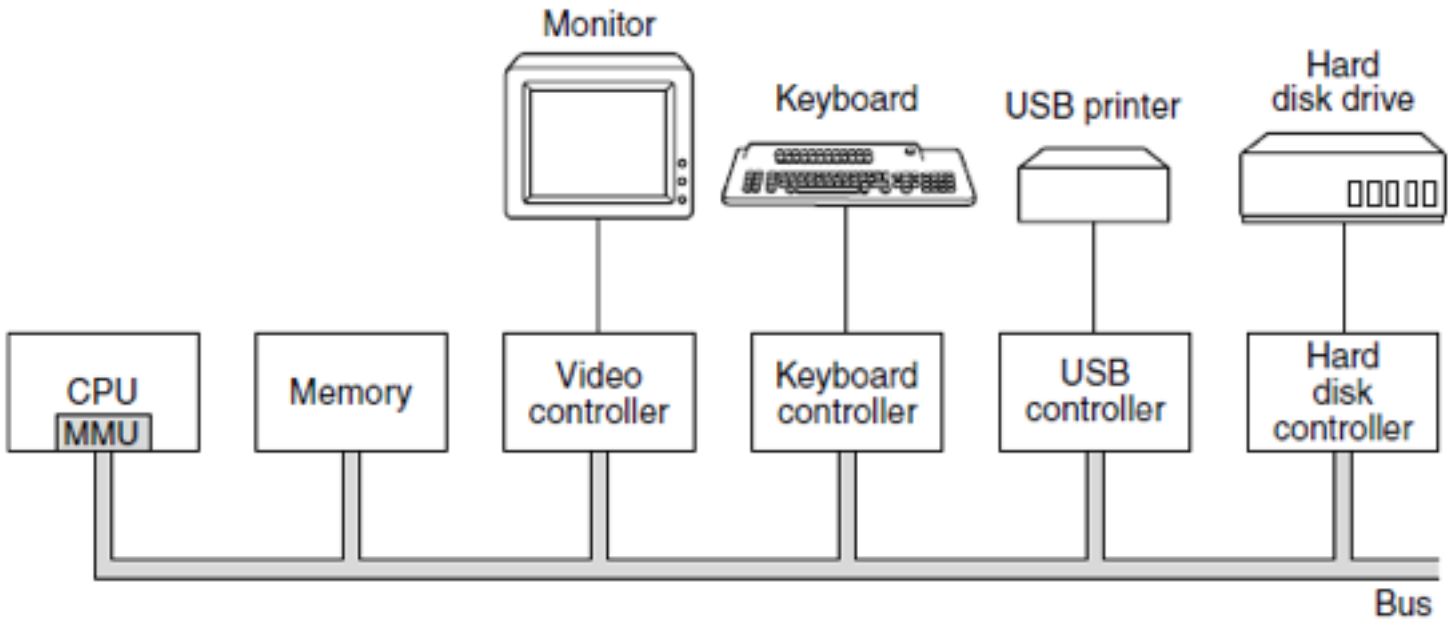

Figure 1-6. Some of the components of a simple personal computer.

Processors (1)

- The “brain” of the computer is the CPU which fetches instructions from memory and executes them
- The basic cycle of every CPU:
  - fetch the first instruction from memory
  - decode it to determine its type and operands
  - execute it
  - fetch, decode, and execute subsequent instructions
- The cycle is repeated until the program finishes
- Each CPU has a specific set of instructions that it can execute (x86 and ARM are not compatible)

Processors (2)

- Accessing memory to get an instruction or data word takes much longer than executing an instruction, all CPUs contain some general registers to hold key variables and temporary results
- Most computers have several special registers:
  - **program counter** contains the memory address of the next instruction to be fetched. After that instruction has been fetched, the program counter is updated to point to its successor
  - **stack pointer** points to the top of the current stack in memory. The stack contains one frame for each procedure that has been entered but not yet exited
  - **PSW (Program Status Word)** contains the condition code bits, which are set by comparison instructions, the CPU priority, the mode (user or kernel), and various other control bits

Processors (3)

- When time multiplexing the CPU, the OS will often stop the running program to (re)start another one
- Every time it stops a running program, the operating system must save all the registers
- In order to improve performance, **pipeline** design was developed:
  - A CPU might have separate fetch, decode, and execute units. While it is executing instruction *n*, it could also be decoding instruction *n + 1* and fetching instruction *n + 2* (as shown on Fig. 1-7a)

Processors (4)

- A **superscalar** CPU design (Fig. 1-7b) is more advanced:
  - multiple execution units are present, for example, one for integer arithmetic, one for floating-point arithmetic, and one for Boolean operations
  - two or more instructions are fetched at once, decoded, and dumped into a holding buffer
  - execution unit looks in the holding buffer to see if there is an instruction it can handle, and if so, it removes the instruction from the buffer and executes it

Processors (5)

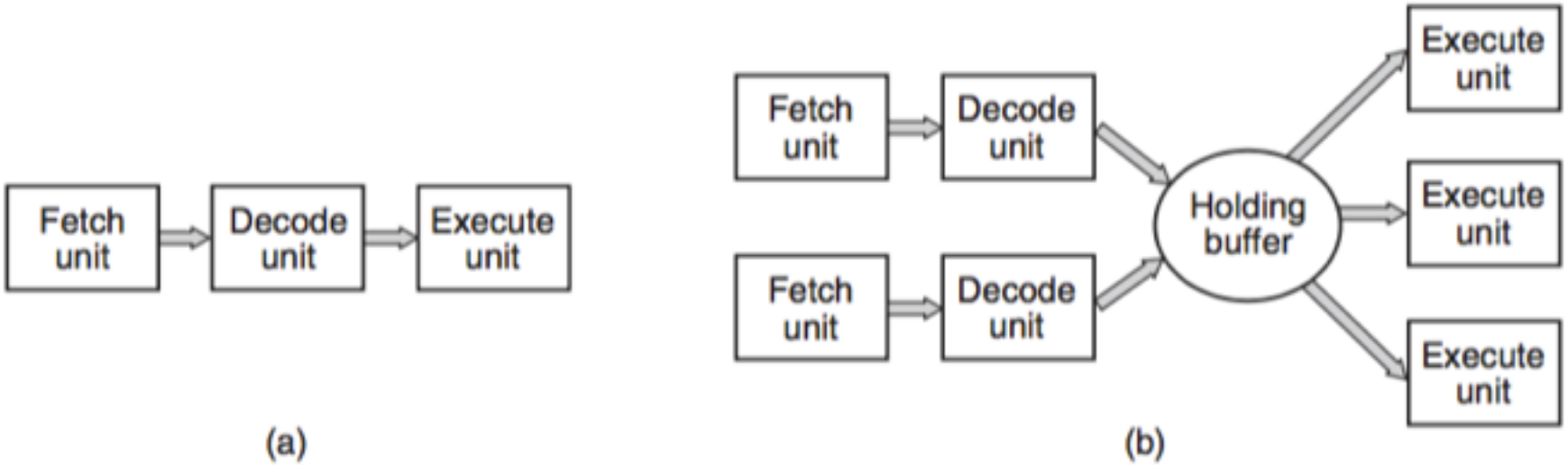

Figure 1-7. (a) A three-stage pipeline. (b) A superscalar CPU.

Processors (6)

- Most CPUs have two modes: **kernel mode** and **user mode**
- Usually, a bit in the PSW controls the mode. In kernel mode the CPU can execute every instruction in its instruction set and use every feature of the hardware
- User programs always run in user mode, which permits only a subset of the instructions to be executed and a subset of the features to be accessed.
- Generally, all instructions involving I/O and memory protection are disallowed in user mode
- To obtain services from the OS, a user program must make a **system call (TRAP)**, which traps into the kernel and invokes the OS

Multithreaded and Multicore Chips (1)

- The Intel Pentium 4 introduced the property, called **multithreading** or **hyperthreading**, to replicate not only the functional units of CPU, but also some of the control logic
- It allows the CPU to hold the state of two different threads and then switch between them in nanoseconds
- If one of the processes needs to read a word from memory (long operation), a multithreaded CPU can just switch to another thread thus saving time
- Multithreading does not offer true parallelism

Multithreaded and Multicore Chips (2)

- Many **CPU chips** now have four, eight, or more complete processors or **cores** (Fig. 1-8)
- Making use of such a multicore chip will require a multiprocessor OS
- A modern **GPU (Graphics Processing Unit)** is a processor with, thousands of tiny cores which are very good for many small computations done in parallel, like rendering polygons in graphics applications
- However, they are not so good at serial tasks and hard to program

Multithreaded and Multicore Chips (3)

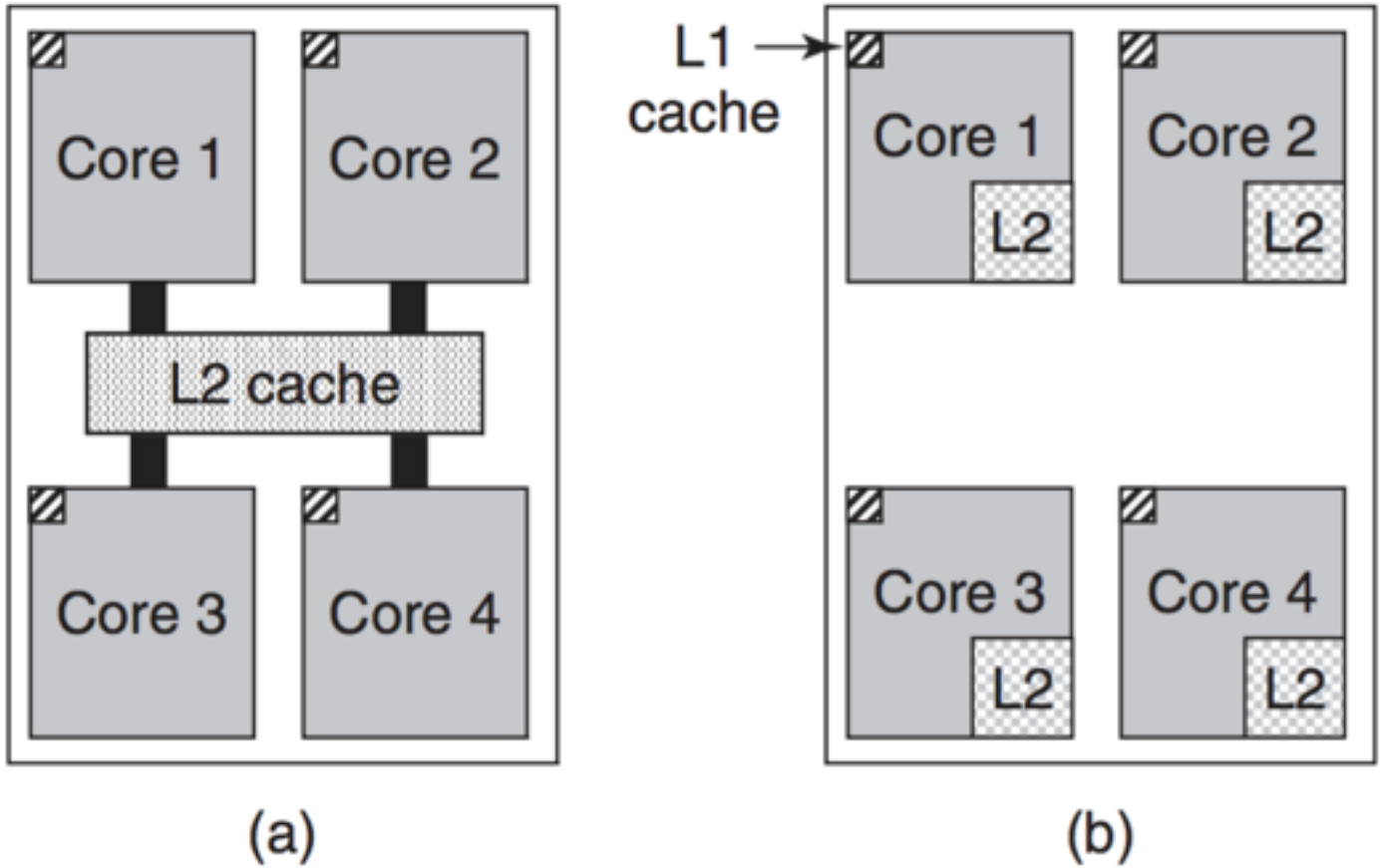

Figure 1-8. (a) A quad-core chip with a shared L2 cache. (b) A quad-core chip with separate L2 caches.

Memory (1)

- Ideally, a memory should be extremely fast, so that the CPU is not held up by it, large and cheap
- No current technology satisfies all of these goals
- The memory system is constructed as a hierarchy of layers (Fig. 1-9)
- The top layers have higher speed, smaller capacity, and greater cost per bit than the lower ones

Memory (2)

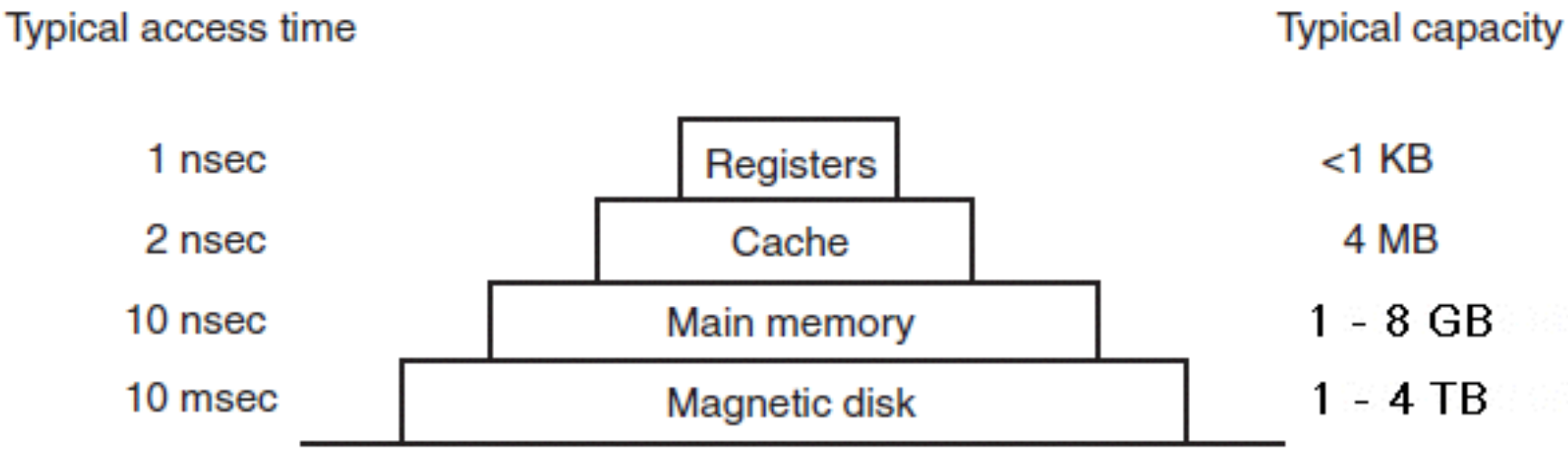

Figure 1-9. A typical memory hierarchy. The numbers are very rough approximations.

Memory (3)

- The top layer consists of the **registers** internal to the CPU which are just as fast as the CPU.
- There is no delay in accessing them.
- The storage capacity available in them is typically 32 × 32 bits on a 32-bit CPU and 64 × 64 bits on a 64-bit CPU (less than 1 KB in both cases)
- Programs must manage the registers (i.e., decide what to keep in them) themselves

Memory (4)

- **Cache memory** is mostly controlled by the hardware
- Main memory is divided up into **cache lines**, typically 64 bytes, with addresses 0 to 63 in cache line 0, 64 to 127 in cache line 1, and so on. The most heavily used cache lines are kept in a high-speed cache located inside or very close to the CPU
- When the program needs to read a memory word, the cache hardware checks to see if the line needed is in the cache. If it is (a **cache hit**), no memory request is sent over the bus to the main memory
- Cache hits normally take about two clock cycles

Memory (5)

- Cache memory is limited in size due to its high cost
- Some machines have two or even three levels of cache, each one slower and bigger than the one before it
- Caching system issues:
  - When to put a new item into the cache
  - Which cache line to put the new item in
  - Which item to remove from the cache when a slot is needed
  - Where to put a newly evicted item in the larger memory

Memory (6)

- Main memory:
  - **RAM (Random Access Memory)** serves all CPU requests that cannot be satisfied out of the cache
  - **ROM (Read Only Memory)** does not lose its contents when the power is switched off, is programmed at the factory and cannot be changed afterward. On some computers, the bootstrap loader used to start the computer is contained in ROM
  - **EEPROM (Electrically Erasable Programmable PROM)** and **flash memory** are also nonvolatile, but in contrast to ROM can be erased and rewritten
  - **Flash memory** is also commonly used as the storage medium in portable electronic devices. It is intermediate in speed between RAM and disk and it wears out
  - Many computers use **CMOS (complementary metal-oxide-semiconductor) memory** to hold the current time and date and the configuration parameters, such as which disk to boot from

Disks (1)

- Disk storage (**magnetic disk** or **hard disk**) much cheaper than RAM per bit and also larger.
- Since the disk is mechanical device (Fig. 1-10), the time to randomly access data on it is close to three orders of magnitude slower
- A disk consists of one or more metal rotating platters
- Information is written onto the disk in a series of concentric circles. At any given arm position, each of the heads can read an annular region called a **track**
- Together, all the tracks for a given arm position form a **cylinder**
- Each track is divided into some number of **sectors**, typically 512 bytes per sector

Disks (2)

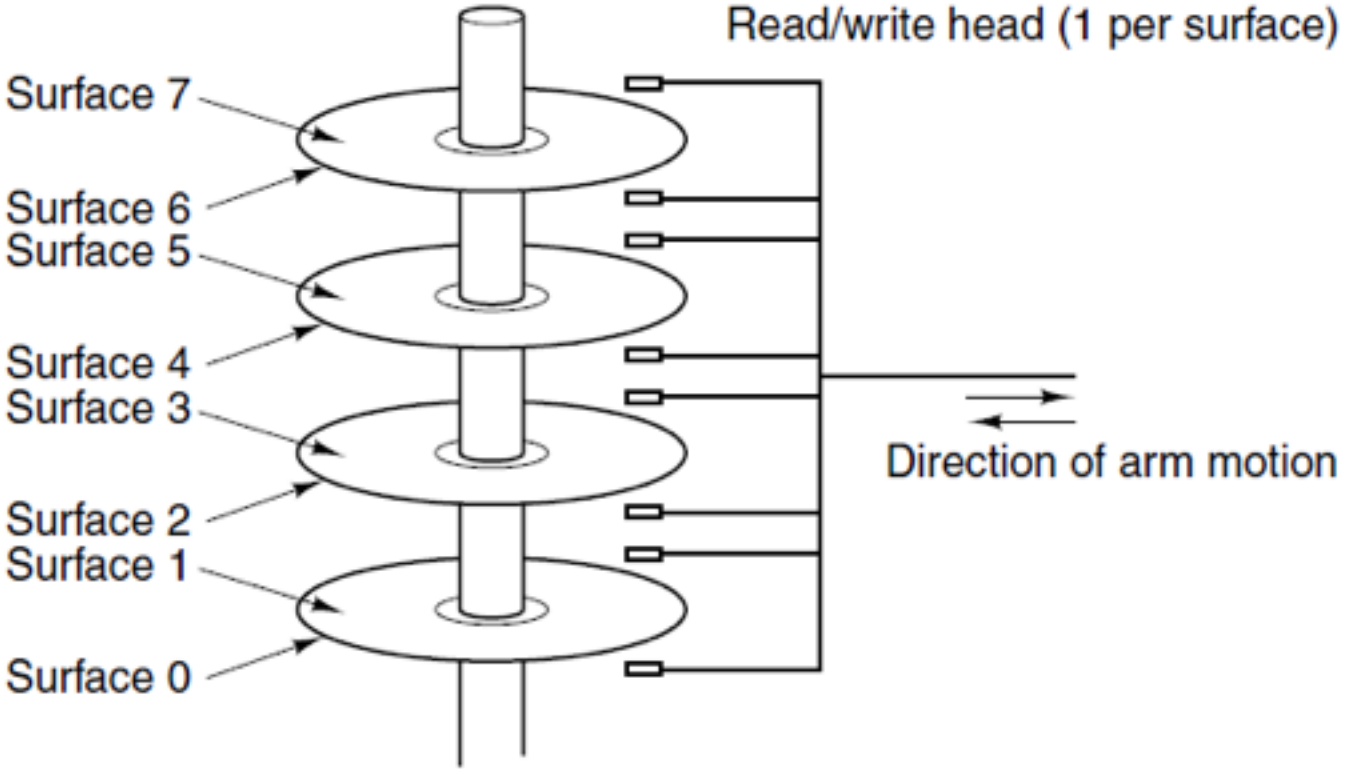

Figure 1-10. Structure of a disk drive.

Disks (3)

- **SSDs (Solid State Disks)** do not have moving parts, do not contain platters in the shape of disks, and store data in (Flash) memory
- **Virtual memory** makes it possible to run programs larger than physical memory by placing them on the disk and using main memory as a kind of cache
- This requires remapping memory addresses on the fly to convert the address the program generated to the physical address in RAM where the word is located. This mapping is done by a part of the CPU called the **MMU (Memory Management Unit)**

I/O Devices (1)

- I/O devices generally consist of two parts: a **controller** and the **device** itself
- The controller is a chip or a set of chips that physically controls the device. It accepts commands from the OS and carries them out
- Devices have fairly simple interfaces, both because they cannot do much and to make them standard. The latter is needed so that any SATA disk controller can handle any SATA disk, for example
- The software that talks to a controller, giving it commands and accepting responses, is called a **device driver**

I/O Devices (2)

- To be used, the driver has to be put into the OS so it can run in kernel mode. There are three ways:
  - to relink the kernel with the new driver and then reboot the system (UNIX)
  - make an entry in an operating system file telling it that it needs the driver and then reboot the system. At boot time, the OS goes and finds the drivers it needs and loads them (Windows)
  - make the OS to be able to accept new drivers while running and install them on the fly without the need to reboot. Hot-pluggable devices, such as USB and IEEE 1394 devices, always need dynamically loaded drivers

I/O Devices (3)

- Every controller has a small number of registers that are used to communicate with it
- To activate the controller, the driver gets a command from the OS, then translates it into the appropriate values to write into the device registers
- The collection of all the device registers forms the **I/O port space**

I/O Devices (4)

- On some computers, the device registers are mapped into the OS's address space (the addresses it can use), so they can be read and written like ordinary memory words
- On other computers, the device registers are put in a special I/O port space, with each register having a port address. On these machines, special IN and OUT instructions are available in kernel mode to allow drivers to read and write the registers

I/O Devices (5)

- Input and output can be done in three different ways:
  - **busy waiting:** the driver starts the I/O and sits in a tight loop continuously polling the device to see if it is done
  - **interrupts:** the driver starts the device and asks it to give an interrupt when it is finished (Fig. 1-11)
  - **DMA (Direct Memory Access):** DMA chip can control the flow of bits between memory and a controller without constant CPU intervention. The CPU sets up the DMA chip, telling it how many bytes to transfer, the device and memory addresses involved, and the direction. When the DMA chip is done, it causes an interrupt

I/O Devices (6)

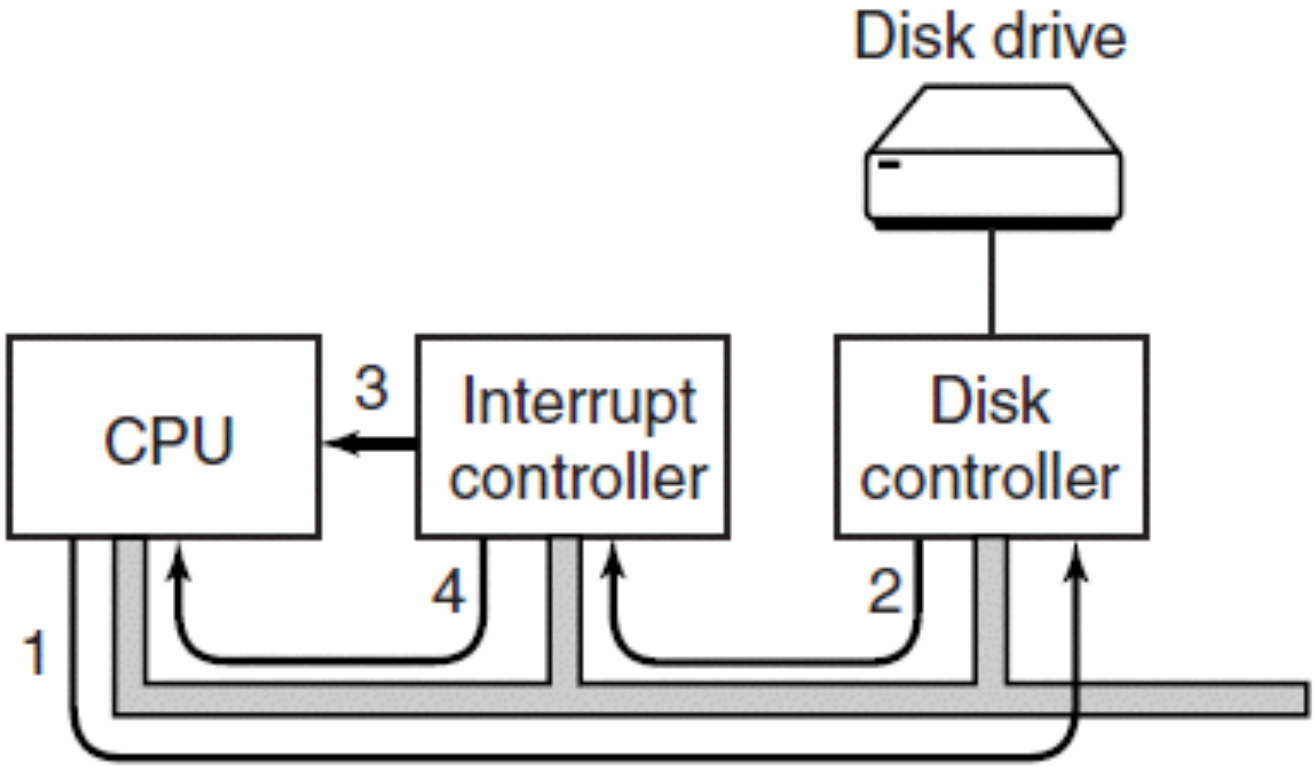

Figure 1-11. (a) The steps in starting an I/O device and getting an interrupt.

I/O Devices (7)

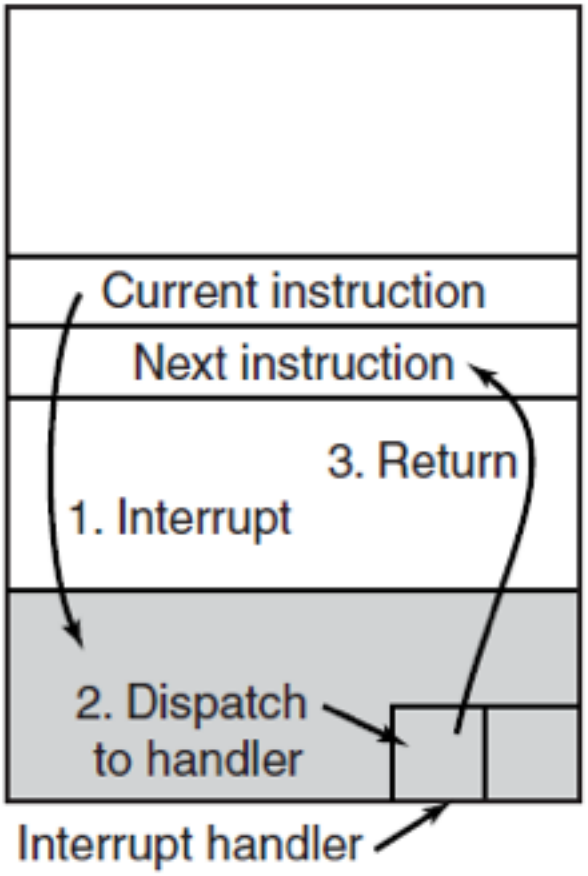

Figure 1-11. (b) Interrupt processing involves taking the interrupt, running the interrupt handler, and returning to the user program.

Buses (1)

- At some point additional buses were added, both for faster I/O devices and for CPU-to-memory traffic (Fig. 1-12)
- This system has many buses (e.g., cache, memory, PCIe, PCI, USB, SATA, and DMI), each with a different transfer rate and function
- The OS must be aware of all of them for configuration and management. The main bus is the **PCIe (Peripheral Component Interconnect Express) bus**
- Up to its creation in 2004, most buses were **parallel** and **shared**
- PCIe makes use of dedicated, point-to-point connections

Buses (2)

- A **shared bus architecture** means that multiple devices use the same wires to transfer data which needs an arbiter to determine who can use the bus
- A **parallel bus architecture** as used in traditional PCI means that you send each word of data over multiple wires. For instance, in regular PCI buses, a single 32-bit number is sent over 32 parallel wires
- PCIe uses a **serial bus architecture** and sends all bits in a message through a single connection, known as a **lane**. Parallelism is still used, because you can have multiple lanes in parallel (send 32 messages via 32 lanes)

This content is for your curiosity only. It will be explained in details later.

Buses (3)

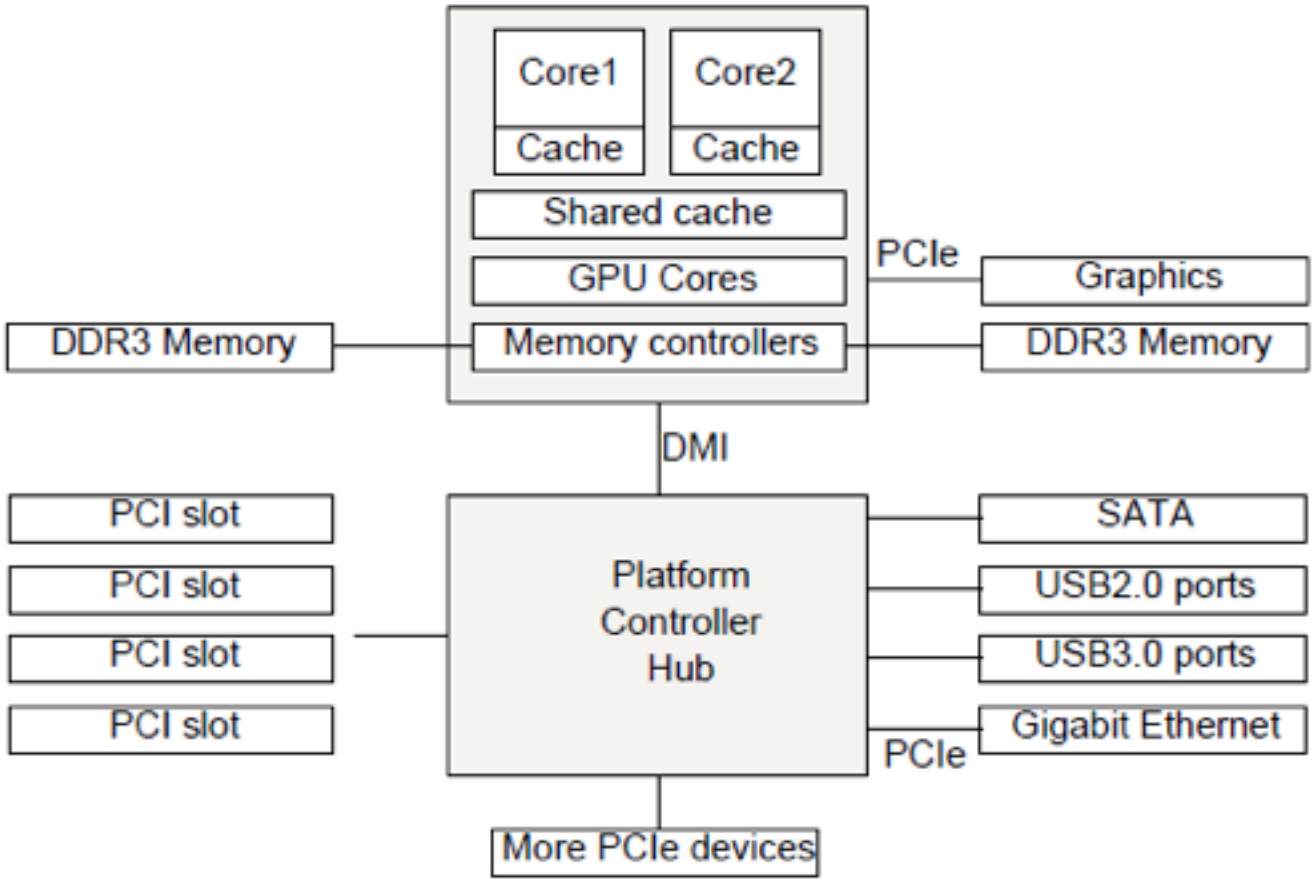

Figure 1-12. The structure of a large x86 system

Buses (4)

- The CPU talks to memory over a fast **DDR3 (Double Data Rate 3)** bus, to an external graphics device over **PCIe** and to all other devices via a hub over a **DMI (Direct Media Interface) bus**. The hub in turn connects all the other devices
- The **USB (Universal Serial Bus)** was invented to attach all the slow I/O devices, such as the keyboard and mouse, to the computer. USB is a centralized bus in which a root device polls all the I/O devices every 1 msec to see if they have any traffic
- The **SCSI (Small Computer System Interface)** bus is a high-performance (up to 640 MB/sec) bus intended for fast disks, scanners, and other devices needing considerable bandwidth

Buses (5)

- **Plug and play:** system that allows the OS to know what peripheral devices are connected to the computer and configure them:
  - based on a similar concept first implemented in the Apple Macintosh
  - before plug and play, each I/O card had a fixed interrupt request level and fixed addresses for its I/O registers (keyboard: interrupt 1 and I/O addresses 0x60 to 0x64, floppy disk controller: interrupt 6 and I/O addresses 0x3F0 to 0x3F7, and so on)
  - **problem: two different pieces of hardware might use the same interrupt, so they will conflict**
  - plug and play makes the system automatically collect information about the I/O devices, centrally assign interrupt levels and I/O addresses, and then tell each card what its numbers are

Bootstrap

Source: http://www.xda-developers.com/wp-content/uploads/2012/06/Bootstrap.jpg#Boot%20Strap%20340x400

Booting The Computer (1)

- The system **BIOS (Basic Input Output System)** contains low-level I/O software, including procedures to read the keyboard, write to the screen, do disk I/O, etc.
- After the BIOS is started it first checks to see how much RAM is installed and whether the keyboard and other basic devices are installed and responding correctly
- It starts out by scanning the PCIe and PCI buses to detect all the devices attached to them
- Then it determines the boot device by trying a list of devices stored in the CMOS memory

Booting The Computer (2)

- The first sector from the boot device is read into memory and executed. This sector contains a program that normally examines the partition table at the end of the boot sector to determine which partition is active
- A secondary boot loader is read in from that partition. This loader reads in the OS from the active partition and starts it
- The OS then queries the BIOS to get the configuration information. For each device, it checks to see if it has the device driver
- Once it has all the device drivers, the OS loads them into the kernel. Then it initializes its tables, creates whatever background processes are needed, and starts up a login program or GUI

The Operating System Zoo

- Mainframe Operating Systems
- Server Operating Systems
- Multiprocessor Operating Systems
- Personal Computer Operating Systems
- Handheld Computer Operating Systems
- Embedded Operating Systems
- Sensor Node Operating Systems
- Real-Time Operating Systems
- Smart Card Operating Systems

Mainframe Operating Systems

- Mainframes are room-sized computers still found in major corporate data centers.
- The operating systems for mainframes are heavily oriented toward processing many jobs at once, most of which need prodigious amounts of I/O.
- They typically offer three kinds of services: batch, transaction processing, and timesharing.

Mainframe Operating Systems

- A batch system is one that processes routine jobs without any interactive user present
- Transaction-processing systems handle large numbers of small requests, for example, check processing at a bank or airline reservations
- Timesharing systems allow multiple remote users to run jobs on the computer at once, such as querying a big database

Server Operating Systems

- They run on servers, which are either very large personal computers, workstations, or even mainframes
- Typical server operating systems are Solaris, FreeBSD, Linux and Windows Server 201x

Multiprocessor Operating Systems

- The systems that have multiple CPUs are called as parallel computers, multi-computers, or multiprocessors
- Many popular operating systems, including Windows and Linux, run on multiprocessors

Personal Computer Operating Systems

- Common examples are Linux, FreeBSD, Windows 7, Windows 8, and Apple's OS X

Handheld Computer Operating Systems

- A handheld computer, originally known as a PDA (Personal Digital Assistant), is a small computer that can be held in your hand during operation
- Handheld computers are dominated by Google's Android and Apple's iOS

Embedded Operating Systems

- Examples of embedded system devices are microwave ovens, TV sets, cars, DVD recorders, traditional phones, and MP3 players
- OSs such as Embedded Linux, QNX and VxWorks are popular in this domain

Sensor-Node Operating Systems

- Networks of tiny sensor nodes are being deployed for numerous purposes
- These nodes are tiny computers that communicate with each other and with a base station using wireless communication
- Each sensor node is a real computer, with a CPU, RAM, ROM, and one or more environmental sensors
- TinyOS is a well-known operating system for a sensor node

Real-Time Operating Systems

- These systems are characterized by having time as a key parameter
- A **hard real-time** system must provide absolute guarantees that a certain action will occur by a certain time

Real-Time Operating Systems

- A **soft real-time** system, is one where missing an occasional deadline, while not desirable, is acceptable and does not cause any permanent damage. Such as smart phones
- Sometimes the operating system is simply a library linked in with the application programs

Smart Card Operating Systems

- The smallest operating systems run on smart cards, which are credit-card-sized devices containing a CPU chip
- Some smart cards are Java oriented. This means that the ROM on the smart card holds an interpreter for the Java Virtual Machine (JVM). Java applets (small programs) are downloaded to the card and are interpreted by the JVM interpreter

End

Week 01 – Lecture 2

References

- Tanenbaum & Bo, Modern Operating Systems: 4th edition, 2013 Prentice-Hall, Inc.
- Images on slides 16-18 are courtesy of Vladimir Ivanov.