1 First Meeting

- CENG 328 Operating Systems Spring 2011
- THURSDAY 10:40-12:30 (T1) B301/302
- FRIDAY 08:40-10:30 (T2) B308/309
- TUESDAY 12:40-14:30 (L1) MPLab
- FRIDAY 12:40-14:30 (L2) MPLab
- FRIDAY 14:40-16:30 (L3) MPLab
- Instructor: Cem Özdoğan, Department of Materials Science and Engineering, A318
- TA: Efe Çiftçi
- WEB page:
  http://siber.cankaya.edu.tr/ozdogan/OperatingSystems/spring2011/index.html
- Announcements: Watch this space for the latest updates.

  ```
  Pazar 13.Subat.2011 23:42 In the first lecture,
  there will be first metting and Introduction/
  Overview. The laboratory notes for the first
  week is published, see Course Schedule section.
  ```

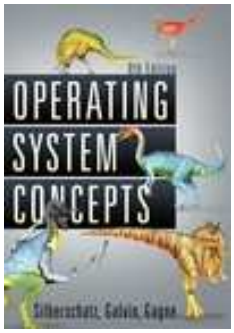
- Important announcements will be posted to the **Announcements section of the web page**, so please check this page frequently.
- You are responsible for all such announcements, as well as announcements made in lecture.
- All/Some the example c-files (for lecturing and lab. sessions) will/may be accessible via the link.
- The tutorial link is active.
- Anyone wants to get a live CD without installing linux, download from local server.
  - Ubuntu live CD
  - Pardus live CD

1.1 Lecture Information

- There are two groups for lecturing, you may attend any one of the lecture hours.
- But, "Please" attend your predefined sessions regularly.
- You will be expected to do significant programming assignments, as well as run programs we supply and analyse the output.
- These programs will be written in C programming language. For programming assignments, other languages will be accepted (such as Java, C++, but no programming assistance will be given).
- The UNIX operating system will be introduced to you first in the lab sessions.
- You MAY have quizzes (10-15 minutes, may be less; but not scheduled as before) for **the previous lecture/chapter's subjects**.
- There won't be any make-up for these quizzes.

1.2 Overview

- Ceng 328 is intended as a general introduction to the techniques used to implement operating systems and related kinds of systems software.
- Among the topics covered will be;
  - basic operating system structure,
  - process and thread synchronization,
  - process scheduling and resource management,
  - process management (creation, synchronization, and communication),
  - memory management techniques, main-memory management, virtual memory management,
  - file-system structure,
  - control of disks and other input/output devices,
  - deadlock prevention, avoidance, and recovery.
- This course assumes familiarity with basic computer organization (e.g., processors, memory, and I/O devices).



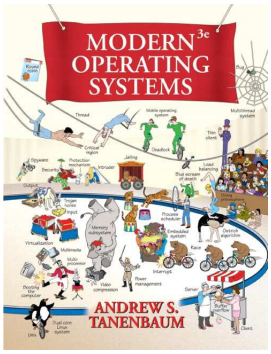


Figure 1: Required and Recommended

1.3 Text Book

- **Required:** Readings will be assigned in Operating System Concepts, 8th Edition by Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, John Wiley and Sons, January 2008.
- **Recommended:** Modern Operating Systems, 3rd Edition by Andrew S. Tanenbaum, Prentice Hall, 2008. Another frequently used text book that covers the same material with a different approach

1.4 Grading Criteria & Policies

- There will be a midterm and a final exam, will count 20% and 40% of your grade, respectively.
- Quiz: 15% (worst of the quizzes will be discarded).
- Assignments (or Term Project): 15%.
- Attendance is required and constitutes part of your course grade; 10%. Attendance is not compulsory, but you are responsible for everything said in class.
- I encourage you to ask questions in class. You are supposed to ask questions. Don't guess, ask a question!
- The code/homework you submit must be written completely by you. You can use anything from the textbook/notes with a clear understanding.

2 Introduction/Overview

An operating system (OS) acts as an intermediary between the user of a computer and the computer hardware. The purpose of an OS is to provide an environment in which a user can execute programs in a **convenient** and **efficient** manner.

2.1 What Is An Operating System?

- An OS is software that manages the computer hardware.
  - *Mainframe OSs* are designed primarily to optimize utilization of hardware.
  - *Personal computer (PC) OSs* support complex games, business applications, and everything in between.
  - *OSs for handheld computers* are designed to provide an environment in which a user can easily interface with the computer to execute programs.
- Thus, some OSs are designed to be convenient, others to be efficient, and others some combination of the two.

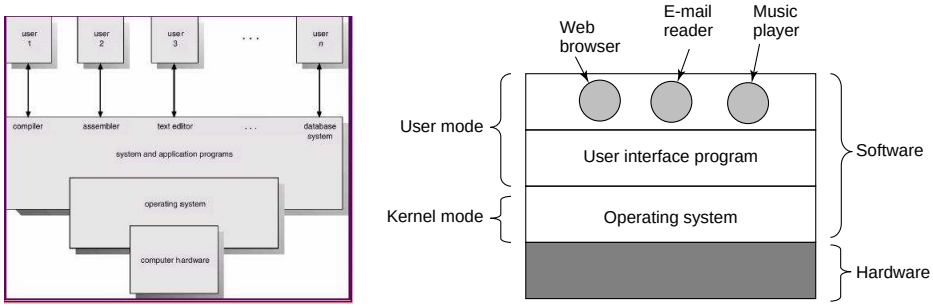


Figure 2: Abstract view. Where the OS fits in.

- A computer system can be divided roughly into four components: *the hardware*, *the OS*, the *application programs*, and *the users* (see Fig. 2).
  1. *Hardware*
     - Electronic, mechanical, optical devices.

- The central processing unit (CPU), the memory, and the input/output (I/O) devices-provides the basic computing resources for the system.
- The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent user programs from interfering with the proper operation of the system.

2. *Software*
   - Programs. Without support software, a computer is of little use. With its *software*, however, a computer can store, manipulate, and retrieve information.
   - Software can be grouped into the following categories:
     * *systems software* (OS & utilities)
     * *applications software* (user programs; word processors, spreadsheets, compilers, database systems, games, web browsers etc.)

- As a summary;
  - Hardware provides basic computing resources (CPU, memory, I/O devices).
  - OS controls and coordinates the use of the hardware among the various application programs for the various users.
  - Provides orderly and controlled allocation (i.e., *sharing, optimization of resource utilization*) and use of the resources by the users (jobs) that *compete* for them.
- An OS is similar to a government. Like a government, it performs no useful function by itself. It simply provides an environment within which other programs can do useful work.

2.1.1 User View - The OS as an Extended Machine

- An OS
  - provides an abstraction layer over the concrete hardware,
  - use the computer hardware in an efficient manner (converting hardware into useful form),
  - "hide" the complexity of the underlying hardware. See Fig. 3

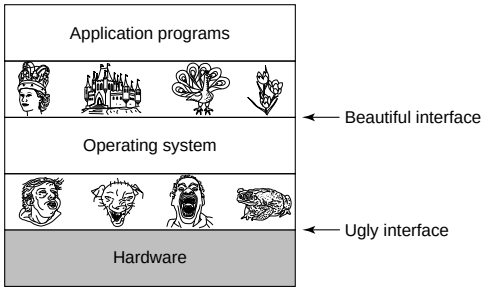


Figure 3: Operating systems turn ugly hardware into beautiful abstractions.

- Most computer users sit in front of a PC, consisting of a monitor, keyboard, mouse, and system unit. Such a system is designed for one user to monopolize its resources.
- The goal is to maximize the work (or play) that the user is performing. In this case, the OS is designed mostly for ease of use.
- Performance is, of course, important to the user; but rather than resource utilization, such systems are optimized for the single-user experience.
- In other cases, a user sits at a terminal connected to a *mainframe* or *minicomputer*. Other users are accessing the same computer through other terminals. These users share resources and may exchange information.
- The OS in such cases is designed to maximize resource utilization to assure that all available CPU time, memory, and I/O are used efficiently and that no individual user takes more than her *fair share*.
- In still other cases, users sit at workstations connected to networks of other workstations and servers. These users have dedicated resources at their disposal, but they also share resources such as networking and servers-file, compute, and print servers.
- Therefore, their OS is designed to compromise between individual usability and resource utilization.
- Recently, many varieties of handheld computers have come into fashion. Most of these devices are standalone units for individual users.

- Their OSs are designed mostly for individual usability, but performance per amount of battery life is important as well.
- Some computers have little or no user view. For example, embedded computers in home devices and automobiles may have numeric keypads and may turn indicator lights on or off to show status, but they and their OSs are designed primarily to run without user intervention.

2.1.2 System View - The OS as a Resource Manager

- From the computer's point of view, the OS is the program most intimately involved with the hardware. In this context, we can view an OS as a resource allocator.
- Resource - "Something valuable" e.g. CPU time, memory space (RAM), file-storage space, I/O devices (disk), and so on.
- The OS acts as the manager of these resources. Includes multiplexing (sharing) resources in two different ways. Each program gets
  - **time** with the resource
  - **space** on the resource
- Multiple users/applications can share, why share:
  - devices are expensive,
  - there is need to share data as well as communicate
- Facing numerous and possibly conflicting requests for resources, the OS must decide
  - how to allocate them to specific programs (processes, jobs)
  - how to protect applications from one another,
  - how to provide fair and efficient access to resources,
  - how to operate and control the various I/O devices.

2.1.3 Defining OS and Functionalities

- In general, we have no completely adequate definition of an OS.
- The fundamental goal of computer systems is to execute user programs and to make solving user problems easier.

- Since bare hardware alone is not particularly easy to use, application programs are developed. These programs require certain common operations, such as those controlling the I/O devices.
- The common functions of controlling and allocating *resources* are then brought together into one piece of software: the OS.
- In addition, we have no universally accepted definition of what is part of the OS. A more common definition is that the OS is the one program running at all times on the computer (usually called the kernel), with all else being systems programs and application programs.
- OS cannot help all the people all the time, but it should help most of the people most of the time.
  - What mechanisms?
  - What policies?
- *Challenges:* Desired functionalities of OS depend on outside factors like users' & application's "Expectations" and "Technology changes" in Computer Architecture (hardware). *OS must adapt:*
  - change abstractions provided to users,
  - change algorithms to change these abstractions,
  - change low-level implementation to deal with hardware.
- The current OSs are driven by such evolutions.

2.2 Computer-System Organization

Before we can explore the details of how computer systems operate, we need a general knowledge of the structure of a computer system.

2.2.1 Computer-System Operation

- A modern general-purpose computer system consists of one or more CPUs and a number of device controllers connected through a common bus that provides access to shared memory (see Fig. 4).
- Each device controller is in charge of a specific type of device (for example, disk drives, audio devices, and video displays).

- The CPU and the device controllers can execute concurrently, *competing for memory cycles.*
- To ensure orderly access to the shared memory, a memory controller is provided whose function is to synchronize access to the memory.

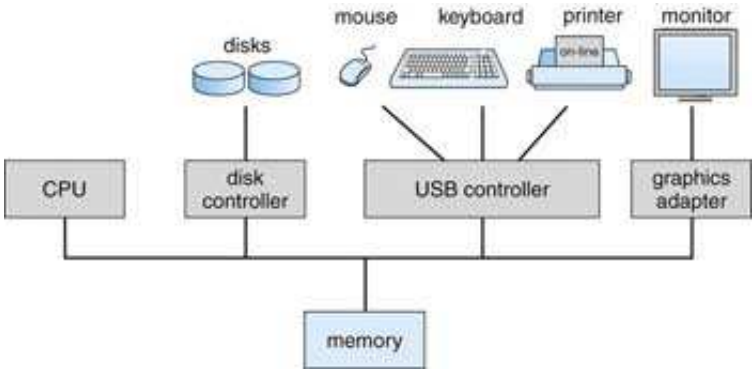


Figure 4: A modern computer system.

- For a computer to start running, when it is powered up or rebooted-it needs to have an initial program (bootstrap program) to run.
- Typically, it is stored in read-only memory (ROM) or electrically erasable programmable read-only memory (EEPROM), known by the general term firmware, within the computer hardware.
- It initializes all aspects of the system, from CPU registers to device controllers to memory contents.
- The bootstrap program must know how to load the OS and to start executing that system. To accomplish this goal, the bootstrap program must locate and load into memory the OS kernel.
- The process of initializing the computer and loading the OS is known as *bootstrapping* (see Fig. 5).

System Startup

- On power up
  - everything in system is in random, unpredictable state
  - special hardware circuit raises RESET pin of CPU
    - sets the program counter to 0xffffff0
      - this address is mapped to ROM (Read-Only Memory)
- BIOS (Basic Input/Output Stream)
  - set of programs stored in ROM
  - some OS's use only these programs
    - MS DOS
  - many modern systems use these programs to load other system programs
    - Windows, Unix, Linux

BIOS

- General operations performed by BIOS
  1) find and test hardware devices
     - POST (Power-On Self-Test)
  2) initialize hardware devices
     - creates a table of installed devices
  3) find *boot sector*
     - may be on floppy, hard drive, or CD-ROM
  4) load boot sector into memory location 0x00007c00
  5) sets the program counter to 0x00007c00
     - starts executing code at that address

Boot Loader

- Small program stored in boot sector
- Loaded by BIOS at location 0x00007c0
- Configure a basic file system to allow system to read from disk
- Loads kernel into memory
- Also loads another program that will begin kernel initialization

Initial Kernel Program

- Determines amount of RAM in system
  - uses a BIOS function to do this
- Configures hardware devices
  - video card, mouse, disks, etc.
  - BIOS may have done this but usually redo it
    - portability
- Switches the CPU from *real* to *protected* mode
  - real mode: fixed segment sizes, 1 MB memory addressing, and no segment protection
  - protected mode: variable segment sizes, 4 GB memory addressing, and provides segment protection
- Initializes paging (virtual memory)

Final Kernel Initialization

- Sets up page tables and segment descriptor tables
  - these are used by virtual memory and segmentation hardware (more on this later)
- Sets up interrupt vector and enables interrupts
- Initializes all other kernel data structures
- Creates initial process and starts it running
  - *init* in Linux
  - *smss* (Session Manager SubSystem) in NT

Figure 5: Booting the computer.

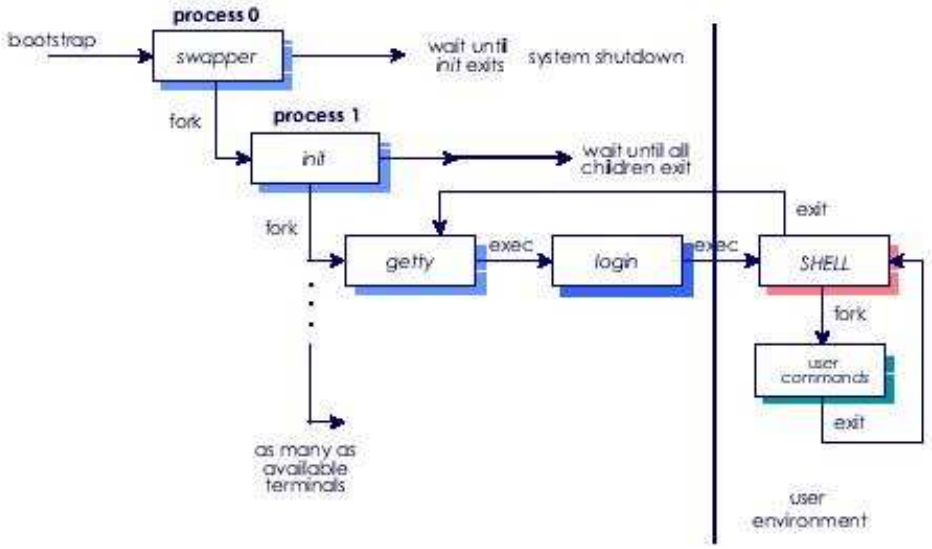


Figure 6: UNIX System initialization

- UNIX System initialization and Bootstrapping;
  - Once the kernel boots, we have a running Linux system. It isn't very usable, since the kernel doesn't allow direct interactions with "user space".
  - So, the system runs one program: **init** and waits for some event to occur. This program is responsible for everything else and is regarded as the father of all processes.
  - The kernel then retires to its rightful position as system manager handling "kernel space" (see Fig. 6).
  - Some portions of the OS remain in main memory to provide services for critical operations, such as dispatching, interrupt handling, or managing (critical) resources.
  - These portions of the OS are collectively called the *kernel*.

  Kernel = OS - transient components
  remains *comes and goes*

- The occurrence of an event is usually signaled by an interrupt from either the hardware or the software.

– Hardware may trigger an interrupt at any time by sending a signal to the CPU, usually by way of the system bus. Software may trigger an interrupt by executing a special operation called a **system call** (also called a monitor call).

- When the CPU is interrupted, it stops what it is doing and immediately transfers execution to a fixed location.
- The fixed location usually contains the starting address where the service routine for the interrupt is located.
- The interrupt service routine executes; on completion, the CPU resumes the interrupted computation. A time line of this operation is shown in Fig. 7.

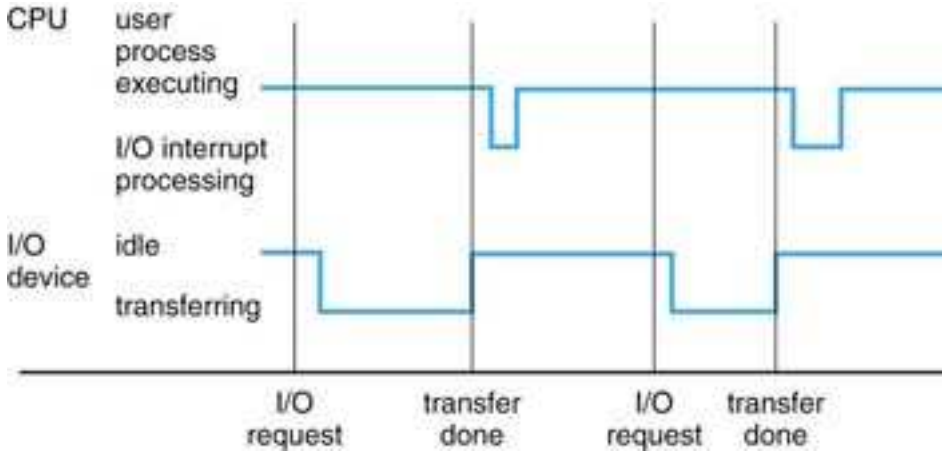


Figure 7: Interrupt time line for a single process doing output.

- Interrupts are an important part of a computer architecture. The interrupt must transfer control to the appropriate interrupt service routine (ISR) (see Fig. 8).
- The straightforward method for handling this transfer would be to invoke a generic routine to examine the interrupt information; the routine, in turn, 'would call the interrupt-specific handler.
- Since only a predefined number of interrupts is possible, a table of pointers to interrupt routines can be used. Generally, the table of pointers is stored in low memory (the first 100 or so locations). These

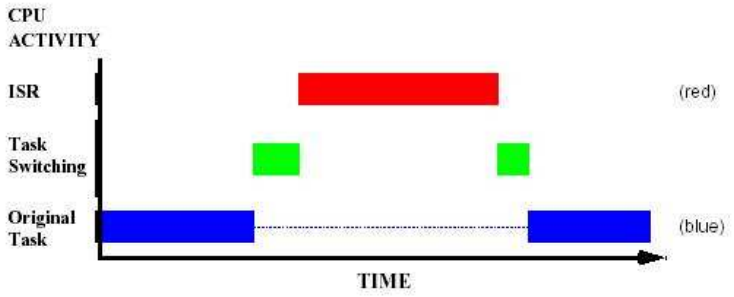


Figure 8: Interrupt

locations hold the addresses of the interrupt service routines for the various devices.

- This array, or interrupt vector, of addresses is then indexed by a unique device number, given with the interrupt request, to provide the address of the interrupt service routine for the interrupting device.
- Operating systems as different as Windows and UNIX dispatch interrupts in this manner.

2.2.2 Storage Structure

- Computer programs must be in main memory (high speed semiconductor, also called random-access memory or RAM) to be executed. We say random access because the CPU can access any byte of storage in any order.
- Referred to as real memory or primary memory. Volatile, because its contents are lost when the power is removed.
  - Interaction is achieved through a sequence of load or store instructions to specific memory addresses.
  - The load instruction moves a word (collection of bytes, each word has its own address) from main memory to an internal register within the CPU, whereas the store instruction moves the content of a register to main memory.
  - Aside from explicit loads and stores, the CPU automatically loads instructions from main memory for execution.

- A typical *instruction-execution cycle*, as executed on a system with a *von Neumann architecture*,
  - First fetches an instruction from memory and stores that instruction in the instruction register.
  - The instruction is then decoded and may cause operands to be fetched from memory and stored in some internal register.
  - After the instruction on the operands has been executed, the result may be stored back in memory.
- *Fetch-execute cycle* (see Fig. 9)

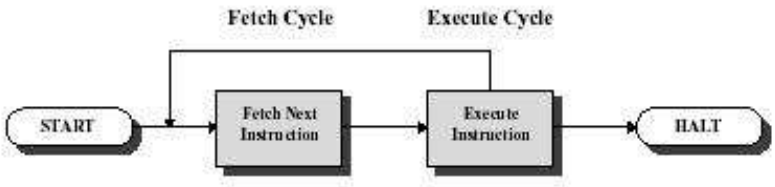


Figure 9: Fetch and Execute Cycle

  - Program counter (PC) holds address of the instruction to be fetched next,
  - The processor fetches the instruction from memory,
  - Program counter is incremented after each fetch,
  - Overlapped on modern architectures (pipelining).
- Notice that the memory unit sees only a stream of memory addresses; it does not know how they are generated (by the instruction counter, indexing, indirection, literal addresses, or some other.
- Ideally, we want the programs and data to reside in main memory permanently. This arrangement usually is not possible for the following two reasons:
  1. Main memory is usually too small to store all needed programs and. data permanently.
  2. Main memory is a volatile storage device that loses its contents when power is turned off or otherwise lost.

- Thus, most computer systems provide secondary storage as an *extension* of main memory. The most common secondary-storage device is a magnetic disk.
- Many programs then use the disk as both a source and a destination of the information for their processing. Hence, the proper management of disk storage is of *central importance* to a computer system.
- The main differences among the various storage systems lie in speed, cost, size, and volatility. The wide variety of storage systems in a computer system can be organized in a hierarchy (See Fig. 10) according to speed and cost. The higher levels are expensive, but they are fast. As we move down the hierarchy, the cost per bit generally decreases, whereas the access time generally increases.
- Stages such as the CPU registers and cache are typically located within the CPU chip so distances are very short and buses can be made very very wide (e.g. 128-bits), yielding very fast speeds.

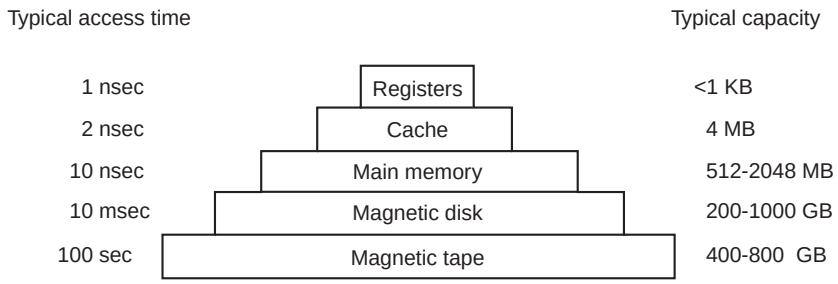


Figure 10: A typical memory hierarchy. The numbers are very rough approximations.

- The design of a complete memory system must balance all the factors. It must use only as much expensive memory as necessary while providing as much inexpensive, nonvolatile memory as possible. Caches can be installed to improve performance where a large access-time or transfer-rate disparity exists between two components.
- Cache memory;

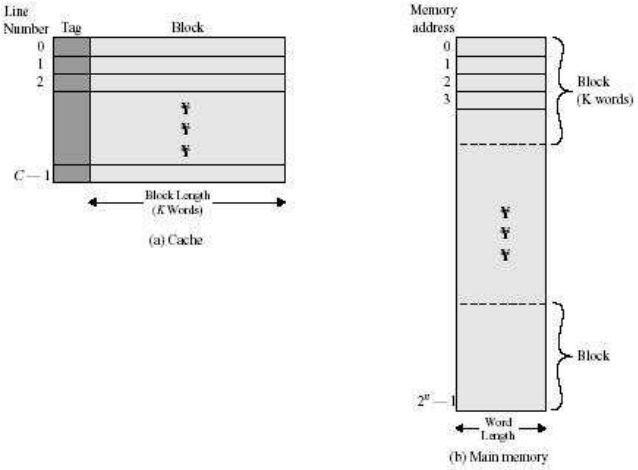


Figure 11: Cache and Main Memory

- Main memory should be, fast, abundant, cheap, Unfortunately, that's not the reality. Solution: combination of fast & expensive and slow & cheap memory (see Fig. 12left)
- Contain a small amount of very fast storage which holds a subset of the data held in the main memory.
- Processor first checks cache. If not found in cache, the block of memory containing the needed information is moved to the cache replacing some other data.

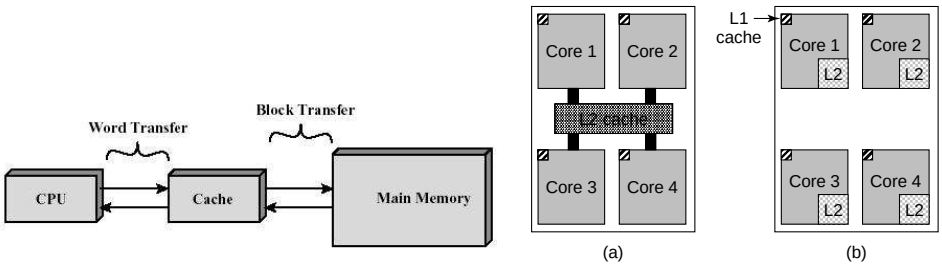


Figure 12: Left: Cache Memory. Right: (a) A quad-core chip with a shared L2 cache. (b) A quad-core chip with separate L2 caches.

- Cache design;
  - *Cache size*, small caches have a significant impact on performance.
  - *Line size* (block size), the unit of data exchanged between cache and main memory (see Fig. 12left).

  - *Cache Hit* means the information was found in the cache. Larger line size ⇒ higher hit rate.
  - *Cache Miss* ??
  - Questions when dealing with cache:
    * 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.

- Disk Cache
  - A portion of main memory used as a buffer to temporarily to hold data for the disk.
  - Some data written out may be referenced again. The data are retrieved rapidly from the software cache instead of slowly from disk.

- Future storage technology includes 3-dimensional crystal structures which allow optical access to a dense 3-dimensional storage facility (see Fig. 13).
  http://www.voyle.net/Guest Writers/Michael E. Thomas/Atomic_press.htm

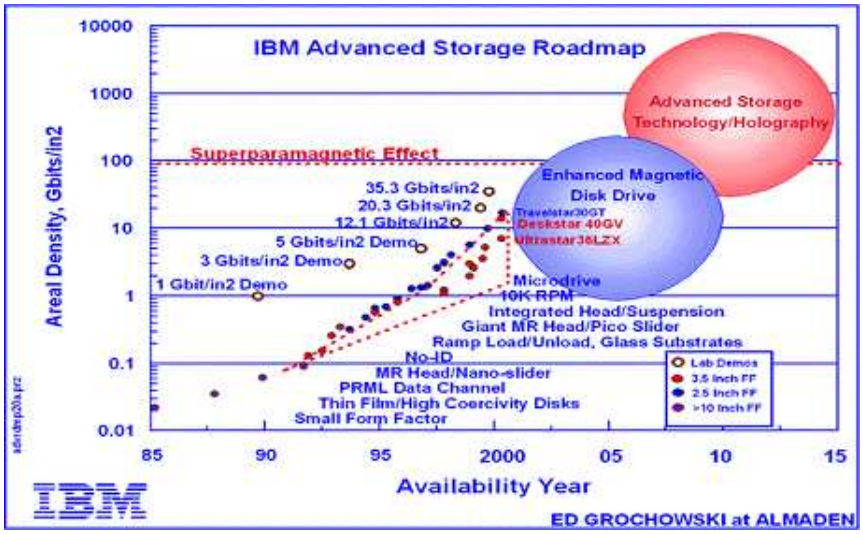


Figure 13: IBM Advanced Storage Roadmap.

2.2.3 I/O Structure

- Storage is only one of many types of I/O devices within a computer. A large portion of OS code is dedicated to managing I/O, both because of its importance to the reliability and performance of a system and because of the varying nature of the devices.

- A source of cheaper-per-byte and non-volatile storage is provided by magnetic disk. However, the computer does not have direct random access to any byte at any time on the disk – the magnetic discs in the drive are rotating and magnetic heads move in and out in order to access any part of the surface area on the disc that holds data. This means access usually involves a disc rotation delay and also a head positioning delay (see Fig. 14left).

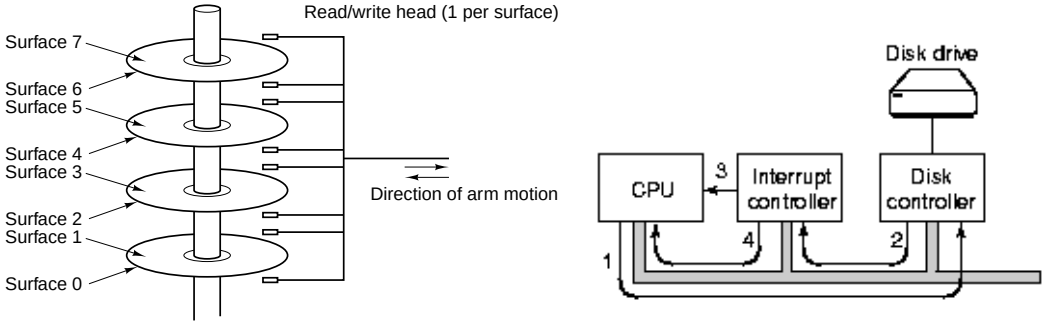


Figure 14: Left: Structure of a disk drive. Right: The steps in starting an I/O device.

- A general-purpose computer system consists of CPUs and multiple device controllers that are connected through a common bus (see Fig. 15).
  - Each device controller is in charge of a specific type of device. Depending on the controller, there may be more than one attached device.
  - The device controller is responsible for moving the data between the peripheral devices that it controls and its local buffer storage (see Fig. 14right).
  - Typically, OSs have a device driver for each device controller. Software that communicates with controller is called device driver. This device driver *understands* the device controller and presents

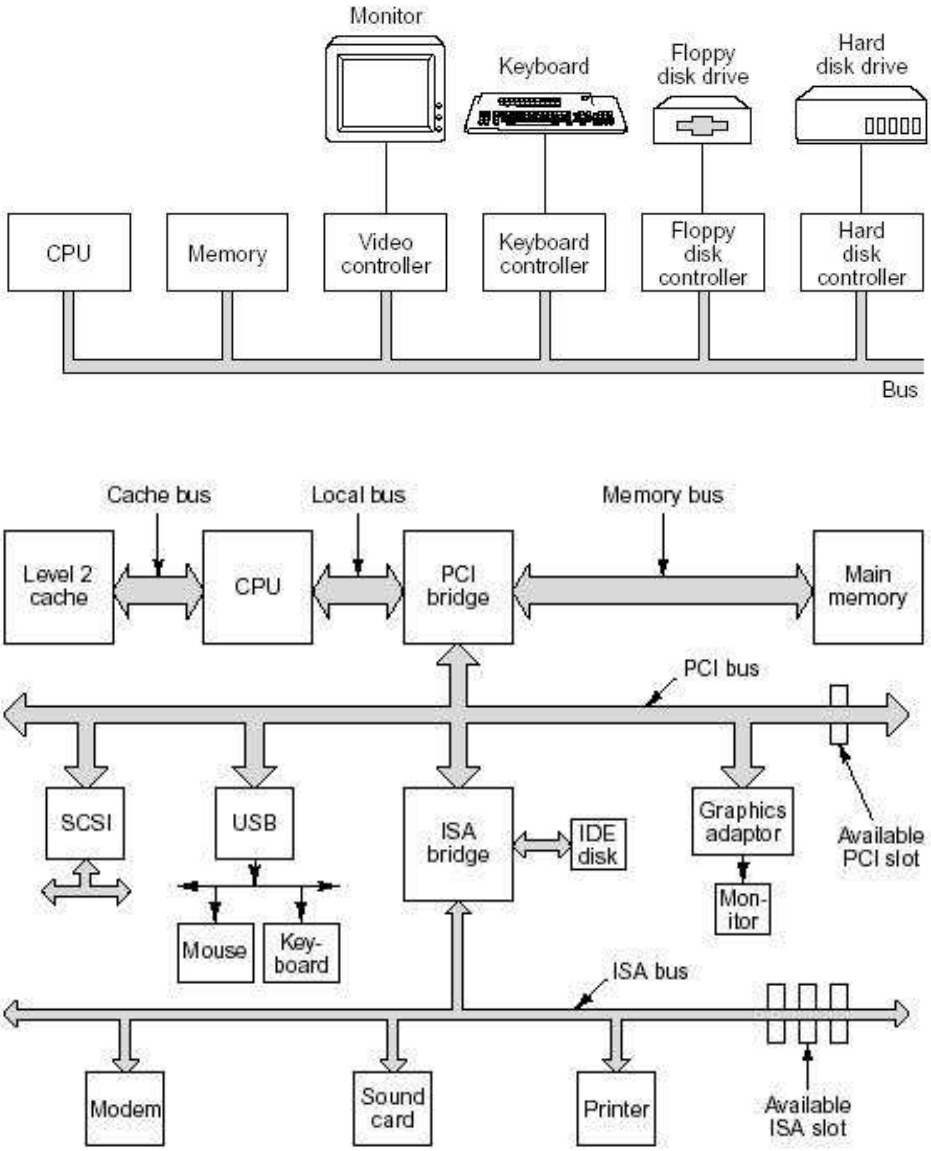


Figure 15: Upper: Top-level Components. Lower: The structure of a large Pentium system.

a uniform interface to the device to the rest of the OS. Most drivers run in kernel mode. To put new driver into kernel, system may have to be relinked, or be rebooted, or dynamically load new driver.

  - A system bus would link the CPU and memory. This structure would involve a pathway along which data could travel (usually 32-bits side-by-side i.e. in bit-wise parallel).

- To start an I/O operation;

  - The device driver loads the appropriate registers within the device controller.
  - The device controller, in turn, examines the contents of these registers to determine what action to take (such as "read a character from the keyboard").
  - The controller starts the transfer of data from the device to its local buffer. Once the transfer of data is complete, the device controller informs the device driver via an interrupt that it has finished its operation.
  - The device driver then returns control to the OS, possibly returning the data or a pointer to the data if the operation was a read.

- Interrupts normal sequence of execution. I/O requests can be handled *synchronously* or *asynchronously.*

  - *In a synchronous system,* a program makes the appropriate OS call, as the CPU is now executing OS code, the original program's execution is halted i.e. it waits.
  - *In an asynchronous system,* a program makes its request via the OS call, then its execution resumes, it will most likely not have had its request serviced yet! The advantage of having an asynchronous mechanism available is that the programmer is free to organize other CPU activity while the I/O request is handled.

- This form of interrupt-driven I/O is fine for moving small amounts of data but can produce high overhead when used for bulk data movement such as disk I/O. CPU much faster than I/O devices, hence waiting for I/O operation to finish is inefficient.

- To solve this problem, direct memory access (DMA) is used. After setting up buffers, pointers, and counters for the I/O device, the device

controller transfers an entire block of data directly to or from its own buffer storage to memory, with no intervention by the CPU.

- Only one interrupt is generated per block, to tell the device driver that the operation has completed, rather than the one interrupt per byte generated for low-speed devices.
- While the device controller is performing these operations, the CPU is available to accomplish other work.