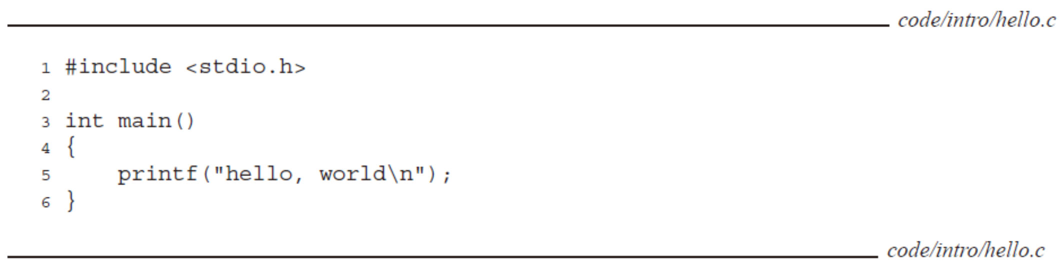
# Chapter 1 A Tour of Computer Systems

$$\text{A computer system}\begin{cases} hardware \\ software \end{cases}$$



code/intro/hello.c

```c
#include <stdio.h>

int main()
{
    printf("hello, world\n");
}
```

code/intro/hello.c

Figure 1.1: **The** `hello` **program.**

## 1.1 Information Is Bits + Context

$$\text{files}\begin{cases} ASCII\ characters \rightarrow text\ files \\ others \rightarrow binary\ files \end{cases} \rightarrow \text{a bunch of bits}$$

Aside：Origins of the C programming language

The reason of C is successful:

1、C was closely tied with the Unix operating system.

← Developed from the beginning as the system programming language for Unix.

2、C is a small, simple language. ← The design was controlled by a single person.

3、C was designed for a practical purpose.

← C was designed to implement the Unix operating system.

## 1.2 Programs Are Translated by Other Programs into Different Forms

linux > gcc − o hello hello. c

⬇

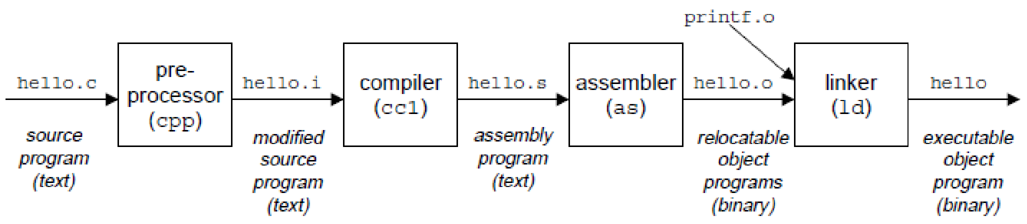
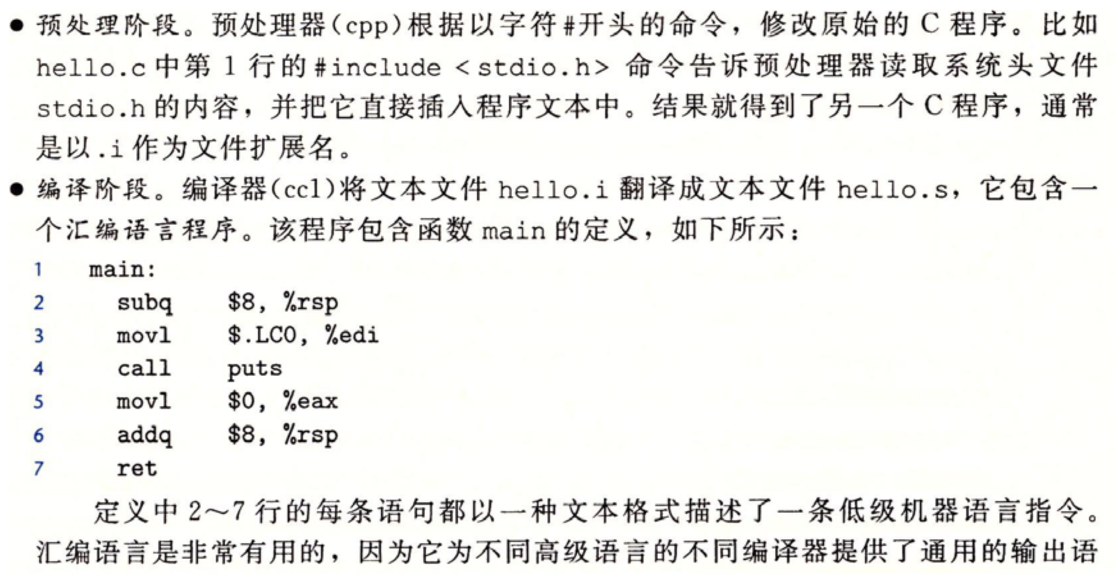


Figure 1.3: **The compilation system.**

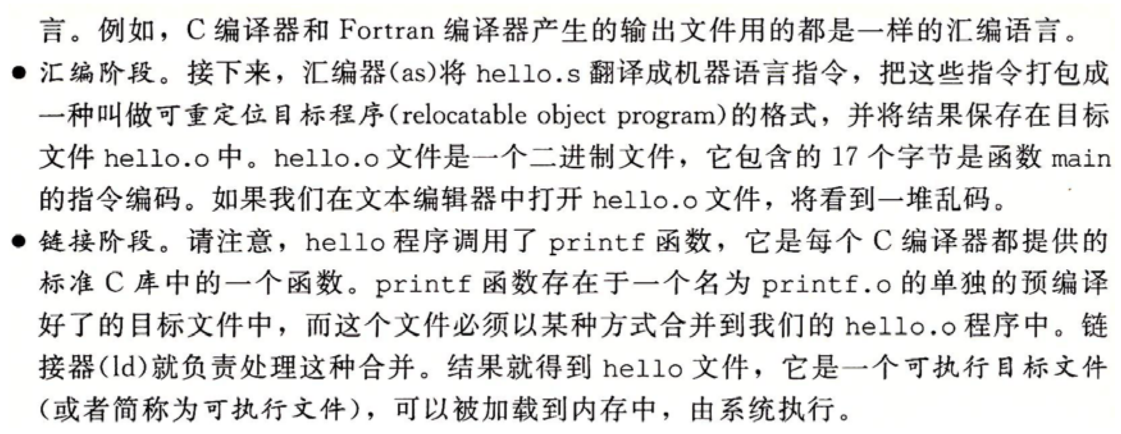
The gcc compiler driver reads the source file translates it into an execuable object file.



- 预处理阶段。预处理器(cpp)根据以字符#开头的命令，修改原始的 C 程序。比如 hello.c 中第 1 行的#include <stdio.h> 命令告诉预处理器读取系统头文件 stdio.h 的内容，并把它直接插入程序文本中。结果就得到了另一个 C 程序，通常是以.i 作为文件扩展名。
- 编译阶段。编译器(ccl)将文本文件 hello.i 翻译成文本文件 hello.s，它包含一个汇编语言程序。该程序包含函数 main 的定义，如下所示：

```
1   main:
2     subq    $8, %rsp
3     movl    $.LC0, %edi
4     call    puts
5     movl    $0, %eax
6     addq    $8, %rsp
7     ret
```

定义中 2～7 行的每条语句都以一种文本格式描述了一条低级机器语言指令。汇编语言是非常有用的，因为它为不同高级语言的不同编译器提供了通用的输出语



言。例如，C 编译器和 Fortran 编译器产生的输出文件用的都是一样的汇编语言。
- 汇编阶段。接下来，汇编器(as)将 hello.s 翻译成机器语言指令，把这些指令打包成一种叫做可重定位目标程序(relocatable object program)的格式，并将结果保存在目标文件 hello.o 中。hello.o 文件是一个二进制文件，它包含的 17 个字节是函数 main 的指令编码。如果我们在文本编辑器中打开 hello.o 文件，将看到一堆乱码。
- 链接阶段。请注意，hello 程序调用了 printf 函数，它是每个 C 编译器都提供的标准 C 库中的一个函数。printf 函数存在于一个名为 printf.o 的单独的预编译好了的目标文件中，而这个文件必须以某种方式合并到我们的 hello.o 程序中。链接器(ld)就负责处理这种合并。结果就得到 hello 文件，它是一个可执行目标文件(或者简称为可执行文件)，可以被加载到内存中，由系统执行。

Aside: The GUN project

GUN tools provide the environment for the Linux kernel.

## 1.3 It Pays to Understand How Compilation Systems Work

1) Optimizing program performance
2) Understanding link – time errors
3) Avoiding security holes ←
buffer overflow vulerablities have accounted for many of the security holes
in network and Internet Servers. → need to carefully restrict the quantity
and forms of data they accept from untrustde sources.

## 1.4 Processors Read and Interpret Instructions Stored in Memory

Linux >./hello type its name to an application program known as a shell

### 1.4.1 Hardware Organization of a System

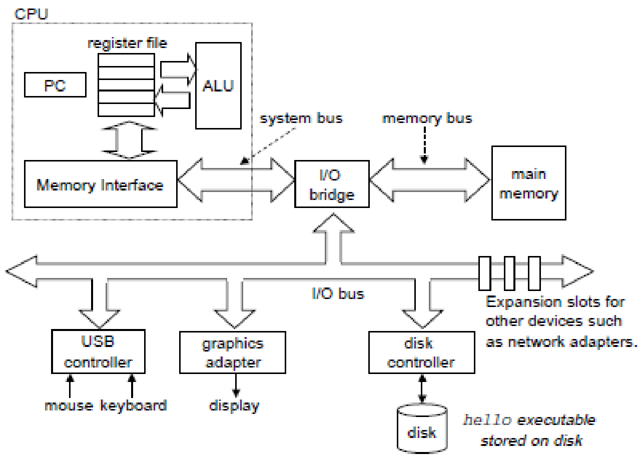


Figure 1.4: **Hardware organization of a typical system.** CPU: Central Processing Unit, ALU: Arithmetic/Logic Unit, PC: Program counter, USB: Universal Serial Bus.

1、Bus 总线

Running throughout the system is a collection of electrical conduits
Carrying bytes of information back and forth between the components.

2、I/O *Devices* I/O 设备

Input/output($I/O$)devices are the system's connection to the external world.
Each I/O device is connected to the I/O bus by either a controller or an adapter.
The distinction between the two is mainly one of packaging.

3、Main Memory 主存

a temporary storage device that holds both a program and the data it manipulates
while the processor is executing the program.

Physically, main memory consists of a collection of Dynamic Random Access Memory (DRAM) chips.
Logically, as a linear array of bytes, each with its own unique address (array index) starting at zero.

4、Processor 处理器

The central processing unit (CPU), or simply processor, is the engine
that interprets (or executes)instructions stored in main memory.

At its core is a word — sized storage device (or register)called the program counter (PC).
model defined by its instruction set architecture.
The register file is a small storage device that consists of a collection of word
— sized registers, each with its own unique name.
The ALU computes new data and address values.

CPU carries the instruction: Load、Store、Update、I/ O Read、I/O Write、Jump.

## 1.4.2 Running the `hello` Program

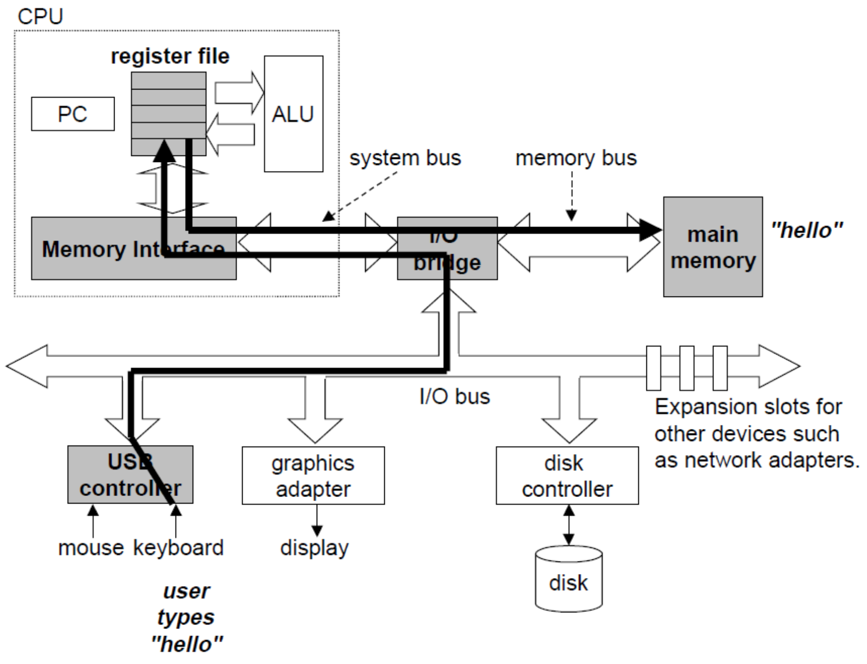


Figure 1.5: **Reading the `hello` command from the keyboard.**

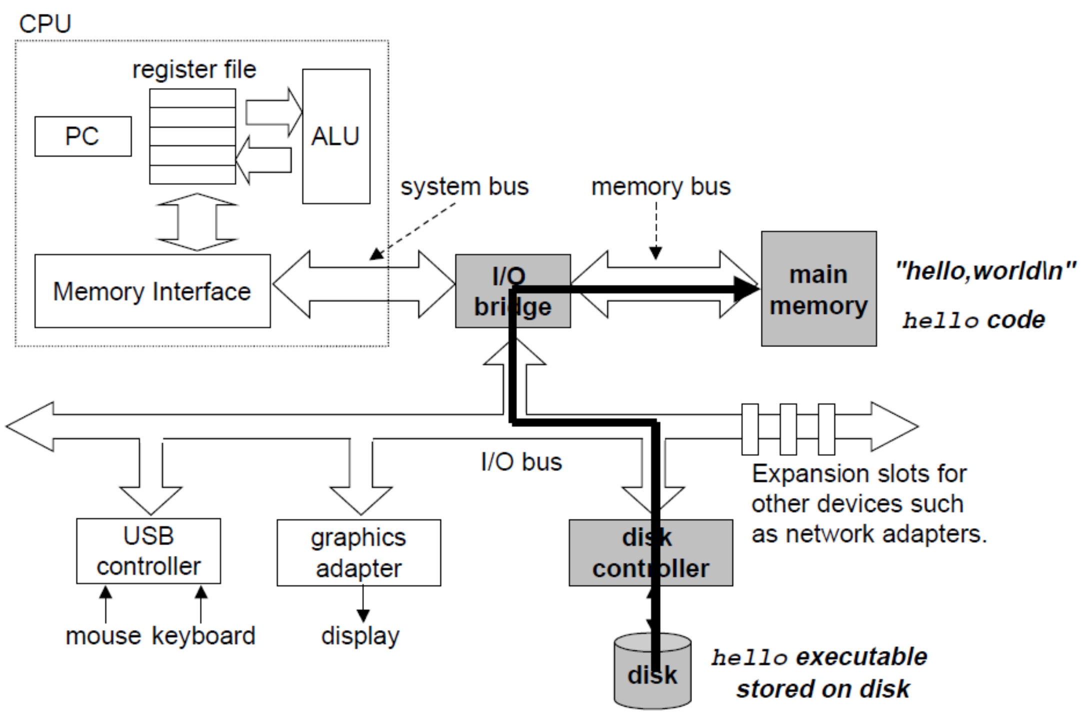


Figure 1.6: **Loading the executable from disk into main memory.**

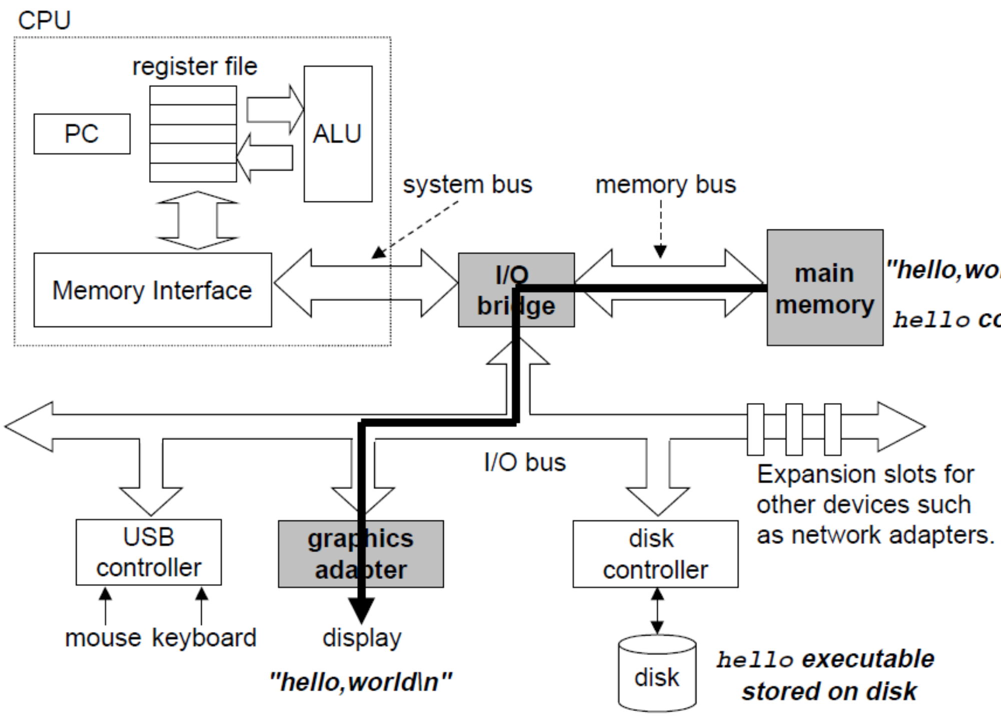


Figure 1.7: **Writing the output string from memory to the display.**

## 1.1 Caches Matter

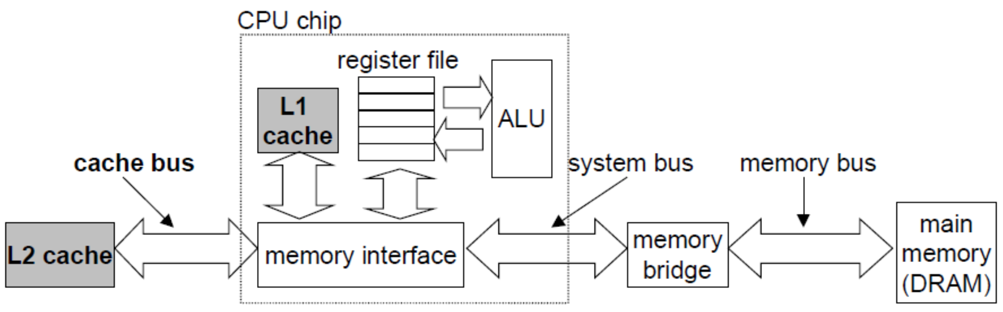


Figure 1.8: **Caches.**

faster storage devices： *cache memories* (or simply caches)

that serve as temporary staging areas for information that the processor is likely to need in the near future

## 1.6 Storage Devices Form a Hierarchy

The main idea of a memory hierarchy is that storage at one level serves as a cache for storage at the next lower level.

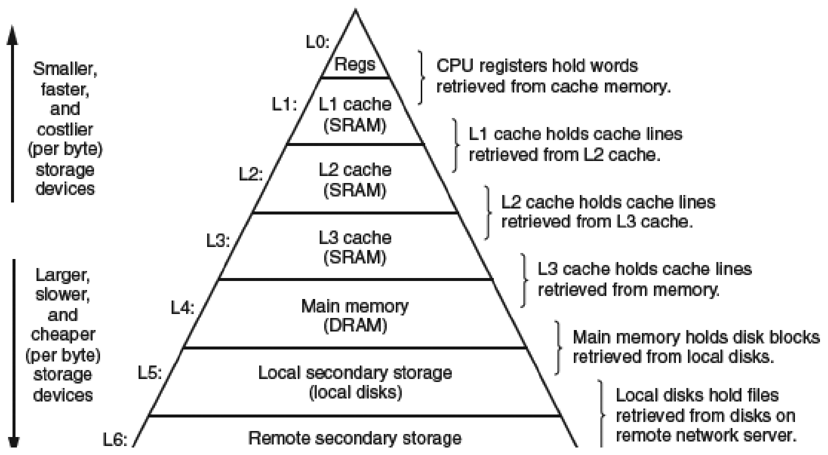

Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
L0: Regs
L1: L1 cache (SRAM)
L2: L2 cache (SRAM)
L3: L3 cache (SRAM)
L4: Main memory (DRAM)
L5: Local secondary storage (local disks)
L6: Remote secondary storage
CPU registers hold words retrieved from cache memory.
L1 cache holds cache lines retrieved from L2 cache.
L2 cache holds cache lines retrieved from L3 cache.
L3 cache holds cache lines retrieved from memory.
Main memory holds disk blocks retrieved from local disks.
Local disks hold files retrieved from disks on remote network server.

**Figure 1.9 An example of a memory hierarchy.**

## 1.7 The Operating System Manages the Hardware

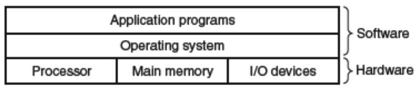


**Figure 1.10 Layered view of a computer system.**

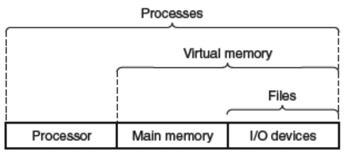


**Figure 1.11 Abstractions provided by an operating system.**

The operating system has two primary purposes:
(1) to protect the hardware from misuse by runaway applications and
(2) to provide applications with simple and uniform mechanisms for manipulating complicated and often wildly different low-level hardware devices.

## 1.7.1 Processes

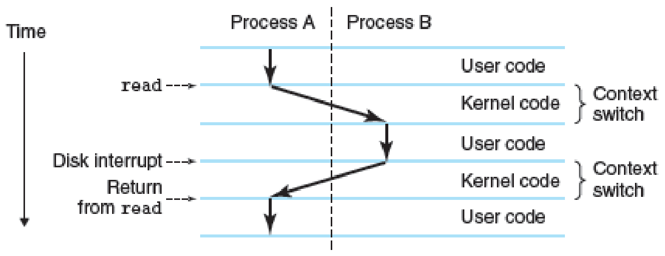


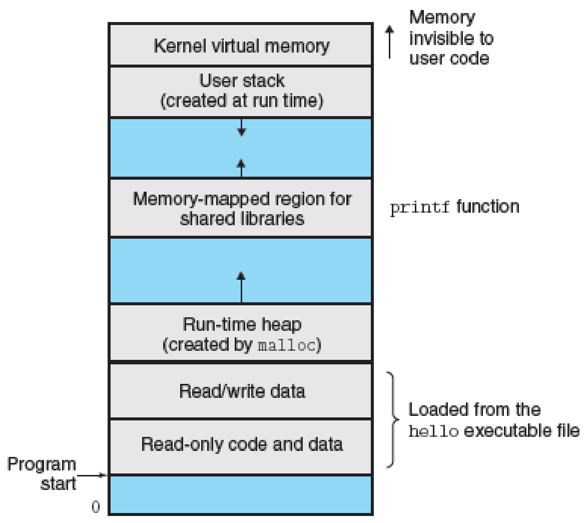
**Figure 1.12 Process context switching.**

the transition from one process to another is managed by the operating system *kernel*.

## 1.7.2 Threads

multiple execution units

## 1.7.3 Virtual Memory



Kernel virtual memory
User stack (created at run time)
Memory-mapped region for shared libraries
Run-time heap (created by malloc)
Read/write data
Read-only code and data
Memory invisible to user code
printf function
Loaded from the hello executable file
Program start
0

**Figure 1.13 Process virtual address space.**

(The regions are not drawn to scale.)

**Program code and data**
**Heap.**
**Shared libraries**
**Stack**
**Kernel virtual memory.**

### 1.7.4 Files

a sequence of bytes

## 1.8 Systems Communicate with Other Systems Using Networks

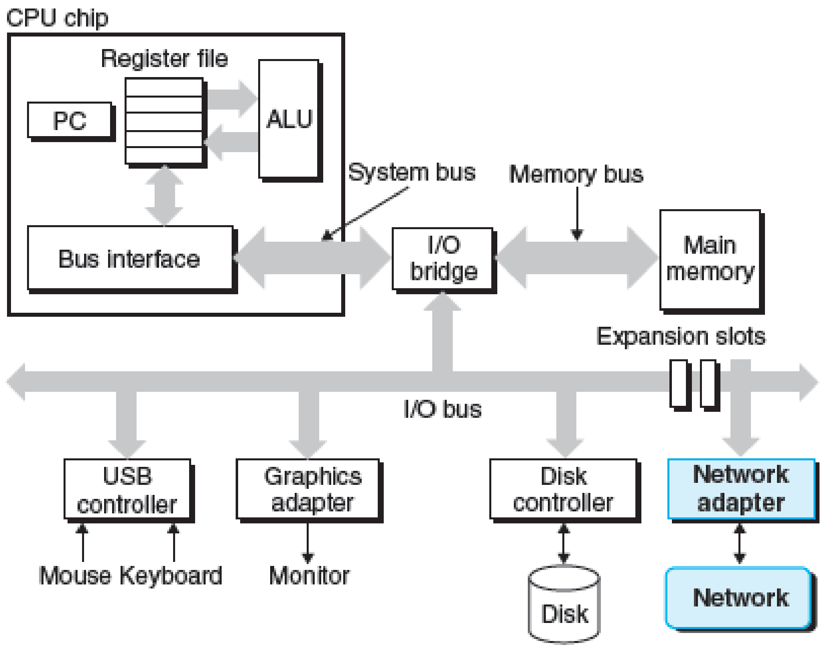


**Figure 1.14 A network is another I/O device.**

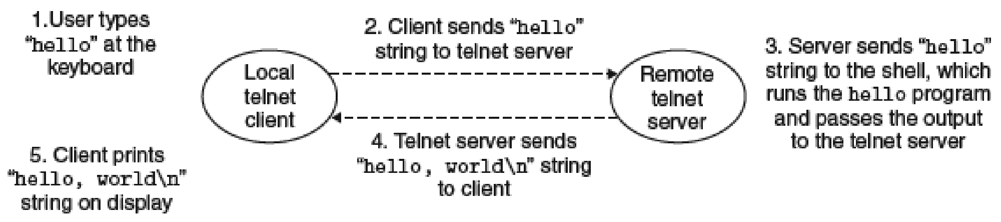

1.User types "hello" at the keyboard
2. Client sends "hello" string to telnet server
Local telnet client
Remote telnet server
3. Server sends "hello" string to the shell, which runs the hello program and passes the output to the telnet server
4. Telnet server sends "hello, world\n" string to client
5. Client prints "hello, world\n" string on display

**Figure 1.15 Using telnet to run `hello` remotely over a network.**

## 1.9 Important Themes

### 1.9.1 Amdahl's Law

the speedup $S = T_{\text{old}}/T_{\text{new}}$ as

S=1(1−α)+α/k
a system in which executing some application requires time $T_{\text{old}}$.
some part of the system requires a fraction α of this time
this part now requires time $(\alpha T_{\text{old}})/k$

### 1.9.2 Concurrency and Parallelism

#### *1)Thread-Level Concurrency*

with threads, we can even have multiple control flows executing within a single process.

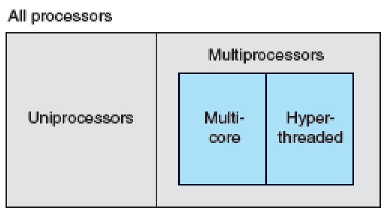


**Figure 1.16 Categorizing different processor configurations.**

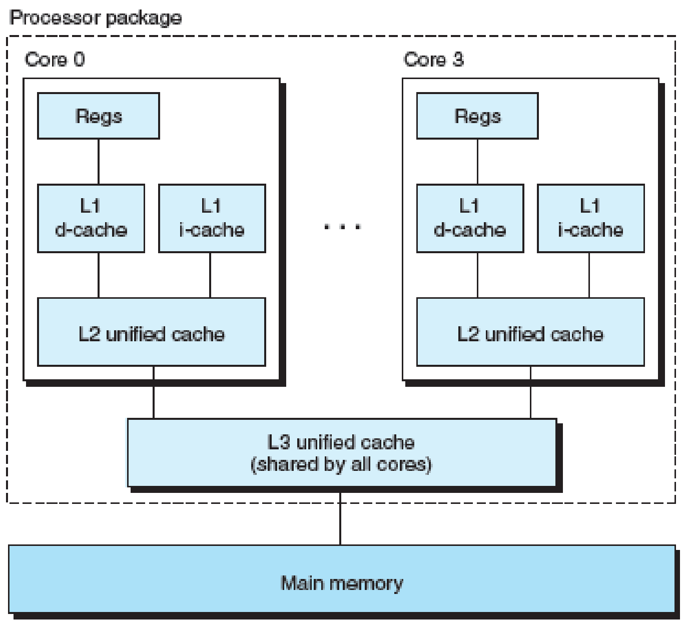


**Figure 1.17 Multi-core processor organization.**

Hyperthreading, sometimes called *simultaneous multi-threading*, is a technique that allows a single CPU to execute multiple flows of control.

### *2)Instruction-Level Parallelism*

modern processors can execute multiple instructions at one time

### *3)Single-Instruction, Multiple-Data (SIMD) Parallelism*

At the lowest level, many modern processors have special hardware that allows a single instruction to cause multiple operations to be performed in parallel, a mode known as *single-instruction, multiple-data*(SIMD) parallelism.

## 1.9.3 The Importance of Abstractions in Computer Systems

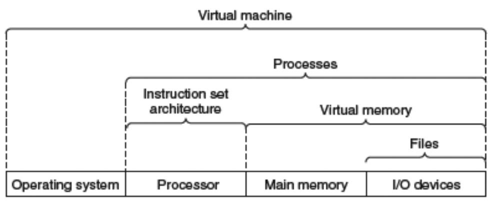


**Figure 1.18 Some abstractions provided by a computer system.**

## 1.10 Summary

A computer system consists of hardware and systems software that cooperate to run application programs. Information inside the computer is represented as groups of bits that are interpreted in different ways, depending on the context. Programs are translated by other programs into different forms, beginning as ASCII text and then translated by compilers and linkers into binary executable files.

Processors read and interpret binary instructions that are stored in main memory. Since computers spend most of their time copying data between memory, I/O devices, and the CPU registers, the storage devices in a system are arranged in a hierarchy, with the CPU registers at the top, followed by multiple levels of hardware cache memories, DRAM main memory, and disk storage. Storage devices that are higher in the hierarchy are faster and more costly per bit than those lower in the hierarchy. Storage devices that are higher in the hierarchy serve as caches for devices that are lower in the hierarchy. Programmers can optimize the performance of their C programs by understanding and exploiting the memory hierarchy.

The operating system kernel serves as an intermediary between the application and the hardware. It provides three fundamental abstractions: (1) Files are abstractions for I/O devices. (2) Virtual memory is an abstraction for both main memory and disks. (3) Processes are abstractions for the processor, main memory, and I/O devices.

Finally, networks provide ways for computer systems to communicate with one another. From the viewpoint of a particular system, the network is just another I/O device.

emor

emor

Problems
Using Amdahl's Law

the speedup S = Told/Tnew as $S=1(1-\alpha)+\alpha/k$

to calculate the speedup.
What is more,  Amdahl's law applies to more than just computer systems