A computer system consists of hardware and systems software that work together to run application programs. Specific implementations of systems change over time, but the underlying concepts do not. All computer systems have similar hardware and software components that perform similar functions. This book is written for programmers who want to get better at their craft by understanding how these components work and how they affect the correctness and performance of their programs. You are poised for an exciting journey. If you dedicate yourself to learning the concepts in this book, then you will be on your way to becoming a rare “power programmer,” enlightened by an understanding of the underlying computer system and its impact on your application programs. You are going to learn practical skills such as how to avoid strange numerical errors caused by the way that computers represent numbers. You will learn how to optimize your C code by using clever tricks that exploit the designs of modern processors and memory systems. You will learn how the compiler implements procedure calls and how to use this knowledge to avoid the security holes from buffer overflow vulnerabilities that plague network and Internet software. You will learn how to recognize and avoid the nasty errors during linking that confound the average programmer. You will learn how to write your own Unix shell, your own dynamic storage allocation package, and even your own Web server. You will learn the promises and pitfalls of concurrency, a topic of increasing importance as multiple processor cores are integrated onto single chips. In their classic text on the C programming language [61], Kernighan and Ritchie introduce readers to C using the hello program shown in Figure 1.1. Although hello is a very simple program, every major part of the system must work in concert in order for it to run to completion. In a sense, the goal of this book is to help you understand what happens and why when you run hello on your system. We begin our study of systems by tracing the lifetime of the hello program, from the time it is created by a programmer, until it runs on a system, prints its simple message, and terminates. As we follow the lifetime of the program, we will briefly introduce the key concepts, terminology, and components that come into play. Later chapters will expand on these ideas. - Hệ thống máy tính bao gồm phần cứng và phần mềm hệ thống làm việc cùng nhau để chạy các chương trình ứng dụng. Việc triển khai cụ thể của các hệ thống thay đổi theo thời gian, nhưng các khái niệm cơ bản thì không. Tất cả các hệ thống máy tính đều có các thành phần phần cứng và phần mềm tương tự nhau thực hiện các chức năng tương tự. Cuốn sách này được viết cho các lập trình viên muốn trở nên giỏi hơn trong công việc của họ bằng cách hiểu cách các thành phần này hoạt động và cách chúng ảnh hưởng đến tính đúng đắn và hiệu suất của các chương trình của họ. Bạn đã sẵn sàng cho một cuộc hành trình thú vị. Nếu bạn dành bản thân để tìm hiểu các khái niệm trong cuốn sách này, thì bạn sẽ trên đường trở thành một “lập trình viên giỏi” hiếm có, được khai sáng bởi sự hiểu biết về hệ thống máy tính cơ bản và tác động của nó đối với các chương trình ứng dụng của bạn. Bạn sẽ học các kỹ năng thực tế như cách tránh các lỗi số lạ do cách máy tính biểu diễn số. Bạn sẽ học cách tối ưu hóa mã C của mình bằng cách sử dụng các thủ thuật thông minh khai thác các thiết kế của bộ xử lý và hệ thống bộ nhớ hiện đại. Bạn sẽ học cách trình biên dịch thực hiện các lệnh gọi thủ tục và cách sử dụng kiến ​​thức này để tránh các lỗ hổng bảo mật từ các lỗ hổng tràn bộ đệm gây bệnh cho mạng và phần mềm Internet. Bạn sẽ học cách nhận ra và tránh những lỗi khó chịu trong quá trình liên kết khiến các lập trình viên bình thường khó hiểu. Bạn sẽ học cách viết trình bao Unix của riêng mình, gói phân bổ lưu trữ động của riêng bạn và thậm chí cả máy chủ Web của riêng bạn. Bạn sẽ tìm hiểu những hứa hẹn và cạm bẫy của sự đồng thời, một chủ đề ngày càng quan trọng khi nhiều lõi xử lý được tích hợp vào các chip đơn. Trong văn bản cổ điển của họ về ngôn ngữ lập trình C [61], Kernighan và Ritchie giới thiệu với độc giả về C bằng cách sử dụng chương trình hello được trình bày trong Hình 1.1. Mặc dù hello là một chương trình rất đơn giản, nhưng mọi bộ phận chính của hệ thống đều phải hoạt động đồng bộ để nó chạy đến khi hoàn thành. Theo một nghĩa nào đó, mục tiêu của cuốn sách này là giúp bạn hiểu điều gì sẽ xảy ra và tại sao khi bạn chạy hello trên hệ thống của mình. Chúng tôi bắt đầu nghiên cứu các hệ thống bằng cách lần theo vòng đời của chương trình hello, từ khi nó được tạo ra bởi một lập trình viên, cho đến khi nó chạy trên một hệ thống, in ra thông báo đơn giản và kết thúc. Khi chúng tôi theo dõi vòng đời của chương trình, chúng tôi sẽ giới thiệu ngắn gọn các khái niệm, thuật ngữ và các thành phần chính có tác dụng. Các chương sau sẽ mở rộng về những ý tưởng này.

#include <stdio.h>

int main()

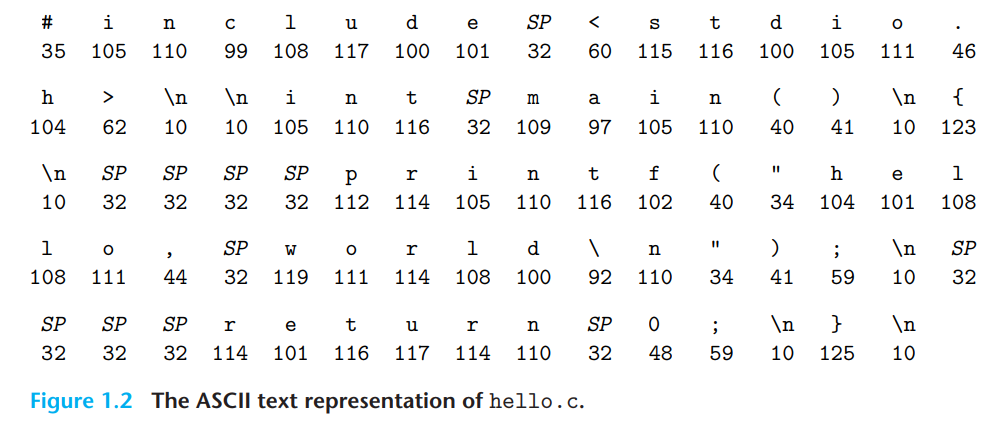
{

    printf("hello, world\n");

    return 0;

}

Figure 1.1 The hello program. (Source: [60])



Our hello program begins life as a source program (or source file) that the programmer creates with an editor and saves in a text file called hello.c. The source program is a sequence of bits, each with a value of 0 or 1, organized in 8-bit chunks called bytes. Each byte represents some text character in the program. – Chương trình hello bắt đầu vòng đời như một source program (hoặc source file) được các programmer tạo với một editor và saves trong một text file được gọi là hello.c. Source program là một chuỗi các bits, tương ứng với giá trị 0 hoặc 1, được tổ chức trong khối 8-bit được gọi là bytes. Từng bytes đại diện cho một vài text trong chương trình.

Most computer systems represent text characters using the ASCII standard that represents each character with a unique byte-size integer value.1 For example, Figure 1.2 shows the ASCII representation of the hello.c program – hầu hết các hệ thống máy tính đại diện các ký tự text bằng cách sử dụng ASCII standard để đại diện cho từng ký tự với một unique byte-size integer value duy nhất. Ví dụ, hình 1.2 shows ASCII representation của chương trình hello.c

The hello.c program is stored in a file as a sequence of bytes. Each byte has an integer value that corresponds to some character. For example, the first byte has the integer value 35, which corresponds to the character ‘#’. The second byte has the integer value 105, which corresponds to the character ‘i’, and so on. Notice that each text line is terminated by the invisible newline character ‘\n’, which is represented by the integer value 10. Files such as hello.c that consist exclusively of ASCII characters are known as text files. All other files are known as binary files. – Chương trình hello.c được lưu trữ trong một file như một chuỗi các bytes. Từng byte có một giá trị integer tương ứng với vài ký tự. Ví dụ, byte đầu tiên có integer value là 35, điều này tương ứng với ký tự ‘#’. Second byte có integer valute là 105, tương ứng với ký tự ‘i’,... Lưu ý rằng từng text line được ngắt bởi invisible newline character ‘\n’, điều này đại diện cho giá trị integer 10. Các files như hello.c chỉ bao gồm các ASCII character được gọi là text files. Tất cả các files khác được biết đến như binary files.

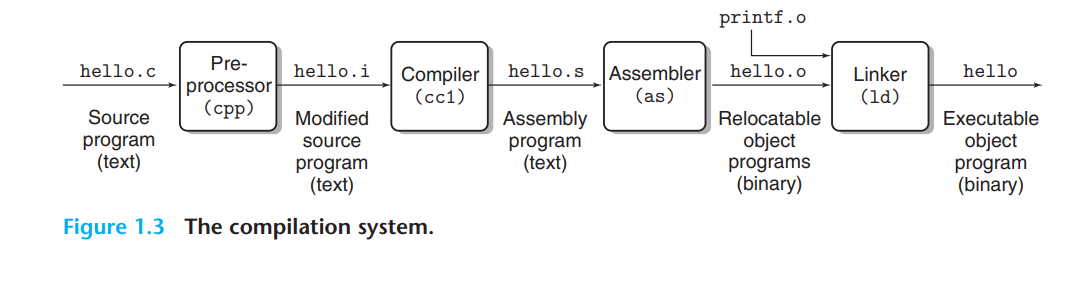
The representation of hello.c illustrates a fundamental idea: All information in a system—including disk files, programs stored in memory, user data stored in memory, and data transferred across a network—is represented as a bunch of bits. The only thing that distinguishes different data objects is the context in which we view them. For example, in different contexts, the same sequence of bytes might represent an integer, floating-point number, character string, or machine instruction. – Biểu diễn của hello.c minh hoạ một ý tưởng cơ bản: Tất cả các thông tin trong một system – bao gồm disk files, programs stored in memory, user data được lưu trữ trong memory, và data được transfer thông qua network – được biểu diễn dưới dạng một loạt các bits. Điều duy nhất giúp chúng ta phân biệt được sự khác biệt của các data objects là ngữ cảnh mà chúng ta xem xét chúng. Ví dụ, trong các ngữ cảnh khác, một chuỗi các bytes giống nhau có thể đại diện cho một integer, floating-point number, character string, hoặc machine instruction.

As programmers, we need to understand machine representations of numbers because they are not the same as integers and real numbers. They are finite approximations that can behave in unexpected ways. This fundamental idea is explored in detail in Chapter 2. – Với các programmers, chúng ta cần hiểu cách machine biểu diễn các con số bởi vì chúng không giống như các integers và real numbers. Chúng là các finite approximations (xấp xỉ hữu hạn) rằng có thể hoạt động theo cách không mong muốn. Ý tưởng cơ ban này sẽ được khám phá trong chi tiết chương 2.

1.2 Programs Are Translated by Other Programs into Different Forms

The hello program begins life as a high-level C program because it can be read and understood by human beings in that form. However, in order to run hello.c on the system, the individual C statements must be translated by other programs into a sequence of low-level machine-language instructions. These instructions are then packaged in a form called an executable object program and stored as a binary disk file. Object programs are also referred to as executable object files – Chương trình hello bắt đầu vòng đời như một high-level C program bởi vì nó có thể đọc và hiểu bởi con người trong khuôn mẫu. Tuy nhiên, thứ tự để run hello.c trên system, các individual C statements cần phải được translated bằng các chương trình khác trong mội chuỗi các tập lệnh low-level machine-language. Các tập lệnh này sau đó được đóng gói trong một form được gọi là một executable object program và được lưu trữ dưới dạng một binary disk file. Object programs cũng được biết tới là executable object files.

On a Unix system, the translation from source file to object file is performed by a compiler driver: - Trên một Unix system, translation từ source file thành object file được thực hiện bởi một compiler driver:



linux> gcc -o hello hello.c

Here, the gcc compiler driver reads the source file hello.c and translates it into an executable object file hello. The translation is performed in the sequence of four phases shown in Figure 1.3. The programs that perform the four phases (preprocessor, compiler, assembler, and linker) are known collectively as the compilation system. - Ở đây, gcc compiler driver reads source file hello.c và translates nó into một executable object file hello. Sự chuyển đổi này được thực hiện trong một chuỗi 4 phase như hình 1.3. Các chương trình này được thực hiện trong 4 phase (preprocessor, compiler, assembler, và linker) được gọi chung là compilation system.

* Preprocessing phase.The preprocessor (cpp) modifies the original C program according to directives that begin with the ‘#’ character. For example, the #include command in line 1 of hello.c tells the preprocessor to read the contents of the system header file stdio.h and insert it directly into the program text. The result is another C program, typically with the .i suffix. – Preprocessing phase. The preprocessor (cpp) sửa chương trình C nguyên bản theo các câu lệnh bắt đầu với ký tự ‘#’. Ví dụ, #include command trong line 1 của hello.c tells cho preprocessor để đọc contents của system header file stdio.h và chèn nó trực tiếp vào trong program text. Kết quả là ta có một C program khác, thường đi với hậu tố .i
* Compilation phase. The compiler (cc1) translates the text file hello.i into the text file hello.s, which contains an assembly-language program. This program includes the following definition of function main: - Compilcation phase. Compiler (cc1) translates text file hello.i thành text file hello.s, nó chứa một assembly-language program. Program này bao gồm các định nghĩa theo sau hàm main:

1 main:

2 subq $8, %rsp

3 movl $.LC0, %edi

4 call puts

5 movl $0, %eax

6 addq $8, %rsp

7 ret

Each of lines 2–7 in this definition describes one low-level machinelanguage instruction in a textual form. Assembly language is useful because it provides a common output language for different compilers for different high-level languages. For example, C compilers and Fortran compilers both generate output files in the same assembly language. – Từng dòng từ 2-7 là định nghĩa mô tả một low-level machine-language instruction trong một textual form. Assembly language là hữu ích vì nó cung cấp một common output language cho các compilers khác với high-level languages. Ví dụ, C compilers và Fortan compilers cả hai đều generate output files với assembly language giống nhau.

* Assembly phase. Next, the assembler (as) translates hello.s into machinelanguage instructions, packages them in a form known as a relocatable object program, and stores the result in the object file hello.o. This file is a binary file containing 17 bytes to encode the instructions for function main. If we were to view hello.o with a text editor, it would appear to be gibberish. – Assembly phase. Tiếp theo, assembler (as) translates hello.s thành machine-language instructions, đóng gói chúng trong một form được biết tới như là một relocate object program, và lưu trữ kết quả trong object file hello.o. File này là một binary file đang chứa 17 bytes để encode các instructions cho function main. Nếu chúng ta xem hello.o từ một text editor, nó xuất hiện những thứ kỳ lạ vô nghĩa.
* Linking phase.Notice that our hello program calls the printf function, which is part of the standard C library provided by every C compiler. The printf function resides in a separate precompiled object file called printf.o, which must somehow be merged with our hello.o program. The linker (ld) handles this merging. The result is the hello file, which is an executable object file (or simply executable) that is ready to be loaded into memory and executed by the system. – Linking phase. Lưu ý rằng hello program của chúng ta gọi tới printf function, nó là một phần của standard C library được cung cấp bởi mọi C compiler. The printf function được lưu trữ trong một object file đã được biên dịch trước được gọi là printf.o, nó cần vài cách để được merge với hello program của chúng ta. The linker (ld) handles quá trình merge này. Kết quả là ta có hello file, nó là một executable object file (hoặc đơn giản là executable) đã sẵn sàng để được load vào trong memory và thực thi bởi system.

1.3 It Pays to Understand How Compilation Systems Work

For simple programs such as hello.c, we can rely on the compilation system to produce correct and efficient machine code. However, there are some important reasons why programmers need to understand how compilation systems work: - Bởi chương trình đơn giản như hello.c, chúng ta có thể dựa vào compilation system để cung cấp chính xác và hiệu quả machine code. Tuy nhiên, có một vài lý do quan trọng rằng tại sao các programmers cần phải hiểu cách compilation system hoạt động:

* Optimizing program performance. Modern compilers are sophisticated tools that usually produce good code. As programmers, we do not need to know the inner workings of the compiler in order to write efficient code. However, in order to make good coding decisions in our C programs, we do need a basic understanding of machine-level code and how the compiler translates different C statements into machine code. – Optimizing program performance. Các compiler đời mới là các công cụ tinh vi thường cung cấp code tốt. Với các programmers, chúng ta không cần biết cách hoạt động bên trong của compiler để viết code hiệu quả. Tuy nhiên, theo trình tự quyết định để làm tốt việc coding trong các chương trình C, chúng ta cần phải hiểu basic machine-level code và cách compiler translates các C statements khác thành machine code.

For example, is a switch statement always more efficient than a sequence of if-else statements? How much overhead is incurred by a function call? Is a while loop more efficient than a for loop? Are pointer references more efficient than array indexes? Why does our loop run so much faster if we sum into a local variable instead of an argument that is passed by reference? How can a function run faster when we simply rearrange the parentheses in an arithmetic expression? – Ví dụ, là một switch statements luôn luôn hiệu quả hơn một chuỗi các lệnh if-else? Bao nhiêu chi phí phát sinh với một lệnh gọi function? Là while loop hiệu quả hơn một for loop? Là pointer tham chiếu hiệu quả hơn array indexes? Tại sao vòng lặp của chúng ta chạy nhanh hơn nếu chúng ta sum into một local variable thay vì một argument được chuyền bởi tham chiếu? Cách để có thể một function run nhanh hơn khi chúng ta chỉ cần đơn giản sắp xếp lại các dấu ngoặt trong một biểu thức toán học?

In Chapter 3, we introduce x86-64, the machine language of recent generations of Linux, Macintosh, and Windows computers. We describe how compilers translate different C constructs into this language. In Chapter 5, you will learn how to tune the performance of your C programs by making simple transformations to the C code that help the compiler do its job better. In Chapter 6, you will learn about the hierarchical nature of the memory system, how C compilers store data arrays in memory, and how your C programs can exploit this knowledge to run more efficiently – Trong chương 3, chúng tôi giới thiệu về x86-64, machine language thế hệ gần đây của Linux, Macintosh, và Windows computers. Chúng tôi sẽ mô tả cách các compilers translate cấu trúc C trong language này (x86-64). Trong chương 5 bạn sẽ học cách làm thế nào để điều chỉnh performance trong C program của bạn bằng cách thực hiện một số phép biến đổi cơ bản để giúp cho compiler làm việc tốt hơn. Trong chương 6, bạn sẽ học về bản chất thứ bậc của hệ thống memory, cách C compiler lưu trữ data arrays trong memory và cái cách mà chương trình C của bạn khai thác các kiến thức này để run hiệu quả hơn.

* Understanding link-time errors.In our experience, some of the most perplexing programming errors are related to the operation of the linker, especially when you are trying to build large software systems. For example, what does it mean when the linker reports that it cannot resolve a reference? What is the difference between a static variable and a global variable? What happens if you define two global variables in different C files with the same name? What is the difference between a static library and a dynamic library? Why does it matter what order we list libraries on the command line? And scariest of all, why do some linker-related errors not appear until run time? You will learn the answers to these kinds of questions in Chapter 7. – Hiểu về link-time errors. Theo kinh nghiệm của chúng tôi, một vài lỗi chương trình phổ biến có liên quan đến hoạt động của linker, đặc biệt là khi bạn cố gắng build một software systems lớn. Ví dụ, nghĩa là gì khi linker reports rằng nó không thể giải quyết một tham chiếu? Sự khác biệt giữa một static variable và một global variable? Cái gì xảy rra nếu bạn define 2 biến global trong các C files khác nhau với cùng một cái tên? Sự khác nhau giữa một static library và một dynamic library? Tại sao thứ tự chúng ta liệt kê các thư viện trên các dòng lệnh quan trọng? Và đáng sợ nhất, vì sao một vài linker-related errors vẫn không xuất hiện khi run time? Bạn sẽ có câu trả lời cho các câu hỏi này trong chương 7.
* Avoiding security holes. For many years, buffer overflow vulnerabilities have accounted for many of the security holes in network and Internet servers. These vulnerabilities exist because too few programmers understand the need to carefully restrict the quantity and forms of data they accept from untrusted sources. A first step in learning secure programming is to understand the consequences of the way data and control information are stored on the program stack. We cover the stack discipline and buffer overflow vulnerabilities in Chapter 3 as part of our study of assembly language. We will also learn about methods that can be used by the programmer, compiler, and operating system to reduce the threat of attack. – Tránh các security holes. Trong nhiều năm, buffer overflow vulnerabilities được thống kê cho nhiều lỗ hổng bảo mật trong network và Internet servers. Các lỗ hổng này tồn tại bởi vì có rất ít các programmers hiểu rằng cần cẩn thận hạn chế số lượng và các forms của data họ cho phép từ các nguồn không đáng tin cậy. Bước đầu tiên trong việc học secure programming là hiểu hậu quả các cách data và thông tin điều khiển được lưu trữ trong program stack. Chúng ta sẽ nói về stack discipline và buffer overflow vulnerabilities trong chương 3 như một phần của việc học assembly language. Chúng ta cũng sẽ học về methods có thể được sử dụng bởi các programmer, compiler, và OS để giảm nguy cơ tấn công.

1.4 Processors Read and Interpret Instructions Stored in Memory

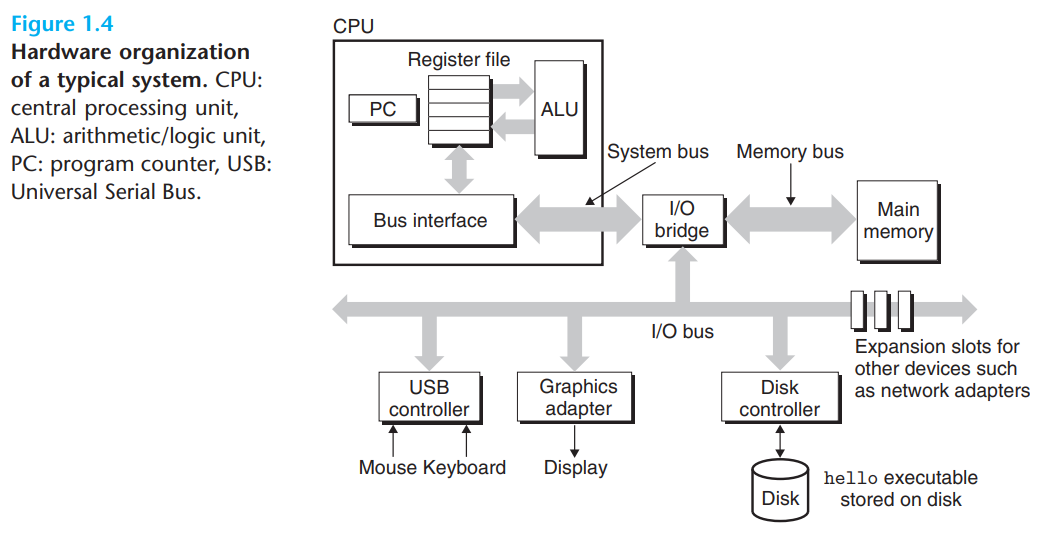
At this point, our hello.c source program has been translated by the compilation system into an executable object file called hello that is stored on disk. To run the executable file on a Unix system, we type its name to an application program known as a shell: - Tới thời điểm này, hello.c source program của chúng ta được translated bởi compilcation system thành một executable object file được gọi là hello được lưu trữ trên disk. Để run executable file trên một Unix system, chúng ta type tên của nó tới một application program được biết tới như là shell:

linux> ./hello

hello, world

linux>

The shell is a command-line interpreter that prints a prompt, waits for you to type a command line, and then performs the command. If the first word of the command line does not correspond to a built-in shell command, then the shell assumes that it is the name of an executable file that it should load and run. So in this case, the shell loads and runs the hello program and then waits for it to terminate. The hello program prints its message to the screen and then terminates. The shell then prints a prompt and waits for the next input command line. – The shell là một command-line interperter in ra một lời nhắc, chờ cho bạn nhập vào một command line, và sau đó thực hiện command. Nếu từ đầu tiên của command line không giống với các câu lệnh shell được build sẵn, thì sau đó shell sẽ giả định rằng nó là tên của một executable file và rằng nó có thể load và run. Vì thế trong trường hợp này, shell loads và runs hello program và sau đó đợi nó chấm dứt. The hello program in message của nó ra screen và sau đó chấm dứt. Shell sau đó prints một promt và đợi cho next input command line.



1.4.1 Hardware Organization of a System

To understand what happens to our hello program when we run it, we need to understand the hardware organization of a typical system, which is shown in Figure 1.4. This particular picture is modeled after the family of recent Intel systems, but all systems have a similar look and feel. Don’t worry about the complexity of this figure just now. We will get to its various details in stages throughout the course of the book. – Để hiểu cái gì xảy ra với chương trình hello của chúng ta khi ta chạy nó, chúng ta cần hiểu về hardware organization của một system thông thường, xem hình 1.4. Đây là hình ảnh cụ thể về các mẫu được ra mắt gần đây của Intel systems, nhưng tất cả các systems khác đều có cái nhìn và cảm nhận tương tự với nó. Đừng sợ về sự phức tạp của hình ảnh ấy bây giờ. Chúng ta sẽ lấy từng phần chi tiết của nó trong các giai đoạn xuyên suốt course của sách.

Buses

Running throughout the system is a collection of electrical conduits called buses that carry bytes of information back and forth between the components. Buses are typically designed to transfer fixed-size chunks of bytes known as words. The number of bytes in a word (the word size) is a fundamental system parameter that varies across systems. Most machines today have word sizes of either 4 bytes (32 bits) or 8 bytes (64 bits). In this book, we do not assume any fixed definition of word size. Instead, we will specify what we mean by a “word” in any context that requires this to be defined. – Running xuyên suốt system là một tập hợp của các ống dẫn điện được gọi là các buses, nó mang theo các bytes của thông tin qua lại giữa các components. Buses thường được thiết kế để transfer các khối fixed-size của các bytes thường được biết tới là words. Số bytes trong một word (là word size) là biến system cơ bản khác nhau giữa các hệ thống. Hầu hết các machines ngày nay có word sizes là 4 bytes (32 bits) hoặc 8 bytes (64 bits). Trong cuốn sách này, chúng tôi không giả định bất kỳ định nghĩa cố định nào của word size. Thay vào đó, chúng tôi sẽ chỉ định cái gì là “word” trong bất cứ ngữ cảnh nào yêu cầu điều này được định nghĩa.

I/O Devices

Input/output (I/O) devices are the system’s connection to the external world. Our example system has four I/O devices: a keyboard and mouse for user input, a display for user output, and a disk drive (or simply disk) for long-term storage of data and programs. Initially, the executable hello program resides on the disk. – Input/Output (I/O) devices là các connection của hệ thống để tương tác với thế giới bên ngoài. Ví dụ như hệ thống của chúng ta có 4 I/O devices: một keyboard và mouse cho user input, một display cho user output, và một disk driver (hoặc đơn giản là disk) cho lưu trữ dài hạn data và programs. Ban đầu, executable file hello program lưu trữ trong disk.

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. Controllers are chip sets in the device itself or on the system’s main printed circuit board (often called the motherboard). An adapter is a card that plugs into a slot on the motherboard. Regardless, the purpose of each is to transfer information back and forth between the I/O bus and an I/O device. – Từng I/O device được kết nối tới I/O bus bằng 2 cách: một controller hoặc một adapter. Sự phân biệt giữa 2 chúng chủ yếu là cách packaging. Controllers là chip sets trong device của chính nó hoặc trên main printed circuit board của system (thường được gọi là motherboard). Một adapter là một card được gắn vào trong một slot trên motherboard. Bất kể, mục đích của mỗi chúng là transfer information qua lại giữa I/O bus và một I/O device.

Chapter 6 has more to say about how I/O devices such as disks work. In Chapter 10, you will learn how to use the Unix I/O interface to access devices from your application programs. We focus on the especially interesting class of devices known as networks, but the techniques generalize to other kinds of devices as well. – Chương 6 sẽ nói thêm về cách I/O devices như disks hoạt động. Trong chương 10, bạn sẽ học cách sử dụng Unix I/O interface để truy cập devices từ ứng dụng của bạn. Chúng tôi tập trung đặc biệt tới lớp devices thú vị là networks, nhưng các kỹ thuật này cũng tổng quát hoá cách hoạt động của các thiết bị khác.

Main Memory

The main memory is 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, memory is organized as a linear array of bytes, each with its own unique address (array index) starting at zero. – Main memory là một thiết bị lưu trữ tạm thời để giữ cho cả hai program và data nó nắm giữ trong khi processor đang thực thi chương trình. Về mặt vật lý, main memory bao gồm một tập hợp các chips random access memory (DRAM). Về mặt logic, memory được tổ chức như một array các bytes tuyến tính, từng phần tử trong chúng sở hữu unique address (array index) bắt đầu với 0.

In general, each of the machine instructions that constitute a program can consist of a variable number of bytes. The sizes of data items that correspond to C program variables vary according to type. For example, on an x86-64 machine running Linux, data of type short require 2 bytes, types int and float 4 bytes, and types long and double 8 bytes. Chapter 6 has more to say about how memory technologies such as DRAM chips work, and how they are combined to form main memory. – Thông thường, từng machine instructions cấu thành một program có thể bao gồm một biến số của các bytes. Kích thước của data items giống với các biến của chương trình C thay đổi theo type. Ví dụ, trên một x86-6 machine running Linux, data của type short yêu cầu 2 bytes, types int và float yêu cầu 4 bytes, và types long và double yêu cầu 8 bytes. Chương 6 sẽ nói thêm về cách các kỹ thuật memory như là DRAM chips hoạt động, và cách chúng kết hợp để cấu thành main memory.

Processor

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-size storage device (or register) called the program counter (PC). At any point in time, the PC points at (contains the address of) some machine-language instruction in main memory – The central processing unit (CPU), hoặc đơn giản là processor, là engine để interprets (hoặc executes) instructions được lưu trong main memory. Cốt lõi của nó là một thiết bị lưu trữ (hoặc register) word-size được gọi là program counter (PC). Tại bất kỳ thời điểm nào, PC (program counter) trỏ vào (chứa địa chỉ) của một vài tập lệnh mã máy trong main memory.

From the time that power is applied to the system until the time that the power is shut off, a processor repeatedly executes the instruction pointed at by the program counter and updates the program counter to point to the next instruction. A processor appears to operate according to a very simple instruction execution model, defined by its instruction set architecture. In this model, instructions execute in strict sequence, and executing a single instruction involves performing a series of steps. The processor reads the instruction from memory pointed at by the program counter (PC), interprets the bits in the instruction, performs some simple operation dictated by the instruction, and then updates the PC to point to the next instruction, which may or may not be contiguous in memory to the instruction that was just executed. – Từ thời điểm cấp nguồn vào hệ thống cho tới khi nguồn được tắt, một processor lặp lại việc thực thi các tập lệnh được trỏ vào bởi program counter và cập nhật program counter point tới tập lệnh tiếp theo. Một processor dường như hoạt động theo một mô hình thực thi rất đơn giản, được định nghĩa bởi kiến trúc tập lệnh của nó. Trong model này, các instructions thực thi theo một trình tự nghiêm ngặt, và thực thi một single instructions liên quan đến việc thực hiện một loạt các bước. The processor đọc các tập lệnh từ memory được trỏ bởi PC (program counter), thông dịch các bits trong instruction, thực hiện một vài toán tử đơn giản được ra lệnh bởi tập lệnh, và sau đó updates PC point tới instruction tiếp theo, điều này có thể hoặc không liền kề với instruction trong memory vừa được thực thi.

There are only a few of these simple operations, and they revolve around main memory, the register file, and the arithmetic/logic unit (ALU). The register file is a small storage device that consists of a collection of word-size registers, each with its own unique name. The ALU computes new data and address values. Here are some examples of the simple operations that the CPU might carry out at the request of an instruction: - Chỉ có một vài toán tử đơn giản, và chúng có liên quan đến xung quanh main memory, register file, và arithmetic/logic unit (ALU). The register file là một storage device nhỏ bao gồm một collection các word-size registers, mỗi chúng đều có tên duy nhất. The ALU tính toán dữ liệu với và các giá trị address. Ở đây có một vài ví dụ về các hoạt động đơn giản mà CPU có thể mang theo với request của một instruction:

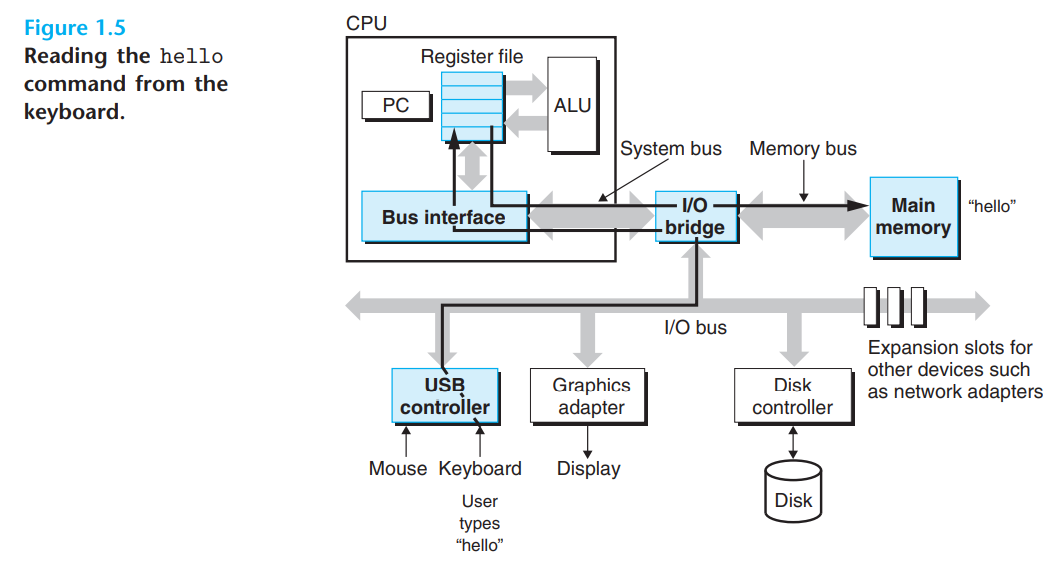
* Load: Copy a byte or a word from main memory into a register, overwriting the previous contents of the register. – Load: Copy một byte hoặc một word từ main memory thành một register, ghi đè contents trước đó của register.
* Store: Copy a byte or a word from a register to a location in main memory, overwriting the previous contents of that location – Store: Copy một byte hoặc một word từ một register tới một location trong main memory, ghi đè contents trước đó của location.
* Operate: Copy the contents of two registers to the ALU, perform an arithmetic operation on the two words, and store the result in a register, overwriting the previous contents of that register – Operate: Copy contents của 2 registers từ ALU, thực hiện một toán tử số học trên 2 words, và store kết quả trong một register, ghi đè contents trước đó của register.
* Jump: Extract a word from the instruction itself and copy that word into the program counter (PC), overwriting the previous value of the PC. – Trích xuất một word từ chính instruction của nó và copy word này vào trong program counter (PC), ghi đè giá trị trước đó của PC.

We say that a processor appears to be a simple implementation of its instruction set architecture, but in fact modern processors use far more complex mechanisms to speed up program execution. Thus, we can distinguish the processor’s instruction set architecture, describing the effect of each machine-code instruction, from its micro-architecture, describing how the processor is actually implemented. When we study machine code in Chapter 3, we will consider the abstraction provided by the machine’s instruction set architecture. Chapter 4 has more to say about how processors are actually implemented. Chapter 5 describes a model of how modern processors work that enables predicting and optimizing the performance of machine-language programs. – Chúng tôi nói rằng một processor xuất hiện để khiến việc triển khai các kiến trúc tập lệnh trở nên đơn giản, nhưng sự thật rằng các processors đời mới sử dụng cơ chế phức tạp hơn để speed up program execution. Như vậy, chúng ta có thể phân biệt kiến trúc tập lệnh của các processor, mô tả ảnh hưởng của từng machine-code instruction, từ micro-architecture của nó, mô tả cách processor thực sự được implemented. Khi chúng ta học về machine code trong chương 3, chúng ta sẽ xem xét phần trừu tượng được cung cấp bởi instruction set architecture của machine. Chương 4 nói thêm về cách processor thường được triển khai. Chương 5 mô tả một model của cách processor hiện đại cho phép dự đoán và tối ưu hiệu suất của machine-language programs.

1.4.2 Running the hello Program

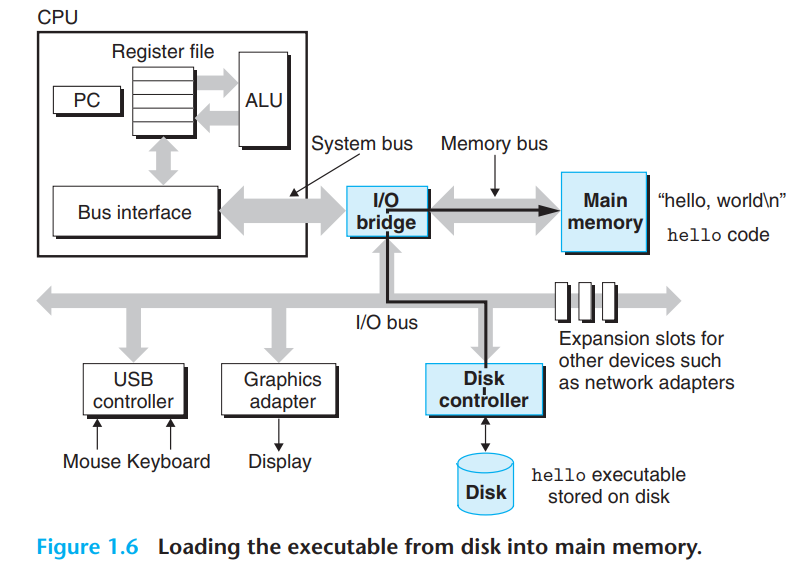
Given this simple view of a system’s hardware organization and operation, we can begin to understand what happens when we run our example program. We must omit a lot of details here that will be filled in later, but for now we will be content with the big picture. – Đưa góc nhìn đơn giản về một system’s hardware organization và operation, chúng ta có thể bắt đầu hiểu cái gì xảy ra khi chúng ta run chương trình ví dụ. Chúng ta cần bỏ qua rất nhiều chi tiết ở đây và sẽ lấp đầy sau, nhưng ngay bây giờ, chúng ta sẽ có content với bức tranh toàn cảnh.

Initially, the shell program is executing its instructions, waiting for us to type a command. As we type the characters ./hello at the keyboard, the shell program reads each one into a register and then stores it in memory, as shown in Figure 1.5. – Ban đầu, shell program thực thi các instructions của nó, chờ đợi cho chúng ta gõ command. Như chúng ta đã gõ ./hello bằng keyboard, shell program read từng thứ một bên trong một register và sau đó lưu trữ nó trong memory, như hình 1.5

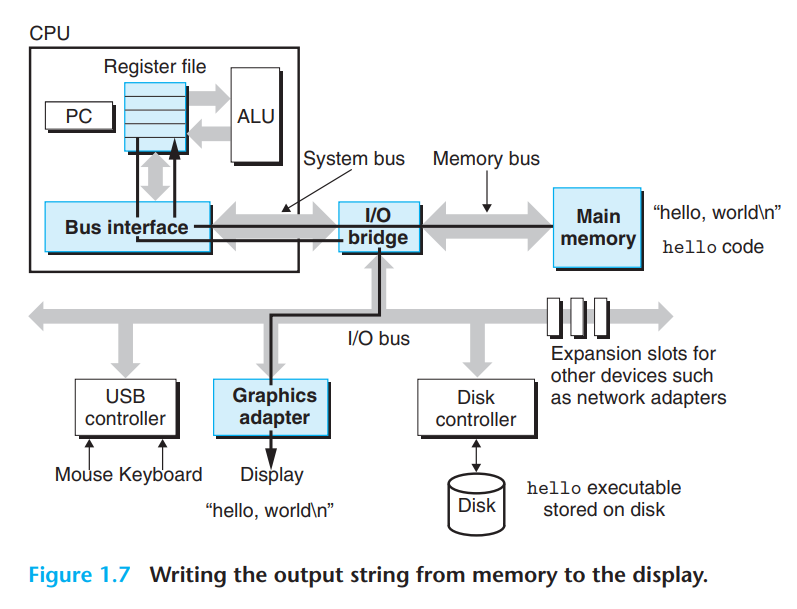


When we hit the enter key on the keyboard, the shell knows that we have finished typing the command. The shell then loads the executable hello file by executing a sequence of instructions that copies the code and data in the hello object file from disk to main memory. The data includes the string of characters hello, world\n that will eventually be printed out. – Khi chúng ta hit enter key on the keyboard, shell biết rằng chúng ta đã kết thúc việc gõ lệnh. Shell sau đó loads excutable hello file bằng cách thực thi một chuỗi các instructions đã được copies code và data trong hello object file từ disk tới main memory. Data bao gồm chuỗi các ký tự hello, world\n cuối cùng sẽ được in ra.

Using a technique known as direct memory access (DMA, discussed in Chapter 6), the data travel directly from disk to main memory, without passing through the processor. This step is shown in Figure 1.6. – Bằng cách sử dụng kỹ thuật được biết tới như là direct memory access (DMA, được thảo luận trong chương 6), data travel trực tiếp từ disk tới main memory, mà không cần passing qua processor.



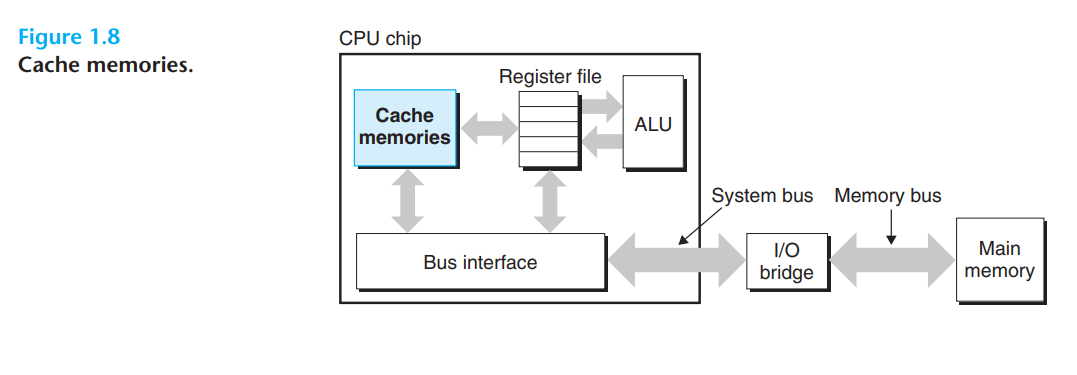
Once the code and data in the hello object file are loaded into memory, the processor begins executing the machine-language instructions in the hello program’s main routine. These instructions copy the bytes in the hello, world\n string from memory to the register file, and from there to the display device, where they are displayed on the screen. This step is shown in Figure 1.7 – Một lần code và data trong hello object file được loaded vào trong memory, processor bắt đầu thực thi các machine-language instructions trong chương trình hello theo thứ tự trong main. Các instructions này copy các bytes hello, world\n string từ memory tới register file, và từ đó display ra device, nơi chúng được in ra màn hình. Các bước được thực hiện như trong hình 1.7.



1.5 Caches Matter

An important lesson from this simple example is that a system spends a lot of time moving information from one place to another. The machine instructions in the hello program are originally stored on disk. When the program is loaded, they are copied to main memory. As the processor runs the program, instructions are copied from main memory into the processor. – Một bài học quan trọng từ ví dụ đơn giản này là một hệ thống dành rất nhiều time moving information từ một địa điểm này tới một địa điểm khác. Các machine instructions trong hello program được lưu trữ ban đầu trên disk. Khi program được load, chúng được copied tới main memory. Rồi processor chạy program, các instructions được copied từ main memory sẽ vào trong processor.

Similarly, the data string hello,world\n, originally on disk, is copied to main memory and then copied from main memory to the display device. From a programmer’s perspective, much of this copying is overhead that slows down the “real work” of the program. Thus, a major goal for system designers is to make these copy operations run as fast as possible. – Tương tự, data string hello, world\n, ban đầu trên disk, được copied tới main memory và sau đó được copy từ main memory tới display device. Từ một quan điểm của một programmer, phần lớn việc sao chép này làm chậm đi “sự làm việc thật sự” của chương trình. Như vậy, một mục tiêu chính cho system designers đó là làm sao để copy các hoạt động này và chạy nhanh nhất có thể.



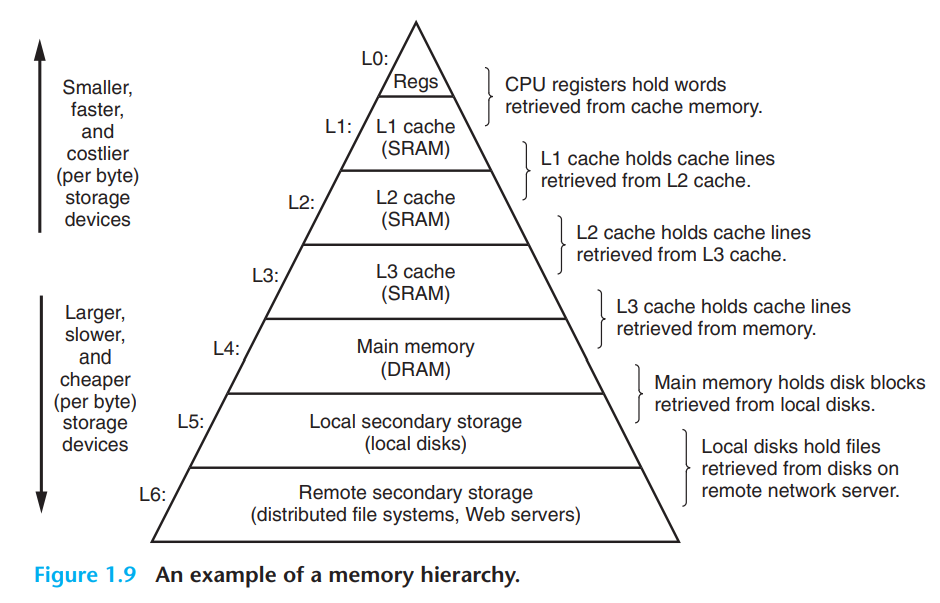
Because of physical laws, larger storage devices are slower than smaller storage devices. And faster devices are more expensive to build than their slower counterparts. For example, the disk drive on a typical system might be 1,000 times larger than the main memory, but it might take the processor 10,000,000 times longer to read a word from disk than from memory. – Bởi theo quy tắc vật lý, thiết bị lưu trữ lớn hơn sẽ chậm hơn thiết bị lưu trữ nhỏ hơn. Và các thiết bị nhanh hơn sẽ tốn nhiều tài nguyên để build hơn các thiết bị chậm hơn. Ví dụ, disk drive trên các hệ thống thông thường có thể lớn hơn 1000 lần so với main memory, nhưng nó có thể tốn lâu hơn 10,000,000 lần để đọc một word từ disk hơn là từ memory.

Similarly, a typical register file stores only a few hundred bytes of information, as opposed to billions of bytes in the main memory. However, the processor can read data from the register file almost 100 times faster than from memory. Even more troublesome, as semiconductor technology progresses over the years, this processor–memory gap continues to increase. It is easier and cheaper to make processors run faster than it is to make main memory run faster. – Tương tự, một register file thông thương chỉ lưu trữ vài trăm bytes của information, ngược lại so với hàng tỷ bytes trong main memory. Tuy nhiên, processor có thể đọc data từ register file hầu hết là nhanh hơn 100 lần so với từ memory. Rắc rối hơn nữa, công nghệ bán dẫn tiến bộ qua từng năm, khoảng cách giữa processor-memory tiếp tục tăng. Nó sẽ dễ dàng và rẻ hơn để làm cho processor chạy nhanh hơn so với việc làm cho main memory chạy nhanh hơn.

To deal with the processor–memory gap, system designers include smaller, faster storage devices called cache memories (or simply caches) that serve as temporary staging areas for information that the processor is likely to need in the near future. Figure 1.8 shows the cache memories in a typical system. An L1 cache on the processor chip holds tens of thousands of bytes and can be accessed nearly as fast as the register file. A larger L2 cache with hundreds of thousands to millions of bytes is connected to the processor by a special bus. – Để đối phó với khoảng cách giữa processor-memory, các system designers thêm vào các storage devices nhỏ hơn, nhanh hơn được gọi là cache memories (đơn giản là caches) đóng vai trò là các staging areas cho information mà processor có thể cần đến trong tương lai gần. Hình 1.8 shows cache memories trong một hệ thống thông thường. Một L1 cache trên processor chip giữ hàng chục nghìn bytes và có thể được truy cập gần như tức thời với register file. Một L2 cache lớn hơn với hàng trăm nghìn đến hàng triệu bytes được connect tới processor bằng một bus đặc biệt.

It might take 5 times longer for the processor to access the L2 cache than the L1 cache, but this is still 5 to 10 times faster than accessing the main memory. The L1 and L2 caches are implemented with a hardware technology known as static random access memory (SRAM). Newer and more powerful systems even have three levels of cache: L1, L2, and L3. The idea behind caching is that a system can get the effect of both a very large memory and a very fast one by exploiting locality, the tendency for programs to access data and code in localized regions. By setting up caches to hold data that are likely to be accessed often, we can perform most memory operations using the fast caches. – Nó có thể lâu hơn 5 lần cho việc processor truy cập tới L2 so với L1, nhưng nó lại nhanh hơn từ 5 – 10 lần để truy cập đến main memory. L1 và L2 caches được triển khai với một công nghệ phần cứng được biết tới là static random access memory (SRAM). Mới hơn và mạnh mẽ hơn là các hệ thống có 3 level caches: L1, L2 và L3. Ý tưởng đằng sau việc caching là một system có thể lấy hiệu quả của cả very large memory và very fast bằng cách exploiting locality – khai thác cục bộ, xu hướng cho các chương trình để access data và code trong localized regions. Bằng cách setting up caches để hold data rất có thể được truy cập thường xuyên, chúng ta có thể thực hiện hầu hết các memory operations bằng cách sử dụng fast caches.

One of the most important lessons in this book is that application programmers who are aware of cache memories can exploit them to improve the performance of their programs by an order of magnitude. You will learn more about these important devices and how to exploit them in Chapter 6. – Một trong những bài học quan trọng nhất trong cuốn sách này là các lập trình viên ứng dụng nhận thức được cache memories để có thể khai thác chúng nhằm cải thiện performance trong chương trình của họ theo mức độ nhất định. Bạn sẽ học thêm về những thiết bị quan trọng và cách khai thác chúng trong chương 6.



1.6 Storage Devices Form a Hierarchy

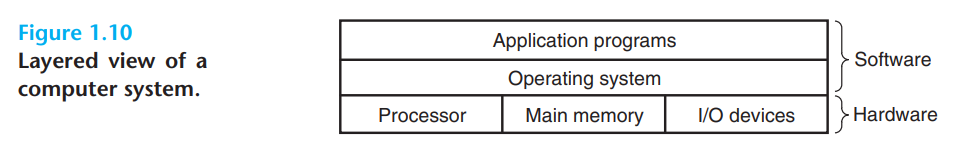
This notion of inserting a smaller, faster storage device (e.g., cache memory) between the processor and a larger, slower device (e.g., main memory) turns out to be a general idea. In fact, the storage devices in every computer system are organized as a memory hierarchy similar to Figure 1.9. As we move from the top of the hierarchy to the bottom, the devices become slower, larger, and less costly per byte. The register file occupies the top level in the hierarchy, which is known as level 0 or L0. We show three levels of caching L1 to L3, occupying memory hierarchy levels 1 to 3. Main memory occupies level 4, and so on. – Khái niệm này là việc chèn một thiết bị lưu trữ nhỏ hơn, nhanh hơn (ví dụ: cache memory) giữa processor và một thiết bị lưu trữ lớn hơn, chậm hơn (ví dụ: main memory) hoá ra là một ý tưởng chung. Trong thực tế, các thiết bị lưu trữ trong mọi hệ thống máy tính được tổ chức như một memory hierarchy như hình 1.9. Chúng ta đi từ đầu đến cuối hierarchy, các thiết bị trở nên chậm hơn, lớn hơn, và tốn ít tài nguyên hơn cho mỗi byte. Register file được đặt tại đầu level trong hierarchy, tương ứng với level 0 hoặc L0. Chúng ta xem tiếp 3 level của caching từ L1 đến L3, chiếm giữ level 1 đến 3 của memory hierarchy. Main memory chiếm giữ level 4, ...

The main idea of a memory hierarchy is that storage at one level serves as a cache for storage at the next lower level. Thus, the register file is a cache for the L1 cache. Caches L1 and L2 are caches for L2 and L3, respectively. The L3 cache is a cache for the main memory, which is a cache for the disk. On some networked systems with distributed file systems, the local disk serves as a cache for data stored on the disks of other systems. – Ý tưởng chính của một memory hierarchy là tại một level nào đó đóng vai trò như một cache cho storage cho cấp thấp tiếp theo. Như vậy, register file là một cache cho L1 cache. Caches L1 và L2 là caches cho L2 và L3, tương tự. L3 cache là một cache cho main memory, nó cũng là cache cho disk. Trên một vài hệ thống network với các file hệ thống được phân phối, local disk serves là một cache cho data được lưu trữ của các system khác.

Just as programmers can exploit knowledge of the different caches to improve performance, programmers can exploit their understanding of the entire memory hierarchy. Chapter 6 will have much more to say about this. - Cũng giống như các lập trình viên có thể khai thác kiến thức về các bộ nhớ đệm khác nhau để cải thiện hiệu suất, các lập trình viên có thể khai thác hiểu biết của họ về toàn bộ hệ thống phân cấp bộ nhớ. Chương 6 sẽ có nhiều điều hơn để nói về điều này

1.7 The Operating System Manages the Hardware

Back to our hello example. When the shell loaded and ran the hello program, and when the hello program printed its message, neither program accessed the keyboard, display, disk, or main memory directly. Rather, they relied on the services provided by the operating system. We can think of the operating system as a layer of software interposed between the application program and the hardware, as shown in Figure 1.10. All attempts by an application program to manipulate the hardware must go through the operating system. – Trở lại với ví dụ hello của chúng ta. Khi shell đã load và đã chạy chương trình hello, và khi chương trình hello đã in ra message của nó, chương trình đã không truy cập keyboard, display, disk hay main memory một cách trực tiếp. Thay vào đó, nó dựa vào các services được cung cấp bởi OS. Chúng ta có thể nghĩ về OS như một layer of software xen kẽ giữa các chương trình ứng dụng và hardware, như hình 1.10. Tất cả các nỗ lực của một application program để thao tác với hardware cần thông qua hệ điều hành.



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. – OS có hai mục đích chính: (1) để bảo vệ hardware từ sự lạm dụng từ các applications đang chạy và (2) để cung cấp cho các applications với cơ chế đồng điệu và đơn giản cho các thao tác phức tạp và thông thường các low-level hardware devices là rất khác nhau.

The operating system achieves both goals via the fundamental abstractions shown in Figure 1.11: processes, virtual memory, and files. As this figure suggests, files are abstractions for I/O devices, virtual memory is an abstraction for both the main memory and disk I/O devices, and processes are abstractions for the processor, main memory, and I/O devices. We will discuss each in turn. – OS đạt được các mục tiêu thông qua sự trừu tượng cơ bản như hình 1.11: processes, virtual memory, và files. Như hình 1.11 đã gợi ý, files và abstractions cho I/O devices, virtual memory là một abstraction cho cả main memory và disk I/O devices, và processes là abstraction của processor, main memory và I/O devices. Chúng ta sẽ thảo luận về từng cái một.

1.7.1 Processes

When a program such as hello runs on a modern system, the operating system provides the illusion that the program is the only one running on the system. The program appears to have exclusive use of both the processor, main memory, and I/O devices. The processor appears to execute the instructions in the program, one after the other, without interruption. And the code and data of the program appear to be the only objects in the system’s memory. These illusions are provided by the notion of a process, one of the most important and successful ideas in computer science. – Khi một program như hello run trên một system hiện đại, OS cung cấp giả định rằng chương trình chỉ chạy trên một system. Program dường như độc quyền sử dụng cả processor, main memory, và I/O devices. Processor xuất hiện để thực thi các instructions trong program, từ instruction này đến cái khác, không bị gián đoạn. Và code và data của chương trình xuất hiện để trở thành các đối tượng duy nhất trong memory của system. Những giả định này được cung cấp bởi khái niệm của một process, một trong những ý tưởng quan trọng và hữu ích bậc nhất trong computer science.