第1章 计算机系统漫游

hello world(用C语言打印hello world)

通过跟踪hello程序来对系统的学习

hello的生命周期

代码文件被程序员创建,到在系统上运行,输出hello信息

信息就是 位 + 上下文

源程序(代码文件)实际是由0和1组成的位(比特)序列,8位=1字节,每个字节表示某些文本符号

大部分现在计算机都使用ASCII标准来表示文本字符,即一个字节大小的整数值表示一个字符

如 A ->65 , # -> 35 \n(换行符) -> 10

像这种只由ASCII构成的文件为文本文件,其他文件为二进制文件

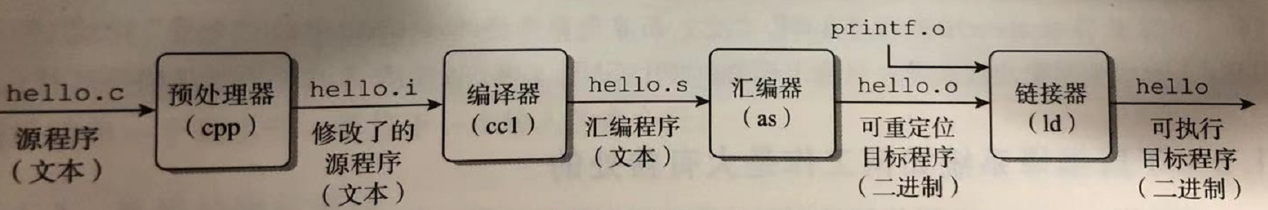
系统中所有的信息包括磁盘文件,内存文件,网络传送的数据都是由一串比特表示的

区别不同数据对象的唯一方法是要看上下文,在不同的上下文中,一个同样的字节序列可能表示是一个整数或浮点数或字符串或机器指令

程序被其他程序翻译成不同的格式

Linux> gcc -o hello hello.c

hello为最终生成的执行程序, hello.c为源代码程序



预处理阶段

预处理器根据以#开头的命令,修改原始的C程序,如 #include <stdio.h>是告诉预处理器充取stdio.h的内容并插入文本中,结果就得到了了以.i作为文件扩展名的另一个C程序

编译阶段

编译器将hello.i文本文件翻译成hello.s文本文件,里面包含一个汇编语言程序(包含函数main的定义)

汇编语言为不同高级语言的不同编译器提供了通用的输出语言(如c,python编译器产生的是一样的汇编语言)

汇编阶段

汇编器将hello.s翻译成机器语言指令,并打包成叫可重定位目标程序的格式保存在hello.o文件(二进制文件),用文本编译器打开是乱码)中

链接阶段

里面调用了prinft函数(标准C库的一个函数,存在于一个叫prinft.o预编好的目标文件中),要以某种方式合并到hello.o中,链接器(ld)就是处理这种合并,最终得到hello执行文件,可加载到内存中由系统执行

系统的硬件组成

总线

一组电子管道,携带信息字长(字word)在各部件间传递,字长可以是4或8字节

I/O设备

输入输出设备,每个I/O设备通过控制器/适配器与I/O总线相连,控制器与适配器的区别在于封装方式,控制器是主板上的芯片组,而适配器是扩展槽上的卡,两者的作用都是在I/O设备与I/O总线间传递信息

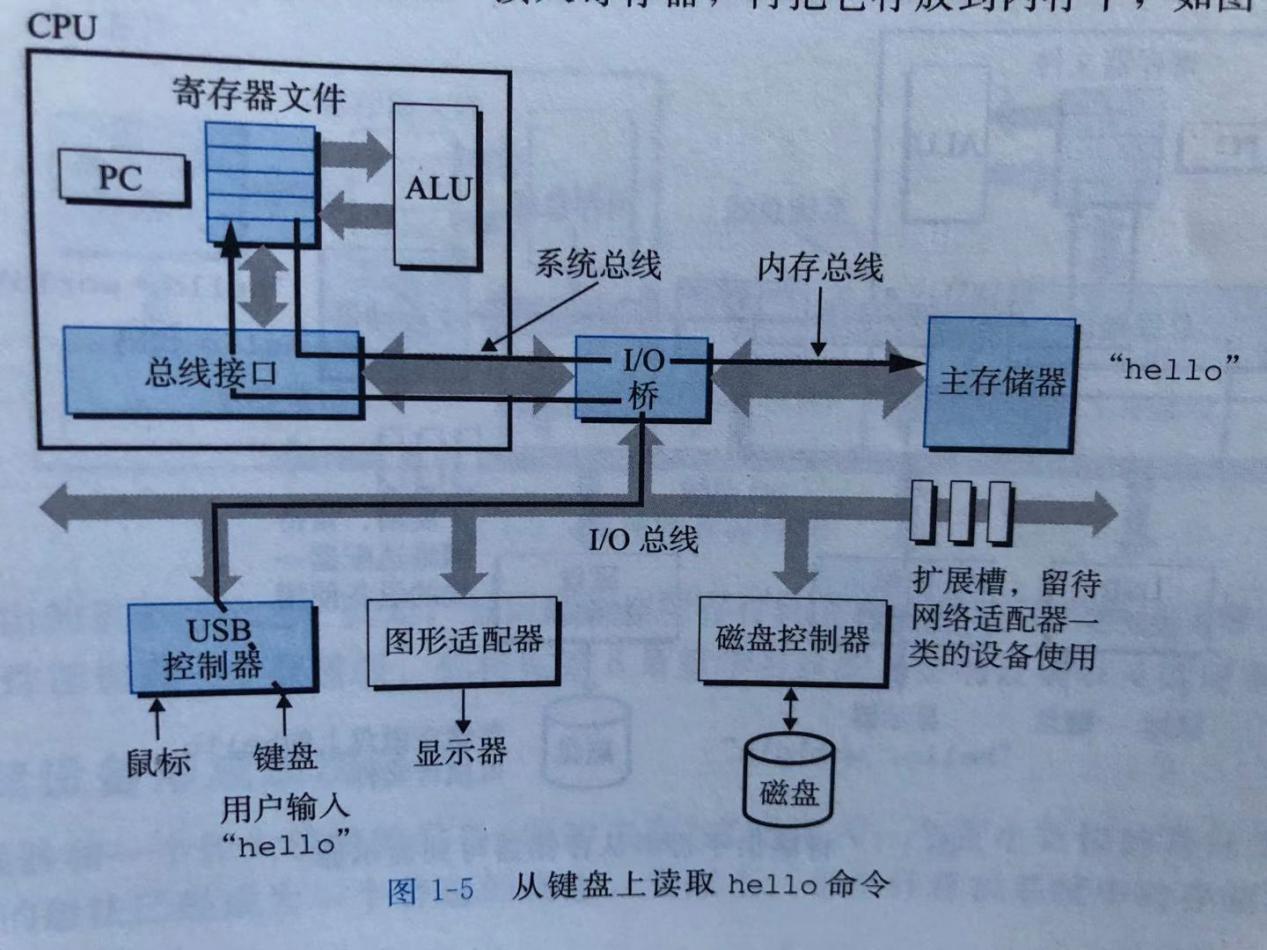
主存

CPU执行程序时,用来存放程序与程序要处理的数据的临时的设备,由DRAM(动态随机访问存储)的芯片组成

处理器

CPU,解释或执行存储在主存上的指令

运行hello时系统硬件如何协调实现



首先通过输入设备键盘在shell中 ./hello 回车后,shell会把字符读入寄存器,再放到内存

然后shell执行一系列指令来加载可执行的hello(把hello的代码和数据复制到主存)

CPU执行主存中hello程序的main程序中的机器语言指令,指令会将’hello world\n’字符串复制到寄存器文件,再从寄存器文件复制到输出设备(显示器)

高速缓存

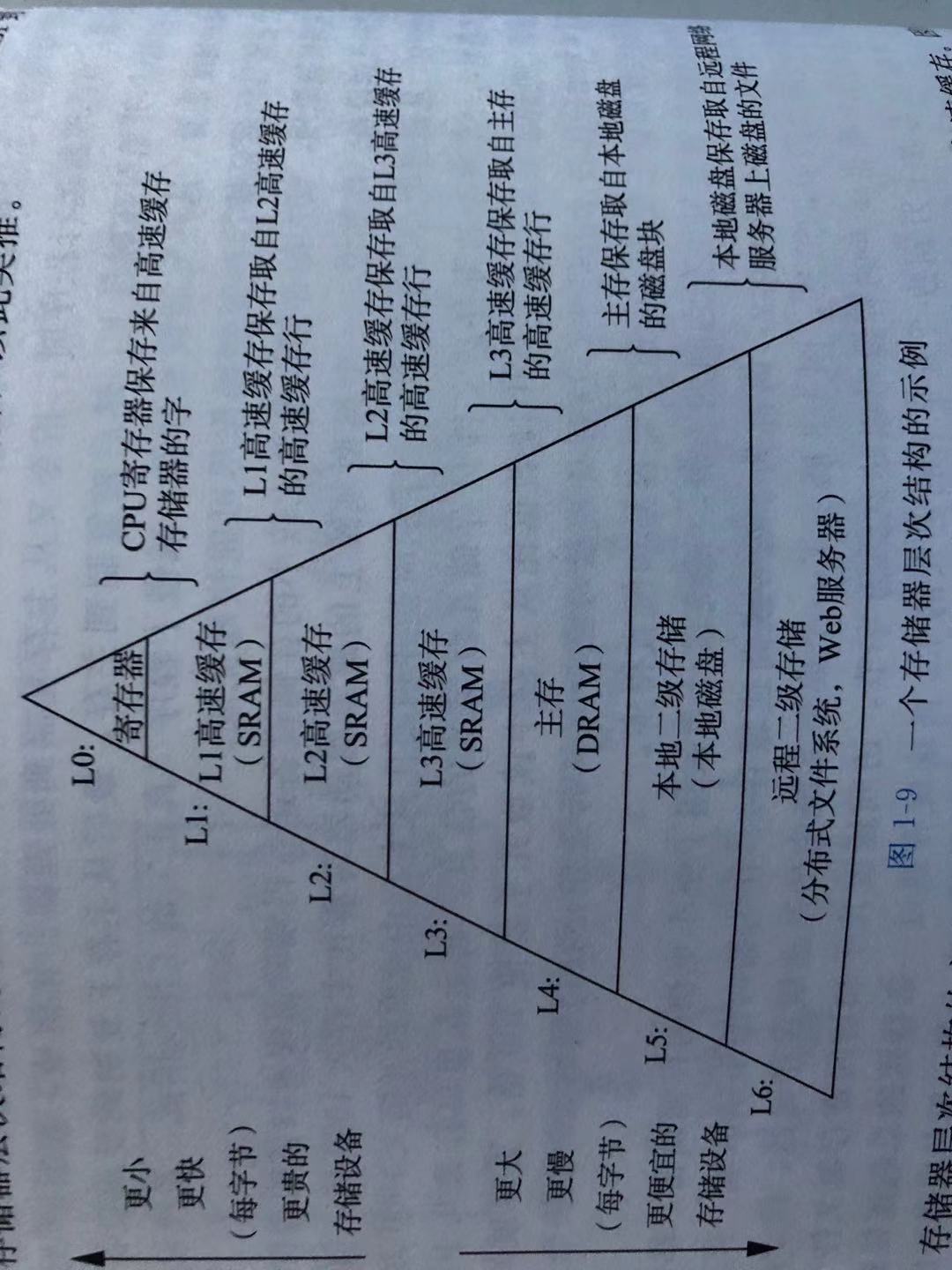
通过执行hello执行程序,知道代码与数据要从碰盘复制到主存,又要复制寄存器,复制需要开销,且寄存器,主存,磁盘之前的存取速度不一样(越大速度超慢,反之相等),为了让系统整体更快,就有了高速缓存的出现

高速缓存L1(与寄存器速度差不多一样快),L2容量更大,速度比L1慢但仍比主存快,更好的CPU可能还有L3

L1 L2是通过用SRAM(静态随机访问存储器的硬件技术实现在)

通过在高速缓存里存放可能经常访问的数据来提升整体速度

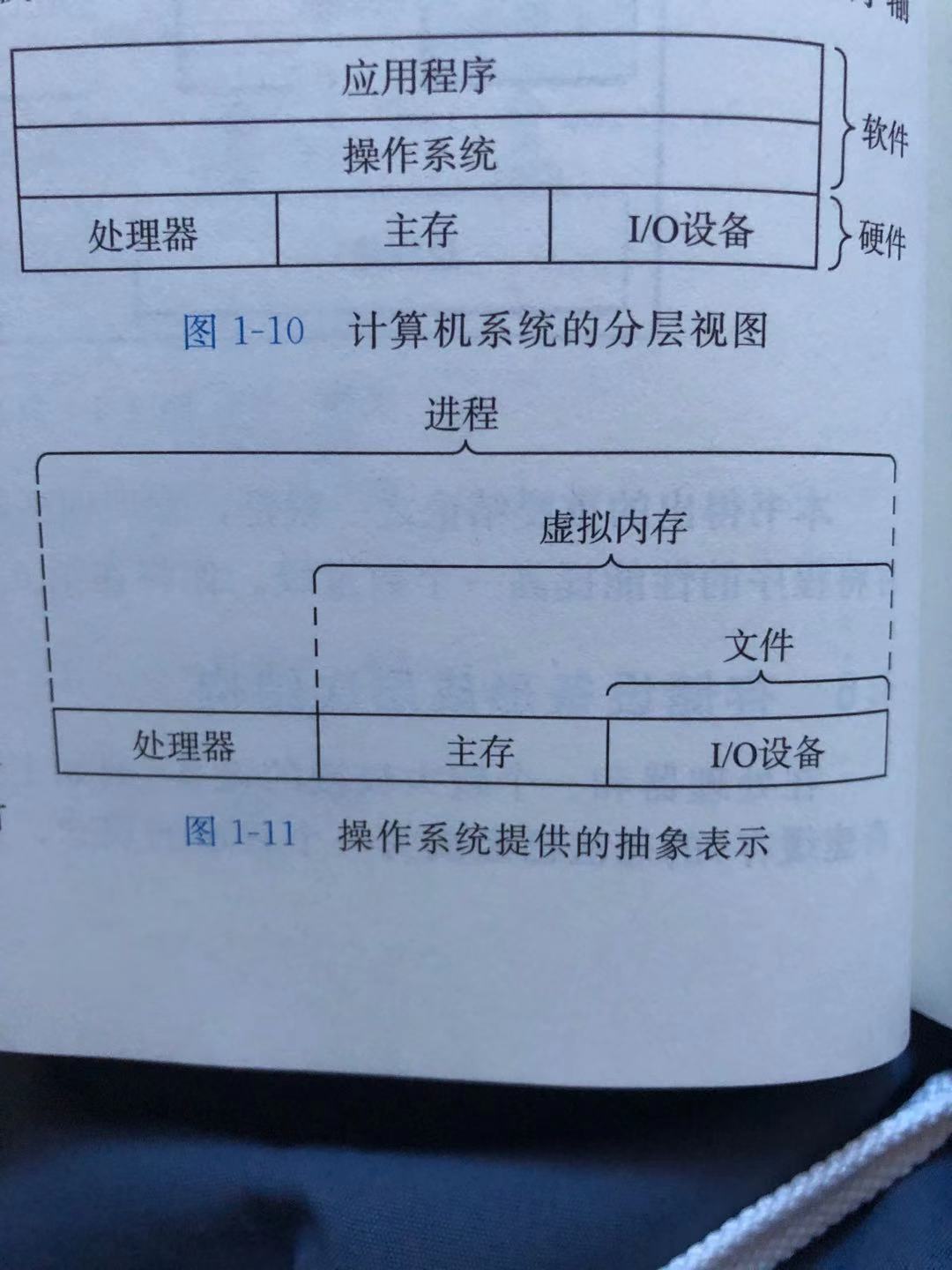
存储设备形成层次结构



主要思想是上一层存储设备作为下一层的高速缓存

操作系统管理硬件

如上面的shell与hello程序是没有直接访问键盘,显示器,磁盘和主存的,是依靠操作系统提供的服务



应用程序与硬件之间多了一层操作系统,所有应用程序对硬件的尝试操作都要通过操作系统

操作系统的两个功能

避免失控的应用程序对硬件的滥用

为应用程序提供一个简单一致的机致来控制复杂又不同的低级硬件设备

操作系统通过几个基本的抽象概念(进程,虚以内存和文件)来实现上面两个功能

如上图文件是对I/O设备的抽象,虚拟内存是对主存与磁盘I/O设备的抽象,进程则是对处理器,主存和I/O设备的抽象

进程

进程是操作系统对一个正在运行的程序的一个抽象

系统可以同时运行好多个进程

并发运行

一个进程的指令与另一个进程的指令产错运行,是通过处理器在进程间切换来实现的,实现这种 交错执行的机制称上下文切换

操作系统跟踪进程进行时的所有状态信息(即上下文,包括比如PC和寄存器和内存的值),在任何时候单处理器只能执行一个进程的代码,当系统决定要把控制权转到另一个进程时就会进行上下文切换(即保存当前进程的上下文,恢复新进程的上下文,然后将控制权传递到新进程,新进程就会从上次停止的地方开始)

如上shell与hello进程并发,开始只有shell运行(等命令行输入),当输入hello程序后,shell通过调用一个函数(系统调用)来执行请求,系统调用会将控制权转给操作系统,系统就会保存shell的上下文,创建新的hello进程和其上下文并将控制权传给hello进程,hello执行结束后,系统恢复shell的上下文并把控制权给回shell进程(继续等待下一个命令输入)

具体点进程间的转换是由操作系统内核来管理的,它是操作系统代码常驻在主存的部分,内核不是一个独立的进程,而是系统管理全部进程所用代码和数据结构的集合

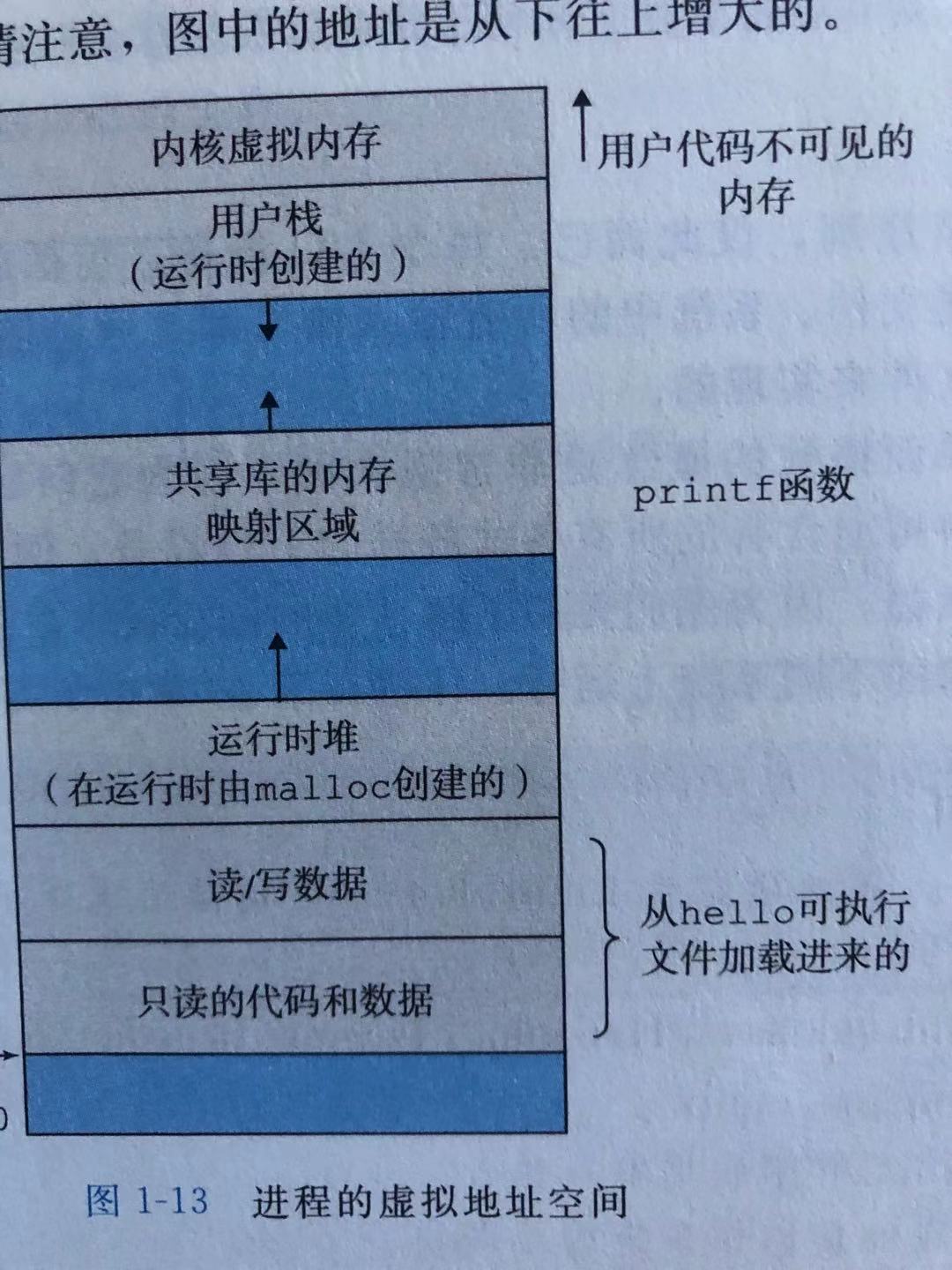
线程

现在计算机中,一个进程实际上由多个线程(执行单元)组成,线程运行在进程的上下文中,并共享同样的代码和全局数据,多线程比多进程更加容易共享数据,所以更高效

多线程是一种让程序运行更快的方法

虚拟内存

抽象根念,为每个进程提供一个假象,即每个进程都在独占地使用主存,看到的内存是一致的,称为虚拟地址空间



上图是Linux进程的虚拟地址空间,地址空间最上面的区域是保留给操作系驻的代码和数据的,这对所有进程都是一样的

每个区域都有专门的功能,逐步向上简单介绍

程序代码和数据

代码和数据是直接按照执行目标文件的内容初始化的

堆

接着是运行时堆

共享库

用来存放像C标准库函数和数学库的代码和数据的区域

栈

编译器用它来实现函数调用,调用一个函数时栈会增长,一个函数返回时栈就会收缩

内核虎拟内存

为内核保留,应用程序不能读写其内容或直接调用内核代码定义的函数,相反它们必须调用内核来执行这些操作

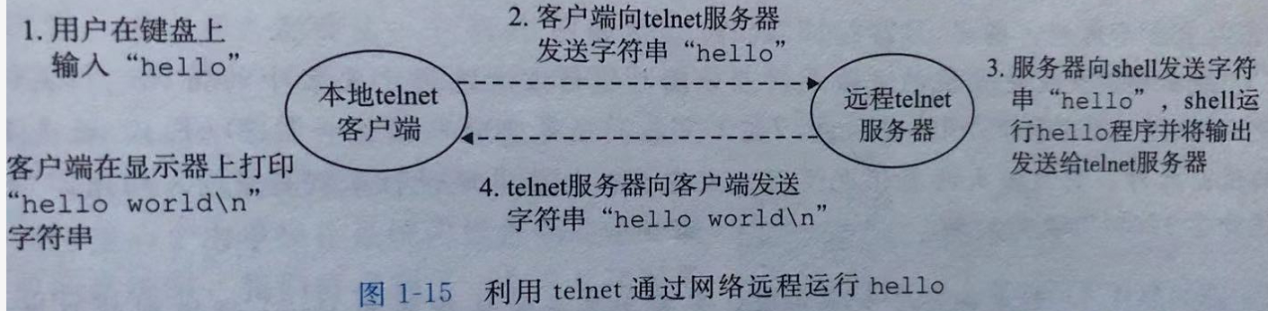
文件

就是字节序列,每个I/O设备(磁盘键盘网络等)都可以看作是文件,系统中的所有输入输出都是通过使用一个小组称为 unix I/O的系统函数调用读写文件来实现在

通过文件,应用程序无需了解磁盘技术就可以处理磁盘文件内容

系统间利用网络通信

如用本地的telnet客户端连接远程的telnet服务端并在远程执行hello程序



Amdahl定律

对系统某个部分加速时,其对系统的影响取决于该部分的重要性和加速程度

S=1/(1-a)+a/k

S加速式, a为部件执行时间占总时间, k为部件的加速因子

如a=0.6, k=3, 那S结果为1.67

并发和并行

并发

可以进行多活动的系统

并行

利用并发使一个系统运行得更快

并行可以在多个抽象层次上运用

线程级并发

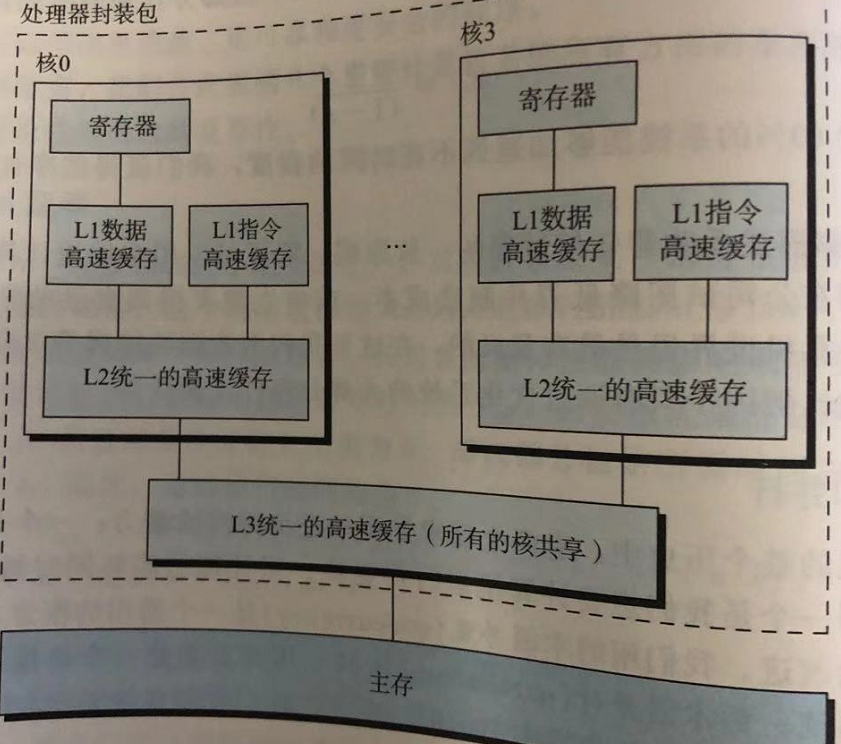
能够多个程序执行的系统就是导致了并发,使用线程甚于可以在一个进程内执行多个控制流

以前传统上的并发是在执行的进程之间快速切换来实现的,一边打开浏览器一边听歌等,都是由一个处理器来完成的(单处理器系统)

一个单操作系统内控制的多处理器组成时,就是一个多处理器系统

多核处理器是将多个CPU(核)集成在一个集成电路芯片上

如经典的多核处理器组织结构中,每个核都有自己的L1和L2高速缓存,其中L1又分为两个部分,一个存放最近取得的指令(指令高速缓存),另一个存放数据(数据高速缓存),这些核共享更高层次的高速缓存以及到主存的接口



超线程

有时称同时多线程,是一项允许一个CPU执行多个控制流的技术,常规的处理器需要大约20000个时钟周期做不同线程的切换,而超线程的处理器可以在单个周期的基础上决定要执行哪一个线程

假设一个线程须等某些数据被装载到高速缓存中,那CPU就可以续续执行另一个线程

如i7处理器可以让每个核执行两个线程,那4核的系统实际上可以并行执行8个线程

指令级并行

现代处理器可以同时执行多条指令的属性称为指令级并行

以前执行一条指令需要可能3到10或更多的时钟周期,现在的处理器可以保持每个时钟周期2到4条指令的执行效率

如果处理器可以达到1个周期执行一条指令更快,就称为超标量

单指令,多数据并行

允许一条指令产生多个可以并行执行的操作,称为单指令,多数据,SIMD并行

虚拟机是对整个计算机的抽象,包括操作系统,处理器和程序

第二章(信息的表示和处理)

信息存储

大多数计算机使用8位的块也就是字节(byte)作主最小的可寻址的内存单元,机器级程序将内存为一个非常大的字节数组,称为虚拟内存,内存的每个字节都有一个唯一的数字来标识,称为地址.所有可能地址的集合称为虚拟地址空间(可参考第一章的Linux进程的虚拟地址空间)

十六进制表示法

二进制太冗长,十进制转换太麻烦,用十六进制替换

十六进制就是0到9,A,B,C,D,E,F

在C语言中,开头0x或0X的数字常量被认为是十六进制,如0XFA1D37B,大小写可以混合

十六进制转成二进制

1 7 3 A 4 C

0001 0111 0011 1010 0100 1100

所以二进制就是 000101110011101001001100

二进制转十六进制就是反过来(拿4位转成一个十六进制,位数不够可在前面补0,小数后的在后面补0)

快速换法(只针对2的负整数幂次)

i+4j

如2的11次方=2048 11=3+4×2

可以把开头的十六进制为1(i=0) 2(i=1) 4(i=2) 8(i=3),后面跟j个0

所以2的11次方的8进制为 0x800

十进制转十六进制

类似十进制转二进制,除数为16,如

314156 = 19634\*16 + 12 (C)

19634 = 1227 \*16 + 2 (2)

1227 = 76\*16 + 11 (B)

76= 4\*16 + 12 (C)

4= 0 \*16 + 4(4)

所以314156十进制转十六进制结果为 0x4CB2C

十六进制转十进制

0x7AF

7\*16\*16 + 10\*16 + 15 = 1967