# 计算机系统漫游

## 信息就是位+上下文

1、像hello.c这样只由ASCII字符构成的文件称为文本文件

2、基本思想：系统中的所有信息--包括磁盘文件、内存中的程序、内存中存放的用户数据以及网络上传送的数据，都是由一串bit表示的，区分不同数据对象的唯一方法使我们读到这些数据对象时的上下文。例如，在不同的上下文中，一个同样的字节序列可能表示一个整数、浮点数、字符串或者机器指令

3、C语言的起源

1. C语言与Unix操作系统关系密切：C从一开始就是作为一种用于Unix系统的程序语言开发出来的，大部分的Unix内核(操作系统的核心部分)，以及所有支撑工具和函数库都是用C语言编写的
2. C语言小而简单：C语言的设计师由一个人而非一个协会掌控的，因此这是一个简洁明了、没有什么冗赘的设计
3. C语言是为实践目的的设计：C语言是用来实现Unix操作系统的，后来其他人发现能够用这门语言无障碍地编写他们想要的程序

## 程序被其他程序翻译成不同的格式

1、hello程序的生命周期是从一个高级C语言程序开始的，因为这种形式能够被人读懂。为了在系统上运行hello.c程序，每条C语句都必须被其他程序转化为一系列低级机器语言指令。然后这些指令按照一种称为可执行目标程序的格式打包，并以二进制磁盘文件的形式存放起来。目标程序也成为可执行文件

2、在Unix系统上，从源文件到目标文件的转化是由编译器驱动程序完成的

gcc -o hello hello.c

* GCC编译器驱动程序读取源程序文件hello.c，并把它翻译成一个可执行目标文件hello。这个翻译过程可分为四个阶段完成，执行这四个阶段的程序(预处理器、编译器、汇编器、链接器)一起构成了编译系统

1. 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中的第一行的#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。结果得到了另一个C程序，通常以.i作为文件扩展名
2. 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
3. 汇编阶段：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是一个二进制文件
4. 链接阶段：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，可以被加载到内存中，由系统执行

## 了解编译系统如何工作是大有益处的

1、优化程序性能

2、理解链接时出现的错误

3、避免安全漏洞

## 处理器读并解释存储在内存中的指令

1、在Unix系统上运行可执行文件，将它的文件名输入到称为shell的应用程序中

### 系统的硬件组成

1、总线：

* 贯穿整个系统的是一组电子管道，称为总线，它携带信息字节并负责在各个部件间传递
* 通常总线被设计成传送定长的字节块，也就是字(word)。字中的字节数(即字长)是一个基本的系统参数，各个系统不尽相同。现在大多数机器字长要么是4个字节(32位)，要么是8个字节(64位)

2、I/O设备：

* I/O(输入/输出)设备是系统与外部世界的联系通道
* 每个I/O设备都通过一个控制器或适配器与I/O总线相连
* 控制器与适配器之间的区别主要在于它们的封装方式
* 控制器是I/O设备本身或者系统的主印制电路板(通常称为主板)上的芯片组
* 而适配器则是一块插在主板插槽上的卡
* 无论如何，它们的功能都是在I/O总线和I/O设备之间传递信息

3、主存

* 主存是一个临时存储设备，在处理器执行程序时，用来存放程序和程序处理的数据
* 物理上来说，主存是由一组动态随机存取存储器(DRAM)芯片组成的
* 从逻辑上来说，存储器是一个线性的字节数组，每个字节都具有唯一的地址(数组索引)，这些地址是零开始的
* 一般来说，组成程序的每条机器指令都由不同数量的字节构成
* 与C程序变量相对应的数据项的大小是根据类型变化的

4、处理器

* 中央处理单元(CPU)，简称处理器，是解释(或执行)存储在主存中指令的引擎
* 处理器的核心是一个大小为一个字的存储设备(或寄存器)，称为程序计数器(PC)。在任何时候，PC都指向主存中的某条机器语言指令(即含有该条指令的地址)
* 从系统通电开始，直到系统断电，处理器一直不断地执行程序计数器指向的指令，再更新程序计数器，使其指向下一条指令。处理器看上去是按照一个非常简单的指令执行模型来操作的，这个模型是由指令集架构决定的。在这个模型中，指令按照严格的顺序执行，而执行一条指令包含执行一系列的步骤。处理器从程序计数器指向的内存处读取指令，解释指令中的位，执行该指令指示的简单操作，然后更新PC，使其指向下一条指令，而这条指令并不一定和在内存中刚刚执行的指令相邻
* 以下是CPU在指令的要求下可能执行的操作

1. 加载：从主存复制一个字节或一个字到寄存器，以覆盖寄存器原来的内容
2. 存储：从寄存器复制一个字节或一个字到主存的某个位置，以覆盖这个位置上原来的内容
3. 操作：把两个寄存器的内容复制到ALU，ALU对这两个字做算数运算，并将结果存放到一个寄存器中，以覆盖该寄存器中原来的内容
4. 跳转：从指令本身中抽取一个字，并将这个字复制到程序计数器(PC)中，以覆盖PC中原来的值

## 高速缓存至关重要

1、系统花费了大量的时间把信息从一个地方挪到另一个地方

* hello程序的机器指令最初存放在磁盘上，当程序加载时，它们被复制到主存
* 当处理器运行程序时，指令又从主存复制到处理器
* 数据串"hello,world\n"开始在磁盘上，然后被复制到主存，最后从主存上复制到显示设备

2、从程序员的角度看，这些复制就是开销，减慢了程序"真正"的工作，因此系统设计者的一个主要目标就是使这些复制操作尽可快地完成

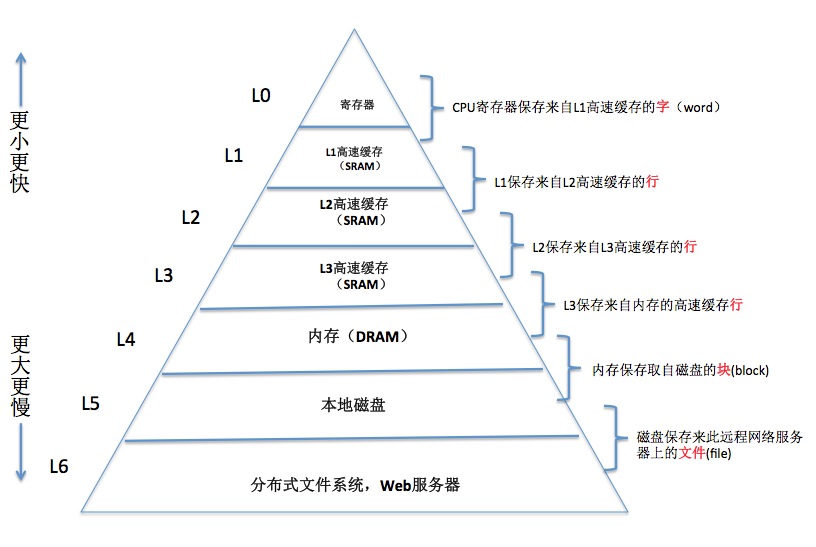
3、根据机械原理

* 较大的存储设备要比较小的存储设备运行得慢，而快速设备的造价远高于同类的低速设备。例如，一个典型系统上的磁盘驱动器可能比主存大1000倍，但是对于处理器而言，从磁盘驱动器上读取一个字的时间开销要比从主存中读取的开销大1000万倍
* 类似地，一个典型的寄存器文件只存储几百字节的信息，而主存里可存放几十亿字节，然而，处理器从寄存器文件中读取数据比从主存中读取几乎要快100倍
* 更麻烦的是，随着半导体技术的进步，这种处理器与主存之间的差距还在持续增大。加快处理器运行速度比加快主存运行速度要容易得多

4、针对这种处理器与主存之间的差异，系统设计采用了更小更快的存储设备，称为高速缓存存储器(cache memory，简称为cache或高速缓存)，作为暂时的集结区域，存放处理器近期可能会需要的信息

## 存储设备形成层次结构

1、在处理器和一个较大较慢的设备(例如主存)之间插入一个更小更快的存储设备(例如高速缓存)的想法已经成为一个普遍的观念。实际上，每个计算机系统中的存储设备都被组织成了一个存储器层次结构



2、存储器层次结构的主要思想是上一层存储器作为低一层存储器的高速缓存。因此，寄存器文件就是L1的高速缓存，L1是L2的高速缓存，L2是L3的高速缓存，L3是主存的高速缓存，而主存又是磁盘的高速缓存。在某些具有分布式文件系统的网络系统中，本地磁盘就是存储在其他系统中磁盘上的数据的高速缓存

## 操作系统管理硬件

1、shell和hello程序都没有直接访问键盘、显示器、磁盘或者主存。取而代之的是，它们依靠操作系统提供的服务。我们可以把操作系统看成是应用程序和硬件之间插入的一层软件，所有程序对硬件的操作尝试都必须通过操作系统

2、操作系统的基本功能

1. 防止硬件被失控的应用程序滥用
2. 向应用程序提供简单一致的机制来控制复杂而又通常大不相同的低级硬件设备

3、操作系统通过几个基本抽象概念来实现这两个功能

1. 进程
2. 虚拟内存
3. 文件

4、计算机系统分层图以及操作系统抽象表示如下图所示





* 文件是I/O设备的抽象表示
* 虚拟内存是对主存和磁盘I/O设备的抽象表示
* 进程则是对处理器、主存和I/O设备的抽象表示

### 进程

1、像hello这样的程序在现代操作系统上运行时，操作系统会提供一种假象，就像系统上只有这个程序在运行，程序看上去独占使用处理器、主存和I/O设备。处理器看上去像不间断地一条接一条地执行程序中的命令，即该程序代码和数据是系统内存中唯一的对象。这些假象是通过进程的概念来实现的，进程是计算机科学中最重要和最成功的的概念之一

2、进程是操作系统对一个正在运行的程序的一种抽象，在一个系统上可以同时运行多个进程，而每个进程都好像独占地使用硬件。而并发运行，则是说一个进程的指令和另一个进程的指令是交错执行的。在大多数系统中，需要运行的进程是多余可以运行它们的CPU个数的

3、传统系统在一个时刻只能执行一个程序，而先进的多核处理器同时能够执行多个程序。无论在单核还是多核系统中，一个CPU看上去都像是在并发地执行多个进程，这是通过处理器在进程间切换来实现的。操作系统实现这种交错执行的机制称为上下文切换

4、操作系统保持跟踪进程运行所需的所有状态信息。这种状态，就是上下文，包括许多信息，比如PC和寄存器文件的当前值，以及主存的内容。当操作系统决定要把控制权从当前进程转移到某个新进程时，就会进行上下文切换，即保存当前进程的上下文、恢复进程的上下文，然后将控制权传递到新进程，新进程就会从它上次停止的地方开始

5、从一个进程到另一个进程的转换时由操作系统内核(kernel)管理的，内核是操作系统代码常驻主存的部分。当应用程序需要操作系统的某些操作时，比如读文件，它就执行一条特殊的系统调用(system call)指令，将控制权传递给内核，然后内核执行被请求的操作并返回给应用程序。

6、注意，内核不是一个独立的进程，相反，它是系统管理全部进程所用代码和数据结构的集合

### 线程

1、通常我们认为一个进程只有单一的控制流，但在现代系统中，一个进程实际上可以由多个称为线程的执行单元组成，每个线程都运行在进程的上下文中，并共享同样的代码和全局数据。

2、由于网络服务器中对并行处理的需求，线程成为越来越重要的编程模型，因为多线程之间比多进程之间**更容易**共享数据(进程间也能共享数据)，也因为线程一般来说比进程更高效

3、到底什么是线程和进程呢

### 虚拟内存

1、虚拟内存是一个抽象概念，它为每个进程提供了一个假象，即每个进程都在独占地使用主存，每个进程看到的内存都是一致的，称为虚拟地址空间



* 图中地址从下往上是增大的

2、逐个介绍每个内存区域

1. 程序代码和数据

* 对所有进程来说，代码是从同一固定地址开始，紧接着是和C全局变量相对应的数据位置。代码和数据区是直接按照可执行目标文件的内容初始化的

1. 堆

* 代码和数据区后紧随着是运行时堆，代码区和数据区在进程一开始运行时就被指定了大小
* 与此不同，当调用像malloc和free这样的C标准库函数时，堆可以在运行时动态地扩展和收缩

1. 共享库

* 大约在内存地址空间中间部分是一块用来存放像C标准库和数学库这样的共享库的代码和数据的区域
* 共享库的概念非常强大，也相当难懂，第七章有详细介绍

1. 栈

* 位于用户虚拟地址空间顶部的是用户栈，编译器用它来实现函数调用
* 和堆一样，用户栈在程序执行期间可以动态地扩展和收缩
* 每次调用一个函数时，栈就会增长
* 每次从一个函数返回，栈就会收缩
* 第三章将会学习到编译器如何使用栈

1. 内核虚拟内存

* 地址空间顶部的区域是为内核保留的
* 不允许应用程序读写这个区域的内容或者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

1、文件就是字节序列，仅此而已

2、每个I/O设备，包括磁盘、键盘、显示器，甚至网络，都可以看成是文件

3、系统中的所有输入输出都是通过一小组称为Unix I/O的系统函数调用读写文件来实现的

4、文件这个简单而精致的概念是非常强大的，因为它向应用程序提供了一个统一的视图，来看待系统中可能含有的所有各式各样的I/O设备

## 系统之间利用网络通信

1、实际上，现代系统经常通过网络和其他系统连接到一起，从一个单独的系统来看，网络可视为一个I/O设备，当系统从主存复制一串字节到网络适配器时，数据流经过网络到达另一台机器，而不是比如说到达本地磁盘驱动器，相似地，系统可以读取从其他机器发来的数据，并把数据复制到自己的主存

2、客户端和服务器之间交互的类型在所有的网络应用中是非常典型的

## 重要主题

### Amdahl定律

1、想要显著加速整个系统，必须提升全系统中相当大的部分的速度

### 并发和并行

1、两个术语

1. 并发：指一个同时具有多个活动的系统
2. 并行：指的是用并发来使系统运行得更快

2、并行可以在计算机系统的多个抽象层次上运用

1. 线程级并发
2. 指令级并发
3. 单指令多数据并行

### 计算机系统中抽象的重要性

1、抽象的使用时计算机科学中最为重要的概念之一。例如，为一组函数规定一个简单的应用程序接口(API)就是一个很好地编程习惯，程序员无需了解它内部的工作便可以使用这些代码

2、在处理器里的抽象

1. 指令集架构提供了对实际处理器硬件的抽象，使用这个抽象，机器代码程序表现的就好像运行在一个一次只执行一条指令的处理器上
2. 底层的硬件远比抽象描述的要复杂精细，它并行地执行多条指令，但又总是与那个简单有序的模型保持一致，只要执行模型一样，不同的处理器实现也能执行相同的机器码，而又能提供不同的开销和性能

3、操作系统里的抽象

1. 文件：对I/O设备的抽象
2. 虚拟内存：对程序存储器的抽象
3. 进程：对一个正在运行的程序的抽象
4. 虚拟机：它提供对整个计算机的抽象

# 信息的表示和处理

1、二进制信号能够容易地被表示、存储和传输。例如可以表示为穿孔卡片上的有洞或无洞、导线上的高压电和低压电，或者顺时针或逆时针的磁场

2、二进制信号进行存储和执行计算的电子电路非常简单和可靠，制造商能够在一个单独的硅片上集成数百万甚至数亿个这样的电路

3、孤立地讲，单个位不是非常有用，当把位组合在一起，再加上某种解释，即赋予不同的可能位模式以含义，我们就能够表示任何有限集合的元素

## 信息存储

1、大多数计算机使用8位的块，或者字节，作为最小的可寻址的内存单位，而不是访问内存中单独的位

2、机器级程序将内存视为一个非常大的字节数组，称为虚拟内存

3、内存的每个字节都由一个唯一的数字来标识，称为它的地址，所有可能地址的集合就成为虚拟地址空间

### 十六进制表示法

1、十六进制采用0~9，以及A~F来表示16个可能的值

### 字数据大小

1、每台计算机都由一个字长(word size)，指明指针数据的标称大小(nominal size)。因为虚拟地址是以这样的一个字来编码的，所以字长决定的最重要的系统参数就是虚拟地址空间的最大大小。对于一个字长为w位的机器而言，虚拟地址的范围为0~2w-1，即最多访问2w个字节

2、大多数64位机器可以运行32位机器编译的程序，这是一种向后兼容

3、32位程序或64位程序，区别在于该程序是如何编译的，而不是其运行的机器类型

### 寻址和字节顺序

1、对于跨多字节的程序对象，我们必须建立两个规则：这个对象地址是什么，以及在内存中如何排列这些字节

2、在几乎所有机器上，多字节对象都被存储为连续的字节序列，对象的地址为所使用字节中最小的地址

3、排列表示一个对象的字节由两个通用的规则

* 考虑一个w位的整数，其为表示为[xw-1,xw-1,...,x1,x0]，其中xw-1是最高有效位，x0是最低有效位
* 某些机器选择在内存中按照从最低有效字节到最高有效字节的顺序存储对象，而另一些机器则按照从最高有效字节到最低有效字节的顺序存储
* 最低有效字节在最前面的方式，称为小端法(little endian)
* 最高有效字节在最前面的方式，称为大端法(big endian)

4、大多数Intel兼容机都只用小端模式

5、对于大多数应用程序员来说，其机器所使用的的字节顺序是完全不可见的，无论为哪种类型的机器所编译的程序都会得到相同的结果

6、不过有时候，字节顺序会造成问题

* 首先是在不同类型的机器之间通过网络传送二进制数据时，一个常见的问题是当小端法机器产生的数据被发送到大端法机器或者反过来说时，接受程序会发现，字里的字成了反序的，为了避免这种问题，网络应用程序的代码编写必须遵守已建立的关于字节顺序的规则，以确保发送方机器将它的内部表示转换成网络标准，而接收方机器则将网络标准转换为它的内部表示
* 当阅读表示整数数据的字节序列时字节顺序也很重要