# 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

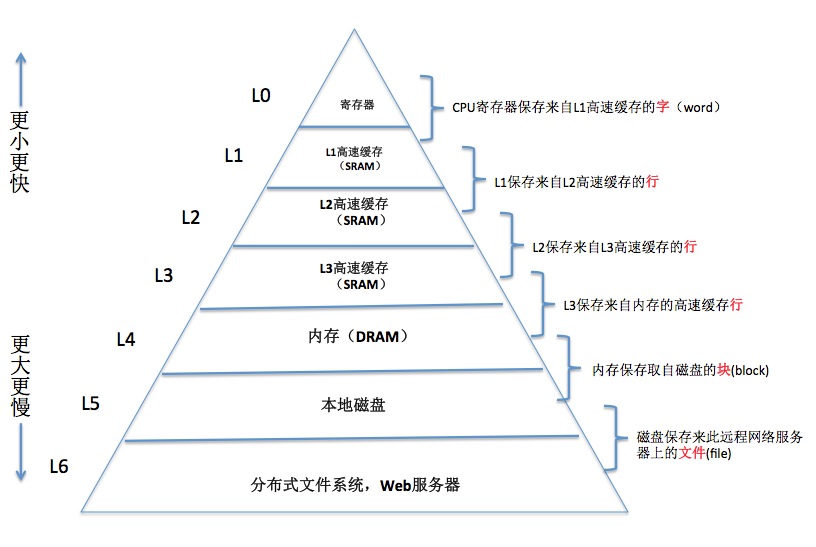
3、根据机械原理

* 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