读书内容：《程序员的自我修养》第一章和第六章

内容总结：

第一章：

1. 对Hello World程序：

#include<stdio.h>

Int main()

{

Printf(“Hello World\n”);

Return 0;

}

提出思考：程序为什么要被编译器编译了之后才可以运行？

编译器在把C语言程序转换成可以执行的机器码的过程中做了什么，怎么做的？

最后编译出来的可执行文件里面是什么？除了机器码还有什么？它们怎么存放的，怎么组织的？

#include<stdio.h>是什么意思？把stdio.h包含进来意味着什么？C语言库又是什么？它怎么实现的？

不同的编译器（Microsoft VC、GCC）和不同的硬件平台（x86、SPARC、MIPS、ARM），以及不同的操作系统（Windows、Linux、UNIX、Solaris），最终编译出来的结果一样吗？为什么？

Hello World程序是怎么运行起来的？操作系统是怎么装载它的？它从哪开始执行，到哪结束？main函数之前发生了什么？main函数结束以后又发生了什么？

如果没有操作系统，Hello World可以运行吗？如果要在一台没有操作系统的机器上运行Hello World需要什么？应该怎么实现？

Printf是怎么实现的？它为什么可以有不定数量的参数？为什么它能够在终端上输出字符串？

Hello World程序在运行时，它在内存中是什么样子的？

本书将从最基本的编译、静态链接到操作系统如何装载程序、动态链接及运行库和标准库的实现，甚至一些操作系统的机制，深入浅出地剥开这些问题，理清程序运行机制的脉络。

1. 本书将计算机的范围限定在采用兼容x86指令集和32位CPU的个人计算机。计算机有三个关键部件：中央处理器CPU、内存和I/O控制芯片。

对称多处理器（SMP）简单讲就是每个CPU在系统中所处的地位和所发挥的功能都是一样的，是相互对称的。（多CPU计算机的一种常用形式）

多核处理器的基本想法：将多个处理器“合并在一起打包出售”，这些“被打包”的处理器之间共享比较昂贵的缓存部件，只保留多个核心。（SMP的简化版）

1. 系统软件可以分为两块，一块是平台性的，比如操作系统内核、驱动程序、运行库和数以千计的系统工具；另外一块的用于程序开发的，比如编译器、汇编器、链接器等开发工具和开发库。本书着重介绍链接器和库（包括运行库和开发库）的内容。

每个层次都要互相通信，通信协议被称为接口，接口下面的那层是接口的提供者，由它定义接口；接口的上面那层是接口的使用者，它使用该接口来实现所需要的功能。

除了硬件和应用程序，其他都是中间层，每个中间层都是对它下面那层的包装和扩展。中间层使得应用程序和硬件之间保持相对独立。

在软件体系中位于最上层的是应用程序。从整个层次结构上看，开发工具与应用程序是属于同一层次的，因为它们都使用一个接口——操作系统应用程序编程接口。应用程序接口的提供者是运行库，什么样的运行库提供什么样的API。

运行库使用操作系统提供的系统调用接口，系统调用接口在实际中常以软件中断的方式提供。

操作系统内核层对于硬件层来说是硬件接口的使用者，而硬件是接口的定义者，硬件的接口定义决定了操作系统内核，即驱动程序如何操作硬件，如何与硬件进行通信。这种接口被叫做硬件规格。

1. 操作系统的一个功能是提供抽象的接口，另外一个主要功能是管理硬件资源。

多道程序：当某个程序暂时不用CPU时，监控程序就把另外的正在等待CPU资源的程序启动，使得CPU能够得到充分的利用。

分时系统：每个程序运行一段时间以后都主动让出CPU给其他程序，使得一段时间内每个程序都有机会运行一段时间。

多任务系统：操作系统接管了所有的硬件资源，并且本身运行在一个受硬件保护的级别。所有的应用程序都以进程的方式运行在比操作系统权限更低的级别，每个进程有自己独立的地址空间，使得进程之间的地址空间相互隔离。CPU的分配方式为抢占式，即操作系统可以强制将CPU资源分配给它认为目前最需要的进程。

1. 虚拟地址空间是指虚拟的、人们想象出来的地址空间，每个进程都有自己独立的虚拟空间，而且每个进程只能访问自己的地址空间，这样就有效实现进程的隔离。

分段：把一段与程序所需要的内存空间大小的虚拟空间映射到某个地址空间。

分页：把地址空间人为地等分成固定大小的页，每一页的大小由硬件决定，或硬件支持多种大小的页，由操作系统选择决定页的大小。虚拟空间的页称为虚拟页（VP），物理内存的页叫做物理页（PP），磁盘中的页叫磁盘页（DP）。几乎所有的硬件都采用一个叫MMU的部件来进行页映射。

1. 线程：程序执行流的最小单元。一个标准的线程由线程ID、当前指令指针（PC）、寄存器集合和堆栈组成。一个进程由一个到多个线程组成，各个线程之间共享程序的内存空间（包括代码段、数据段、堆等）及一些进程级的资源（如打开文件和信号）。

线程的私有存储空间：栈、线程局部存储（TLS）、寄存器。

线程调度的三种状态：运行，就绪，等待。

时间片：处于运行中线程拥有的一段可以执行的时间。

调度方法有优先级调度和轮转法。

频繁等待的线程称为IO密集型线程，很少等待的线程称为CPU密集型线程。IO密集型线程总是比CPU密集型线程容易得到优先级的提升。在优先级调度下有饿死现象——始终有较高优先级的线程要执行，那么低优先级的线程就无法执行。

单指令操作称为原子的。

同步：在一个线程访问数据未结束的时候，其他线程不得对同一个数据进行访问。常用方法为锁。每一个线程在访问数据或资源之前首先试图获取锁，并在访问结束之后释放锁。其余线程会等待至锁重新可用。

二元信号量是最简单的一种锁，只有占用与非占用两种状态。还有多元信号量，简称信号量。

互斥量与二元信号量类似，同时只允许一个线程访问，不同的是，信号量被一个线程获取后可以由另一个线程释放，互斥量要求哪个线程获取就由哪个线程释放。

临界区更加严格。把临界区的锁的获取称为进入临界区，把锁的释放称为离开临界区。临界区和互斥量与信号量的区别在于，互斥量和信号量在系统的任何进程都是可见的，即一个进程创建了互斥量或者信号量，另一个进程试图去获取该锁是合法的，但是临界区的作用范围仅限于本进程，其他进程无法获取该锁。除此之外，临界区和互斥量性质相同。

读写锁有共享的和独占的两种获取方式。

条件变量作用类似一个栅栏。线程可以等待和唤醒条件变量。条件变量可以让许多线程一起等待，当被唤醒时，所有线程一起恢复执行。

一个函数可重入表示这个函数没有执行完成，由于外部因素或内部调用，又一次进入该函数执行。

三种线程模型：一对一模型，多对一模型，多对多模型。

第六章

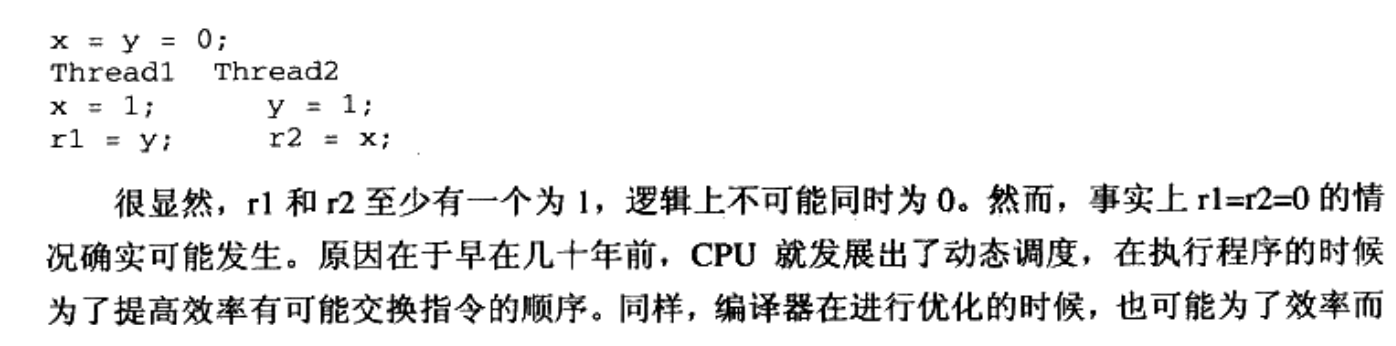
1. 程序是个静态概念，它是预先编译好的指令和数据集合的一个文件，进程是一个动态的概念，它是程序运行时的一个过程。
2. 页错误：当CPU开始执行某地址的指令时，发现页面是空页面。
3. Segment：包含一个或多个属性类似的Section。在将目标文件链接成可执行文件的时候，链接器会尽量把相同权限属性的段分配在同一空间。使用readelf可以查看ELF的Segment。描述Section属性的结构叫做段表，描述Segment的结构叫做程序头，它描述了ELF文件该如何被操作系统映射到进程的虚拟空间。
4. Linux系统如何装载ELF文件并执行它：

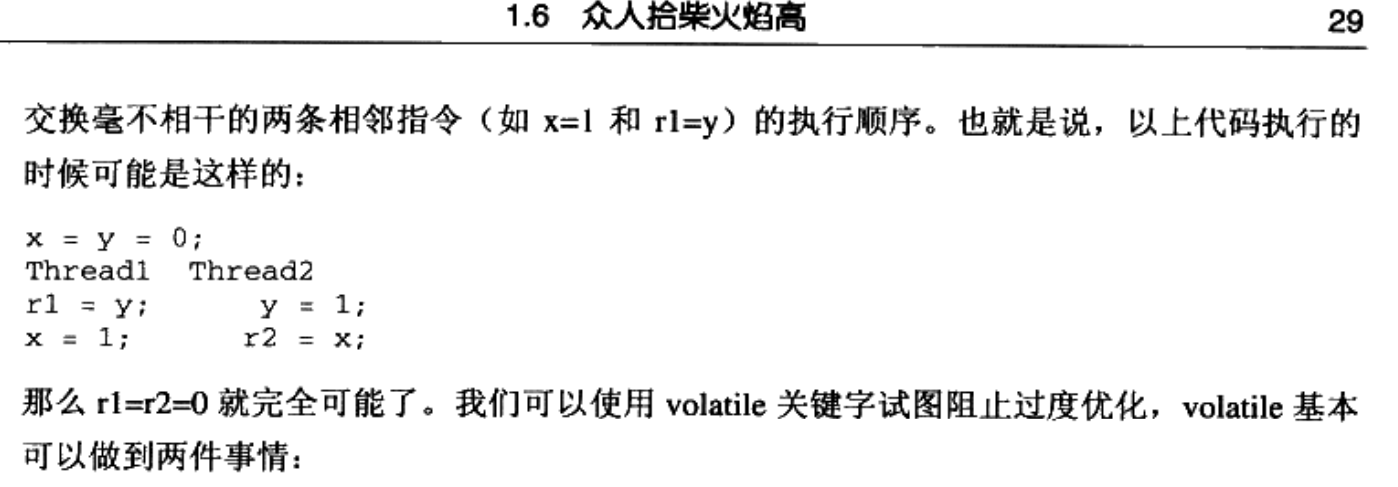
首先在用户层面，bash进程会调用fork（）系统调用创建一个新的进程，然后新的进程调用execve（）系统调用执行指定的ELF文件，原先的bash进程继续返回等待刚才启动的新进程结束，然后继续等待用户输入命令。在进入execve（）系统调用之后，Linux内核就开始进行真正的装载工作。在内核中，execve（）系统调用相应的入口是sys\_execve（），它进行一些参数的检查复制之后，调用do\_execve（）。do\_execve（）会首先查找被执行的文件，如果找到文件，则读取文件的前128个字节，目的是判断文件的格式。每种可执行文件的格式的开头几个字节都是特殊的，尤其是开头4个字节，常被称为魔数。接着调用search\_binary\_handle（）去搜索和匹配合适的可执行文件装载处理过程。当load\_elf\_binary（）执行完毕，返回至do\_execve（）再返回至sys\_execve（）时，系统调用的返回地址已经改成了被装载的ELF程序的入口地址。所以当sys\_execve（）系统调用从内核态返回到用户态时，EIP寄存器直接跳转到了ELF程序的入口地址，新的程序开始执行，ELF可执行文件装载完成。

相对虚拟地址（RVA，PE中的常见术语）：相对于PE文件的装载基地址的一个偏移地址。装载PE可执行文件的过程：先读取文件的第一个页。然后检查进程地址空间中，目标地址是否可用，如果不可用就另选一个装载地址。使用段表中提供的信息，将PE文件中所有的段一一映射到地址空间中相应的位置。如果装载地址不是目标地址，则进行Rebasing。装载所有PE文件所需要的DLL文件。对PE文件中的所有导入符号进行解析。根据PE头中指定的参数，建立初始化栈和堆。建立主线并且启动进程。

PE文件中，与装载相关的主要信息都包含在PE扩展头和段表。

疑问：





为什么说交换指令的顺序后可能出现r1=r2=0？