课外作业

1.计算机启动的过程

第一阶段：BIOS

BIOS 是直接与硬件打交道的底层代码，它为操作系统提供了控制硬件设备的基本功能。BIOS 一般被存放在 ROM(只读存储芯片)之中，即使在关机或掉电以后，这些代码也不会消失。当我们按下电源开关时，CPU 马上就从地址 FFFF0H 处开始执行指令，从前面的介绍可知，这个地址实际上在系统 BIOS 的地址范围内，无论是 Award BIOS 还是 AMI BIOS，放在这里的只是一条跳转指令，跳到系统 BIOS 中真正的启动代码处。

1.1 硬件自检

BIOS 中主要存放的程序包括：自诊断程序（通过读取 CMOS RAM 中的内容识别硬件配置，并对其进行自检和初始化）、CMOS 设置程序（引导过程中，通过特殊热键启动，进行设置后，存入 CMOS RAM 中）、系统自动装载程序（在系统自检成功后，将磁盘相对 0 道 0 扇区上的引导程序装入内存使其运行）和主要 I/O 驱动程序和中断服务（BIOS 和硬件直接打交道，需要加载 I/O 驱动程序）。BIOS 程序首先检查，计算机硬件能否满足运行的基本条件，这叫做"硬件自检"（Power-On Self-Test），缩写为 POST。如果硬件出现问题，主板会发出不同含义的蜂鸣，启动中止。如果没有问题，屏幕就会显示出 CPU、内存、硬盘等信息。

1.2 启动顺序

硬件自检完成后，BIOS 把控制权转交给下一阶段的启动程序。这时，BIOS 需要知道，"下一阶段的启动程序"具体存放在哪一个设备。也就是说，BIOS 需要有一个外部储存设备的排序，排在前面的设备就是优先转交控制权的设备。这种排序叫做"启动顺序"（Boot Sequence）。

第二阶段：主引导记录

BIOS 按照"启动顺序"，把控制权转交给排在第一位的储存设备。即根据用户指定的引导顺序从软盘、硬盘或是可移动设备中读取启动设备的 MBR，并放入指定的位置（0x7c000）内存中。

这时，计算机读取该设备的第一个扇区，也就是读取最前面的 512 个字节。如果这 512 个字节的最后两个字节是 0x55 和 0xAA，表明这个设备可以用于启动；如果不是，表明设备不能用于启动，控制权于是被转交给"启动顺序"中的下一个设备。

这最前面的 512 个字节，就叫做"主引导记录"（Master boot record，缩写为 MBR）。"主引导记录"只有 512 个字节，放不了太多东西。它的主要作用是，告诉计算机到硬盘的哪一个位置去找操作系统。

主引导记录由三个部分组成：

（1） 第 1-446 字节：调用操作系统的机器码。

（2） 第 447-510 字节：分区表（Partition table）。

（3） 第 511-512 字节：主引导记录签名（0x55 和 0xAA）。

其中，第二部分"分区表"的作用，是将硬盘分成若干个区。硬盘分区有很多好处。考虑到每个区可以安装不同的操作系统，"主引导记录"因此必须知道将控制权转交给哪个区。

第三阶段：硬盘启动

这时，计算机的控制权就要转交给硬盘的某个分区了，这里又分成三种情况。

情况 A：卷引导记录

硬盘的四个主分区里面，只有一个是激活的。计算机会读取激活分区的第一个扇区，叫做"卷引导记录"（Volume boot record，缩写为 VBR）。"卷引导记录"的主要作用是，告

诉计算机，操作系统在这个分区里的位置。然后，计算机就会加载操作系统了。

情况 B：扩展分区和逻辑分区

随着硬盘越来越大，四个主分区已经不够了，需要更多的分区。但是，分区表只有四项，因此规定有且仅有一个区可以被定义成”扩展分区”（Extended partition）。

所谓”扩展分区”，就是指这个区里面又分成多个区。这种分区里面的分区，就叫做”逻辑分区”（logical partition）。

计算机先读取扩展分区的第一个扇区，叫做”扩展引导记录”（Extended boot record，缩写为 EBR）。它里面也包含一张 64 字节的分区表，但是最多只有两项（也就是两个逻辑分区）。

计算机接着读取第二个逻辑分区的第一个扇区，再从里面的分区表中找到第三个逻辑分区的位置，以此类推，直到某个逻辑分区的分区表只包含它自身为止（即只有一个分区项）。因此，扩展分区可以包含无数个逻辑分区。

但是，似乎很少通过这种方式启动操作系统。如果操作系统确实安装在扩展分区，一般采用下一种方式启动。

情况 C：启动管理器

在这种情况下，计算机读取”主引导记录”前面 446 字节的机器码之后，不再把控制权转交给某一个分区，而是运行事先安装的”启动管理器”（boot loader），由用户选择启动哪一个操作系统。

第四阶段：操作系统

控制权转交给操作系统后，操作系统的内核首先被载入内存。

以 Linux 系统为例，先载入/boot 目录下面的 kernel。内核加载成功后，第一个运行的程序是/sbin/init。它根据配置文件（Debian 系统是/etc/initab）产生 init 进程。这是 Linux 启动后的第一个进程，pid 进程编号为 1，其他进程都是它的后代。

然后，init 线程加载系统的各个模块，比如窗口程序和网络程序，直至执行/bin/login 程序，跳出登录界面，等待用户输入用户名和密码。

至此，全部启动过程完成。

2.局部性原理和程序设计及数据结构的关系

程序的局部性原理是指程序在执行时呈现出局部性规律，即在一段时间内，整个程序的执行仅限于程序中的某一部分。相应地，执行所访问的存储空间也局限于某个内存区域。局部性原理又表现为：时间局部性和空间局部性。时间局部性是指如果程序中的某条指令一旦执行，则不久之后该指令可能再次被执行；如果某数据被访问，则不久之后该数据可能再次被访问。空间局部性是指一旦程序访问了某个存储单元，则不久之后。其附近的存储单元也将被访问。

利用局部性来提高程序效率:二维数组是以行优先来存放的，按照先行后列的顺序遍历二维数组会连续读取内存空间，第一次 cache 的内容后面都会再用到，这就提高了程序的效率，反之 cache 的内容被利用地较少，效率变低。其它的连续存放在内存空间中的数据结构也是一样。

另外，还需要将注意力集中在内部循环上，大部分计算和存储器访问都发生在这里；一旦从存储器中读入了一个数据对象，就尽可能多的使用它，从而使得程序的时间局部性最大，关联性强的代码放到一起，变量定义在使用的地方。

3.什么是线程安全(thread-safe)的代码，及相关内容

线程安全就是多线程访问时，采用了加锁机制，当一个线程访问该类的某个数据时，进

行保护，其他线程不能进行访问直到该线程读取完，其他线程才可使用。不会出现数据不一致或者数据污染。而从代码上说，局部变量局部使用是安全的，全局原生变量多线程读写是不安全的，函数静态变量多线程读写也是不安全的，对于安全性：局部变量>成员变量>全局变量，任何时刻两个线程同时操作一个共享变量，当其中一个为写操作时，这两个线程必须使用原子操作。

4.使用二元信号量来实现普通信号

```
semaphore S=1, B=0;
int counter=0;
wait(S,B,counter)
{
   wait(S);
   counter--;
   if(counter < 0)
   {
       signal(S);
       wait(B);
   }
   signal(S);
}
signal(S,B,counter)
{
   wait(S);
   counter++;
   if(counter <= 0)
       signal(B);
   signal(S);
}
```