西南林业大学

本科毕业（设计）论文

（二〇一八届）

题　　目： 操作系统探索

分院系部： 大数据与智能工程学院

专　　业： 计算机科学与技术专业

姓　　名： 尹志成

导师姓名： 王晓林

导师职称： 讲师

二〇一八年六月

操作系统探索

尹志成

（西南林业大学 大数据与智能工程学院, 云南昆明 650224）

摘　要: 操作系统最初的诞生是为了搭配进行简单繁重的数字运算机, 但随着时代的演进, 计算机不仅作为处理各种运算的机器, 其附加价值也越来越被人们看重, 跟随着计算机的发展, 操作系统的使命也在一代代的改变, (约两百字)

关键词: 操作系统

Operating system exploration

Zach Yin

School of Big Data and Intelligence Engineering
Southwest Forestry University
Kunming 650224, Yunnan, China

Abstract: 英文摘要

Key words: Operate System

目　录

插图目录

表格目录

1 绪论

在数字时代, 操作系统的重要性不言而喻, 它作为计算机软硬件之间的桥梁, 存在于日常生活的每一个角落, 而研究一个只有庞大的公司聚集成百上千的高级工程师才能完成的操作系统对于学生而言是一个几乎不可能完成的挑战, 但是克服难关是锻炼技术的必经之路[1], 所以研究并完成一个基本满足日常功能需求的操作系统作为此次的目标, 并以此为跳板对操作系统进行更深一步的探究。

2　思路

此次的思路由四部分组成:

1、操作系统探究

2、空白操作系统的启动

3、编写操作系统内核

4、实现对外兼容及安全防护

2.1　操作系统探究

从历史上计算机操作系统的发展联系到人们的日常生活, 寻求符合操作系统发展且适应用户使用的特征要素。

2.2　空白操作系统的启动

利用汇编语言及操作系统相关知识探究操作系统如何从电气设备到软件代码的衔接。

2.3　完善操作系统内核

从内存管理, 输入输出, 多进程, 分时四个模块完成操作系统的内核设计及实现。

2.4　实现对外接口及安全防护

从接口设计及安全防护的角度完善操作系统。

3　操作系统探索

3.1　操作系统的诞生

3.1.1　第一代:真空管

操作系统最初出现的场景是一个工程师小组设计、建造一台机器, 之后使用机器语言编写程序并通过将上千根电缆接到插线板上连接成电路, 控制机器的基本功能, 进而操作机器运算诸如制作正弦、余弦、对数表或计算炮弹弹道的简单数学运算。

这里的人工拔插电缆就充当着操作系统的角色——根据程序直接操作硬件使其运算得出结果。

3.1.2　第二代:晶体管

在晶体管发明后, 计算机可靠程度大大增加, 计算机开始被一些公司、政府部门或大学使用。

改进后出现了操作系统的载体, 卡片和较后期磁带打孔纸带。

由于打孔纸带是分次读入, 一次只能读入一个的作业, 出现了批处理系统如图 3-1 **??**:

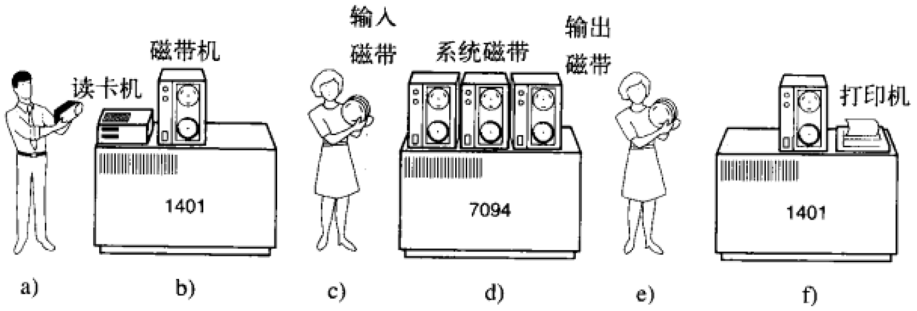

图 3-1　批处理系统

a. 程序员将打孔纸带拿到 1401 机[1]

b. 1401 机将批处理作业读到磁带上

[1]IBM 1401:数据处理计算机[2]

c. 操作员将输入磁带送至 7094 机[2]

d. 7094 机进行计算

e. 操作员将输出磁带送到 1401 机

f. 1401 机打印输出

在这里, 操作系统的工作已经由完全的人工转换到一部分人工操作交由机器完成, 工作效率较之前大大提高, 并且, 由于加入了磁带, 计算机完成的工作也将及时得到保存。

操作系统的工作已经开始有一定的流程化了。

3.1.3 第三代:集成电路

采用集成电路的第三代计算机较分立晶体管的第二代计算机在性能/价格比上有了很大的提高,

第三代操作系统也加入了多道程序设计和分时系统。

多道程序设计主要目的是解决 CPU 因等待磁带或其他 I/O 操作而暂停工作, 多道程序设计可以使 CPU 在程序 a 的 I/O 操作时运行程序 b。

分时系统解决的主要问题是多用户使用分离的终端, 却操作同一台计算机。

3.1.4 第四代:个人计算机

大规模集成电路进一步减小了计算机的大小

3.1.5 第五代:移动计算机

3.2 操作系统的规范化

[2]IBM 7094: 专为大型科学计算而设计, 具有出色的性价比和扩展的计算能力[3]

4　空白操作系统的启动

4.1　操作系统启动流程

按下电源键后计算机开始启动, 启动过程分为 3 个阶段[4]:

BIOS -> MBR -> 操作系统

1. 在 BIOS 完成 POST（硬件自检, Power-On Self Test）并根据启动顺序（Boot Sequence）来选择启动设备。本系统是从 U 盘启动。
2. 计算机读取该设备的 MBR（Master boot record, 位于第一个扇区, 即最前面的 512 个字节, 见图 4-1）, 并运行其中的启动程序 IPL（Initial Program Loader）, 将 ZOS 加载入内存。本部分的实现代码, 参见附录程序 B.1;
3. 控制权转交给操作系统后, Kernel 开始运行, 操作系统启动完成。

4.2　制作 MBR

MBR 的主要部分是 Bootstrap code area, 即前 446 个字节。负责指出操作系统的位置, 主分区第一个扇区的物理位置（柱面、磁头、扇区号等等）, 参见附录程序B-1。

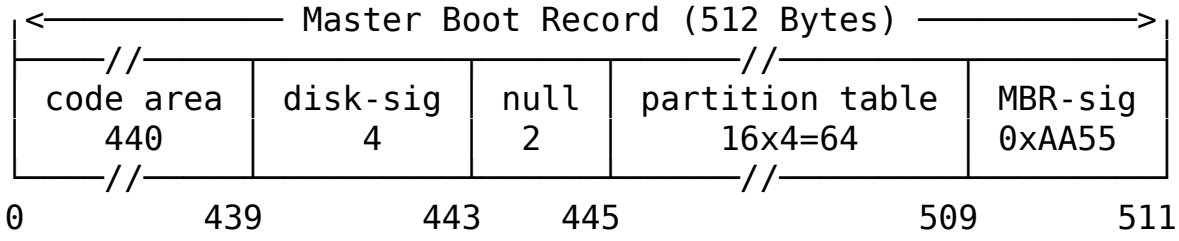

图 4-1　MBR

一个扇区大小为 512 字节, MBR 位于 C0-H0-S1（柱面 0, 磁头 0, 扇区 1）[5] 从下一个扇区 (C0-H0-S2, 柱面 0, 磁头 0, 扇区 2) 开始加载。

```
43  MOV    CH,0    ; 柱面 0
44  MOV    DH,0    ; 磁头 0
45  MOV    CL,2    ; 扇区 2
```

程序 4-1　初始化读取柱面、磁头和扇区的起点

完成定位后开始将磁盘数据读入内存, 策略是

1. 磁头 0, 柱面 0, 读取 1-18 扇区 (C0-H0-S2)-(C0,H0,S18)
2. 磁头 1, 柱面 0, 读取 1-18 扇区 (C0-H1-S1)-(C0-H1-S18)
3. 磁头 0, 柱面 1, 读取 1-18 扇区 (C1-H0-S1)-(C1-H0-S18)
4. ...
5. 磁头 1, 柱面 78, 读取 1-18 扇区 (C78-H1-S1)-(C78-H1-S18)
6. 磁头 0, 柱面 79, 读取 1-18 扇区 (C79-H0-S1)-(C79-H0-S18)
7. 磁头 1, 柱面 79, 读取 1-18 扇区 (C79-H1-S1)-(C79-H1-S18)

按以上策略将磁盘内容读入内存, 核心代码如程序4-2

```
138 ADD    CL,AL     ; 将 CL 加上 AL
139 CMP    CL,18     ; 将 CL 与 18 比较
140 JBE    readfast  ; CL <= 18 则跳转至 readfast
141 MOV    CL,1
142 ADD    DH,1
143 CMP    DH,2
144 JB    readfast   ; DH < 2 则跳转至 readfast
145 MOV    DH,0
146 ADD    CH,1
147 JMP    readfast
```

程序 4-2　读取磁盘数据到内存

readfast 位于代码第 76 行, JMP readfast 在此代表循环执行。

4.3 制作空白操作系统

为测试操作系统是否成功被 MBR 启动, 设计将操作系统设置为启动后待机。

```
fin:
HLT
JMP fin
```

按下电源键, 经过启动步骤系统循环执行 HLT[1]使得操作系统计算机始终处于待机状态, 启动成功。

[1]HLT: 让 CPU 停止动作并进入待机状态

5　编写操作系统内核

从内存管理, 输入输出, 多进程, 分时四个模块丰富操作系统的内容

5.1　内存管理

内存 (RAM) 是计算机中不可缺少的重要硬件, 所有程序的运行都是在内存中进行的, 而 CPU 访问硬盘数据也必须先经过内存交换才得以实现, 内存在加速 CPU 访问硬盘居功至伟。由内存的重要性可知内存管理在操作系统中也非常重要。

内存的结构通常用地址和空间表示, 在计算机运行过程中, 内存的分配是随机的, 导致内存的释放也是相对无序的, 这样导致了很多的碎片化问题, 也就是随计算机运行时间变长, 内存中到处遍布小块零散空闲空间, 虽然零散空间总数很大, 但很难满足新进程序的内存需求, 于是内存管理显得十分必要。

内存管理设计的主要目的是快速并且高效的分配内存空间, 并在适当的时间释放并回收内存空间。根据内存管理的设计目的, 内存管理的数据结构如下:

```
137 struct FREEINFO {  /* 剩余容量信息 */
138   unsigned int addr, size;
139 };
140 struct MEMMAN {    /* 内存管理 */
141   int frees, maxfrees, lostsize, losts;
142   struct FREEINFO free[MEMMAN_FREES];
143 };
```

frees: 空闲空间数量

maxfrees: 用于观察可用状况: frees 的最大值

lostsize: 释放失败的内存的大小总和

losts: 释放失败次数

经过内存初始化和释放所有内存空间后, 内存管理正常运行。

5.1.1　内存分配

内存的分配方式与之后的内存释放息息相关, 好的分配方式会使得效率大大提高。根据内存的大小来划分内存如何使用, 预计使用 32KB 用于内存分配的管理空间, 则共有 4000 组左右的内存用于分配给各个程序使用。

每一组内存经过初始化都拥有自己的数据结构, 即每一组空闲内存的地址和大小都被记录到空闲内存表 free。

组号	地址
1	0x00003000
2	0x00004000
3	0x00005000

表 5-1　表格示例

一旦系统接收到程序申请内存的请求(需求的内存大小), 就开始在内存中寻找足够大的内存完成这次申请, 并返回可供使用的空闲内存的地址。完成申请后系统需要重新整理空闲内存表 free, 将最大可用内存组数减一, 将返回给程序空闲空间大小根据程序需求进行调整, 并对剩余的可用内存表进行整理。

```
68  /* 找到了足够大的内存 */
69  a = man->free[i].addr;
70  man->free[i].addr += size;
71  man->free[i].size -= size;
72  if (man->free[i].size == 0) {
73    /* 如果 free[i] 变成了 0,就减掉一条可用信息 */
74    man->frees--;
75    for (; i < man->frees; i++) {
76      man->free[i] = man->free[i + 1]; /* 代入结构体 */
77    }
78  }
```

5.1.2 内存释放

为保证磁盘空闲空间尽可能少的碎片化, 内存释放首先考虑的是与附近空闲空间进行合并。

具体分为三种情况:

前端空闲: 释放内存的相连前端是空闲内存或释放内存相连两端都是空闲内存

后端可用: 释放内存的相连后端是空闲空间

前端后端均不可用: 挪动空闲空间以合并

已知:待释放的空间的地址和空间大小

根据空闲内存表 free 的编号查找地址大于待释放空间的空闲内存, 并根据得到的空闲空间编号及大小区分此时的待释放内存应当采取何种方式释放。

```
91 for (i = 0; i < man->frees; i++) {
92   if (man->free[i].addr > addr) {
93     break;
94   }
95 }
```

前端空闲:

当待释放的空间前方有空闲空间时, 将 free[i-1] 的大小加上释放空间的大小

内存释放前后情况如图5-1和图5-2所示:

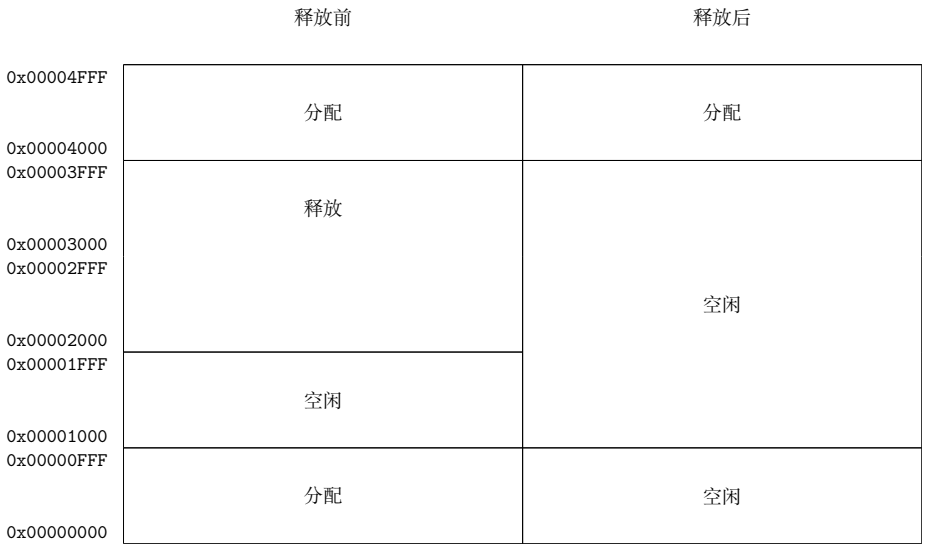

图 5-1　前端空闲

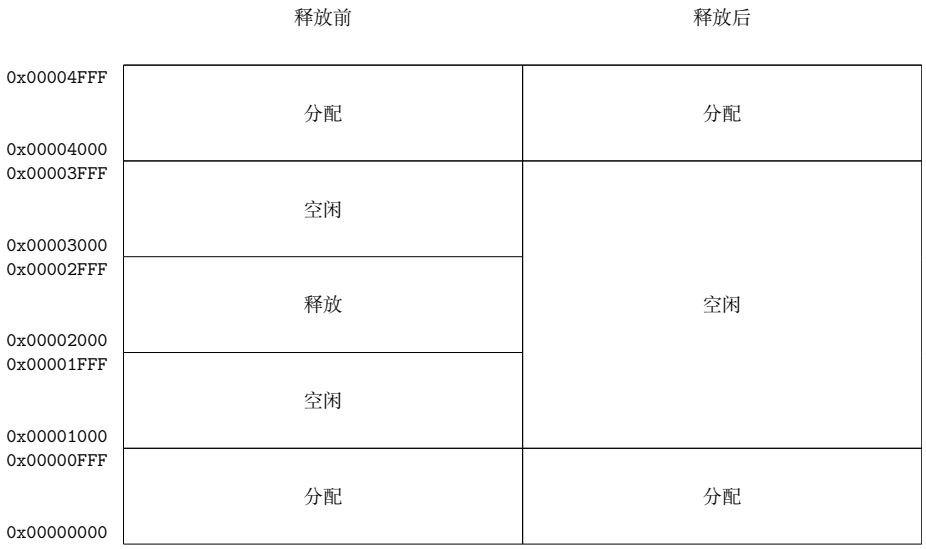

图 5-2　前端可用, 且后端空闲

实现如下：

```
98  /* 前面有可用内存 */
99  if (man->free[i - 1].addr + man->free[i - 1].size == addr) {
100   /* 可以与前面的可用内存归纳到一起 */
101   man->free[i - 1].size += size;
102   if (i < man->frees) {
103     /* 后面也有 */
104     if (addr + size == man->free[i].addr) {
105       /* 也可以与后面的可用内存归纳到一起 */
106       man->free[i - 1].size += man->free[i].size;
107       /* man->free[i] 删除 */
108       /* free[i] 变成 0 后归纳到前面去 */
109       man->frees--;
110       for (; i < man->frees; i++) {
111         man->free[i] = man->free[i + 1]; /* 结构体赋值 */
112       }
113     }
114   }
115   return 0; /* 成功完成 */
116 }
```

后端空闲:

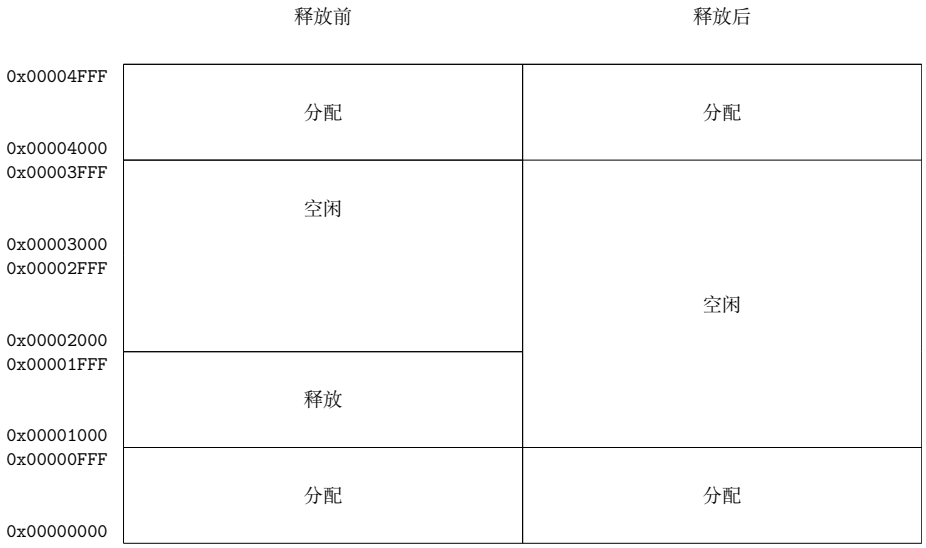

图 5-3 后端空闲

实现如下:

```
118 /* 不能与前面的可用空间归纳到一起 */
119 if (i < man->frees) {
120   /* 后面还有 */
121   if (addr + size == man->free[i].addr) {
122     /* 可以与后面的内容归纳到一起 */
123     man->free[i].addr = addr;
124     man->free[i].size += size;
125     return 0; /* 成功完成 */
126   }
127 }
```

前端后端均被占用:

由于被释放空间周围没有空闲内存, 为保证 free 内各段内存仍然按照内存地址升序排列, 使空闲空间计数最大值加一, free[i] 后续空闲内存序号加一, 并将释放空间序号定为 i。

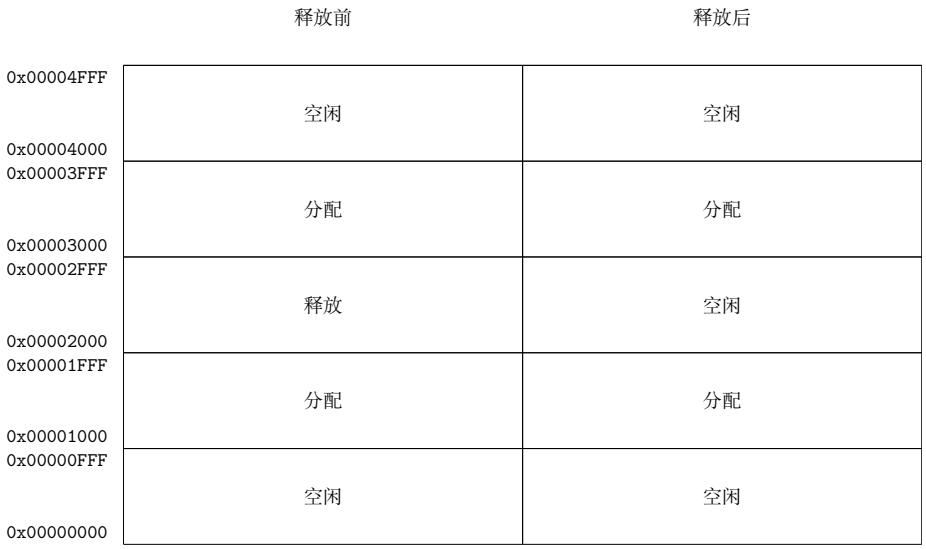

图 5-4　前端后端均被占用

实现如下:

```
128  /* 既不能与前面归纳到一起,也不能与后面归纳到一起 */
129  if (man->frees < MEMMAN_FREES) {
130    /* free[i] 之后的,向后移动,腾出一点可用空间 */
131    for (j = man->frees; j > i; j--) {
132      man->free[j] = man->free[j - 1];
133    }
134    man->frees++;
135    if (man->maxfrees < man->frees) {
136      man->maxfrees = man->frees; /* 更新最大值 */
137    }
138    man->free[i].addr = addr;
139    man->free[i].size = size;
140    return 0; /* 成功完成 */
141  }
```

5.2 输入输出

输入作为人与计算机之间最基本的交互方式, 其中键盘和鼠标作为标准输入设备。

```
151  i = fifo32_get(&fifo);
```

5.2.1 键盘输入

当输入为 256 到 511, 系统判定为键盘输入。

```
161 if (256 <= i && i <= 511) { /* 键盘数据 */
```

键盘输入分为字母输入, 常规按键, 特殊按键

5.2.2 鼠标输入

当输入为 512 到 767, 系统判定为鼠标输入。

```
247 } else if (512 <= i && i <= 767) { /* 鼠标数据 */
```

5.2.3 标准输出

5.3 多道程序系统

现代的计算机已经不仅仅作为数字计算的工具, 而进入大众生活的计算机被赋予了更多的生活需求, 用户可能在看电影的同时查看电子邮件, 也有可能在写论文的时候进入浏览器查询相关资料, 但是更重要的是计算机往往在用户不经意间打开防病毒软件等保证用户计算机的安全[6]。

由此可见多进程的工作方式在计算机工作中同样不可或缺。

但是在实际的处理过程中, 计算机并不能同时处理多个程序, 所以必须采用分时的设计, 关于分时操作系统的设计在下一节。在此有两个概念, 同时处理和多个程序, 同时处理属于分时, 多道程序属于多道程序设计。

首先要处理的问题是如何在运行多个程序,。

5.4 分时操作系统

在上一节中说到分时是使得在用户看来计算机的多道程序同时运行, 多道程序已经实现了, 分时简单说是使得 CPU 在用户不能明显感觉到的时间间隔内切换运行多个程序, 在切换的过程中...

6　实现对外兼容及安全防护

从接口设计及安全防护的角度完善操作系统

参考文献

[1] K. Hidemi, *30 天完成! OS 自作入門*, minabi publication, **2006**.

[2] IBM, 1401 Data Processing System, **2003**.

[3] IBM, 7094 Data Processing System, **2003**.

[4] 阮一峰, 如何变得有思想, **2014**.

[5] 刘伟, 数据恢复技术深度揭秘, **2010**.

[6] A. S. Tanenbaum, *Modern operating system*, Pearson Education, Inc, **2009**.

指导教师简介

王晓林, 男, 49 岁, 硕士, 讲师, 毕业于英国格林尼治大学, 分布式计算系统专业。现任西南林业大学计信学院教师。执教 Linux、操作系统、网络技术等方面的课程, 有丰富的 Linux 教学和系统管理经验。

致　谢

感谢,

附录 A　我也不知道为什么要写附录

可以参考模版目录中的 appendix.tex 文件来写。

附录B　主要程序代码

B.1　初始启动程序代码（ipl09.nas）节选

```
12  DB    " zbote  "  ; 启动扇区名称(8 字节)
13  DW    512    ; 每个扇区(sector)大小(必须 512 字节)
14  DB    1    ; 簇(cluster)大小(必须为 1 个扇区)
15  DW    1    ; FAT 起始位置(一般为第一个扇区)
16  DB    2    ; FAT 个数(必须为 2)
17  DW    224    ; 根目录大小(一般为 224 项)
18  DW    2880    ; 该磁盘大小(必须为 2880 扇区 1440*1024/512)
19  DB    0xf0    ; 磁盘类型(必须为 0xf0)
20  DW    9    ; FAT 的长度(必须为 9 扇区)
21  DW    18    ; 一个磁道(track)有几个扇区(必须为 18)
22  DW    2    ; 磁头数(必须为 2)
23  DD    0    ; 不使用分区,必须是 0
24  DD    2880    ; 重写一次磁盘大小
25  DB    0,0,0x29  ; 意义不明(固定)
26  DD    0xffffffff  ; (可能是)卷标号码
27  DB    "    ZOS    "  ; 磁盘的名称(必须为 11 字节,不足填空格)
28  DB    "FAT12   "  ; 磁盘格式名称(必须为 8 字节,不足填空格)
29  RESB  18      ; 先空出 18 字节
```

程序 B-1　FAT12 格式磁盘专用代码

```
76  readfast:  ; 使用 AL 尽量一次性读取数据 从此开始
77  ; ES: 读取地址, CH: 柱面, DH: 磁头, CL: 扇区, BX: 读取扇区数
78
79      MOV    AX,ES      ; < 通过 ES 计算 AL 的最大值 >
80      SHL    AX,3      ; 将 AX 除以 32,将结果存入 AH(SHL 是左移位指令)
81      AND    AH,0x7f      ; AH 是 AH 除以 128 所得的余数(512*128=64K)
82      MOV    AL,128      ; AL = 128 - AH; AH 是 AH 除以 128 所得的余数
        ↪ (512*128=64K)
83      SUB    AL,AH
84
85      MOV    AH,BL      ; < 通过 BX 计算 AL 的最大值并存入 AH >
86      CMP    BH,0      ; if (BH != 0) { AH = 18; }
87      JE    .skip1
88      MOV    AH,18
```

程序 B-2　将磁盘内容读入内存