[2][1][3]

西南林业大学

本科毕业（设计）论文

(二〇一八届)

题　　目： RongOS — 一个简单操作系统的实现

分院系部： 计算机与信息科学学院

专　　业： 计算机科学与技术专业

姓　　名： 蒲启元

导师姓名： 王晓林

导师职称： 讲师

二〇一八 年 六 月

RongOS — 一个简单操作系统的实现

蒲启元

（西南林业大学 计算机与信息科学学院，云南 昆明 650224）

摘　要： 操作系统管理着计算机的硬件和软件资源，它是向上层应用软件提供服务（接口）的核心系统软件，这些服务包括进程管理，内存管理，文件系统，网络通信，安全机制等。操作系统的设计与实现则是软件工业的基础。为此，在国务院提出的《中国制造 2025》中专门强调了操作系统的开发。但长期以来，操作系统核心开发技术都掌握在外国人手中，技术受制，对于我们的软件工业来说很不利。本文从零开始设计开发一个简单的操作系统，包括 boot loader，中断，内存管理，图形接口，多任务，以及在这个系统上的几个小应用等。尽管这个系统很简单，但它为自主开发操作系统做了一个小小的尝试。

关键词： 操作系统，进程，内存，中断，boot loader

RongOS — A simple OS implementation

Qiyuan Pu

School of Computer and Information Science
Southwest Forestry University
Kunming 650224, Yunnan, China

Abstract: Operating system manages the sources of hardware, it lies in the core of the system software and provides services(interfaces) to upper applications. These service including process management, memory management, file system, network communication, security mechanism etc. Operating system development is the foundation and core of software industry. Therefore, «Made in China 2025» emphasize the development of operating system that put forward by The State Council of China. For a long time, however, the OS kernel development technology grasped in the hand of foreigner, it's bad for our software industry cause of limited technology. So this article will design and develop a simple operating system, including boot loader, interrupt, memory management, graphic interface, multitasking, and some little applications based on this system. In spite of the simplicity of this system, it's a small trying for autonomous development operating system.

Key words: operating system, boot loader, process, interrupt, memory management

目　录

插图目录

表格目录

1 Introduction

1.1 Background

1.2 Preliminary Works

For development,

1.2.1 Development Environment

OS platform: 4.12.0-1-amd64

Editor: GNU Emacs 25.2.2

Run time VM: QEMU emulator 2.8.1

1.2.2 Tools

Some tools used to develop RongOS, see tools.[1].

[1] https://github.com/Puqiyuan/RongOS/tree/master/Tools

2 Design

2.1 Top Level Design

2.2 Detailed Design

2.2.1 Boot Loader

This is working flow of boot loader:

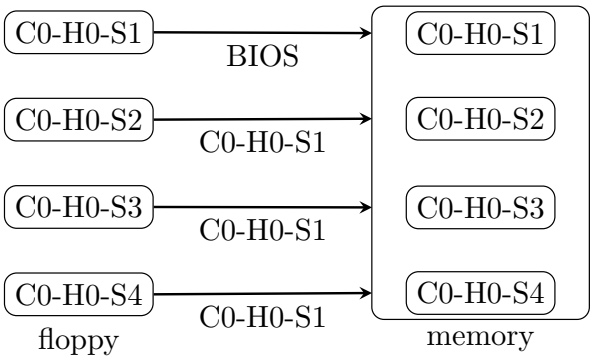

图 2-1 Working Flow of Boot Loader

The instructions of boot loader saved in C0-H0-S1 of floppy, the first cylinder, head 0, the first sector, total 512 byte. These instructions end with 0x55 0xaa, so BIOS will load C0-H0-S1 to memory, then the instructions in C0-H0-S1 will load C0-H0-S2 — C9-H1-S18, total $10 * 2 * 18 * 512 = 184320byte = 180KB$(including boot sector, C0-H0-S1) to main memory.

2.2.2 32-bit Mode and Import C Codes

3 Implementation

3.1 Boot Loader

3.1.1 Choose Disk

There are many ways to boot an operating system, from hard disk, USB, floppy disk etc. I choose floppy disk, although it is out of date. For my purpose is that develop a simple operating system, pay my attention on how to development. The structure of floppy disk is simple and for my simple operating system it's enough.

3.1.2 The Structure of a Floppy Disk

Fig. 3-1 shows the inside of a floppy disk:

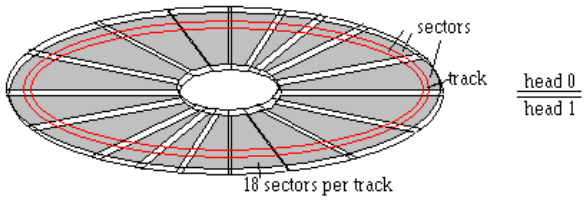

图 3-1 Floppy Disk Structure

A floppy disk, also called a floppy, diskette, or just disk, is a type of disk storage composed of a disk of thin and flexible magnetic storage medium, sealed in a rectangular plastic enclosure lined with fabric that removes dust particles. Floppy disks are read and written by a floppy disk drive (FDD).

For 3.5 inch HD floppy, There are 80 cylinders from the outermost to the core on each side, numbering 0, 1, ..., 79. The head can assign be 0 or 1, representing two sides of floppy. When specify head number and cylinder number, forming a ring, named track in jargon. The track is large so we divide it to 18 small parts, named sector. A sector can store 512 byte. So the capacity of a floppy is:

$$18 * 80 * 2 * 512 = 1474560Byte = 1440KB.$$

3.1.3 Flowchart of Boot Loader

Fig. 3-2 shows how the boot loader works.

The boot loader is implemented in Intel assembly.

1. Display boot information: Firstly, the boot sector display some boot information, when $al = 0$, the null character of boot information hit. Interrupt $0x10$ is used for show a character. Appendix 1 is the code to perform this function.
2. Read the second sector: Then jump to load C0-H0-S2, ax register saved the address where beginning puts the sectors from floppy. And preparing parameters for interrupt $0x13$ in registers. The $0x13$ interrupt used for read sector from floppy to memory. Appendix 2 is the code to perform this function.
3. Read two sides of a track: If there is a carry, representing some thing wrong when read floppy, so reset the registers and try again read floppy, until five times trying. Register si is a counter. If no carry, jump to next segmentation, as one sector read to memory already, the address space should increase 512 byte. Then sector number(cl register) added 1 and compare it to 18, if it's smaller than 18, jump to $readloop$, read the next sector. If the value of cl register bigger or equal to than 18, meaning that one track 18 sector in this side of floppy read already, then reversed the head, add 1 to dh register. If the value of dh register after adding larger than or equal to 2, it's saying the original head is 1, one track of two sides read already. Otherwise the value of dh register smaller than 2, read this side indicating by dh register, jump to $readloop$ segmentation. Appendix 3 is the code to perform this function.
4. The next cylinder: So the next step is moving a cylinder, add 1 to register ch. Otherwise the value of dh register smaller than 2, read this side indicating by dh register, jump to $readloop$ segmentation. After ch register add 1, if it's smaller than 10, jump to $readloop$, otherwise end loading floppy to memory process, for we only load ten cylinders of floppy. Appendix 4 is the code to perform this function.

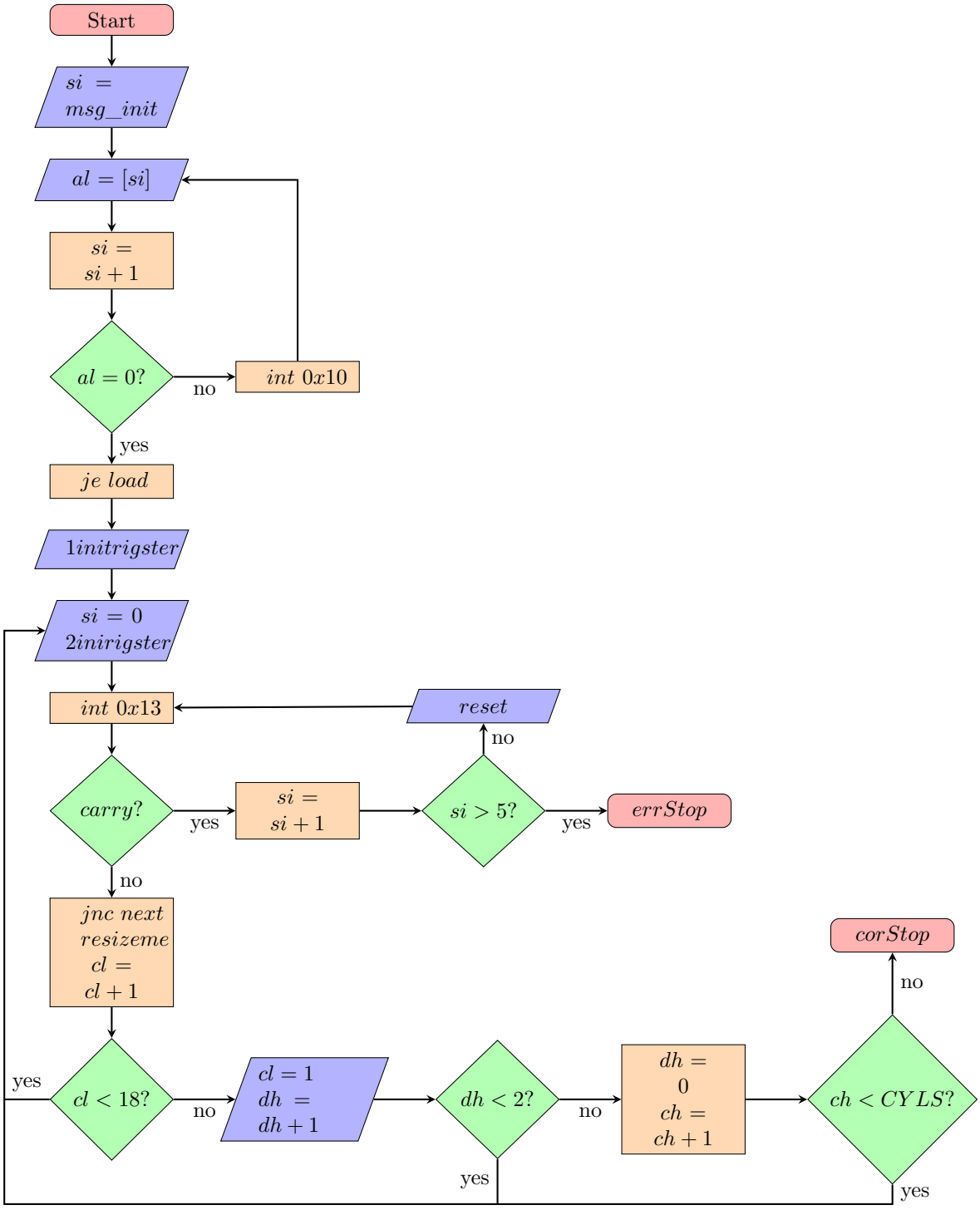

图 3-2 Flowchart of Boot Loader

3.1.4 Running Result

Fig. 3-3 shows the running results of boot loader. From this picture we see that the boot loader loaded 10 cylinders from floppy successfully.

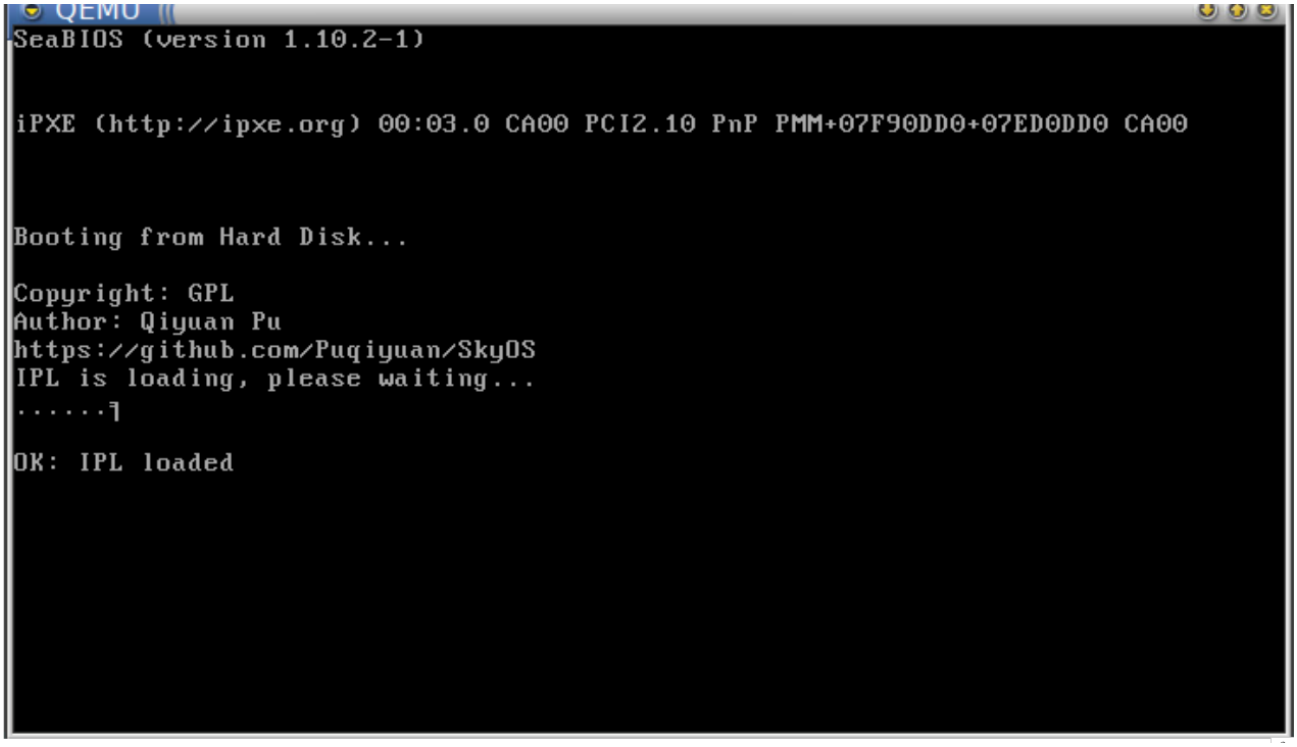

图 3-3 Running Result of Boot Loader

3.2 32-bit Mode and Import C Codes

3.3 Screen Display and Text

3.4 Control Mouse

3.5 Memory Management

3.6 Making Window

3.7 Timer

3.8 Multitasking

3.9 Command Line Window

3.10 API

3.11 OS Protection

3.12 Graphics Processing

3.13 Window Operation

3.14 Application Protection

3.15 File Operation

3.16 Some Applications

3.17 Prospects and Shortages

参考文献

[1] 国务院。*中国制造 2025*，2015-05。。

[2] WiKipedia. *Operating System*, 2017-08..

[3] 川合秀实. *30 天自制操作系统*. 人民邮电出版社, 2012-08.

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

首先我想感谢我的老师，王晓林。大学期间，他给了我很多指导，包括专业方面和上大学的意义等。很多时候，他对学生的要求看起来都是不近情理的，但正是通过这个“痛苦”的过程，我锻炼了坚强的意志，和战胜困难的信心。谢谢你，王老师。我最想感谢的是我的女友，她容忍我在完成这个设计时的很多个夜晚不陪她，给我支持，鼓励我，不抱怨。所以我愿意把这个简单操作系统命名为 RongOS, 蓉便是她名字的最后一个字。谢谢你，我最亲爱的。

附录 A　Main Program Code

```
55 init:
56         mov al, [si]
57         add si, 1 ; increment by 1.
58         cmp al, 0
59         je load ; if al == 0, jmp to load, the msg_init info displayed.
60 ; the lastest character is null character, coding in 0.
61
62         mov ah, 0x0e ; write a character in TTY mode.
63         mov bx, 15   ; specify the color of the character.
64         int 0x10 ; call BIOS function, video card is number 10.
65         jmp init
```

程序 1　Display boot information

```
87  load:
88          mov ax, 0
89          mov ax, 0x0820 ; load C0-H0-S2 to memory begin with 0x0820.
90          mov es, ax
91          mov ch, 0 ; cylinder 0.
92          mov dh, 0 ; head 0.
93          mov cl, 2 ; sector 2.
94
95
96  readloop:
97          mov si, 0 ; si register is a counter, try read a sector
98  ; five times.
99
100
101 retry:
102         mov ah, 0x02 ; parameter 0x02 to ah, read disk.
103         mov al, 1 ; parameter 1 to al, read disk.
104         mov bx, 0
105         mov dl, 0x00 ; the number of driver number.
106         int 0x13 ; after prepared parameters, call 0x13 interrupted.
```

程序 2　Read the second sector

```
108         jnc next ; if no carry read next sector.
109         add si, 1 ; tring again read sector, counter add 1.
110         cmp si, 5 ; until five times
111         jae error ; if tring times large than five, failed.
112
113         ; reset the status of floppy and read again.
114         mov ah, 0x00
115         mov dl, 0x00
116         int 0x13
117         jmp retry
118
119
120 next:
121         mov ax, es
122         ; we can not directly add to es register.
123         add ax, 0x0020 ; add 0x0020 to ax
124         mov es, ax ; the memory increase 0x0020 * 16 = 512 byte.
125         ; size of a sector.
126         add cl, 1 ; sector number add 1.
127         cmp cl, 18 ; one track have 18 sector.
128         jbe readloop ; jump if below or equal 18, read the next sector.
129         mov cl, 1 ; cl number reset to 1, ready to read the other side.
130         add dh, 1 ; the other side of floppy.
131         cmp dh, 2 ; only two sides of floppy.
132         jb readloop ; if dh < 2, read 18 sectors of the other sides
```

程序 3　Read two sides of a track

```
134  mov dh, 0 ; after finished read the other side, reset head to 0.
135  add ch, 1 ; two sides of a cylinder readed, add 1 to ch.
136  cmp ch, CYLS ; read 10 cylinders.
137  jb readloop
```

程序 4　The next cylinder