

西南林业大学  
本科毕业（设计）论文  
(二〇一八届)

题    目： RongOS — 简单操作系统的实现

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： 蒲启元

导师姓名： 王晓林

导师职称： 讲师

二〇一八年六月

# RongOS — 简单操作系统的实现

蒲启元

(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<sup>[1]</sup>。但长期以来，操作系统核心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。本项目从零开始设计开发个简单的操作系统，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 Big Data and Intelligence Engineering  
Southwest Forestry University  
Kunming 650224, Yunnan, China

**Abstract:** Operating system manages the resources of hardware and software, it lies in the core of the system software and provides services(interfaces) to upper applications. These services include process management, memory management, file system, network communication, security mechanism and more. Operating system development is the foundation and core of software industry. Therefore, *Made in China 2025* emphasizes the development of operating system that put forward by The State Council of China. For a long time, however, the OS kernel development technology is mastered by foreigner, due to technical limitations it is detrimental to our software industry. So this project 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

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background . . . . .	1
1.2	Preliminary Works . . . . .	1
1.2.1	Development Environment . . . . .	1
1.2.2	Tools . . . . .	2
1.2.3	Platform Setup . . . . .	2
<b>2</b>	<b>Leading Knowledge</b>	<b>3</b>
2.1	Instruction Set . . . . .	3
2.2	Register . . . . .	3
<b>3</b>	<b>Design</b>	<b>4</b>
3.1	Top Level Design . . . . .	4
3.2	Detailed Design . . . . .	4
3.2.1	Boot Loader . . . . .	4
3.2.2	32-bit Mode and Import C Codes . . . . .	5
<b>4</b>	<b>Implementation</b>	<b>6</b>
4.1	Boot Loader . . . . .	6
4.1.1	Choose Disk . . . . .	6
4.1.2	The Structure of a Floppy Disk . . . . .	6
4.1.3	Flowchart of Boot Loader . . . . .	7
4.1.4	Running Result . . . . .	8
4.2	32-bit Mode and Import C Codes . . . . .	10
4.3	Screen Display and Text . . . . .	10
4.4	Control Mouse . . . . .	10
4.5	Memory Management . . . . .	10

4.6	Making Window . . . . .	10
4.7	Timer . . . . .	10
4.8	Multitasking . . . . .	10
4.9	Command Line Window . . . . .	10
4.10	API . . . . .	10
4.11	OS Protection . . . . .	10
4.12	Graphics Processing . . . . .	10
4.13	Window Operation . . . . .	10
4.14	Application Protection . . . . .	10
4.15	File Operation . . . . .	10
4.16	Some Applications . . . . .	10
<b>5</b>	<b>Prospects And Shortages</b>	<b>11</b>
	<b>参考文献</b>	<b>12</b>
	<b>指导教师简介</b>	<b>12</b>
	<b>致 谢</b>	<b>14</b>
<b>A</b>	<b>Main Program Code</b>	<b>15</b>
A.1	Boot loader . . . . .	15
A.1.1	Display boot information . . . . .	15
A.1.2	Read the second sector . . . . .	15
A.1.3	Read two sides of a track . . . . .	16
A.1.4	The next cylinder . . . . .	17

# 插图目录

3-1	Workflow of the boot loader . . . . .	4
4-1	Floppy Disk Structure . . . . .	6
4-2	Flowchart of Boot Loader . . . . .	7
4-3	Running Result of Boot Loader . . . . .	8

# 表格目录

# Todo list

compiler? . . . . .	2
debugger? . . . . .	2
need more words. . . . .	2
What's Qemu? . . . . .	2
What's Wine? . . . . .	2
need rephrase . . . . .	2
Figure: System overview diagram . . . . .	4
need some description . . . . .	4
Question: is this section about how bootloader works, or about how to load it into the mem? . . . . .	4
need a new figure . . . . .	4
Figure: MBR structure . . . . .	4
the 446 bytes code in MBR is for ... . . . .	4
need rephrase . . . . .	6
cite wikipedia . . . . .	6
need a better chart . . . . .	7
need a better pic. . . . .	8



# 1 Introduction

## 1.1 Background

Contemporary software systems are beset by problems that create challenges and opportunities for broad new OS research. There are five areas could improve user experience including dependability, security, system configuration, system extension, and multiprocessor programming.

The products of forty years of OS research are sitting in everyone's desktop computer, cell phone, car, etc., and it is not a pretty picture. Modern software systems are broadly speaking complex, insecure, unpredictable, prone to failure, hard to use, and difficult to maintain. Part of the difficult is that good software is hard to write, but in the past decade, this problem and more specific shortcomings in systems have been greatly exacerbated by increased networking and embedded systems, which placed new demands that existing architectures struggled to meet. These problems will not have simple solutions, but the changes must be pervasive, starting at the bottom of the software stack, in the operating system.

The world needs broad operating system research. Dependability, security, system configuration, system extension, and multi-processor programming illustrate areas where contemporary operating systems have failed to meet the software challenges of the modern computing environment.

## 1.2 Preliminary Works

### 1.2.1 Development Environment

**OS platform:** Debian 9, Linux kernel 4.12.0-1-amd64

**Editor:** GNU Emacs 25.2.2

**Run time VM:** QEMU emulator 2.8.1

**Assembler:** Nask.

**Compiler:** \_\_\_\_\_ } compiler?

**Debugger:** \_\_\_\_\_ } debugger?

## 1.2.2 Tools

Some tools used to develop RongOS, see tools.<sup>1</sup>

need more words.

## 1.2.3 Platform Setup

Debian System: there is a small tutorial.<sup>2</sup>

### Qemu

What's Qemu?

QEMU, for my x86\_64 architecture:

```
$ sudo apt-get install qemu-system-x86_64
```

### Wine

What's Wine?

Note that the tools is exe formate, so on Debian system, you need to install wine:

```
$ sudo apt-get update
```

```
$ sudo apt-get install wine
```

### Debian i386 support

Maybe you also need to add i386 architecture cause of AMD64 on your machine to use these tools: \_\_\_\_\_

```
$ sudo dpkg --add-architecture i386
```

```
$ sudo apt-get update
```

} need  
rephrase

<sup>1</sup><https://github.com/Puqiyuan/RongOS/tree/master/Tools>

<sup>2</sup>[http://cs2.swfc.edu.cn/~wx672/lecture\\_notes/linux/install.html](http://cs2.swfc.edu.cn/~wx672/lecture_notes/linux/install.html)

## 2 Leading Knowledge

### 2.1 Instruction Set

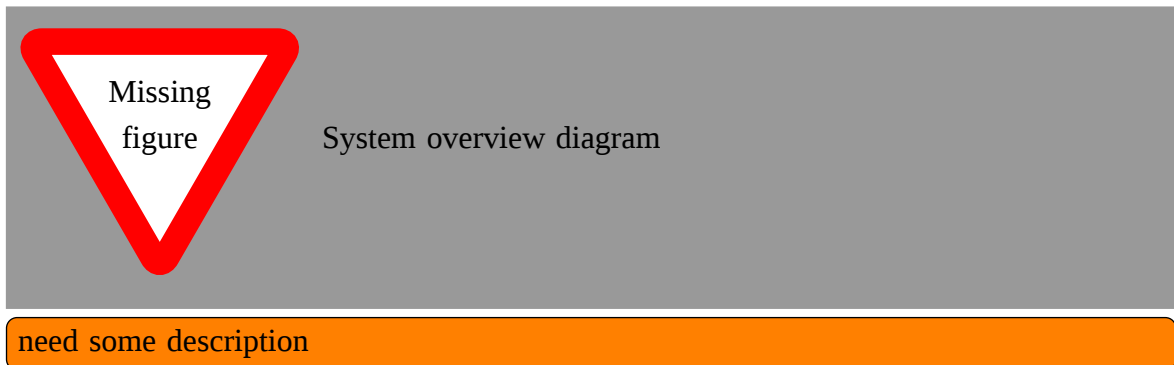
- db: the abbreviation of define byte, write a byte, also 8 bits to file.
- resb: the abbreviation of reserve byte, reserved bytes and filling 0x00 in these reserved space.
- dw: the abbreviation of define word, write two bytes, also 16 bits to file.
- dd: the abbreviation of define double-word, write four bytes, also 32 bits to file.
- org: load the program to specified address.
- jmp: jump to another instruction.
- mov: assign the right value to left variable.

### 2.2 Register

- ax: accumulator    • cx: counter    • dx: data
- bx: base    • sp: stack pointer    • bp: base pointer
- si: source index    • di: destination index    • al: accumulator low
- cl: counter low    • dl: data low    • bl: base low
- ah: accumulator high    • ch: counter high    • dh: data high
- bh: base high    • es: extra segment    • cs: code segment

## 3 Design

### 3.1 Top Level Design



### 3.2 Detailed Design

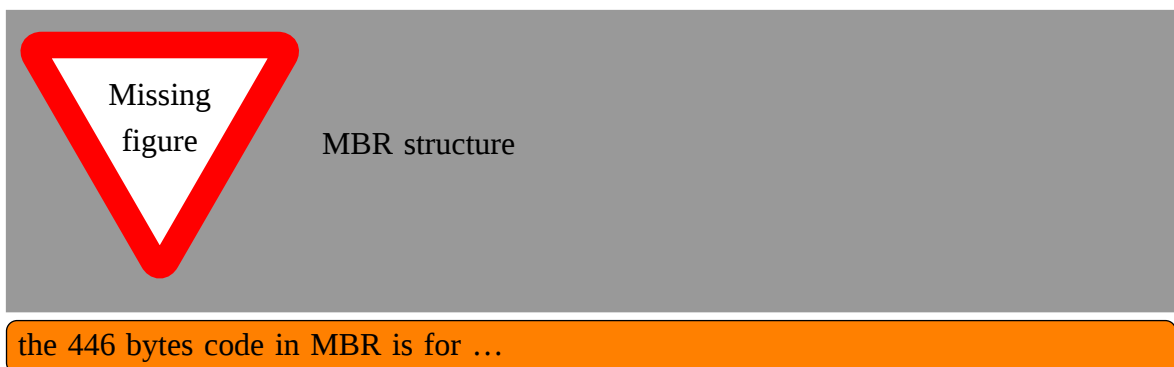
#### 3.2.1 Boot Loader

~~This is the workflow of the boot loader:~~

Question: is this section about how bootloader works, or about how to load it into the mem?

need a new figure

图 3-1 Workflow of the boot loader



The instructions of the 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  $10 * 2 * 18 * 512 = 184320\text{byte} = 180KB$  (including boot sector, C0-H0-S1) to main memory.

### 3.2.2 32-bit Mode and Import C Codes

## 4 Implementation

### 4.1 Boot Loader

#### 4.1.1 Choose Disk

need rephrase

There are many ways to boot an operating system, from hard disk, USB, floppy disk, etc. Among which, floppy disk is the simplest one to deal with, though it's out of date. 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.

#### 4.1.2 The Structure of a Floppy Disk

Fig. 4-1 shows the inside of a floppy disk:

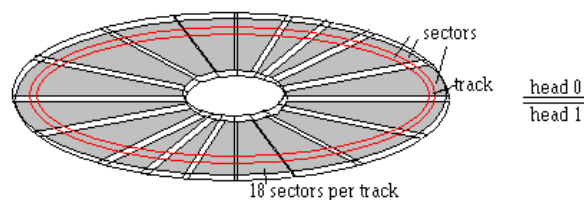


图 4-1 Floppy Disk Structure

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

cite  
wikipedia

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:

$$18 \times 80 \times 2 \times 512 = 1474560 \text{ Byte} = 1440 \text{ KB}$$

### 4.1.3 Flowchart of Boot Loader

Fig. 4-2 shows how the boot loader works.

need a better chart

图 4-2 Flowchart of Boot Loader

The boot loader is implemented in Intel assembly. **It works as following:**

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 A.1.1 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 A.1.2 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 A.1.3 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 A.1.4 is the code to perform this function.

### 4.1.4 Running Result

Fig. 4-3 shows the running results of boot loader. From this picture we see that the boot loader loaded 10 cylinders from floppy successfully.

need a better pic.

图 4-3 Running Result of Boot Loader





## **4.2 32-bit Mode and Import C Codes**

## **4.3 Screen Display and Text**

## **4.4 Control Mouse**

## **4.5 Memory Management**

## **4.6 Making Window**

## **4.7 Timer**

## **4.8 Multitasking**

## **4.9 Command Line Window**

## **4.10 API**

## **4.11 OS Protection**

## **4.12 Graphics Processing**

## **4.13 Window Operation**

## **4.14 Application Protection**

## **4.15 File Operation**

## **4.16 Some Applications**

## **5   Prospects And Shortages**

## 参考文献

- [1] 国务院, 中国制造 2025, chinese, **2015-05**.

## 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

## A.1 Boot loader

### A.1.1 Display boot information

```
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
```

### A.1.2 Read the second sector

```
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.
```

### A.1.3 Read two sides of a track

```
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
```

#### **A.1.4 The next cylinder**

```
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
```