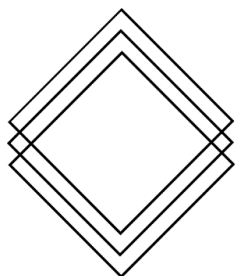




中国石油大学 (华东)  
CHINA UNIVERSITY OF PETROLEUM



OURCE  
Operating System  
UPC-OS-Homework  
BY TOCHUSC

# Ource OS 说明文档

《计算机操作系统》学期作业

姓 名	许祖耀
学 号	2107010120
专业班级	计算 2101
学 院	计算机科学与技术学院

2024 年 5 月 27 日

# 目录

开发环境.....	1
功能介绍.....	1
支撑功能 .....	1
中断处理 .....	1
时钟管理 .....	2
原语操作 .....	2
资源管理 .....	2
进程管理 .....	2
内存管理 .....	2
设备管理 .....	3
文件管理 .....	3
更多细节 .....	3
模块架构.....	3
工作流程.....	5
整体工作流程 .....	5
内核主程序 bootpack.c 工作流程 .....	6
详解说明.....	6
BOOTINFO 结构体 .....	7
FIFO32 结构体 .....	7
SEGMENT_DESCRIPTOR 结构体 .....	8
GATE_DESCRIPTOR 结构体 .....	9
TIMER 结构体 .....	9
TIMERCTL 结构体 .....	9
TSS32 结构体 .....	10
TASK 结构体 .....	10
TASKLEVEL 结构体 .....	11
TASKCTL 结构体 .....	11
FILEINFO 结构体 .....	12
运行测试.....	13
系统启动 .....	13
相关命令 .....	13
输入 help 命令，查看帮助。 .....	13
输入 mem 命令，查看系统内存使用情况 .....	14
输入 dir 命令，查看外存文件目录。 .....	14
通过 type 命令，查看某个文件的内容 .....	15
通过 cls 指令刷新屏幕 .....	15
通过命令行运行 .hrb 可执行二进制文件 .....	16
总结.....	17

# 开发环境

Ource OS 参考川合秀实编著，周自恒翻译的《30 天自制操作系统》中的 Haribote 系统编写，原书的附带资源中提供了一系列开发工具，Ource OS 正是通过原书附带的开发工具编写而成的。

原书提供的开发工具包括但不限于以下内容：

- GNU Make - GNU 项目中的工具之一，用于自动化编译和链接程序。
- Nask - NASM 开源汇编器，支持 Intel 语法，将 .nas 文件编译为 .obj 文件
- ccl - GCC 的一个组件，负责将 C 源代码编译为 AT&T 语法的 .gas 汇编文件。
- gas2nask - 作者编写的工具，用于将 GNU 的汇编代码转换为 NASM 汇编代码。
- obj2bim - 作者编写的链接工具，可以将对象文件转换为 .bim 二进制映像文件。
- bim2hrb - 作者提供的工具，可以将 .bim 二进制映像文件转换为 HariboteOS 中的 .hrb 可执行二进制文件。
- edimg - 用于创建和编辑磁盘映像文件。
- makefont - 可以将字库文件转换为二进制字库文件。
- bin2obj - 将二进制字库文件转换为语言数组，并储存在 .obj 文件中。
- QEMU - 通用开源的计算机仿真器和虚拟机。

在《30 天自制操作系统》这本书的上下文中，作者为了教学与实验目的，使用了自创的 .hrb 文件格式作为书中 Haribote 操作系统的可执行二进制文件的目标格式，Ource 也因此将其沿袭了下来。

Ource 是通过利用上述开发工具，效仿书中 Haribote OS 的开发方式而编写的。

## 功能介绍

Ource OS 实现了操作系统内核程序应该提供的支撑功能和资源管理功能。

### 支撑功能

### 中断处理

通过初始化可编程中断控制器，设置相应的中断向量表，Ource 实现了对中

断的处理，目前支持的中断信号有：

- 异常中断 中断信号 0x0d
- 时钟中断 中断信号 0x20
- 键盘中断 中断信号 0x21
- 鼠标中断 中断信号 0x2c（支持但未使用）
- 电气中断 中断信号 0x27
- 系统调用 中断信号 0x40

## 时钟管理

通过初始化可编程中断计时器 PIC，定义计时器数据结构，实现时钟中断服务程序，Ource 完成了时钟管理功能。

## 原语操作

通过汇编语言编写的 `_io_cli_` 和 `_io_sti_` 函数设置 PIC 的中断屏蔽位，Ource OS 实现了关中断和开中断功能，以执行相应的原语操作。

## 资源管理

Ource 充分担当了 OS 内核应具备的资源管理者身份，实现了进程管理、内存管理、设备管理、文件管理功能。

## 进程管理

Ource 是一个多道分时操作系统，每一个运行在其上的程序都一一对应着一个 TCB 任务控制块（类似 PCB），Ource 使用多级优先队列调度算法实现任务的调度，采用时间片轮转的方式进行任务切换，具有高优先级的任务将会被分配到更长的时间片。

## 内存管理

Ource 使用空闲分区链表记录和追踪内存中的空闲分区使用情况，并采用分段存储管理的方式进行内存管理，通过设置全局段描述符表 GDT 和局部段描述符表 LDT，将内核程序和不同的用户程序分配在具有不同权限的内存段中，以进行内存的管理。

## 设备管理

Ource 通过数个简单的.c 代码文件，分别与键盘、屏幕等设备控制器进行直接通信，实现设备的驱动和管理功能。

## 文件管理

Ource 使用 FAT12 文件系统进行文件管理，目前仅支持文件系统中的文件读取操作。

## 更多细节

Ource 以《30 天自制操作系统中的》中作者提出的.hrb 文件格式作为系统的可执行文件，并使用其提供的工具集进行程序代码编译和链接。

Ource 通过设置不同程序的内存段属性，实现操作系统和应用程序之间的隔离与保护。Ource 向用户提供了联机命令接口，用户可以通过在命令行窗口中输入命令进行操作；同时 Ource 也提供程序接口，用户程序可以通过系统调用函数，发出中断信号，转移到相应的服务例程中，目前提供了往控制台终端输出字符和换行的系统调用函数 `api_putchar` 和 `api_end`。

Ource 的屏幕显示模式默认为 1024x768x8bit 彩色，并通过调色板模式显示彩色，通过修改内存中 VRAM 指定的区域，Ource 控制屏幕中每个像素的显示色彩，并通过外部挂载的二进制字库文件逐像素绘制字符。

## 模块架构

Ource OS 的内核功能由 14 个功能模块组成，其分别为.c 文件或.nas 汇编语言文件分别负责各个独立功能，共同组成 Ource 的核心内核程序。Ource OS 的核心功能文件如下：

- `ipl10.nas` - 汇编语言编写的操作系统引导扇区 IPL 程序，负责初始化计算机硬件并加载操作系统的其余部分。
- `asmhead.nas` - 汇编语言编写的内核启动代码，负责检测硬件环境，初始化内核运行环境等。
- `naskfunc.nas` - 定义了内核程序需要的且只能通过汇编语言实现的辅助函数等，如关中断 `_io_cli` 和开中断 `_io_sti`。
- `bootpack.c` - C 语言编写的内核主程序，负责初始化描述符表、内存、进程

等。

- `console.c` - 命令行终端的相关功能实现，向用户提供命令接口。
- `dsctbl.c` - 用于管理各种描述表，如全局段描述符表 GDT，中断描述符表 IDT。
- `fifo.c` - 先入先出型缓冲区的相关实现代码。
- `file.c` - 对 FAT12 文件系统进行操作与管理的相关代码实现。
- `int.c` - 负责初始化可编程中断控制器 PIC，并实现部分必需的中断服务程序。
- `keyboard.c` - 负责驱动键盘，与键盘设备控制器进行通信，定义了键盘中断的中断服务程序。
- `memory.c` - 内存管理部分的代码实现，使用空闲分区链表进行内存分配、回收等，
- `mtask.c` - 任务（进程）管理的代码实现，使用多级优先队列，采用时间片轮转方式进行进程调度等。
- `timer.c` 时钟管理部分的代码实现，负责初始化可编程间隔计时器，向应用程序分配计时器等，定义了时钟中断的中断服务程序。
- `graphic.c` - 负责驱动屏幕，进行图形绘制的相关代码实现。
- `window.c` - 在屏幕上绘制应用窗口的相关代码。

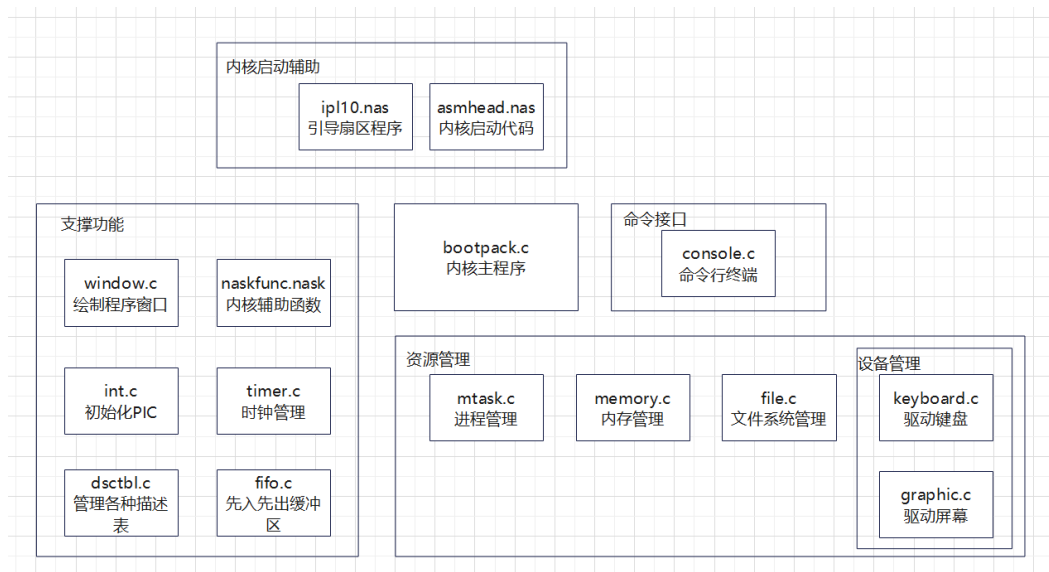


图 1 Ource 内核模块组织结构图

# 工作流程

## 整体工作流程

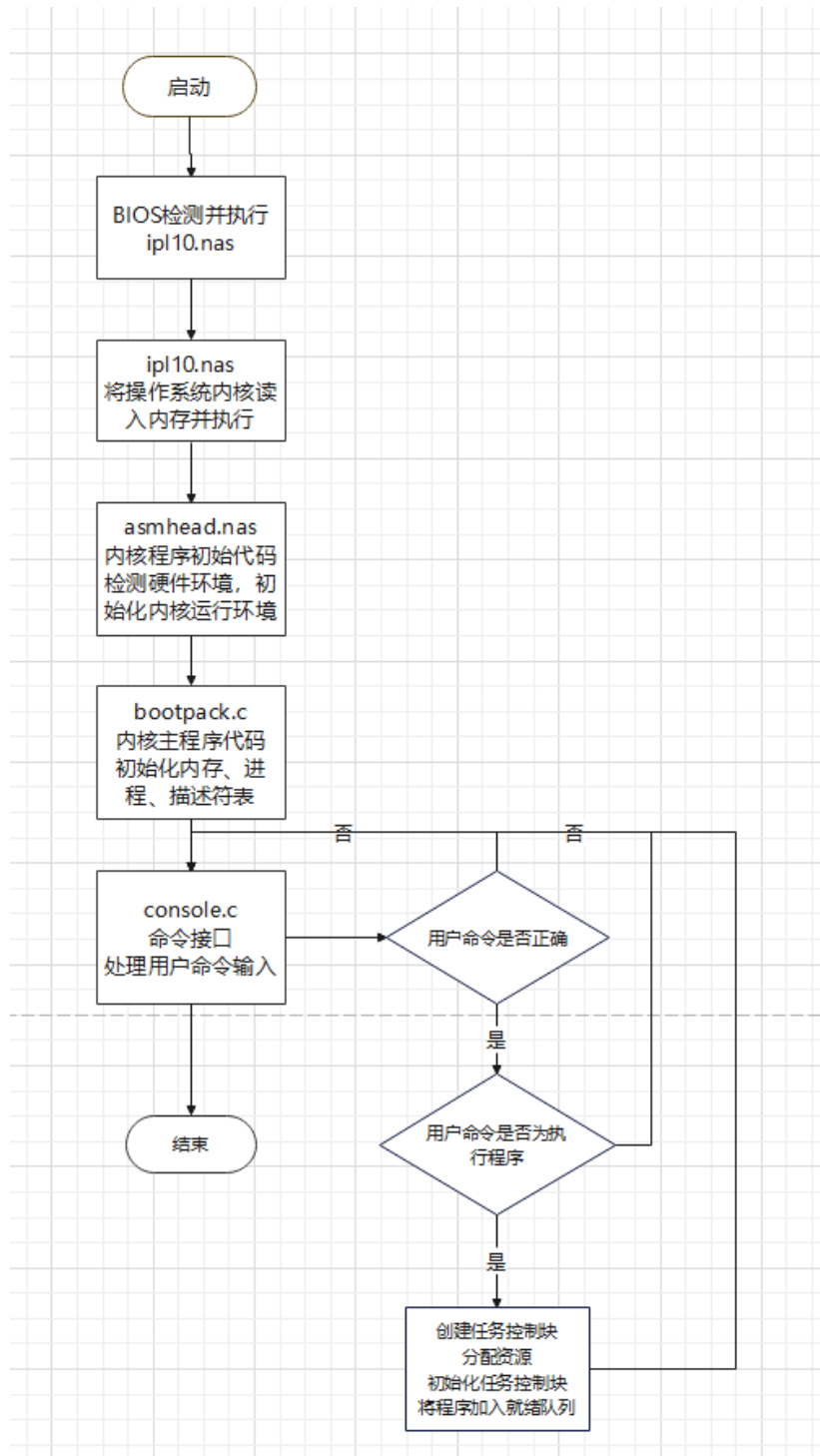


图 2 Oource 启动后的工作流程

# 内核主程序 bootpack.c 工作流程

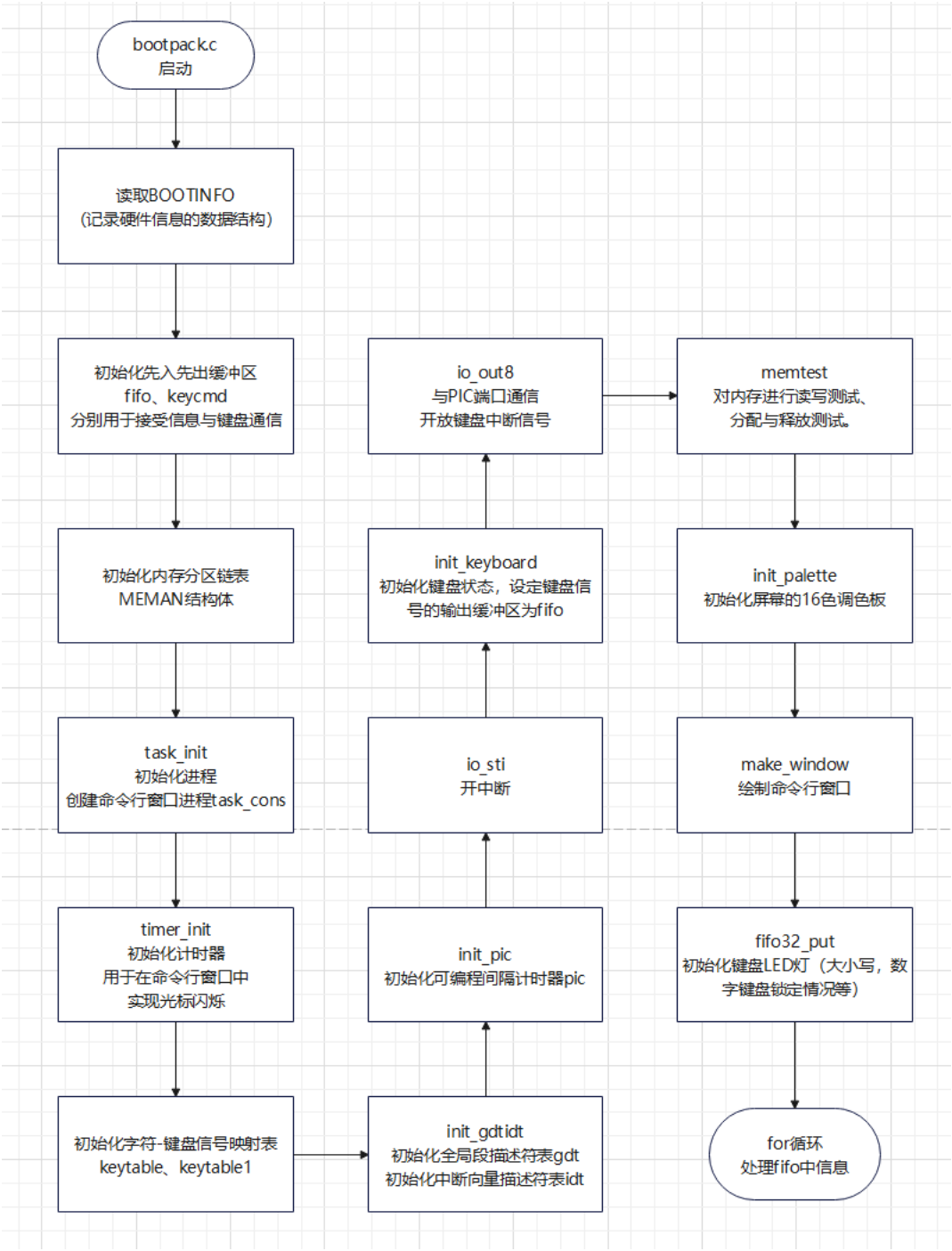


图 3 bootpack.c 内核主程序的工作流程

## 详解说明

Ource 的代码中有着若干个重要的数据结构，其储存了 OS 内核运行所需要的关键信息。



## BOOTINFO 结构体

用于储存计算机的相关硬件信息的结构体。

```
/* asmhead.nas */
struct BOOTINFO { /* 0x0ff0-0x0fff */
    char cyls; /* 启动区读磁盘读到此为止 */
    char leds; /* 启动时键盘的LED的状态 */
    char vmode; /* 显卡模式为多少位彩色 */
    char reserve;
    short scrnx, scrny; /* 画面分辨率 */
    char *vram;
};
#define ADR_BOOTINFO 0x00000ff0
```

BOOTINFO 结构体定义在 asmhead.nas 文件中，其位于内存地址单元 0xff0 处。

```
; BOOT_INFO 相关
CYLS    EQU    0x0ff0        ; 引导扇区设置
LEDS     EQU    0x0ff1
VMODE    EQU    0x0ff2        ; 关于颜色的信息
SCRNX    EQU    0x0ff4        ; 分辨率X
SCRNY    EQU    0x0ff6        ; 分辨率Y
VRAM     EQU    0x0ff8        ; 图像缓冲区的起始地址
```

成员变量说明：

- cyls: 引导扇区的结束位置
- leds: 启动时键盘 LED 灯状态
- vmode: 显卡的显示模式
- reserve: 保留字节
- scrnx, scrny: 画面的纵横分辨率
- vram: 图像缓冲区的起始位置，通过修改 vram 可设置屏幕像素的显示颜色。

## FIFO32 结构体

实现先入先出的消息队列的数据结构，每个消息大小为 4 字节，32 位，以 int 型表示。

```

/* fifo.c */
struct FIF032 {
    int *buf;
    int p, q, size, free, flags;
    struct TASK *task;
};
void fifo32_init(struct FIF032 *fifo, int size, int *buf, struct TASK *task);
int fifo32_put(struct FIF032 *fifo, int data);
int fifo32_get(struct FIF032 *fifo);
int fifo32_status(struct FIF032 *fifo);

```

相关功能实现在 fifo.c 文件中。

成员变量说明：

- buf: 指向消息队列的指针
- p: 队首指针
- q: 队尾指针
- size: 消息队列的大小
- free: 剩余的空闲空间数
- flags: 表示消息队列的状态（为 1 时表示发生溢出）
- task: 指向消息队列所服务的进程

## SEGMENT\_DESCRIPTOR 结构体

段描述符数据结构，用于定义段的属性。

```

struct SEGMENT_DESCRIPTOR {
    short limit_low, base_low;
    char base_mid, access_right;
    char limit_high, base_high;
};

```

相关功能实现在 dsctbl.c 文件中。

成员变量说明：

- limit\_low: 段界限的低 16 位。
- base\_low: 段基地址的低 16 位。
- base\_mid: 段基地址的中间 8 位。
- access\_right: 访问权限字段，定义了段的类型和访问权限。
- limit\_high: 段界限的最高 4 位。
- base\_high: 段基地址的最高 8 位。

## GATE\_DESCRIPTOR 结构体

门描述符数据结构，用于定义中断或异常处理程序的入口点。

```
struct GATE_DESCRIPTOR {
    short offset_low, selector;
    char dw_count, access_right;
    short offset_high;
};
```

相关功能实现在 dsctbl.c 文件中。

成员变量说明：

- offset\_low: 中断或异常处理程序的偏移地址的低 16 位。
- selector: 段选择器，指向包含中断或异常处理程序代码的段。
- dw\_count: 储存访问权限字段的高 8 位。
- access\_right: 访问权限字段，定义了门的类型和访问权限。
- offset\_high: 中断或异常处理程序的偏移地址的高 16 位。

## TIMER 结构体

实现计时器对象的数据结构。

```
struct TIMER {
    struct TIMER *next;
    unsigned int timeout, flags;
    struct FIFO32 *fifo;
    int data;
};
```

相关功能实现在 dsctbl.c 文件中。

成员变量说明：

- next: 指向下一个 TIMER 结构体的指针，用于构建定时器链表。
- timeout: 表示定时器的超时时间。
- flags: 用于存储定时器的状态（0 为未启用，1 为启用）
- fifo: 指向 FIFO32 先进先出队列，当定时器触发时，向该队列传输消息。
- data: 用于存储与定时器触发时向 FIFO32 传输的数据。

## TIMERCTL 结构体

计时器控制块，实现对计时器对象进行控制的数据结构。

```
struct TIMERCTL {
    unsigned int count, next;
    struct TIMER *t0;
    struct TIMER timers0[MAX_TIMER];
};
```

相关功能实现在 dsctbl.c 文件中。

成员变量说明：

- count: 用于存储当前启用的定时器数量。
- next: 表示下一个将要超时的定时器的时间。
- t0: 指向下一个将要超时的计时器的指针，指向计时器链表的头部。
- timers0: TIMER 结构体数组，存储系统中所有定时器对象。

## TSS32 结构体

储存当前进程的处理器上下文的数据结构，包含了 CPU 在执行任务切换时需要保存和恢复的寄存器状态。

```
struct TSS32 {
    int backlink, esp0, ss0, esp1, ss1, esp2, ss2, cr3;
    int eip, eflags, eax, ecx, edx, ebx, esp, ebp, esi, edi;
    int es, cs, ss, ds, fs, gs;
    int ldtr, iomap;
};
```

相关操作实现在 mtask.c 文件中

成员变量说明：

- backlink: 指向先前任务的 TSS 的链接。
- esp0 到 ss2: 分别为不同特权级别的堆栈指针和堆栈段选择器。
- cr3: 存储当前任务的页目录基地址寄存器值。
- eip, eflags, eax 到 edi: 通用寄存器和指令指针，保存当前任务的执行状态。
- es 到 gs: 段寄存器。
- ldtr: 局部描述符表寄存器。
- iomap: I/O 位图的基地址，用于处理 I/O 权限。

## TASK 结构体

用于表示进程控制块 PCB 的数据结构，包含了操作系统了解进程状态，以及对进程进行管理的所有必要信息。

```

struct TASK {
    int sel, flags;    /* sel用来存放GDT的编号*/
    int level, priority; /* 优先级 */
    struct FIFO32 fifo;
    struct TSS32 tss;
};

```

相关操作实现在 mtask.c 文件中

成员变量说明：

- sel: 任务的全局描述符表（GDT）编号。
- flags: 任务的标志位。
- level: 任务的优先级级别。
- priority: 任务的具体优先级。
- fifo: 一个先进先出（FIFO）消息队列
- tss: 任务的状态，通过 TSS32 结构体定义。

## TASKLEVEL 结构体

某一特定优先级级别的就绪进程队列。

```

struct TASKLEVEL {
    int running; /*正在运行的任务数量*/
    int now; /*这个变量用来记录当前正在运行的是哪个任务*/
    struct TASK *tasks[MAX_TASKS_LV];
};

```

相关操作实现在 mtask.c 文件中

成员变量说明：

- running: 当前级别上正在运行的任务数量。
- now: 当前正在运行的任务的索引。
- tasks[MAX\_TASKS\_LV]: 一个数组，存储指向当前级别上所有任务的指针。

## TASKCTL 结构体

用于实现多级优先队列的数据结构，用于完成进程调度功能。

```

struct TASKCTL {
    int now_lv; /*现在活动中的LEVEL */
    char lv_change; /*在下次任务切换时是否需要改变LEVEL */
    struct TASKLEVEL level[MAX_TASKLEVELS];
    struct TASK tasks0[MAX_TASKS];
};

```

相关操作实现在 mtask.c 文件中

成员变量说明：

- now\_lv：当前活动的优先级级别。
- lv\_change：指示在下次任务切换时是否需要改变优先级级别。
- level[MAX\_TASKLEVELS]：每个元素是一个 TASKLEVEL 结构体，代表一个优先级级别。
- tasks0[MAX\_TASKS]：存储所有进程控制块的数组。

## FILEINFO 结构体

用于表示文件系统中文件基本信息的数据结构。

```

/* file.c */
struct FILEINFO {
    unsigned char name[8], ext[3], type;
    char reserve[10];
    unsigned short time, date, clustno;
    unsigned int size;
};

void file_readfat(int *fat, unsigned char *img);
void file_loadfile(int clustno, int size, char *buf, int *fat, char *img);
struct FILEINFO *file_search(char *name, struct FILEINFO *finfo, int max);

```

相关操作实现在 file.c 文件中

成员变量说明：

- name[8]：用于存储文件的名称。
- ext[3]：用于存储文件的扩展名。
- type：用于存储文件的类型或属性。
- reserve[10]：10 字节的保留字段
- time：表示文件的最后修改时间。
- date：表示文件的最后修改日期。
- clustno：表示文件在 FAT12 文件系统中的第一个簇的编号。
- size：表示文件的大小。

以上数据结构记录了内核程序所需要的关键信息，是内核程序中的重要变量。

# 运行测试

## 系统启动

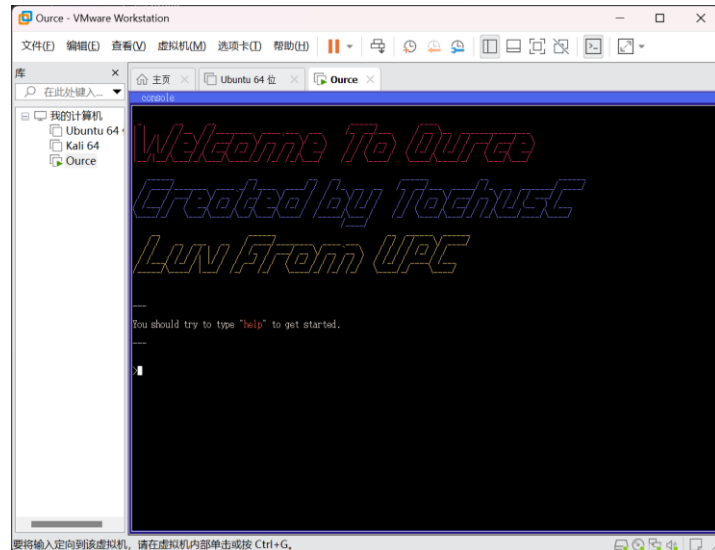


图 4 Ource 启动画面

如图 4 所示，Ource 成功启动并输出了欢迎信息及提示信息。

## 相关命令

输入 help 命令，查看帮助。

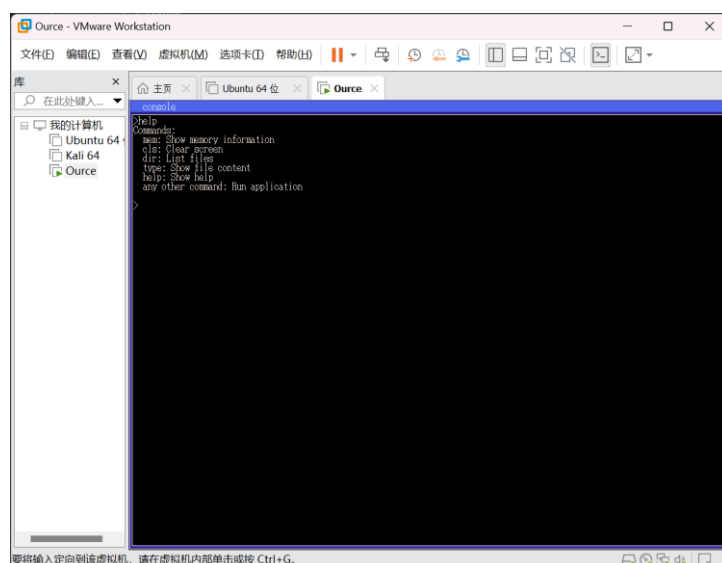


图 5 help 命令运行结果

如图 5 所示，输入 help 命令后，命令行中列出了支持的所有指令。

## 输入 mem 命令，查看系统内存使用情况

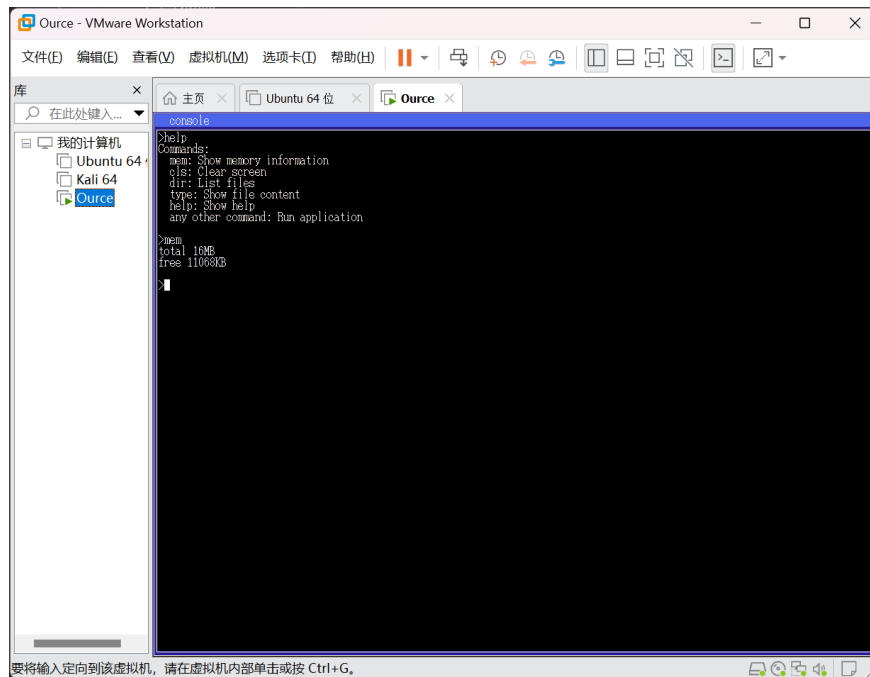


图 6 mem 命令运行结果

如图 6 所示，mem 命令输出了系统的所有内存和空闲内存情况。

## 输入 dir 命令，查看外存文件目录。

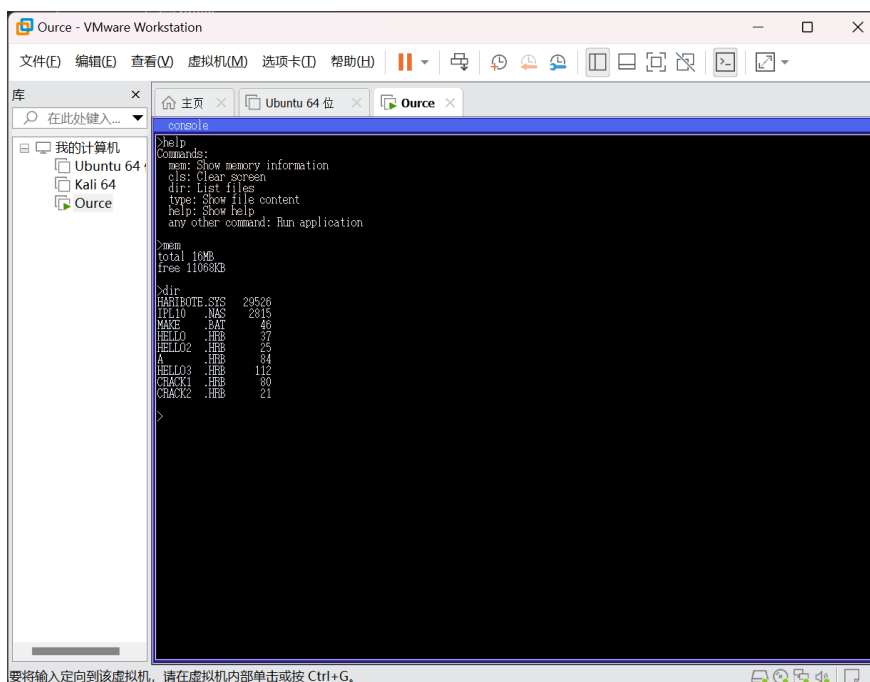


图 7 dir 命令运行结果

从图 7 可以看到，dir 命令列出了文件系统中所包含文件的基本信息。



## 通过 type 命令，查看某个文件的内容

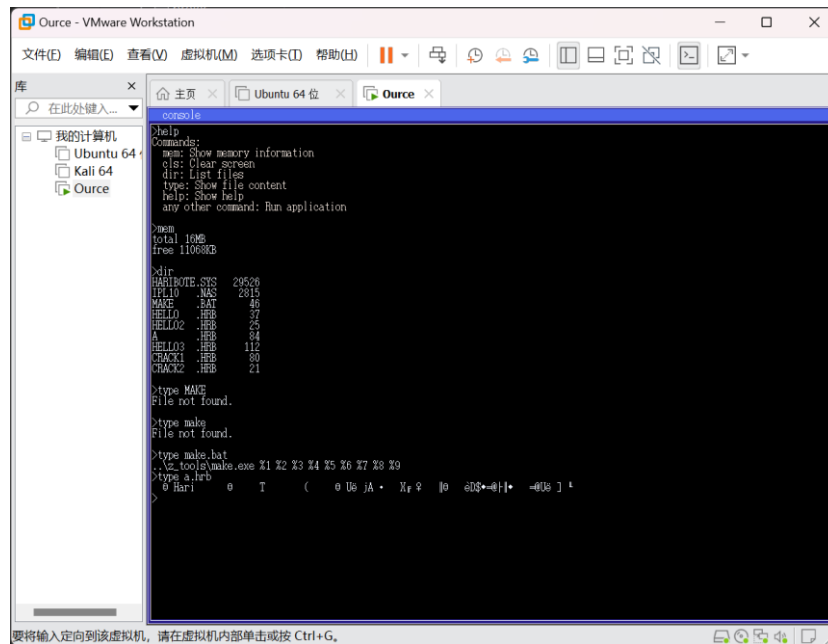


图 8 type 命令运行结果

如图 8 所示，type 命令输出了 make.bat 和 a.hrb 的内容，其中因为 a.hrb 为二进制格式的可执行文件，所以输出了乱码信息。

## 通过 cls 指令刷新屏幕

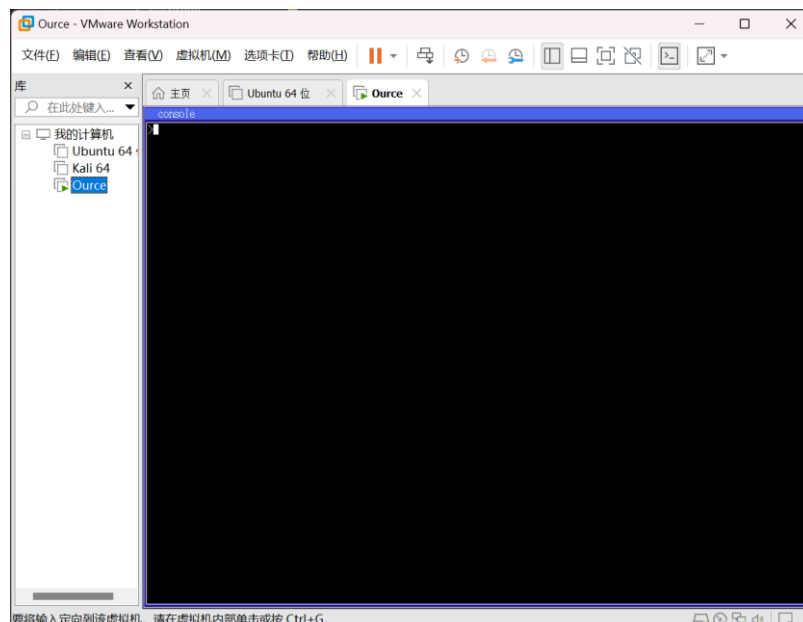


图 9 cls 命令运行结果

如图 9 所示，cls 指令刷新了命令行屏幕。



# 总结

以上就是 Ource OS 的总体说明了，感谢老师您拨冗评阅我的课程作业，毫无疑问，相比于一个真正的 OS 内核来说，现在的 Ource 完全是一个用于学习的“玩具”，但这也是我在有限的时间和精力下，所能做到的最好。

在一开始拿到“制作一个操作系统内核”的作业时，我可以说是完全不知道该从何下手，但在经过不断地在网上查询学习后，我也最终确定选择依据《30 天自制操作系统》这本书作为参考，编写并制作自己的操作系统。

虽说是 30 天，但我应该在 2 周不到的时间内，把它完完全全读了下来；在阅读完之后，我的第一感受是《30 天自制操作系统》这本书的内容还是太少、太不够充分了，作者为了“易读性”和“30 天内完成”的这两个目标，省去了很多内容的讲解，涉及的部分也不够深入。

很明显，想充分深入到操作系统的开发中，单凭这一本书的内容是远远不够的，学生我还仍需要再下功夫。

非常令我感到遗憾的一点是，相较于《30 天自制操作系统》中的 Haribote 系统，我所编写的 Ource 系统几乎没有什么核心功能上的变化，并且还删去了 Haribote 系统中的图形化界面部分。

最主要的原因是我认为在《30 天自制操作系统》中的图形化界面的实现代码并不是很好，书中添加图形化界面的主要目的是为了吸引读者阅读兴趣，图形接口的很多功能的实现都比较勉强且低效，因此我有意地删去了那部分代码。

但我可以很自信地说，对于 Ource 中目前涉及到的所有代码，我都已然理解掌握，并且可以在此基础上自如修改；据我现在能力和技能来说，我有信心自己能够开发出功能更齐全、更完善的操作系统：比如进一步完善进程调度方式，采用多级反馈优先队列；亦或是添加更多的系统调用，完善文件系统功能等等。这些都只是时间和精力上的问题。但在综合取舍下，我没有继续进行下去，这学期实在是有着太多需要忙碌的事情了……

经过这次 OS 内核的开发实践后，我进一步加深了自己对操作系统的理解，并在代码能力和项目能力上有了十足长进；最后，在收获满满的同时，我心中也洋溢着成功挑战 OS 内核开发后的自豪感。

感谢老师在本学期内的辛勤教学，也再次感谢老师您阅读至此，衷心地祝愿老师您工作顺利、生活美满。