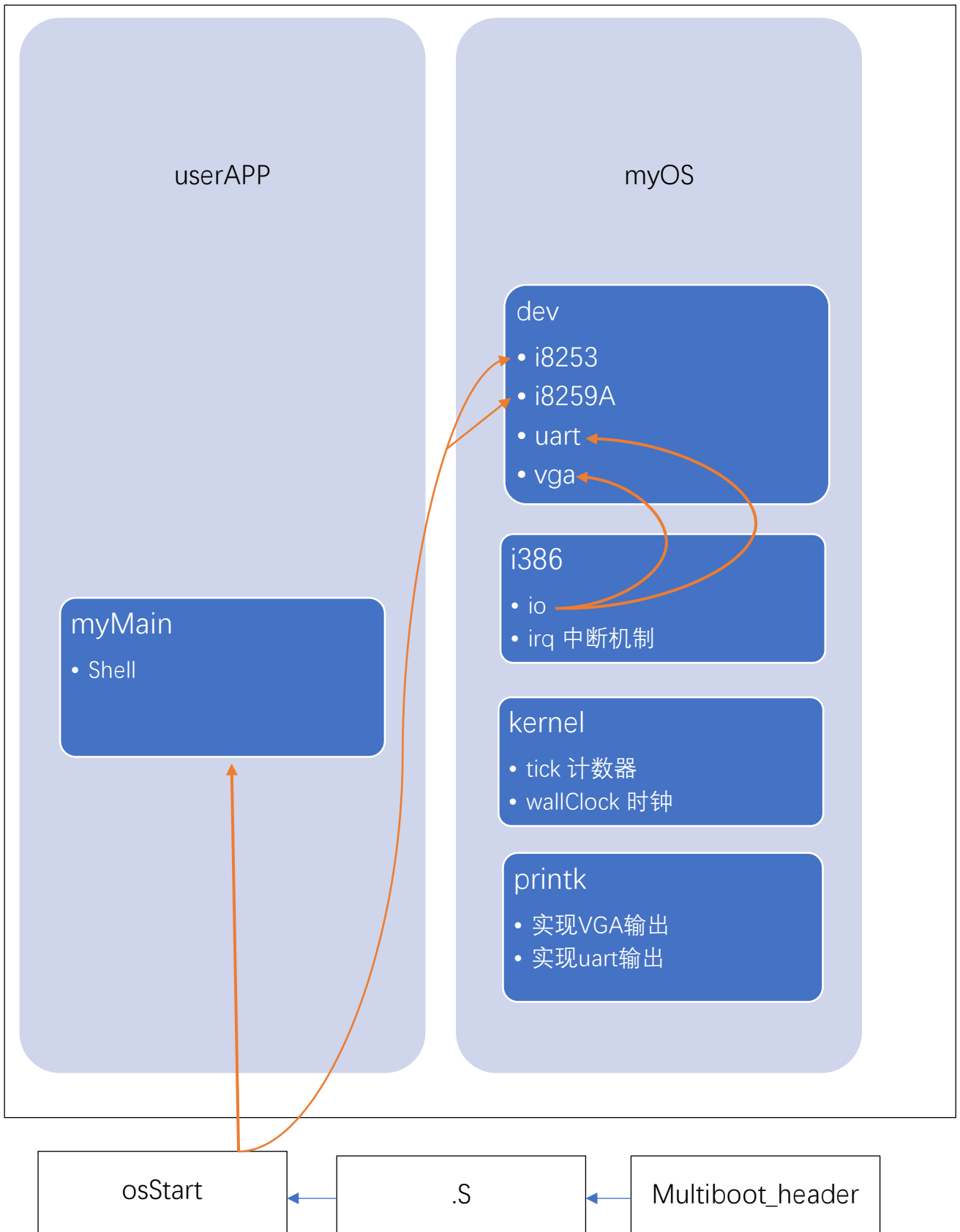


操作系统实验三报告

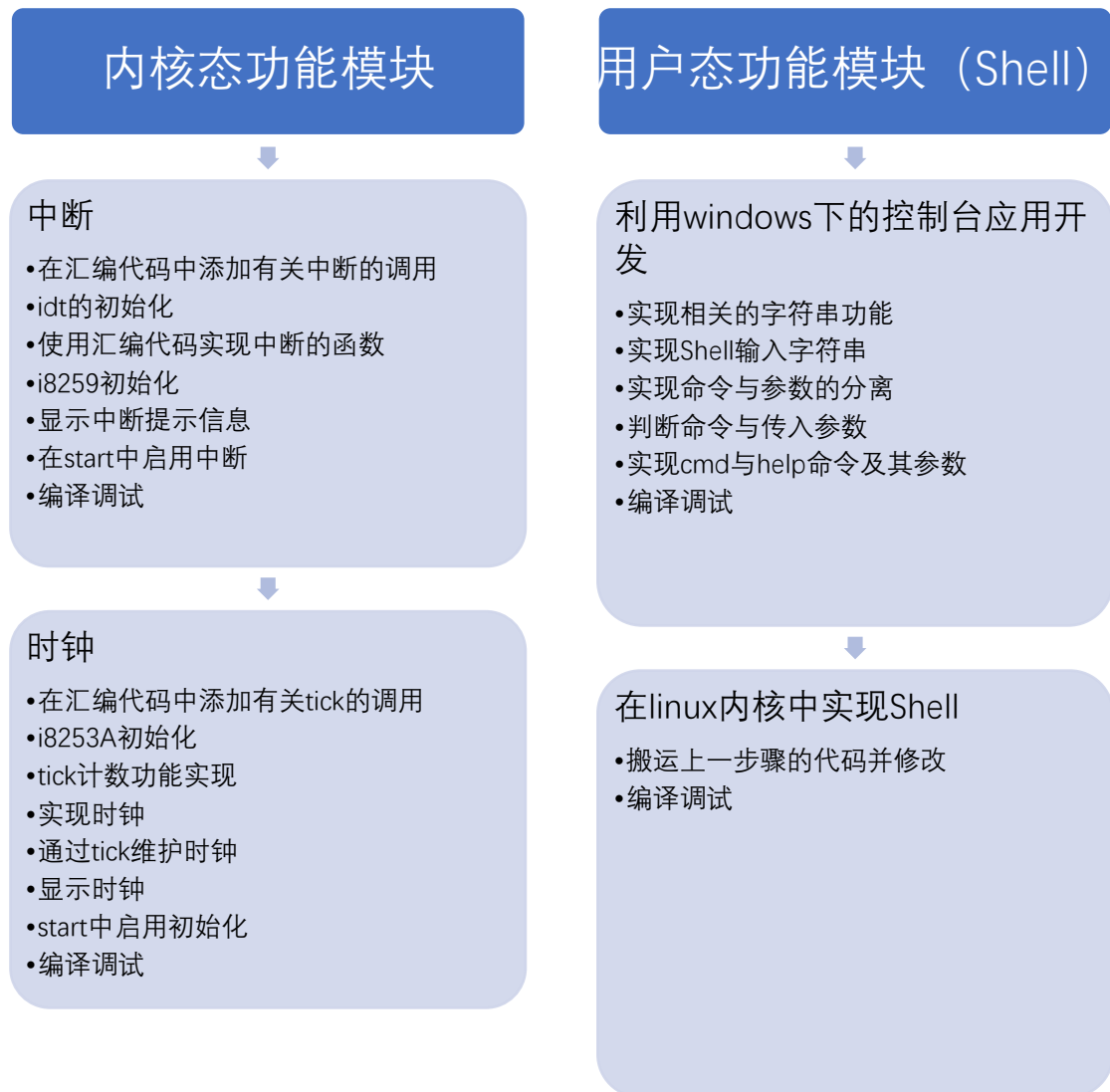
软件框图



主功能模型及其实现

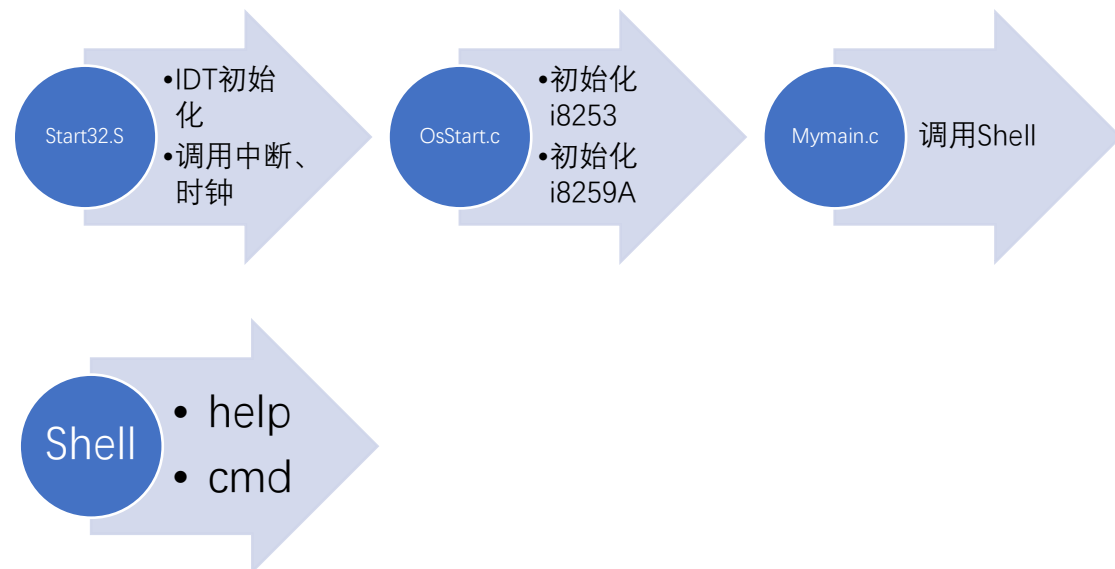
本次实验中将要实现的功能模块分为两部分,一部分为用户态,一部分为内核态。

实际操作中我先实现了内核态的中断与时钟,然后再实现 shell 的相关内容。



主要流程实现

在上次实验的基础上，执行的流程的增加了如下的内容。



源代码说明

在本次实验的根目录下，除了配置了 makefile 与 DS 文件，还有 4 个文件夹：multibooheader、myOS、output、userApp。multibooheader 中放置了有关 multibooheader 协议的 S 文件。在 output 中，原本为空文件夹，用来输出我们编译后生成的文件。在该文件夹中的目录结构与根目录相似，主要是为了对每个文件输出时不产生混淆。在 userApp 中存放的是 main.c 文件与该文件夹下的 makefile 文件，Main 文件是用来测试本次实验的功能的入口。在 myOS 文件夹中存放了本次实验的主要源文件。直接存放在此该目录下的文件有 DS 文件、该文件夹下的 makefile 文件、链接器文件、一个配置地址的 S 文件和一个包含了 main 函数的与 multibooheader 中的 S 文件相关联的 C 文件。这个文件也是从汇编到 C 的转变。在该目录下则有 5 个实现相关功能的文件夹：dev、i386、printk、kernel、lib。每个文件夹下都有一个相应的 makefile 文件以及实现功能的 C 语言

源文件。相比于上一次实验，i386 文件夹下增加了队中断处理的源文件，dev 下额外增加了对 i8253 和 i8259A 的初始化的源文件，而在 lib 中则添加关于此次实验中用到的一些文件，比如 String 和 Color，分别用于字符串的相关处理与颜色宏定义的归纳，kernel 文件夹中则存放了有关时钟和计数处理的源文件。每个子目录下的 makefile 文件的输出目录都是在 output 中的同名目录。相关源代码如图：

```
1     extern void outb(unsigned short int port_to, unsigned char value);
2
3     //253Counter0端口号
4     #define COUNTER0 (0x40)
5     //PC的频率
6     #define PC_FREQ (1193180L)
7     //8253模式控制寄存器
8     #define MODE_CTL (0x43)
9     //1秒钟发生100次时钟中断，即每隔0.01秒发生1次
10    #define MY_HZ (100)
11
12    void init8253(void)
13    {
14        outb(MODE_CTL, 0x34);
15        //写入分频参数
16        outb(COUNTER0, (unsigned char)(PC_FREQ / MY_HZ));
17        outb(COUNTER0, (unsigned char)((PC_FREQ / MY_HZ) >> 8));
18    }
```

初始化 i8253

```
#include "../lib/color.h"
extern void append2screenAtpoint(int line, int column, char* str, int color);

void ignoreIntBody()
{
    append2screenAtpoint(24, 0, "Unknown interrupt1\0", RED);
}
```

显示异常中断信息

```

1  extern unsigned char inb(unsigned short int port_from);
2  extern void outb(unsigned short int port_to, unsigned char value)
3
4  //8259A主片的控制端口号1
5  #define INT_MASTER_CTL (0x20)
6  //8259A从片的控制端口号1
7  #define INT_SLAVE_CTL (0xa0)
8  //8259A主片的控制端口号2
9  #define INT_MASTER_CTLMSK (0x21)
10 //8259A从片的控制端口号2
11 #define INT_SLAVE_CTLMSK (0xa1)
12 //外中断的个数
13 #define IRQ_NUM (16)
14
15 void init8259A(void)
16 {
17     /* 8259A 主片和从片, OCW1, 起屏蔽外中断作用 */
18     outb(INT_MASTER_CTLMSK, 0xff);
19     outb(INT_SLAVE_CTLMSK, 0xff);
20     /* 8259A 主片和从片, ICW1 */
21     outb(INT_MASTER_CTL, 0x11);
22     outb(INT_SLAVE_CTL, 0x11);
23     /* 8259A 主片和从片, ICW2 */
24     outb(INT_MASTER_CTLMSK, 0x20);
25     outb(INT_SLAVE_CTLMSK, 0x28);
26     /* 8259A 主片和从片, ICW3 */
27     outb(INT_MASTER_CTLMSK, 4);
28     outb(INT_SLAVE_CTLMSK, 2);
29     /* 8259A 主片和从片, ICW4 */
30     outb(INT_MASTER_CTLMSK, 3);
31     outb(INT_SLAVE_CTLMSK, 1);
32 }

```

初始化 i8259A

```

1  #include "../kernel/wallClock.h"
2
3  static unsigned long int TickCount = 0;
4
5  //获取计数
6  int getTickCount()
7  {
8      return TickCount;
9  }
10
11 //每秒被调用100次的技术函数
12 void tick(void)
13 {
14     TickCount += 1;
15     //达到100次则更新时钟
16     if (TickCount >= 100)
17     {
18         int h, m, s;
19         getWallClock(&h, &m, &s);
20         setWallClock(h, m, s + (TickCount / 100));
21         TickCount %= 100;
22     }
23     //更新时钟毫秒
24     setms(TickCount * 10);
25 }
26
27

```

tick

```

7 //采用hook机制处理时钟设置
3 void setWallClockHook(void (*func)(void))
3 {
0     (*func)();
1 }
2
3 //更新时钟的显示
4 void UpdateWallClock(void)
5 {
6     char buf[9];
7     myvsprintf(buf, "%t", hh, mm, ss);
8     append2screenAtpoint(24, 71, buf, YELLOW);
9 }
0
1 //设置时钟
2 void setWallClock(int h, int m, int s)
3 {
4     //进位处理
5     ss = s % 60;
6     m += s / 60;
7     mm = m % 60;
8     h += m / 60;
9     h = h % 24;
0     setWallClockHook(UpdateWallClock);
1 }
2
3 //设置毫秒
4 void setms(int _ms)
5 {
6     ms = _ms;
7 }
8
9 //获取时钟
0 void getWallClock(int* h, int* m, int* s)
1 {
2     *h = hh;
3     *m = mm;
4     *s = ss;
5 }

```

时钟

```

1 #define DARK_BLUE 1 //蓝色
2 #define DARK_GREEN 2 //深绿
3 #define BLUE 3 //蓝
4 #define RED 4 //红
5 #define PURPLE 5 //紫
6 #define BROWN 6 //棕
7 #define GREY 7 //灰
8 #define BLACK 8 //黑
9 #define CYAN 9 //青
10 #define GREEN 10 //绿
11 #define LIGHT_BLUE 11 //浅蓝
12 #define ORANGE 12 //橙
13 #define PINK 13 //粉
14 #define YELLOW 14 //黄
15 #define WHITE 15 //白
16

```

颜色代码

//指定坐标的VGA输出格式化字符串

```
void append2screenAtpoint(int line, int column, char* str, int color)
{
    int i = 0;
    //判断字符串的结尾
    while (*str != '\0')
    {
        VGAputcharAtpoint(line, column + i, *str, color);
        str++;
        i++;
    }
}
```

//指定坐标的VGA输出字符

```
void VGAputcharAtpoint(int line, int column, unsigned char c, int color)
{
    //坐标校正
    if (column > ColumnLenth)
    {
        line += column / LineLenth;
        column %= LineLenth;
    }
    if (line > LineLenth) line = LineLenth;
    //计算光标位置
    unsigned int pos = line * LineLenth + column;
    //写入字符以及颜色
    *(char*)(VGA_BASE + pos * 2) = c;
    *(char*)(VGA_BASE + pos * 2 + 1) = color;
}
```

VGA 增添功能

```
1 //取字符串长
2 +int strlen(char* str) { ... }
8
9 //字符串拷贝
10 +int strcpy(char* src, char* dst) { ... }
20
21 //字符串比较
22 +int strcmp(char* str1, char* str2) { ... }
34
35 //判断字符是否为换输信号
36 +int ischareempty(char c) { ... }
40
41 //判断字符是否为确认输入信号
42 +int ischarendline(char c) { ... }
```

字符串处理函数，此处忽略展开内容

在 UserApp 中，本次实验主要增加了 Shell 源文件用于实现有关命令行的功能模块。以下是 Shell 的代码：

```
1  #include "../myOS/lib/color.h"
2  #include "../myOS/dev/uart.h"
3  #include "../myOS/lib/string.h"
4  #include "../myOS/printk/myPrintk.h"
5  #define NULL 0
6
7  extern void append2screen(char* str, int color);
8
9  // 命令处理函数
10 int cmd_handler(int, char**);
11 int help_handler(int, char**);
12
13 // 帮助处理函数
14 void help_help(void);
15
16 struct command
17 {
18     char* cmd;
19     int (*func)(int argc, char** argv);
20     void (*help_func)(void);
21     const char* desc;
22 } cmds[] = {
23     {"cmd", cmd_handler, NULL, "list all commands"},
24     {"help", help_handler, help_help, "help [cmd]"},
25     {"", NULL, NULL, ""}
26 };
27
28
29 // help 的帮助
30 void help_help(void)
31 {
32     myPrintk(WHITE, "using \"help cmd_name\" to get help of the command\n");
33 }
34
```

Shell 代码


```

35 // help 命令处理函数
36 int help_handler(int argc, char** argv)
37 {
38     if (strcmp(*argv, "") == 0)
39     {
40         help_help();
41     }
42     //查找命令
43     for (int i = 0;; i++)
44     {
45         //查找失败
46         if (strcmp("", cmds[i].cmd) == 0)
47         {
48             myPrintk(WHITE, "help does not know the command:%s\n", *argv);
49             break;
50         }
51         //查找成功
52         if (strcmp(*argv, cmds[i].cmd) == 0)
53         {
54             //描述对应命令
55             myPrintk(WHITE, "%s:%s\n", cmds[i].cmd, cmds[i].desc);
56             //执行对应命令的help函数
57             if (cmds[i].help_func)
58                 (*cmds[i].help_func)();
59             break;
60         }
61     }
62     return 0;
63 }

```

Shell 代码

```

65 // cmd 命令处理函数
66 int cmd_handler(int argc, char** argv)
67 {
68     myPrintk(WHITE, "List all registered commands:\ncommand name:description\n");
69     //遍历命令并输出信息
70     int i = 0;
71     while (strcmp(cmds[i].cmd, "") != 0)
72     {
73         myPrintk(WHITE, "%s:%s\n", cmds[i].cmd, cmds[i].desc);
74         i++;
75     }
76     return 0;
77 }
78

```

Shell 代码

```

8
9  int startShell()
0  {
1      myPrintk(LIGHT_BLUE, "Waiting for a command >:");
2      char line[100] = { 0 };
3      char cmdline[20] = { 0 };
4      char arrayline[80] = { 0 };
5      int i = 0;
6      char c;
7      //命令输入
8      while (1)
9      {
0          //获取命令
1          c = uart_get_char();
2          uart_put_char(c);
3          //持续输入
4          line[i] = c;
5          i++;
6          //终止输入
7          if (ischarendline(c) == 1)
8              break;
9          //超出输入长度限制
0          if (i == 99)
1          {
2              myPrintk(WHITE, "Warning:command too long.\n");
3              return 0;
4          }
5      }

```

Shell 代码

```

106 //打印一遍输入的命令
107 myPrintk(WHITE, line);
108 int k = 0;
109 //分离参数
110 for (; ischarempy(line[k]) == 0 && k < 20; k++)
111 {
112     cmdline[k] = line[k];
113 }
114 cmdline[k] = '\0';
115 for (; ischarempy(line[k]) != 0 && k <= i; k++);
116 if (k < i)
117     for (int j = 0; isharendline(line[k + j]) == 0 && j < 80; j++)
118     {
119         arrayline[j] = line[k + j];
120     }
121 //命令判断
122 for (i = 0;; i++)
123 {
124     //查找失败
125     if (strcmp("\0", cmds[i].cmd) == 0)
126     {
127         myPrintk(WHITE, "\nUNKNOWN command:%s\n", line);
128         break;
129     }
130     //查找成功
131     if (strcmp(cmdline, cmds[i].cmd) == 0)
132     {
133         //UART修正输出
134         uart_put_char('\n');
135         //执行命令
136         myPrintk(WHITE, "%s", line);
137         char* arrayc = arrayline;
138         (*cmds[i].func)(0, (char**)&arrayc);
139         break;
140     }
141 }
142 }

```

Shell 代码

代码布局说明

所有的引导模块将按页（4KB）边界对齐，物理内存地址从 1M 处开始

编译过程说明：

在 Ubuntu 中先搜索到 lab3 的目录，然后通过指令 make 完成编译，可以看到

在 output 目录中的对应目录中分别输出了与根目录下对应文件相同文件。

```
ws@LAPTOP-J9NGC786: ~$ cd /mnt/d/360MoveData/Users/asus/Desktop/os2020-labs/lab3/src/
ws@LAPTOP-J9NGC786: /mnt/d/360MoveData/Users/asus/Desktop/os2020-labs/lab3/src$ export DISPLAY=:0
ws@LAPTOP-J9NGC786: /mnt/d/360MoveData/Users/asus/Desktop/os2020-labs/lab3/src$ make
ld -n -T myOS/myOS.ld output/multibootheader/multibootHeader.o output/myOS/start32.o output/myOS/osStart.o output/myOS/dev/uart.o output/myOS/dev/vga.o output/myOS/dev/i8253.o output/myOS/dev/i8259A.o output/myOS/i386/io.o output/myOS/i386/irq.o output/myOS/i386/irqs.o output/myOS/printk/myPrintk.o output/myOS/printk/vsprintf.o output/myOS/lib/string.o output/myOS/kernel/tick.o output/myOS/kernel/wallClock.o output/userApp/main.o output/userApp/shell.o -o output/myOS.elf
```

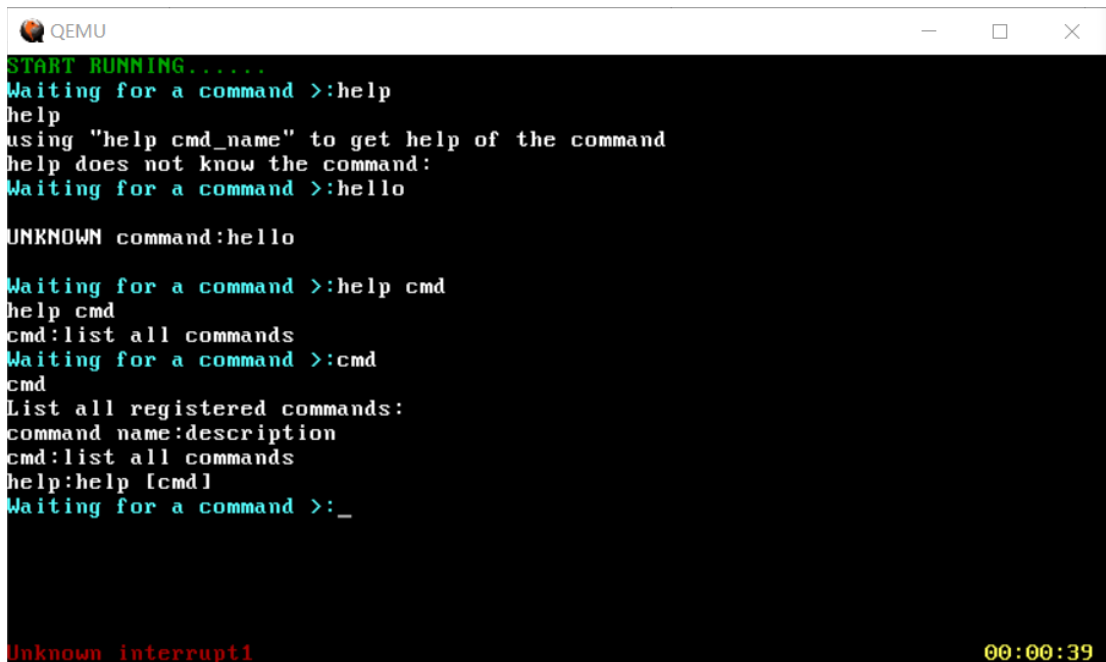
> os2020-labs > lab3 > src > output

名称	修改日期	类型
multibootheader	2020/4/5 17:43	文件夹
myOS	2020/4/5 17:43	文件夹
userApp	2020/4/5 17:43	文件夹
myOS.elf	2020/4/5 17:43	ELF 文件

可以看到，make 命令确保源代码目录下没有不正确的.o 文件以及文件的互相依赖。它们分别链向源代码目录下的真正的 i386 所需要的真正的子目录。编译后产生的文件在图上可见，有 multiboot、start32、osStart、uart、vga、i8259A、i8253、tick、shell、wallClock、string、io、myPrintk、vsprintf、main、myOS 的输出文件以及标注了他们的输出目录。

运行和运行结果说明：

在 Ubuntu 中通过 QEMU 启动已经编译生成的 bin 文件或者直接运行 SH 文件，得到 Linux 的图形化界面运行结果，显示需要的输出。然后在使用 screen 命令打开得到伪终端窗口，如下是输入与输出。



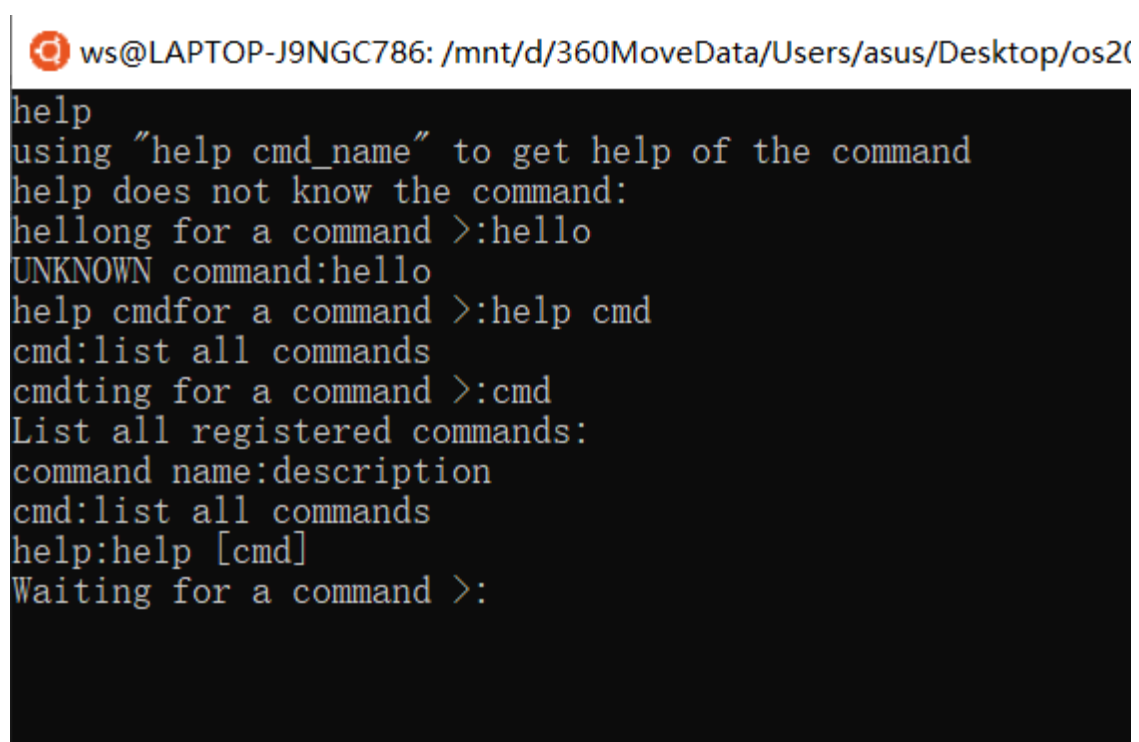
```
QEMU
START RUNNING.....
Waiting for a command >:help
help
using "help cmd_name" to get help of the command
help does not know the command:
Waiting for a command >:hello

UNKNOWN command:hello

Waiting for a command >:help cmd
help cmd
cmd:list all commands
Waiting for a command >:cmd
cmd
List all registered commands:
command name:description
cmd:list all commands
help:help [cmd]
Waiting for a command >:_

Unknown interrupt1 00:00:39
```

VGA 输出中可以看到左下角的中断提示语右下角的时钟显示。本实验我以 00: 00: 00 作为时钟的起始，而引发非时钟中断的方式为按下任意键。



```
ws@LAPTOP-J9NGC786: /mnt/d/360MoveData/Users/asus/Desktop/os2(
help
using "help cmd_name" to get help of the command
help does not know the command:
hellong for a command >:hello
UNKNOWN command:hello
help cmdfor a command >:help cmd
cmd:list all commands
cmdting for a command >:cmd
List all registered commands:
command name:description
cmd:list all commands
help:help [cmd]
Waiting for a command >:
```

在串口中的输出结果，本次实验依次演示了 help、非注册命令、help[cmd]、cmd 的输出。

遇到的问题和解决方案：

1. 不会自己 IDT 的初始化

查阅网络资料，参考老师的代码

2. VGA 显示时钟时可能会因为换行而导致显示异常

修改 VGA 代码，将最大行数改为 23

3. Shell 功能模块的实现难以直接在内核中编写和调试

利用 Windows 的控制台应用实现交互功能与需要的字符串处理函数，再修改个别函数名称进行移植。