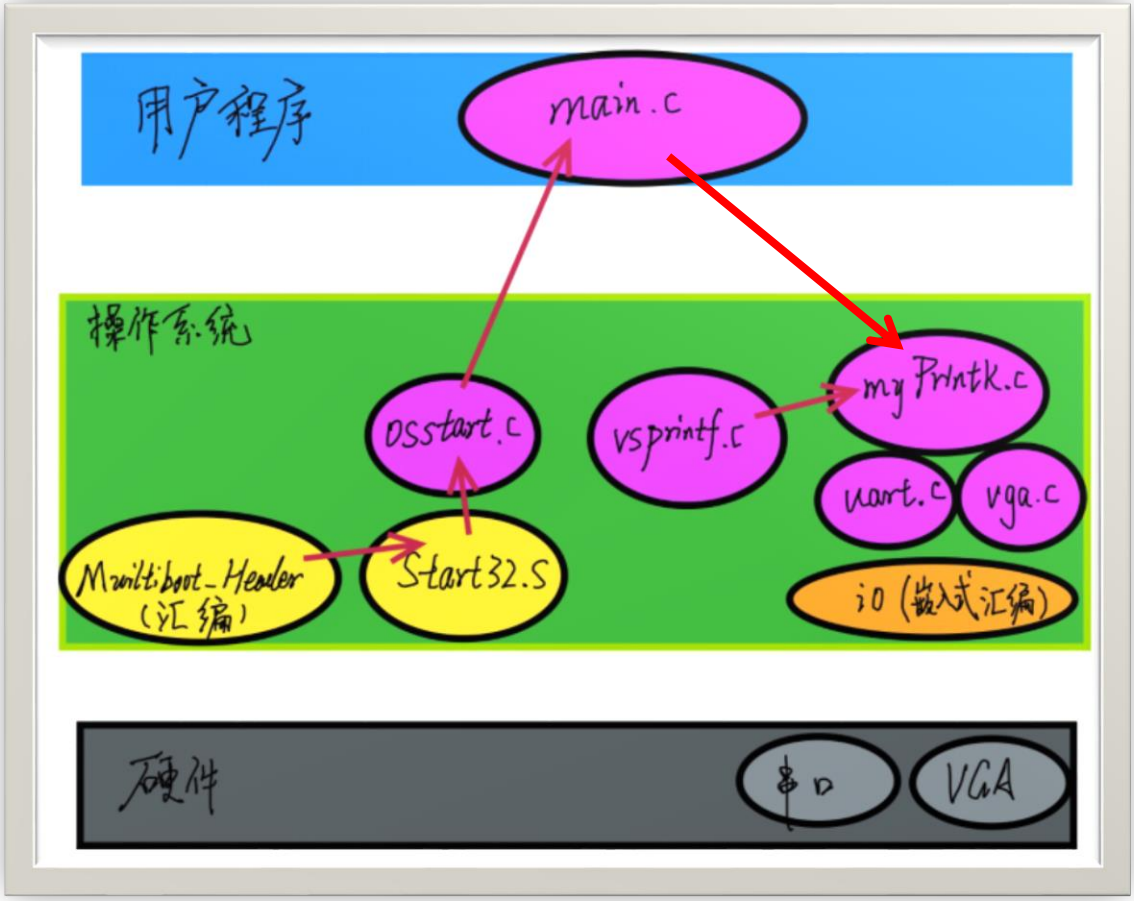
操作系统实验报告

实验二　Multiboot2myMain

学号：PB18111683
姓名：童俊雄
完成时间：2020-03-21

一、　软件框图

****注：图中的header不属于操作系统，只是囿于画幅才画在里面了**
紫色为C程序，黄色为汇编程序，橙色为嵌入式汇编

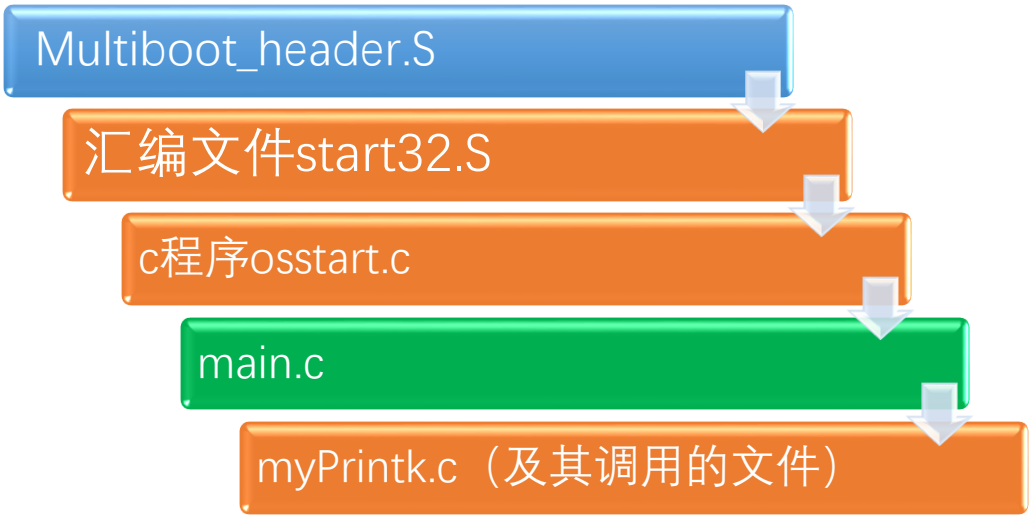


概述：软件大体上可以分为两个层次：用户程序和操作系统，其中操作系统又可以分为与用户程序的接口（左半部分）和IO设备的驱动程序（右半部分）两块，在每块中又可以进一步划分层次（越上方的层次越高），如图所示。

二、 主流程说明

1. qemu启动header；
2. header通过调用myOS提供的_start，跳转到汇编文件start32.S；
   （进入myOS）
3. 从start32调用C语言入口，进入到c程序osstart.c；
4. 通过osstart.c对myOS进行初始化，并调用用户程序的接口，从而执行用户程序main.c；
   （进入UserApp）
5. main.c调用myprintk.c中的函数实现IO;

流程图：
（橙色表示myOS内的程序，绿色表示用户程序）



三、 主要功能模块说明&源代码说明

1. IO（IO.c）

采用嵌入式汇编，用outb指令实现端口输出，源代码为：

```
/* IO operations */
unsigned char inb(unsigned short int port_from){
    //参考下面的outb函数，实现inb函数
    unsigned char value;
    __asm__ __volatile__("inb %w1,%0":"=a" (value) : "Nd" (port_from))
    return value;
}

void outb (unsigned short int port_to, unsigned char value){
    __asm__ __volatile__ ("outb %b0,%w1"::"a" (value),"Nd" (port_to));
}
```

2. **串口输出(uart.c)**

调用IO中的outb和inb实现，单个字符输入/出时直接使用串口地址作为port_from/to：

（需要改\n的环境中：）

```
void uart_put_char(unsigned char c){
    if (c != '\n')
        outb(uart_base, c);
    else                                //change line
        outb(uart_base, '\r');
}
```

（无需更改\n的环境中：）

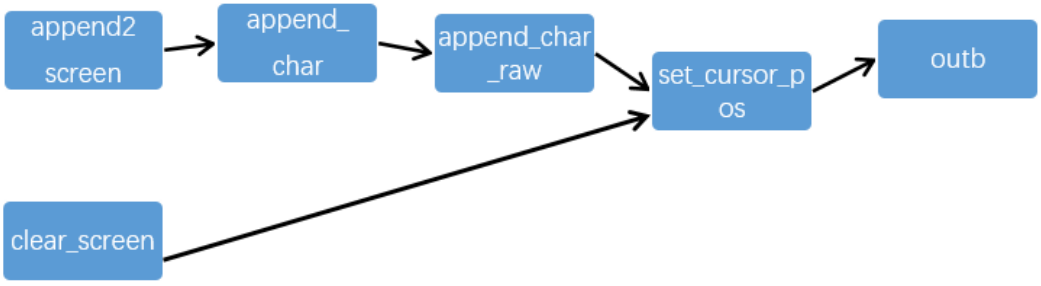
```
void uart_put_char(unsigned char c){

        outb(uart_base, c);

}
```

3. **VGA输出(vga.c)**

采用嵌入式汇编直接写显存。append_char_raw函数中，调用set_cursor_pos函数(在其中调用IO中的outb)用xy两个坐标对光标进行定位，然后通过嵌入式汇编的movl把字符写入这一位置对应的显存地址当中，并将光标移动到字符后；append2screen则调用append_char函数，并在append_char函数中唤起append_char_raw函数逐一输出。调用流程如下：



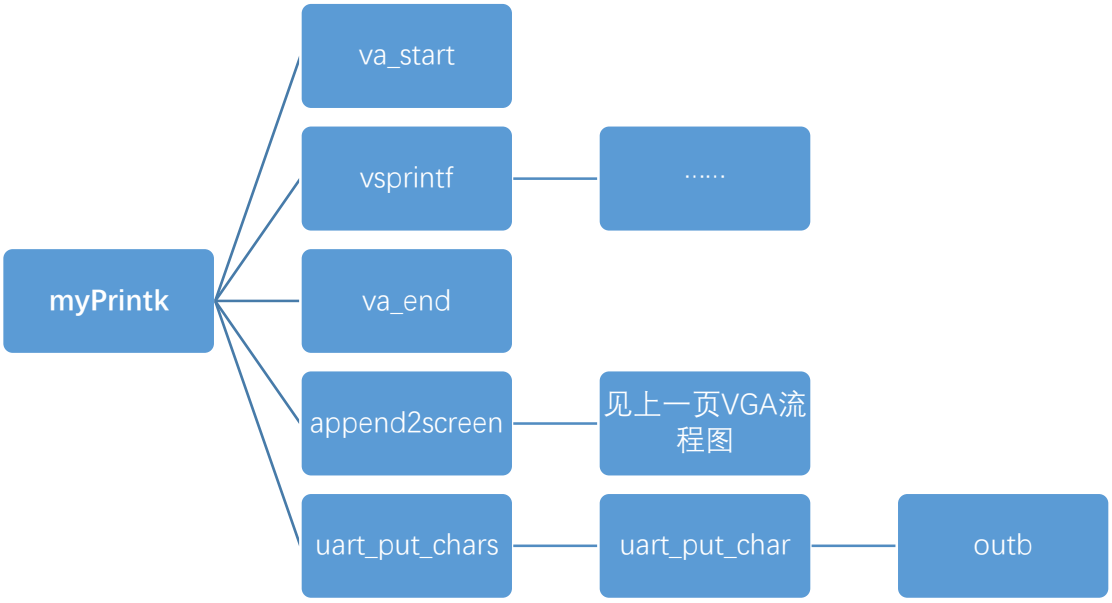
各个函数的具体实现参见代码注释，此处不加赘述。

4. **myPrintk（myPrintk.c）**

myprintk/f调用头文件<stdarg.h>中的va_start处理可变参数，再用调用vsprintf函数，对字符串进行格式化并存入buf数组中，随后调用va_end释放arg，再用uart_put_chars函数和append2screen函数，从VGA和串口输出，代码如下：

```
int myPrintk(int color, const char* format, ...) {
    va_list args;
    int i;
    va_start(args, format);             //可变参数处理
    i = vsprintf(kbuf, format, args);   //将格式字符串写入到kbuf
    va_end(args);                       //释放args
    append2screen(kbuf, color);
    uart_put_chars(kbuf);
    return i;
}
```

流程图为：



四、 目录组织

所有文件：

```
.
├── Makefile
├── multibootheader
│   └── multibootHeader.S
├── myOS
│   ├── Makefile
│   ├── dev
│   │   ├── Makefile
│   │   ├── uart.c
│   │   └── vga.c
│   ├── i386
│   │   ├── Makefile
│   │   ├── io.c
│   │   └── io.h
│   ├── myOS.ld
│   ├── osStart.c
│   ├── printk
│   │   ├── Makefile
│   │   ├── myPrintk.c
│   │   └── vsprintf.c
│   └── start32.S
├── source2run.sh
└── userApp
    ├── Makefile
    └── main.c
```

Makefile组织：

```
____src
  |____Makefile
  |____userApp
  | |____Makefile
  |____myOS
  | |____Makefile
  | |____i386
  | | |____Makefile
  | |____dev
  | | |____Makefile
  | |____printk
  | | |____Makefile
```

五、 代码布局

由myOS.ld的代码可知，myOS.elf文件中有三个 section：

1. 第一个section为.text，位置从1M处开始，在.text内的分布为8字节对齐，前12字节为魔术，从第16字节开始是代码部分；
   代码结束后16位对齐；
2. 第二个section为.data，位置从.text结束并对齐后开始；
   末尾16位对齐；
3. 第三个section为.bss，位置从.data结末尾对齐后开始；
   末尾16位对齐；
4. .bss结束后是_end，此处是堆空间的开始，512位对齐；

六、 编译过程说明

由makefile可知，编译过程有以下两步：

1. 编译汇编代码（header.S和start32.S）和C代码（osstart.c等）生成.o文件；
2. 根据myOS.ld的部署要求，把上述.o文件链接成myOS.elf文件
   如下图所示：

```
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/i386/io.o output/myOS/printk/myPrintk.o output/myOS/printk/vsprintf.o output/userApp/main.o -o output/myOS.elf
```

七、 运行和运行结果说明

运行指令qemu-system-i386 -kernel output/myOS.elf -serial stdio
运行结果如下如所示：

1. VGA输出：

2. 串口输出：

```
jin@DESKTOP-7SF7V9B:~/lab2$ qemu-system-i386 -kernel output/myOS.elf -serial stdio
START RUNNING......
main
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
STOP RUNNING......ShutDown
```

八、 遇到的问题和解决办法

问题：发现目录与助教所给不一致

解决：联系助教，发现所移植的vsprintf不一致；

问题：\n后面显示为蓝色

解决：将\n的color改为0

这次在写vga部分遇到了非常非常多的各种各样的问题，在郭同学和汪同学的指导下才解决了大部分的问题。