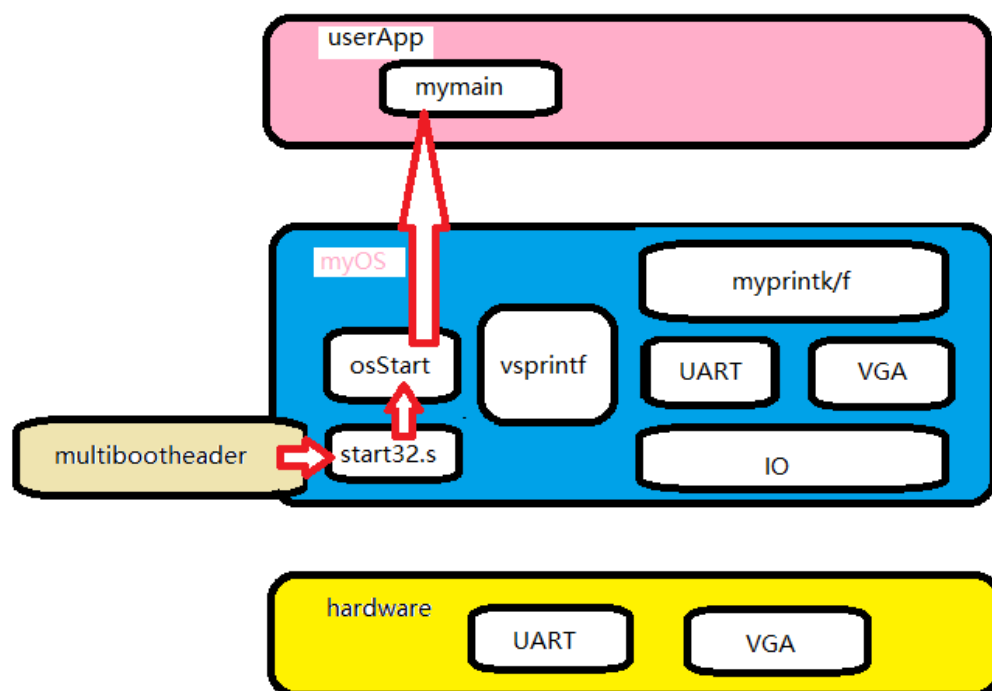


# Lab2 实验报告

PB18051098 徐碧涵

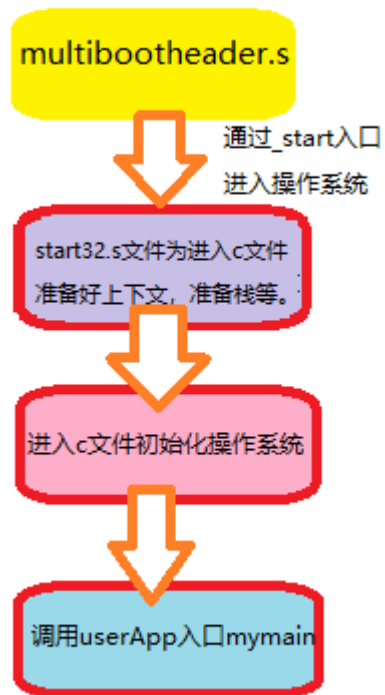
## 软件框图



该软件通过调用 myOS 的功能模块从而实现 userApp 的功能。本实验旨在调用 myOS 的功能模块实现端口输入输出。

## 主流程及其实现

主流程如下：



通过遵循 multiboot 启动协议的 multibootheader.s 文件中的 multiboot\_header 头部启动内核，利用 \_start 入口进入操作系统内部，执行 start32.s 文件，在其中为进入 c 文件准备好上下文，随后进入 c 文件对操作系统进行初始化，随后调用 userApp 入口进行测试。

## 主要功能模块及其实现

### 主要功能模块

#### io 端口输入输出（嵌入式汇编）

通过端口地址向端口进行输入输出，源代码如下：

```
unsigned char inb(unsigned short int port_from){
    unsigned char _in_value;
    __asm__ __volatile__ ("inb %w1,%0"::"a"(_in_value):"Nd"(port_from));
    return _in_value;
}

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

#### uart 输入输出

串口地址为 0x3F8，利用 inb(),outb()端口输入输出函数向串口端口地址写入或读取字符以及字符串实现串口输入输出，并且输出时对换行符进行正确处理。

##### uart 输出字符

```
void uart_put_char(unsigned char c){
    outb(0x3F8,c);    //向串口输出字符
}
```

**uart 输入字符**

```
unsigned char uart_get_char(void){
    unsigned char c;
    c=inb(uart_base);    //读入字符
    return c;
}
```

**uart 输出字符串并正确处理换行符**

```
void uart_put_chars(char *str){    //串口输出字符串
    int i=0;
    while(str[i]){
        if(str[i]=='\n'){
            i++;
            uart_put_char(0x0d);
            uart_put_char(0x0a);    //对换行进行处理，输出\r\n
            continue;}
        outb(uart_base,str[i]);    //输出字符
        i++;
    }
}
```

**vga 输出（实现清屏，滚屏，彩色输出）**

利用指针指向 vga 显存首地址从而实现 vga 输出，

```
unsigned char* vgabuffer=(unsigned char*)0x000B8000;
```

且 vga 输出的字符由两个字节构成，前一个为字符本身，第二个表示背景以及字体颜色。

设置 pos 表示当前输出位置，已知 vga 屏幕尺寸为 25\*80.

**清屏**

通过输出黑底无字实现清屏 clear\_screen()

```
int pos=0;    //pos指示当前屏幕输出位置，vga尺寸为25*80
```

**//清屏，即输出黑底无字**

```
void clear_screen(void){
    int i,j;
    for(i=0;i<25;i++){
        for(j=0;j<80;j++){
            pos=i*80+j;
            vgabuffer[pos*2+1]=0x9;
            vgabuffer[pos*2]=' ';//显示黑底无字
        }
        pos=0;
    }
}
```

**光标定位**

通过 inb(),outb()函数正确设置光标位置,向索引端口写入保存光标位置的寄存器编号接着对光标位置进行修改实现光标定位, x、y 表示要设置的坐标值。

```
void setcursor(int x,int y){
    int _pos=x*80+y;
    outb(0x3D4,0xE);           //通过向寄存器14, 15写入值设置光标位置
    outb(0x3D5,(_pos>>8)&0xff); //向寄存器14写入地址高八位
    outb(0x3D4,0xF);
    outb(0x3D5,_pos&0xff);      //向寄存器15写入地址低八位
}
```

#### 实现 append2screen()

对换行符进行处理,当遇到换行符时 pos 指向下一行。

```
if(str[i]=='\n'||str[i]=='\r'){ //对换行符进行处理
    pos=pos+80-pos%80;        //pos移向下一行
    i++;
}
```

正常字符即进行如下处理输出带颜色的字符

```
vgabuffer[2*pos+1]=(unsigned char)color;
```

```
vgabuffer[2*pos]=str[i];
```

当屏满时进行滚屏,执行如下代码

```
if(pos>=2000){ //实现滚屏
    pos-=80;
    for(j=0;j<3840;j++) //所有元素上移一行
        vgabuffer[j]=vgabuffer[j+160];
    for(j=0;j<80;j++){ //将最后一行清除
        vgabuffer[2*(pos+j)]=' ';
        vgabuffer[2*(pos+j)+1]=0x9;
    }
}
```

最后通过执行 setcursor(pos/80,pos%80);实现光标定位。

通过移植 vsprintf()函数处理格式化字符串。

#### myprintf/f

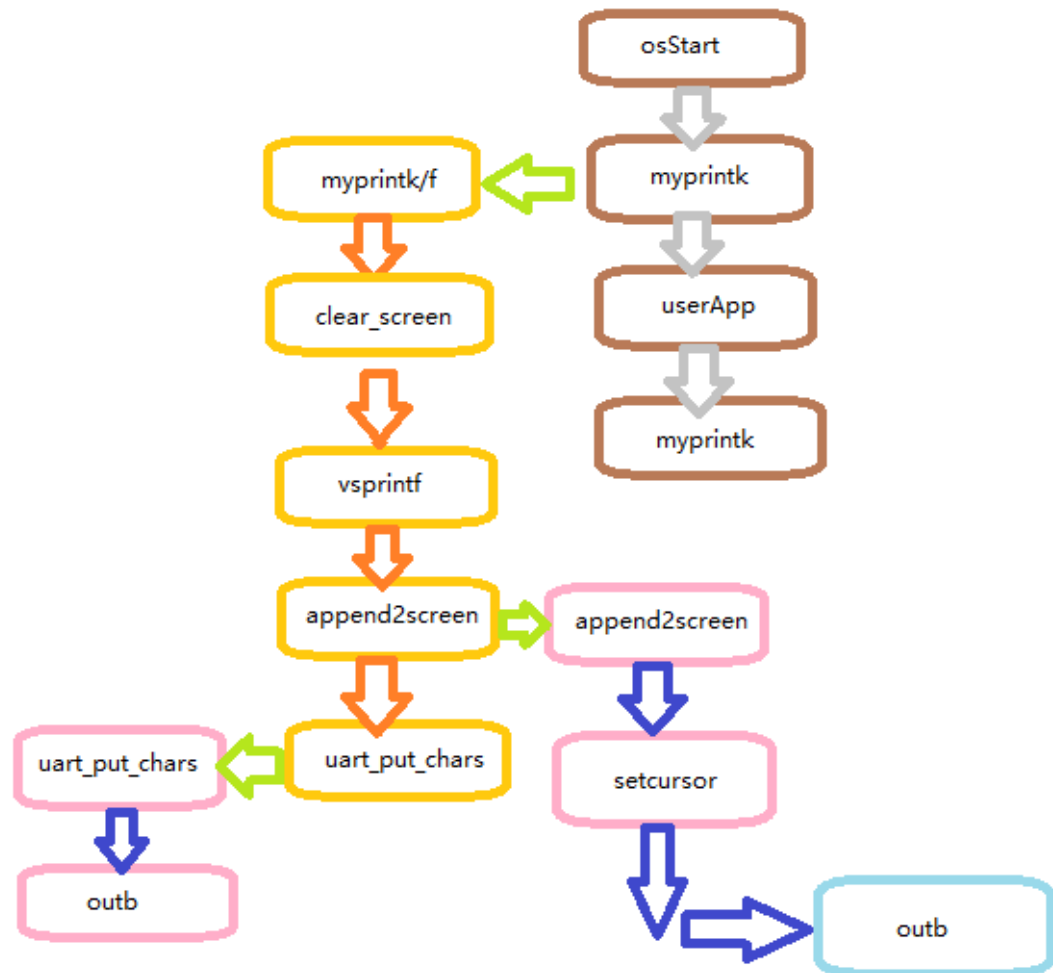
对多参数列表进行处理,再通过 vsprintf()函数处理格式化字符串并写入字符串数组中,再调用 uart\_put\_chars()以及 append2screen()通过串口以及 vga 端口输出字符串。代码如下:

```

int myPrintk(int color,char*format,...){
    int n;
    va_list args;
    va_start(args,format);
    n=vsprintf(kBuf,format,args);
    va_end(args);
    uart_put_chars(kBuf);
    append2screen(kBuf,color);
    return n;
}

```

流程图如下：



## Makefile 组织

基本上每一个文件夹下都有一个 `Makefile` 文件将目标文件生成中间文件，便于后期整体组织

*目录组织如下：*

multibooheader

```
    multibootheader.s
myOS
    i386
        io.c
        Makefile
    dev
        uart.c
        vga.c
        Makefile
    printk
        vsprintf.c
        myprintk.c
        Makefile
    Makefile
    myOS.ld
    osStart.c
    start32.s
userApp
    main.c
    Makefile
Makefile
source2run.sh
```

底层文件夹下的 **Makefile** 文件使当前文件夹下的.c 文件生成中间文件，处于上层文件夹的 **Makefile** 将下层文件夹中生成的中间文件进行组织最后由最顶层文件夹下的 **Makefile** 汇总生成目标文件。

## 代码布局说明(地址空间)

代码在内存空间中的地址以及布局由 **myOS.ld** 文件所确定，代码将从物理内存 **1M** 的位置开始存放，代码段 **8** 个字节对齐，之后存放程序中已经初始化的全局变量并按 **16** 字节对齐，在之后存放未初始化的全局变量按 **16** 字节对齐，下一个存放的代码按 **512** 个字节对齐

## 编译过程说明

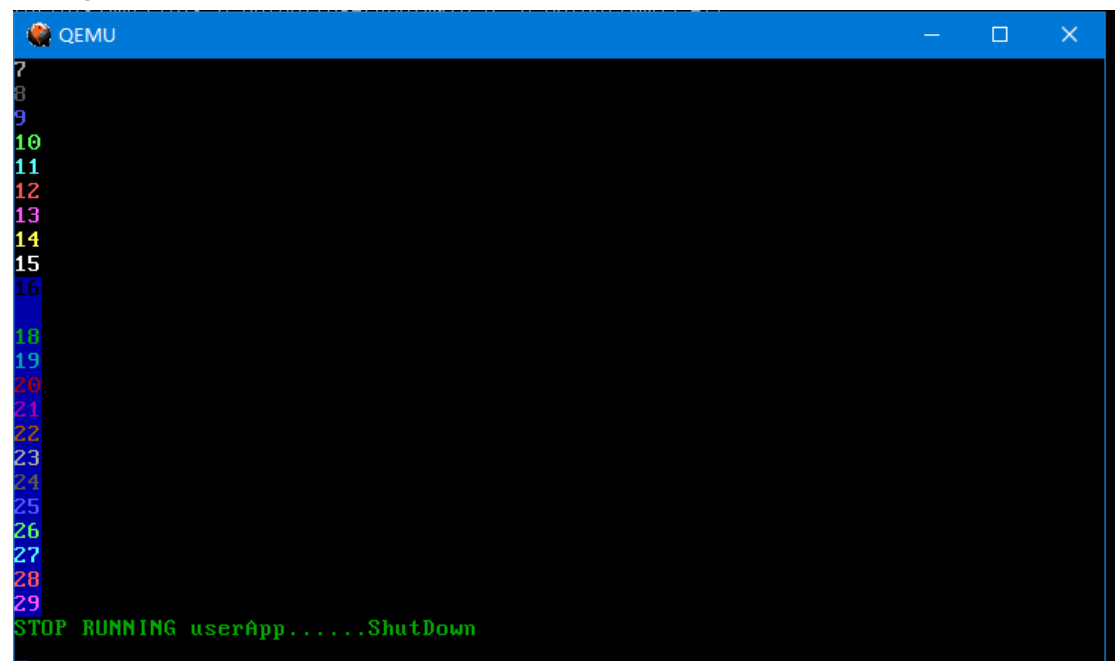
通过 `./source2run.sh` 命令执行 **make** 命令在 **Makefile** 的组织下使所有.c.s 文件生成中间文件最终按照 **myOS.ld** 文件代码布局以及地址空间的要求在内存空间生成并存放.elf 文件。

## 运行和运行结果说明

输入 `./source2run.sh` 命令运行脚本文件，编译后进而执行.elf 文件，运行结果如下：  
串口输出如下：

```
make succeed
START RUNNING userApp.....
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 userApp.....ShutDown
```

vga 输出如下:



```
QEMU
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 userApp.....ShutDown
```

## 遇到的问题 and 解决方案说明

问题: 直接执行脚本文件无法运行, 而分开执行 `make Makefile` 以及 `qemu-system-i386`

-kernel output/myOS.elf -serial stdio 命令则可以正常输出。

解决方案: 通过 `vim source2run.sh` 查看.sh 文件, 输入 `set ff` 发现文件格式是 dos 类型, 而 linux 只能执行格式为 unix 格式的脚本。通过 `set ff=unix` 命令即可将文件格式改为 unix 格式。从而该问题得到解决。