中国科学技术大学计算机学院
《操作系统原理与设计》

实验题目：multibootheader2mymain

学生姓名：叶子昂

学生学号：PB20020586

完成时间：2022年3月26日

实验题目

multibootheader2mymain

实验目的

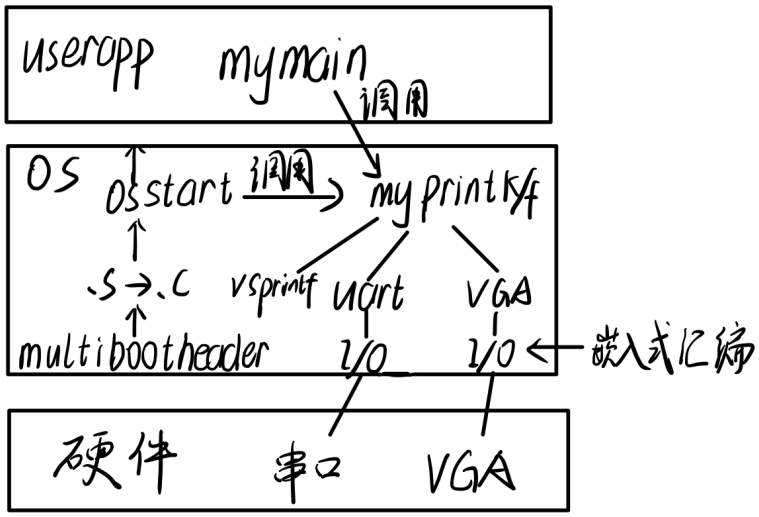
- 实现从multibootheader到myos最后能够运行mymain
- 实现彩色格式化输入输出（从串口和VGA），为调试做好准备

实验环境

wsl2，qemu

实验内容

1. 软件框图



myos运行在在qemu构建的虚拟硬件环境，myos通过ostart调用用户程序mymain来运行用户程序，myos通过提供myprintk/f为自身和用户程序提供格式化输出的功能。

2.系统启动流程

在qemu构建的虚拟环境中通过multibootheader启动，在start32.S中进行栈的构建和运行环境的初始化，然后系统由osStart.c接管系统开始运行，通过“调用”mymain来运行用户程序。

3.主功能模块及代码实现

- 系统栈的构建
  设置栈底为0xFFFFFF，栈大小为0x4000

```
  movl    $0xFFFFFF, %eax      # 填入栈底地址//TODO//done
  movl    %eax, %esp           # set stack pointer
  movl    %eax, %ebp           # set base pointer
```

- 嵌入式汇编完成单字符和字符串的串口输出(调用接口向端口输出数据即可)

```
void uart_put_char(unsigned char ch) {
    /* todo//done*/
     outb(0x3F8,ch);
}

void uart_put_chars(char *str) {
    /* todo//done*/
    for (int i = 0;str[i]!='\0';i++){
        uart_put_char(str[i]);
    }
}
```

- 单字符和字符串的VGA输出

  光标置位功能：通过设置索引端口依次写入光标偏移量前八位和后八位

```
void set_cursor_pos(unsigned short int pos) {
    /* todo//done */
    unsigned char line = 0;
    unsigned char column = 0;
    line = pos>>8;
    column = pos;
    outb(CURSOR_INDEX_PORT,CURSOR_LINE_REG);//写数据
    outb(CURSOR_DATA_PORT,line);
    outb(CURSOR_INDEX_PORT,CURSOR_COL_REG);
    outb(CURSOR_DATA_PORT,column);
}
```

获取光标位置功能：通过索引端口依次读出前后八位再拼接成光标偏移量

```
unsigned short int get_cursor_pos(void) {
    /* todo//done */
    unsigned char line = 0;
    unsigned char column = 0;
    outb(CURSOR_INDEX_PORT,CURSOR_LINE_REG);//读数据
    line=inb(CURSOR_DATA_PORT);
    outb(CURSOR_INDEX_PORT,CURSOR_COL_REG);
    column=inb(CURSOR_DATA_PORT);
    int pos = 0;
    pos = (pos | line)<<8;
    pos = pos | column;
    return pos;
}
```

滚屏：通过两个指针分别指向第一行和第二行，然后逐行复制，最后将光标置为最后一行第一位。

```
void scroll_screen(void) {
    /* todo//done */
    unsigned short int* p = VGA_BASE;
    unsigned short int* q = VGA_BASE+2*VGA_SCREEN_WIDTH;
    for (int i = 0; i < (VGA_SCREEN_WIDTH)*(VGA_SCREEN_HEIGHT-1); i++) {//逐行复制
        *p=*q;
        p++;q++;
    }
    q-=80;
    for(int i=0; i < VGA_SCREEN_WIDTH; i++){//清空最后一行
        *q=0x0F00;
        q++;
    }
    set_cursor_pos(80*24);
}
```

清屏功能：通过指针向整个VGA显存写入“黑底白标"(*p=0x0F00;)

```
void clear_screen(void) {
    /* todo//done */
    unsigned short int* p = VGA_BASE;
    for (int i = 0; i < VGA_SCREEN_WIDTH * VGA_SCREEN_HEIGHT; i++){
        *p=0x0F00;//黑底白标
        p++;
    }
    set_cursor_pos(0);
}
```

单字符输出功能：将字符和颜色数据拼接成显存数据并写入当前光标位置，同时注意处理换行和滚屏问题。

```
void put_char2pos(unsigned char c, int color, unsigned short int pos) {
    /* todo//done */
    unsigned short int* p=pos*2+VGA_BASE;
    unsigned short int data=0;
    if(c=='\n') {
        if(pos/80==24) scroll_screen();//光标在最底端换行滚屏
        else set_cursor_pos((pos/80+1)*80);//换行
    }
    else{
        data = data | (color<<8);
        data = data | c;
        *p=data;
        if(pos==80*25-1) scroll_screen();//已写入最后一个位置滚屏
        else set_cursor_pos(pos+1);//光标置下一位
    }
}
```

字符串输出功能：由于单字符输出已经处理好了各种问题，字符串输出只需循环调用单字符输出即可。

```
void append2screen(char *str, int color) { //换行和滚屏由put_char2pos()内部处理
    /* todo//done */
    for (int i = 0;str[i]!='\0';i++){
        put_char2pos(str[i],color,get_cursor_pos());
    }
}
```

- 格式化字符串的识别和处理（移植修改实现）

  通过在网上查找依赖较少的库，找到一个仅依赖于string.h的vsprintf函数，将其所调用的strlen()等功能自行实现后能够识别并处理格式化字符串。

- 实现类C库的myprintk/f

  首先将读入格式化字符串通过vsprintf处理暂存到kbuf/ubuf，后交由VGA和串口输出函数输出即可。

```
char kBuf[400];
int myPrintk(int color, const char *format, ...) {
    va_list args;

    va_start(args, format);
    int cnt = vsprintf(kBuf, format, args);
    va_end(args);
```

```
    /* todo//done */
    append2screen(kBuf,color);
    uart_put_chars(kBuf);
    return cnt;
}

char uBuf[400];
int myPrintf(int color, const char *format, ...) {
    va_list args;

    va_start(args, format);
    int cnt = vsprintf(uBuf, format, args);
    va_end(args);

    /* todo//done */
    append2screen(uBuf,color);
    uart_put_chars(uBuf);

    return cnt;
}
```

4.源代码组织说明

- 目录组织结构

```
.
├── Makefile
├── multibootheader
│   └── multibootHeader.S
├── myOS
│   ├── Makefile
│   ├── dev
│   │   ├── Makefile
│   │   ├── uart.c
│   │   └── vga.c
│   ├── i386
│   │   ├── Makefile
│   │   └── io.c
│   ├── include
│   │   ├── io.h
│   │   ├── myPrintk.h
│   │   ├── uart.h
│   │   ├── vga.h
│   │   └── vsprintf.h
│   ├── lib
│   │   ├── Makefile
│   │   └── vsprintf.c
│   ├── myOS.ld
│   ├── osStart.c
│   ├── printk
│   │   ├── Makefile
│   │   └── myPrintk.c
│   ├── start32.S
│   └── userInterface.h
├── source2img.sh
└── userApp
    ├── Makefile
    └── main.c
```

- makefile组织结构

```
.
|--MULTI_BOOT_HEADER
|    |__output/multibootheader/multibootHeader.o
|--OS_OBJS
     |__MYOS_OBJS
     |    |__output/myOS/start32.o
     |    |__output/myOS/osStart.o
     |    |__DEV_OBJS
     |    |    |__output/myOS/dev/uart.o
     |    |    |__output/myOS/dev/vga.o
     |    |__I386_OBJS
     |    |    |__output/myOS/i386/io.o
     |    |__PRINTK_OBJS
     |    |    |__output/myOS/printk/myPrintk.o
     |    |__LIB_OBJS
     |         |__output/myOS/lib/vsprintf.o
     |__USER_APP_OBJS
          |__output/userApp/main.o
```

5.代码布局说明

```
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/lib/vsprintf.o output/userApp/main.o -o output/myOS.elf
make succeed
Starting the OS...
16
4288
4320
4320
5136
```

Section	Offset	说明
.multibootheader	0	开头
.text	16	multibootheader占4×3=12,ALIGN(8)对齐后到16
.data	4288	通过myprint打印出
.bss	4320	通过myprint打印出
_end	5136	通过myprint打印出

6.编译过程说明

- 编译所用指令

  > make

- 编译过程

  1. 根据makefile找到并编译各个.c,.s文件到.o文件。
  2. 根据目标文件连接规则按照.ld文件描述进行连接生成myOS.elf

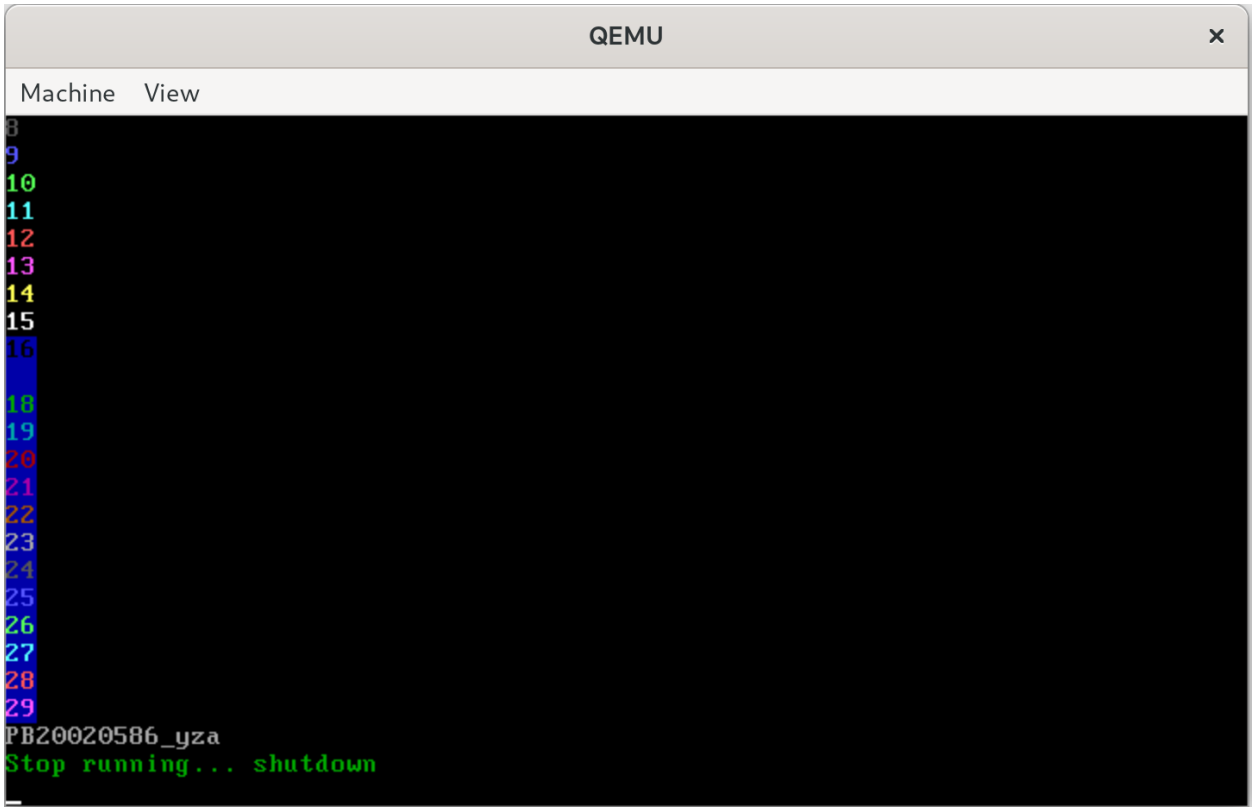
7.运行和运行结果说明

- 运行方式

  > zsh source2img.sh

- 运行结果

  qemu界面



  串口输出

```
Starting the OS...
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
PB20020586_yza
Stop running... shutdown
```

8.遇到的问题及解决

- 清屏时简单的将显存置为0x0000，致使光标无法显示。更改为0x0F00后解决

总结与思考

- 通过本次实验，我进一步熟悉了OS的启动方式和流程，能够再已有的代码框架下，理解代码并补全确实功能。
- 了解了如何通过嵌入式汇编沟通软硬件，将硬件输出封装成易用的函数供OS和用户程序使用。