

中国科学技术大学计算机学院
《操作系统原理与设计》实验报告



实验题目：lab2_Multiboot2myMain

学生姓名：胡毅翔

学生学号：PB18000290

完成日期：2020 年 3 月 17 日

计算机实验教学中心制

2019 年 09 月

实验目的

1. 在源代码层面，实现从汇编语言到 C 语言的衔接。
2. 在功能上，实现清屏、格式化输入输出，I/O 设备包括 VGA 和串口。
3. 在软件层次和结构上，完成 multiboot_header、myOS 和 userApp 的划分，体现在文件目录组织和 Makefile 组织上。

实验环境

1. PC 一台
2. Windows 系统
3. Ubuntu
4. QEMU
5. Xserver

软件框图

本实验的软件框图如图 1 所示。

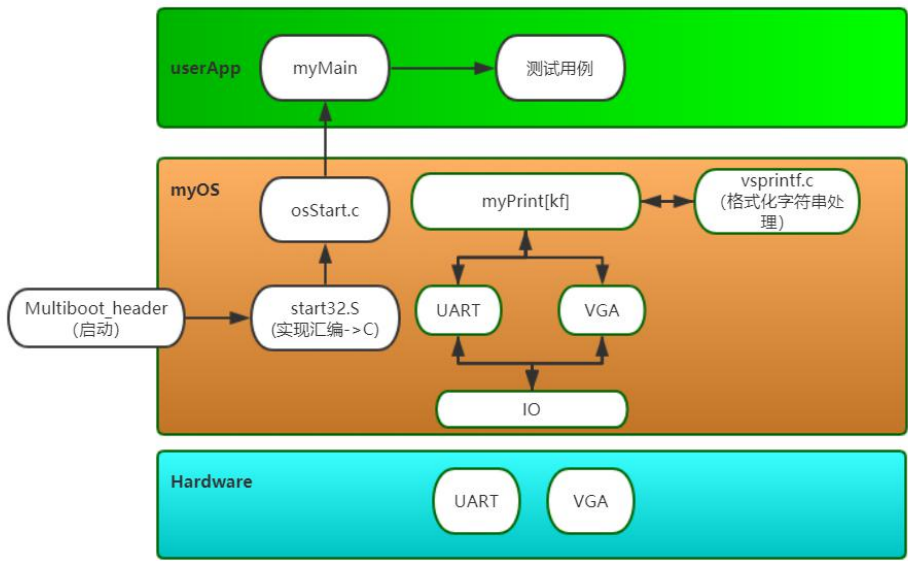


图 1

软件层次分为 multiboot_header、myOS 和 userApp 三部分。multiboot_header 为系统启动部分，系统启动后进入 myOS，在 osStart.c 中调用 myMain.c 进入 userApp 部分。若 userApp 部分，myMain 及其调用的程序需要输出字符串时，通过 myPrintk/f，将经 vsprintf 处理后的字符串通过 IO 接口，在 VGA 和 UART 上完成输出。

主流程

本实验的主流程如下图（图 2）所示：



图 2

1. 在 `multiboot_header` 中完成系统的启动。
2. 在 `start32.S` 中准备好上下文，最后调用 `osStart.c` 把进入 `c` 程序。
3. 在 `osStart.c` 中完成清屏等初始化操作，调用 `myMain`，进入 `userApp` 部分。
4. 运行 `myMain` 中的代码。

主要功能模块及其实现

I/O 的实现

该功能模块用于串口 `UART` 输出以及光标控制，主要目的是实现 `inb` 及 `outb` 的嵌入式汇编，将 `inb`、`outb` 两条汇编语句，写成 `c` 函数供串口 `UART` 输出及光标控制时调用。

`inb` 函数实现读取指定端口的值，并以返回值的形式，返回该值。

`outb` 函数将指定的值输出到指定端口。

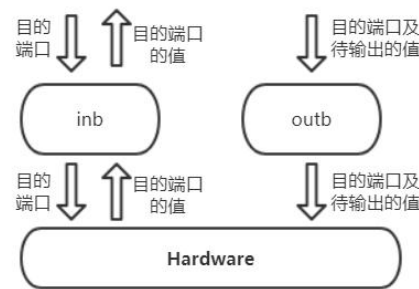


图 3

串口 `uart` 输出

该功能模块用于在串口输出单个字符及字符串，主要目的是实现 `uart_put_char` 以及 `uart_put_chars` 函数。

`uart_put_char` 实现输出单个字符，调用 `outb` 函数，将待输出字符及串口的端口（`0x3F8`）传递给 `outb` 函数，完成输出。

`uart_put_chars` 实现输出字符串，循环调用 `outb` 函数，将字符逐个输出到端口（`0x3F8`），直至字符串结束（`'\0'`），跳出循环，完成字符串输出。

示意图如图 4，图 5。

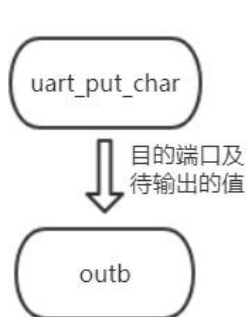


图 4

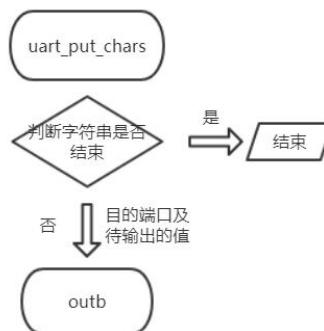


图 5

VGA 输出

该功能模块用于在 QEMU 中输出字符及控制光标，主要目的是实现 `clear_screen` 以及 `append2screen` 函数。

为实现对光标的控制，编写了 `rd_row`, `rd_col`, `wr_row`, `wr_col` 四个函数（利用 `inb`, `outb` 对 `0x3D4`, `0x3D5` 端口进行读写控制以实现），用于读写光标的位置，并用 `wr_cursor` 对 `wr_row` 及 `wr_col` 进行封装，可同时完成光标行列的控制。

为实现滚屏操作，编写了 `move` 函数，实现将屏幕中的第 2-25 行的内容移动到第 1-24 行，并将第 25 行清空，最后将光标移动至第 25 行起始位置。

同时，用全局变量 `row`, `col` 记录当前所在行列（以 0 作为起始行列序号，故行取值范围`[0,24]`，列取值范围`[0,79]`）。

在以上子函数的基础上，进一步完成 `clear_screen` 函数，将黑底空格输出满整个屏幕，并将光标写到坐标位 `(0, 0)` 的位置，且 `row`, `col` 变量置为 0。

而 `append2screen` 函数则，先对待输出字符进行判断，若为一般字符，则将指针赋值为 `(0xB8000+row*802+col*2)`，将指针所指位置定位的值为 `(color256(28)+对应字符的 ASCII 码)`，再将光标后移，完成输出。若为控制字符（如 `'\n'`），则根据对应含义完成相应的输出，并移动光标。

示意图如图 6。

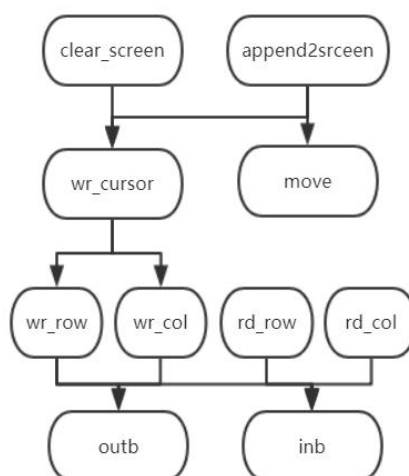


图 6

myPrint[kf]的实现

该模块的主要功能是将字符串格式化，并调用 `uart_put_chars` 及 `append2screen` 将格式化后的字符串输出。

实现过程为先调用 `vsprintf` 对字符串进行格式化，调用 `uart_put_chars` 及 `append2screen` 完成输出。

示意图如图 7。

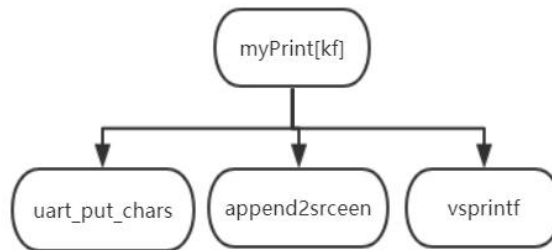


图 7

源代码说明

目录组织

目录组织如图 8 所示：

```
hyx@DESKTOP-LP1A2G2: ~/Github/os2020-labs/lab2/src
hyx@DESKTOP-LP1A2G2: ~/Github/os2020-labs/lab2/src$ tree
.
├── Makefile
├── README_multiboot2myMain.txt
├── multibootHeader
│   └── multibootHeader.S
├── myOS
│   ├── Makefile
│   ├── lib
│   │   ├── Makefile
│   │   ├── uart.c
│   │   └── vga.c
│   ├── i386
│   │   ├── Makefile
│   │   ├── io.c
│   │   ├── io.h
│   │   ├── myOS.ld
│   │   ├── osStart.c
│   │   ├── printf.c
│   │   ├── Makefile
│   │   ├── myPrintk.c
│   │   ├── types.h
│   │   └── vsprintf.c
│   └── start32.S
├── source2run.sh
├── userApp
│   ├── Makefile
│   └── main.c
└── 6 directories, 20 files
```

图 8

Makefile 组织

Makefile 组织结构如图 9 所示：

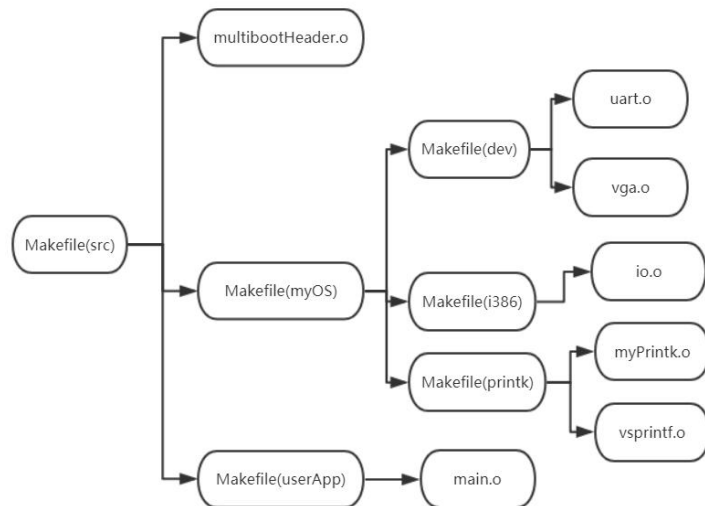


图 9

代码说明

IO

IO 部分用嵌入式汇编实现 `inb`, `outb` 函数。

`inb` 函数关键代码如下：

```
__asm__ __volatile__ ("inb %w1,%0":"=a"(_in_value):"Nd"(port_from));
```

功能为从 `port_from` 端口读取一个字节 并将对应值存至 `_in_value`。

`outb` 函数关键代码如下：

```
__asm__ __volatile__ ("outb %b0,%w1":"=a" (value),"Nd" (port_to));
```

功能为将 `value` 的值输出到 `port_to` 端口。

串口 uart 输出

串口 `uart` 输出部分实现输出单个字符及字符串。

关键代码如下：

```
#define uart_base 0x3F8
// 实现串口输出 单个字符
void uart_put_char(unsigned char c){
    outb(uart_base,c);
}
```

宏定义输出端口，调用 `outb` 函数将字符的 ASCII 码输出至对应端口。字符串输出的 `uart_put_chars` 函数同理，只需增加判断 (`str[i] != '\0'`)，以判断字符串输出是否结束。

VGA 输出

VGA 输出部分实现光标控制以及字符串在屏幕上的输出。

光标的读写仅需调用 `inb` 及 `outb` 函数，需要注意的是光标的坐标以 16 位表示，故读写光标的坐标时，输入输出的是高 8 位及低 8 位，而非屏幕中对应的行和列，须进行以下转换：

```
c = row*80+col;

wr_cursor(c/256,c%256);
```

滚屏部分代码及解释如下:

```
void move()
{
    int i,j;
    unsigned short int *tmp;
    unsigned short int a;
    for(i = 0; i < srceen_height; i++)//将内容向上移动一行
    {
        for(j = 0; j < srceen_width; j++)
        {
            tmp = (unsigned short int*) (vga_base + (i + 1) * 160 + 2 * j);
            //指针指向第 i 行第 j 列
            a = *tmp;//取值
            tmp = (unsigned short int*) (tmp - 80);//指向第 (i-1) 行第 j 列
            *tmp = a;//赋值
        }
    }
    for(j = 0; j < srceen_width; j++)//最后一行 全部置为空格
    {
        tmp = (unsigned short int*) (vga_base + 24 * 160 + 2 * j);
        *tmp = 0x0f20;
    }
}
```

清屏部分代码及解释如下:

```
void clear_screen(void) {
    int i;
    port = (unsigned short int*) vga_base; //指针指向 第 0 行 第 0 列
    p = vga_base; //指针对应地址
    for(i = 0; i < srceen_width * srceen_height; i++) //输出空格 实现清屏
    {

        *port = 0x0f20;
        p = p + 2;
        port = (unsigned short int*) p;
    }

    p = vga_base; //指针对应地址 置为 0xB8000
    row = 0; //行列置零
    col = 0;
    wr_cursor(row,col); //光标置零
}
```

```
}
```

字符串输出至屏幕部分代码及解释如下:

```
void append2screen(char *str,int color){
    int i,j,c;
    unsigned short int output;
    for(i = 0; ; i++)
    {
        if(str[i] != '\0')//字符串是否结束
        {
            if(str[i] != '\n')//是否换行
            {

                output = color * 16 * 16 + str[i];
                //颜色 : 前 8 位   ASCII 码: 后 8 位
                port = (unsigned short int*) (vga_base + row * 2 * srceen_width + col*2);
                //把指针指向 当前行列所指位置
                *port = output;//赋值
                col++;//列移动
                if(col > (srceen_width - 1))//判断是否需要换行
                {
                    row++;
                    col = 0;
                }
            }
            else
            {
                row++;
                col = 0;
            }

        }
        else
        {
            break;
        }
    }
    if(row == srceen_height)//是否需要滚屏
    {
        move();
        row = 24;
        col = 0;
    }
    c=row*80+col;
    wr_cursor(c/256,c%256);//光标移动
```



```
    }  
}
```

myPrint[kf]的实现

myPrint[kf]两个函数代码相同，先调用 `vsprintf` 对待输出字符串进行格式化处理，再调用 `uart_put_chars` 函数，`append2screen` 函数输出，代码如下：

```
int myPrintk(int color,const char *format, ...){  
    va_list args;  
    int i;  
    va_start(args, format);  
    i = vsprintf(kBuf, format, args); //格式化字符串 format，输出到 kBuf  
    va_end(args);  
    uart_put_chars(kBuf); //串口输出  
    append2screen(kBuf,color); //VGA 输出  
    return 0;  
}
```

代码布局说明（地址空间）

从物理内存 1M 的位置开始放代码和数据，前面 12 个字节为 `multiboot_header`，向后对齐 8 个字节，放代码。再向后对齐 16 个字节，用于放初始化的数据（数据段）。在数据段之后，再向后对齐 16 个字节。之后为 BSS（Block Started by Symbol）段，用于存放程序中未初始化的全局变量和静态变量。并在 BSS 段后，再向后对齐 16 个字节。剩余部分为堆栈段。

示意图如图 10 所示：

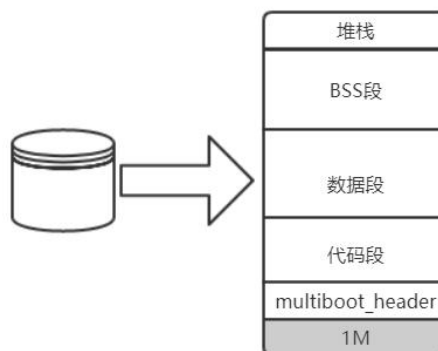


图 10

编译过程说明

```
ASM_FLAGS= -m32 --pipe -Wall -fasm -g -O1 -fno-stack-protector
```

-m32 :用 32 位机器的编译器来编译这个文件

--pipe:使用管道代替编译中临时文档

-Wall :打开警告选项

-fasm :识别 asm 关键字

-g :使用调试器 GDB

-O1 :优化生成代码

-fno-stack-protector :停止使用 stack-protector 功能

`C_FLAGS = -m32 -fno-stack-protector -fno-pic -fno-builtin -g`

-fpic 如果支持这种目标机,编译器就生成位置无关目标码.适用于共享库(shared library)

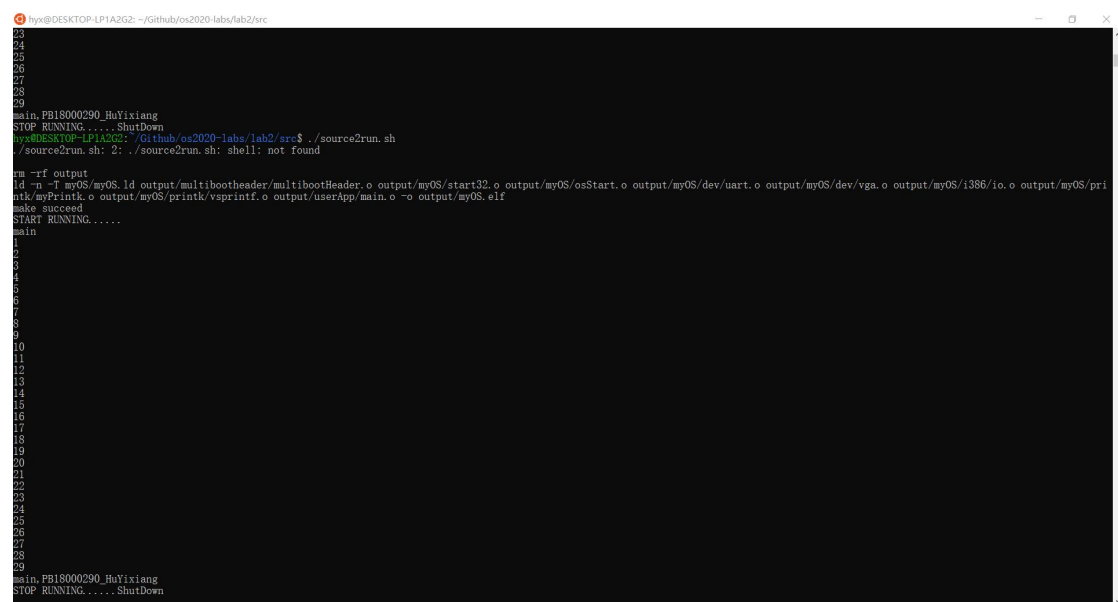
-fno-builtin :不使用 C 语言的内建函数

编译生成 multiHeader.o、osStart.o、start32.o、uart.o、vga.o、io.o、myPrintk.o、vsprintf.o 及 main.o 多个目标文件,链接生成 myOS.elf 文件。

运行和运行结果说明

输入 ./source2run.sh 指令后,编译,链接,生成 myOS.elf 文件并运行之。程序通过串口和 VGA,按 main.c 中的要求输出相应内容。运行结果如下图:

Ubuntu 中:



```
hyye@DESKTOP-LP1A2G2: ~/Github/os2020-labs/lab2/src
23
24
25
26
27
28
29
main: FB18000290_HuYixiang
STOP RUNNING..... ShutDown
hyye@DESKTOP-LP1A2G2: ~/Github/os2020-labs/lab2/src$ ./source2run.sh
./source2run.sh: 2: ./source2run.sh: shell: not found
rm -rf output
ld -m i386 -T myOS.ld output/multiHeader.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
make succeed
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
main: FB18000290_HuYixiang
STOP RUNNING..... ShutDown
```

图 11

QEMU 中:

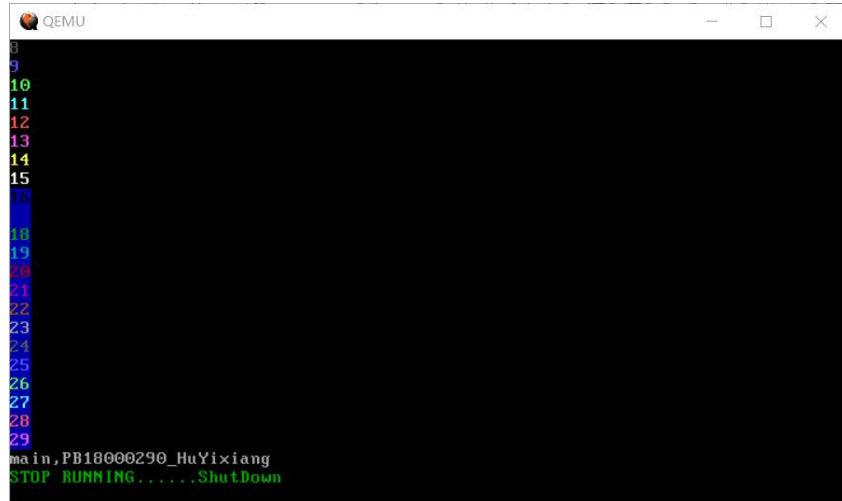


图 12

遇到的问题 and 解决方案说明

1. 编译时出现报错“对'_GLOBAL_OFFSET_TABLE_'未定义的引用”。
解决方案：在 src 目录下的 Makefile 文件的 CFLAGS 变量中添加-fno-pic。
2. 编译出现 fatal error: bits/libc-header-start.h: No such file or directory
解决方案:在 Ubuntu 中输入 apt-get install gcc-multilib，完善编译环境。
3. 编译时出现 warning: assignment makes pointer from integer without a cast
解决方案：在给指针赋值前进行强制格式转化(unsigned short int*).
4. 编译时出现 warning: conflicting types for built-in function
解决方案：在 src 目录下的 Makefile 文件的 CFLAGS 变量中添加-fno-builtin。