

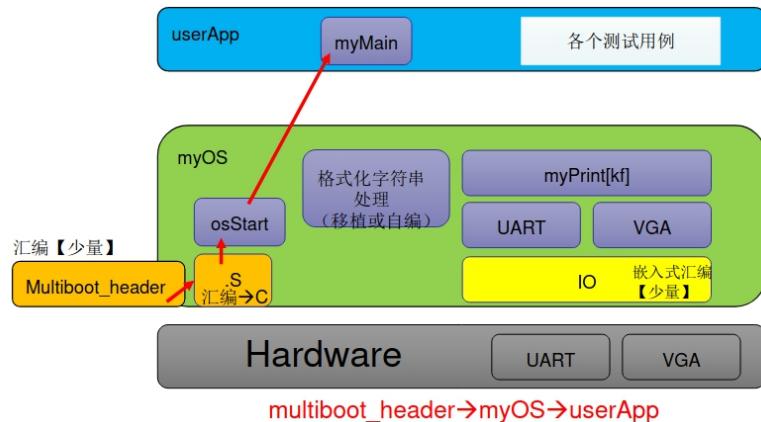
第二次试验报告

PB20000024 陈奕衡

一、实验目标

在源代码的语言层面，完成从汇编语言到 C 语言的衔接；在功能上，实现清屏、格式化输入输出，设备包括 VGA 和串口，接口符合要求；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；采用自定义测试用例和用户（助教）测试用例相结合的方式进行验收；提供脚本完成编译和执行。

二、软件架构与功能说明



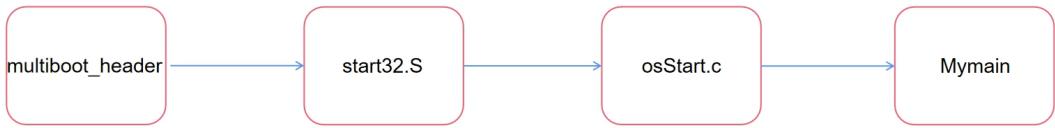
硬件方面：使用 qemu 虚拟机的 uart(串口) 输出至终端和 vga 输出至屏幕

软件方面：分成三个部分，即启动头，包含：Multiboot_header 文件进行启动并进入系统软件模块；系统软件 (myOS)，包含：对于格式化字符串的处理，串口输出以及 vga 输出的相应函数，以及系统输出 myPrintk 函数；最后是 userApp，包含：用户可编写的 myMain 函数

三、主流程说明

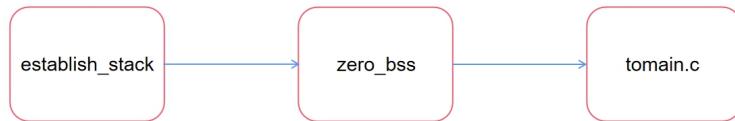
首先是 qemu 虚拟机启动并运行 Multiboot_header 文件，通过启动头进入 myOS 操作系统(访问 start.S)，在操作系统中通过 start32.S 文件运行 osStart.c

文件，并在此文件中运行用户应用 myMain 程序查看 vga 以及串口输出。



四、功能模块以及相应源代码说明

首先介绍从汇编语言到 c 语言的连接，该过程主要在 start32.S 中实现，具体过程为首先在启动之后构建栈，之后清空.bss 段，最后调用 osstart.c 程序。此时便进入了 c 语言的编写与调用过程，之后通过 io.c 文件的内嵌汇编成功使用 c 语言调用汇编的 outb 和 inb 函数从而进行对输入输出的编写。



```
# Set up the stack
establish_stack:
    # 指定栈底地址（栈是逆向增长的，栈底是高地址），提供两种参考方案
    # 1. 根据操作系统在内存中的结束地址 和 STACK_SIZE，计算栈底（下面三行被注释掉的内容）
    # 2. 硬编码，手动填入一个固定的栈底地址（下面第四行未被注释掉的内容）
    # 使用其中一种，请注释/删除掉另一种
    movl $end, %eax      # 填入正确的内容
    addl $STACK_SIZE, %eax  # make room for stack
    andl $0xffffffe0, %eax  # align

    # movl ????????, %eax      # 填入栈底地址

    movl %eax, %esp        # set stack pointer
    movl %eax, %ebp        # set base pointer
```

上图为构建栈

i386 模块：

该模块主要完成 c 语言内嵌汇编函数的建立

```
// inb, 从端口号为 port_from 的端口读取一个字节
unsigned char inb(unsigned short int port_from) {
    unsigned char value;
    __asm__ __volatile__ ("inb %w1, %b0": "=a"(value): "Nd"(port_from));
    return value;
}

// outb, 向端口号为 port_to 的端口输出一个字节
void outb(unsigned short int port_to, unsigned char value) {
    __asm__ __volatile__ ("outb %b0, %w1": "a"(value), "Nd"(port_to));
}
```

Dev 模块：

该模块有两个主要文件 uart.c 和 vga.c 分别对应输出。

通过 inb 与 outb 函数，就可以进行 vga.c 和 uart.c 中的各功能函数编写：

```
/* 将光标设定到特定位置
 * 提示：使用 outb */
void set_cursor_pos(unsigned short int pos) {
    /* todo */

    outb(CURSOR_INDEX_PORT, CURSOR_LINE_REG);
    outb(CURSOR_DATA_PORT, (unsigned char)((pos >> 8) & 0xff)); //set line

    outb(CURSOR_INDEX_PORT, CURSOR_COL_REG);
    outb(CURSOR_DATA_PORT, (unsigned char)(pos & 0xff)); //set column

    return;
}
```

此函数只需将 pos 高八位和低八位送至对应寄存器即可设置当前光标位置

```
/* 获取光标当前所在位置
 * 提示：使用 inb */
unsigned short int get_cursor_pos(void) {
    /* todo */
    unsigned short int x, y; //16 bits enough

    outb(CURSOR_INDEX_PORT, CURSOR_LINE_REG);
    x = inb(CURSOR_DATA_PORT); //get line

    outb(CURSOR_INDEX_PORT, CURSOR_COL_REG);
    y = inb(CURSOR_DATA_PORT); //get column

    return ((x << 8) | y);
}
```

此函数只需从对应寄存器中读取偏移量 pos 高八位和低八位即可

```
/* 滚屏，vga 屏幕满时使用。丢弃第一行内容，将剩余行整体向上滚动一行
 * 提示：使用指针修改显存 */
void scroll_screen(void) {
    /* todo */
    unsigned short int pos = get_cursor_pos();
    unsigned short int *ptr;

    ptr = (short int *) VGA_BASE;
    int i = 0;

    for(; i <= pos - VGA_SCREEN_WIDTH; i++) //change the line
        *(ptr + i) = *(ptr + i + VGA_SCREEN_WIDTH);
    for(; i <= pos; i++)
        *(ptr + i) = 0;

    pos -= VGA_SCREEN_WIDTH; //get new pos
    set_cursor_pos(pos);

    return;
}
```

此函数只需对每行内容进行偏移，并清空最后一行编码即可

```

/* 向 vga 的特定光标位置 pos 输出一个字符
 * 提示：使用指针修改显存 */
void put_char2pos(unsigned char c, int color, unsigned short int pos) {
    /* todo */
    unsigned short int out = (color << 8) + c;
    unsigned short int *ptr;

    ptr = (short int *) VGA_BASE;

    if(c == '\n'){//get the new pos
        pos += VGA_SCREEN_WIDTH - (pos % VGA_SCREEN_WIDTH) + 1;
    }
    else{
        *(ptr + pos - 1) = out;
        pos++;
    }

    set_cursor_pos(pos);
    if (pos >= VGA_SCREEN_HEIGHT * VGA_SCREEN_WIDTH){//set the new pos
        scroll_screen();
    }

    return;
}

```

此函数分成两部分实现，分别是输出字符（或者换行）和设定光标位置，在设定光标位置时需要考虑页面已满的情况。

```

/* 清除屏幕上所有字符，并将光标位置重置到顶格
 * 提示：使用指针修改显存 */
void clear_screen(void) {
    /* todo */
    unsigned short int pos = get_cursor_pos();
    unsigned short int *ptr;

    ptr = (short int *) VGA_BASE;
    for(int i = 0; i <= pos; i++) //clear the context
        *(ptr + i) = 0;

    pos = 0; //set the new pos
    set_cursor_pos(pos);

    return;
}

```

此函数只需读取当前光标位置并且清除所有之前的内容即可

```

/* 向 vga 的当前光标位置输出一个字符串，并移动光标位置到串末尾字符的下一位
 * 如果超出了屏幕范围，则需要滚屏
 * 需要能够处理转义字符 \n */
void append2screen(char *str, int color) {
    /* todo */
    unsigned short int pos = get_cursor_pos();

    for(int i = 0; *(str + i) != '\0'; i++){
        put_char2pos(*(str + i), color, pos);
        pos = get_cursor_pos();
    }

    return;
}

```

此函数只需在 put_char2pos 基础上循环输出即可

```

/* 向串口输出一个字符
 * 使用封装好的 outb 函数 */
void uart_put_char(unsigned char ch) {
    /* todo */
    unsigned short int port = UART_PORT;

    outb(port, ch);

    return;
}

/* 向串口输出一个字符串
 * 此函数接口禁止修改 */
void uart_put_chars(char *str) {
    /* todo */
    for(int i = 0; *(str + i) != '\0'; i++)
        uart_put_char(*(str + i));

    return;
}

```

相比于 vga 输出，串口输出会简单很多（不用特殊处理滚屏和换行），直接进行输出即可。以上是 dev 模块的介绍，该模块主要进行 qemu 虚拟机相应端口的调用。最后补全头文件：

```

void append2screen(char *str, int color);
void clear_screen(void);
void set_cursor_pos(unsigned short int pos);
unsigned short int get_cursor_pos(void);
void scroll_screen(void);
void put_char2pos(unsigned char c, int color, unsigned short int pos);

```

lib 模块：

该模块主要为格式化字符串处理文件。

若要实现 printk 函数，还需要有对格式化字符串的处理，这里采用标准 c 语言库中的 vsprintf 函数进行对字符串的处理

```

int vsprintf(char *buf, const char *fmt, va_list args)
{
    char *str;
    int field_width;      /* Width of output field */

    for (str = buf; *fmt; fmt++)
    {
        unsigned long num;
        int base = 10;

```

printk 模块：

该模块实现自己的 printf 函数，从而能够进行输出操作

```

/* 内核 print 函数
 * 调用已经完成的 vga 和 串口 输出接口，补全此函数
 * 禁止修改此函数接口 */
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 */
    unsigned short int pos = get_cursor_pos();

    for (int i = 0; i < cnt; i++)
    {
        uart_put_char(kBuf[i]);
    }

    for(int i = 0; i < cnt; i++){
        put_char2pos(kBuf[i], color, pos);
        pos = get_cursor_pos();
    }

    return cnt;
}

```

上图为内核 print 函数的实现，下图为用户 print 函数的实现：

```

/* 用户 print 函数
 * 调用已经完成的 vga 和 串口 输出接口，补全此函数
 * 禁止修改此函数接口 */
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 */
    unsigned short int pos = get_cursor_pos();

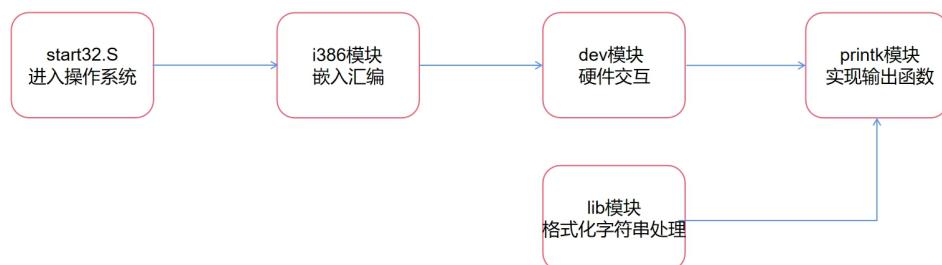
    for (int i = 0; i < cnt; i++)
    {
        uart_put_char(uBuf[i]);
    }

    for(int i = 0; i < cnt; i++){
        put_char2pos(uBuf[i], color, pos);
        pos = get_cursor_pos();
    }

    return cnt;
}

```

以上为操作系统各功能模块介绍，总流程图如下：



四、代码分布说明

代码的地址空间分布如下：

```
SECTIONS {
    . = 1M;
    .text : {
        *(.multiboot_header)
        . = ALIGN(8);
        *(.text)
    }

    . = ALIGN(16);
    .data      : { *(.data*) }

    . = ALIGN(16);
    .bss       :
    {
        __bss_start = .;
        __bss_start = .;
        *(.bss)
        __bss_end = .;
    }
    . = ALIGN(16);
    _end = .;
    . = ALIGN(512);
}
```

其中.text 代码段在 1m 位置进行存储，以 8 字节形式对齐；.data 段、.bss 段以 16 字节形式对齐；最后的_end 以及后续代码以 512 字节形式对齐。

五、编译过程说明

根据文件中所用脚本，直接使用脚本进行编译：

```
cyh@cyh-ROG-Strix-G732LWS-G732LWS:~/os/Lab2/src$ ./source2img.sh
rm -rf output
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
```

编译所用 Makefile 文件如下，首先对启动头进行编译，编译完成后进入 myOS 文件夹编译系统部分，即 dev、lib、i386、printk 功能模块和 start32.S、osStart 文件，最后进入 userApps 编译 main.c 文件。在编译完成后，新建 output 文件存储相应结果，而在每层文件夹中都会存储相应 Makefile 文件编译的.o 文件，最后通过 myOS.elf 文件进行链接。

```

#makefile
SRC_RT = $(shell pwd)

CROSS_COMPILE=
ASM_FLAGS = -m32 --pipe -Wall -fasm -g -O1 -fno-stack-protector
C_FLAGS = -m32 -fno-stack-protector -fno-builtin -g

.PHONY: all
all: output/myOS.elf

MULTI_BOOT_HEADER = output/multibootheader/multibootHeader.o
include ${SRC_RT}/myOS/Makefile
include ${SRC_RT}/userApp/Makefile

OS_OBJS = ${MYOS_OBJS} ${USER_APP_OBJS}

output/myOS.elf: ${OS_OBJS} ${MULTI_BOOT_HEADER}
${CROSS_COMPILE}ld -n -T myOS/myOS.ld ${MULTI_BOOT_HEADER} ${OS_OBJS} -o output/myOS.elf

output/%.o : %.S
@mkdir -p $(dir $@)
${CROSS_COMPILE}gcc ${ASM_FLAGS} -c -o $@ $<

output/%.o : %.c
@mkdir -p $(dir $@)
${CROSS_COMPILE}gcc -I myOS/include -I myOS ${C_FLAGS} -c -o $@ $<

clean:
rm -rf output

```

编译所用脚本如下：

```

#!/bin/bash

make clean

make

if [ $? -ne 0 ]; then
    echo "make failed"
else
    echo "make succeed"
    qemu-system-i386 -kernel output/myOS.elf -serial stdio
fi

```

六、运行结果说明

vga 输出：



uart 输出：

```
cyh@cyh-ROG-Strix-G732LWS-G732LWS: ~/os/Lab2/src
```

```
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
PB20000024_cyh
Stop running... shutdown
```

测试输出所用的 main.c 文件如下：

```
#include "userInterface.h"

void myMain(void) {

    int i;
    for (i = 1; i < 30; i++)
        myPrintf(i, "%d\n", i);

    myPrintk(0x7, "PB20000024_cyh\n"); // 你的学号、姓名

    return;
}
```

测试输出成功，运行方式同样为脚本运行

七、遇到的问题及解决方案

```
myOS/dev/vga.c: In function 'scroll_screen':
myOS/dev/vga.c:47:18: warning: assignment to 'short unsigned int *' from 'int' makes pointer from integer without a cast [-Wint-conversion]
    47 |     VGA_BASE_SET = VGA_BASE;
          |     ^
myOS/dev/vga.c: In function 'put_char2pos':
myOS/dev/vga.c:65:18: warning: assignment to 'short unsigned int *' from 'int' makes pointer from integer without a cast [-Wint-conversion]
    65 |     VGA_BASE_SET = VGA_BASE;
          |     ^
```

首先是指针赋值问题，直接赋值会报错，这里通过强制修改类型完成赋值。

```
    / todo /
    unsigned short int pos = get_cursor_pos();
    unsigned short int *ptr;

    ptr = (short int *) VGA_BASE;
    for(int i = 0; i <= pos; i++) //clear the context
        *(ptr + i) = 0;
```

之后还有一点是 main.c 中提示无法找到函数的问题，这里是因为头文件中

未添加相应函数导致。

```
myOS/printk/myPrintk.c: In function ‘myPrintk’:  
myOS/printk/myPrintk.c:17:30: warning: implicit declaration of function ‘get_cursor_pos’ [-Wimplicit-function-declaration]  
  17 |     unsigned short int pos = get_cursor_pos();  
      |  
myOS/printk/myPrintk.c:19:9: warning: implicit declaration of function ‘put_char2pos’ [-Wimplicit-function-declaration]  
  19 |         put_char2pos(kBuf[i],color,pos);  
      |
```

添加相应函数至头文件即可