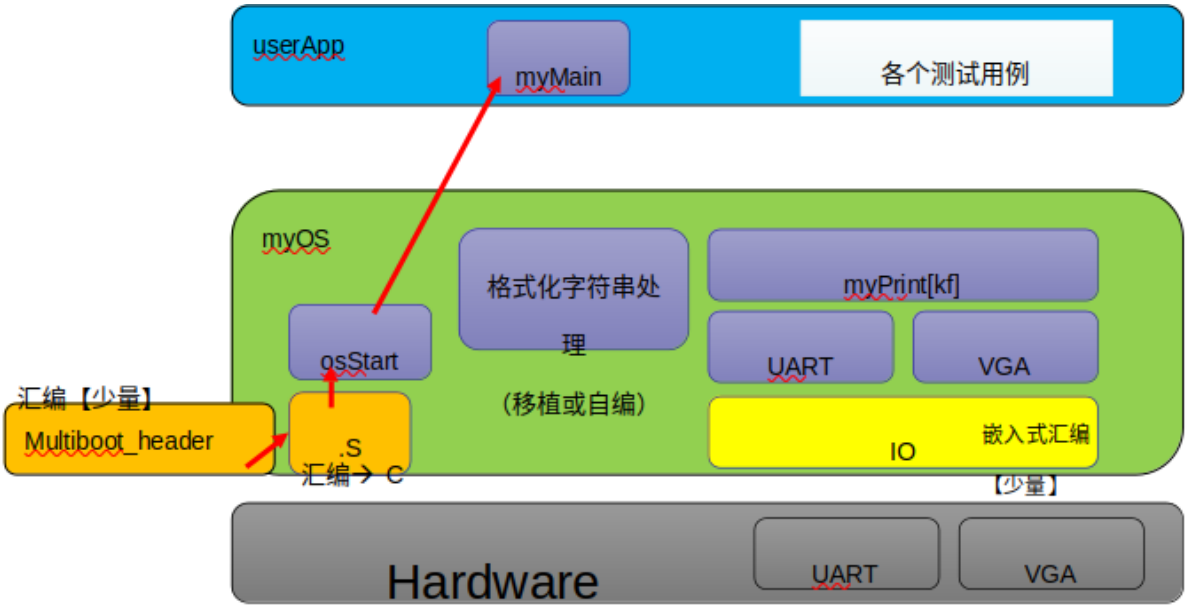
lab2实验报告

毛陈诚 PB20111694

详细说明主要功能模块及其实现，画出流程图



multiboot_header→myOS→userApp

1. `multibootheader.s` 是一段操作系统启动代码，在设置好启动头标的代码后，将程序引导执行 `start.s`
2. `start.s` 设置堆和栈的大小，然后将BSS段清零，将程序引导执行 `osstart.c`
3. `osstart` 调用 `mymain` 函数，用于将 `myOS` 层与 `userApp` 层连接
4. `myprint[kf]` 对输入字符串进行格式化处理，是用户可以调用的函数
5. `vga/uga` 实现vga的相关功能，清屏和屏幕输出,调用 `IO` 的 `inb`,`outb`
6. `IO` 实现端口输出，采用嵌入式汇编
7. `userApp` 是最外层的接口来使操作者调用 `myprintk` 或 `myprintf` 函数实现输出功能

详细说明主流程及其实现，画出流程图

1. `Multiboot header` 中为进入C程序准备好上下文，然后调用 `_start` 入口
2. 在操作系统初始化完毕后， `OSstart` 调用 `mymain` 函数，转入用户程序运行
3. 用户程序只能调用操作系统定义的接口和用户自定义的函数，这里 `mymain` 函数调用 `myprintk` 函数解析字符串
4. `myprintk` 函数调用 `myOS` 中的函数包括: `IO` 端口输出;串口 `uart` 输出,VGA输出

源代码说明（目录组织、Makefile组织）

1. Makefile　　提供，可修改

2. Multibootheader子目录
3. 内核子目录　子目录下可以进一步按功能划分子目录
    - dev: `vga`, `uart`
    - i386: `IO`
    - printk: `myprintk`
4. userApp子目录
5. main.c
6. output子目录（所有编译链接生成的文件在此）
7. source2run.sh　　脚本文件，提供，不要修改

代码布局说明

start32.s

```
movl    $_end, %ecx
```

填入eax = end of bss/start of heap即可

vga.c

updata_cursor

将索引号写入索引端口0x3D4，然后对数据端口0x3D5写光标的行值/列值

```
void update_cursor(void){//通过当前行值cur_cline与列值cur_column回写光标
    //填写正确的内容
    outb(index_base,LINE);
    outb(data_base,cur_line);
    outb(index_base,COLUMN);
    outb(data_base,cur_column);
}
```

get_cursor_position

将索引号写入索引端口0x3D4，然后对数据端口0x3D5读光标的行值/列值

```
short get_cursor_position(void){//获得当前光标，计算出cur_line和cur_column的值
    //填写正确的内容
    outb(index_base,LINE);
    cur_line = inb(data_base);
    outb(index_base,COLUMN);
    cur_column = inb(data_base);
}
```

clear_screen

向屏幕范围内的数组输入`0x0007`

```
void clear_screen(void) {
    char str;
    for (int i = 0; i < LINENUMBER; ++i)
        for (int j = 0; j < COLUMNNUMBER * 2; j += 2) {
            vga_init_p[i * COLUMNNUMBER * 2 + j] = 0x00;
            vga_init_p[i * COLUMNNUMBER * 2 + j + 1] = 0x07;
        }
    cur_line = 0;
    cur_column = 0;
    update_cursor();
}
```

scroll_up

把除了第一行的所有数据向前移动一行，将最后一行清零

```
void scroll_up(void){
    int backline = COLUMNNUMBER*SIZEOFWORD;
    pos = get_pos();
    for (int i = backline; i < pos; i++){
        vga_init_p[i - backline] = vga_init_p[i];
    }
    for(int i = pos -backline; i < pos; i = i + 2){
        vga_init_p[i] = 0x00;
        vga_init_p[i+1] = 0x07;
    }
}
```

outputonechar

单个字符输出

- 如果是换行符
  - 如果列满，则滚动屏幕，将光标更新到第一列
  - 如果列未满，则将光标换行
- 如果是一般字符
  - 如果行满
    - 如果列满，则滚动屏幕，将光标更新到第一列
    - 如果列未满，则将光标换行
  - 如果行未满，将光标后移一格

```
void outputonechar(char *str,int color){
    get_pos();
    if(*str == '\r'|| *str == '\n'){
```

```
        if(cur_line == LINENUMBER -1){
            scroll_up();
            //cur_line++;
            cur_column = 0;
        }
        else{
            cur_line++;
            cur_column = 0;
        }
    }
    else{
        vga_init_p[pos] = *str;
        vga_init_p[pos + 1] = color;
        if(cur_column == COLUMNNUMBER - 1){
            if(cur_line == LINENUMBER - 1){
                scroll_up();
                cur_column = 0;
            }
            else{
                cur_column = 0;
                cur_line++;
            }
        }
        else{
            cur_column += 1;
        }
    }
    update_cursor();
}
```

append2screen

对字符串每个字符调用`outputonechar`

```
void append2screen(char *str,int color){
    get_pos();
    while(*str){
        outputonechar(str,color);
        str++;
    }
}
```

vsprintf

自定义了有关字符串的库函数包括

strlen

```
int strlen(const char *str) { int len = 0; while (*str != '\0') { len++; str++; }
return len; }
```

strcpy

```
void strcpy(char *dest, const char *src) { while (*src != '\0') { *dest = *src;
dest++; src++; } *dest = '\0'; }
```

itoa

整数转字符串

```
void itoa(int value, char *str) {
    int sign = value < 0 ? -1 : 1;
    int i = 0;
    do {
        str[i++] = '0' + sign * (value % 10);
        value /= 10;
    } while (value != 0);
    if (sign < 0) {
        str[i++] = '-';
    }
    str[i] = '\0';
    reverse(str); // 反转字符串
}
```

reverse

```
void reverse(char *str) {
    int len = strlen(str);
    int i = 0; int j = len - 1;
    char temp;
    while (i < j) {
        temp = str[i];
        str[i] = str[j];
        str[j] = temp;
        i++;
        j--;
    }
}
```

vsprintf

利用可变参数的`va_arg`,从可变参数列表中获取整数值

```
/*
 * 识别格式化字符串的核心代码写在本文件中
 * 可以从网上移植代码
```

```
 */

#include <stdarg.h>
#define MAX_BUFFER_SIZE 1024
#define BUFFER_OVERFLOW -2

int vsprintf(char* str, const char* format, va_list arg) {
    char buffer[MAX_BUFFER_SIZE];
    // 定义一个指向缓冲区的指针
    char* ptr = buffer;
    // 定义一个用于存储整数值的字符串
    char num[12];
    // 定义一个用于存储整数值的变量
    int value;
    // 遍历格式字符串
    while (*format != '\0') {
        // 如果遇到%符号，表示需要格式化输出
        if (*format == '%') {
            // 跳过%符号
            format++;
            // 根据不同的格式标记，处理不同的数据类型
            switch (*format) {
                // 如果是d，表示输出十进制整数
                case 'd':
                    // 从可变参数列表中获取整数值
                    value = va_arg(arg, int);
                    // 将整数值转换为字符串
                    itoa(value, num);
                    // 将字符串复制到缓冲区中
                    strcpy(ptr, num);
                    // 更新缓冲区指针位置
                    ptr += strlen(num);
                    format++;
                    break;
            }
        }

        // 如果不是%符号，表示直接输出字符
        else {
            // 将字符复制到缓冲区中
            *ptr++ = *format++;
        }
    }
    // 在缓冲区末尾添加空字符
    *ptr = '\0';

    // 将缓冲区的内容复制到目标字符串中
    strcpy(str, buffer);

    // 返回写入的字符数，不包括空字符
    return strlen(str);
}
```

编译过程说明

不知道要说明个啥

运行和运行结果说明

遇到的问题和解决方案说明

1. 编写`vsprintf`不能调用库函数，比如说`strcpy`等。

   解决方案：自己实现，详细在代码布局中已经说明

2. 光标位置在各种操作（清屏，换行）中的变化，容易搞错

   解决方案：不断试错与调整