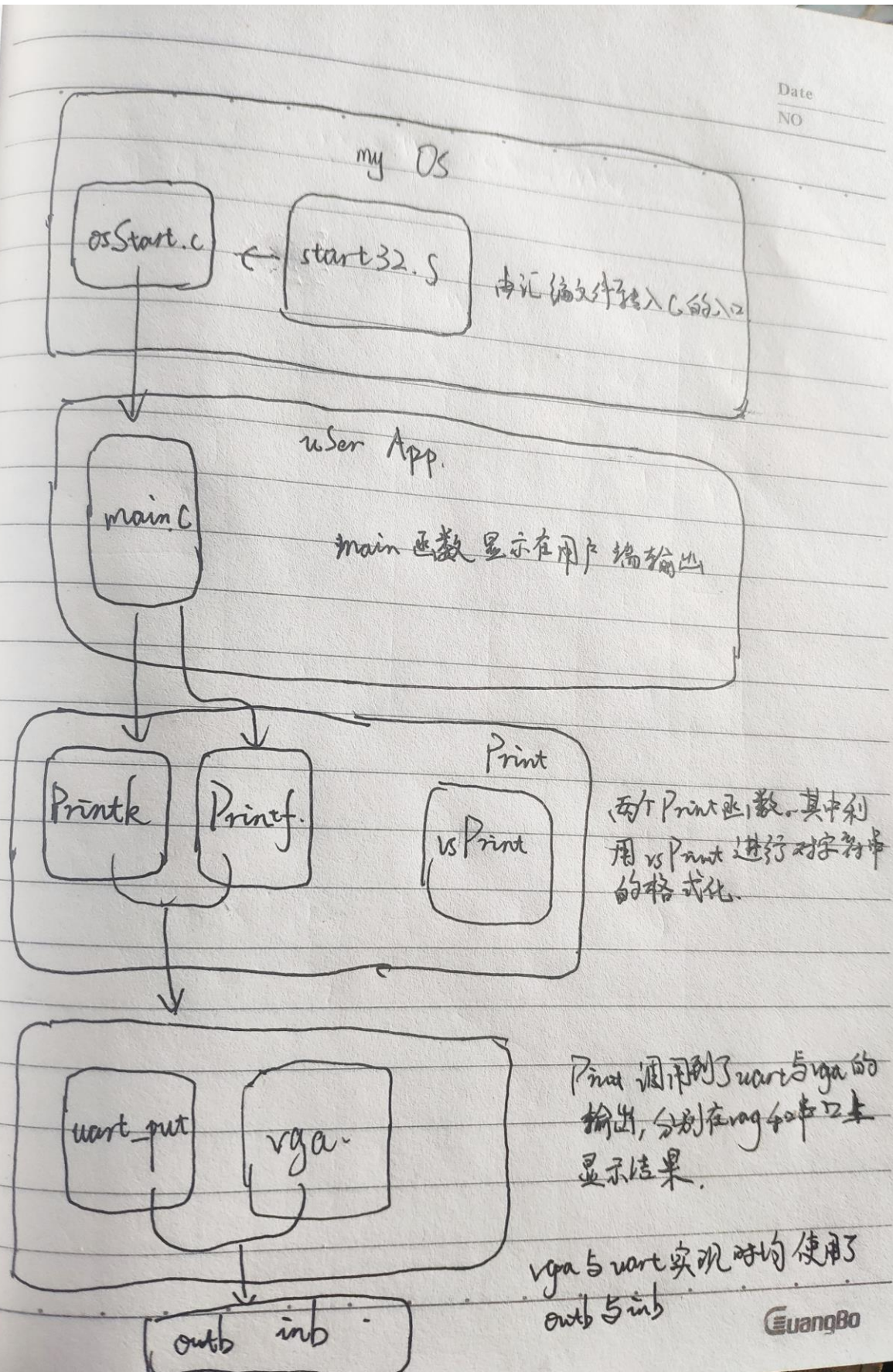
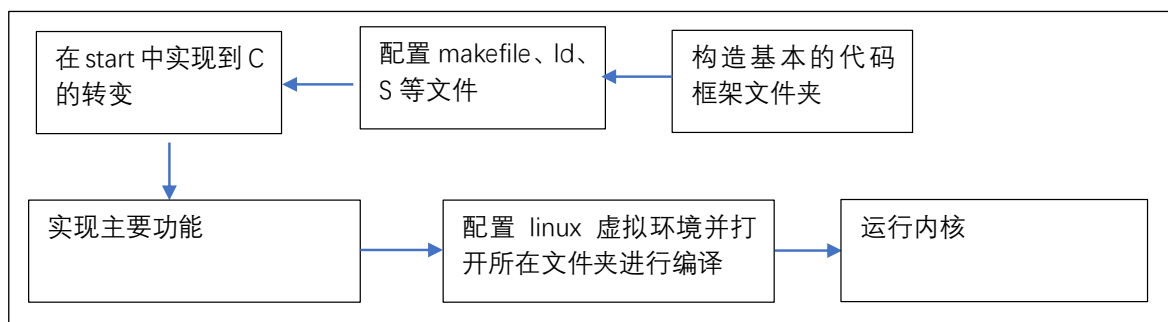


操作系统实验 2 报告

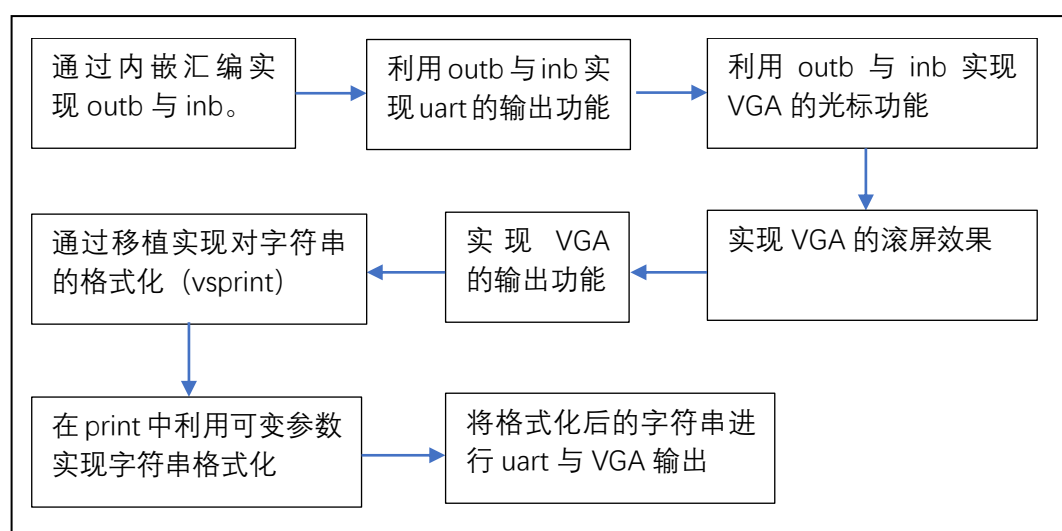
软件框图



主流程及其实现



主要功能模块及其实现



源代码说明

在本次实验的根目录下，除了配置了 makefile 与 DS 文件，还有 4 个文件夹：multibooheader、myOS、output、userApp。multibooheader 已经不陌生，其中放置了有关 multibooheader 协议的 S 文件。在 output 中，原本为空文件夹，用来输出我们编译后生成的文件。在该文件夹中的目录结构与根目录相似，主要是为了对每个文件输出时不产生混淆。在 userApp 中存放的是 main.c 文件与该文件夹下的 makefile 文件，Main 文件是用来测试本次实验的功能的入口。在 myOS 文件夹中存放了本次实验的主要源文件。直接存放在此该目录下的文件有

DS 文件、该文件夹下的 makefile 文件、链接器文件、一个配置地址的 S 文件和一个包含了 main 函数的与 multibootheader 中的 S 文件相关联的 C 文件。这个文件也是从汇编到 C 的转变。在该目录下则有 3 个实现相关功能的文件夹: dev、i386、printk。每个文件夹下都有一个相应的 makefile 文件以及实现功能的 C 语言源文件。i386 文件夹下是实现 IO (inb、outb) 的源文件, dev 下存放的是在 uart 与 vga 上实现给定字符串并输出的功能的源文件, 而在 printk 中则是实现 printf 与 printk 的源文件。每个子目录下的 makefile 文件的输出目录都是在 output 中的同名目录。相关源代码如图:

```
1  /* IO operations */
2  //使用内嵌汇编实现对给定地址的读写操作
3  unsigned char inb(unsigned short int port_from){
4      unsigned char value;
5      __asm__ __volatile__ ("inb %w1,%b0":"=a" (value):"Nd" (port_from));
6      return value;
7  }
8
9  void outb (unsigned short int port_to, unsigned char value){
10     __asm__ __volatile__ ("outb %b0,%w1::" "a" (value), "Nd" (port_to));
11 }
```

实现 VGA 功能的代码

```
1  extern void outb(unsigned short int port_to, unsigned char value);
2  extern unsigned char inb(unsigned short int port_from);
3
4  //VGA地址
5  #define VGA_BASE (0xB8000)
6  //行列宽度
7  #define LineLenth (80)
8  #define ColumnLenth (25)
9  //记录读写地址
10 static unsigned char* VGA_Base;
11
12 //记录光标位置
13 static int m_CursorX;
14 static int m_CursorY;
15
16 //设置光标
17 void SetCursor(int x, int y)
18 {
19     unsigned short pos = y * 80 + x;
20     outb(0x3d4, 14);
21     outb(0x3d5, (pos >> 8) & 0xff);
22     outb(0x3d4, 15);
23     outb(0x3d5, pos & 0xff);
24     //同步记录的光标位置
25     m_CursorX = x;
26     m_CursorY = y;
27 }
28
29 //实现滚屏一行
30 void ScrollUp()
31 {
32     for (int line = 0; line < ColumnLenth; line++)
33         for (int i = 0; i < 160; i++)
34             *(VGA_Base + line * 160 + i) = *(VGA_Base + (line + 1) * 160 + i);
35     m_CursorY = 24;
36 }
```

```

38     //VGA输出字符
39 void VGAputchar(unsigned char c, int color)
40 {
41     switch (c)
42     {
43         //特殊字符的处理
44         case '\r':
45         case '\n':
46             m_CursorX = 0;
47             m_CursorY++;
48             break;
49         case '\t':
50             m_CursorX = (m_CursorX / 8 + 1) * 8;
51             if (m_CursorX >= LineLenth)
52             {
53                 m_CursorX -= LineLenth;
54                 m_CursorY++;
55             }
56             break;
57         //正常字符的输出
58     default:
59     {
60         //计算光标位置
61         unsigned int pos = m_CursorY * LineLenth + m_CursorX;
62         //写入字符以及颜色
63         *(VGA_Base + pos * 2) = c;
64         *(VGA_Base + pos * 2 + 1) = color;
65         pos++;
66         m_CursorX = pos % LineLenth;
67         m_CursorY = pos / LineLenth;
68     }
69     break;
70 }
71 //如果行数达到最大则滚屏
72 if (m_CursorY == ColumnLenth)
73     ScrollUp();
74 //同步光标位置
75 SetCursor(m_CursorX, m_CursorY);
76 }
77

```

```

78     //清屏
79     void clear_screen(void)
80     {
81         //初始化记录的地址
82         VGA_Base = (char*)VGA_BASE;
83         unsigned char* p;
84         p = (char*)VGA_BASE;
85         //空格覆盖实现清屏
86         for (int i = 0; i < 1000; i++)
87             VGAputchar(' ', 0);
88         p = (char*)(VGA_Base + 1);
89         *p = 0;
90         //重置记录地址
91         VGA_Base = (char*)VGA_BASE;
92     }
93
94     //VGA输出格式化字符串
95     void append2screen(char* str, int color)
96     {
97         //判断字符串的结尾
98         while (*str != '\0')
99         {
100             VGAputchar(*str, color);
101             str++;
102         }
103     }

```

实现 uart 读写的代码

```

1  extern unsigned char inb(unsigned short int port_from);
2  extern void outb (unsigned short int port_to, unsigned char value);
3
4  //串口地址
5  #define uart_base 0x3F8
6
7  void uart_put_char(unsigned char c){
8      outb (uart_base, c);
9  }
10
11  unsigned char uart_get_char(void){
12      return inb (uart_base);
13  }
14
15  void uart_put_chars(char *str){
16      //判断字符串尾
17      while(*str!='\0'){
18          uart_put_char(*str);
19          str++;
20      }
21  }

```


myprint

```
1  #include <stdarg.h>
2  extern int vsprintf(char* buf, const char* fmt, va_list args);
3  extern void append2screen(char *str,int color);
4  extern void uart_put_chars(char* str);
5
6  char uBuf[400]; //TODO: fix me
7  int myPrintf(int color, const char *format, ...){
8      va_list args;
9      int i;
10     va_start(args, format);
11     //将获取到的fmt格式字符串写入到buf这个缓存里去
12     i = vsprintf(uBuf, format, args);
13     //释放args
14     va_end(args);
15     append2screen(uBuf, color);
16     uart_put_chars(uBuf);
17     return i;
18 }
19
20 char kBuf[400]; //TODO: fix me
21 int myPrintk(int color, const char *format, ...){
22     va_list args;
23     int i;
24     va_start(args, format);
25     //将获取到的fmt格式字符串写入到buf这个缓存里去
26     i = vsprintf(uBuf, format, args);
27     //释放args
28     va_end(args);
29     append2screen(uBuf, color);
30     uart_put_chars(kBuf);
31     return i;
32 }
```

移植的字符串格式化代码的概览

```
1  #include <stdarg.h>
2  #define NULL 0
3
4  +static inline int isdigit(int ch) { ... }
5
6
7  //如果字符串中为数字，则返回数字
8
9  +static int skip_atoi(const char** s) { ... }
10
11
12
13
14
15
16
17
18
19  #define ZEROPAD 1      /* pad with zero */
20  #define SIGN 2        /* unsigned/signed long */
21  #define PLUS 4         /* show plus */
22  #define SPACE 8        /* space if plus */
23  #define LEFT 16        /* left justified */
24  #define SMALL 32       /* Must be 32 == 0x20 */
25  #define SPECIAL 64     /* 0x */
26
27  //这个函数主要用来实现判断是要转化成什么进制数
28  +int __do_div(int n, int base) { ... }
29
30
31
32
33
34
35
36  +void Number(char** str, int date, int base) { ... }
37
38
39
40
41
42
43
44
45  +static char* number(char* str, long num, int base, int size, int precision, int type) { ... }
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146  +int vsprintf(char* buf, const char* fmt, va_list args) { ... }
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241  +int myvsprintf(char* buf, const char* format, ...) { ... }
242
243
244
245
246
247
248  + // ...
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
```

代码布局说明

所有的引导模块将按页（4KB）边界对齐，物理内存地址从 1M 处开始

编译过程说明：

在 Ubuntu 中先搜索到 lab2 的目录，然后通过指令 make 完成编译，可以看到

在 output 目录中的对应目录中分别输出了与根目录下对应文件相同文件。

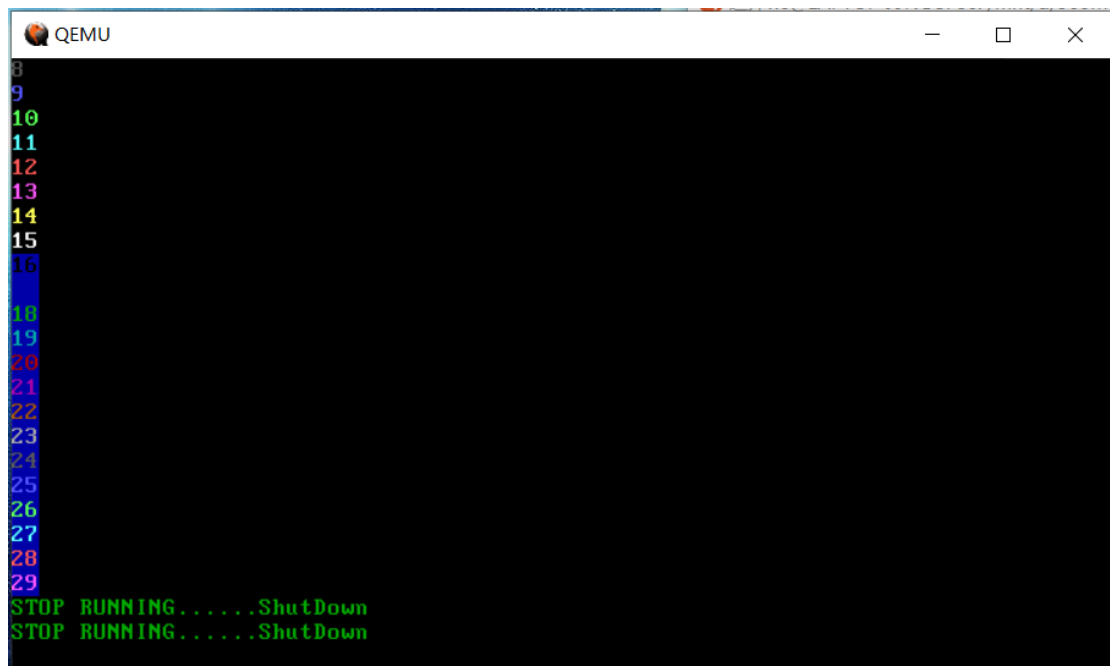
```
ws@LAPTOP-J9NGC786:/mnt/d/360MoveData/Users/asus/Desktop/os2020-labs/lab2/src/2_multiboot2myMain$ make
ld -n -T myOS/myOS.ld output/multibootheader/multibootHeader.o output/myOS/start32.o output/myOS/osStart.o output/myOS/dev/uart.o output/myOS/dev/vga.o output/myOS/i386/io.o output/myOS/printk/myPrintk.o output/myOS/printk/vsprintf.o output/userApp/main.o -o output/myOS.elf
ws@LAPTOP-J9NGC786:/mnt/d/360MoveData/Users/asus/Desktop/os2020-labs/lab2/src/2_multiboot2myMain$ export DISPLAY=:0
ws@LAPTOP-J9NGC786:/mnt/d/360MoveData/Users/asus/Desktop/os2020-labs/lab2/src/2_multiboot2myMain$ qemu-system-i386 -kern
```

multibootheader	2020/3/18 12:5
myOS	2020/3/19 18:4
userApp	2020/3/20 12:4
myOS.elf	2020/3/20 12:4

可以看到，make 命令确保源代码目录下没有不正确的.o 文件以及文件的互相依赖。它们分别链向源代码目录下的真正的 i386 所需要的真正的子目录。编译后产生的文件在图上可见，有 multiboot、start32、osStart、uart、vga、io、myPrintk、vsprintf、main、myOS 的输出文件以及标注了他们的输出目录。

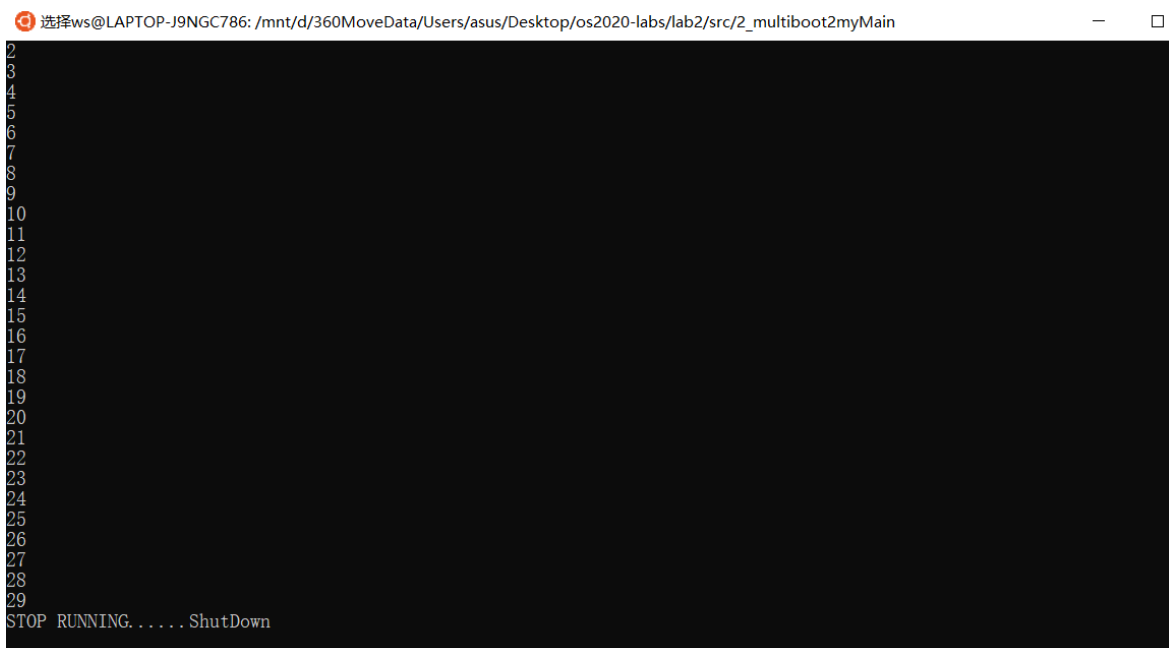
运行和运行结果说明：

在 Ubuntu 中通过 QEMU 启动已经编译生成的 bin 文件，得到 Linux 的图形化界面运行结果，显示需要的输出。其中 VGA 输出结果是 1-29 个数字并以与该数字对应的编码的颜色渲染，同时在前后有执行的开始与结束说明。



```
QEMU
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
STOP RUNNING.....ShutDown
STOP RUNNING.....ShutDown
```

在 linux 命令窗口的显示结果则不带有颜色，但是输出内容与前者相同。



```
选择ws@LAPTOP-J9NGC786: /mnt/d/360MoveData/Users/asus/Desktop/os2020-labs/lab2/src/2_multiboot2myMain
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.....ShutDown
```

遇到的问题和解决方案：

1. 不会自己编写字符串的格式化函数。

通过网络工具移植。

2. 不了解如何输出到 VGA 上。

咨询助教并选择了直接用 C 语言的指针进行修改值

3. 不知道光标的实现方法。

参考老师给出的文件并且查阅网络资料完成。