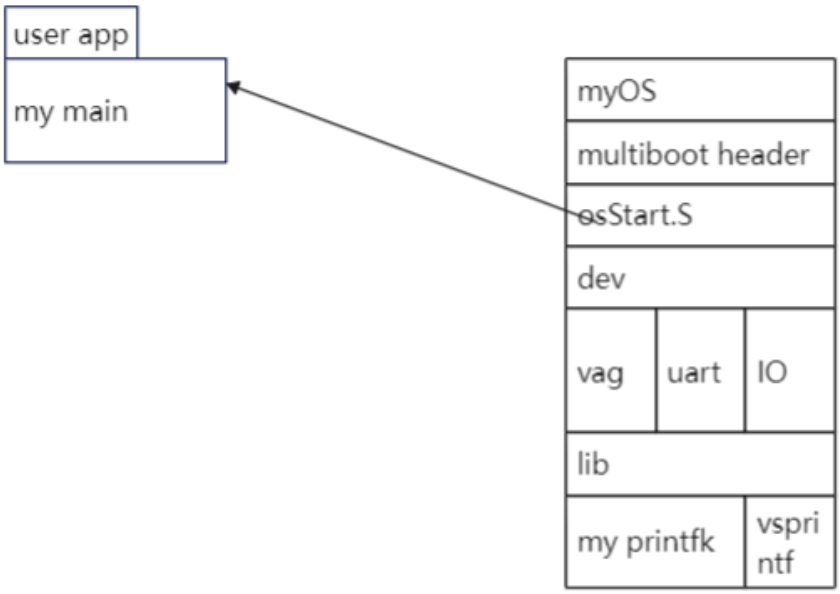
lab2

系统布局

OSapp

> 包含了multi header（给qemu识别）和start进入main函数
>
> lib 库：myprintfk & vsprintf字符串处理
>
> 设备与用户接口 dev
>
> - IO(VGA 和uart 都是基于IO处理的)
>   - VGA
>   - uart

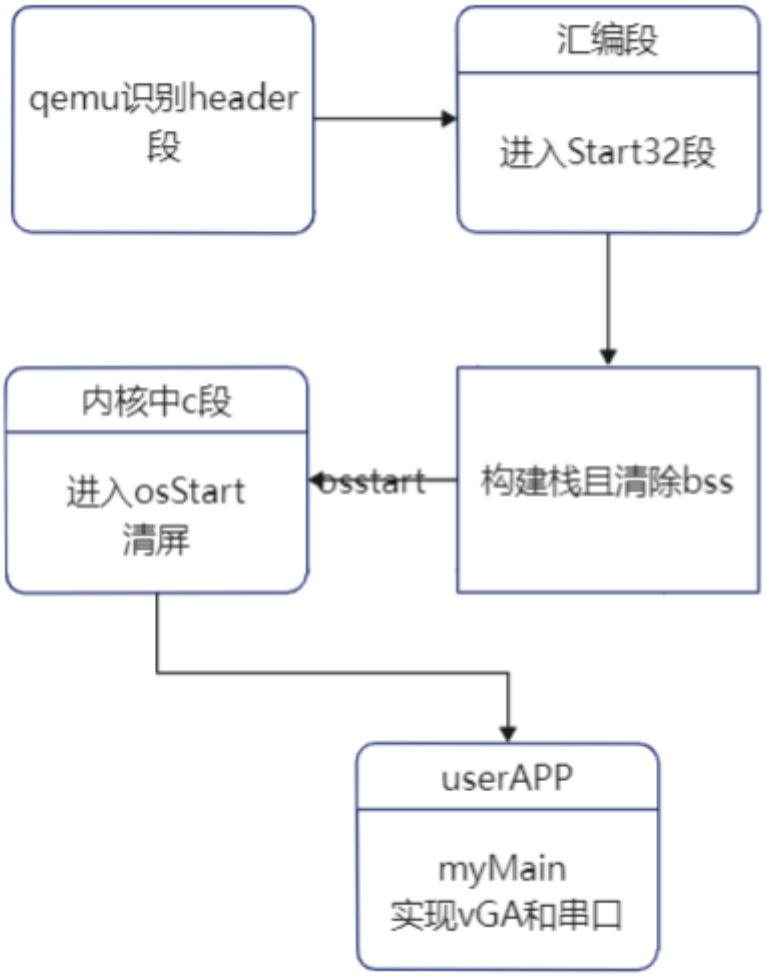


流程实现

qemu识别header段后进入Start32段(汇编段)（data bss等段布局在gcc生成时已完成）

构建栈且清除bss 进入OSstart(C段)， 先清屏再进入mymain函数实现VGA和uart 输出

> multiboot_header→myOS→userApp



主要功能模块实现

I/O实现

```
#include "io.h"

// 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));
}
```

基于内联汇编实现对特定地址（port： Nd）的内存实现访存

VGA实现

> 对显存的写入基于
>
> c语言类型转化
>
> 将int 型变量的地址量 转化为 指向short型变量的指针以达到修改显存地址

```c
#include "io.h"
#include "vga.h"

#define VGA_BASE 0xB8000 // vga 显存起始地址
#define VGA_END  0xB8FA0 // vga 显存结束地址
#define VGA_LAST_LINE  0xB8F00 //vga 显存倒数第一行
#define VGA_SCREEN_WIDTH 80 // vga 屏幕宽度（可容纳字符数）
#define VGA_SCREEN_HEIGHT 25 // vga 屏幕高度

#define CURSOR_LINE_REG 0xE // 高八位
#define CURSOR_COL_REG 0xF // 低八位
#define CURSOR_INDEX_PORT 0x3D4 // 光标行列索引端口号
#define CURSOR_DATA_PORT 0x3D5 // 光标数据端口号


/* 将光标设定到特定位置
 * 提示：使用 outb */
void set_cursor_pos(unsigned int pos) {
    outb(CURSOR_INDEX_PORT,0xE);
    outb(CURSOR_DATA_PORT,(pos>>8));//设定高八位
    outb(CURSOR_INDEX_PORT,0xF);
    outb(CURSOR_DATA_PORT,(pos&0xFF));//设定低八位
}

/* 获取光标当前所在位置
 * 提示：使用 inb */
unsigned int get_cursor_pos(void) {
    int unsigned pos;
    outb(CURSOR_INDEX_PORT,0xE);
    pos=(inb(CURSOR_DATA_PORT)<<8);
    outb(CURSOR_INDEX_PORT,0xF);
    pos+=(inb(CURSOR_DATA_PORT));
    return pos;
}

/* 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 */
//最后一行内容写入前一行后附 foregroud 白 background 黑
void scroll_screen(void) {
    unsigned long int* cl=(unsigned long int*)VGA_BASE;
    do{
            *cl=*(cl+40);
    }while((int)(cl++)!=VGA_LAST_LINE);
    cl=(unsigned long int*)VGA_LAST_LINE;
    do{*cl=0x0F000F00;}while((int)(cl+=1)!=VGA_END);
}

/* 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 int pos) {
    short int *p;
    p=(short int *)(pos*2+0xB8000);
    *p=(short int)c+(color<<8);
}
//将偏移量和VGA_BASE相加得到VGA显存输出位置

/* 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) {
    unsigned long int* cl=(unsigned long int*)VGA_BASE;
    do{*cl=0x0F000F00;}while((int)(cl+=1)!=VGA_END);
    set_cursor_pos(0);

}
//清屏时为显存赋初值foregroud 白 background 黑


/* 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) {
    unsigned int pos=get_cursor_pos();
    while (*str!='\0')
    {
        if(*str=='\n'){
            if(pos>=(VGA_LAST_LINE-0xB8000)/2){scroll_screen();pos=
(VGA_LAST_LINE-0xB8000)/2;}
            else{pos=(pos/80+1)*80;}
            // set_cursor_pos(pos);
            str++;
            continue;
        }
        else{put_char2pos(*str,color,pos);}
        str++;
        pos+=1;
        if(pos==(VGA_END-0xB8000)/2){scroll_screen();pos=((VGA_LAST_LINE-
0xB8000)/2);}
    }
    set_cursor_pos(pos);
}
```

串口实现

> 通过outb向0x3F8输入字符即可

```
#include "io.h"
#include "uart.h"

#define UART_PORT 0x3F8 // 串口端口号

/* 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) {
    outb(UART_PORT,ch);
}

// 向串口输出一个字符串
void uart_put_chars(char *str) {
    while(*str!='\0'){
        uart_put_char(*str);
        str++;
    }
}
```

字符串处理实现

> 该部分太长 不贴全
>
> 仅解释重要部分

```
static size_t strnlen(const char *s, size_t count);
//实现对字符串大小识别
static int skip_atoi(const char **s);
//将字符串数字连接起来（无大小），转化为有大小的int型变量
static char *number(char *str, long num, int base, int size, int precision, int
type);
//进制转化
int vsprintf(char *buf, const char *fmt, va_list args);
//*buf格式化处理后的字符串 *fmt函数用户定义输出格式 args基于格式需调用的参数
//通过寻找 % 后的标识符 对参数列表进行相应的处理后与fmt 写入buf
```

源代码组织

目录组织

```
.
├── makefile
├── multibootheader
│   └── multibootHeader.S
├── myOS
│   ├── dev
│   │   ├── Makefile
│   │   ├── uart.c
│   │   └── vga.c
│   ├── i386
│   │   ├── io.c
│   │   └── Makefile
│   ├── include
│   │   ├── io.h
│   │   ├── myPrintk.h
│   │   ├── uart.h
│   │   ├── vga.h
│   │   └── vsprintf.h
│   ├── lib
│   │   ├── Makefile
│   │   └── vsprintf.c
│   ├── Makefile
│   ├── myOS.ld
│   ├── osStart.c
│   ├── printk
│   │   ├── Makefile
│   │   └── myPrintk.c
│   ├── start32.S
│   └── userInterface.h
├── output
│   ├── multibootheader
│   │   └── multibootHeader.o
│   ├── myOS
│   │   ├── dev
```

```
|   |   |   ├── uart.o
|   |   |   └── vga.o
|   |   ├── i386
|   |   |   └── io.o
|   |   ├── lib
|   |   |   └── vsprintf.o
|   |   ├── osStart.o
|   |   ├── printk
|   |   |   └── myPrintk.o
|   |   └── start32.o
|   ├── myOS.elf
|   └── userApp
|       └── main.o
├── source2img.sh
├── tree
└── userApp
    ├── main.c
    └── Makefile

16 directories, 35 files
```

makefile组织

```
├── myOS.elf
   ├── multibootheader
   |   └── multibootHeader.o
   |       └── multibootHeader.s
   ├── myOS
   |   ├── dev
   |   |   ├── uart.o
   |   |   |   └── uart.c
   |   |   └── vga.o
   |   |       └── vga.c
   |   ├── i386
   |   |   └── io.o
   |   |       └── io.c
   |   ├── lib
   |   |   └── vsprintf.o
   |   |       └── vsprintf.c
   |   ├── osStart.o
   |   |   └── osStart.c
   |   ├── printk
   |   |   └── myPrintk.o
   |   |       └── myPrintk.c
   |   └── start32.o
   |       └── start32.s
   └── userApp
      └── main.o
          └── main.c
```

代码布局

由于添加了-g参数增加debug和符号表等段

```
There are 19 section headers, starting at offset 0x31f8:
```

```
Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00100000 000080 000bf4 00  AX  0   0  4

  //在ld为文件中将header和text合并成了text段 真实的text段起始于16 由于align 8 按八字节对齐
header一共12个字节基于8字节对齐
  //即在16地址处开始
  [ 2] .text.__x86.get_p PROGBITS        00100bf4 000c74 000004 00  AX  0   0  1

  [ 3] .text.__x86.get_p PROGBITS        00100bf8 000c78 000004 00  AX  0   0  1
  [ 4] .text.__x86.get_p PROGBITS        00100bfc 000c7c 000004 00  AX  0   0  1
  [ 5] .eh_frame         PROGBITS        00100c00 000c80 0002c4 00   A  0   0  4
  [ 6] .rodata           PROGBITS        00100ec4 000f44 00012e 00   A  0   0  4
  [ 7] .data             PROGBITS        00101000 001080 000008 00  WA  0   0  4
  [ 8] .got.plt          PROGBITS        00101008 001088 00000c 04  WA  0   0  4
  [ 9] .bss              NOBITS          00101020 001094 000330 00  WA  0   0 32
  [10] .debug_line       PROGBITS        00000000 001094 00087e 00      0   0  1
  [11] .debug_info       PROGBITS        00000000 001912 00079d 00      0   0  1
  [12] .debug_abbrev     PROGBITS        00000000 0020af 00043e 00      0   0  1
  [13] .debug_aranges    PROGBITS        00000000 0024f0 000120 00      0   0  8
  [14] .debug_str        PROGBITS        00000000 002610 000318 01  MS  0   0  1
  [15] .comment          PROGBITS        00000000 002928 00002a 01  MS  0   0  1
  [16] .symtab           SYMTAB          00000000 002954 000530 10     17  57  4
  [17] .strtab           STRTAB          00000000 002e84 00028e 00      0   0  1
  [18] .shstrtab         STRTAB          00000000 003112 0000e4 00      0   0  1
```

编译过程说明

编译所用指令

```
gcc -m32 --pipe -Wall -fasm -g -O1 -fno-stack-protector
gcc -m32 -fno-stack-protector -fno-builtin -g
```

编译大致过程

编译汇编文件和c语言文件形成可重定向的二进制文件，再通过ld 命令将可重定向文件重新组织链接形成elf文件

运行结果

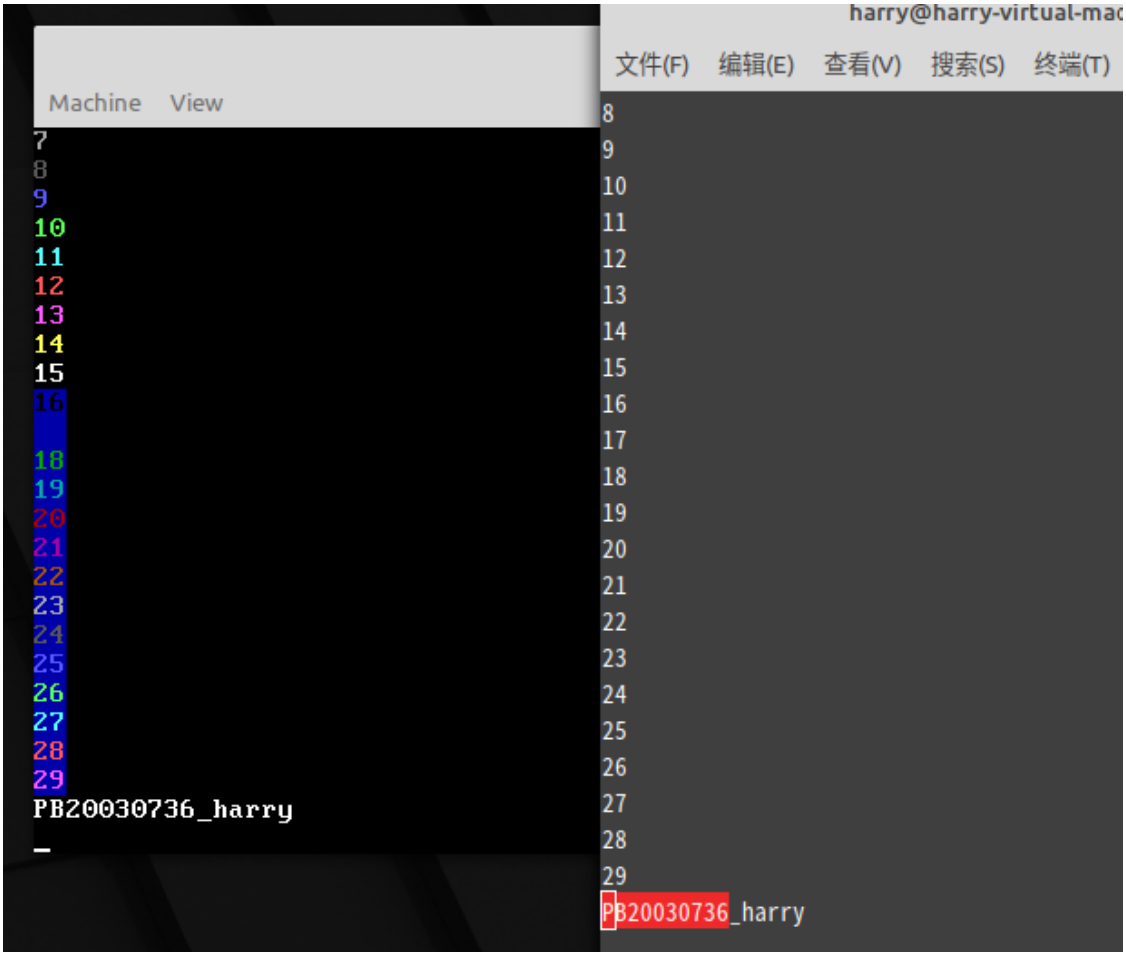
运行过程

```
./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
```

进入qemu

运行结果


Machine View
7
8
9
10
11
12
13
14
15
16
18
19
20
21
22
23
24
25
26
27
28
29
PB20030736_harry
harry@harry-virtual-mac
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T)
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
PB20030736_harry