

实验 2： Multiboot2myMain

目录

- 软件结构说明
- 主流程说明
- 功能模块说明
- 源代码说明
- 地址空间说明
- 编译过程说明
- 运行及结果说明
- 遇到的问题及解决

软件框图

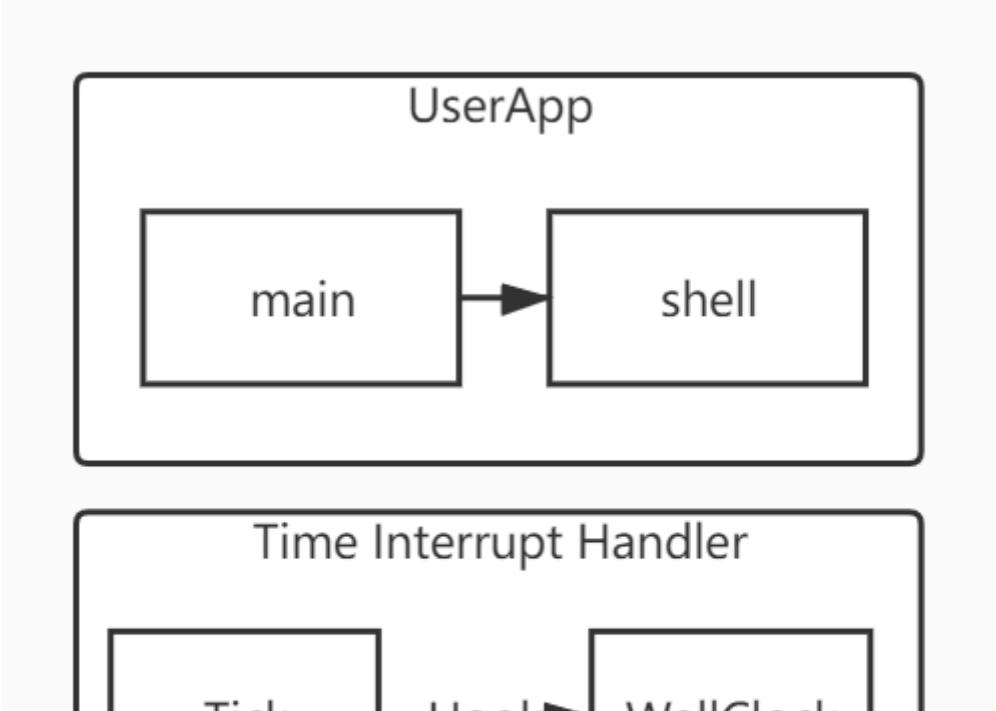
- 软件结构

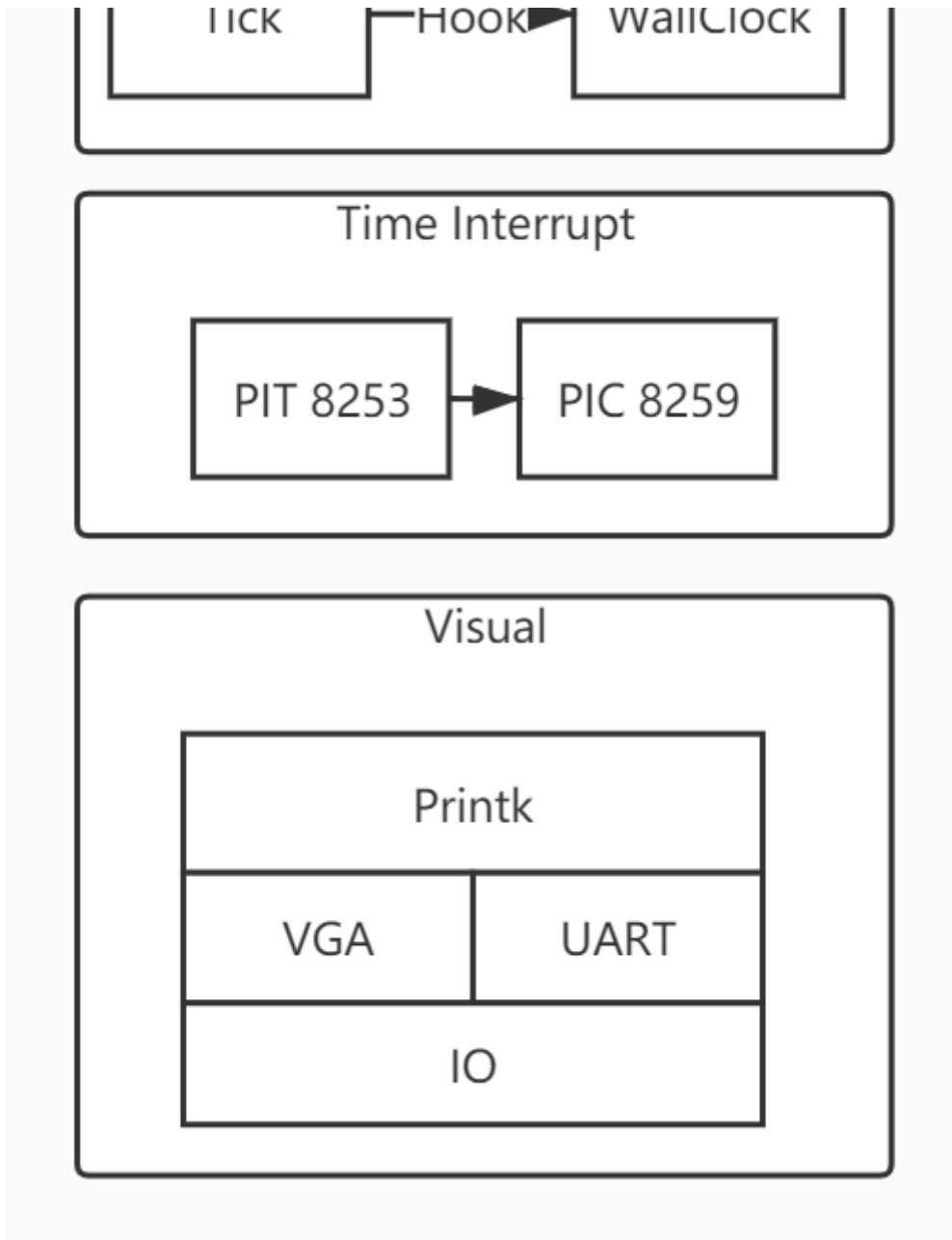
本次实验主要开发中断功能和命令行功能。

中断功能可分为中断的发生（PIT）， 中断的捕获（PIC）， 中断的处理（Tick&WallClock）， 通过Hook机制实现。

命令行功能依赖于格式化输出模块和字符串处理。

- 结构图（底层在下， 顶层在上）：



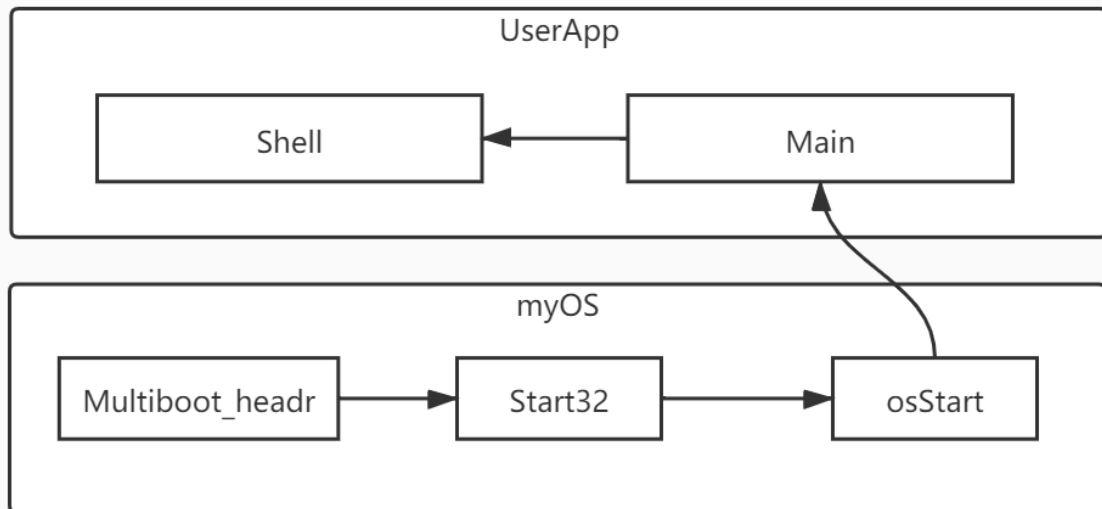


主流程

- 流程说明

主流程从Multiboot_header开始，首先进入Start32。在Start32中，程序进行了堆栈的初始化、IDT的初始化等必要的准备工作，然后将控制权移交到osStart。在osStart中，进行PIT、PIC的初始化，并开启中断，之后调用用户程序入口函数Main。Main调用Shell的开启子程序，进入控制台。

- 流程图



功能模块

概述

软件的功能模块主要有格式化输出模块，中断控制模块，中断处理模块和命令行模块。以下是各个模块的详细说明，由于之前的实验已经对格式化输出模块进行了足够说明，故不再列出。

中断控制模块

- 开辟IDT的内存空间(256*8)

```
.data
# IDT
.p2align 4
.globl IDT
IDT:
.rept 256
.word 0,0,0,0
.endr
idtptr:
.word (256*8 - 1)
.long IDT
```

- 设置IDT中的条目

```
setup_idt:
    movl $ignore_int1,%edx /* default handler */
    movl $0x00080000,%eax /* selector */
    movw %dx,%ax
    movw $0x8E00,%dx /* interrupt gate - dpl=0, present */
    movl $IDT,%edi
```

```

        mov $256,%ecx
rp_sidt:
        movl %eax,(%edi)
        movl %edx,4(%edi)
        addl $8,%edi
        dec %ecx
        jne rp_sidt
setup_time_int_32:
        movl $time_interrupt,%edx /* time interrupt handler */
        movl $0x00080000,%eax /* selector: 0x0010 = cs */
        movw %dx,%ax
        movw $0x8E00,%dx /* interrupt gate - dpl=0, present */
        movl $IDT,%edi
        addl $(32*8), %edi
        movl %eax,(%edi)
        movl %edx,4(%edi)
        ret

```

- 初始化8259 (PIC)

```

void init8259A(void){

    /* master chip ICW1~ICW4 */
    outb(0x20, 0x11);
    io_wait();
    outb(0x21, 0x20);
    io_wait();
    outb(0x21, 0x04);
    io_wait();
    outb(0x21, 0x03);
    io_wait();

    /* slave chip ICW1~ICW4 */
    outb(0xA0, 0x11);
    io_wait();
    outb(0xA1, 0x28);
    io_wait();
    outb(0xA1, 0x02);
    io_wait();
    outb(0xA1, 0x01);
    io_wait();

    /* set mask to enable time interrupt */
    char value = inb(0x21);
    outb(0x21, value &= ~1);
}

```

- 初始化8253 (PIT)

```

void init8253(void){
    /* 1,193,180 Hz to 8253 */
    unsigned short fq = 11932;//100Hz
}

```

```

    unsigned char high,low;

    high = fq>>8;
    low = fq;
    outb(0x43,0x34);//set control byte
    io_wait();
    outb(0x40,low);//set high 8 bits
    io_wait();
    outb(0x40,high);// set low 8 bits
    io_wait();
}

```

- 开/关中断

```

enable_interrupt:
    sti
    ret

disable_interrupt:
    cli
    ret

```

中断处理模块

- 中断处理代码块

```

.p2align 4
ignore_int1:
    cld
    pusha
    call ignoreIntBody
    popa
    iret

.p2align 4
time_interrupt:
    cld
    pushf
    pusha
    call tick
    popa
    popf
    iret

```

- ignoreIntBody （默认中断处理子程序）

```

void ignoreIntBody(void){
    put_chars("Unknown interrupt1\0",0x4,24,0);
}

```

- tick (hook机制实现)

```
#define SIZE 256
int ticks=0;
typedef void(*fun)(void); // typedef hook function
fun list[SIZE]; // hook function list
int cnt=0; // hook counter

// add hook function to function list
void set_timer_hook(void (*func)(void)){
    list[cnt++]=func;
}

// call all hook functions
void timer_hook_parse(void){
    for(int i=0;i<cnt;i++) list[i]();
}

// called by time interrupt handler
void tick(void){
    ticks++;
    timer_hook_parse();
}
```

- wallClock (hook机制实现)

```
void maybeUpdateWallClock(void)
{
    if(ticks%100) return;
    ss=(ss+1)%60;
    if(!ss) mm=(mm+1)%60;
    if(!mm && !ss) hh=(hh+1)%24;
    putBottomRight(0x7, "%2d : %2d : %2d",hh,mm,ss);
}

void setWallClock(int h, int m, int s)
{
    if(h<0 || h>23) return;
    if(m<0 || m>59) return;
    if(s<0 || s>59) return;

    hh=h,mm=m,ss=s;
    set_timer_hook(maybeUpdateWallClock);
    putBottomRight(0x7, "%2d : %2d : %2d",hh,mm,ss);
}

void getWallClock(int *h, int *m, int *s)
{
    *h = hh;
    *m = mm;
    *s = ss;
}
```

命令行模块

- 读入命令

```
// 通过uart读入命令
static char buf[MAXSIZE];
char* read(void){
    for(int i=0;i<MAXSIZE;i++) buf[i]=0;//init buffer
    char c;
    int cursor = 0;
    /* read from uart */
    while(cursor<MAXSIZE-1 && (c=uart_get_char())!=13){
        uartPrintf("%c",c);
        switch (c){
            /* backspace */
            case 127:
                if(cursor>0) buf[--cursor]=0;
                break;
            default:
                buf[cursor++]=c;
                break;
        }
        show_input(0x7,buf);
    }
    myPrintk(0x7,"%s\n",buf);
    return buf;
}
```

- 参数提取

```
struct param{
    int argc;
    char args[MAXARGS][MAXARGLEN];
    char *argv[MAXARGS];
}parameters;

// parse raw command and extract parameters
void parseCmdStr(char* str){

    int p=0;
    int pp=0;
    int len=strlen(str);
    parameters.argc=0;
    while(p<len){
        if(str[p]==' '){
            parameters.argv[parameters.argc]=parameters.args[parameters.argc];
            parameters.args[parameters.argc++][pp]=0;
            p++;
            pp=0;
            continue;
        }

        parameters.args[parameters.argc][pp++]=str[p++];
    }
}
```

```

    }
    parameters.argv[parameters argc]=parameters.args[parameters argc];
    parameters.args[parameters argc++][pp]=0;

}

```

- 命令处理主函数

```

// 主命令处理函数
int handler(int argc, char **argv)
{
    if(strlen(argv[0])==0) return 0;
    int i;
    for(i=0;strlen(cmds[i].cmd)!=0;i++){
        if(streq(argv[0],cmds[i].cmd)){
            cmds[i].func(argc,argv);
            return i;
        }
    }
    myPrintk(0x7,"Unknown Command\n");
    return 0;
}

```

- 命令处理分支函数

```

// help 的帮助
void help_help(void)
{
    myPrintk(0x7,"%s\n",cmds[1].desc);
}

// help 命令处理函数
int help_handler(int argc, char *argv[])
{
    int i;
    for(i=0;strlen(cmds[i].cmd)!=0;i++){
        if(streq(argv[1],cmds[i].cmd)){
            if(cmds[i].help_func) cmds[i].help_func();
            else myPrintk(0x7,"Help: No Help Information\n");
            return i;
        }
    }
    myPrintk(0x7,"Help: Unknown Command\n");
    return 0;
}

// cmd 命令处理函数
int cmd_handler(int argc, char **argv){
    int i;
    for(i=0;strlen(cmds[i].cmd)!=0;i++){
        myPrintk(0x7,"%s: %s\n",cmds[i].cmd,cmds[i].desc);
    }
}

```



```

        return i;
    }

    // cls 命令处理函数
    int cls_handler(int argc, char **argv){
        clear_screen();
        return 1;
    }

```

- 字符串处理函数

```

// get Length of a string
int strlen(char* str){
    int i;
    for(i=0;str[i]!=0;i++);
    return i;
}

// return whether str1 equals str2 (1=equal,2=not)
int streq(char* str1, char* str2){
    int len1,len2;
    len1=strlen(str1);
    len2=strlen(str2);
    if(len1!=len2) return 0;
    for(int i=0;i<len1;i++){
        if(str1[i]!=str2[i]) return 0;
    }
    return 1;
}

```

- 命令行入口函数

```

void startShell(void)
{
    while(1){
        myPrintk(0xa,"shell@myos:");
        char* cmd;
        cmd=read();
        parseCmdStr(cmd);
        handler(parameters.argc,parameters.argv);
    }
}

```

源代码说明

- 代码组织

```

|---- lab3/
|---- src/

```

```

|---- source2img.sh      生成elf脚本
|---- myOS/
|    |---- start32.S
|    |---- osStart.c
|    |---- dev/
|        |---- i8253.c
|        |---- i8259A.c
|        |---- uart.c
|        |---- vga.c
|    |---- i386/
|        |---- io.c
|        |---- io.h
|        |---- irqs.c
|    |---- kernel/
|        |---- tick.c
|        |---- wallClock.c
|    |---- printk/
|        |---- myPrintk.c
|        |---- vsprintf.c
|---- userApp/
|    |---- main.c
|    |---- shell.c
|---- multibootHeader/
|    |---- multibootHeader.S

```

- Makefile 组织

```

include $(SRC_RT)/myOS/Makefile
include $(SRC_RT)/userApp/Makefile

```

```

|---- lab3/
|    |---- src/
|        |---- myOS/
|            |---- dev/
|            |---- i386/
|            |---- kernel/
|            |---- printk/
|---- userApp/

```

地址空间说明

- ld文件

```

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);
}

```

- 地址空间表

Offset	Field	Macro
0	.code	
1M	.text	
ALIGN(16)	.data	
ALIGN(16)	.bss	__bss_start, _bss_start
		_bss_end
ALIGN(16)		_end

编译过程说明

- 主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 ${C_FLAGS} -c -o $@ $<

```

- 说明

根据Makefile分为两步：编译和链接。

第一步，编译汇编代码(*.S)和c代码(*.c)并输出对象文件(*.o)。

第二步，将这些对象文件链接并输出可执行可链接文件(myOS.elf)。

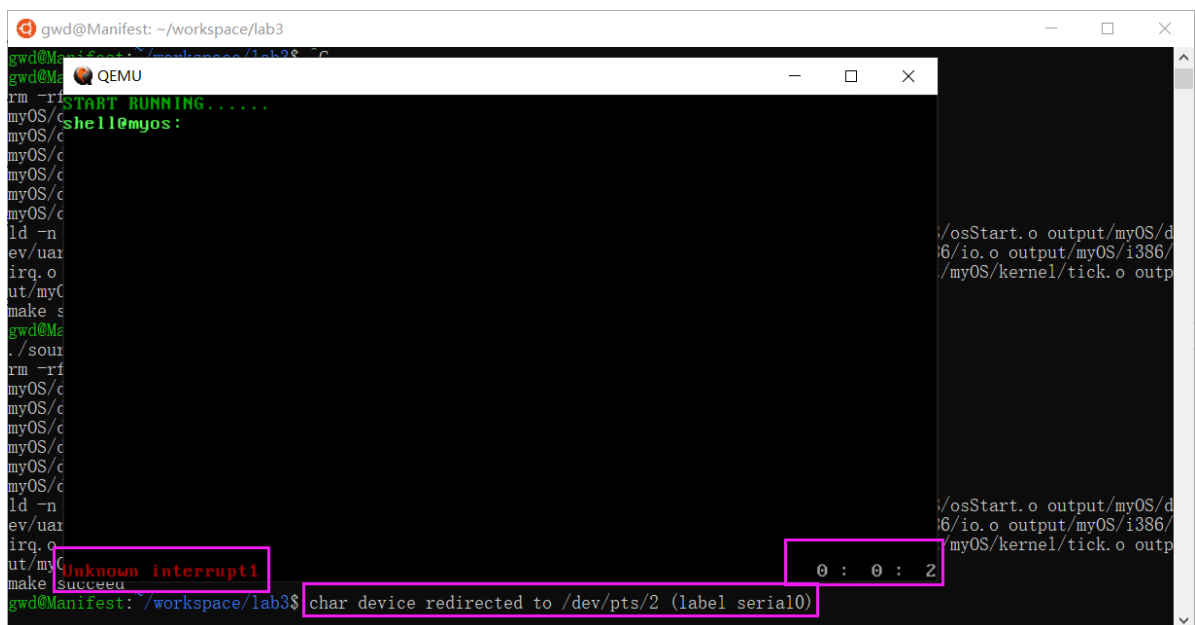
运行和运行结果说明

- 运行

执行命令：`qemu-system-i386 -kernel output/myOS.elf -serial pty &`

将之前编译链接生成的elf文件，加载到qemu中运行。

- 运行结果



```
gwd@Manifest: ~/workspace/lab3
gwd@Manifest:~/workspace/lab3$ qemu-system-i386 -kernel output/myOS.elf -serial pty &
QEMU
START RUNNING.....
shell@myos:
myos/c
myos/c
myos/c
myos/c
myos/c
myos/c
myos/c
ld -n
ev/uar
irq.o
ut/myC
make s
gwd@Manifest:~/workspace/lab3$ ./sour
rm -rf
myos/c
myos/c
myos/c
myos/c
myos/c
myos/c
ld -n
ev/uar
irq.o
ut/myC
make s
gwd@Manifest:~/workspace/lab3$ char device redirected to /dev/pts/2 (label serial0)
```

可以看到，中断功能和时间显示都正常。

