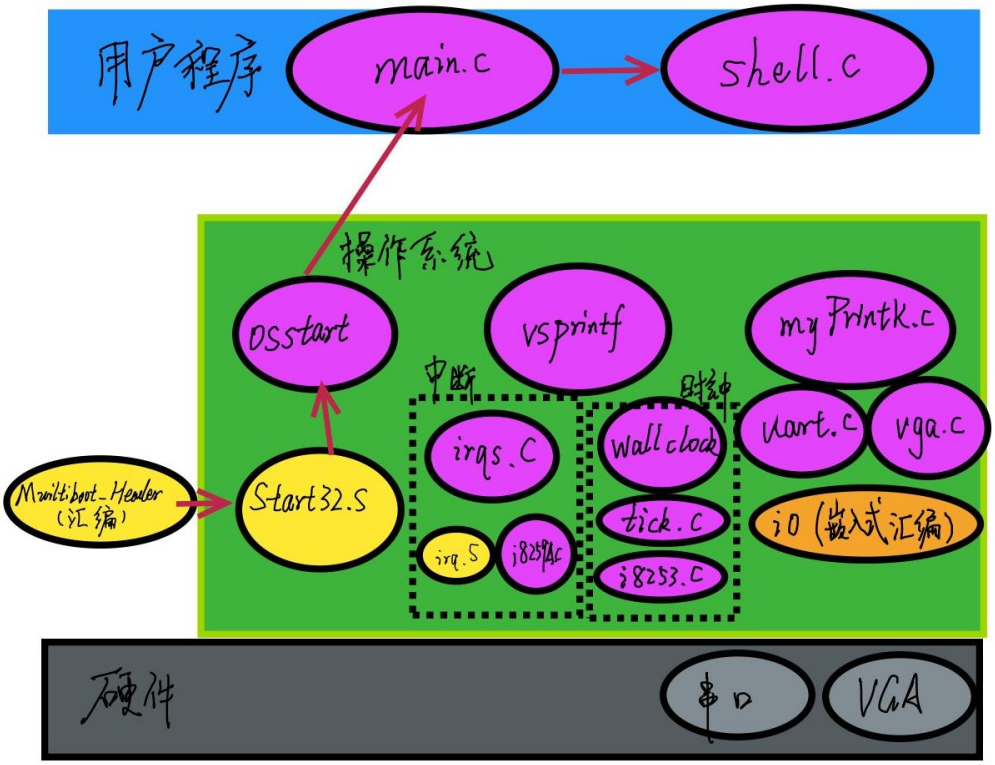
操作系统实验报告

实验三　shell&Interrupt&timer

学号：PB18111683
姓名：童俊雄
完成时间：2020-03-21

一、　软件框图

紫色为C程序，黄色为汇编程序，橙色为嵌入式汇编



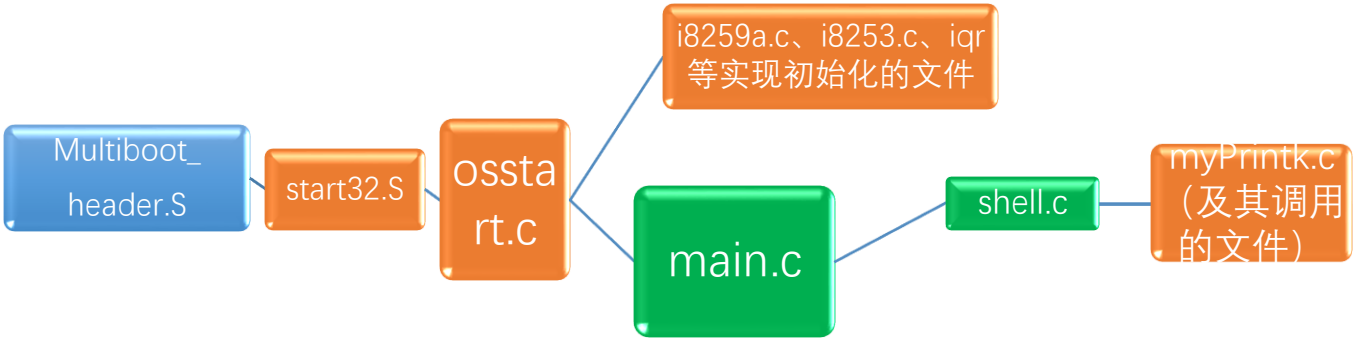
概述：软件大体上可以分为两个层次：用户程序和操作系统，其中操作系统又可以分为与用户程序的接口、IO设备的驱动程序、中断控制程序和时钟功能程序等各功能模块，在每块中又可以进一步划分更细的层次（越上方的层次越高），如图所示。

二、　主流程说明

1. qemu启动header；
2. header通过调用myOS提供的_start，跳转到汇编文件start32.S；
   （进入myOS）
3. 从start32调用C语言入口，进入到c程序osstart.c；
4. osstart.c调用i8259A和i8253函数进行初始化，调用enable_interrupt函数启用中断；并调用用户程序的接口，即myMain函数，从而执行用户程序main.c；
   （进入UserApp）
5. main.c调用shell接口；
6. shell使用myprintk.c中的函数实现交互;

流程图：

（橙色表示myOS内的程序，绿色表示用户程序）



三、 主要功能模块说明&源代码说明

1. 中断模块

中断的主要功能函数有：

a) iqrs.c中的IgnoreIntBody函数，用于提示其他中断，遇到非时钟中断时显示“Unknown interrrput”，通过put_chars函数实现；

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

```
void put_chars(char* str, int color, int line, int loc) {
    int addr = vga_base + (line * buffer_width + loc) * 2;
    int i = 0;
    while (str[i] != '\0') {
        char c = str[i];
        __asm__ __volatile__("movb %0,(%1)"::"a"(c), "b"(addr++));      //在对应地址写入字符
        __asm__ __volatile__("movb %0,(%1)"::"a"(color), "b"(addr++));  //写入颜色
        i++;
    }
}
```

IgnoreIntBody在start32.S中含有接口，start32.S通过接口调用上述函数；

```
.p2align 4
ignore_int1:
        cld
        pusha
        call    ignoreIntBody
        popa
        iret
```

b) iqr.S中的enable_interrupt和disable_interrupt函数；

```
enable_interrupt:
    sti
    ret

disable_interrupt:
    cli
    ret
```

在osstart中调用之；

c) i8259A函数，用于可编程中断控制器的初始化，通过调用IO中的outb和inb函数实现控制器端口的读写：

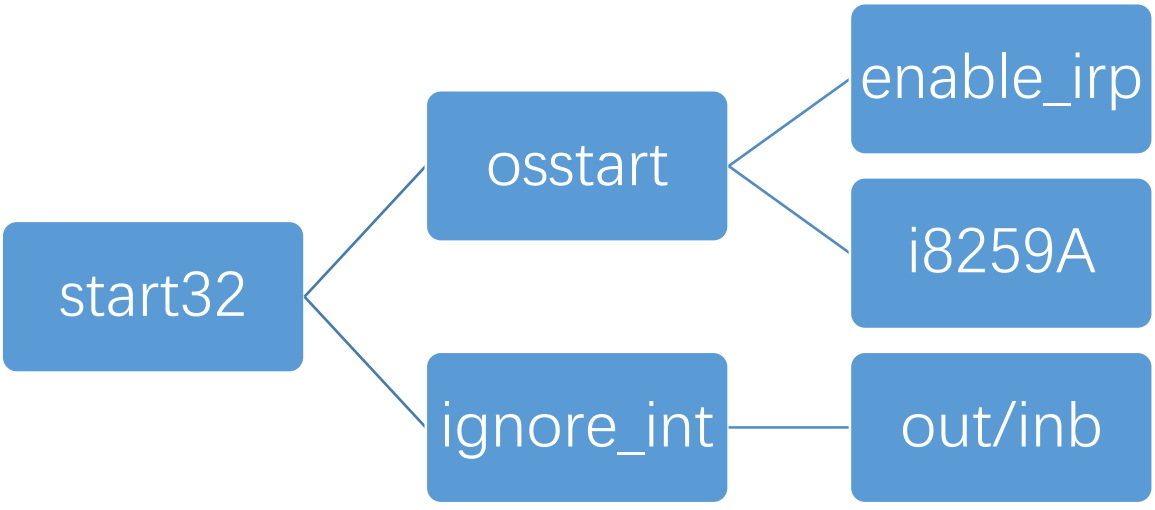
```
void init8259A(void){

//FP
    outb(0x20, 0x11);
    io_wait();
    outb(0x21, 0x20);
    io_wait();
    outb(0x21, 0x04);
    io_wait();
    outb(0x21, 0x03);
    io_wait();
```

```
void io_wait(void)
{
    __asm__ __volatile__( "jmp 1f\n\t"
                          "1:jmp 2f\n\t"
                          "2:" );
}
```

以上为主片的赋参，从片过程与之类似；

中断模块流程图：



2. 时钟模块

时钟部分的代码有：

a) i8253.c，用于PIT的初始化：

```
void init8253(void){
    unsigned short para = 11932;//分频参数
    unsigned char high,low;
    high = (para>>8)&0xff ;
    low = para & 0xff;
    myPrintk(0x7, "%d,%d\n",high,low);
    outb(0x43,0x34);
    io_wait();
    outb(0x40,low);
    io_wait();
    outb(0x40,high);
    io_wait();
}
```

b) tick.c，由start32中的接口“call tick”调用，每次调用时全局变量+1：

```
void tick(void) {
    ticks++;
    timer_hook_parse();
}
```

c) Wallclock.c：用于时钟的维护和读取：

设置初始值为hms：

```
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;//超出范围

        hh=h,mm=m,ss=s;
        set_timer_hook(maybeUpdateWallClock);
        putf(0x20, "%02d : %02d : %02d",hh,mm,ss);
}
```

维护（直接在其中调用putf函数输出时间）：

```
void maybeUpdateWallClock(void)
{
        if(ticks%100) return;
        //100次tick记为1秒
        ss=(ss+1)%60;
        if(!ss) mm=(mm+1)%60;
        if(!mm && !ss) hh=(hh+1)%24;
        //计时
        putf(0x20, "%02d : %02d : %02d",hh,mm,ss);
}
```

读取当前时间：

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

3. shell模块

main调度的函数：

```
void startShell(void)
{
    while (1) {
        myPrintk(0xa, "TongJunxiong->");
        char* cmd;
        cmd = getcmd();//读取命令
        docmd(cmd);//翻译命令
        handler(parameters.argc, parameters.argv);//执行命令
    }
}
```

各个handler：

help命令处理：

```
int help_handler(int argc, char* argv[])
{
    int i;
    for (i = 0;strlen(cmds[i].cmd) != 0;i++) {
        if (strcmp(argv[1], cmds[i].cmd) == 0) {
            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;
}
```

判断命令类别：

```
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 ((strcmp(argv[0], cmds[i].cmd))==0)
            cmds[i].func(argc, argv);
            return i;
        }
    }
    myPrintk(0x7, "Unknown Command\n");
    return 0;
```

四、 目录组织

所有文件：

```
Makefile
README.txt
multibootheader
└── multibootHeader.S
myOS
├── Makefile
├── dev
│   ├── Makefile
│   ├── i8253.c
│   ├── i8259A.c
│   ├── uart.c
│   └── vga.c
├── i386
│   ├── Makefile
│   ├── io.c
│   ├── io.h
│   ├── irq.S
│   └── irqs.c
├── kernel
│   ├── Makefile
│   ├── tick.c
│   └── wallClock.c
├── lib
│   ├── Makefile
│   └── string.c
├── myOS.ld
├── osStart.c
├── printk
│   ├── Makefile
│   ├── myPrintk.c
│   └── vsprintf.c
└── start32.S
source2img.sh
userApp
├── Makefile
├── main.c
└── shell.c
```

Makefile组织：

见上图中的各Makefile文件

五、 代码布局

由myOS.ld的代码可知，myOS.elf文件中有三个 section：

1. 第一个section为.text，位置从1M处开始，在.text内的分布为8字节对齐，前12字节为魔术，从第16字节开始是代码部分；
   代码结束后16位对齐；
2. 第二个section为.data，位置从.text结束并对齐后开始；
   末尾16位对齐；
3. 第三个section为.bss，位置从.data结末尾对齐后开始；
   末尾16位对齐；
4. .bss结束后是_end，此处是堆空间的开始，512位对齐；

六、 编译过程说明

由makefile可知，编译过程有以下两步：

1. 编译汇编代码（header.S和start32.S）和C代码（osstart.c等）生成.o文件；
2. 根据myOS.ld的部署要求，把上述.o文件链接成myOS.elf文件
   如下图所示：

```
ld -n -T myOS/myOS.ld output/multibootheader/multibootHeader.o output/myOS/start32.o output/myOS/osStart.o output/myOS/dev/u
art.o output/myOS/dev/vga.o output/myOS/dev/i8253.o output/myOS/dev/i8259A.o output/myOS/i386/io.o output/myOS/i386/irq.o ou
tput/myOS/i386/irqs.o output/myOS/printk/myPrintk.o output/myOS/printk/vsprintf.o output/myOS/kernel/tick.o output/myOS/kern
el/wallClock.o output/userApp/main.o output/userApp/shell.o -o output/myOS.elf
```

七、 运行和运行结果说明

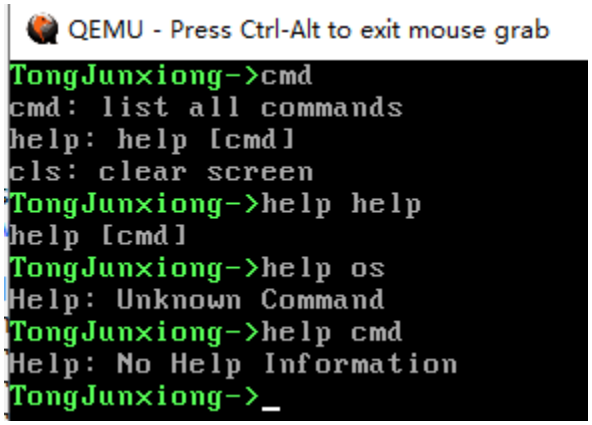
运行指令qemu-system-i386 -kernel output/myOS.elf -serial stdio
运行结果如下如所示：

1. 不明中断的输出：

2. 时间显示：

3. shell交互：

***注：命令输入方式为串口输入



八、 遇到的问题和解决办法

问题：不了解hook机制
解决：查阅资料，并通过求助同学，解决了这部分问题，对hook有了一点点了解