

Lab3实验文档

实验目标

- 实现简单的 shell 程序，至少提供 cmd 和 help 命令，允许注册新的命令
- 实现中断机制和中断控制器 i8259A 初始化
- 实现时钟 i8253 和周期性时钟中断
- 实现 VGA 输出的调整：
 - 时钟中断之外的其他中断，一律输出“Unknown interrupt”。
 - 右下角：从某个时间开始，大约每秒更新一次，格式为：HH:MM:SS。

程序阅读流程：

```
multibooheader/multibootHeader.S → myOS/start32.S → myOS/osStart.c →  
userApp/startShell.c
```

任务

1.myOS/start32.S 中的 time_interrupt 和 ignore_int1 的填写

在该模块中你需要用到的汇编语言：

- cld: 将标志寄存器的方向标志位 DF 清零
- pushf: 将标志寄存器(Flag)中的值存入栈中
- popf: 将栈中的内容存入标志寄存器
- pusha: 将通用寄存器(eax, ebx 等)中的值存入栈中
- popa: 将栈中的内容存入通用寄存器中
- call: 函数调用(可以调用 C 语言程序中的函数)
- iret: 返回调用该函数的函数

关于 call 和 iret: 本质上 call 和 iret 就是 CS 和 IP 寄存器的入栈和出栈，对于 CS 和 IP 寄存器的理解可以对应你们计组课上的 PC(Program Counter, 程序计数器)

如何编写这两个函数？

1. 利用 push 来进行现场保护
2. call 相应的函数(可以调用 C 语言程序中的函数)
3. 利用 pop 来进行恢复现场
4. 利用 iret 来返回调用该函数的函数

2.myOS/dev/i8253.c 和 myOS/dev/i8259A.c 的填写

i8253 和 i8259A 都是可编程芯片，什么叫可编程逻辑芯片？就意味着它可以在配置后进行使用，至于如何配置他们呢？我们只需要用 outb 函数往相应的地址输出相应的数值即可(ppt 中有详细的配置地址与相应配置数值说明)

关于 i8253 和 i8259A 的工作方式说明：

在配置好 i8253 和 i8259A 后，i8253 就相当于一个特定频率的时钟源，而且输出的时钟信号就当作中断信号挂载在 i8259A 上，i8259A 作为一个中断控制器，会使 CPU 去执行相应中断号的中断子程序(也即 time_interrupt 和 ignore_int1)

3.myOS/i386/irq.s 的填写

在该模块中你需要用到的汇编语言：

ret：返回调用该函数的函数 sti：开中断 cli：关中断

4.Tick的实现

我们会在由 i8253 引起的时钟中断而引起的中断子程序 time_interrupt 处理中调用 tick 函数，由于 tick 函数的调用是有固定频率的，所以我们可以用它来进行时钟的输出

```
#include "wallClock.h"

int system_ticks;//记录 tick 的调用次数
int HH,MM,SS;//分布代表当前时间的“时：分：秒”
void tick(void){//你需要填写完整
    system_ticks ++;
    //你需要完整对 HH,MM,SS 的处理程序
    //.....
    setWallClock(HH,MM,SS);
}
```

5.WallClock的实现

实现通过 vga 往合适的位置输出 HH:MM:SS，即显示时钟；根据 vga 显存中的数值，返回时钟，并存到相应的指针指向位置中。

```
void setWallClock(int HH,int MM,int SS){//通过 vga 往合适的位置输出 HH:MM:SS，即显示时钟
}
void getWallClock(int *HH,int *MM,int *SS){//根据 vga 显存中的数值，返回时钟，并存到相应
//的指针指向位置中
}
```

6.Shell的实现

功能： 显示交互界面；接收并处理命令行。

一个命令的元信息为（命令名，描述命令的字符串，func）本实验至少需要提供2个命令：

`cmd`，列出所有命令；

`help [cmd]`，调用指定命令的help函数，若没有指定命令，则调用 help 的help 函数。

添加命令可以使用静态的方式手动添加，也可以用数组等方式动态注册。

源代码如下：

```
typedef struct myCommand {
    char name[80]; //命令名(可以作为唯一标识符使用)
    char help_content[200]; //该命令的使用说明
    int (*func)(int argc, char (*argv)[8]); ;//函数指针的概念，用于指向该命令的处理函数
    //你可以添加自定义的命令信息
}myCommand;

int func_cmd(int argc, char (*argv)[8]){
```

```

//输出所有命令的命令名
//你可以设计一个 myCommand 类型的数组，然后遍历它，输出所有命令的命令名
//由于本实验只需要实现 cmd 和 help 命令，所以也可以直接输出 cmd 的命令名“cmd”和 help
//的命令名“help”
}
int func_help(int argc, char (*argv)[8]);

//在这里我们用静态的方式定义了类型为 myCommand 的 cmd 命令和help命令
myCommand cmd={"cmd\0", "List all command\n\0", func_cmd};
myCommand help={"help\0", "usage: help [command]\n\0Display info about
[command]\n\0", func_help};

void startShell(void){
//我们通过串口来实现数据的输入
char BUF[256]; //输入缓存区
int BUF_len=0; //输入缓存区的长度

int argc;
char argv[8][8];

do{
    BUF_len=0;
    myPrintk(0x07, "Student>>\0");
    while((BUF[BUF_len]=uart_get_char())!='\r'){
        uart_put_char(BUF[BUF_len]); //将串口输入的数存入BUF数组中
        BUF_len++; //BUF数组的长度加
    }
    uart_put_char('\n');

    //助教已经帮助你们实现了“从串口中读取数据存储到BUF数组中”的任务，接下来你们要做
    //的就是对BUF数组中存储的数据进行处理(也即，从BUF数组中提取相应的argc和argv参
    //数)，再根据argc和argv，寻找相应的myCommand ***实例，进行***.func(argc, argv)函
    //数
    //调用。
    //比如BUF中的内容为 “help cmd”
    //那么此时的argc为2 argv[0]为help argv[1]为cmd
    //接下来就是 help.func(argc, argv)进行函数调用即可

}while(1);

}

```

目录组织

```

.
├─ compile_flags.txt
├─ Makefile
├─ multibootheader
│   └─ multibootHeader.S
├─ myOS
│   └─ dev
│       ├── i8253.c
│       ├── i8259A.c
│       ├── Makefile
│       ├── uart.c
│       └─ vga.c
└─ i386

```

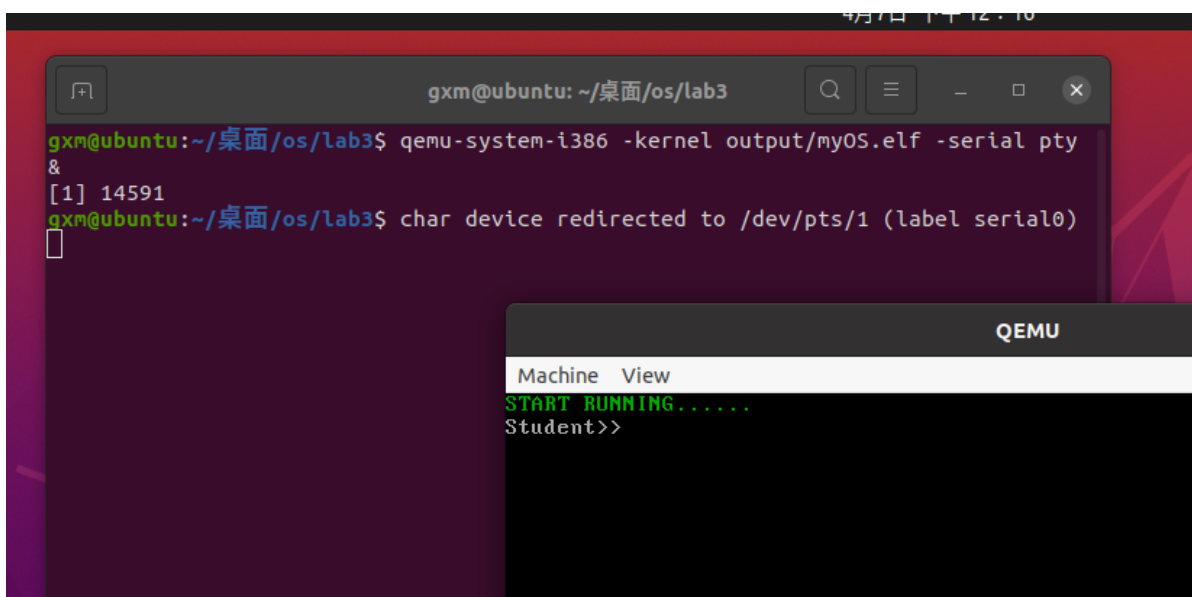
```

|   |   |   | io.c
|   |   |   | irq.S
|   |   |   | irqs.c
|   |   |   | Makefile
|   |   |   |
|   |   |   | include
|   |   |   |   |   | i8253.h
|   |   |   |   |   | i8259A.h
|   |   |   |   |   | io.h
|   |   |   |   |   | irqs.h
|   |   |   |   |   | myPrintk.h
|   |   |   |   |   | tick.h
|   |   |   |   |   | uart.h
|   |   |   |   |   | vga.h
|   |   |   |   |   | vsprintf.h
|   |   |   |   |   | wallClock.h
|   |   |   |   |
|   |   |   | kernel
|   |   |   |   |   | Makefile
|   |   |   |   |   | tick.c
|   |   |   |   |   | wallClock.c
|   |   |   |   |
|   |   |   | Makefile
|   |   |   | myOS.ld
|   |   |   | osStart.c
|   |   |   | printk
|   |   |   |   |   | Makefile
|   |   |   |   |   | myPrintk.c
|   |   |   |   |   | vsprintf.c
|   |   |   |   |
|   |   |   | start32.S
|   |   |   |
|   |   |   | source2run.sh
|   |   |   |
|   |   |   | userApp
|   |   |   |   |   | main.c
|   |   |   |   |   | Makefile
|   |   |   |   |   | startShell.c

```

运行及运行效果

输入 `./source2run.sh` 指令后，编译，链接，生成 `myOS.elf` 文件。输入指令 `qemu-system-i386 -kernel myOS.elf -serial pty &` 运行。运行结果如下图：



```

gxm@ubuntu: ~/桌面/os/lab3
gxm@ubuntu:~/桌面/os/lab3$ qemu-system-i386 -kernel output/myOS.elf -serial pty &
[1] 14591
gxm@ubuntu:~/桌面/os/lab3$ char device redirected to /dev/pts/1 (label serial0)

```

QEMU

```

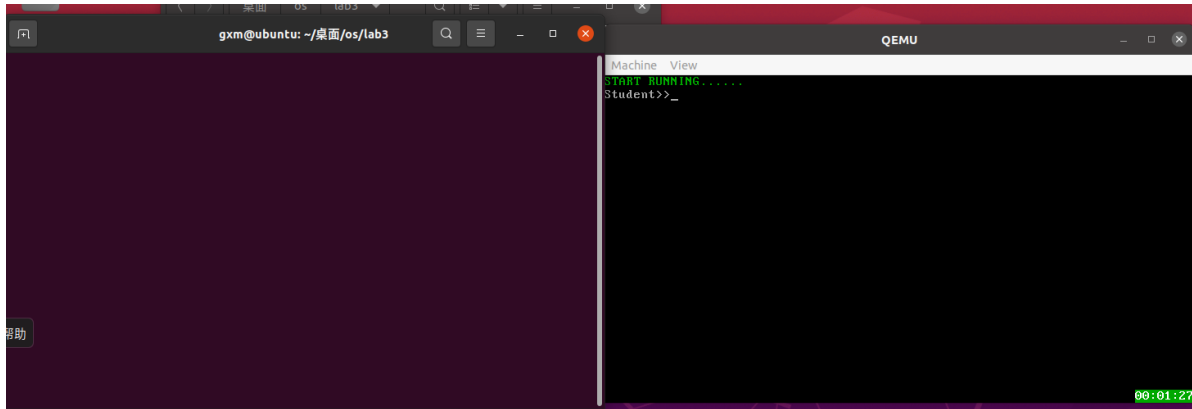
Machine View
START RUNNING.....
Student>>

```

串口重定向到伪终端，运行时会告知具体是哪个： `/dev/pts/1`

使用 `screen` 命令进入交互界面 (与QEMU) `sudo screen /dev/pts/1`

时钟如右下角所示:



执行 `cmd`, `help`, `help cmd` 和未知命令, 结果如下:

