本科生实验报告

实验课程:　操作系统原理实验

任课教师:　刘宁

实验题目:　Lab4 从实模式到保护模式

专业名称:　计算机科学与技术

学生姓名:　孙凯

学生学号:　23336212

实验地点:　实验中心B202

实验时间:　2025.4.9

一、实验要求：

在这次实验中，我们将掌握使用C语言来编写内核的方法，理解保护模式的中断处理机制和处理时钟中断，为后面的二级分页机制和多线程/进程打下基础。

具体内容如下:

1. 学习如何使用C/C++和汇编混合编程
2. 保护模式下内核的加载
3. 保护模式下的中断实现以及IDT初始化
4. 实时钟中断的处理

二、预备知识和实验环境：

预备知识：x86汇编语言程序设计，x86架构下的计算机保护模式的进入，保护模式下的中断机制

实验环境:

```
1. 虚拟机版本/处理器型号: Ubuntu 20.04 LTS
2. 代码编辑环境: Vscode+nasm+C/C++插件+qemu仿真平台
3. 代码编译工具: gcc/g++ （64位）
4.  重要三方库信息: 无
5. 代码程序调试工具: gdb
```

三、实验任务：

- 任务一：复现网址中“一个混合编程的例子”部分
- 任务二：复现网址中“内核的加载”部分，在进入 setup_kernel 函数后，将输出 Hello World 改为输出“学号+姓名首字母”，保存结果截图并说说你是怎么做的。

- 任务三：复现网址中“初始化IDT”部分，你可以更改默认的中断处理函数为你编写的函数，然后触发之，对结果进行截图并说说你是怎么做的。要求：调用处理函数时输出个人学号或姓名信息
- 任务四：复现网址中“8259A编程——实时钟中断的处理”部分，要求：仿照该章节中使用C语言来实现时钟中断的例子，利用 C/C++ 、 InterruptManager 、 STDIO 和你自己封装的类来实现你的时钟中断处理过程(例如，通过时钟中断，你可以在屏幕的第一行实现一个跑马灯。跑马灯显示自己学号和英文名，即类似于LED屏幕显示的效果)，保存结果截图并说说你的思路和做法。

四、实验步骤和实验结果：

任务一：复现网址中“一个混合编程的例子”部分

代码在`lab4/1`

- 任务要求：
  1. 将原例子中最后一行的输出"Done"（参考下图）改为"Done by 学号 姓名首字母"
  2. 结合具体的代码说明C代码调用汇编函数的语法和汇编代码调用C函数的语法。例如，结合关键代码说明 global 、 extern 关键字的作用，为什么C++的函数前需要加上 extern "C" 等， 保存结果截图并说说你是怎么做的；
  3. 学习make的使用，并用make来构建项目，保存结果截图并说说你是怎么做。
- 实验思路：在本节中，我们需要做的工作如下：
  - 在文件`c_func.c`中定义C函数`function_from_C`。
  - 在文件`cpp_func.cpp`中定义C++函数`function_from_CPP`。
  - 在文件`asm_func.asm`中定义汇编函数`function_from_asm`，在`function_from_asm`中调用`function_from_C`和`function_from_CPP`。
  - 在文件`main.cpp`中调用汇编函数`function_from_asm`。
- 实验步骤：

1. 编写`c_func.c`、`cpp_func.cpp`、`asm_func.asm`、`main.cpp`：

这里的代码与实验指导仓库基本相同，只是在`main.cpp`文件中修改了一个输出学号和姓名首字母缩写的代码如下：

```
std::cout << "Done by 23336212 sk" << std::endl;
```

2. 编写`makefile`：

```
main.out: main.o c_func.o cpp_func.o asm_func.o
    g++ -o main.out main.o c_func.o cpp_func.o asm_func.o -m32

c_func.o: c_func.c
    gcc -o c_func.o -m32 -c c_func.c

cpp_func.o: cpp_func.cpp
    g++ -o cpp_func.o -m32 -c cpp_func.cpp

main.o: main.cpp
    g++ -o main.o -m32 -c main.cpp
```

```
asm_func.o: asm_func.asm
    nasm -o asm_func.o -f elf32 asm_func.asm
clean:
    rm *.o
```

3. 用命令行编译启动

```
gcc -o c_func.o -m32 -c c_func.c
g++ -o cpp_func.o -m32 -c cpp_func.cpp
g++ -o main.o -m32 -c main.cpp
nasm -o asm_func.o -f elf32 asm_func.asm
g++ -o main.out main.o c_func.o cpp_func.o asm_func.o -m32
```

然后执行`./main.out`

实验结果如下：成功输出学号+姓名首字母缩写：23336212 sk

```
sk@sk-virtual-machine:~/OSlab/lab4/1$ make
g++ -o main.o -m32 -c main.cpp
gcc -o c_func.o -m32 -c c_func.c
g++ -o cpp_func.o -m32 -c cpp_func.cpp
nasm -o asm_func.o -f elf32 asm_func.asm
g++ -o main.out main.o c_func.o cpp_func.o asm_func.o -m32
sk@sk-virtual-machine:~/OSlab/lab4/1$ ./main.out
Call function from assembly.
This is a function from C.
This is a function from C++.
Done by 23336212 sk
sk@sk-virtual-machine:~/OSlab/lab4/1$
```

4. 用`make`构建

先运行`make`命令，然后运行`./make.out`。

实验结果如下：成功输出学号+姓名首字母缩写：23336212 sk

```
sk@sk-virtual-machine:~/OSlab/lab4/1$ make
g++ -o main.o -m32 -c main.cpp
gcc -o c_func.o -m32 -c c_func.c
g++ -o cpp_func.o -m32 -c cpp_func.cpp
nasm -o asm_func.o -f elf32 asm_func.asm
g++ -o main.out main.o c_func.o cpp_func.o asm_func.o -m32
sk@sk-virtual-machine:~/OSlab/lab4/1$ ./main.out
Call function from assembly.
This is a function from C.
This is a function from C++.
Done by 23336212 sk
sk@sk-virtual-machine:~/OSlab/lab4/1$
```

- 问题回答：

1. `c++`函数前加上`extern`关键字的原因是：因为C++支持函数重载，为了区别同名的重载函数，C++在编译时会进行名字修饰。因此，`extern "C"`目的是告诉编译器按C代码的规则编译，不进行名字修饰。
2. 汇编代码中`global`关键字的作用是：将汇编代码中定义的符号（如函数或变量）标记为全局可见，使得该符号能够被其他编译单元（如C/C++代码）正确链接和调用

`extern`关键字的作用是声明一个符号（变量或函数）的定义存在于其他编译单元（如其他源文　　件或汇编文件）中。

任务二：复现网址中“内核的加载”部分，在进入 setup_kernel 函数后，将输出 Hello World 改为输出“学号+姓名首字母”。

代码放在`lab4/2`

- 实验要求：在进入 setup_kernel 函数后，将输出 Hello World 改为输出“学号+姓名首字母”
- 实验思路：只需要修改`asm_utils.asm`中的`asm_hello_world`函数即可。
- 实验步骤：

整个项目的架构如下：

```
sk@sk-virtual-machine:~/OSlab/lab4/2$ tree
.
├── build
│   ├── asm_utils.o
│   ├── bootloader.bin
│   ├── entry.obj
│   ├── kernel.bin
│   ├── kernel.o
│   ├── makefile
│   ├── mbr.bin
│   └── setup.o
├── include
│   ├── asm_utils.h
│   ├── boot.inc
│   ├── os_type.h
│   └── setup.h
├── README.md
├── run
│   ├── gdbinit
│   └── hd.img
└── src
    ├── boot
    │   ├── bootloader.asm
    │   ├── entry.asm
    │   └── mbr.asm
    ├── kernel
    │   └── setup.cpp
    └── utils
        └── asm_utils.asm

7 directories, 20 files
sk@sk-virtual-machine:~/OSlab/lab4/2$
```

整个项目运行流程为：

```
BIOS
 └── 加载 → mbr.asm
      ├── 调用 → asm_read_hard_disk
      └── 跳转 → bootloader.asm
           ├── 调用 → asm_read_hard_disk
           └── 跳转 → entry.asm
                └── 跳转 → setup_kernel (setup.cpp)

setup_kernel (setup.cpp)
  └──调用 → asm_hello_world (asm_utils.asm)
```

1. 改写`asm_utils.asm`中的`asm_hello_world`函数:

其他文件和仓库所给相同

```
[bits 32]

global asm_hello_world

asm_hello_world:
    push eax
    xor eax, eax

    mov ah, 0x03 ;青色
    mov al, '2'
    mov [gs:2 * 0], ax

    mov al, '3'
    mov [gs:2 * 1], ax

    mov al, '3'
    mov [gs:2 * 2], ax

    mov al, '3'
    mov [gs:2 * 3], ax

    mov al, '6'
    mov [gs:2 * 4], ax

    mov al, '2'
    mov [gs:2 * 5], ax

    mov al, '1'
    mov [gs:2 * 6], ax

    mov al, '2'
    mov [gs:2 * 7], ax

    mov al, ' '
    mov [gs:2 * 8], ax

    mov al, 's'
```

```
    mov [gs:2 * 9], ax

    mov al, 'k'
    mov [gs:2 * 10], ax

    pop eax
    ret
```

2. 编写`makefile`:

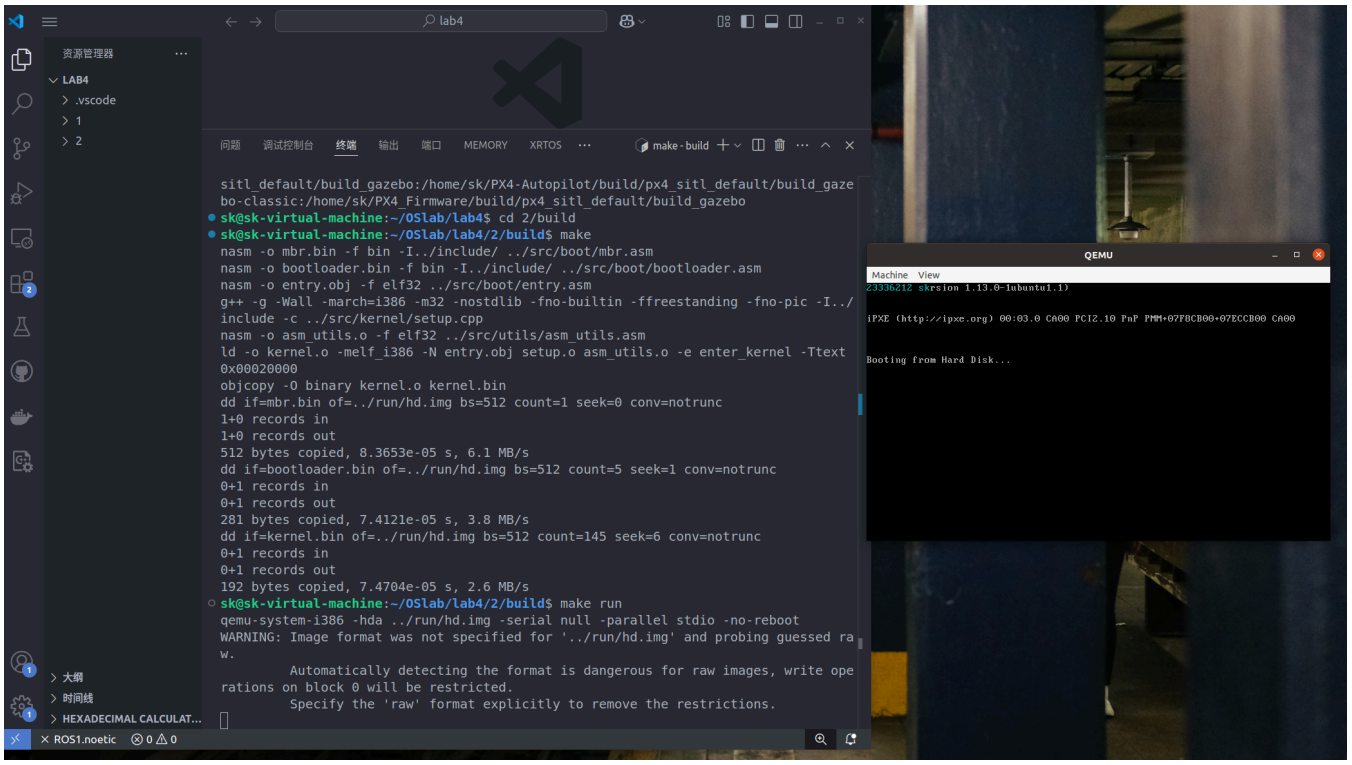
   各部分代码的含义注释如下:

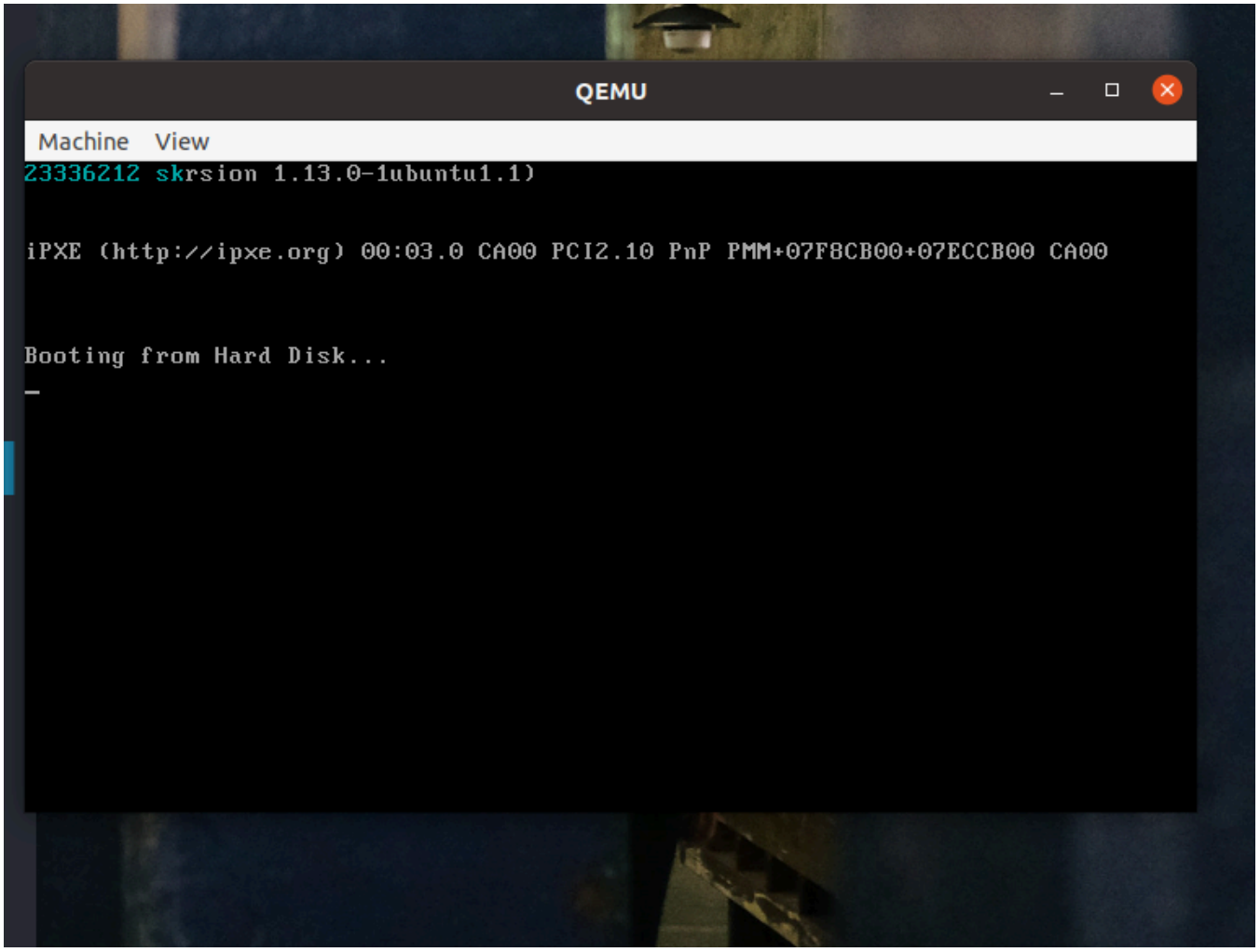
```
ASM_COMPILER = nasm                # 汇编器: nasm
C_COMPLIER = gcc                   # C编译器: gcc
CXX_COMPLIER = g++                 # C++编译器: g++
CXX_COMPLIER_FLAGS = -g -Wall -march=i386 -m32 -nostdlib -fno-builtin -ffreestanding -fno-
pic  # C++编译选项
LINKER = ld                        # 链接器: ld

SRCDIR = ../src                    # 源代码目录
RUNDIR = ../run                    # 运行目录（存放镜像）
BUILDDIR = build                   # 构建目录（未实际使用）
INCLUDE_PATH = ../include          # 头文件目录

CXX_SOURCE += $(wildcard $(SRCDIR)/kernel/*.cpp)  # 收集所有 C++ 源文件
CXX_OBJ += $(CXX_SOURCE:$(SRCDIR)/kernel/%.cpp=%.o)
 # 转换为目标文件列表（如 main.cpp → main.o）

ASM_SOURCE += $(wildcard $(SRCDIR)/utils/*.asm)   # 收集所有汇编源文件
ASM_OBJ += $(ASM_SOURCE:$(SRCDIR)/utils/%.asm=%.o) # 转换为目标文件列表（如 asm_utils.asm →
asm_utils.o）

OBJ += $(CXX_OBJ) $(ASM_OBJ)  # 合并所有目标文件

build : mbr.bin bootloader.bin kernel.bin kernel.o
    # 将生成的二进制文件写入磁盘镜像
    dd if=mbr.bin of=$(RUNDIR)/hd.img bs=512 count=1 seek=0 conv=notrunc
    dd if=bootloader.bin of=$(RUNDIR)/hd.img bs=512 count=5 seek=1 conv=notrunc
    dd if=kernel.bin of=$(RUNDIR)/hd.img bs=512 count=145 seek=6 conv=notrunc
mbr.bin : $(SRCDIR)/boot/mbr.asm
    $(ASM_COMPILER) -o mbr.bin -f bin -I$(INCLUDE_PATH)/ $(SRCDIR)/boot/mbr.asm

bootloader.bin : $(SRCDIR)/boot/bootloader.asm
    $(ASM_COMPILER) -o bootloader.bin -f bin -I$(INCLUDE_PATH)/
$(SRCDIR)/boot/bootloader.asm

entry.obj : $(SRCDIR)/boot/entry.asm
    $(ASM_COMPILER) -o entry.obj -f elf32 $(SRCDIR)/boot/entry.asm

kernel.bin : kernel.o
    objcopy -O binary kernel.o kernel.bin

kernel.o : entry.obj $(OBJ)
```

```
    $(LINKER) -o kernel.o -melf_i386 -N entry.obj $(OBJ) -e enter_kernel -Ttext 0x00020000

$(CXX_OBJ):
    $(CXX_COMPLIER) $(CXX_COMPLIER_FLAGS) -I$(INCLUDE_PATH) -c $(CXX_SOURCE)

asm_utils.o : $(SRCDIR)/utils/asm_utils.asm
    $(ASM_COMPILER) -o asm_utils.o -f elf32 $(SRCDIR)/utils/asm_utils.asm
clean:
    rm -f *.o* *.bin

run:
    qemu-system-i386 -hda $(RUNDIR)/hd.img -serial null -parallel stdio -no-reboot

debug:
    qemu-system-i386 -S -s -parallel stdio -hda $(RUNDIR)/hd.img -serial null&
    @sleep 1
    gnome-terminal -e "gdb -q -tui -x $(RUNDIR)/gdbinit"
```

3. 运行`make`指令后，再运行`make run`指令

实验结果如下：成功输出`23336212 sk`



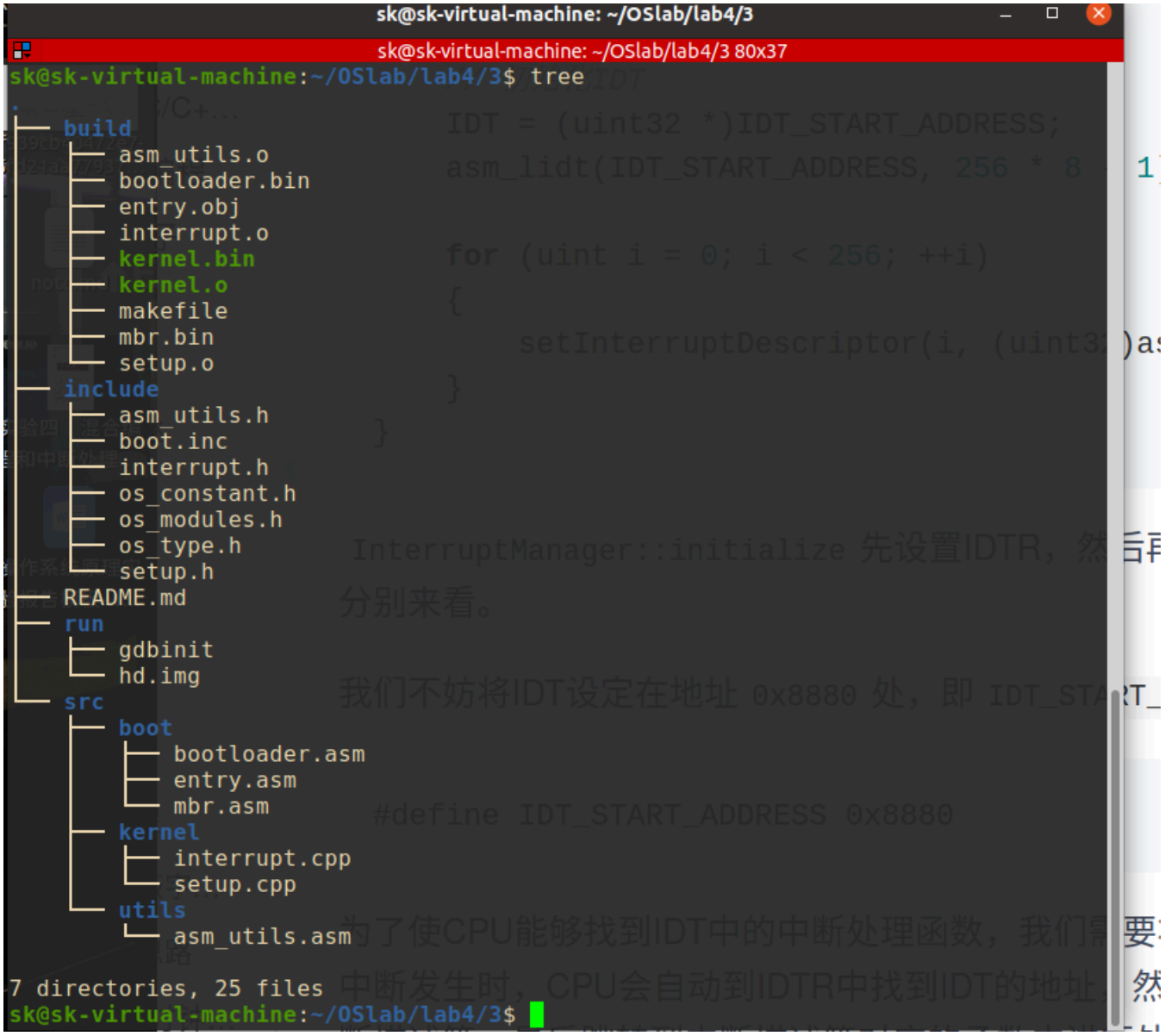


任务三：复现网址中“初始化IDT”部分，你可以更改默认的中断处理函数为你编写的函数，然后触发之。

代码放在`lab4/3`

- 实验要求：调用自己编写的处理函数时输出个人学号或姓名信息
- 实验思路：在`asm_utils.asm`汇编文件中添加自己的中断处理函数，输出学号和姓名首字母缩写23336212sk，然后修改`interrupt.cpp`中初始化IDT部分，将中断描述符和中断处理函数链接起来，保证发生中断的时候，能够调用中断处理函数。
- 实验步骤：

整个项目的架构如下：



整个项目运行流程如下：

```
BIOS
 └── 加载 → mbr.asm
      ├── 调用 → asm_read_hard_disk
      └── 跳转 → bootloader.asm
           ├── 调用 → asm_read_hard_disk
           └── 跳转 → entry.asm
                └── 跳转 → setup_kernel (setup.cpp)

setup_kernel (setup.cpp)
 ├── 调用 → InterruptManager::initialize (interrupt.cpp)
 │        ├── 调用 → asm_lidt (asm_utils.asm)
 │        └── 调用 → setInterruptDescriptor (interrupt.cpp)
 │             └── 调用 → asm_my_interrupt (asm_utils.asm)
 ├── 调用 → asm_hello_world (asm_utils.asm)
 └── 调用 → asm_halt (asm_utils.asm)
```

其他部分代码和仓库所给相同

1. 在`asm_utils.asm`汇编文件中添加自己的中断处理函数:

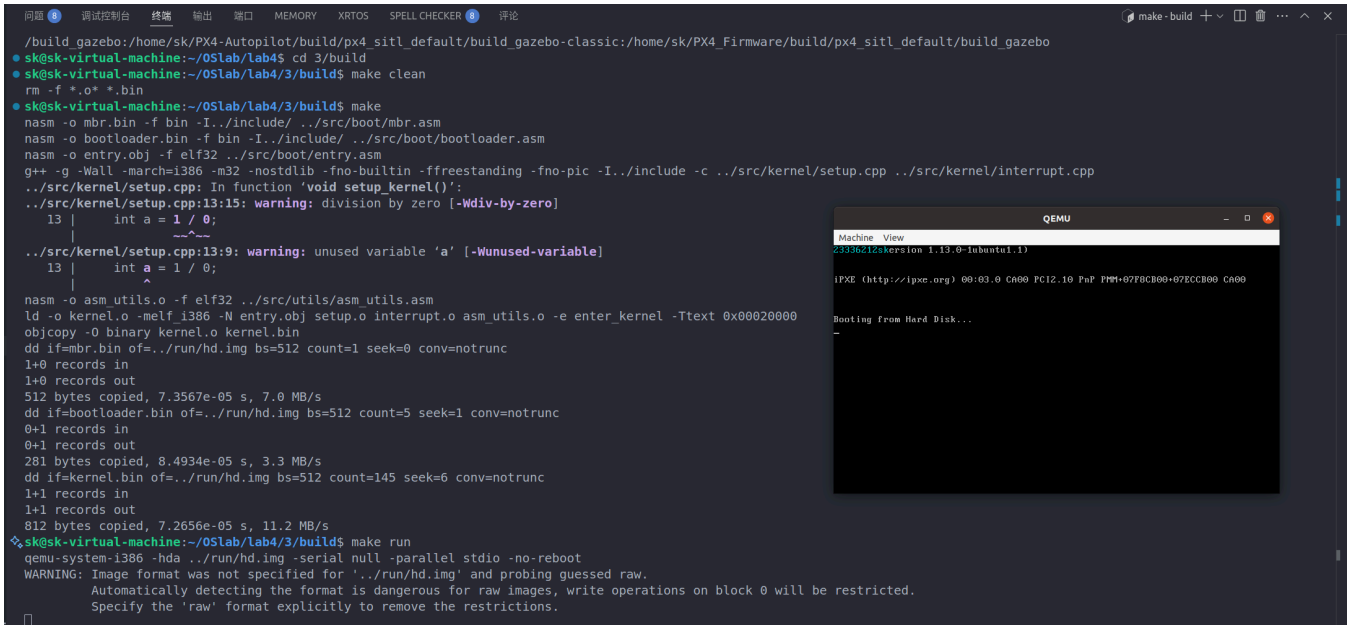
```
ASM_MY_INTERRUPT_INFO db '23336212sk'
                      db 0

;中断输出23336212sk的信息
asm_my_interrupt:
    cli ;中断处理程序的入口，cli指令：禁用中断，防止在处理中断时被其他中断打断。确保中断处理程序的执行是原子性的。
    xor ebx, ebx
    mov esi, ASM_MY_INTERRUPT_INFO
    xor eax, eax
    mov ah, 0x03
;输出字符串到屏幕
.output_information:
    cmp byte[esi], 0
    je .end
    mov al, byte[esi]
    mov word[gs:bx], ax
    inc esi
    add ebx, 2
    jmp .output_information
.end:
    jmp $
```

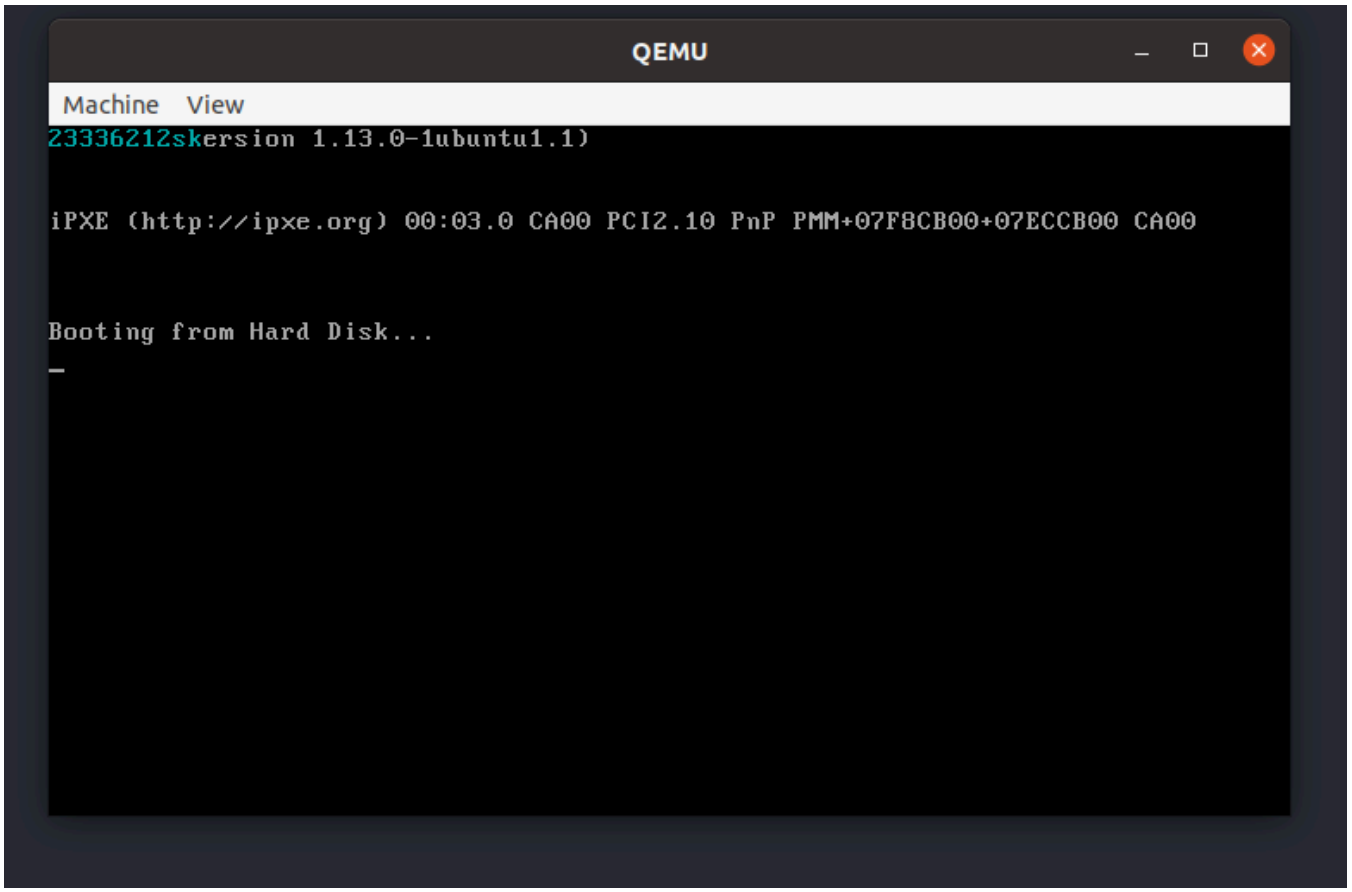
2. 修改`interrupt.cpp`中初始化IDT部分:

```
void InterruptManager::initialize()
{
    // 初始化IDT
    IDT = (uint32 *)IDT_START_ADDRESS;
    asm_lidt(IDT_START_ADDRESS, 256 * 8 - 1);

    for (uint i = 0; i < 256; ++i)
    {
        setInterruptDescriptor(i, (uint32)asm_my_interrupt, 0);
    }

}
```

3. 运行`make`指令后，再运行`make run`指令

实验结果如下：成功输出`23336212sk`



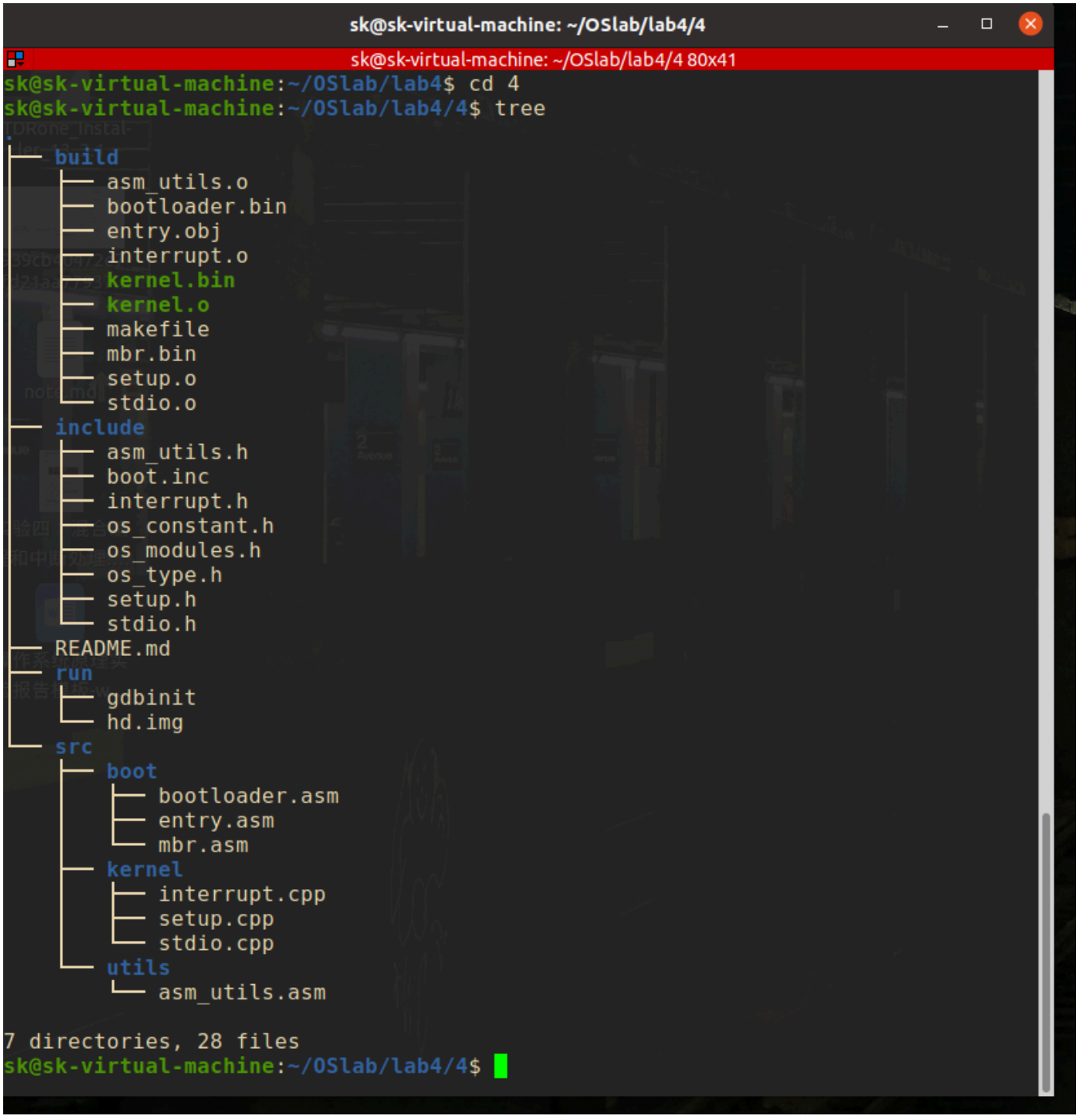


任务四：复现网址中“8259A编程——实时钟中断的处理”部分

代码放在`lab4/4`

- 实验要求：仿照该章节中使用C语言来实现时钟中断的例子，利用 C/C++ 、 InterruptManager 、 STDIO 和你自己封装的类来实现你的时钟中断处理过程(例如，通过时钟中断，你可以在屏幕的第一行实现一个跑马灯。跑马灯显示自己学号和英文名，即类似于LED屏幕显示的效果)，保存结果截图并说说你的思路和做法。
- 实验思路：整个任务只需在`interrupt.cpp`中编写自己的时钟中断处理函数即可
- 实验步骤：

整个项目的架构如下：



整个项目运行流程如下:

```
BIOS
 └── 加载 → mbr.asm
      ├── 调用 → asm_read_hard_disk
      └── 跳转 → bootloader.asm
           ├── 调用 → asm_read_hard_disk
           └── 跳转 → entry.asm
                └── 跳转 → setup_kernel (setup.cpp)

setup_kernel (setup.cpp)
 ├── 调用 → InterruptManager::initialize (interrupt.cpp)
 │        ├── 调用 → asm_lidt (asm_utils.asm)
 │        └── 调用 → setTimeInterrupt (interrupt.cpp)
 │             └── 设置 → c_time_interrupt_handler (interrupt.cpp)
```

```
  ├── 调用 → asm_enable_interrup(asm_utils.asm)
  └── 调用 → asm_halt() 死循环

c_time_interrupt_handler (interrupt.cpp)
  ├── 清空屏幕第一行
  ├── 显示跑马灯字符串 "23336212sk"
  └── 更新跑马灯索引
```

其他的代码与仓库一致

1. 编写时钟中断处理程序：

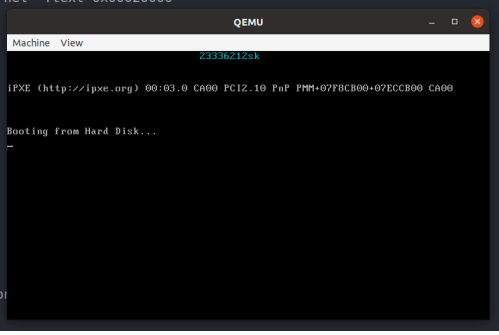
```
// 定义跑马灯字符串和索引
char marquee[] = "23336212sk";
int marquee_index = 0;

// 中断处理函数
extern "C" void c_time_interrupt_handler()
{
    // 清空屏幕第一行
    for (int i = 0; i < 80; ++i)
    {
        stdio.print(0, i, ' ', 0x07);
    }

    // 获取跑马灯字符串长度
    int len = sizeof(marquee) - 1; // 字符串长度（不包括 '\0'）

    // 显示跑马灯字符串
    for (int i = 0; i < len; ++i)
    {
        // 计算当前字符在屏幕上的位置
        int pos = (marquee_index + i) % 80; // 屏幕宽度为 80
        stdio.print(0, pos, marquee[i], 0x03);
    }

    // 更新跑马灯索引
    marquee_index = (marquee_index + 1) % 80;

}
```

2. 运行`make`指令后，再运行`make run`指令

实验结果如下：可以看到`23336212sk`在屏幕的第一行滚动显示，成功实现跑马灯。

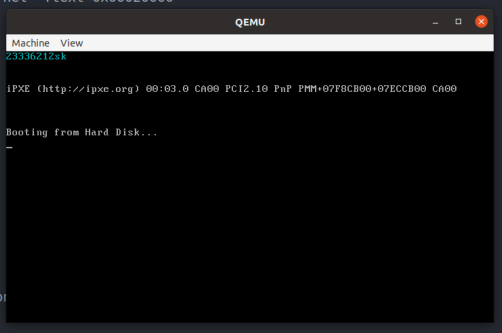
问题 1 调试控制台 终端 输出 端口 MEMORY XRTOS SPELL CHECKER 1 评论

```
pilot/build/px4_sitl_default/build_gazebo-classic
GAZEBO_MODEL_PATH /home/sk/PX4_Firmware/Tools/sitl_gazebo/models:/home/sk/PX4-Autopilot/Tools/simulation/gazebo-classic/sitl_gazebo-classic/models
LD_LIBRARY_PATH /home/sk/Tutorial_2024/workspace/Task1/catkin_ws/devel/lib:/home/sk/catkin_ws/devel/lib:/opt/ros/noetic/lib:/home/sk/PX4-Autopilot/build/px4_sitl_
default/build_gazebo-classic:/home/sk/PX4_Firmware/build/px4_sitl_default/build_gazebo:/home/sk/PX4-Autopilot/build/px4_sitl_default/build_gazebo-classic
GAZEBO_PLUGIN_PATH :/home/sk/PX4-Autopilot/build/px4_sitl_default/build_gazebo-classic:/home/sk/PX4_Firmware/build/px4_sitl_default/build_gazebo:/home/sk/PX4-Auto
pilot/build/px4_sitl_default/build_gazebo-classic:/home/sk/PX4_Firmware/build/px4_sitl_default/build_gazebo
GAZEBO_MODEL_PATH /home/sk/PX4_Firmware/Tools/sitl_gazebo/models:/home/sk/PX4-Autopilot/Tools/simulation/gazebo-classic/sitl_gazebo-classic/models:/home/sk/PX4_Fi
rmware//Tools/sitl_gazebo/models
LD_LIBRARY_PATH /home/sk/catkin_ws/devel/lib:/opt/ros/noetic/lib:/home/sk/PX4-Autopilot/build/px4_sitl_default/build_gazebo-classic:/home/sk/PX4_Firmware/build/px
4_sitl_default/build_gazebo:/home/sk/PX4-Autopilot/build/px4_sitl_default/build_gazebo-classic:/home/sk/PX4_Firmware/build/px4_sitl_default/build_gazebo
sk@sk-virtual-machine:~/OSlab/lab4$ cd 4/build
sk@sk-virtual-machine:~/OSlab/lab4/4/build$ make
g++ -g -Wall -march=i386 -m32 -nostdlib -fno-builtin -ffreestanding -fno-pic -I../include -c ../src/kernel/setup.cpp ../src/kernel/interrupt.cpp ../src/kernel/std
io.cpp
ld -o kernel.o -melf_i386 -N entry.obj setup.o interrupt.o stdio.o asm_utils.o -e enter_kernel -Ttext 0x00020000
objcopy -O binary kernel.o kernel.bin
dd if=mbr.bin of=../run/hd.img bs=512 count=1 seek=0 conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000444363 s, 1.2 MB/s
dd if=bootloader.bin of=../run/hd.img bs=512 count=5 seek=1 conv=notrunc
0+1 records in
0+1 records out
281 bytes copied, 0.00035991 s, 781 kB/s
dd if=kernel.bin of=../run/hd.img bs=512 count=145 seek=6 conv=notrunc
5+1 records in
5+1 records out
2691 bytes (2.7 kB, 2.6 KiB) copied, 0.000477586 s, 5.6 MB/s
sk@sk-virtual-machine:~/OSlab/lab4/4/build$ make run
qemu-system-i386 -hda ../run/hd.img -serial null -parallel stdio -no-reboot
WARNING: Image format was not specified for '../run/hd.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations o
         Specify the 'raw' format explicitly to remove the restrictions.
```



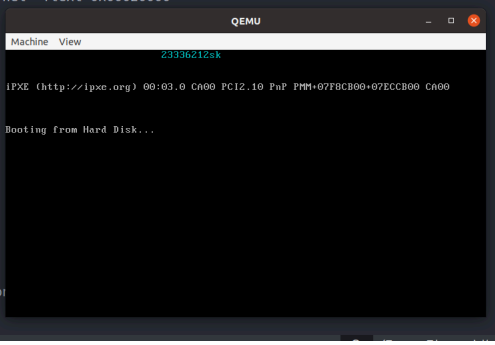
问题 1 调试控制台 终端 输出 端口 MEMORY XRTOS SPELL CHECKER 1 评论

```
pilot/build/px4_sitl_default/build_gazebo-classic
GAZEBO_MODEL_PATH /home/sk/PX4_Firmware/Tools/sitl_gazebo/models:/home/sk/PX4-Autopilot/Tools/simulation/gazebo-classic/sitl_gazebo-classic/models
LD_LIBRARY_PATH /home/sk/Tutorial_2024/workspace/Task1/catkin_ws/devel/lib:/home/sk/catkin_ws/devel/lib:/opt/ros/noetic/lib:/home/sk/PX4-Autopilot/build/px4_sitl_
default/build_gazebo-classic:/home/sk/PX4_Firmware/build/px4_sitl_default/build_gazebo:/home/sk/PX4-Autopilot/build/px4_sitl_default/build_gazebo-classic
GAZEBO_PLUGIN_PATH :/home/sk/PX4-Autopilot/build/px4_sitl_default/build_gazebo-classic:/home/sk/PX4_Firmware/build/px4_sitl_default/build_gazebo:/home/sk/PX4-Auto
pilot/build/px4_sitl_default/build_gazebo-classic:/home/sk/PX4_Firmware/build/px4_sitl_default/build_gazebo
GAZEBO_MODEL_PATH /home/sk/PX4_Firmware/Tools/sitl_gazebo/models:/home/sk/PX4-Autopilot/Tools/simulation/gazebo-classic/sitl_gazebo-classic/models:/home/sk/PX4_Fi
rmware//Tools/sitl_gazebo/models
LD_LIBRARY_PATH /home/sk/catkin_ws/devel/lib:/opt/ros/noetic/lib:/home/sk/PX4-Autopilot/build/px4_sitl_default/build_gazebo-classic:/home/sk/PX4_Firmware/build/px
4_sitl_default/build_gazebo:/home/sk/PX4-Autopilot/build/px4_sitl_default/build_gazebo-classic:/home/sk/PX4_Firmware/build/px4_sitl_default/build_gazebo
sk@sk-virtual-machine:~/OSlab/lab4$ cd 4/build
sk@sk-virtual-machine:~/OSlab/lab4/4/build$ make
g++ -g -Wall -march=i386 -m32 -nostdlib -fno-builtin -ffreestanding -fno-pic -I../include -c ../src/kernel/setup.cpp ../src/kernel/interrupt.cpp ../src/kernel/std
io.cpp
ld -o kernel.o -melf_i386 -N entry.obj setup.o interrupt.o stdio.o asm_utils.o -e enter_kernel -Ttext 0x00020000
objcopy -O binary kernel.o kernel.bin
dd if=mbr.bin of=../run/hd.img bs=512 count=1 seek=0 conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000444363 s, 1.2 MB/s
dd if=bootloader.bin of=../run/hd.img bs=512 count=5 seek=1 conv=notrunc
0+1 records in
0+1 records out
281 bytes copied, 0.00035991 s, 781 kB/s
dd if=kernel.bin of=../run/hd.img bs=512 count=145 seek=6 conv=notrunc
5+1 records in
5+1 records out
2691 bytes (2.7 kB, 2.6 KiB) copied, 0.000477586 s, 5.6 MB/s
sk@sk-virtual-machine:~/OSlab/lab4/4/build$ make run
qemu-system-i386 -hda ../run/hd.img -serial null -parallel stdio -no-reboot
WARNING: Image format was not specified for '../run/hd.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations o
         Specify the 'raw' format explicitly to remove the restrictions.
```



问题 1 调试控制台 终端 输出 端口 MEMORY XRTOS SPELL CHECKER 1 评论

```
pilot/build/px4_sitl_default/build_gazebo-classic
GAZEBO_MODEL_PATH /home/sk/PX4_Firmware/Tools/sitl_gazebo/models:/home/sk/PX4-Autopilot/Tools/simulation/gazebo-classic/sitl_gazebo-classic/models
LD_LIBRARY_PATH /home/sk/Tutorial_2024/workspace/Task1/catkin_ws/devel/lib:/home/sk/catkin_ws/devel/lib:/opt/ros/noetic/lib:/home/sk/PX4-Autopilot/build/px4_sitl_
default/build_gazebo-classic:/home/sk/PX4_Firmware/build/px4_sitl_default/build_gazebo:/home/sk/PX4-Autopilot/build/px4_sitl_default/build_gazebo-classic
GAZEBO_PLUGIN_PATH :/home/sk/PX4-Autopilot/build/px4_sitl_default/build_gazebo-classic:/home/sk/PX4_Firmware/build/px4_sitl_default/build_gazebo:/home/sk/PX4-Auto
pilot/build/px4_sitl_default/build_gazebo-classic:/home/sk/PX4_Firmware/build/px4_sitl_default/build_gazebo
GAZEBO_MODEL_PATH /home/sk/PX4_Firmware/Tools/sitl_gazebo/models:/home/sk/PX4-Autopilot/Tools/simulation/gazebo-classic/sitl_gazebo-classic/models:/home/sk/PX4_Fi
rmware//Tools/sitl_gazebo/models
LD_LIBRARY_PATH /home/sk/catkin_ws/devel/lib:/opt/ros/noetic/lib:/home/sk/PX4-Autopilot/build/px4_sitl_default/build_gazebo-classic:/home/sk/PX4_Firmware/build/px
4_sitl_default/build_gazebo:/home/sk/PX4-Autopilot/build/px4_sitl_default/build_gazebo-classic:/home/sk/PX4_Firmware/build/px4_sitl_default/build_gazebo
sk@sk-virtual-machine:~/OSlab/lab4$ cd 4/build
sk@sk-virtual-machine:~/OSlab/lab4/4/build$ make
g++ -g -Wall -march=i386 -m32 -nostdlib -fno-builtin -ffreestanding -fno-pic -I../include -c ../src/kernel/setup.cpp ../src/kernel/interrupt.cpp ../src/kernel/std
io.cpp
ld -o kernel.o -melf_i386 -N entry.obj setup.o interrupt.o stdio.o asm_utils.o -e enter_kernel -Ttext 0x00020000
objcopy -O binary kernel.o kernel.bin
dd if=mbr.bin of=../run/hd.img bs=512 count=1 seek=0 conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000444363 s, 1.2 MB/s
dd if=bootloader.bin of=../run/hd.img bs=512 count=5 seek=1 conv=notrunc
0+1 records in
0+1 records out
281 bytes copied, 0.00035991 s, 781 kB/s
dd if=kernel.bin of=../run/hd.img bs=512 count=145 seek=6 conv=notrunc
5+1 records in
5+1 records out
2691 bytes (2.7 kB, 2.6 KiB) copied, 0.000477586 s, 5.6 MB/s
sk@sk-virtual-machine:~/OSlab/lab4/4/build$ make run
qemu-system-i386 -hda ../run/hd.img -serial null -parallel stdio -no-reboot
WARNING: Image format was not specified for '../run/hd.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations o
         Specify the 'raw' format explicitly to remove the restrictions.
```



五、实验总结和心得体会

1. 通过这次实验，我学会如何使用C/C++和汇编混合编程，学会了如何在C代码中调用汇编代码编写的函数和如何在汇编代码中调用使用C编写的函数，同时使用`makefile`来管理项目。
2. 通过这次实验，我了解到了保护模式下的中断机制的实现，掌握了IDT的初始化和中断程序的编写。

六、参考资料

无