

中山大學

SUN YAT-SEN UNIVERSITY

# 本科生实验报告

实验课程: 操作系统实验

实验名称: 中断

专业名称: 19 级计算机科学与技术（人工智能与大数据方向）

学生姓名: 陈恩婷

学生学号: 19335015

实验地点: 实验中心 D503

实验成绩:

报告时间: 2021/4/9

## 1. 实验要求

### Assignment 1 混合编程

复现 Example 1，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，结果截图并说说你是怎么做的。同时，学习 `make` 的使用，并用 `make` 来构建 Example 1，结果截图并说说你是怎么做的。

### Assignment 2 使用 C/C++ 来编写内核

复现 Example 2，在进入 `setup_kernel` 函数后，将输出 Hello World 改为输出你的学号，结果截图并说说你是怎么做的。

### Assignment 3 中断的处理

复现 Example 3，你可以更改 Example 中默认的中断处理函数为你编写的函数，然后触发之，结果截图并说说你是怎么做的。

### Assignment 4 时钟中断

复现 Example 4，仿照 Example 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，结果截图并说说你是怎么做的。注意，不可以使用纯汇编的方式来实现。

## 2. 实验过程

### Assignment 1 混合编程

#### 1.1 C 代码调用汇编函数

1. 在汇编代码中将函数声明为 `global`

```
global function_from_asm
```

2. 用 `extern` 在 C/C++ 中将其声明来自外部

```
extern void function_from_asm();
```

在 C++ 中需要声明为 `extern "C"`

```
extern "C" void function_from_asm();
```

这么做的原因：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。也就是说，`function_from_CPP` 编译后的标号不再是 `function_from_CPP`，而是要带上额外的信息。而 C 代码编译后的标号还是原来的函数名。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。

## 1.2 汇编代码调用 C 函数

1. 在汇编代码中用 `extern` 声明这个函数来自于外部。

```
extern function_from_C
```

2. 在汇编代码中直接使用 C 函数

```
call function_from_C
```

## 1.3 复现 Example 1

1. 直接复现

根据 Makefile 中原有的命令，输入命令编译运行代码。

```
$ make clean
$ make c_func.o
$ make cpp_func.o
$ make asm_func.o
$ make main.o
$ make make.out
$ ./main.out
```

2. 学习 make 的使用，并用 make 来构建 Example 1

在 Makefile 中编写 `clean`, `build`, `run`，用它们来编译运行程序。

## Assignment 2 使用 C/C++ 来编写内核

1. 修改相关代码

在 `asm_utils.asm` 中相应位置修改代码，将输出修改为自己的学号。

2. 编译运行

使用 `make` 与 `make run` 命令编译运行

## Assignment 3 中断的处理

1. 修改相关代码

在 asm\_utils.asm 中相应位置修改代码，编写自己的中断处理函数：通过时钟中断，在屏幕的第一行实现一个跑马灯，显示自己的学号和英文名。

## 2. 编译运行

使用 make 与 make run 命令编译运行。

## Assignment 4 时钟中断

### 1. 修改相关代码

在 interrupt.cpp 中相应位置修改代码，编写自己的中断处理函数。

### 2. 编译运行

使用 make 与 make run 命令编译运行。

## 3. 关键代码

## Assignment 1 混合编程

```
clean:
    @rm *.o
build:
    @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
run:
    @./main.out
```

如上，在 Makefile 中编写 clean, build, run，用它们来编译运行程序。

## Assignment 2 使用 C/C++来编写内核

```
asm_hello_world:
    push eax
    xor eax, eax

    mov ah, 0x03 ;青色
    mov al, '1'
    mov [gs:2 * 0], ax

    mov al, '9'
    mov [gs:2 * 1], ax
```

```

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

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

mov al, '5'
mov [gs:2 * 4], ax

mov al, '0'
mov [gs:2 * 5], ax

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

mov al, '5'
mov [gs:2 * 7], ax

pop eax
ret

```

如上，修改 asm\_utils.asm 中的代码，将输出内容改为自己的学号。

### Assignment 3 中断的处理

在原有的中断处理函数代码的基础上，添加了一段实现字符弹射的代码，具体可以看附件中的实现。

### Assignment 4 时钟中断

```

extern "C" void c_time_interrupt_handler()
{
    // 清空屏幕
    for (int i = 0; i < 80; ++i)
    {
        stdio.print(0, i, ' ', 0x07);
    }

    // 中断发生的次数
    ++times;
    //char str[] = "interrupt happend: ";
    char number[] = "19335015 enting";
    int temp = times;

    stdio.moveCursor(temp/10%15);
    stdio.print(number[temp/10%15]);
}

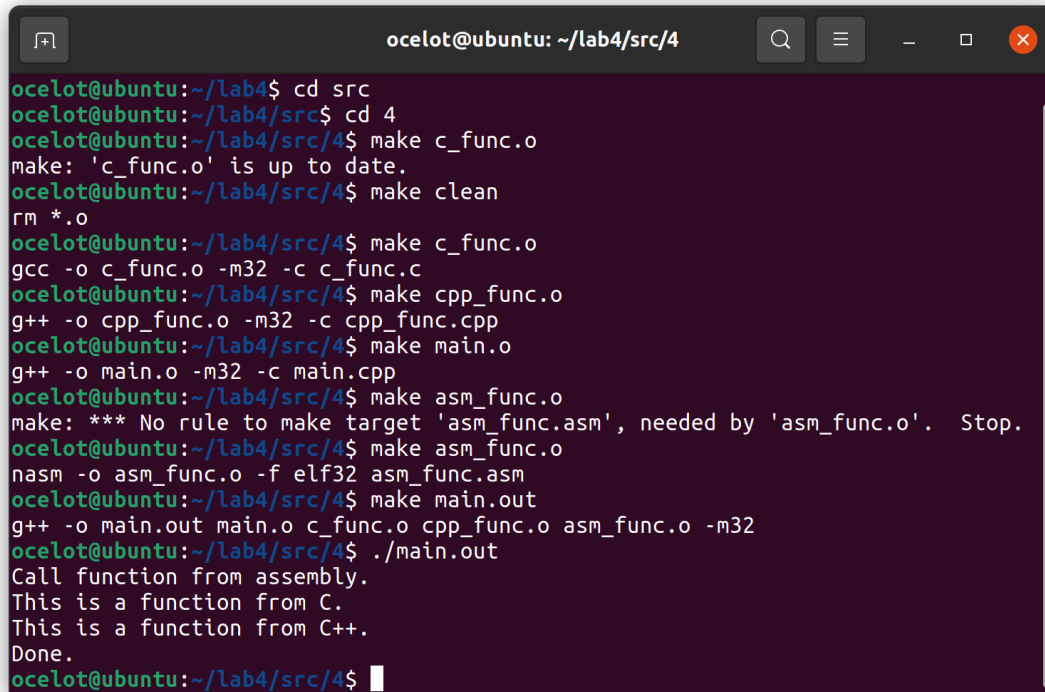
```

如上，修改 `asmutils.asm` 中的 `ctimeinterrupthandler()` 函数，通过时钟中断，在屏幕的第一行实现一个跑马灯，显示自己的学号和英文名。

## 4. 实验结果

### Assignment 1 混合编程

#### 1.1 直接复现



```
ocelot@ubuntu: ~/lab4/src/4
ocelot@ubuntu:~/lab4$ cd src
ocelot@ubuntu:~/lab4/src$ cd 4
ocelot@ubuntu:~/lab4/src/4$ make c_func.o
make: 'c_func.o' is up to date.
ocelot@ubuntu:~/lab4/src/4$ make clean
rm *.o
ocelot@ubuntu:~/lab4/src/4$ make c_func.o
gcc -o c_func.o -m32 -c c_func.c
ocelot@ubuntu:~/lab4/src/4$ make cpp_func.o
g++ -o cpp_func.o -m32 -c cpp_func.cpp
ocelot@ubuntu:~/lab4/src/4$ make main.o
g++ -o main.o -m32 -c main.cpp
ocelot@ubuntu:~/lab4/src/4$ make asm_func.o
make: *** No rule to make target 'asm_func.asm', needed by 'asm_func.o'. Stop.
ocelot@ubuntu:~/lab4/src/4$ make asm_func.o
nasm -o asm_func.o -f elf32 asm_func.asm
ocelot@ubuntu:~/lab4/src/4$ make main.out
g++ -o main.out main.o c_func.o cpp_func.o asm_func.o -m32
ocelot@ubuntu:~/lab4/src/4$ ./main.out
Call function from assembly.
This is a function from C.
This is a function from C++.
Done.
ocelot@ubuntu:~/lab4/src/4$
```

如图所示，根据 `Makefile` 中原有的命令，输入命令编译运行代码，正确输出了结果。

#### 1.2 学习 `make` 的使用，并用 `make` 来构建 Example 1

在 `Makefile` 中添加内容，使用新添加的命令编译运行：

```
ocelot@ubuntu: ~/lab4/src/4
ocelot@ubuntu:~/lab4/src/4$ make cpp_func.o
g++ -o cpp_func.o -m32 -c cpp_func.cpp
ocelot@ubuntu:~/lab4/src/4$ make main.o
g++ -o main.o -m32 -c main.cpp
ocelot@ubuntu:~/lab4/src/4$ make asm_func.o
make: *** No rule to make target 'asm_func.asm', needed by 'asm_func.o'. Stop.
ocelot@ubuntu:~/lab4/src/4$ make asm_func.o
nasm -o asm_func.o -f elf32 asm_func.asm
ocelot@ubuntu:~/lab4/src/4$ make main.out
g++ -o main.out main.o c_func.o cpp_func.o asm_func.o -m32
ocelot@ubuntu:~/lab4/src/4$ ./main.out
Call function from assembly.
This is a function from C.
This is a function from C++.
Done.
ocelot@ubuntu:~/lab4/src/4$ make clean
ocelot@ubuntu:~/lab4/src/4$ make build
ocelot@ubuntu:~/lab4/src/4$ make run
Call function from assembly.
This is a function from C.
This is a function from C++.
Done.
ocelot@ubuntu:~/lab4/src/4$
```

如图所示，程序正确输出了结果。

## Assignment 2 使用 C/C++来编写内核

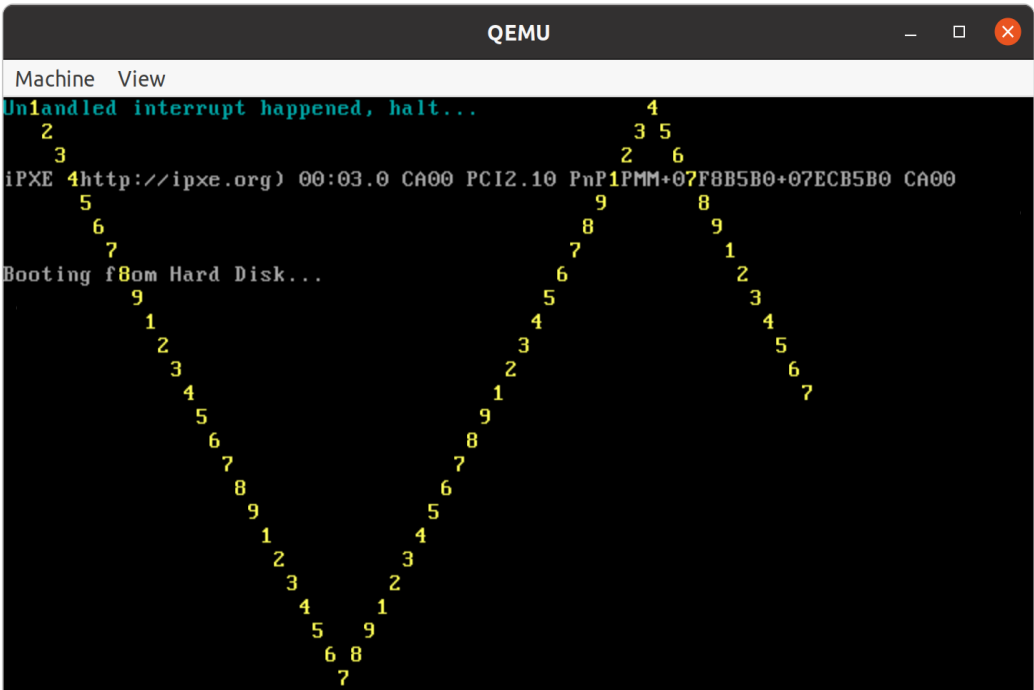
```
QEMU
Machine View
19335015(version 1.14.0-1)

iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07F8B5B0+07ECB5B0 CA00

Booting from Hard Disk...
```

如图所示，在 qemu 中正确输出了自己的学号。

### Assignment 3 中断的处理



如图所示，成功调用了中断处理函数，打印出了 `unhandled interrupt happened, halt...` 字符串，并进行了字符弹射。

### Assignment 4 时钟中断



如图所示，成功调用了中断处理函数，呈现了一个跑马灯来显示自己的学号和英文名，图为显示“5”时的屏幕截图。



## 5. 实验总结

### 1. 实验中遇到的问题

使用时钟中断实现跑马灯时，跑马灯闪动太快导致看不清楚

解决方案：修改代码，将原来的 `temp%15` 改为 `temp/10%15`，便可将移动速度降低为原来的 10 倍，跑马灯显示效果便可以看清了。

### 2. 对此次实验的感想

在这次实验中，我更加深入地了解了汇编语言与 C/C++ 在操作系统编程上的对接，进一步熟悉了 Ubuntu 下的操作。这次实验任务虽然不简单，但是过程中在正确的引导下成功少走了不少弯路，增强了对操作系统学习的自信心。

这次实验中我也开始接触到了时钟中断等更加复杂的知识，理解起来比较困难，但是也收获很多。我非常期待未来实现自己的操作系统。