Operating System Design & Implementation

Lab3: X86 I/O System and Interrupt

TA: Rudy

Objective:

In this lab you can learn

* Understand how the x86 system real/ protected mode and basic memory segmentation mechanism.
* Understand the basic I/O mechanism.
* Understand how to write an x86 interrupt service routine (ISR).
* Implement keyboard and timer interrupt initial functions.

# Trace the simple kernel code

In this lab we already prepared a simple kernel in lab3 folder. You can clone it from the Gitlab.

|  |
| --- |
| $ git clone <http://grass8.cs.nctu.edu.tw:8888/2017/nctuos.git> |

After cloning the repository, you can just use below commands to start this kernel. Moreover you can use the qemu monitor to dump more register information that will help you debugging the OS.

Reference: <http://en.wikibooks.org/wiki/QEMU/Monitor>

|  |
| --- |
| $make all  $qemu –hda kernel.img –monitor stdio |

## Files

|  |  |
| --- | --- |
| File Name | Description |
| boot/boot.S | A simple boot loader. It only changes CPU into protected mode and setup basic GDT. |
| boot/main.c | A simple ELF loader that will load the kernel image to expect memory address. |
| kernel/entry.S | Kernel entry, there we just setup 8\*4096 bytes as kernel stack space and jump into C environment. |
| kernel/main.c | Kernel initial function |
| kernel/picirq.c | Programmable interrupt controller driver, it is used for setup external hardware interrupt. |
| kernel/kbd.c | Keyboard driver, used for read character from keyboard. |
| kernel/screen.c | Simple video driver, it allows you output string to screen. |
| kernel/timer.c | Simple system clock driver |
| kernel/trap.c | Trap handler |
| kernel/trap\_entry.S | The trap/interrupt entry, you need to define the interrupt entry point here. |
| kernel/shell.c | A simple command shell, You can use it to debug your kernel. |
| kernel/kern.ld | Kernel linker script, it tells linker how kernel memory placement is. |
| Lib/\*.c | In this folder we prepared some useful library for you (Such as printf, memcpy, strlen…) |

# Lab Background

See the lab3\_reference.pdf file and trace the lab3 source code.

# Lab3 Requirements

## Setup GDT

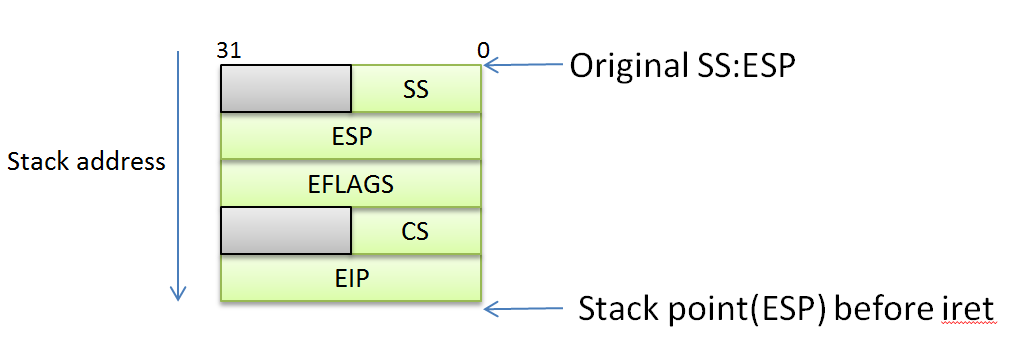
In x86, it leverages segmentation to implement isolation. To utilize this feature, we need to setup GDT before entering the protected mode from real mode. You need to modify boot/boot.S

## Setup IDT

In x86, the interrupt vector table is implemented by IDT data structure. After filling up our interrupt vector table (idt), we need to use lidt command with information related to the table. You need to modify kernel/trap.c

## Setup the trap stack same as Trapframe structure

In x86 system when a trap/interrupt occurred CPU will push some information into stack such as program counter (CS:EIP), stack point(SS:ESP) and error flag (EFLAGS) before trap. The placement will like below picture.



In this lab you need to declare an interface of interrupt handler leveraging provided macro and push all needed registers for later usage as Trapframe data structure. You need to modify kernel/trap\_entry.S

## Setup keyboard and timer interrupt in IDT

Please implement trap\_init() and trap\_dispatch() inside kernel/trap.c to initialize the IDT with entries of keyboard and timer interrupt.

## Add kerninfo command

In shell.c, complete the mon\_kerninfo function to print the kernel code and data size when user input kerninfo command.

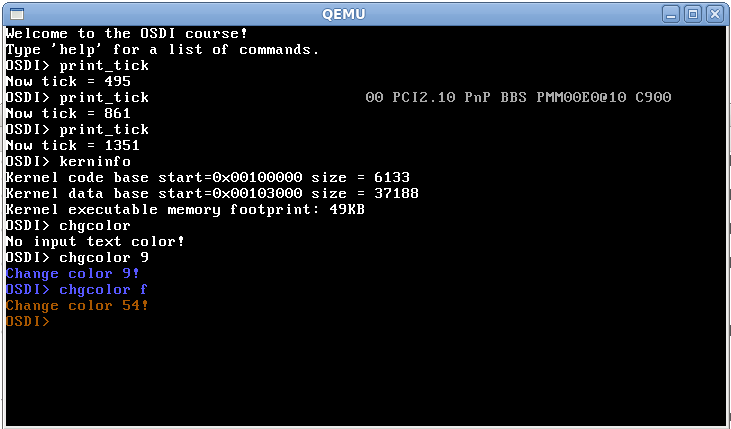
Hint: You can use the symbol from linker script (kern.ld) to calculate the section and memory footprint size.

## Add chgcolor command

To understand better of how the VGA screen works, add a command “*chgcolor*” command to shell.c. When user input this command, the screen changes the text color.

# Lab Result Demo

In this demo, you need to show result just like the following picture, and simply explain the interrupt flow.



After you complete the lab, don’t forget push your modification to gitlab.

Please create another project called nctuos\_{YOUR STUDENT\_ID}. You don’t have to submit additional patch.