Operating System Design & Implementation

A031521　清大資工所　王維寬

Lab1 Establish Lab Environment

Lab goal

1. 建制 OSDI 開發環境(VMware workstation + fedora)
2. 使用 gdb 與 QEMU 進行 kernel code debug

Todo

- **Debug the Makefile error**
  a. makefile 縮排錯誤
  b. tools/build.sh 沒有 execute 權限

- **Using gdb**
  a. **qemu -m 16M -boot a -fda Image -hda ../osdi.img -s -S -serial stdio**
  b. Open other console
  c. cd linux-0.11 gdb tools/system
  d. (gdb) target remote localhost:1234

- **Debug the kernel code error**
  a. There is a excess panic() in main.c
     - 當 kernel 遇到 bug，或硬體錯誤，kernel 呼叫 panic()來停止 OS 並印出錯誤資訊
  b. The const number NR_TASKS is zero(should be 64)
     - NR_TASKS 位於 include/linux/sched.h，其代表 OS 允許 task 的最大數量。若為 0，find_empty_process()(位於 kernel/fork.c)會出現錯誤。所以必須將其設為 64。

QEMU

QEMU 一個 Open source code 的 x86 虛擬機。

Parameter

-m *X* ： 指定 RAM 大小為 ***X***

-fda file
-fdb file： 選擇 ***file*** 當 floppy disk image

-hda *file*
-hdb *file*
-hdc *file*
-hdd *file* ： 選擇 ***file*** 當 hard disk image

-boot [a | c | d] ： 使用 **(a)**floppy disk **(c)**hard disk **(d)**CD-ROM **(n)**network

-s : shorthand for -gdb tcp::1234
-S： 在啟動時暫停，直到 gdb 下達繼續執行的指令

-serial *dev*： 選擇 serial port 為 **dev**

GDB

gdb 是 GNU 軟體所使用的標準偵錯器，支援的偵錯語言有 ： C、C++、Pascal、FORTRAN

Parameter

b : set break point (can use funciton name or line number)

n : execute program line by line

c : continue program

list : list code

backtrace : show call stack

info r : show current registers value

Ctrl+c : stop program

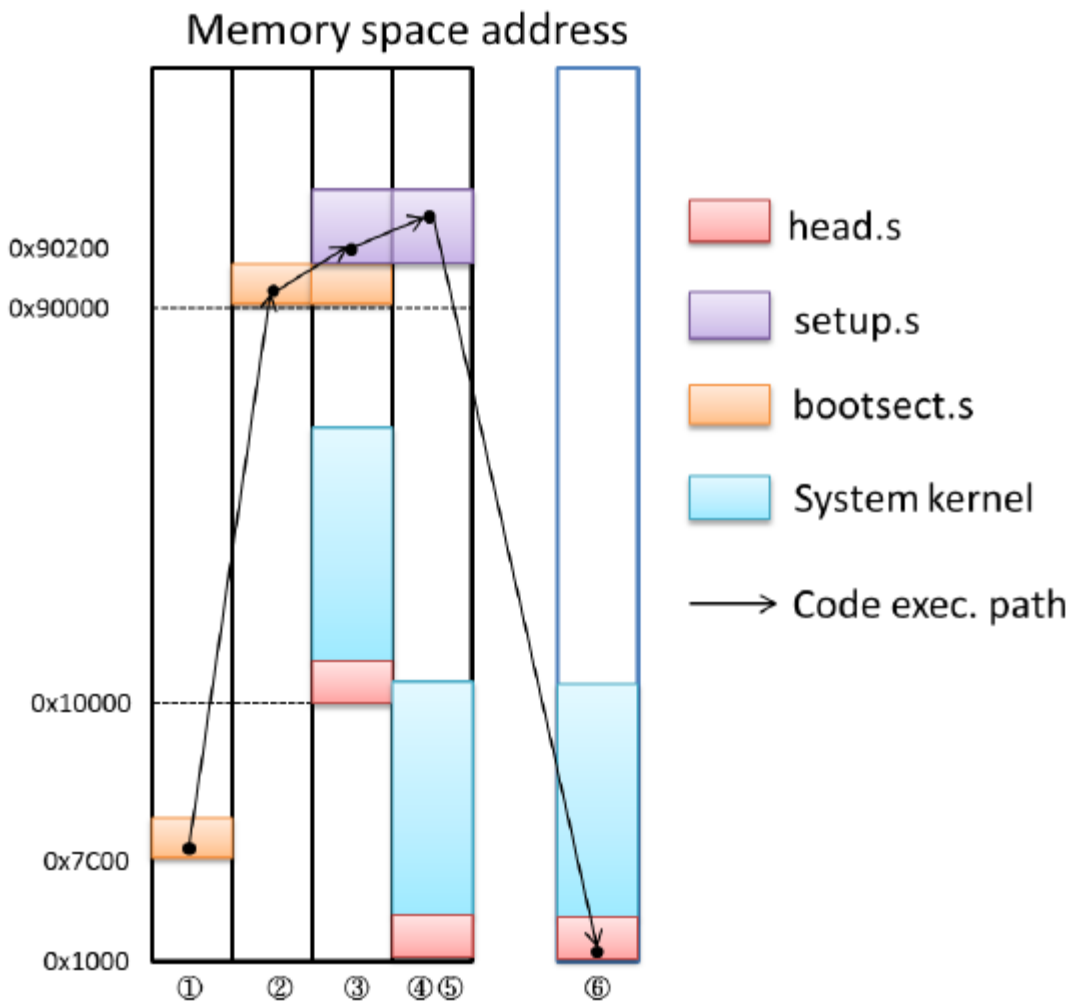
Lab2 Kernel Booting

Lab goal

- 了解 x86 開機流程
- 在 boot 後秀出 "Hello world"
- 修改 bootsect 以實現 multi-boot
- 以 int 13H 從硬碟讀取資料
- 以 int 16H 讓使用者從鍵盤輸入資料

Boot flow

1. BIOS load the bootsect from disk MBR to 0x7c00 memory address
2. boosect copy itself to 0x90000 memory address and jump to 0x90000.
3. boosect load setup from disk to 0x90200 memory address.
4. Get some system peripheral device parameters (video, root disk, keyboard,…,etc.) and jump to 0x90200.
5. Switch system into protected mode move kernel from 0x10000(64K) to 0x01000
6. Jump to 0x1000 and execute head.s for kernel boot



Todo

- Modify the shell code at tools/build.sh

```
bootsect=$1
setup=$2
system=$3
IMAGE=$4
hello_img=$5
root_dev=$6


[ ! -f "$hello_img" ] && echo "there is no hello binary file there" && exit -1
```

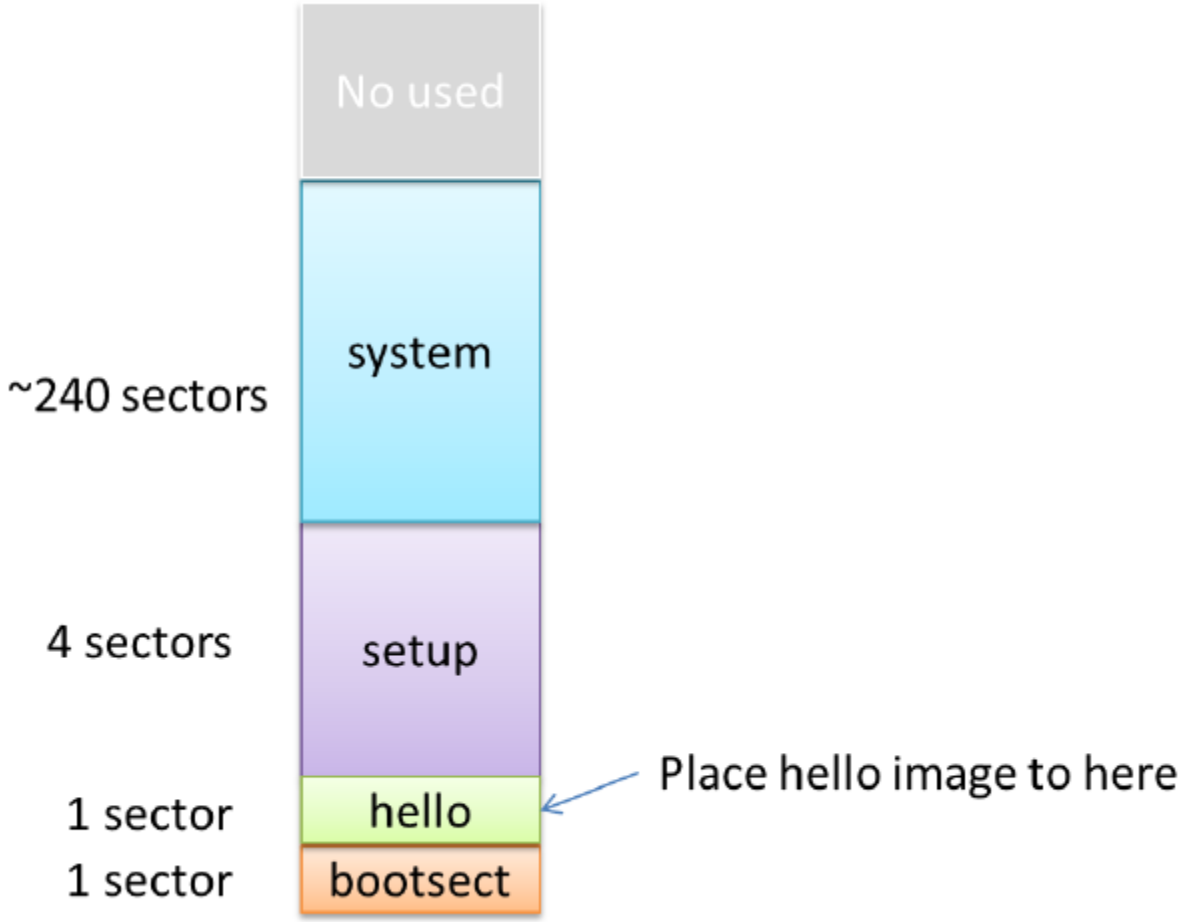
```
dd if=$ hello_img seek=1 bs=512 count=1 of=$IMAGE 2>&1 >/dev/null

# Write setup(4 * 512bytes, four sectors) to stdout
[ ! -f "$setup" ] && echo "there is no setup binary file there" && exit -1
dd if=$setup seek=2 bs=512 count=4 of=$IMAGE 2>&1 >/dev/null

[ $system_size -gt $SYS_SIZE ] && echo "the system binary is too big" &&
exit -1
dd if=$system seek=6 bs=512 count=$((2887-1-4)) of=$IMAGE 2>&1
>/dev/null
```

此檔案可將指定的 file load 進記憶體

Shell parameter

- dd ： 將來源數據 load 到指定位址
- if ： 傳送目標
- seek ： 在輸出開始處跳過指定 sector 數
- bs ： byte 數
- counter ： sector 數

由 build.sh 得知，記憶體擺放順序為 **bootsect -> hello_img ->setup ->system** bootsect 和 hello_image 各占一個 sector，setup 則占 4 個，一個 sector 大小為 512byte。所以 bootsect、hello_img、setup、system 的 seek 分別為 0、1、2、6。如下圖



- 在 Makefile 中增加 hello.s 的 make target

```
hello: hello.s
	@$(AS) -o hello.o hello.s -g
	@$(LD) $(LDFLAGS) -o hello hello.o
	@objcopy -R .pdr -R .comment -R.note -S -O binary hello
head.o: head.s
	@$(AS) -o head.o head.s
```

修改 boot/bootsect.s 以實現 multi-boot

程式流程

1. 使用者輸入'1' 或 '0'
2. 若為'1'，則正常開機
3. 若為'2'，則印出 "Hello world!"
4. 若為其他字元，則回到步驟 1

Code

```
read_input:
        mov   $0x0000, %ax       # 將 AH 設為 0(Get character form keyboard)
        int     $0x16             # 從 Keyboard 讀取一個字元，結果將在 AL
        cmp   $0x31, %al        # 若為'1'
        je      PRESS_1           # Jump to PRESS_1
        cmp   $0x32, %al        #  若為'2'
        je      PRESS_2           # Jump to PRESS_2
        jmp   PRESS_OTHER    #若以上皆非，Jump to PRESS_OTHER
PRESS_1:
        jmp   load_setup
PRESS_2:
        jmp   load_hello
PRESS_OTHER:
        jmp   read_input

load_hello:
        mov   $0x0000, %dx          # drive 0, head 0 (INT 13H 參數)
        mov   $0x0002, %cx          # sector 2, track 0 (INT 13H 參數)
        mov   $0x0200, %bx          # address = 512, in INITSEG (INT 13H 參
數)
        .equ    AX, 0x0200+1        # 1 個 sector(INT 13H 參數)
        mov     $AX, %ax            # 填入 ax
                                    #以上參數皆可從 tools/build.h 得知
        int     $0x13               # 將 sectors load 進 memory
        jnc     ok_load_hello       # 若成功，jump to ok_load_hello
        mov   $0x0000, %dx          # 重設 dx
        mov   $0x0000, %ax          # 重設 ax
        int     $0x13
```

```
        jmp    load_hello              #跳回 load_hello 重來

ok_load_hello:
        ljmp   $SETUPSEG, $0           #jump to 0x9020

load_setup:
        mov    $0x0000, %dx            # drive 0, head 0 (INT 13H 參數)
        mov    $0x0002, %cx            # sector 2, track 0 (INT 13H 參數)
        mov    $0x0200, %bx            # address = 512, in INITSEG (INT 13H 參
數)
        .equ    AX, 0x0200+SETUPLEN        # 4 個 sector(INT 13H 參數)
        mov     $AX, %ax                   # 填入 ax
                                           # 以上參數皆可從 tools/build.h 得
知
        int     $0x13                  # 將 sectors load 進 memory
        jnc     ok_load_setup          # 若成功，jump to ok_load_hello
        mov    $0x0000, %dx            # 重設 dx
        mov    $0x0000, %ax            # 重設 ax
        int     $0x13
        jmp    load_setup              #跳回 load_setup 重來

ok_load_setup:
```

INT 13H

這個 x86 指令會觸發 interrupt，並且將 sectors 從硬碟 load 進 Memory

Parameter

- AH = 02h
- AL = number of sectors to read (must be nonzero)
  CH = low eight bits of cylinder number
- CL = sector number 1-63 (bits 0-5)
- high two bits of cylinder (bits 6-7, hard disk only)
  DH = head number
- DL = drive number (bit 7 set for hard disk)
- ES:BX -> data buffer

Return

- CF set on error if AH = 11h (corrected ECC error), AL = burst length
- CF clear if successful(所以 jnc 才會跳到 ok_load_hello)
- AH = status

AL = number of sectors transferred (only valid if CF set for some BIOSes)

INT 16H

這個 x86 指令會觸發 interrupt，並從鍵盤讀取一個 character

Parameter

- AH = 00h

Return

- AH = BIOS scan code
- AL = ASCII character

Hint

在修改 tools/build.sh 時，有權限問題。當使用 chmod 出現 "is not in the sudoers file. This incident will be reported." 可使用 su - 進入 super user，再使用 chmod

Lab3 x86 I/O System and Interrupt

Lab goal

- 了解 trap 是如何註冊及產生的
- Shell 如何運作
- 註冊 timer handler
- 註冊 keyboard handler

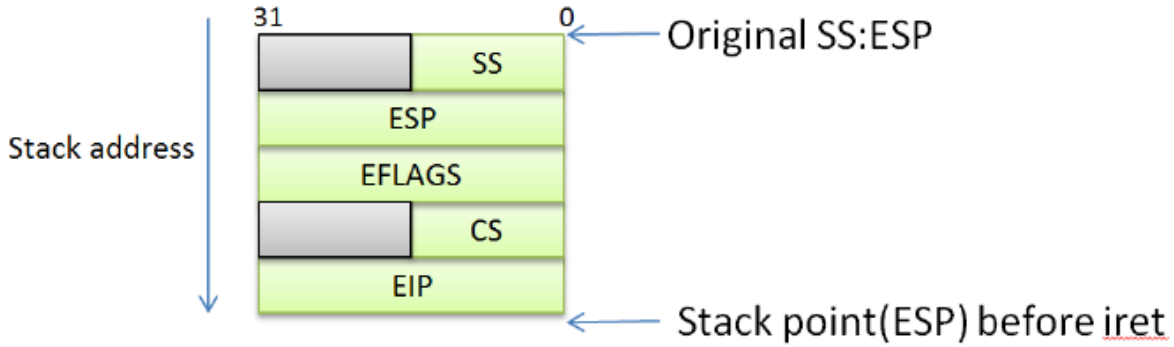
Todo

在 trap_entry.S 定義 interrupt entry point

code

```
TRAPHANDLER_NOEC(Default_ISR, T_DEFAULT)
#註冊 timer interrupt, _idt_irq0_timer 為 entry name
TRAPHANDLER_NOEC(_idt_irq0_timer, IRQ_OFFSET+IRQ_TIMER)
#註冊 keyboard interrupt, _idt_irq1_kbd 為 entry name
TRAPHANDLER_NOEC(_idt_irq1_kbd, IRQ_OFFSET+IRQ_KBD)


.globl default_trap_handler;
_alltraps:

        push  %ds
        push  %es  #push ds 和 es
        pushal  #push trapframe structure 的 register
        mov   $GD_KD, %ax  #將 GD_KD load 到 ax
        mov   %ax, %ds #將 ax load 到 ds
        mov   %ax, %es #將 ax load 到 es

        push  %esp #將 trapframe 的 stack point push 進 stack(trap_handler
要用)
        call    default_trap_handler # Jump to deflaut trap handler
        add   $4, %esp
        popal
        popl   %es
        popl   %ds

        add $8, %esp # Cleans up the pushed error code and pushed ISR number
        iret # pops 5 things at once: CS, EIP, EFLAGS, SS, and ESP!
```

Trapframe structure



Trapframe ： 進入 trap 所需要的 register 組

在 trap.c 指定 trap handler

code

```
void trap_init()
{
        int i;
        extern void Default_ISR();
        extern void _idt_irq0_timer();
        extern void _idt_irq1_kbd();

        for (i = 0; i < 256; i++)
        {
                SETGATE(idt[i], 1, GD_KT, Default_ISR, 0);
        }

        /* Setup keboard trap */
        SETGATE(idt[IRQ_OFFSET+IRQ_KBD], 0, GD_KT, _idt_irq1_kbd, 0);

        /* Setup timer trap*/
        SETGATE(idt[IRQ_OFFSET+IRQ_TIMER], 0, GD_KT, _idt_irq0_timer, 0)

        lidt(&idt_pd);
}
static void trap_dispatch(struct Trapframe *tf)
{

        switch(tf->tf_trapno){ //檢查是 tf 是何種 trap
```

```
            case IRQ_OFFSET + IRQ_KBD: //若為 keyboard，執行 kbd_intr()
                kbd_intr();
                break;

            case IRQ_OFFSET + IRQ_TIMER: //若為 timer，執行
timer_handler()
                timer_handler();
                break;

            default: break;
        }
}
```

在 Shell 增加 kerninfo 指令

從 kern.ld 這個 load script 中可以看出 memory 使用資訊

Code

```
int mon_kerninfo(int argc, char **argv)
{
        extern char _start, etext, _edata, __STAB_BEGIN__, end;

        cprintf("Kernel code base start = %x size=%d\n",&_start,&etext-&_start);
        cprintf("Kernel data base start = %x size=%d\n",&_edata,&end-&_edata);
        cprintf("Kernel executable memory footprint: %dKB\n",(&end-
&_start)/1024);
        return 0;
}
```

可用 extern char *symbol_name* ，來取得 kern.ld 中的 symbol

code base start address

為 _start 的位址(entry point)，code 尾端為 etext，兩者相減可得 code size

data base start address

我用 **PROVIDE(_edata = .)**在 kern.ld 中定義了 symbol “_edata”(在.data 之前)，所以_edata 為 data base start address，最尾端為 end，兩者相減可得 data size

Kernel executable memory footprint
end 與_start 相減則為 kernel 的總共用的 memory 大小

在 Shell 增加 chgcolor 指令

Code

```
int chgcolor(int argc, char **argv){

    char color = argv[1][0];

    if(color >= '0' && color <= '9' ) settextcolor(color-'0',0);
    else if(color >= 'A' && color <= 'F') settextcolor(color-'A'+0x0A,0);
    else if(color >= 'a' && color <= 'f') settextcolor(color-'a'+0x0A,0);
    else {
        cprintf("Undefined color!!\n");
        return 0;
     }

    cprintf("Change color %c!\n",color);
    return 1;
}
```

只要 call settextcolor function，變可指定文字顏色

Lab4: Minimal Kernel

Lab goal

- 了解 context switch 運作原理
- 實作 fork

- 實作簡易的 schedule routine

To do

產生一個 task

code

```
int task_create()
{

	int i = 0;
	Task *ts = NULL;

	/*若找到一個 task 為 free 或 stop，使用它*/
	for (i = 0; i < NR_TASKS; i++)
	{
		if (tasks[i].state == TASK_FREE || tasks[i].state == TASK_STOP)
		{
			ts = &(tasks[i]);
			break;
		}
	}
	if (i >= NR_TASKS)
		return -1;

	/* Initial Trapframe */
	memset( &(ts->tf), 0, sizeof(ts->tf));

	ts->tf.tf_cs = GD_UT | 0x03;
	ts->tf.tf_ds = GD_UD | 0x03;
	ts->tf.tf_es = GD_UD | 0x03;
	ts->tf.tf_ss = GD_UD | 0x03;
	ts->tf.tf_esp = ts->usr_stack + USR_STACK_SIZE;


	ts->task_id = i; /* task id 為 i */
	ts->state = TASK_RUNNABLE; /* state 設為 runnable，等 scheduler run 它 */
```

```
        /* parrent 為 current task */
        if(cur_task) ts->parent_id = cur_task->task_id;
        else        ts->parent_id = 0;

        ts->remind_ticks = 100;


        return i;
}

int sys_fork()
{
        int pid = -1;

        /* Initial task space */
        pid = task_create();

        if (pid < 0 )
                return -1;

        if ((uint32_t)cur_task != NULL)
        {
                /* Copy parent's tf to new task's tf */
                tasks[pid].tf = cur_task->tf;
                /* 將 current task 的 stack 複製給新的 task*/
                memcpy(tasks[pid].usr_stack, cur_task->usr_stack,
USR_STACK_SIZE);

        /*以 current task 的 stack pointer 的 offset，並加到新的 task 的 stack
start address，計算出新的 task 的 stack pointer*/
        tasks[pid].tf.tf_esp =  cur_task->tf.tf_esp - (uintptr_t)cur_task->usr_stack +
(uintptr_t)tasks[pid].usr_stack;
                /*fork()時，若為 parrent 則 return0，若為 child 則 return 自己的
ID */
                tasks[pid].tf.tf_regs.reg_eax = tasks[pid].task_id;

                return 0;
        }
```

```
        return 0;


}
```

實作簡易 scheduler

程式流程

從 current task 的 ID 開始往下搜尋，若有 task 為 runnable，則將它設為 runnning，若 current task 為 running，則將它改為 runnable。並將 current task 改為新的 task，並以

env_pop_tf(&tasks[i].tf)跳回 user node

code

```
void sched_yield(void)
{

        extern Task tasks[];
        extern Task *cur_task;
        int i, n;
        int next_i = 0;

                if(cur_task) i = cur_task->task_id+1;
        else        i = 0;

        if(i>NR_TASKS) i = 0;

        for( n=0 ; n<NR_TASKS ; n++ ){
          if(tasks[i].state == TASK_RUNNABLE){
                if(cur_task->state == TASK_RUNNING) cur_task->state =
TASK_RUNNABLE;
                cur_task = &tasks[i];
                tasks[i].state = TASK_RUNNING;
                env_pop_tf(&tasks[i].tf);
         }
          i++;
          if(i==NR_TASKS) i=0;
```

```
    }
}
```

Lab5: Kernel Lock

Questions

2.1 因為 env_pop_tf() 中的 IRET 指令，會將 EEFLAG register 中的 IF flag 設為 1，所以 kernel 自然會恢復 unlock 的狀態

2.2 Works fine，因為進入 trap 之後，所有東西都 lock 住了，當然不會有問題

2.3 無法 work。

```
cur_task->tf = *tf;
tf = &(cur_task->tf);
```

這兩行 code 中，cur_task 為 share variable，所以可能被其他的 task 更改，必須以 lock() 保護

2.4

因為在執行 trap handler 時，並沒有 lock，所以必須在有使用到 share variable 的地方進行 lock。以下為進行 lock 的 function

- **sched.c**
  - **sched_yield**
- **syscall.c**
  - **sys_kill**
  - **sys_fork**
  - **sys_sleep**
- **timer.c**
  - **timer_handler**