Report for the 1st lab

姓名：黄骏齐　　　学号：2100012956

2023年10月7日

exercise2

Set register `%ss` and `%esp`(`%esp` to `0x7000`).

Get some data from disk using `in` and `out`.

Set PE in `%cr0` to move in the protected mode and jump.

exercise3

After `%cr0` is reset, the computer enter 32-bit mode, through setting the PE symble in `%cr0`.

The last instruction which boot loader executed is `call *0x10018`

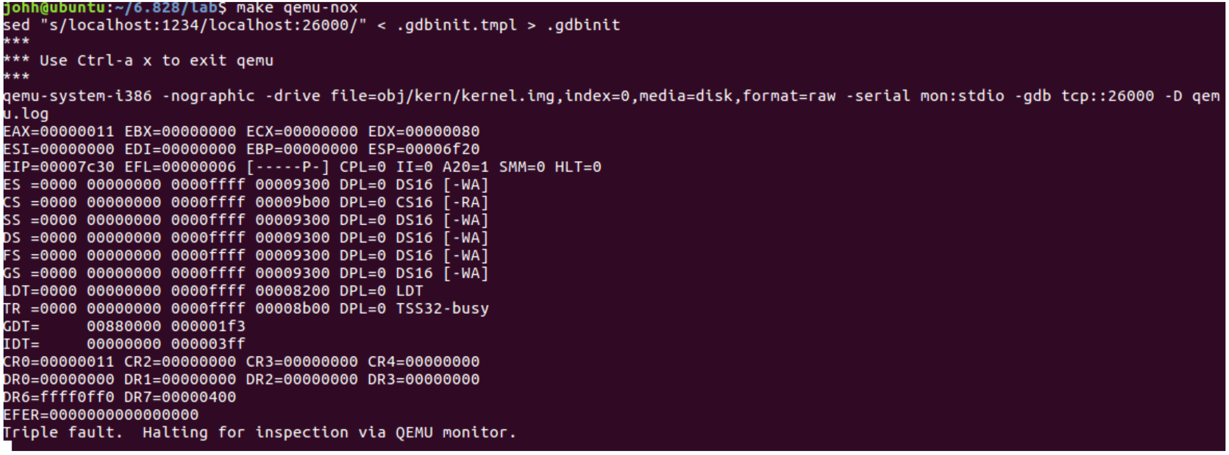
The first instruction of kernal is that at `0x0010000c`, which is `movw $0x1234, 0x472`.

exercise4

```
1: a = 000000000062FDC0, b = 0000000000BF1410, c = 0000000000000001
//a and b point to random address,c is a NULL pointer.
2: a[0] = 200, a[1] = 101, a[2] = 102, a[3] = 103
//a[x] == *(a+x)
3: a[0] = 200, a[1] = 300, a[2] = 301, a[3] = 302
//c=a, so 3[c]=c[3]=a[3]
4: a[0] = 200, a[1] = 400, a[2] = 301, a[3] = 302
//c=c+1 == c=(int*)((int c)+4)=> c points to a[1]
5: a[0] = 200, a[1] = 128144, a[2] = 256, a[3] = 302
//c points to ((char*)a + 1), and modify two integer.
6: a = 000000000062FDC0, b = 000000000062FDC4, c = 000000000062FDC1
//(int *) a + 1 = (int*)((int a) + 4); (int*)((char*)a+1) = (int*)((int a) + 1) ;
```

exercise5

I modify `0x7C00` to `0x7C2D`, and get the result below.



```
john@ubuntu:~/6.828/lab$ make qemu-nox
sed "s/localhost:1234/localhost:26000/" < .gdbinit.tmpl > .gdbinit
***
*** Use Ctrl-a x to exit qemu
***
qemu-system-i386 -nographic -drive file=obj/kern/kernel.img,index=0,media=disk,format=raw -serial mon:stdio -gdb tcp::26000 -D qem
u.log
EAX=00000011 EBX=00000000 ECX=00000000 EDX=00000080
ESI=00000000 EDI=00000000 EBP=00000000 ESP=00006f20
EIP=00007c30 EFL=00000006 [-----P-] CPL=0 II=0 A20=1 SMM=0 HLT=0
ES =0000 00000000 0000ffff 00009300 DPL=0 DS16 [-WA]
CS =0000 00000000 0000ffff 00009b00 DPL=0 CS16 [-RA]
SS =0000 00000000 0000ffff 00009300 DPL=0 DS16 [-WA]
DS =0000 00000000 0000ffff 00009300 DPL=0 DS16 [-WA]
FS =0000 00000000 0000ffff 00009300 DPL=0 DS16 [-WA]
GS =0000 00000000 0000ffff 00009300 DPL=0 DS16 [-WA]
LDT=0000 00000000 0000ffff 00008200 DPL=0 LDT
TR =0000 00000000 0000ffff 00008b00 DPL=0 TSS32-busy
GDT=     00880000 000001f3
IDT=     00000000 000003ff
CR0=00000011 CR2=00000000 CR3=00000000 CR4=00000000
DR0=00000000 DR1=00000000 DR2=00000000 DR3=00000000
DR6=ffff0ff0 DR7=00000400
EFER=0000000000000000
Triple fault.  Halting for inspection via QEMU monitor.
```

exercise6

When entering bootloader, the few words at `0x00100000` are all 0.

After loading the kernal, the few words at `0x00100000` are not all 0.

The reason is bootloader load the kernal to `0x00100000`.

exercise7

Afer this instruction, both `0x00100000` and `0xf0100000` point to the same pysical address, and after comment out this instruction, the `jmp` lead to crash.

exercise8

Fill the code just like `%h`.

```
//in lib/printfmt.c
case 'o':
    // Replace this with your code.
    num = getuint(&ap, lflag);
    base = 8;
    goto number;
```

1

`console.c` export the function `cputchar()` for `printf.c`'s function `putch`, which is used to print a single character. In the function `vcprintf` in `printf.c`, push `putch` as an argument to the function `vprintfmt`.

2

When the screen is full, move out the first line, move up the other lines to insert the new line.

3

`fmt` points to the char `x`(the 1st char of the string `x %d, y %x, z %d\n`)，`ap` points to the first argument (value `x`)

1. `vcprintf (fmt=0xf0101937 "x %d, y %x, z %d\n", ap=0xf010ffd4 "\001")`
2. `cons_putc(c=120)`
3. `cons_putc(c=32)`
4. `va_arg` ap→x ⇒ ap→y
5. `cons_putc (c=49)`
6. `cons_putc (c=44)`
7. `cons_putc (c=32)`
8. `cons_putc (c=121)`
9. `cons_putc (c=32)`
10. `va_arg` ap→y ⇒ ap-→z
11. `cons_putc (c=51)`
12. `cons_putc (c=44)`
13. `cons_putc (c=32)`
14. `cons_putc (c=122)`
15. `cons_putc (c=32)`
16. `va_arg` ap→z ⇒ ap→the address next z in the stack.
17. `cons_putc (c=52)`
18. `cons_putc (c=10)`

4

”He110 World”
Modify i to 0x726c6400.
No need to change 57616.

5

the value at the address next `&x` in the stack.

The `ap` will point to the value next `&x` in the stack and print it to the screen as an `int`.

6

I use another stack(manually) to offset the effect of reversal.

exercise9

There is a instrction in `entry.S` that `movl $(bootstacktop),%esp` to initialize the stack.

The virtual address `0xf0110000` is the end of the stack.

`.data` reserves the space of the stack.

exercise10

1. push the arguments reversely;
2. push the return address(address that the next instrction of `call`);
3. push the `%ebp` register of the last function;
4. save `%esp` in the `%ebp` register;
5. push callee registers to the stack;
6. push some local values.

Every execution of `test_backtrace` will make `%esp` move 32 byte (8 32-bit words.)

They're `%eip`,`%ebp`,`%ebx`,and 5 arguments.

exercise11,12

The type is `N_SLINE`

```
//in kern\kdebug.c
    stab_binsearch(stabs,&lline,&rline,N_SLINE,addr);
    if (lline>rline) return -1;
    info->eip_line = rline-lfile;
```

从汇编可以看出，进入函数时，先执行`push %ebp`再`mov %esp %ebp`，因此：

`*ebp` saves the `%ebp` this function.

`*(ebp+1)` saves the `%ebp` last function.

`*(ebp+2)` to `*(ebp+6)` is the first 5 arguments.

```
//in kern\monitor.c
int
mon_backtrace(int argc, char **argv, struct Trapframe *tf)
{
    cprintf("Stack␣backtrace:\n");
    int* ebp = ((int*)read_ebp());
    while (ebp != 0)
    {
        int eip = *(ebp+1);
        cprintf("␣␣ebp␣%08x␣␣eip␣%08x␣␣args␣%08x␣%08x␣%08x␣%08x␣%08x\n",ebp+1,eip,*(ebp
            +2),*(ebp+3),*(ebp+4),*(ebp+5),*(ebp+6));
        struct Eipdebuginfo info;
        int tmp = debuginfo_eip(eip, &info);
        cprintf("␣␣␣␣␣␣␣␣␣%s:%d:␣",info.eip_file,info.eip_line);
        for (int i=0;i<info.eip_fn_namelen;i++) //info.eip_fn_namelen saves the length of the
            function
            cprintf("%c",info.eip_fn_name[i]);
        cprintf("+%d\n",info.eip_fn_narg);
        ebp = (int*)(*(ebp));
    }
    return 0;
}
```