

南京航空航天大学《计算机组成原理II课程设计》报告

- 姓名：唐希
- 班级：1619201
- 学号：161920122
- 报告阶段：PA2.1
- 完成日期：2021.6.14
- 本次实验，我完成了所有内容。

目录

南京航空航天大学《计算机组成原理II课程设计》报告

目录

思考题和git

- 1.什么是操作系统？
- 2.我们不一样，吗？
- 3.操作系统的实质
- 4.程序真的结束了吗
- 5.触发系统调用
- 6.有什么不同？
- 7.段错误
- 8.对比异常与函数调用
- 9.诡异的代码
- 10.注意区分事件号和系统调用号。
- 11.打印不出来。
- 12.理解文件管理函数。
- 13.不再神秘的秘技
- 14.必答题
7. `git log` 和 `git branch` 截图

操作题

PA3.1 异常控制流

任务1：实现loader

- 1.实现简单loader，触发未实现的指令int
- 2.实现引入文件后的loader。

任务2.中断机制前的准备工作和实现中断机制

- 1.添加寄存器结构体
- 2.在 `restart()` 中正确设置寄存器初始值
- 3.实现 `lidt` 指令
- 4.实现 `int` 指令

任务3.重新组织TrapFrame结构体

`pusha`

`_RegSet`

任务4：实现系统调用

- 1.完善 `do_event`
- 2.实现正确的 `SYSCALL_ARGx()` 宏
- 3.添加现阶段需要的所有系统调用
- 4.实现 `popa` 和 `iret` 指令

任务5：在nanos-lite上运行hello world

[任务6: 实现堆区管理](#)
[任务7: 让loader使用文件](#)
[任务8: 实现完整的文件系统和系统调用](#)
[任务9: 把VGA显存抽象成文件](#)
[任务10: 把设备输入抽象成文件](#)
[任务11: 在 NEMU 中运行仙剑奇侠传](#)
[遇到的问题及解决方法](#)
[实验心得](#)
[其他备注](#)

思考题和git

1.什么是操作系统?

是管理计算机硬件与软件资源的计算机程序，本质以一种软件，是最基本的软件。

2.我们不一样，吗？

nanos-lite相对于AM来说实现了更多的功能。但本质上是相同的。

3.操作系统的实质

操作系统就是一个大型的程序。

4.程序真的结束了吗

已经结束嘞，main函数执行之前，初始化相关资源。执行之后，获取main函数的返回值，然后退出程序。`concat` 连接为 `exec_name` 的形式

5.触发系统调用

编写一个程序hello.c。

```
const char str[] = "Hello world!\n";

int main() {
    asm volatile ("movl $4, %eax;"
                  "movl $1, %ebx;"
                  "movl $str, %ecx;"
                  "movl $13, %edx;"
                  "int $0x80");

    return 0;
}
```

编译运行

6.有什么不同？

与函数调用的过程十分类似。函数调用时，获取调用函数的起始地址，并跳转过去。在触发函数调用前，会保存相关寄存器到栈中，函数调用完毕后再恢复。

可以。因为这个过程跟函数调用的过程基本一致。

“服务程序”的工作都是硬件自动完成的，不需要程序员编写指令来完成相应的内容。在函数运行的过程中遇到异常，才会触发。而用户编写的函数则要指令来完成工作。

7.段错误

段错误就是访问的内存超过了程序给的内存空间，编译的时候只会检查语法，不检查具体的指令操作，只有在访问的时候访问到不访问的地方时才会报错。

8.对比异常与函数调用

异常调用需要保存寄存器，错误码，EFLAGS，寄存器等等而函数调用只要保存寄存器即可，所以异常调用需要保存更多信息。

9.诡异的代码

因为在构造trapframe的时候改变了esp的值。以指向trapframe 内容的指针（esp）作为参数，调用trap函数。把 eip 作为入口参数传进去，然后在执行 irq_handle 这个函数之前，通过 pusha，在栈帧中形成了 _RegSer 这个结构体，把 eip 作为一个结构体的起始地址，通过成员 irq 来分发事件。

10.注意区分事件号和系统调用号。

事件号是标明我们一个没有实现的系统调用事件的编号。

而系统调用号是在识别出系统调用事件后，从寄存器中取出系统调用号和输入参数，根据系统调用号查找系统调用分派表，执行相应的处理函数，并记录返回值。

11.打印不出来。

printf打印是行缓冲，读取到的字符串会先放到缓冲区里，直到一行结束或者整个程序结束，才输出到屏幕，因为我们打印的字符串一行没有结束，所以就先执行后面的*p=NULL从而导致报错。

只要在字符串后面加上\n表示一行结束。

```
#include <stdio.h>
int main(){
    int *p = NULL;
    printf("I am here!\n");//换行
    *p = 10;
    return 0;
}
```

12.理解文件管理函数。

fs_open():用文件名参数到文件描述符表中匹配并返回下标,没找到则报错.

fs_read():通过fd参数获取文件偏移和长度,再从ramdisk或dispinfo中读取数据到buf中.

fs_write():通过fd选择写方式.若是文件写,则计算偏移和读取长度进行文件读写.

fs_lseek():通过whence选择移动文件偏移的方式,然后将新的偏移赋给对应文件描述符.

fs_close():直接返回0.

13.不再神秘的秘技

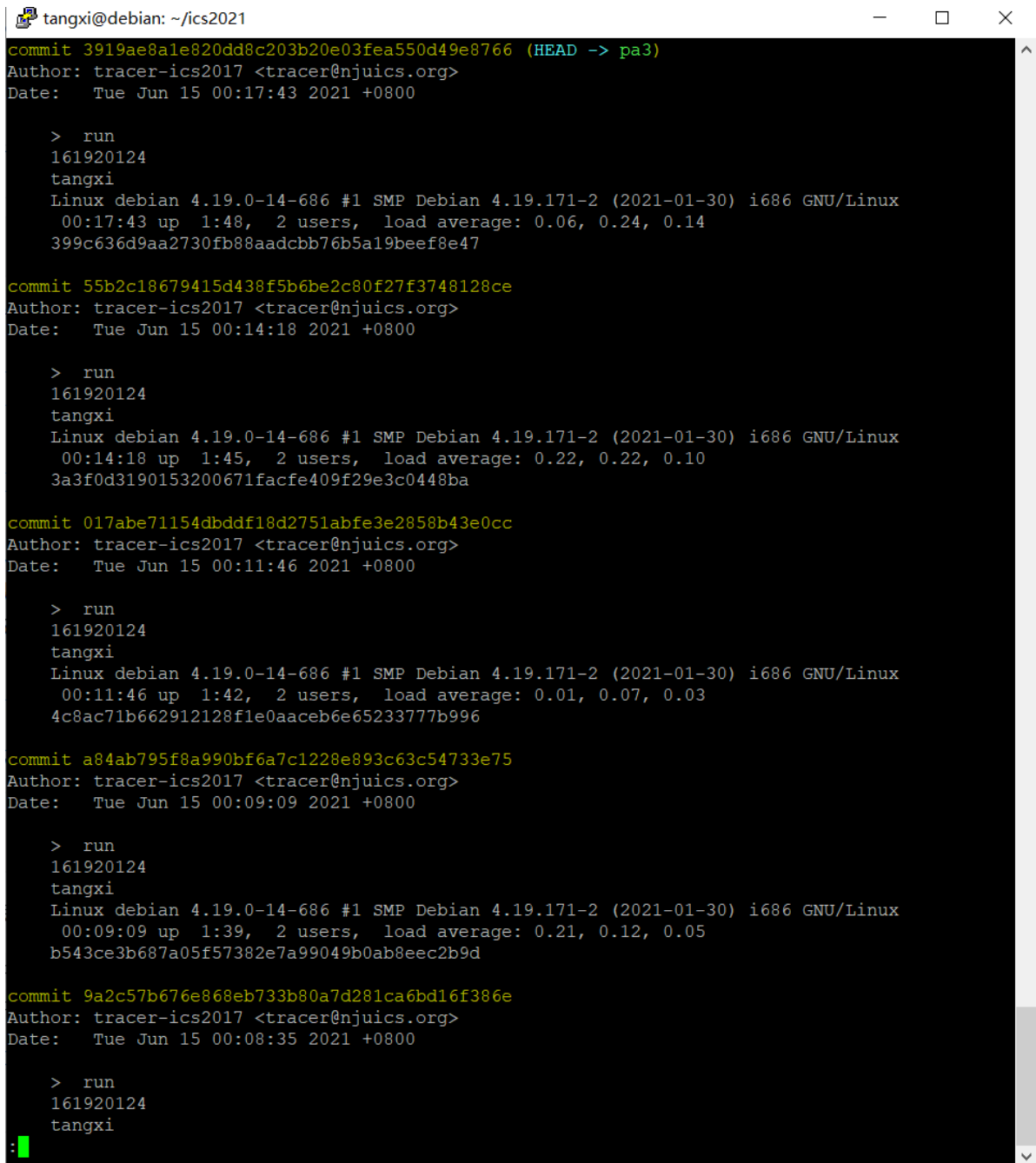
变量类型是无符号数之类的，在在机器数的模运算下溢出

14.必答题

存档读取：PAL_LoadGame()先打开指定文件然后调用fread()从文件里读取存档相关信息（其中包括调用nanos.c里的_read()以及syscall.c中的sys_raed()），随后关闭文件并把读取到的信息赋值（用fs_write()修改），接着使用AM提供的memcpy()拷贝数据，最后使用nemu的内存映射I/O修改内存。

更新屏幕：redraw()调用ndl.c里面的NDL_DrawRect()来绘制矩形，NDL_Render()把VGA显存抽象成文件，它们都调用了nanos-lite中的接口，最后nemu把文件通过I/O接口显示到屏幕上。

7. git log和git branch截图



```
tangxi@debian: ~/ics2021
commit 3919ae8a1e820dd8c203b20e03fea550d49e8766 (HEAD -> pa3)
Author: tracer-ics2017 <tracer@njuics.org>
Date: Tue Jun 15 00:17:43 2021 +0800

    > run
    161920124
    tangxi
    Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux
    00:17:43 up 1:48, 2 users, load average: 0.06, 0.24, 0.14
    399c636d9aa2730fb88aadcb76b5a19beef8e47

commit 55b2c18679415d438f5b6be2c80f27f3748128ce
Author: tracer-ics2017 <tracer@njuics.org>
Date: Tue Jun 15 00:14:18 2021 +0800

    > run
    161920124
    tangxi
    Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux
    00:14:18 up 1:45, 2 users, load average: 0.22, 0.22, 0.10
    3a3f0d3190153200671facfe409f29e3c0448ba

commit 017abe71154dbddf18d2751abfe3e2858b43e0cc
Author: tracer-ics2017 <tracer@njuics.org>
Date: Tue Jun 15 00:11:46 2021 +0800

    > run
    161920124
    tangxi
    Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux
    00:11:46 up 1:42, 2 users, load average: 0.01, 0.07, 0.03
    4c8ac71b662912128f1e0aaceb6e65233777b996

commit a84ab795f8a990bf6a7c1228e893c63c54733e75
Author: tracer-ics2017 <tracer@njuics.org>
Date: Tue Jun 15 00:09:09 2021 +0800

    > run
    161920124
    tangxi
    Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux
    00:09:09 up 1:39, 2 users, load average: 0.21, 0.12, 0.05
    b543ce3b687a05f57382e7a99049b0ab8eec2b9d

commit 9a2c57b676e868eb733b80a7d281ca6bd16f386e
Author: tracer-ics2017 <tracer@njuics.org>
Date: Tue Jun 15 00:08:35 2021 +0800

    > run
    161920124
    tangxi
```

```
tangxi@debian:~/ics2021$ git branch
master
pa0
pa1
pa2
* pa3
tangxi@debian:~/ics2021$
```

操作题

PA3.1 异常控制流

任务1：实现loader

1.实现简单loader，触发未实现的指令int

打开 `loader.c` 文因为 `ramdisk` 中目前只要一个文件，所以可以忽略 `filename` 参数。去实现 `ramdisk_read` 函数，其中第一个参数填入 `DEFAULT_ENTRY`，偏移量为 0，长度为 `ramdisk` 的大小即可。

```
void ramdisk_read(void *buf, off_t offset, size_t len);
void ramdisk_write(const void *buf, off_t offset, size_t len);
size_t get_ramdisk_size();

uintptr_t loader(_Protect *as, const char *filename) {
    ramdisk_read(DEFAULT_ENTRY, 0, get_ramdisk_size());
    return (uintptr_t)DEFAULT_ENTRY;
}
```

然后在nanos-lite目录下运行make run，发现还有为实现的指令。

```
[src/main.c,19,main] 'Hello World!' from Nanos-lite
[src/main.c,20,main] Build time: 23:07:41, Jun  7 2021
[src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x101ae0, end = 0x106e1c,
size = 21308 bytes
invalid opcode(eip = 0x04001ec0): cd 80 5b 5d c3 66 90 90 ...

There are two cases which will trigger this unexpected exception:
1. The instruction at eip = 0x04001ec0 is not implemented.
2. Something is implemented incorrectly.
Find this eip(0x04001ec0) in the disassembling result to distinguish which case
it is.
```

If it is the first case, see

for more details.

If it is the second case, remember:

- * The machine is always right!
- * Every line of untested code is always wrong!

说明已经成功加载dummy。

2.实现引入文件后的loader。

根据讲义打开 nanos-lite/Makefile 修改

```
34 update: update-ramdisk-fsimg src/syscall.h
35 | @touch src/initrd.S
```

在 loader.c 中实现loader

```
uintptr_t loader(_Protect *as, const char *filename) {
    int fd = fs_open(filename, 0, 0);
    size_t f_size = fs_filesz(fd);
    fs_read(fd, DEFAULT_ENTRY, f_size);
    fs_close(fd);

    return (uintptr_t)DEFAULT_ENTRY;
}
```

任务2.中断机制前的准备工作和实现中断机制

为了实现，任务二的内容，我们要修改寄存器结构体，跟PA1的工作一样往寄存器结构体中添加IDTR，CS寄存器并完成相关的工作

1.添加寄存器结构体

```
typedef struct {
    union{
        union {
            uint32_t _32;
            uint16_t _16;
            uint8_t _8[2];
        } gpr[8];

        /* Do NOT change the order of the GPRs' definitions. */

        /* In NEMU, rtlreg_t is exactly uint32_t. This makes RTL instructions
         * in PA2 able to directly access these registers.
         */
        struct{
            uint32_t eax,ecx,edx,ebx,esp,ebp,esi,edi;
        };
    };
    vaddr_t eip;
    union{
        uint32_t value;
        struct{
            uint32_t CF:1;
            uint32_t ZF:1;
            uint32_t SF:1;
            uint32_t IF:1;
            uint32_t OF:1;
        };
    };
    eflags;

    struct{
```

```

uint16_t limit;
uint32_t base;
} idtr;
uint16_t cs;

}CPU_state;

```

2.在 restart() 中正确设置寄存器初始值

与之前一样将EFLAGS初始化为0x2，而CS寄存器初始值设为8

```

static inline void restart() {
    /* Set the initial instruction pointer. */
    cpu.eip = ENTRY_START;
    cpu.eflags_num = 0x2;
    cpu.cs = 8;

#ifdef DIFF_TEST
    init_qemu_reg();
#endif
}

```

3.实现lidt指令

查表可知，LIDT在gpr7中，填表

```

make_group(gp7,
    EMPTY, EMPTY, EMPTY, EX(lidt),
    EMPTY, EMPTY, EMPTY, EMPTY)

```

Operation

```

IF instruction = LIDT
THEN
    IF OperandSize = 16
    THEN IDTR.Limit:Base ← m16:24 (* 24 bits of base loaded *)
    ELSE IDTR.Limit:Base ← m16:32
    FI;
ELSE (* instruction = LGDT *)
    IF OperandSize = 16
    THEN GDTR.Limit:Base ← m16:24 (* 24 bits of base loaded *)
    ELSE GDTR.Limit:Base ← m16:32;
    FI;
FI;

```

如果operandsize是16/32，则limit读取16位，base读取24/32位。

进入 `nemu/src/cpu/exec/system.c` 中完成 `make_EHelper(lidt)` 函数，

```

make_EHelper(lidt) {
    cpu.idtr.limit = vaddr_read(id_dest->addr, 2);
    if (decoding.is_operand_size_16) {
        cpu.idtr.base = vaddr_read(id_dest->addr + 2, 4) & 0x00ffffff;
    }
    else {
        cpu.idtr.base = vaddr_read(id_dest->addr + 2, 4);
    }

    print_asm_template1(lidt);
}

```

4.实现int指令

使用INT的helper函数调用raise_intr()。因为执行int指令后保存的EIP指向的是int指令的下一条指令，所以第二个参数为decoding.seq_eip。

填表

```

make_group(gp7,
    EMPTY, EMPTY, EMPTY, EX(lidt),
    EMPTY, EMPTY, EMPTY, EMPTY)

```

```

/* 0xcc */ EMPTY, IDEXW(I ,int ,1 ), EMPTY, EMPTY,

```

进入 `nemu/src/cpu/exec/system.c` 中完成 `make_EHelper(int)` 函数

```

make_EHelper(int) {
    raise_intr(id_dest->val,decoding.seq_eip);

    print_asm("int %s", id_dest->str);

#ifdef DIFF_TEST
    diff_test_skip_nemu();
#endif
}

```

根据讲义进入 `nemu/src/cpu/intr.c` 中完成 `raise_instr()` 函数

```

void raise_intr(uint8_t NO, vaddr_t ret_addr) {
    /* TODO: Trigger an interrupt/exception with ``NO''.
     * That is, use ``NO'' to index the IDT.
     */

    vaddr_t idt_addr;
    GateDesc gateDesc;

    rtl_push((rtlreg_t*)&cpu.eflags);
    rtl_push((rtlreg_t*)&cpu.cs);
    rtl_push((rtlreg_t*)&ret_addr);

    idt_addr = cpu.idtr.base + NO * 8;
    *(uint32_t *)&gateDesc = vaddr_read(idt_addr, 4);
    *((uint32_t *)&gateDesc + 1) = vaddr_read(idt_addr + 4, 4);
}

```



```

        decoding.is_jump = 1;
        decoding.jump_eip = (gateDesc.offset_31_16 << 16) | (gateDesc.offset_15_0 &
0xffff);
    }
}

```

查阅手册可知填表如下：


```
/* 0xcc */ EX(int3), IDEXW(I,int,1), EMPTY, EMPTY,
```

运行后发现程序在60处即pusha指令没有实现

```
(nemu) c
[src/main.c,19,main] 'Hello World!' from Nanos-lite
[src/main.c,20,main] Build time: 16:43:55, Jun  8 2021
[src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x101de0, end = 0x10711c,
size = 21308 bytes
[src/main.c,27,main] Initializing interrupt/exception handler...
invalid opcode(eip = 0x00101295): 60 54 e8 36 fe ff ff 83 ...

There are two cases which will trigger this unexpected exception:
1. The instruction at eip = 0x00101295 is not implemented.
2. Something is implemented incorrectly.
Find this eip(0x00101295) in the disassembling result to distinguish which case
it is.

If it is the first case, see



```

 0x00101295 60 54 e8 36 fe ff ff
 0x00101296 83
 0x00101297 83
 0x00101298 83
 0x00101299 83
 0x0010129a 83
 0x0010129b 83
 0x0010129c 83
 0x0010129d 83
 0x0010129e 83
 0x0010129f 83
 0x00101300 83
 0x00101301 83
 0x00101302 83
 0x00101303 83
 0x00101304 83
 0x00101305 83
 0x00101306 83
 0x00101307 83
 0x00101308 83
 0x00101309 83
 0x0010130a 83
 0x0010130b 83
 0x0010130c 83
 0x0010130d 83
 0x0010130e 83
 0x0010130f 83
 0x00101310 83
 0x00101311 83
 0x00101312 83
 0x00101313 83
 0x00101314 83
 0x00101315 83
 0x00101316 83
 0x00101317 83
 0x00101318 83
 0x00101319 83
 0x0010131a 83
 0x0010131b 83
 0x0010131c 83
 0x0010131d 83
 0x0010131e 83
 0x0010131f 83
 0x00101320 83
 0x00101321 83
 0x00101322 83
 0x00101323 83
 0x00101324 83
 0x00101325 83
 0x00101326 83
 0x00101327 83
 0x00101328 83
 0x00101329 83
 0x0010132a 83
 0x0010132b 83
 0x0010132c 83
 0x0010132d 83
 0x0010132e 83
 0x0010132f 83
 0x00101330 83
 0x00101331 83
 0x00101332 83
 0x00101333 83
 0x00101334 83
 0x00101335 83
 0x00101336 83
 0x00101337 83
 0x00101338 83
 0x00101339 83
 0x0010133a 83
 0x0010133b 83
 0x0010133c 83
 0x0010133d 83
 0x0010133e 83
 0x0010133f 83
 0x00101340 83
 0x00101341 83
 0x00101342 83
 0x00101343 83
 0x00101344 83
 0x00101345 83
 0x00101346 83
 0x00101347 83
 0x00101348 83
 0x00101349 83
 0x0010134a 83
 0x0010134b 83
 0x0010134c 83
 0x0010134d 83
 0x0010134e 83
 0x0010134f 83
 0x00101350 83
 0x00101351 83
 0x00101352 83
 0x00101353 83
 0x00101354 83
 0x00101355 83
 0x00101356 83
 0x00101357 83
 0x00101358 83
 0x00101359 83
 0x0010135a 83
 0x0010135b 83
 0x0010135c 83
 0x0010135d 83
 0x0010135e 83
 0x0010135f 83
 0x00101360 83
 0x00101361 83
 0x00101362 83
 0x00101363 83
 0x00101364 83
 0x00101365 83
 0x00101366 83
 0x00101367 83
 0x00101368 83
 0x00101369 83
 0x0010136a 83
 0x0010136b 83
 0x0010136c 83
 0x0010136d 83
 0x0010136e 83
 0x0010136f 83
 0x00101370 83
 0x00101371 83
 0x00101372 83
 0x00101373 83
 0x00101374 83
 0x00101375 83
 0x00101376 83
 0x00101377 83
 0x00101378 83
 0x00101379 83
 0x0010137a 83
 0x0010137b 83
 0x0010137c 83
 0x0010137d 83
 0x0010137e 83
 0x0010137f 83
 0x00101380 83
 0x00101381 83
 0x00101382 83
 0x00101383 83
 0x00101384 83
 0x00101385 83
 0x00101386 83
 0x00101387 83
 0x00101388 83
 0x00101389 83
 0x0010138a 83
 0x0010138b 83
 0x0010138c 83
 0x0010138d 83
 0x0010138e 83
 0x0010138f 83
 0x00101390 83
 0x00101391 83
 0x00101392 83
 0x00101393 83
 0x00101394 83
 0x00101395 83
 0x00101396 83
 0x00101397 83
 0x00101398 83
 0x00101399 83
 0x0010139a 83
 0x0010139b 83
 0x0010139c 83
 0x0010139d 83
 0x0010139e 83
 0x0010139f 83
 0x001013a0 83
 0x001013a1 83
 0x001013a2 83
 0x001013a3 83
 0x001013a4 83
 0x001013a5 83
 0x001013a6 83
 0x001013a7 83
 0x001013a8 83
 0x001013a9 83
 0x001013aa 83
 0x001013ab 83
 0x001013ac 83
 0x001013ad 83
 0x001013ae 83
 0x001013af 83
 0x001013b0 83
 0x001013b1 83
 0x001013b2 83
 0x001013b3 83
 0x001013b4 83
 0x001013b5 83
 0x001013b6 83
 0x001013b7 83
 0x001013b8 83
 0x001013b9 83
 0x001013ba 83
 0x001013bb 83
 0x001013bc 83
 0x001013bd 83
 0x001013be 83
 0x001013bf 83
 0x001013c0 83
 0x001013c1 83
 0x001013c2 83
 0x001013c3 83
 0x001013c4 83
 0x001013c5 83
 0x001013c6 83
 0x001013c7 83
 0x001013c8 83
 0x001013c9 83
 0x001013ca 83
 0x001013cb 83
 0x001013cc 83
 0x001013cd 83
 0x001013ce 83
 0x001013cf 83
 0x00
```


```

任务3.重新组织TrapFrame结构体

pusha

查手册，填表

Operation

```
IF OperandSize = 16 (* PUSHA instruction *)
THEN
    Temp ← (SP);
    Push(AX);
    Push(CX);
    Push(DX);
    Push(BX);
    Push(Temp);
    Push(BP);
    Push(SI);
    Push(DI);
ELSE (* OperandSize = 32, PUSHAD instruction *)
    Temp ← (ESP);
    Push(EAX);
    Push(ECX);
    Push(EDX);
    Push(EBX);
    Push(Temp);
    Push(EBP);
    Push(ESI);
    Push(EDI);
FI;
```

```
/* 0x60 */ EX(pusha), EMPTY, EMPTY, EMPTY,
```

在 `data-mov.c` 中实现 `pusha`, 然后填表以及声明

```
make_EHelper(pusha) {
    if (decoding.is_operand_size_16) {
        t0 = reg_w(R_SP);
        //保存当前sp
        rtl_push((rtlreg_t *)&reg_w(R_AX));
        rtl_push((rtlreg_t *)&reg_w(R_CX));
        rtl_push((rtlreg_t *)&reg_w(R_DX));
        rtl_push((rtlreg_t *)&reg_w(R_BX));
        rtl_push(&t0);
        rtl_push((rtlreg_t *)&reg_w(R_BP));
        rtl_push((rtlreg_t *)&reg_w(R_SI));
        rtl_push((rtlreg_t *)&reg_w(R_DI));
    }
    else {
        t0 = reg_w(R_ESP);
        //保存当前esp
        rtl_push((rtlreg_t *)&reg_w(R_EAX));
        rtl_push((rtlreg_t *)&reg_w(R_ECX));
        rtl_push((rtlreg_t *)&reg_w(R_EDX));
        rtl_push((rtlreg_t *)&reg_w(R_EBX));
        rtl_push(&t0);
        rtl_push((rtlreg_t *)&reg_w(R_EBP));
        rtl_push((rtlreg_t *)&reg_w(R_ESI));
    }
}
```

```

    rtl_push((rtlreg_t *)&reg_w(R_EDI));
}

print_asm("pusha");
}

```

_RegSet

进入 `nexus-am/am/arch/x86-nemu/src/trap.S`

```

edi, esi, ebp, esp, ebx, edx, ecx, eax;
irq;
error_code;
ret_addr, cs, eflags; //int指令中压栈的数据

```

可以得知保存的顺序：先硬件保存EFLAGS，CS，EIP然后vecsys()函数会压入错误码以及异常号#irq，最后asm_trap()会把用户进程的通用寄存器保存到堆栈上。而且出栈的时候倒序恢复。现在来定义_Regset寄存器

```

struct _RegSet {
    uintptr_t edi, esi, ebp, esp, ebx, edx, ecx, eax;
    int irq;
    uintptr_t error, eip, cs, eflags;
};

```

然后在nanos-lite中重新运行，触发了bad trap

```

[src/main.c,19,main] 'Hello World!' from Nanos-lite
[src/main.c,20,main] Build time: 22:49:19, Jun 12 2021
[src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x101d90
size = 21308 bytes
[src/main.c,27,main] Initializing interrupt/exception handler.
[src/irq.c,5,do_event] system panic: Unhandled event ID = 8
nemu: HIT BAD TRAP at eip = 0x001000f1

```

任务4：实现系统调用

1.完善do_event

根据讲义要求，识别8号系统调用事件_EVENT_SYSCALL，然后调用do_syscall即可

```

extern _RegSet* do_syscall(_RegSet *r);

static _RegSet* do_event(_Event e, _RegSet* r) {
    switch (e.event) {
        case _EVENT_SYSCALL:
            do_syscall(r);
            break;
        default: panic("Unhandled event ID = %d", e.event);
    }

    return NULL;
}

```

2.实现正确的SYSCALL_ARGx ()宏

根据讲义可知，宏的实现如下：

```
#define SYSCALL_ARG1(r) r->eax
#define SYSCALL_ARG2(r) r->ebx
#define SYSCALL_ARG3(r) r->ecx
#define SYSCALL_ARG4(r) r->edx
```

3.添加现阶段需要的所有系统调用

首先先完善do_syscall()函数。添加如下代码

```
a[0] = SYSCALL_ARG1(r);
a[1] = SYSCALL_ARG2(r);
a[2] = SYSCALL_ARG3(r);
a[3] = SYSCALL_ARG4(r);
```

然后实现none和exit的系统调用。对于sys_none(), 该函数什么都不做，只要返回1就好

```
static inline uintptr_t sys_none(_RegSet *r) {
    //设置系统调用的返回值
    SYSCALL_ARG1(r)=1;
    return 1;
}
```

然后是sys_exit.这个函数接收一个退出状态的参数，用这个参数来调用_halt()即可

```
static inline uintptr_t sys_exit(_RegSet *r) {
    _halt(SYSCALL_ARG2(r));
    return 1;
}
```

4.实现popa和iret指令

popa

按照手册的顺序pop即可。然后填表以及声明

```
/* 0x60 */ EX(pusha), EX(popa), EMPTY, EMPTY,
```

进入 `nemu/src/cpu/exec/data-mov.c` 中完成 popa

```

make_EHelper(popa) {
    rtl_pop(&cpu.edi);
    rtl_pop(&cpu.esi);
    rtl_pop(&cpu.ebp);
    rtl_pop(&t0);
    rtl_pop(&cpu.ebx);
    rtl_pop(&cpu.edx);
    rtl_pop(&cpu.ecx);
    rtl_pop(&cpu.eax);

    print_asm("popa");
}

```

iret

根据手册，按照顺序将eip cs eflags出栈即可。填表

```
/* 0xcc */ EMPTY, IDEXW(I ,int ,1 ), EMPTY, EX(iret),
```

进入 nemu/src/cpu/exec/system.c 中完成 iret

```

make_EHelper(iret) {
    rtl_pop(&t0);
    decoding.jump_eip = t0;
    rtl_pop(&t0);
    cpu.cs = (uint16_t)t0;
    rtl_pop(&t0);
    cpu.eflags.value = t0;
    //恢复eip, cs, eflags.

    decoding.is_jump = 1;

    print_asm("iret");
}

```

运行成功

```

make[1]: Entering directory '/home/chenkai/ics2021/nemu'
./build/nemu -l /home/chenkai/ics2021/nanos-lite/build/nemu-log.txt /home/chenkai/ics2021/nanos-lite/build/nanos-lite-x86-nemu.bin
[src/monitor/diff-test/diff-test.c,96,init_diff_test] Connect to QEMU successfully
[src/monitor/monitor.c,65,load_img] The image is /home/chenkai/ics2021/nanos-lite/build/nanos-lite-x86-nemu.bin
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 16:43:55, Jun  8 2021
For help, type "help"
(nemu) c
[src/main.c,19,main] 'Hello World!' from Nanos-lite
[src/main.c,20,main] Build time: 22:01:16, Jun  8 2021
[src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x101ee0, end = 0x10721c, size = 21308 bytes
[src/main.c,27,main] Initializing interrupt/exception handler...
nemu: HIT GOOD TRAP at eip = 0x001000f1

```

任务5: 在nanos-lite上运行hello world

首先去 `do_syscall` 中实现 `SYS_write` 函数

```
static inline uintptr_t sys_write(uintptr_t fd, uintptr_t buf, uintptr_t len) {
    if (fd == 1 || fd == 2) {
        int i;
        for (i = 0; i < len; i++, buf++) {
            _putc(*(char*)buf);
        }
    }

    return len;
}
```

然后再 `do_syscall` 中添加调用

```
_RegSet* do_syscall(_RegSet *r) {
    uintptr_t a[4];
    a[0] = SYSCALL_ARG1(r);
    a[1] = SYSCALL_ARG2(r);
    a[2] = SYSCALL_ARG3(r);
    a[3] = SYSCALL_ARG4(r);
    switch (a[0]) {
        case SYS_none:
            sys_none(r);
            break;
        case SYS_exit:
            sys_exit(r);
            break;
        case SYS_write:
            SYSCALL_ARG1(r) = sys_write(a[1], a[2], a[3]);
            break;
        default:
            panic("Unhandled syscall ID = %d", a[0]);
    }
    return NULL;
}
```

不要忘记在 `navy-apps/libs/libos/src/nanos.c` 的 `_write()` 中调用系统调用接口函数

```
int _write(int fd, void *buf, size_t count){
    _syscall_(SYS_write, fd, (uintptr_t)buf, count);
}
```

然后成功运行hello world

```
[src/main.c,19,main] 'Hello World!' from Nanos-lite
[src/main.c,20,main] Build time: 02:13:31, Jun 13 2021
[src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x101f50, end = 0x10828c, size = 25404
bytes
[src/main.c,27,main] Initializing interrupt/exception handler...
Hello World!
Hello World for the 2th time
Hello World for the 3th time
Hello World for the 4th time
Hello World for the 5th time
Hello World for the 6th time
```

任务6：实现堆区管理

在 `nanos-lite` 中实现 `sys_brk` 的系统调用

```
static inline uintptr_t sys_brk(uintptr_t new_brk) {
    return (uintptr_t)mm_brk(new_brk);
}

case SYS_brk:
    SYSCALL_ARG1(r)=0;
    break;
```

而具体的 `_sbrk()` 的实现步骤：

program break 一开始的位置位于 `_end`；被调用时，根据记录的 program break 位置和参数 `increment`，计算出新 program break；通过 `sys_brk` 系统调用来让操作系统设置新 program break；若 `sys_brk` 系统调用成功，该系统调用会返回 0，此时更新之前记录的 program break 的位置，并将旧 program break 的位置作为 `_sbrk()` 的返回值返回；若该系统调用失败，`_sbrk()` 会返回 -1；进入 `navy-apps/libs/libos/src/nanos.c` 实现 `_sbrk()`

```
extern char _end;
//引用ld默认添加的符号
intptr_t program_break = (intptr_t)&_end;

...

void *_sbrk(intptr_t increment){
    intptr_t old_program_break = program_break;

    if (_syscall(SYS_brk, program_break + increment, 0, 0) == 0) {
        program_break = program_break + increment;
        return (void *)old_program_break;
    }
    else {
        return -1;
    }
}
```

任务7：让loader使用文件

在 `loader.c` 中实现

```
uintptr_t loader(_Protect *as, const char *filename) {
    int fd = fs_open(filename, 0, 0);
    size_t f_size = fs_filesz(fd);
    fs_read(fd, DEFAULT_ENTRY, f_size);
    fs_close(fd);

    return (uintptr_t)DEFAULT_ENTRY;
}
```

任务8：实现完整的文件系统和系统调用

添加系统调用，在 `syscall.c` 中的 `do_syscall` 实现所有的系统调用

```
static inline uintptr_t sys_open(uintptr_t pathname, uintptr_t flags, uintptr_t
mode) {
    return fs_open((char *)pathname, flags, mode);
}

static inline uintptr_t sys_read(uintptr_t fd, uintptr_t buf, uintptr_t len) {
    return fs_read(fd, (void*)buf, len);
}

static inline uintptr_t sys_lseek(uintptr_t fd, uintptr_t offset, uintptr_t
whence) {
    return fs_lseek(fd, offset, whence);
}

static inline uintptr_t sys_close(uintptr_t fd) {
    return fs_close(fd);
}

static inline uintptr_t sys_brk(uintptr_t new_brk) {
    return (uintptr_t)mm_brk(new_brk);
}

static inline uintptr_t sys_write(uintptr_t fd, uintptr_t buf, uintptr_t len) {
    return fs_write(fd, (void*)buf, len);
}

case SYS_open:
    SYSCALL_ARG1(r) = sys_open(SYSCALL_ARG2(r), SYSCALL_ARG3(r),
SYSCALL_ARG4(r));
    break;
case SYS_read:
    SYSCALL_ARG1(r) = sys_read(SYSCALL_ARG2(r), SYSCALL_ARG3(r),
SYSCALL_ARG4(r));
    break;
case SYS_lseek:
    SYSCALL_ARG1(r) = sys_lseek(SYSCALL_ARG2(r), SYSCALL_ARG3(r),
SYSCALL_ARG4(r));
    break;
case SYS_close:
    SYSCALL_ARG1(r) = sys_close(SYSCALL_ARG2(r));
    break;
case SYS_brk:
    SYSCALL_ARG1(r) = sys_brk(SYSCALL_ARG2(r));
    break;
```

在进入 `navy-apps/libs/libos/src/nanos.c` 中修改对应的接口函数

```
int _open(const char *path, int flags, mode_t mode) {
    return _syscall(SYS_open, (uintptr_t)path, flags, mode);
}
```



```

int _write(int fd, void *buf, size_t count){
    return _syscall_(SYS_write, fd, (uintptr_t)buf, count);
}

int _read(int fd, void *buf, size_t count) {
    return _syscall_(SYS_read, fd, (uintptr_t)buf, count);
}

int _close(int fd) {
    return _syscall_(SYS_close, fd, 0, 0);
}

off_t _lseek(int fd, off_t offset, int whence) {
    return _syscall_(SYS_lseek, fd, offset, whence);
}

```

最后不要忘记修改main.c中的参数

实现后就可以看到pass的信息

```

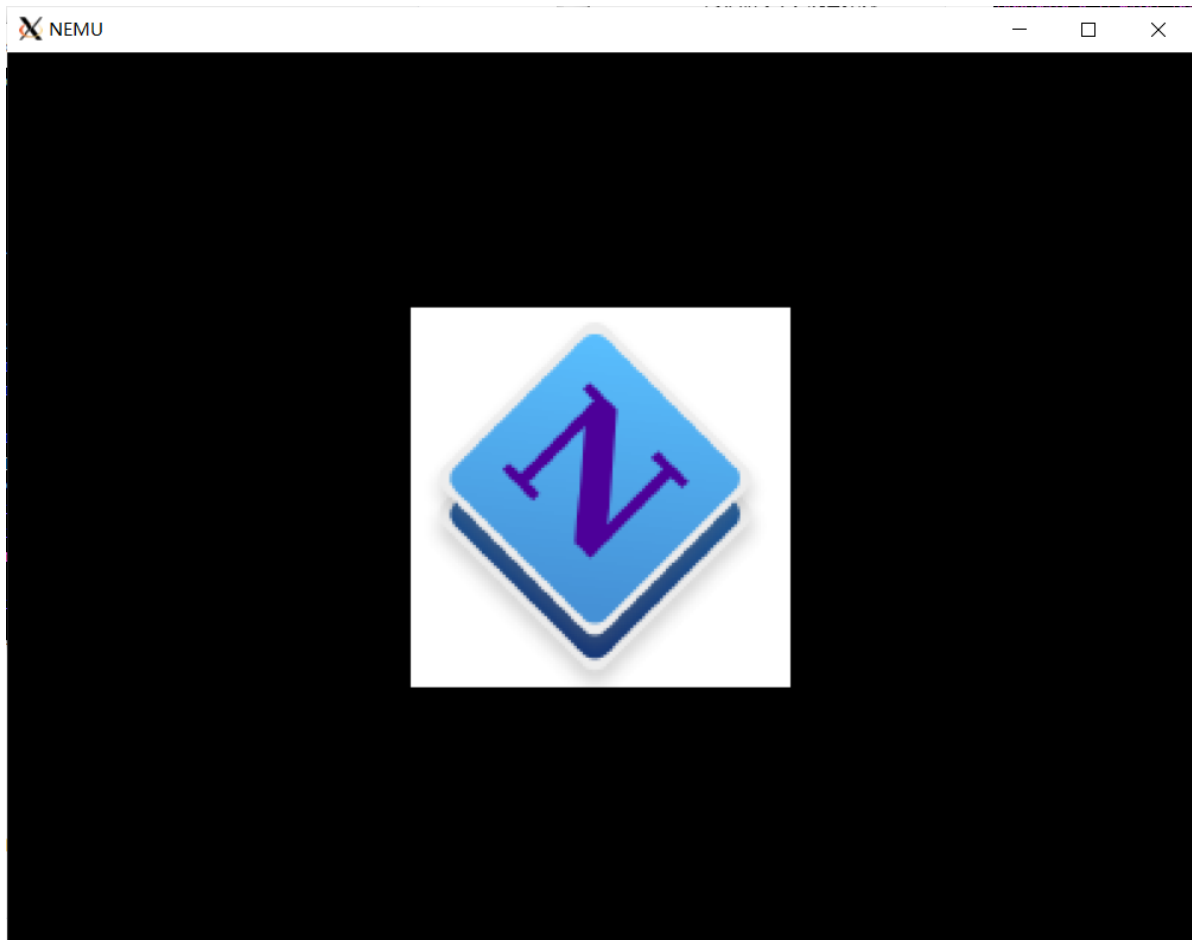
[src/monitor/monitor.c,65,load_img] The image is /home/tangxi/ics2021/nanos-lite
/build/nanos-lite-x86-nemu.bin
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 22:39:02, Jun 14 2021
For help, type "help"
(nemu) c
[src/main.c,19,main] 'Hello World!' from Nanos-lite
[src/main.c,20,main] Build time: 00:08:35, Jun 15 2021
[src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x102100, end = 0x6cc7ad9,
size = 113007065 bytes
[src/main.c,27,main] Initializing interrupt/exception handler...
PASS!!!
nemu: HIT GOOD TRAP at eip = 0x001000f1

(nemu) A

```

任务9：把VGA显存抽象成文件

修改文件名为/bin/bmptest



任务10：把设备输入抽象成文件

修改文件名为/bin/events

```
tangxi@debian: ~/ics2021/nanos-lite
make[1]: Entering directory '/home/tangxi/ics2021/nemu'
./build/nemu -l /home/tangxi/ics2021/nanos-lite/build/nemu-log.txt /home/tangxi/ics2021/nanos-lite/build/nanos-lite-x86-nemu.bin
[src/monitor/monitor.c,65,load_img] The image is /home/tangxi/ics2021/nanos-lite/build/nanos-lite-x86-nemu.bin
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 22:39:02, Jun 14 2021
For help, type "help"
(nemu) c
[src/main.c,19,main] 'Hello World!' from Nanos-lite
[src/main.c,20,main] Build time: 00:14:17, Jun 15 2021
[src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x102100, end = 0x6cc7ad9, size = 113007065 bytes
[src/main.c,27,main] Initializing interrupt/exception handler...
receive event: t 2283
receive event: t 4650
receive event: t 7659
receive event: t 10025
receive event: t 12124
receive event: t 14415
receive event: t 16546
receive event: t 18620
receive event: t 20630
```

任务11：在 NEMU 中运行仙剑奇侠传

解压文件后，更新ramdisk,最后修改main.c中的文件名即可运行



遇到的问题及解决方法

1.不知道那里的内容没做好，导致第一步就出现out of bound，尝试了很多方法都不能解决，最后只能恢复备份重新写了

实验心得

要不是备份了虚拟电脑直接痛苦面具，以后一定要记得一步步小心实现，bug总会找上门的

其他备注

无