

南京航空航天大学《计算机组成原理II课程设计》报告

- 姓名：邵震哲
- 班级：1620204
- 学号：162020130
- 报告阶段：PA3.2
- 完成日期：2022.6.7
- 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

实验内容

1. 添加分页控制相关寄存器 (10分)
2. 修改访存函数 (20 分)
3. page_translate() (30 分)
4. 修改 loader() (30 分)
5. 在分页上运行仙剑奇侠传 (10 分)

遇到的问题及解决办法

实验心得

其他备注

思考题

1. 一些问题 (25)

- 高20位是基地址，低12位是页内偏移量，加在一起一共32位
- 是必须的。若为虚拟地址，则会陷入死循环
- (1) 使用多级页表可以使得页表在内存中离散存储。
(2) 使用一级页表，需要连续的内存空间来存放所有的页表项。使用多级页表可以节省页表内存，多级页表通过只为进程实际使用的那些虚拟地址内存区请求页表来减少内存使用量

2. 空指针真的是'空'的吗? (15)

空指针应该也表示一个虚拟地址，只不过在虚拟地址转换为物理地址后，对应的物理地址会触发空指针解引用的错误。

3. 理解_map 函数 (25)

```

void _map(_Protect *p, void *va, void *pa) {
    PDE *pt = (PDE*)p->ptr;
    PDE *pde = &pt[PDX(va)];
    if (!(*pde & PTE_P)) {
        *pde = PTE_P | PTE_W | PTE_U | (uint32_t)palloc_f();
    }
    PTE *pte = &((PTE*)PTE_ADDR(*pde))[PTX(va)];
    if (!(*pte & PTE_P)) {
        *pte = PTE_P | PTE_W | PTE_U | (uint32_t)pa;
    }
}

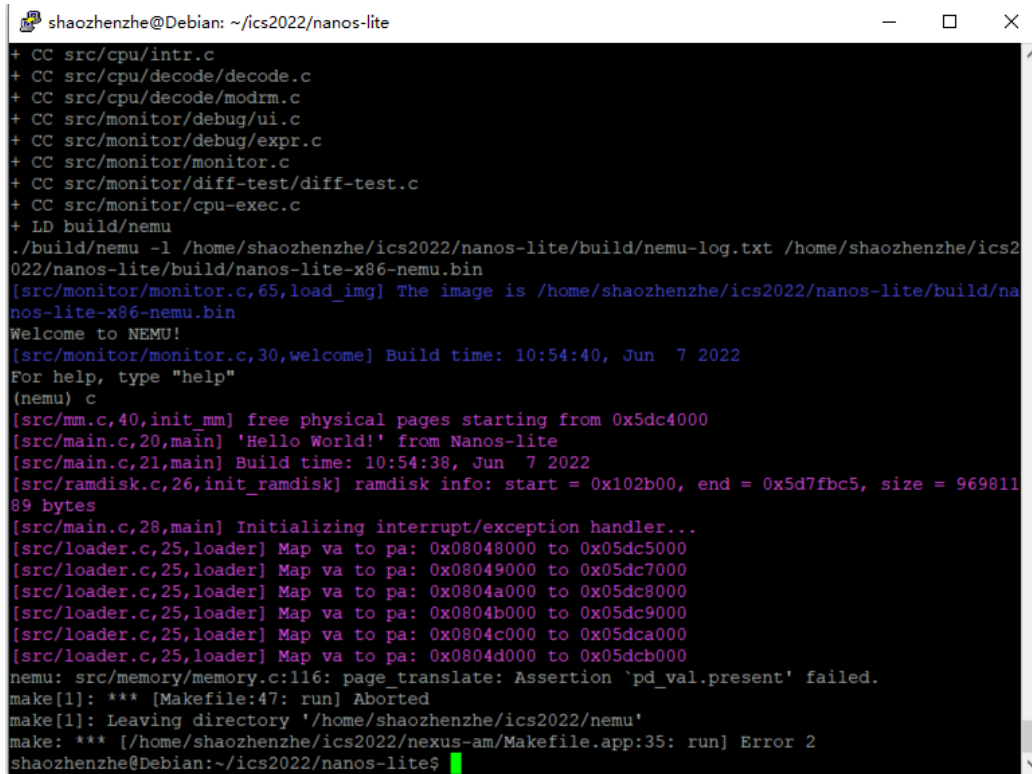
```

通过 `p->ptr` 可以获取页目录的基地址，然后根据传入的虚拟地址和基地址得到页目录项。

if语句判断是否需要申请新的页表，若需要，则可以通过回调函数 `palloc_f()` 向 Nanos-lite 获取一页空闲的物理页，把刚刚申请的物理页地址与其他标志位一并存入一个页目录里。使用这个页目录再申请一个页表项，最后把物理地址连同标志位一起放入这个页表项。

4. 内核映射的作用 (25)

发生的错误：`pd_val.present` 报错



```

shaozhenzhe@Debian: ~/ics2022/nanos-lite
+ CC src/cpu/intr.c
+ CC src/cpu/decode/decode.c
+ CC src/cpu/decode/modrm.c
+ CC src/monitor/debug/ui.c
+ CC src/monitor/debug/expr.c
+ CC src/monitor/monitor.c
+ CC src/monitor/diff-test/diff-test.c
+ CC src/monitor/cpu-exec.c
+ LD build/nemu
./build/nemu -l /home/shaozhenzhe/ics2022/nanos-lite/build/nemu-log.txt /home/shaozhenzhe/ics2022/nanos-lite/build/nanos-lite-x86-nemu.bin
[src/monitor/monitor.c,65,load_img] The image is /home/shaozhenzhe/ics2022/nanos-lite/build/nanos-lite-x86-nemu.bin
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 10:54:40, Jun  7 2022
For help, type "help"
(nemu) c
[src/mm.c,40,init_mm] free physical pages starting from 0x5dc4000
[src/main.c,20,main] 'Hello World!' from Nanos-lite
[src/main.c,21,main] Build time: 10:54:38, Jun  7 2022
[src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x102b00, end = 0x5d7fbc5, size = 96981189 bytes
[src/main.c,28,main] Initializing interrupt/exception handler...
[src/loader.c,25,loader] Map va to pa: 0x08048000 to 0x05dc5000
[src/loader.c,25,loader] Map va to pa: 0x08049000 to 0x05dc7000
[src/loader.c,25,loader] Map va to pa: 0x0804a000 to 0x05dc8000
[src/loader.c,25,loader] Map va to pa: 0x0804b000 to 0x05dc9000
[src/loader.c,25,loader] Map va to pa: 0x0804c000 to 0x05dca000
[src/loader.c,25,loader] Map va to pa: 0x0804d000 to 0x05dcb000
nemu: src/memory/memory.c:116: page_translate: Assertion 'pd_val.present' failed.
make[1]: *** [Makefile:47: run] Aborted
make[1]: Leaving directory '/home/shaozhenzhe/ics2022/nemu'
make: *** [/home/shaozhenzhe/ics2022/nexus-am/Makefile.app:35: run] Error 2
shaozhenzhe@Debian:~/ics2022/nanos-lite$

```

因为没有进行内核映射拷贝，所以没有页表来存放对应的内核区的虚拟地址，就会出现这种报错。

5. git log 和 远程仓库截图 (10)

git log截图

```
shaozhenzhe@Debian: ~/ics2022/nanos-lite
40293f4 (HEAD -> pa3) pa3.2 finished and run correctly
9da2c66 > run 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021
-09-29) i686 GNU/Linux 09:22:17 up 13 min, 2 users, load average: 0.00, 0.00, 0.00 4bcc0032
5fbbbedf29503a9c79d277e3444b2492c
10a3814 > run 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021
-09-29) i686 GNU/Linux 09:16:45 up 7 min, 2 users, load average: 0.15, 0.03, 0.01 d40586335
729807afb7d49c85f8a440d4cf028d1
0b745af > run 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021
-09-29) i686 GNU/Linux 21:56:05 up 54 min, 2 users, load average: 0.00, 0.00, 0.00 6f8a64c8
83f3b3d172a49e3be37e06742f7272d
7903180 > compile 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (
2021-09-29) i686 GNU/Linux 21:56:05 up 54 min, 2 users, load average: 0.00, 0.00, 0.00 259d
114a260a9ea6d97678caaae5bb47ebaa81c
22183eb > run 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021
-09-29) i686 GNU/Linux 21:45:35 up 44 min, 2 users, load average: 0.00, 0.03, 0.00 7909ec69
203alcfa28d2afd8e5e40ee98b390947
ee18c78 > compile 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (
2021-09-29) i686 GNU/Linux 21:45:35 up 44 min, 2 users, load average: 0.00, 0.03, 0.00 d8c9
5aaaba9114f16bf94050b9429a9ba00050ce
8fe9caa > run 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021
-09-29) i686 GNU/Linux 21:40:53 up 39 min, 2 users, load average: 0.00, 0.00, 0.00 3ca628ef
2916026d6a99a6f33dce746256489635
7bd3279 > run 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021
-09-29) i686 GNU/Linux 21:03:29 up 2 min, 2 users, load average: 0.46, 0.12, 0.04 5dc40e612
e745cef37580fcc2ca4df0fa53cc855
ce74363 > run 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021
-09-29) i686 GNU/Linux 21:02:52 up 1 min, 2 users, load average: 0.00, 0.00, 0.00 2f060e13e
2c0133d52ba9a81990a0541dd67e0fd
6995e54 > run 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021
-09-29) i686 GNU/Linux 20:57:49 up 54 min, 2 users, load average: 0.00, 0.14, 0.09 c15c68f3
1781f8e867cac78cf6407e60334caba0
0e85000 > compile 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (
2021-09-29) i686 GNU/Linux 20:57:49 up 54 min, 2 users, load average: 0.00, 0.14, 0.09 a8a7
8
```

远程仓库截图

发现之前忘了push pa2分支，所以先把push上去的pa3分支给删了，然后重新push pa2 pa3分支

项目动态

2022-06-07 星期二

- 10:36 19 shaozhenzhe 推送了新的分支 pa3 到代码仓库: ics2022
- 10:35 19 shaozhenzhe 推送了新的分支 pa2 到代码仓库: ics2022
- 10:34 19 shaozhenzhe 从代码仓库 ics2022 删除了分支: pa3 [40293f4]
- 10:25 19 shaozhenzhe 推送了分支 pa3 到代码仓库: ics2022
 - shaozhenzhe:[40293f4]pa3.2 finished and run correctly
 - tracer-ics2017:[9da2c66]> run
 - tracer-ics2017:[10a3814]> run

实验内容

1. 添加分页控制相关寄存器（10分）

首先在 nanos-lite/src/main.c 中定义宏 HAS_PTE

- CR0, CR3 寄存器的定义;

首先在 nemu/include/cpu/reg.h 引入头文件

```
#include "memory/mmu.h"
```

然后再寄存器结构里添CR0和CR3

```

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{
            rtlreg_t eax, ecx, edx, ebx, esp, ebp, esi, edi;
        };
    };
    vaddr_t eip;

    union{
        uint32_t val;
        struct{
            uint32_t CF:1;
            uint32_t :5; //无名位域
            uint32_t ZF:1;
            uint32_t SF:1;
            uint32_t :1;
            uint32_t IF:1;
            uint32_t :1;
            uint32_t OF:1;
        };
    }eflags;

    struct {
        uint32_t base; //32位base
        uint16_t limit; //16位limit
    }idtr;
    uint32_t cs;

    CR0 cr0;
    CR3 cr3;

} CPU_state;

```

- 初始化 CR0, CR3 寄存器;
在 `nemu/src/monitor/monitor.c` 的 `restart`

```

static inline void restart() {
    /* Set the initial instruction pointer. */
    cpu.eip = ENTRY_START;
    cpu.eflags.val=0x2;
    cpu.cs=0x8;

    cpu.cr0.val=0x60000011;

#ifdef DIFF_TEST
    init_qemu_reg();
#endif
}

```

2. 修改访存函数 (20 分)

- vaddr_read();

按照讲义给的框架，完成 if (data cross the page boundary) 的部分即可

要判断数据是否跨越页的边界，只需要判断地址加上数据长度是否比一页大即可。

一页的大小为4KB，即0x1000。addr对4096取余得到数据在页内的起始位置，起始位置与长度相加再与0x1000比较就可以知道是否跨页了。

addr对4096取余可以写成 `addr&0xFFF`

nemu/src/memory/memory.c

```

uint32_t vaddr_read(vaddr_t addr, int len) {
    if(cpu.cr0.paging) {
        if ((addr&0xFFF)+len>0x1000) {
            /* this is a special case, you can handle it later. */
            assert(0);
        }
        else {
            paddr_t paddr = page_translate(addr, false);
            return paddr_read(paddr, len);
        }
    }
    else
        return paddr_read(addr, len);
}

```

- vaddr_write();

和 `vaddr_read()` 一样的判断方法

```

void vaddr_write(vaddr_t addr, int len, uint32_t data) {
    if(cpu.cr0.paging) {
        if ((addr&0xFFF)+len>0x1000) {
            /* this is a special case, you can handle it later. */
            assert(0);
        }
        else {
            paddr_t paddr = page_translate(addr, true);
            return paddr_write(paddr, len, data);
        }
    }
}

```

```

    }
    else
        return paddr_write(addr, len, data);
}

```

3. page_translate() (30 分)

根据讲义给出的步骤写出来就行。要有Log()输出，`assert` 验证 `present` 位

如何编写 page_translate()

下面我们来看一下 `page_translate()` 的实现：

- 该函数用于地址转换，传入 **虚拟地址** 作为参数，函数返回值为 **物理地址**；
 - 该函数的实现过程即为我们理论课学到的页级转换过程（先找页目录项，然后取出）；
 - 注意使用 `assert` 来验证 `present` 位，否则会造成 **调试困难**；
 - `PDE` 和 `PTE` 的数据结构框架已帮我们定义好，在 `mmu.h` 中；
 - 注意每个页目录项和每个页表项存储在内存中的地址均为 **物理地址**，使用 `paddr_read` 去读取，如果使用 `vaddr_read` 去读取会造成死递归(为什么?)；
 - 此外，还需要实现访问位和脏位的功能；
 - 需要在 `page_translate` 中插入 `Log` 并截图表示实现成功（截图后可去除 `Log` 以免影响性能）；
 - 如何编写这个函数？
- 根据 `CR3` 寄存器得到页目录表基址(是个物理地址)；
 - 用这个基址和从虚拟地址中隐含的 页目录 字段项结合计算出所需页目录项地址(是个物理地址)；
 - 请思考一下这里所谓的“结合”需要经过哪些处理才能得到正确地地址呢？
 - 从内存中读出这个页目录项，并对有效位进行检验；
 - 将取出的 `PDE` 和虚拟地址的 页表 字段相组合，得到所需页表项地址(是个物理地址)；
 - 从内存中读出这个页表项，并对有效位进行检验；
 - 检验 `PDE` 的 `accessed` 位，如果为 `0` 则需变为 `1`，并写回到页目录项所在地址；
 - 检验 `PTE` 的 `accessed` 位如果为 `0`，或者 `PTE` 的脏位为 `0` 且现在正在做写内存操作，满足这两个条件之一时需要将 `accessed` 位，然后更新 `dirty` 位，最后并写回到页表项所在地址；
 - 页级地址转换结束，返回转换结果(是个物理地址)。

```

paddr_t page_translate(vaddr_t vaddr, bool iswrite);

paddr_t page_translate(vaddr_t vaddr, bool iswrite) {

    Log("addr:0x%x\n", vaddr);
    //cr3高20位
    vaddr_t CR3 = cpu.cr3.page_directory_base<<12;
    Log("CR3:0x%x\n", CR3);
    //vaddr高10位
    vaddr_t dir = (vaddr>>22)*4;
    Log("dir:0x%x\n", dir);
    //取出cr3的高20位与vaddr的高8位结合
    paddr_t pdAddr = CR3 + dir;
    Log("pdAddr:0x%x\n", pdAddr);
    //读取
    PDE pd_val;
    pd_val.val = paddr_read(pdAddr, 4);
    Log("pdAddr:0x%x pd_val:0x%x\n", pdAddr, pd_val.val);
    assert(pd_val.present);

    //获取高20位
    vaddr_t t1 = pd_val.page_frame<<12;

```

```

//获取第二个十位page
vaddr_t t2 = ((vaddr>>12)&0x3FF)*4;

paddr_t ptAddr = t1 + t2;
Log("pt_addr:0x%x\n",ptAddr);
PTE pt_val;
pt_val.val = paddr_read(ptAddr,4);
assert(pt_val.present);
Log("pt_addr:0x%x  pt_val:0x%x\n",ptAddr,pt_val.val);
//获取高20位
t1 = pt_val.page_frame<<12;

//获取最后12位
t2 = vaddr&0xFFF;

paddr_t paddr = t1 + t2;
Log("paddr:0x%x\n",paddr);

pd_val.accessed = 1;
paddr_write(pdAddr,4,pd_val.val);

if ((pt_val.accessed == 0) || (pt_val.dirty ==0 && iswrite)){
    pt_val.accessed=1;
    pt_val.dirty=1;
}
paddr_write(ptAddr,4,pt_val.val);

return paddr;
}


```

运行结果

```

shaozhenzhe@Debian: ~/ics2022/nanos-lite
[src/mm.c,40,init_mm] free physical pages starting from 0x5dc4000
invalid opcode(eip = 0x0010135d): 0f 22 d8 90 5d c3 55 89 ...

There are two cases which will trigger this unexpected exception:
1. The instruction at eip = 0x0010135d is not implemented.
2. Something is implemented incorrectly.
Find this eip(0x0010135d) 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!

(nemu)

```

发现有指令 0f 22 未实现

填表

```
/* 0x20 */    EMPTY, EMPTY, IDEX(mov_E2G,mov_r2cr), EMPTY,
```

nemu/src/cpu/exec/system.c

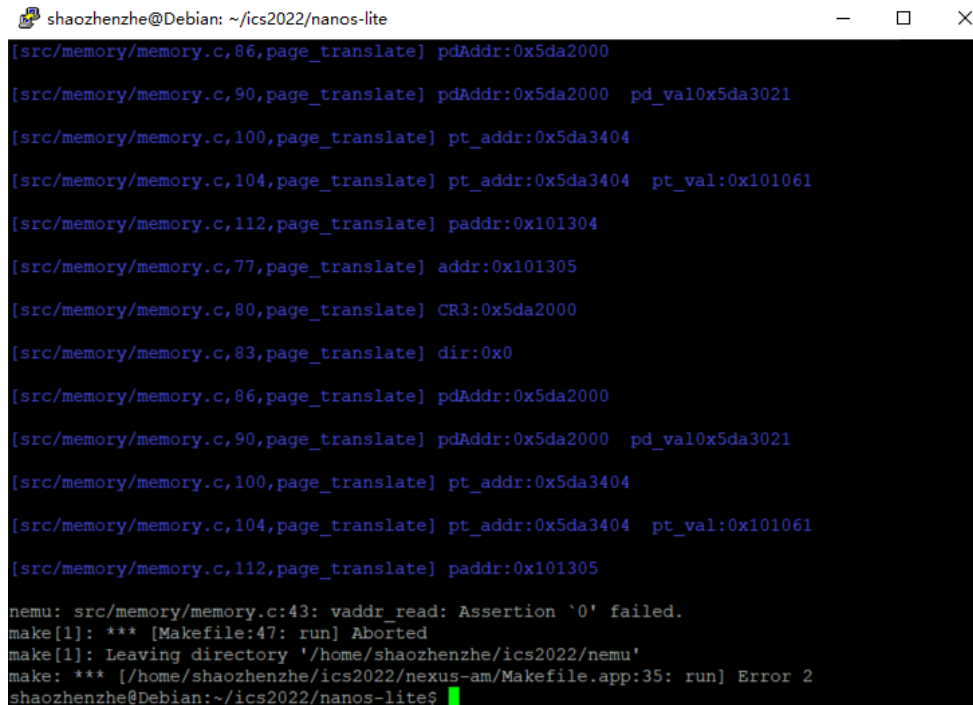

```

        break;
    default:
        assert(0);
        break;
}
print_asm("movl %%cr%d,%%s", id_src->reg, reg_name(id_dest->reg, 4));

#ifdef DIFF_TEST
    diff_test_skip_qemu();
#endif
}

```

运行结果



```

shaozhenzhe@Debian: ~/ics2022/nanos-lite
[src/memory/memory.c,86,page_translate] pdAddr:0x5da2000
[src/memory/memory.c,90,page_translate] pdAddr:0x5da2000 pd_val:0x5da3021
[src/memory/memory.c,100,page_translate] pt_addr:0x5da3404
[src/memory/memory.c,104,page_translate] pt_addr:0x5da3404 pt_val:0x101061
[src/memory/memory.c,112,page_translate] paddr:0x101304
[src/memory/memory.c,77,page_translate] addr:0x101305
[src/memory/memory.c,80,page_translate] CR3:0x5da2000
[src/memory/memory.c,83,page_translate] dir:0x0
[src/memory/memory.c,86,page_translate] pdAddr:0x5da2000
[src/memory/memory.c,90,page_translate] pdAddr:0x5da2000 pd_val:0x5da3021
[src/memory/memory.c,100,page_translate] pt_addr:0x5da3404
[src/memory/memory.c,104,page_translate] pt_addr:0x5da3404 pt_val:0x101061
[src/memory/memory.c,112,page_translate] paddr:0x101305
nemu: src/memory/memory.c:43: vaddr_read: Assertion `0' failed.
make[1]: *** [Makefile:47: run] Aborted
make[1]: Leaving directory '/home/shaozhenzhe/ics2022/nemu'
make: *** [/home/shaozhenzhe/ics2022/nexus-am/Makefile.app:35: run] Error 2
shaozhenzhe@Debian:~/ics2022/nanos-lite$

```

这里是出现了数据跨越虚拟页边界的情况，后面会修复。

4. 修改 loader() (30 分)

- 调用 `_map()` 进行虚拟页映射；

把 `navy-apps/Makefile.compile` 中的链接地址 `-Ttext` 参数改为 `0x8048000` 并重新编译

把 `nanos-lite/src/loader.c` 中的 `DEFAULT_ENTRY` 也需要作相应的修改

```
#define DEFAULT_ENTRY ((void *)0x8048000)
```

在 `nanos-lite/src/main.c` 中把原本加载程序的换成 `load_prog()`，同时要声明

```

extern void load_prog(const char *filename);

int main() {
#ifdef HAS_PTE
    init_mm();
#endif

    Log("Hello World! from Nanos-lite");
    Log("Build time: %s, %s", __TIME__, __DATE__);
}

```

```

init_ramdisk();

init_device();

#ifdef HAS_ASYE
    Log("Initializing interrupt/exception handler...");
    init_irq();
#endif

init_fs();

//uint32_t entry = loader(NULL, "/bin/pal");
//((void (*)(void))entry)();
load_prog("/bin/dummy");

panic("Should not reach here");
}

```

根据讲义给的步骤完成

1. 打开待装入的文件后，还需要获取文件大小；
2. 需要循环判断是否已创建足够的页来装入程序；
3. 对于程序需要的每一页，做三个事情，即4，5，6步：
4. 使用 Nanos-lite 的 MM 提供的 `new_page()` 函数获取一个空闲物理页
5. 使用映射函数 `_map()` 将本虚拟空间内当前正在处理的这个页和上一步申请到的空闲物理页建立映射
6. 读一页内容，写到这个物理页上
7. 每一页都处理完毕后，关闭文件，并返回程序入口点地址（虚拟地址）

nanos-lite/src/loader.c

```

#include "memory.h"
uintptr_t loader(_Protect *as, const char *filename) {
    //TODO();

    //读取文件位置
    int index=fs_open(filename,0,0);
    //读取长度
    size_t length=fs_filesz(index);
    //读取内容
    //fs_read(index,DEFAULT_ENTRY,length);

    void *va;
    void *pa;
    int page_count = length/4096 + 1;//获取页数量

    for (int i=0;i<page_count;i++){
        va = DEFAULT_ENTRY + 4096*i;
        pa = new_page();
        Log("Map va to pa: 0x%08x to 0x%08x",va,pa);
        _map(as,va,pa);
        fs_read(index,pa,4096);
    }
    //关闭文件
    fs_close(index);
    return (uintptr_t)DEFAULT_ENTRY;
}

```

```
}
```

- 需要按讲义格式输出映射页面情况作为实现依据;

运行后发现 `vaddr_read()` 的判断条件报错

```
shaozhenzhe@Debian: ~/ics2022/nanos-lite
[src/memory/memory.c,86,page_translate] pdAddr:0x5da2000
[src/memory/memory.c,90,page_translate] pdAddr:0x5da2000 pd_val0x5da3021
[src/memory/memory.c,100,page_translate] pt_addr:0x5da3404
[src/memory/memory.c,104,page_translate] pt_addr:0x5da3404 pt_val:0x101061
[src/memory/memory.c,112,page_translate] paddr:0x101448
[src/memory/memory.c,77,page_translate] addr:0x101449
[src/memory/memory.c,80,page_translate] CR3:0x5da2000
[src/memory/memory.c,83,page_translate] dir:0x0
[src/memory/memory.c,86,page_translate] pdAddr:0x5da2000
[src/memory/memory.c,90,page_translate] pdAddr:0x5da2000 pd_val0x5da3021
[src/memory/memory.c,100,page_translate] pt_addr:0x5da3404
[src/memory/memory.c,104,page_translate] pt_addr:0x5da3404 pt_val:0x101061
[src/memory/memory.c,112,page_translate] paddr:0x101449
nemu: src/memory/memory.c:43: vaddr_read: Assertion `0' failed.
make[1]: *** [Makefile:47: run] Aborted
make[1]: Leaving directory '/home/shaozhenzhe/ics2022/nemu'
make: *** [/home/shaozhenzhe/ics2022/nexus-am/Makefile.app:35: run] Error 2
shaozhenzhe@Debian:~/ics2022/nanos-lite$
```

推测是出现数据跨越虚拟页边界的情况

可以分别读出两页的内容，再按小端方式组合，就是把第二页的数据放在高位。

据此更新 `vaddr_read` 和 `vaddr_write` 函数

```
uint32_t vaddr_read(vaddr_t addr, int len) {
    if(cpu.cr0.paging) {
        if ((addr&0xFFF)+len>0x1000) {
            /* this is a special case, you can handle it later. */
            //assert(0);

            int len1,len2;
            len1 = 0x1000-(addr&0xfff); //获取前一页的占用空间
            len2 = len - len1; //获取后一页的占用空间

            paddr_t addr1 = page_translate(addr,false); //虚拟地址转换为物理地址
            uint32_t data1 = paddr_read(addr1,len1); //读取内容

            paddr_t addr2 = page_translate(addr+len1,false); //虚拟地址转换为物理地址
            uint32_t data2 = paddr_read(addr2,len2); //读取内容

            //len1<<3表示获取data1的位数
            uint32_t data = (data2<<(len1<<3))+data1; //把data2的数据移到高位，组合读取到的内容。
            return data;
        }
        else {
            paddr_t paddr = page_translate(addr,false);
            return paddr_read(paddr, len);
        }
    }
}
```

```

    }
    else
        return paddr_read(addr, len);
}

void vaddr_write(vaddr_t addr, int len, uint32_t data) {
    if(cpu.cr0.paging) {
        if ((addr&0xFFF)+len>0x1000) {
            /* this is a special case, you can handle it later. */
            //assert(0);

            int len1, len2;
            len1 = 0x1000-(addr&0xfff); //获取前一页的占用空间
            len2 = len - len1; //获取后一页的占用空间

            paddr_t addr1 = page_translate(addr, true); //虚拟地址转换为物理地址
            paddr_write(addr1, len1, data); //写入内容

            uint32_t data2 = data >> (len1<<3);
            paddr_t addr2 = page_translate(addr+len1, true);
            paddr_write(addr2, len2, data2);
        }
        else {
            paddr_t paddr = page_translate(addr, true);
            return paddr_write(paddr, len, data);
        }
    }
    else
        return paddr_write(addr, len, data);
}

```

运行结果

```

shaozhenzhe@Debian: ~/ics2022/nanos-lite
make[1]: Leaving directory '/home/shaozhenzhe/ics2022/nexus-am'
make[1]: Entering directory '/home/shaozhenzhe/ics2022/nexus-am/libs/klib'
make[1]: *** No targets specified and no makefile found. Stop.
make[1]: Leaving directory '/home/shaozhenzhe/ics2022/nexus-am/libs/klib'
make: [/home/shaozhenzhe/ics2022/nexus-am/Makefile.compile:86: klib] Error 2 (ignored)
make[1]: Entering directory '/home/shaozhenzhe/ics2022/nemu'
+ CC src/memory/memory.c
+ LD build/nemu
./build/nemu -l /home/shaozhenzhe/ics2022/nanos-lite/build/nemu-log.txt /home/shaozhenzhe/ics2022/nanos-lite/build/nanos-lite-x86-nemu.bin
[src/monitor/monitor.c,65,load_img] The image is /home/shaozhenzhe/ics2022/nanos-lite/build/nanos-lite-x86-nemu.bin
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 20:57:49, Jun  6 2022
For help, type "help"
(nemu) c
[src/mm.c,40,init_mm] free physical pages starting from 0x5dc4000
[src/main.c,20,main] 'Hello World!' from Nanos-lite
[src/main.c,21,main] Build time: 21:40:53, Jun  6 2022
[src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x102b00, end = 0x5d7fbc5, size = 96981189 bytes
[src/main.c,28,main] Initializing interrupt/exception handler...
[src/loader.c,25,loader] Map va to pa: 0x08048000 to 0x05dc5000
[src/loader.c,25,loader] Map va to pa: 0x08049000 to 0x05dc7000
[src/loader.c,25,loader] Map va to pa: 0x0804a000 to 0x05dc8000
[src/loader.c,25,loader] Map va to pa: 0x0804b000 to 0x05dc9000
[src/loader.c,25,loader] Map va to pa: 0x0804c000 to 0x05dca000
[src/loader.c,25,loader] Map va to pa: 0x0804d000 to 0x05dcb000
nemu: HIT GOOD TRAP at eip = 0x001000f1
(nemu) █

```

5. 在分页上运行仙剑奇侠传（10 分）

`mm_brk()` 框架已经给出，只需要调用即可。

`nanos-lite/src/syscall.c`

`sys_brk()` 函数

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

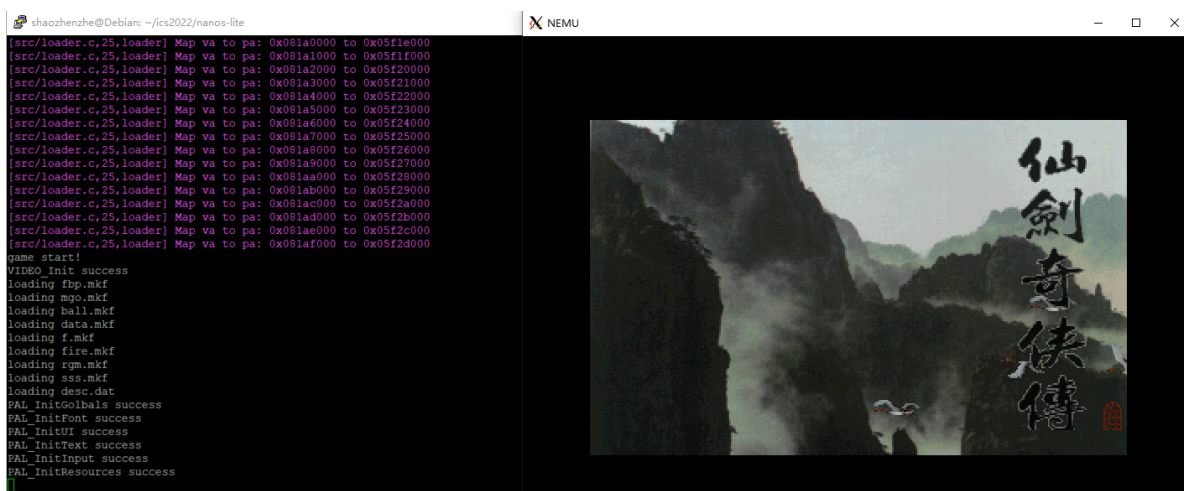
`SYS_brk` 系统调用

```
case SYS_brk:  
    SYSCALL_ARG1(r)=(int)sys_brk(a[1]);  
    break;
```

把 `nanos-lite/src/main.c` 改为

```
load_prog("/bin/pal");
```

运行结果



遇到的问题及解决办法

1.遇到问题：完成`page_translate()`的实现后，尝试运行，出现了`vaddr_read()`的判断条件报错

解决办法：发现讲义上有对这种情况的说明：

最后提醒一下页级地址转换时出现的一种特殊情况. 由于 i386 并没有严格要求数据对齐, 因此可能会出现数据跨越虚拟页边界的情况, 例如一条很长的指令的首字节在一个虚拟页的最后, 剩下的字节在另一个虚拟页的开头. 如果这两个虚拟页被映射到两个不连续的物理页, 就需要进行两次页级地址转换, 分别读出这两个物理页中需要的字节, 然后拼接起来组成一个完整的数据返回. 具体实现的时候可以根据读/写的长度和本页剩余的长度来判断是否跨页. 要注意的是: 如果是读跨页, 则需组合两次读出的数据(位操作实现); 如果是写跨页, 应注意两个页分别需要写的数据长度是多少, 以及写到两页上的内容分别是什么.

实现对应的操作即可。

2.遇到问题：运行仙剑奇侠传时特别慢，比之前的要慢很多。

解决办法：把Log()输出给关了会快一点。

实验心得

本次实验主要了解了分页的机制，实现了在分页上运行仙剑奇侠传，相比之前的实验这次的内容不多，很快就能完成。

其他备注

无