

# 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：PA3.2
- 完成日期：2021.6.30
- 本次实验，我完成了4个任务。

## 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. 一些问题
2. 空指针真的是'空'的吗?
3. 理解 \_map 函数
4. 内核映射的作用
5. git log 和 远程仓库截图

#### 操作题

##### PA3.2 虚拟地址空间

- 任务1：添加分页控制相关寄存器（10分）
- 任务2：修改访存函数
- 任务3：page\_translate()
- 任务4：修改loader（）
- 任务5：在分页上运行仙剑奇侠传

#### 遇到的问题及解决方法

#### 实验心得

#### 其他备注

## 思考题和git

---

### 1. 一些问题

除了20位基地址，还有低地址的12位页内偏移量

是必须的。

一级页表要保存所有空间的页信息，耗费的空间内存过大，不利于上下文切换。

### 2. 空指针真的是'空'的吗?

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

这是map函数的代码

```
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;
    }
}
```

map函数用来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

`pd_val.present` 报错。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

```
tangxi@debian: ~/ics2021
commit 4bae94cc30c496f2ab888d73baabcf2bd1465ea (HEAD -> pa3)
Author: tracer-ics2017 <tracer@njuics.org>
Date: Thu Jul 1 01:11:08 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
    01:11:08 up 2:26, 1 user, load average: 0.18, 0.08, 0.03
    7ba0636d7d0c4914ca9ea830f8d4ddfd3220a72d

commit 34544ade9b93c5e64cda04b0b2a6e8b3029db7d1
Author: tracer-ics2017 <tracer@njuics.org>
Date: Thu Jul 1 01:11:08 2021 +0800

    > compile
    161920124
    tangxi
    Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux
    01:11:08 up 2:26, 1 user, load average: 0.18, 0.08, 0.03
    9846078cb97ac5f14a66d498b2dd24f326ba3dle

commit 6613ffe5e0f39199d74227106f9e94e69f0dc884
Author: tracer-ics2017 <tracer@njuics.org>
Date: Thu Jul 1 01:10:36 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
    01:10:36 up 2:25, 1 user, load average: 0.20, 0.07, 0.03
    c7f50bde99090d513a11f6d899c6f40e258b5eb

commit 61a1c2c2b74d0ebbdaffef6a8b8b23a2588c5499
Author: tracer-ics2017 <tracer@njuics.org>
Date: Thu Jul 1 01:10:15 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
    01:10:15 up 2:25, 1 user, load average: 0.09, 0.05, 0.02
    f43a81c5f4ae4d6c4b9106a14aca7ce6f567906e

commit c351763275bcbcl1dc41f61c9eff086df5afb84c
Author: tracer-ics2017 <tracer@njuics.org>
Date: Thu Jul 1 01:10:15 2021 +0800

    > compile
    161920124
    tangxi
:█
```

## 操作题

### PA3.2 虚拟地址空间

#### 任务1：添加分页控制相关寄存器（10分）

CR0, CR3 寄存器的定义

在定义了寄存器结构体的reg.h里声明头文件

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

在CPU\_state 中添加两个寄存器CR0 cr0; CR3 cr3;

初始化 CR0, CR3 寄存器

在 `monitor.c` 中的 `restart()` 中初始化

```
cpu.cr0.val=0x60000011;
```

## 任务2：修改访存函数

根据讲义来修改即可

- `vaddr_read()`

```
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()`

```
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);
}
```

## 任务3：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) {

    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;
}

```

发现有指令没有实现

```
mov_r2cr() mov_cr2r()
```

填表

```
/* 0x20 */ IDEX(mov_G2E,mov_cr2r), EMPTY, IDEX(mov_E2G,mov_r2cr), EMPTY,
```

```

make_EHelper(mov_r2cr) { //给cr寄存器赋值
    switch (id_dest->reg)
    {
        case 0:
            cpu.cr0.val = id_src->val;
            break;
        case 3:
            cpu.cr3.val = id_src->val;
            break;
        default:
            assert(0);
            break;
    }

    print_asm("movl %%s,%%cr%d", reg_name(id_src->reg, 4), id_dest->reg);
}

make_EHelper(mov_cr2r) { //给寄存器赋值
    switch (id_dest->reg)
    {
        case 0:
            id_src->val = cpu.cr0.val;
            break;
        case 3:
            id_src->val = cpu.cr3.val;
            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
}
```

```
tangxi@debian: ~/ics2021/nemu
+ CC src/cpu/exec/system.c
+ CC src/cpu/exec/special.c
+ CC src/cpu/exec/cc.c
+ CC src/cpu/exec/arith.c
+ CC src/cpu/exec/control.c
+ CC src/cpu/exec/exec.c
+ CC src/cpu/exec/prefix.c
+ CC src/cpu/exec/data-mov.c
+ CC src/monitor/cpu-exec.c
+ CC src/monitor/debug/expr.c
+ CC src/monitor/debug/ui.c
+ CC src/monitor/diff-test/diff-test.c
+ CC src/monitor/monitor.c
+ LD build/nemu
./build/nemu -l ./build/nemu-log.txt
[src/monitor/monitor.c,47,load_default_img] No image is given. Use the default build-in image
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 23:17:42, Jun 30 2021
For help, type "help"
(nemu) c
nemu: HIT GOOD TRAP at eip = 0x00100026
(nemu) █
```

## 任务4: 修改loader ()

接下来我们可以对 `loader()` 函数做如下修改:

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)

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` 中进行如下操作, 别忘了声明函数 `load_prog()`

```
- uintptr_t entry = loader(NULL, "/bin/pal");- ((void (*)(void))entry)();+
load_prog("/bin/dummy");
```

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

    //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;
}

```

运行后发现 `vaddr_read()` 报错,是因为数据跨越虚拟页边界, 使用 `page_translate()` 将虚拟地址转化为物理地址再去读取。

```

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

```



```

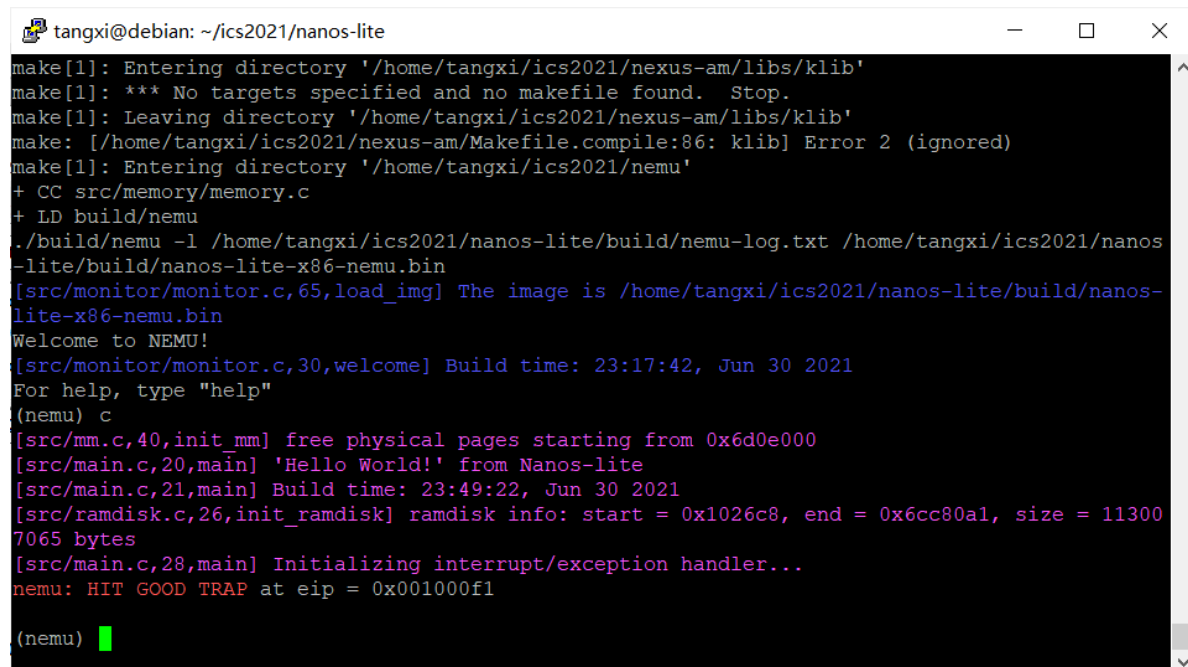
len2 = len - len1; //获取最后一页的占用空间

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

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);
}

```

成功运行



```

tangxi@debian: ~/ics2021/nanos-lite
make[1]: Entering directory '/home/tangxi/ics2021/nexus-am/libs/klib'
make[1]: *** No targets specified and no makefile found. Stop.
make[1]: Leaving directory '/home/tangxi/ics2021/nexus-am/libs/klib'
make: [/home/tangxi/ics2021/nexus-am/Makefile.compile:86: klib] Error 2 (ignored)
make[1]: Entering directory '/home/tangxi/ics2021/nemu'
+ CC src/memory/memory.c
+ LD build/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: 23:17:42, Jun 30 2021
For help, type "help"
(nemu) c
[src/mm.c,40,init_mm] free physical pages starting from 0x6d0e000
[src/main.c,20,main] 'Hello World!' from Nanos-lite
[src/main.c,21,main] Build time: 23:49:22, Jun 30 2021
[src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x1026c8, end = 0x6cc80a1, size = 113007065 bytes
[src/main.c,28,main] Initializing interrupt/exception handler...
nemu: HIT GOOD TRAP at eip = 0x001000f1
(nemu) █

```

## 任务5：在分页上运行仙剑奇侠传

```
tangxi@debian: ~/ics2021/nanos-lite
[src/loader.c,26,loader] Map va to pa: 0x081aa000 to 0x06e72000
[src/loader.c,26,loader] Map va to pa: 0x081ab000 to 0x06e73000
[src/loader.c,26,loader] Map va to pa: 0x081ac000 to 0x06e74000
[src/loader.c,26,loader] Map va to pa: 0x081ad000 to 0x06e75000
[src/loader.c,26,loader] Map va to pa: 0x081ae000 to 0x06e76000
[src/loader.c,26,loader] Map va to pa: 0x081af000 to 0x06e77000
game start!
VIDEO_Init success
loading fbp.mkf
loading mgo.mkf
loading ball.mkf
loading data.mkf
loading f.mkf
loading fire.mkf
loading rgm.mkf
loading sss.mkf
loading desc.dat
physical address(0xa2b24942) is out of bound
nemu: src/memory/memory.c:21: paddr_read: Assertion `addr < (1024 * 1024 * 1024)' failed.
make[1]: *** [Makefile:47: run] Aborted
make[1]: Leaving directory '/home/tangxi/ics2021/nemu'
make: *** [/home/tangxi/ics2021/nexus-am/Makefile.app:35: run] Error 2
tangxi@debian:~/ics2021/nanos-lite$
```

## 遇到的问题及解决方法

1.不知道那里的内容没做好，导致分页运行时一直出现out of bound，尝试了很多方法都不能解决

## 实验心得

讲义实在是太简洁了，很多东西都要自己摸索，十分困难

## 其他备注

无