

南京航空航天大学《计算机组成原理II课程设计》报告

- 姓名：郑伟林
- 班级：1619303
- 学号：061920125
- 报告阶段：PA3.2
- 完成日期：2021.6.17
- 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)

- ①还有12位是一些控制、标志信息，二十位是页框号，而不是具体的地址单元。
- ②必须是物理地址，不能用虚拟地址
- ③采用一级页表会使页表过大，不利于存储与索引

2. 空指针真的是'空'的吗? (15)

C语言空指针的值为null，一般null指针指向进程的最小的地址，通常这个值为0。如果程序对空指针解引用，则会访问进程地址的最小值，从而触发系统的错误，在Linux上是段错误。

3. 理解_map 函数 (25)

先通过p->ptr取出页目录基地址，然后取出页表项地址，如果需要申请新页面，则调用palloc_f()获取一页空闲的物理页，最后将其存放的具体物理地址取出。

4. 内核映射的作用 (25)

会触发Assert，去掉该部分会使虚拟空间无法正常初始化，从而引发冲突触发Assert

5. git log 和 远程仓库截图 (10)

```
zhengweilin@debian: ~/ics2021/nanos-lite
commit 41b5a8565056ba28fd84d7cf2bf885d0c998ab6f (HEAD -> pa3, myrepo/pa3)
Author: tracer-ics2017 <tracer@njuics.org>
Date: Thu Jun 17 17:03:30 2021 +0800

> gdb
061920125
zhengweilin
Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux
17:03:30 up 6:55, 2 users, load average: 0.10, 0.04, 0.01
e09c23068c5914053af4e594dd88a528d2baa05

commit e4a8d445d75c9bbf93e3cff8e5daea220cfcc519
Author: tracer-ics2017 <tracer@njuics.org>
Date: Thu Jun 17 17:03:04 2021 +0800

> gdb
061920125
zhengweilin
Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux
17:03:04 up 6:55, 2 users, load average: 0.16, 0.04, 0.01
233915cec50e92b2a557e2f8353302214040dcd

commit 545885d0b745c37fbb230d2b9a5b51787f3a2e13
Author: 061920125-Zheng Weilin <2529039819@qq.com>
Date: Thu Jun 17 17:02:46 2021 +0800

change sys_brk return

commit cb87ec3f4b789518cd42a6f9da6d181f652291d1
Author: 061920125-Zheng Weilin <2529039819@qq.com>
Date: Thu Jun 17 17:01:28 2021 +0800

change sys_brk return mm_brk

commit cec27abddbbb2bd3b0672814f36be55e6c96b4bb
Author: tracer-ics2017 <tracer@njuics.org>
Date: Thu Jun 17 16:57:58 2021 +0800

> gdb
061920125
zhengweilin
```

 开源软件 企业版 高校版 博客 我的 8周年

搜索

pa3 显示详细时间 全部贡献者 开始日期 结束日期 搜索

2021-06-17 (20)

 > gdb
tracer-ics2017 编写, 并由 吃花椒的热心 提交于 2021-06-17 17:03

41b5a85 浏览文件

 > gdb
tracer-ics2017 编写, 并由 吃花椒的热心 提交于 2021-06-17 17:03

e4a8d44 浏览文件

 change sys_brk return
吃花椒的热心 提交于 2021-06-17 17:02

545885d 浏览文件

 change sys_brk return mm_brk
吃花椒的热心 提交于 2021-06-17 17:01

cb87ec3 浏览文件

 > gdb
tracer-ics2017 编写, 并由 吃花椒的热心 提交于 2021-06-17 16:57

cec27ab 浏览文件

 change loader to load_prog
吃花椒的热心 提交于 2021-06-17 16:55

dc7670a 浏览文件

 change loader

9d4915d 浏览文件

实验内容

1. 添加分页控制相关寄存器（10分）

由于已给出CR0和CR3的结构体，因此只需在 `CPU_state` 中加入下列两个寄存器即可。

```
CR0 cr0;
CR3 cr3;
```

在 `monitor.c` 的 `restart()` 函数中将 `cr0` 的值赋为 `0x60000011`。

```
static inline void restart() {
    .....
    cpu.cr0.val = 0x60000011;
    .....
}
```

2. 修改访存函数（20 分）

`vaddr_read()`

首先判断 `cr0` 是否处于分页机制状态：

1. 是的话进行分页的虚拟内存读取。先判断当前数据是否会超过本页的边界，如果会：将 `addr` 分为两段，`prev_addr` 和 `last_addr`，分别进行读取，然后返回两者合并后的结果；如果不会：直接调用 `page_translate()` 转换为物理地址进行读取。

2. 否的话直接对 `addr` 进行读取。

```
uint32_t vaddr_read(vaddr_t addr, int len)
{
    if (cpu.cr0.paging)
    {
        if ((addr & 0xfff) + len > PGSIZE)
        {
            /* this is a special case, you can handle it later. */
            int prev = PGSIZE - OFF(addr);
            int last = len - prev;
            uint32_t prev_addr = paddr_read(page_translate(addr), prev);
            uint32_t last_addr = paddr_read(page_translate(addr + prev), last);
            return (last_addr << (8 * prev)) | prev_addr;
            //assert(0);
        }
        else
        {
            paddr_t paddr = page_translate(addr);
            return paddr_read(paddr, len);
        }
    }
    else
        return paddr_read(addr, len);
    //return paddr_read(addr, len);
}
```

`vaddr_write()`

首先判断 `cr0` 是否处于分页机制状态：

1.是的话进行分页的虚拟内存读取。先判断当前数据是否会超过本页的边界，如果会：assert(0)处理，暂时无需考虑；如果不会：直接调用 page_translate() 转换为物理地址进行写入。

2.否的话直接对 addr 进行写入。

```
void vaddr_write(vaddr_t addr, int len, uint32_t data)
{
    if (cpu.cr0.paging)
    {
        if ((addr & 0xfff) + len > PGSIZE)
        {
            assert(0);
        }
        else
        {
            paddr_t paddr = page_translate(addr);
            paddr_write(paddr, len, data);
        }
    }
    else
        paddr_write(addr, len, data);
}
```

3.page_translate() (30 分)

根据讲义提示做即可。

当cr0处于保护模式并且开启分页机制时转换地址。根据 CR3 寄存器得到页目录表基址；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 page_dir.val；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 page_table.val；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。

如果不是以上情况直接返回 vaddr

```
paddr_t page_translate(vaddr_t vaddr)
{
    if (cpu.cr0.protect_enable && cpu.cr0.paging)
    {
        paddr_t pde_base = cpu.cr3.page_directory_base;
        PDE page_dir;
        page_dir.val = paddr_read((pde_base << 12) + ((vaddr >> 22) << 2), 4);

        if (!page_dir.present)
        {
            printf("Invalid page_dir:0x%x\n", page_dir.val);
            assert(0);
        }
        PTE page_table;
        page_table.val = paddr_read((page_dir.page_frame << 12) + ((vaddr >> 10) & 0xffc), 4);

        if (!page_table.present)
```

```

{
    printf("Invalid page_table:0x%x\n", page_table.val);
    assert(0);
}

if (page_dir.accessed == 0)
{
    page_dir.accessed = 1;
    paddr_write(page_dir.page_frame, 4, vaddr & 0xfffff000);
}
if (page_table.accessed == 0 || (page_table.dirty == 0 &&
page_table.read_write == 1))
{
    page_table.accessed = 1;
    page_table.dirty = 1;
    paddr_write(page_table.page_frame, 4, vaddr & 0xfffff000);
}
Log("vaddr: 0x%x => paddr: 0x%x\n", vaddr, (page_table.page_frame << 12) +
(vaddr & 0xfff));
return (page_table.page_frame << 12) + (vaddr & 0xfff);
}
else
{
    Log("paddr: 0x%x\n", vaddr);
    return vaddr;
}
}
}

```

完成上述函数后发现还需完成两个指令，分别是 `mov_r2cr` 和 `mov_cr2r`。如下，只需根据操作符情况完成赋值即可。然后填表就完成了。

```

make_EHelper(mov_r2cr) {
    if (id_dest->reg == R_EAX)
        cpu.cr0.val = id_src->val;
    else if (id_dest->reg == R_EBX)
        cpu.cr3.val = id_src->val;
    else
        assert(0);

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

make_EHelper(mov_cr2r) {
    if (id_src->reg == R_EAX)
        operand_write(id_dest, &cpu.cr0.val);
    else if (id_src->reg == R_EBX)
        operand_write(id_dest, &cpu.cr3.val);
    else
        assert(0);

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

#ifdef DIFF_TEST
    diff_test_skip_qemu();
#endif
}

```

```
zhengweilin@debian: ~/ics2021/nanos-lite

[src/memory/memory.c,105,page_translate] vaddr: 0x101734 => paddr: 0x101734
[src/memory/memory.c,105,page_translate] vaddr: 0x101738 => paddr: 0x101738
[src/memory/memory.c,105,page_translate] vaddr: 0x101739 => paddr: 0x101739
[src/memory/memory.c,105,page_translate] vaddr: 0x101713 => paddr: 0x101713
[src/memory/memory.c,105,page_translate] vaddr: 0x101714 => paddr: 0x101714
[src/memory/memory.c,105,page_translate] vaddr: 0x101715 => paddr: 0x101715
[src/memory/memory.c,105,page_translate] vaddr: 0x7b6c => paddr: 0x7b6c
[src/memory/memory.c,105,page_translate] vaddr: 0x101725 => paddr: 0x101725
[src/memory/memory.c,105,page_translate] vaddr: 0x101726 => paddr: 0x101726
[src/memory/memory.c,105,page_translate] vaddr: 0x101727 => paddr: 0x101727
[src/memory/memory.c,105,page_translate] vaddr: 0x101728 => paddr: 0x101728
[src/memory/memory.c,105,page_translate] vaddr: 0x5da3030 => paddr: 0x5da3030
```

```
NEMU

zhengweilin@debian: ~/ics2021/nanos-lite
[src/main.c,19,main] 'Hello World!' from Nan
[src/main.c,20,main] Build time: 15:06:21, J
[src/ramdisk.c,26,init_ramdisk] ramdisk info
size = 96981109 bytes
[src/main.c,27,main] Initializing interrupt/
open file:/bin/pal
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
PAL_InitGolbals success
PAL_InitFont success
PAL_InitUI success
PAL_InitText success
PAL_InitInput success
PAL_InitResources success
```

4.修改 loader() (30 分)

根据讲义步骤

先获取默认入口 `va`，和页数（用文件长度/页大小 +1），再循环每一页进行映射，方法是先用 `new_page()` 获取一个空闲物理页给 `pa`，然后调用 `_map(as, va, pa)`，然后进行文件读取，最后 `va+PGSIZE`。

```
uintptr_t loader(_Protect *as, const char *filename) {
```

```

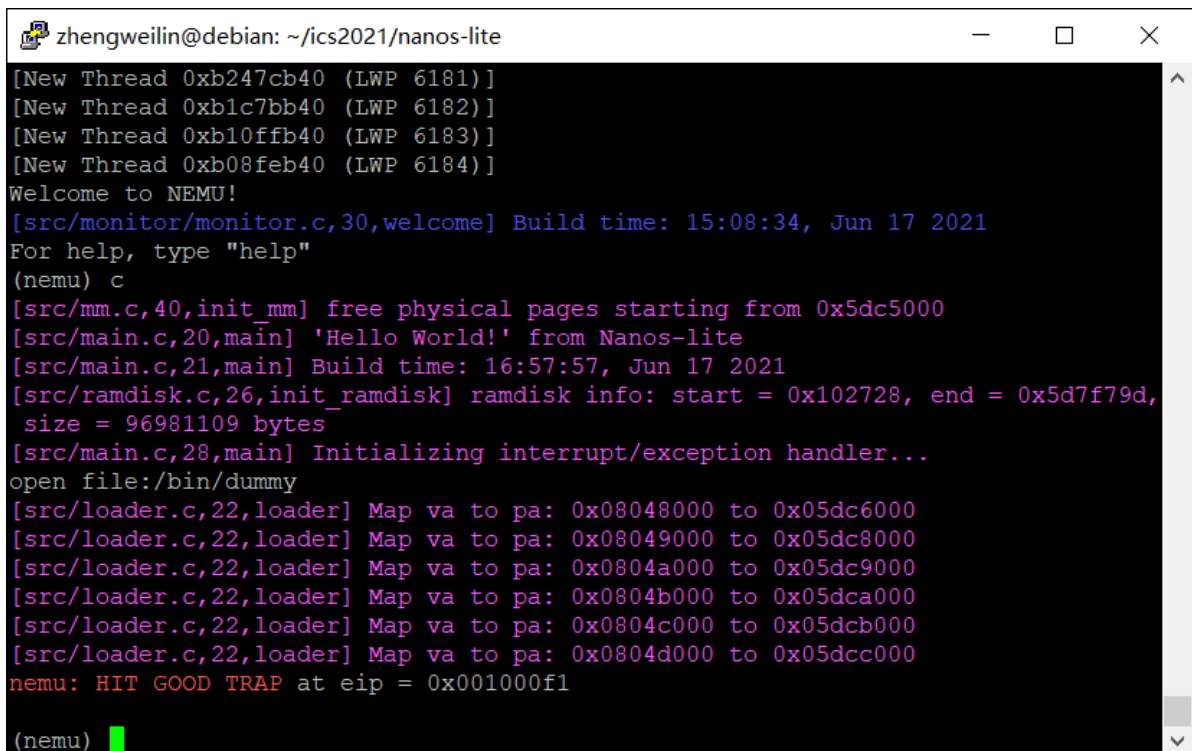
//ramdisk_read(DEFAULT_ENTRY, 0, get_ramdisk_size());
int fd = fs_open(filename,0,0);
size_t len = fs_filesz(fd);
void *va = DEFAULT_ENTRY;
int page_num = len / PGSIZE + 1;

for (int i = 0; i < page_num; i++)
{
    void *pa = new_page();
    Log("Map va to pa: 0x%08x to 0x%08x", va, pa);
    _map(as,va,pa);
    fs_read(fd,pa,PGSIZE);
    va += PGSIZE;
}

fs_close(fd);

return (uintptr_t)DEFAULT_ENTRY;
}

```



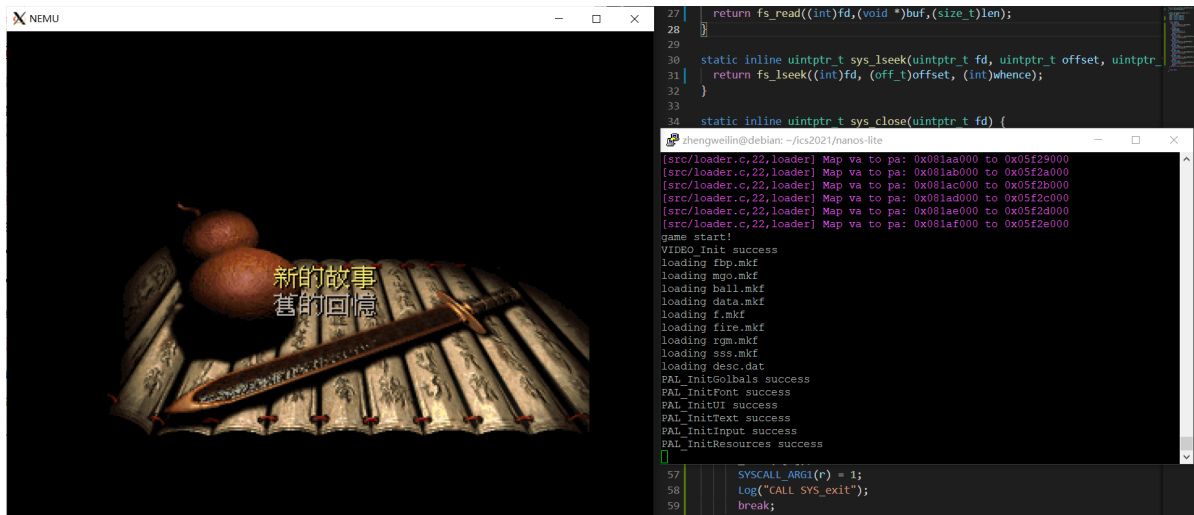
```

zhengweilin@debian: ~/ics2021/nanos-lite
[New Thread 0xb247cb40 (LWP 6181)]
[New Thread 0xb1c7bb40 (LWP 6182)]
[New Thread 0xb10ffb40 (LWP 6183)]
[New Thread 0xb08feb40 (LWP 6184)]
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 15:08:34, Jun 17 2021
For help, type "help"
(nemu) c
[src/mm.c,40,init_mm] free physical pages starting from 0x5dc5000
[src/main.c,20,main] 'Hello World!' from Nanos-lite
[src/main.c,21,main] Build time: 16:57:57, Jun 17 2021
[src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x102728, end = 0x5d7f79d,
size = 96981109 bytes
[src/main.c,28,main] Initializing interrupt/exception handler...
open file:/bin/dummy
[src/loader.c,22,loader] Map va to pa: 0x08048000 to 0x05dc6000
[src/loader.c,22,loader] Map va to pa: 0x08049000 to 0x05dc8000
[src/loader.c,22,loader] Map va to pa: 0x0804a000 to 0x05dc9000
[src/loader.c,22,loader] Map va to pa: 0x0804b000 to 0x05dca000
[src/loader.c,22,loader] Map va to pa: 0x0804c000 to 0x05dcb000
[src/loader.c,22,loader] Map va to pa: 0x0804d000 to 0x05dcc000
nemu: HIT GOOD TRAP at eip = 0x001000f1
(nemu) █

```

5. 在分页上运行仙剑奇侠传 (10 分)

在 `sys_brk()` 函数中, 将返回值修改为调用 `mm_brk()`。



遇到的问题及解决办法

1. 遇到问题：分页机制不能正常执行，会触发assert

解决方案：发现是跨页功能实现不正确导致，重新思考改了一下就可以了。

2. 遇到问题：...

解决方案：...

实验心得

本次实验运用了分页、虚拟地址的知识，通过代码实现了nanos的分页功能，并在分页机制下成功运行了仙剑奇侠传。通过此次实验，我对虚拟存储器的有了进一步的理解，明白操作系统如何分页，分页后如何从虚拟地址找到物理地址读取数据。计算机巧妙的运用虚拟存储技术来使程序将一部分代码和数据传入主存，其它暂时放在硬盘，解决了物理内存大小的限制。

其他备注

无