PA3.2 虚拟地址空间

王山岳

回顾现在的NEMU/NanOS

- 未解之谜：NanOS？NEMU？
- 现在的 NEMU/NanOS 能做什么？
- 现在的 NEMU/NanOS 还差点什么？

提纲

- 虚拟地址空间
- 上下文切换
- 中断

分时多任务

- PA3的Nanos-lite是单任务OS
- 如何运行多任务？

- 分时多任务
  - 进程有独立的存储空间
  - 进程上下文的切换

独立的存储空间

- OS管理所有资源 -> OS来分配内存
- 不同的程序加载到不同的内存上即可

- 把用户程序事先链接到不同的位置行不行？
- 问题：链接的时候如何保证某位置在将来加载的时候是空闲的？
- 无法保证

- 无法在编译时刻假设程序将来加载的位置

位置无关代码（PIC）

- 程序的一种属性
  - 代码不对将来的运行位置进行任何假设
  - 可以被加载到任意位置执行
- 编译器可以编译出PIC
- 但需要loader在加载时刻正确填写程序中的GOT数据结构
- Nanos-lite目前使用raw program loader
  - 无法得知GOT在可执行文件中的哪个位置
- 此方案不可行

CALL 的例子

```
make_EHelper(call) {
  // the target address is calculated at the decode stage
  decoding.is_jmp=1;
  rtl_push(eip);
  rtl_add(&decoding.jmp_eip,eip,&id_dest->val);
  print_asm("call %x", decoding.jmp_eip);
}      161630229-Wang Shanyue, 2 years ago • ics2017 initialized
```

分段（不需实现）

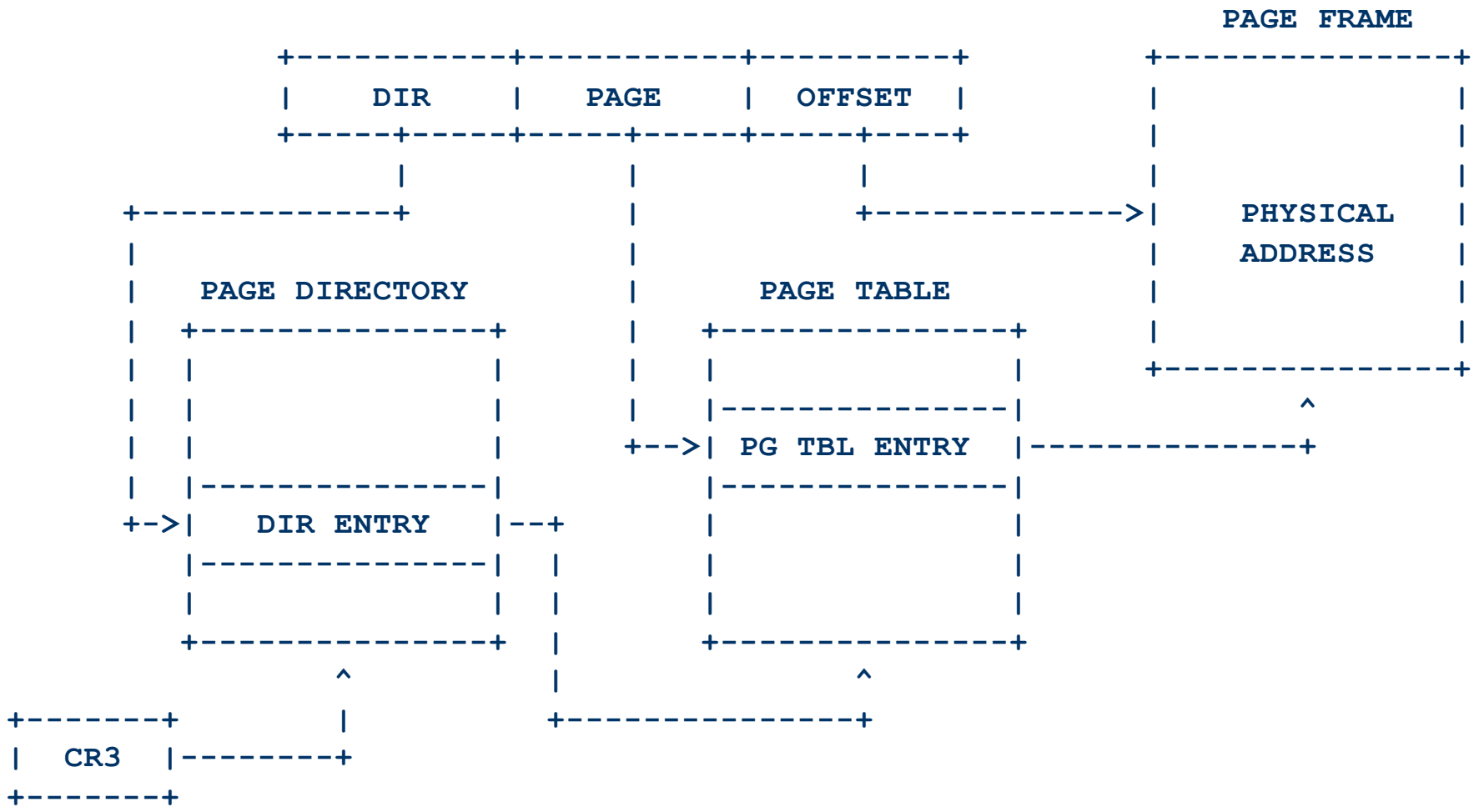
- 每个程序在运行前被赋予不同的段描述符
- 通过段偏移量不同让各个程序在不同的地址运行
- 扁平模式下，令段基址均为 0，大小均为 4GB，即消除了分段的概念
- 既然不用，为什么不能直接去除分段机制？

分页

- 需要一种按需分配的虚存管理机制
  - 程序代码很大，但一次运行的时候只会用到很少的部分，无需全部加载
  - 小程序结束时容易留下碎片空洞，只有比它更小的程序才能把碎片空洞利用起来
- 分段不能满足需求：粒度太大

- 反其道而行之：把连续的存储空间分割成小片段，以这些小片段为单位进行组织，分配和管理

I386分页机制


PAGE FRAME
DIR
PAGE
OFFSET
PHYSICAL ADDRESS
PAGE DIRECTORY
PAGE TABLE
DIR ENTRY
PG TBL ENTRY
CR3

虚拟地址空间

- 刚刚所说的访问相同虚拟地址的实质
- 每个进程拥有一个属于自己的虚拟地址空间，有一个上界、一个下界和一个页目录基址三个属性
- 虚拟地址不是针对全局而言的，而是对于当前虚拟空间而言的
- 在 Nanos-lite 中为 _Protect 结构体

虚存管理

- 把程序看到的地址和物理上访问内存的地址分开
  - 前者为虚拟地址，后者为物理地址
- 维护好虚拟地址到物理地址的映射
- 谁来进行映射？
  - OS无法干涉指令执行的具体过程，需要添加新的硬件MMU来进行映射
- 但只有OS才能知道哪些物理内存是空闲的
- 所以，虚存管理是一个软硬件协同的机制
  - 程序运行前，OS决定好把虚拟地址映射到哪些物理地址，并配置MMU
  - 程序运行时，MMU根据OS的配置进行地址转换

MMU

- Memory Management Unit，内存管理单元
- 属于硬件，用于虚拟地址到物理地址的转换
- Nanos-lite 项目中的 mm.c
- NEMU 中的页级转换函数

问题：两个进程访问相同虚拟地址

- 两个进程同时访问 0x08048000，那么同样地经过两层页表转换，得到的页表项不是同一个吗？
- 不是同一个。为什么？
- 因为每个进程拥有其自己的内核页目录和页表
- 加载程序时，即申请页表，建立本进程的页表到物理页的映射关系

PCB：进程描述符（简单讲解）

- 想听复杂讲解，之后PA4会有
- 每个进程在 OS 中以一个个 PCB 的形式存在

一些思考

- 空指针是空的吗？
- 处理器有“表”的概念吗？
- 一个页面大小为1KB的一级页表的地址转换例子
  - pa = (pg_table[va >> 10] & ~0x3ff) | (va & 0x3ff);
- 地址转换的过程只不过是一些访存和位操作而已
- 再次展示了计算机的本质
  - 一堆美妙的, 蕴含着深刻数学道理和工程原理的... 门电路!

代码实现

添加PTE(硬件)

- 在哪里添加基于分页的 MMU 的硬件部分？
- nemu/src/memory/page.c（自己建立文件）
  - 完成 page_translate() 函数
- 在vaddr_read和vaddr_write中进行地址转换
- 注意理解和使用 mmu.h 中的宏和定义

额外的寄存器和操作指令

- 新的寄存器：CR0 和 CR3
- 用处和理论课所学相同
- 需要添加两条特殊的 MOV 指令，在 system.c 中已定义好，填充即可
- 注意对 CR0 寄存器进行初始化才能开启分页

page_translate() 函数（需要实现）

- 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
- 注意区分这个过程中，哪些是虚拟地址，哪些是物理地址

转换示例

```
src/memory/memory.c,85,page_translate] addr:0x8048003
src/memory/memory.c,87,page_translate] CR3:0x1d9a000
src/memory/memory.c,91,page_translate] dir:0x20
src/memory/memory.c,94,page_translate] pdAddr0x1d9a080
src/memory/memory.c,98,page_translate] pdAddr:0x1d9a080  pd_val: 0x1d9c027
src/memory/memory.c,104,page_translate] ptAddr:0x1d9c120
src/memory/memory.c,109,page_translate] ptAddr:0x1d9c120  pt_val: 0x1d9b047
src/memory/memory.c,122,page_translate] paddr:0x1d9b003
```

数据跨页读写（需要实现）

- 需要的时候再来实现，不必提前实现，免得到时候出错不好定位错误点

- 实现思路
  - 什么时候表示跨页了？
  - 跨页时怎么办？
  - 注意事项

添加PTE(软件)

- 准备内核页表
- nexus-am/am/arch/x86-nemu/src/pte.c中的_pte_init()函数
  - 填写从虚拟地址空间[0, 128MB)到物理地址空间[0, 128MB)的映射，包括页目录和页表
  - 在CR3中设置页目录的首地址
  - 在CR0中设置启动分页的标志
- 从此之后硬件对每一个地址都进行转换

给用户进程分配虚拟地址空间

- 将用户程序链接到0x8048000
  - 避免与内核的虚拟地址空间[0, 128MB)重合
  - 用户程序可以使用不受物理内存容量限制的虚拟地址
  - 想一想，为什么是0x8048000？
- 链接器和程序无需关心运行时的物理地址
  - 交给OS来管理，MMU来落实

给用户进程分配虚拟地址空间

- nanos-lite/src/proc.c中的load_prog()
  - 通过_protect()创建一个默认的虚拟地址空间
  - 修改loader()，以页为单位进行加载
    - 申请一页空闲的物理页
      - nanos-lite/src/mm.c中的new_page()
    - 把这一物理页映射到用户程序的虚拟地址空间中，nexus-am/am/arch/x86-nemu/src/pte.c中的_map()
    - 从文件中读入一页的内容到这一物理页上
      - fs_read()
  - 切换到用户程序的虚拟地址空间中
    - nexus-am/am/arch/x86-nemu/src/pte.c中的_switch()
  - 跳转到程序入口地址

_protect() 函数

- 创建一个虚拟用户空间
- 一个虚拟用户空间由什么标识？
  - 起始虚拟地址
  - 终止虚拟地址
  - 页目录基地址
- 虚拟空间作为进程描述符 PCB 的一个成员

PCB：进程描述符（简单讲解）

- 想听复杂讲解，之后PA4会有
- 每个进程在 OS 中以一个个 PCB 的形式存在
- 目前我们只需要了解其中一个成员为 as，该成员存有当前进程的虚拟地址空间
- 切换到本进程时，将本进程 PCB 中页目录基址装入 CR3，从而实现虚拟地址空间转换

_map() 函数（不需要实现）

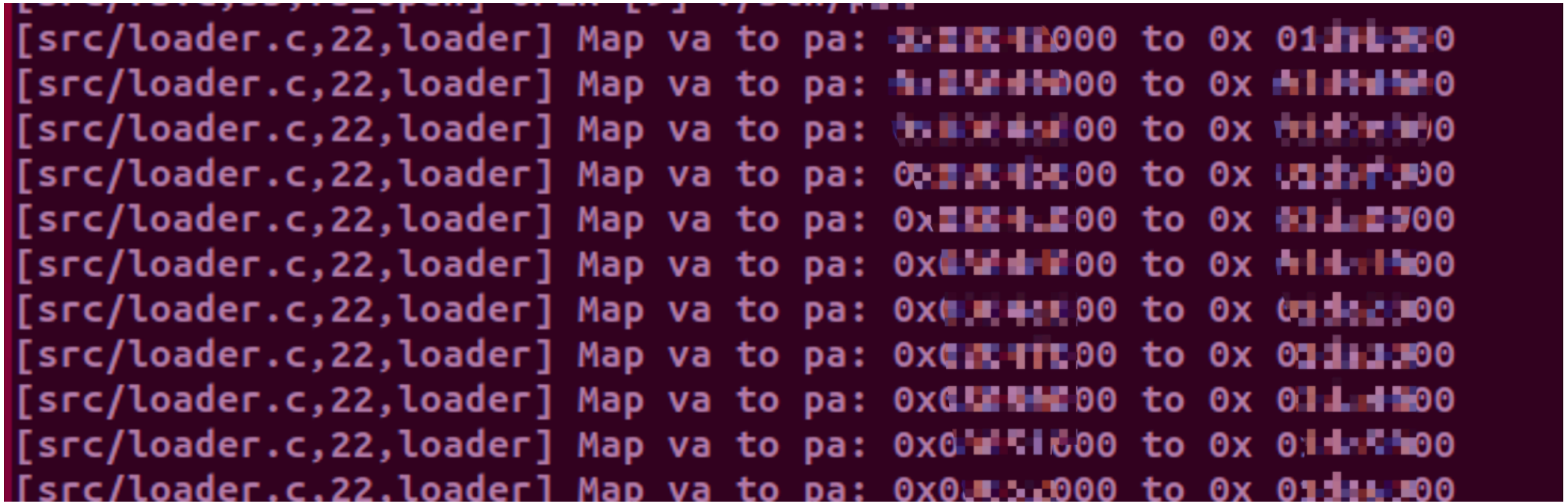
- 该函数用于将一页虚拟内存页映射到物理内存中建立映射关系
- 根据页目录基地址选中一个页目录项，然后申请一张页表
- 选中一个页表项，并与给予的物理页建立映射，建立映射的实质是把物理页的首地址怎么样？

非常重要： Loader() 函数的修改

- 每次调用 _map() 函数前，通过 Log() 显示出每次程序调用 _map() 传入的第二个和第三个参数（va 和 pa），**必须截图写进报告里**

- 必须按照我提供的这种写法来调用_map()，以证明自己成功实现了本功能和前面的所有功能：

```
void *pa = ???;
void *va = ???;
Log("Map va to pa: 0x%08x to 0x%08x", va, pa);
_map(???, va, pa);
```


[src/loader.c,22,loader] Map va to pa:
[src/loader.c,22,loader] Map va to pa:
[src/loader.c,22,loader] Map va to pa:
[src/loader.c,22,loader] Map va to pa:
[src/loader.c,22,loader] Map va to pa:
[src/loader.c,22,loader] Map va to pa:
[src/loader.c,22,loader] Map va to pa:
[src/loader.c,22,loader] Map va to pa:
[src/loader.c,22,loader] Map va to pa:
[src/loader.c,22,loader] Map va to pa:
[src/loader.c,22,loader] Map va to pa:
[src/loader.c,22,loader] Map va to pa:

堆区管理（不需要实现）

- 之前我们令 sys_brk 系统调用总是直接返回 0 表示申请成功，为什么？
- 因为当时整个空间由单个程序使用

- 现在我们令 mm_brk 成为 sys_brk 系统调用的处理函数，怎么实现？
- 很简单，从目前的程序间断点开始，到新的程序间断点为止，逐页地申请一个空闲页并建立映射

不同层次对分页的支持

- 硬件 – MMU，运行时刻的地址转换
- AM – 屏蔽页表结构和虚拟地址空间指针(CR3)等体系结构相关的细节
  - _switch(), _map()
- OS – 页面分配/替换算法

- 实验中的简化 – 只分配不回收

PA总结

	TRM	IOE	ASYE	PTE
Apps	-	文件操作		malloc
库		libos(系统调用接口)		
Nanos-lite	指令 堆 _putc() _halt()	设备文件	系统调用 上下文切换	页面分配/管理
AM		_uptime() _read_key() _draw_rect	保存/恢复现场 事件打包 创建进程上下文	创建虚拟地址空间 创建映射
NEMU	指令生命周期	端口I/O, 内存映射I/O	I386中断机制	I386分页机制

本学期 PA 必做部分到此为止

- 同学们有什么感想？