Project 5 Virtual Memory设计文档

中国科学院大学

张传奇、吴昊轶

2016.12.12

# 内核内存管理设计

（1）你理解的内存虚、实址转换过程

每个程序拥有自己的地址空间，这个虚拟空间被分为很多块，每一块称作一个页或者页面，此次实验中为4096B（4KB）。

先将请求的虚拟地址送到内存管理单元（MMU），根据虚拟地址算出落在哪个页面上，根据映射结果找到相应的页，并将转换后的地址加上低位的偏移量，送到内存总线上；如果得到页面后，读取其present/absent bit（p位）发现页并不在物理内存中，将调用缺页中断（page fault），操作系统决定放弃某一个页，根据是否被修改过决定是否写回硬盘，之后将需要的页装入该页，p位改成1，并修改MMU相应的映射函数，最后重新启动引起中断的指令。

（2）如何管理物理内存，你用来描述管理物理内存页的元数据有哪些？这些元数据信息各自的主要用途是什么？

在memoty.h中定义了page\_map\_entry\_t结构，它将track所有页面信息。

* 1. Node结构用于链表队列管理
  2. 每个页面发生缺页中断的虚拟地址为vaddr；
  3. 该页面若被写回磁盘，对应的程序的扇区起始位置swap\_loc；
  4. 需要写回的扇区数swap\_size；
  5. is\_pinned表明该页是否为不可替换的；
  6. is\_available表明该页是否可用，若已经写入数据则不可用，记为False；
  7. pid在此处用于表明页面的归属
  8. index为其在所有pagemap中的索引；
  9. \*page\_directory指向page\_directory的地址。

（3）内核线程内存管理的初始化过程包括哪些操作？页表项的标记位是如何设置的？

内核线程初始化过程由kernel.c的\_start()函数开始运行至memory.c的setup\_page\_table结束。

先通过init\_memory()对所有内存页面的管理项初始化，并对第一个页面进行设置，将其作为kernel page directory，然后开始从0x0地址开始，不断按需分配页表，并将vaddr=paddr将页表项置位，flag为P、RW，这里面注意不要超过物理地址，也不用超过N\_KERNEL\_PTS个页表。最后需要将SCREEN\_ADDR处的页表项额外修改成P、RW、US，让用户态也可读写。

# 用户态进程内存管理设计

（1）用户态进程内存管理的初始化过程包括哪些操作？页表项的标记位是如何设置的，为什么这么设置？如何决定有多少个页目录和页表？

利用pagealloc取到一个页面并pin上，作为进程页目录，并记录到进程的pcb中。然后为用户栈分配pin住的页面作为栈页表，挂载到页目录中去，然后根据N\_PROCESS\_STACK\_PAGES来分配栈的页，虚拟地址从p->user\_stack开始向下。同样的，也要为进程建立页表，虚拟地址从p->start\_pc开始，向上分配，挂载到页目录中，以上几种页面都是RW、US、P。但是对于进程执行的页面初始化而言，并不需要真正分配页，只是分配好地址，需要的时候，会启动page fault按需分配，并从硬盘中读取数据，所以是RW、US。另外还需要将内核的页表也挂载到页目录中，设置成RW、US、P，以便让进程可访问。

（2）该设计中，哪些页属于pinning pages？这些页的管理与其他页的管理在你的设计中有何不同？

内核页目录，内核页表，进程页目录，进程页表，进程栈页表，进程栈页面。Page map中，pinned设置为True的页属于pinning pages。pinning pages不能被替换，它们不出现在swap queue中。

（3）该设计中，是否会有缺页中断发生？你按照什么策略分配物理页？

该设计中，会发生却页中断，因为我们对于进程代码段是按需分配。如果进程第一次需要执行的时候，会产生一个缺页中断，当然如果页面被换出，再次访问也会发生缺页中断。

先遍历一遍pagemap，找到还没有使用的页面来分配。如果没有可用页面，按照FIFO的方式来清理出一个空余的物理页。

# 缺页中断与swap处理设计

（1）何时发生缺页中断？你设计的缺页中断处理流程是怎样的？

当需要访问的页不在内存中时，发生缺页中断。流程如下：

①在exception\_14中获取fault\_addr;

②找到对应缺页的page directory和page table；

③给这个页表项分配物理页面；

④将页面内容从磁盘加载进来；

⑤更新页表项。

（2）你设计的页替换策略是怎样的？如果有做bonus，请说明为什么你bonus设计的页替换策略能提高效率？

最终采用的是FIFO二次机会用于页面替换。使用该算法是有效的、现实的。因为相较而言LRU算法虽然效率高但是难以实现或实现起来开销太大。

该算法本身易于实现，其避免了把较早进入的但是经常使用的页面替换出去，有效减少缺页中断发生的次数。并且，对于可调度页面不多的情况下，执行效率很高。

# 关键函数功能

（1）init\_memory 对所有内存页面初始化

①对PAGEABLE\_PAGES个页面进行初始化；

②将第一个页面固定，用于kernel page directory；

③将内核页目录清零；

④固定N\_KERNEL\_PTS个页面（不超过PAGEABLE\_PAGES）用于内核页表；

⑤调用set\_ptab\_entry\_flags对页表的标记位进行设置。

（2）page\_alloc 找到一个页空闲页面并初始化页面属性

①先寻找一个空闲页面，若找到了，跳出循环，index\_free即为该页的索引；若找不到，调用page\_replacement\_policy函数，并调用page\_swap\_out将此页替换出去；

②初始化该页面属性，如果没有pin，将之加入队列。将之连接到对应的页表中。

③将index\_free对应的进程页清零。

1. setup\_page\_table 为一个线程或进程建立二级页表
2. 判断是否为线程，若是，直接返回kernel pdir；若不是，进行下述操作；

②利用page alloc建立页目录；

③对于进程，还需要建立栈，位于二级页表顶端；

④依据初始地址和页大小，逐个对页表和页面进行设置。

1. page\_fault\_handler 处理缺页中断
2. 申请锁，防止多个缺页中断同时申请页面；

②调用page\_alloc申请并初始化一个空闲页面（第i个）；

③调用page\_swap\_in将磁盘内容写入第i个页；

④释放锁。

1. page\_swap\_in 将需要的内容从磁盘相应区域写入第i页

调用scsi\_read函数将磁盘相应区域的内容写入第i页，写入后需要将PE\_P置为1。

1. page\_swap\_out 将第i页内容写回磁盘相应区域

通过page\_replacement\_policy函数确定第i页将被替换后，将其PE\_P置0，检查dirty位，若需要写回，计算相应的磁盘区域，调用scsi\_write函数写回。

1. page\_replacement\_policy 决定哪个页被替换，将该页索引返回

首先定义需要被替换的页为victim，在一个循环查询中，不断检查access位，找到的第一个此轮查询过程中未被使用的页将被赋给victim，将之清除出队列，最终返回合法的victim。

参考文献

1. http://www.cs.princeton.edu/courses/archive/fall15/cos318/projects/project5/p5.html