山东大学 软件 学院

**操作系统课程设计** 实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202000300125 | 姓名：贾星宇 | | 班级：计软20.5 |
| 实验编号：Lab7 | | | |
| 实验题目：虚拟内存 | | | |
| 实验学时：6 | | 实验日期：2022年12月5日 | |
| 实验目的：  1. 在未实现虚拟内存管理之前，Nachos在运行一个用户进程的时候，需要将程序运行所需全部内存空间一次性分配。虚拟内存实现将突破物理内存限制。本实验核心任务为根据理论学习中涉及的对换（Swapping）技术，在Lab6的基础上，设计并实现用户空间的虚拟内存管理。  2. 用户进程的帧数采用固定分配(建议5帧)，局部置换。  3. 实现“纯按需调页”(pure demand paging)。  4. 页置换算法可以采用LRU、增强型二次机会、二次机会、FIFO等算法之一，或自己认为合适的其他算法(不包括随机置换)。  5. 对class Statistics进行调用及修改，以便在程序结束时打出页故障次数及将牺牲页写入交换空间的次数。  6. 使用lab7目录中的示例程序n7(若lab7额外实现了多种算法，可用自己的lab7)，测试用户程序用同样ARRAYSIZE参数值的sort，但不同的页置换算法(详见code/lab7/n7readme.txt)多次运行n7。不同页置换算法运行结束时显示的user ticks数是否一样？解释这是为什么？  7. 最优页置换算法(OPT)有最低的页故障率，但需要未来的页面引用信息，因此不能用于实际环境，主要用于评估其他页置换算法的性能。在前述1-5实现的基础上，给出在Nachos中获得最优页置换算法页故障次数的具体实现方法(不要求实现可运行的代码。在实验报告中用文字描述即可，必要时可在文字中结合进关键代码片段、数据结构、对象等说明)。 | | | |
| 硬件环境：  HUAWEI matebook14 2020笔记本  Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz CPU  8GB内存  512GB SSD | | | |
| 软件环境：  宿主机：Windows 10 21H2 64位  虚拟机软件：VMware Workstation Pro 16.1.2 build-17966106  Linux：Ubuntu 14.04.6 LTS Desktop i386 (Trusty Tahr)  gcc/g++：(Ubuntu 4.8.4-2ubuntu1~14.04.4) 4.8.4  MIPS交叉编译器：gcc-2.8.1-mips.tar.gz  Nachos：Nachos-3.4-UALR-2022 | | | |
| 实验步骤与内容：  根据各个实验的内容，以及实现的过程，可写的包括但不限于：  解决问题的思路；  实现步骤；  关键源代码及注释(尽量以文本格式，且必须与提交的源代码一致)；  程序说明，特别是自己认为的精彩之处；  调试过程及记录；  运行结果(可文本格式，必要时抓屏)；  等等…   1. 实现纯按需调页的、用户进程帧数为5的FIFO页置换算法：   类似于实验六，我们在调用-x命令后系统会调用StartProcess方法来实现用户进程的运行，而此方法中关键部分为AddrSpace构造方法和Run之后出现异常而调用的ExceptionHandler方法。因此我们从这儿展开。  **AddrSpace构造方法**  在构造方法中，我们需要首先读取并计算用户进程地址空间需要的大小，随后为其分配并初始化：    102行之前的方法与实验6基本一致，我将初始化页表和文件地址的方法分别写到了两个封装后的方法中。  首先是初始化页表。在初始化页表时，需要为页表初始化一些属性，如虚拟页号、是否dirty等。同时，如果是用户栈，需要将此页的类型修改为用户栈类型（我为页表新增type属性以便后续写入）。  同时，初始化文件地址的方法主要负责为各种段分配地址空间并存到机器的主存中。首先以noff文件的格式读取运行文件头，随后依次判断文件的代码段、初始化数据段、未初始化数据段是否存在，如果存在的话就取出这段所对应的页并设置此页的类型和地址，以便后续修改。  **页错误系统调用**  随后，我们前往exception类新增页错误后的系统调用方法，核心思想为读取出缺页寄存器中存储的地址，随后将此地址转换为虚拟页号并调用FIFO换页方法换页：    FIFO方法的实现方式主要有两个步骤：首先判断是否是最初的5个页，随后，如果是最初的5个页，我们只需要将新页写入pageTable并读取新页数据；如果不是，则调用swap方法。  由于我们需要实现纯按需调页，而纯按需调页要求在一开始不读取页表到内存，而是在需要时发生页错误中断并调用页置换方法来实现新页的换入。但是，如果是传统的旧页换出、新页换入，我们只需要用旧页的物理地址替换新页的物理地址即可，但是如果是初始的前五个页，我们不存在旧页的说法，那么我们只需要通过维护一个pagenum，每次读入新页后加一来顺序分配前五个页的物理地址即可。因此，我们维护一个pagenum数组，初始值为0，当每次执行FIFO时加1，如果超过5说明后续换页都是旧页换出新页换入，是传统的页置换。如果在5以内，说明只需要将新页换入内存并分配地址空间为pagenum即可。具体方式如下：    关于传统的页置换，我们需要调用Swap方法实现旧页换出和新页换入，因此我们在Swap中依次调用WriteBack和ReadIn方法实现上述两个功能。同时，Swap函数返回值为WriteBack函数的返回值，也就是写回的页数目，如果写回的页是dirty，那么为1，否则为0。  在WriteBack方法中，首先判断旧页是否为dirty，如果是的话则 根据页的类型判断如何写入。一般情况下，我们需要写入的为用户栈空间，这里我们借用了在运行示例程序n7后生成的**SWAP0文件**作为我们的磁盘来写。具体方式如下：    在写回后，我们在Swap方法中为初始化新页的属性，随后执行ReadIn方法。ReadIn方法的主要作用为根据不同页的类型前往不同的地址处读取文件并存储到machine的mainMenory中。具体实现方式如下：    至此，我们便实现了纯按需调页的、用户进程帧数为5的FIFO页置换算法。运行结果较大，我们放置到最后展示。   1. 对class Statistics进行调用及修改，以便在程序结束时打出页故障次数及将牺牲页写入交换空间的次数。   这个方式比较简单，我们可以看到在exception类的ExceptionHandler方法中，在执行完FIFO后对stats进行了两项修改  stats->numPageFaults++;  stats->numWriteBack = stats->numWriteBack + k;  其中k是FIFO返回值，也就是是否有写回。这样在程序执行结束并调用Statistics的Print方法后，就可以打印了。Print方法增加如下：     1. 使用lab7目录中的示例程序n7(若lab7额外实现了多种算法，可用自己的lab7)，测试用户程序用同样ARRAYSIZE参数值的sort，但不同的页置换算法(详见code/lab7/n7readme.txt)多次运行n7。不同页置换算法运行结束时显示的user ticks数是否一样？解释这是为什么？   可以发现，不同的页置换算法打印的ticks数确实是不一样的：  最优页置换：    FIFO页置换：    二次机会置换：    增强型二次机会置换：    LRU置换：    可以发现，虽然用户态执行的ticks有所不同，但是user的ticks减去page fault数量皆为331144。这是因为每种页置换算法发生的页错误数不同，而每次发生页错误时需要设置PC并重新执行原来的页，所以页错误次数影响着ticks。   1. 最优页置换算法(OPT)有最低的页故障率，但需要未来的页面引用信息，因此不能用于实际环境，主要用于评估其他页置换算法的性能。在前述1-5实现的基础上，给出在Nachos中获得最优页置换算法页故障次数的**具体实现方法**(不要求实现可运行的代码。在实验报告中用文字描述即可，必要时可在文字中结合进关键代码片段、数据结构、对象等说明)。   最优置换算法（OPT）是指，其所选择的被淘汰页面，将是以后永不使用的，或许是在最长(未来)时间内不再被访问的页面。采用最佳置换算法，通常可保证获得最低的缺页率。但由于人们目前还无法预知一个进程在内存的若干个页面中，哪一个页面是未来最长时间内不再被访问的，因而该算法是无法实现的，但可以利用该算法去评价其它算法。  为了得到最优页置换算法页故障次数，我选择通过实现此算法的方式来完成。首先，为了完成OPT，我们需要利用其他算法获取整个程序运行流程中需要的页。根据n7readme文件的提示，示例程序实现OPT是通过存储文件REFSER0来实现的。我们打开此文件的txt格式：    可以发现这个文件完整的记录了用户程序运行的整个流程依次需要调用的页。生成此文件可以通过Machine类中的ReadMem和WriteMem完成，因为每次系统需要使用页时，就通过Translate方法获取虚拟页所对应的物理页位置，如果没有则返回异常进行中断并执行页置换算法。因此我们分别在ReadMem和WriteMem方法执行转换前向文件REFSER0中写入需要的页，这样在执行完用户程序的同时这个预测文件便构建好了。  随后，我们开始执行OPT方法 。在执行开始同样的执行，我们需要改的是出现页错误后调用的方法，我们在addrspace.cc类中新增OPT方法，当调用此方法后，我们首先使用AddrSpace::Translate方法得到需要换入的页，随后判断是否为前五个初始需要直接换入不换出的页，如果是则直接换入，如果不是，那么要找我们需要换出的页。我们首先设置一个宏变量LOOKAHEAD，表示在做换出新页决策时向前看多少个页，再增加一个大小为5的数组appearNum[]记录每个帧在后面LOOKAHEAD个页中出现的次数。然后执行循环算法，从此页的位置处向后看Translate个，每次循环体内就循环现在有的5个帧，如果看到的这个页在我们的帧上则加一，即appearNum[j]++。结束之后我们找出appearNum数组中加的次数最小的那个帧并选择将其换出，换出时执行Swap算法类似于先前的换入换出即可。  当然，我们也需要在system类中增加pagepos属性记录执行到了哪一页，并在每次读或者写页时自增。   1. 实验结果  |  | | --- | | User program: ../test/sort.noff, SpaceId: 0, Memory size: 2208  Max frames per user process: 5, Swap file: SWAP0, Page replacement algorithm: FIFO  SpaceId: 0, page table dump: 18 pages in total  ============================================  Page, Frame, Valid, Use, Dirty  0, -1, 0, 0, 0  1, -1, 0, 0, 0  2, -1, 0, 0, 0  3, -1, 0, 0, 0  4, -1, 0, 0, 0  5, -1, 0, 0, 0  6, -1, 0, 0, 0  7, -1, 0, 0, 0  8, -1, 0, 0, 0  9, -1, 0, 0, 0  10, -1, 0, 0, 0  11, -1, 0, 0, 0  12, -1, 0, 0, 0  13, -1, 0, 0, 0  14, -1, 0, 0, 0  15, -1, 0, 0, 0  16, -1, 0, 0, 0  17, -1, 0, 0, 0  ============================================  Demand page 0 in(frame 0)  SpaceId: 0, page table dump: 18 pages in total  ============================================  Page, Frame, Valid, Use, Dirty  0, 0, 1, 1, 0  1, -1, 0, 0, 0  2, -1, 0, 0, 0  3, -1, 0, 0, 0  4, -1, 0, 0, 0  5, -1, 0, 0, 0  6, -1, 0, 0, 0  7, -1, 0, 0, 0  8, -1, 0, 0, 0  9, -1, 0, 0, 0  10, -1, 0, 0, 0  11, -1, 0, 0, 0  12, -1, 0, 0, 0  13, -1, 0, 0, 0  14, -1, 0, 0, 0  15, -1, 0, 0, 0  16, -1, 0, 0, 0  17, -1, 0, 0, 0  ============================================  Demand page 1 in(frame 1)  SpaceId: 0, page table dump: 18 pages in total  ============================================  Page, Frame, Valid, Use, Dirty  0, 0, 1, 1, 0  1, 1, 1, 1, 0  2, -1, 0, 0, 0  3, -1, 0, 0, 0  4, -1, 0, 0, 0  5, -1, 0, 0, 0  6, -1, 0, 0, 0  7, -1, 0, 0, 0  8, -1, 0, 0, 0  9, -1, 0, 0, 0  10, -1, 0, 0, 0  11, -1, 0, 0, 0  12, -1, 0, 0, 0  13, -1, 0, 0, 0  14, -1, 0, 0, 0  15, -1, 0, 0, 0  16, -1, 0, 0, 0  17, -1, 0, 0, 0  ============================================  Demand page 17 in(frame 2)  SpaceId: 0, page table dump: 18 pages in total  ============================================  Page, Frame, Valid, Use, Dirty  0, 0, 1, 1, 0  1, 1, 1, 1, 0  2, -1, 0, 0, 0  3, -1, 0, 0, 0  4, -1, 0, 0, 0  5, -1, 0, 0, 0  6, -1, 0, 0, 0  7, -1, 0, 0, 0  8, -1, 0, 0, 0  9, -1, 0, 0, 0  10, -1, 0, 0, 0  11, -1, 0, 0, 0  12, -1, 0, 0, 0  13, -1, 0, 0, 0  14, -1, 0, 0, 0  15, -1, 0, 0, 0  16, -1, 0, 0, 0  17, 2, 1, 1, 0  ============================================  Demand page 2 in(frame 3)  SpaceId: 0, page table dump: 18 pages in total  ============================================  Page, Frame, Valid, Use, Dirty  0, 0, 1, 1, 0  1, 1, 1, 1, 0  2, 3, 1, 1, 0  3, -1, 0, 0, 0  4, -1, 0, 0, 0  5, -1, 0, 0, 0  6, -1, 0, 0, 0  7, -1, 0, 0, 0  8, -1, 0, 0, 0  9, -1, 0, 0, 0  10, -1, 0, 0, 0  11, -1, 0, 0, 0  12, -1, 0, 0, 0  13, -1, 0, 0, 0  14, -1, 0, 0, 0  15, -1, 0, 0, 0  16, -1, 0, 0, 0  17, 2, 1, 1, 1  ============================================  Demand page 6 in(frame 4)  SpaceId: 0, page table dump: 18 pages in total  ============================================  Page, Frame, Valid, Use, Dirty  0, 0, 1, 1, 0  1, 1, 1, 1, 0  2, 3, 1, 1, 0  3, -1, 0, 0, 0  4, -1, 0, 0, 0  5, -1, 0, 0, 0  6, 4, 1, 1, 0  7, -1, 0, 0, 0  8, -1, 0, 0, 0  9, -1, 0, 0, 0  10, -1, 0, 0, 0  11, -1, 0, 0, 0  12, -1, 0, 0, 0  13, -1, 0, 0, 0  14, -1, 0, 0, 0  15, -1, 0, 0, 0  16, -1, 0, 0, 0  17, 2, 1, 1, 1  ============================================  Demand page 7 in(frame 0)  SpaceId: 0, page table dump: 18 pages in total  ============================================  Page, Frame, Valid, Use, Dirty  0, -1, 0, 1, 0  1, 1, 1, 1, 0  2, 3, 1, 1, 0  3, -1, 0, 0, 0  4, -1, 0, 0, 0  5, -1, 0, 0, 0  6, 4, 1, 1, 1  7, 0, 1, 1, 0  8, -1, 0, 0, 0  9, -1, 0, 0, 0  10, -1, 0, 0, 0  11, -1, 0, 0, 0  12, -1, 0, 0, 0  13, -1, 0, 0, 0  14, -1, 0, 0, 0  15, -1, 0, 0, 0  16, -1, 0, 0, 0  17, 2, 1, 1, 1  ============================================  Demand page 8 in(frame 1)  SpaceId: 0, page table dump: 18 pages in total  ============================================  Page, Frame, Valid, Use, Dirty  0, -1, 0, 1, 0  1, -1, 0, 1, 0  2, 3, 1, 1, 0  3, -1, 0, 0, 0  4, -1, 0, 0, 0  5, -1, 0, 0, 0  6, 4, 1, 1, 1  7, 0, 1, 1, 1  8, 1, 1, 1, 0  9, -1, 0, 0, 0  10, -1, 0, 0, 0  11, -1, 0, 0, 0  12, -1, 0, 0, 0  13, -1, 0, 0, 0  14, -1, 0, 0, 0  15, -1, 0, 0, 0  16, -1, 0, 0, 0  17, 2, 1, 1, 1  ============================================  Demand page 1 in(frame 2)  SpaceId: 0, page table dump: 18 pages in total  ============================================  Page, Frame, Valid, Use, Dirty  0, -1, 0, 1, 0  1, 2, 1, 1, 0  2, 3, 1, 1, 0  3, -1, 0, 0, 0  4, -1, 0, 0, 0  5, -1, 0, 0, 0  6, 4, 1, 1, 1  7, 0, 1, 1, 1  8, 1, 1, 1, 1  9, -1, 0, 0, 0  10, -1, 0, 0, 0  11, -1, 0, 0, 0  12, -1, 0, 0, 0  13, -1, 0, 0, 0  14, -1, 0, 0, 0  15, -1, 0, 0, 0  16, -1, 0, 0, 0  17, -1, 0, 1, 0  ============================================  Demand page 17 in(frame 3)  SpaceId: 0, page table dump: 18 pages in total  ============================================  Page, Frame, Valid, Use, Dirty  0, -1, 0, 1, 0  1, 2, 1, 1, 0  2, -1, 0, 1, 0  3, -1, 0, 0, 0  4, -1, 0, 0, 0  5, -1, 0, 0, 0  6, 4, 1, 1, 1  7, 0, 1, 1, 1  8, 1, 1, 1, 1  9, -1, 0, 0, 0  10, -1, 0, 0, 0  11, -1, 0, 0, 0  12, -1, 0, 0, 0  13, -1, 0, 0, 0  14, -1, 0, 0, 0  15, -1, 0, 0, 0  16, -1, 0, 0, 0  17, 3, 1, 1, 0  ============================================  Demand page 2 in(frame 4)  SpaceId: 0, page table dump: 18 pages in total  ============================================  Page, Frame, Valid, Use, Dirty  0, -1, 0, 1, 0  1, 2, 1, 1, 0  2, 4, 1, 1, 0  3, -1, 0, 0, 0  4, -1, 0, 0, 0  5, -1, 0, 0, 0  6, -1, 0, 1, 0  7, 0, 1, 1, 1  8, 1, 1, 1, 1  9, -1, 0, 0, 0  10, -1, 0, 0, 0  11, -1, 0, 0, 0  12, -1, 0, 0, 0  13, -1, 0, 0, 0  14, -1, 0, 0, 0  15, -1, 0, 0, 0  16, -1, 0, 0, 0  17, 3, 1, 1, 0  ………………  0  Demand page 5 in(frame 4)  SpaceId: 0, page table dump: 18 pages in total  ============================================  Page, Frame, Valid, Use, Dirty  0, -1, 0, 1, 0  1, 3, 1, 1, 0  2, 1, 1, 1, 0  3, -1, 0, 1, 0  4, -1, 0, 1, 0  5, 4, 1, 1, 0  6, 2, 1, 1, 0  7, -1, 0, 1, 0  8, -1, 0, 1, 0  9, -1, 0, 1, 0  10, -1, 0, 0, 0  11, -1, 0, 0, 0  12, -1, 0, 0, 0  13, -1, 0, 0, 0  14, -1, 0, 0, 0  15, -1, 0, 0, 0  16, -1, 0, 0, 0  17, 0, 1, 1, 1  ============================================  1  Demand page 9 in(frame 0)  SpaceId: 0, page table dump: 18 pages in total  ============================================  Page, Frame, Valid, Use, Dirty  0, -1, 0, 1, 0  1, 3, 1, 1, 0  2, 1, 1, 1, 0  3, -1, 0, 1, 0  4, -1, 0, 1, 0  5, 4, 1, 1, 0  6, 2, 1, 1, 0  7, -1, 0, 1, 0  8, -1, 0, 1, 0  9, 0, 1, 1, 0  10, -1, 0, 0, 0  11, -1, 0, 0, 0  12, -1, 0, 0, 0  13, -1, 0, 0, 0  14, -1, 0, 0, 0  15, -1, 0, 0, 0  16, -1, 0, 0, 0  17, -1, 0, 1, 0  ============================================  98  99  100  Demand page 0 in(frame 1)  SpaceId: 0, page table dump: 18 pages in total  ============================================  Page, Frame, Valid, Use, Dirty  0, 1, 1, 1, 0  1, 3, 1, 1, 0  2, -1, 0, 1, 0  3, -1, 0, 1, 0  4, -1, 0, 1, 0  5, 4, 1, 1, 0  6, 2, 1, 1, 0  7, -1, 0, 1, 0  8, -1, 0, 1, 0  9, 0, 1, 1, 0  10, -1, 0, 0, 0  11, -1, 0, 0, 0  12, -1, 0, 0, 0  13, -1, 0, 0, 0  14, -1, 0, 0, 0  15, -1, 0, 0, 0  16, -1, 0, 0, 0  17, -1, 0, 1, 0  ============================================  Machine halting!  Ticks: total 332531, idle 0, system 10, user 332521  Disk I/O: reads 0, writes 0  Console I/O: reads 0, writes 0  Paging: faults 1377, write backs 610  Network I/O: packets receivped 0, sent 0  Cleaning up... | | | | |
| 结论分析与体会：  至此，课程设计结束，通过这七个实验，我也学到了各个方面的知识。  操作系统的理论上，我对操作系统的结构更加清楚，对操作系统如何运行用户进程、如何实现虚拟内存等方法更加了解。  同时，我修改bug、解决问题的能力进一步提升，理解代码的能力也有了突破。  我也认识到，只有花费足够的时间在这上面，我才能够更清楚的理解问题、更有信心去将其做好、讲好。  感谢老师的帮助和辛苦付出！ | | | |