**《操作系统原理》实验报告四**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 王粟鹏 | 学号 | U201817044 | 专业班级 | 软工1803 | 时间 | 2020.4.29 |

1. **实验目的**
2. 理解页面淘汰算法原理，编写演示页面淘汰算法。
3. 验证Linux虚拟地址转化为物理地址的机制。
4. 理解和验证程序运行局部性的原理。
5. **实验内容**

（1） 在 Windows 环境下编写一个程序，模拟实现 OPT,FIFO,LRU 等页面淘汰算法。可以使用数组模拟内存，数组中的元素模拟为指令或数据。写不同方式的程序去访问数组来模拟 CPU 访问内存的情况。分析运算结果，在分配不同的物理块情况下， 各算法的缺页情况有什么规律？可以 srand( )和 rand( )等函数定义和产生“指令” 序列，然后将指令序列变换成相应的页地址流，并针对不同的算法计算出相应的命中率。例如，实验中可以产生 320 条“指令”，每个“虚拟页”存放 10 条指令。进程分配的页框是 4（可变，例如 32）。

（2） 在 Linux 环境下，编写一个小程序，获取该程序中的某个变量的虚拟地址，虚拟页号，页内偏移地址，物理页框号，页内偏移地址，物理地址，并将它们打印出来。建议使用/proc/pid/pagemap 技术。

（3） 在 Windows 环境下，编写一个函数（特点：比较耗时，比如大型的多维数组读写），用不同的方法测试其所花费的时间。在不同环境下比较其时间是否不同，并分析其含义。

1. **实验过程**
2. **实验步骤**
3. **页面淘汰算法**

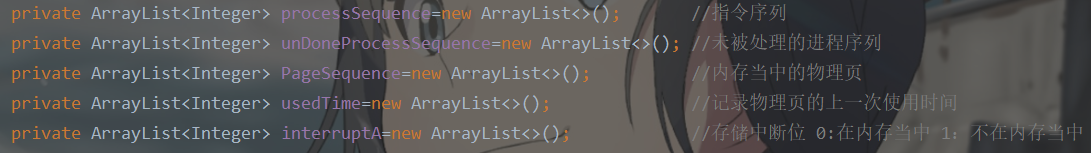
环境：使用IJ java SDK 11

过程：

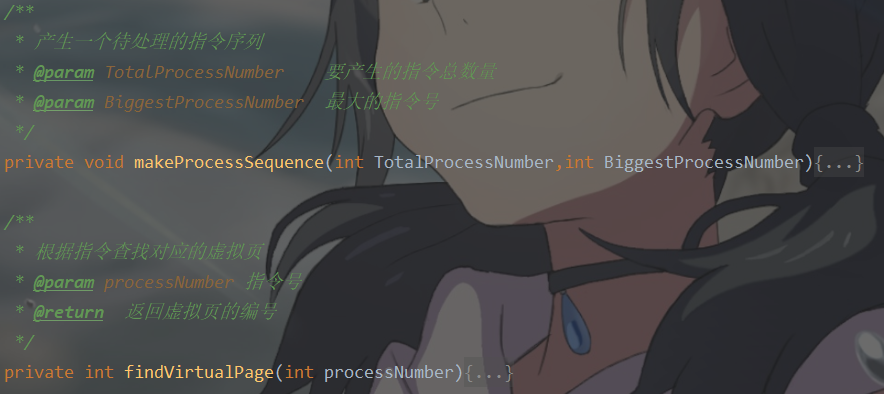
不同的指令使用数字来进行区分，指令和虚拟页的关系为“虚拟页=指令号%10“，使用

容器来代表内存，容器中的数字代表内存中的页。

首先定义在页面淘汰算法中的一些基本参数，比如内存、指令序列等等。



然后创建一些基本的函数，比如产生一个指令序列、根据指令地址查找虚拟地址等。



接下来要写具体的页面淘汰函数。

**FIFO页面淘汰算法：**淘汰在内存存在时间最久的页面，在使用容器代表内存。当有新的页加入时，如果容器的长度小于内存可容纳的最大页数（说明内存当中有空闲位置），则直接使用.add将新的页加入；如果容器的长度等于内存可容纳的最大页数（说明内存当中没有空位置），则将0索引处的页面淘汰（因为每次使用.add添加新页的话，0索引处的元素一定是最早添加的），然后再使用.add将新页加入。在过程中会输出每一步的具体信息，包括内存当中的页、当前访问的页、淘汰的页等。



使用**FIFO**算法对指令序列进行操作：首先要对中断存储容器、内存进行初始化，然后创建一个指令序列，使用FIFO算法进行操作。最后输出该次操作的缺页率。



**OPT页面淘汰算法：**淘汰最长时间不会使用到的页面。为了实现这一算法，要编写一个辅助函数，findLast() 来找到最长时间内不会使用到的页面。

findLast（）遍历未执行的指令unDoneProcessSequence,当该指令的虚拟页在内存当中，而且之前没有访问过该页时，count++。这样当count>=TotalPageNumber-1时，说明只剩下一页未访问，剩下的一页为即最后使用的，该页就为要寻找的最长时间不会使用到的页。



然后编写**OPT**页面淘汰算法的函数，具体步骤和**FIFO**的相似，只不过淘汰页面时使用findLast函数获得要淘汰的页面。然后编写函数使用**OPT**算法对一个指令序列进行操作。

**LRU页面淘汰算法：**如果OPT算法是向后寻找，那么LRU就是向前记录。淘汰最久未使用的页面。在这里要建立一个辅助的存储容器useTime，容器中存放着内存中全部的页框号，当使用某个页时，就从useTime中remove掉该页，然后再使用add将该页添加到useTime当中，这样就可以保证最近使用过的页面在容器的最后，0索引处的页框是最久未使用的页框。

然后要编写LRU页面淘汰算法，基本步骤和FIFO的相似，只不过淘汰的页面为useTime中的0索引页。最后使用LRU算法对一个指令序列进行操作。

**三个页面淘汰算法的对比：**

仍旧首先创建一个指令序列，然后使用三种页面淘汰算法对该相同的指令序列进行访问操作，在换页面淘汰算法时，要注意参数的初始化，然后输出最终的缺页率，进行比较。

1. **虚拟地址物理地址映射。**

环境：Linux下的codeBlocks

过程：使用到了/proc/self/pagemap计数。Linux文件目录/proc下有一个链接目录名为self，哪一个进程打开了它，self中存储的信息就是所链接进程的。self中有一个名为pagemap的文件，专门用来记录所链接进程的物理页号信息。这样通过/proc/pid/pagemap文件，允许一个用户态的进程查看到每个虚拟页映射到的物理页。访问该文件需要提供权限。然后要编写函数，在主线程当中调用该函数打开文件目录，然后根据虚拟地址找到对应的物理地址。

传入虚拟地址，然后计算该虚拟地址的虚拟页号、文件中的偏移量以及页面中的偏移量。



然后打开文件，进行一些检查操作，并将信息写入到item变量当中。



然后根据item的值，计算物理页以及物理地址。



在这过程中输出相应的地址信息。编译完成之后，在终端使用root权限运行程序，即可获得相应的输出结果。

1. **程序运行局部性验证**

环境：Windows下的VS 以及 IJ java SDK 11环境

过程：在不同的环境、以不同的方式将一个二维数组遍历n次，分别统计所需要的时间，然后进行比较。在VS环境下使用clock()记录时间。在java中使用System.currentTimeMillis()记录时间。在遍历之前和之后分别记录一次时间，然后根据差值得到运行时间。

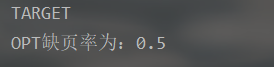
同样是对一个50\*50的数组进行50000次赋值。

1. **解决错误和优化**
2. 错误类型：操作过程错误

错误原因：在页面淘汰算法比较函数中，在更换页面淘汰算法时没有将中断容器初始化



错误现象：LRU算法缺页率比OPT算法的缺页率还要低。



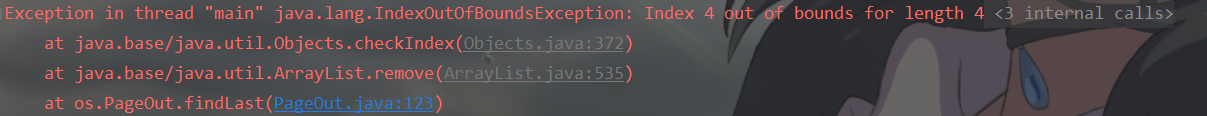
解决办法：在执行页面淘汰算法之前要增加一段代码，用来实现中断容器的初始化。（将错误原因中注释掉的代码恢复到代码当中即可）

1. 错误类型：特殊的语法错误

错误原因：在将页面从容器当中移除时，索引出了错误。因为索引的类型是 int ，而容器中页面对象的类型为Integer，在移除某一页面k时，是这样传递的参数.remove(k)，在这种情况下不会移除页面k，而是会移除索引为k处的页面。



错误现象：会出现溢出现象。



解决方法：在传递参数时使用强制类型转换.remove((Integer) k)，让remove()函数识别出要移除的是对象，而不是索引。

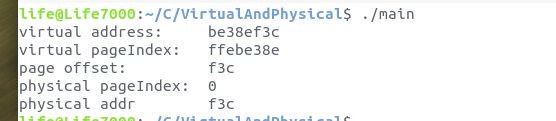


1. 错误类型：操作过程错误。

错误原因：在打开/proc文件时没有使用root权限

错误现象：无法查找虚拟页对应的物理页，显示的物理页为0，物理地址为页内偏移

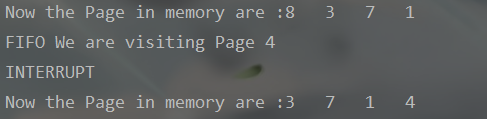
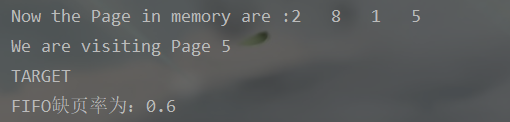
地址。

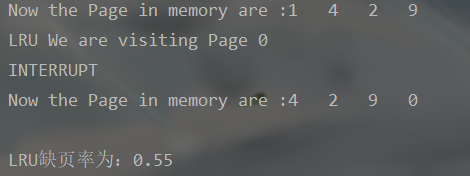
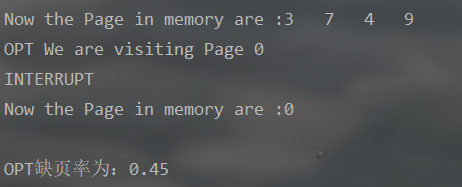


解决办法：在 ./main 指令之前加上 sudo ，以管理员的权限运行该程序，可以正确访问文件，得到对应的物理地址。

1. **实验结果**
   1. 页面淘汰算法。

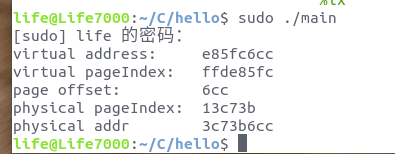
在不同页面淘汰算法当中不仅可以输出每一步的过程，还会输出最终的页面淘汰率。

可以看到缺页率 ：OPT<LRU<FIFO

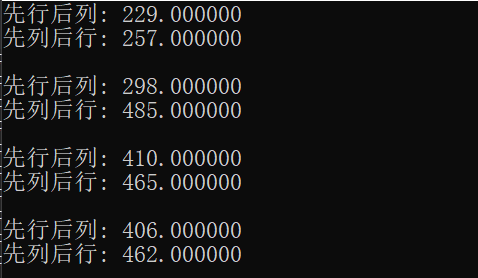
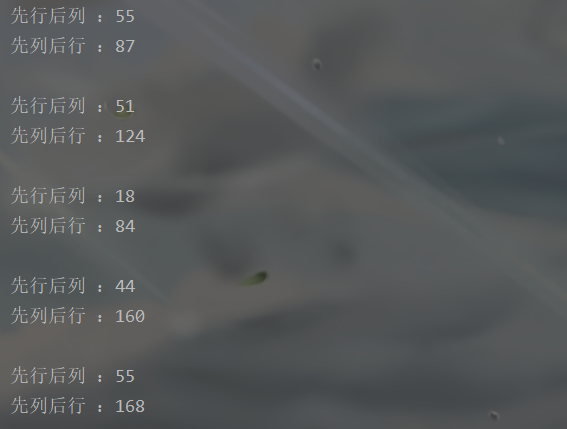
* 1. 虚拟地址物理地址映射



使用root权限打开之后，程序输出相关的虚拟地址和物理地址信息。

* 1. 程序局部性验证

VS环境下： Java SDK 11

可以观察到，先行后列的遍历方式所用的时间要少于先列后行的方式，因为前者的程序局部性高，页跳转的次数少，所以处理时间短。

对于不同环境下，采用相同的遍历方式得到的时间是不同的，即使是同一个环境下，两次相同方式的遍历也消耗时间不同。

1. **体会**

首先对Linux当中的虚拟地址和物理地址的映射有了更加深入的了解，知道了如何获取一个进程中变量的虚拟地址和物理地址。

关于页面淘汰算法部分，通过该此实验的模拟，使得我对页面淘汰算法有了更加具体的认识，清楚地看到了不同页面淘汰算法的优劣，更加深入了解了他们的工作原理。

直观地感受到了代码运行具有较高局部性的好处。