广州大学学生实验报告

**开课学院及实验室：**计算机科学与网络工程学院软件实验室 2019**年**11**月**21**日**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学院** | 计算机科学与网络工程学院 | **年级/专业/班** | 软件171 | **姓名** | 谢金宏 | **学号** | 1706300001 |
| **实验课程名称** | 操作系统实验 | | | | | **成绩** |  |
| **实验项目名称** | 内存管理 | | | | | **指导老师** | 陶文正 |

实验三 内存管理

# 实验目的

1. 了解虚拟存储技术的特点，掌握虚拟存储请求页式存储管理中几种基本页面置换算法的基本思想和实现过程，并比较它们的效率。
2. 了解程序设计技术和内存泄露的原因。

# 实验环境

安装有GCC编译套装的计算机。

# 实验内容

模拟实现请求页式存储管理的几种基本页面置换算法：

1. 最佳淘汰算法（OPT）
2. 先进先出的算法（FIFO）
3. 最近最久未使用算法（LRU）

# 实验原理

## 虚拟存储系统

UNIX中，为了提高内存利用率，提供了内外存进程对换机制；内存空间的分配和回收均以页为单位进行；一个进程只需将其一部分（段或页）调入内存便可运行；还支持请求调页的存储管理方式。

当进程在运行中需要访问某部分程序和数据时，发现其所在页面不在内存，就立即提出请求（向CPU发出缺中断），由系统将其所需页面调入内存。这种页面调入方式叫请求调页。

为实现请求调页，核心配置了四种数据结构：页表、页框号、访问位、修改位、有效位、保护位等。

## 页面置换算法

当CPU接收到缺页中断信号，中断处理程序先保存现场，分析中断原因，转入缺页中断处理程序。该程序通过查找页表，得到该页所在外存的物理块号。如果此时内存未满，能容纳新页，则启动磁盘I/O将所缺之页调入内存，然后修改页表。如果内存已满，则须按某种置换算法从内存中选出一页准备换出，是否重新写盘由页表的修改位决定，然后将缺页调入，修改页表。利用修改后的页表，去形成所要访问数据的物理地址，再去访问内存数据。整个页面的调入过程对用户是透明的。

（1）最佳淘汰算法（OPT）：选择永不使用或在未来最长时间内不再被访问的页面予以替换。

（2）先进先出的算法（FIFO）：选择在内存中驻留时间最久的页面予以替换。

（3）最近最久未使用算法（LRU）：选择过去最长时间未被访问的页面予以替换。

## 生成程序输入

首先用srand( )和rand( )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。

（1）通过随机数产生一个指令序列，共320条指令。指令的地址按下述原则生成：

A：50%的指令是顺序执行的

B：25%的指令是均匀分布在前地址部分

C：25%的指令是均匀分布在后地址部分

具体的实施方法是：

A：在[0，319]的指令地址之间随机选取一起点m

B：顺序执行一条指令，即执行地址为m+1的指令

C：在前地址[0,m+1]中随机选取一条指令并执行，该指令的地址为m’

D：顺序执行一条指令，其地址为m’+1

E：在后地址[m’+2，319]中随机选取一条指令并执行

F：重复步骤A-E，直到320次指令

（2）将指令序列变换为页地址流

设：页面大小为1K；

用户内存容量4页到32页；

用户虚存容量为32K。

在用户虚存中，按每K存放10条指令排列虚存地址，即320条指令在虚存中的存放方式为：

第 0 条-第 9 条指令为第0页（对应虚存地址为[0，9]）

第10条-第19条指令为第1页（对应虚存地址为[10，19]）

………………………………

第310条-第319条指令为第31页（对应虚存地址为[310，319]）

按以上方式，用户指令可组成32页。

# 使用的系统调用函数

这是一个模拟程序，没有使用系统调用函数。

# 实验过程记录与结果分析

实际编码中实现的页面调度算法有平凡页面调度算法Trivial、最佳置换法OPT、先进先出置换法FIFO、最近最久未使用置换法LRU。

平凡页面调度算法Trivial是为了充当测试的基线而编写的在各种运行条件下命中率保持不变的算法，需要进行页面调度时，它总是将缺少的页装入运行内存的第一个位置。因此无论运行内存有多大，Trivial算法仅利用运行内存的第一页，命中率很低。若编程中出现某算法的命中率低于Trivial算法，那么程序中一定存在错误。

下图为Trivial算法的关键部分，完成代码请参见“manager.hpp”中的Trivial Manager。



图表 1 Trivial算法的替换部分

最佳置换法OPT算法是理论上命中率最高的算法。为实现OPT算法，在要调入一个新页面而需要淘汰一个老页面时，检查程序的页面流，并在已经装入运行内存的老页面中选择在最远的将来使用或者不再使用的老页面进行淘汰。若编程中出现某算法的命中率高于OPT的命中率，那么程序中一定存在错误。

下图为OPT算法的关键部分，完成代码请参见“manager.hpp”中的OPT Manager。



图表 2 OPT算法

通过对OPT原理的分析可以得知，OPT算法是不能实现的。因为要真正实现OPT算法，就必须要了解程序在整个运行过程中的页面走向，而这是不现实的。只能实现OPT算法的一些近似算法。

OPT算法的一个近似算法为LRU算法。LRU采用“过去信息能预测较近的未来”的想法，在要淘汰老页面时，选择最近最久未使用的页面进行淘汰。编程实现这个算法时，采用计数器方法。置一个全局的计数器，并使每个页面对应一个访问时间字段。每访问一次页面，全局计数器的值加1，并将全局计数器的值赋给被访问页面的访问时间字段。页面的访问时间字段值越大，意味着这个页面在越近期内被访问过。这样，淘汰老页面时，可以选择淘汰运存中访问时间字段最小的页面。



图表 3 LRU算法

同样的，完整代码请参见“manager.hpp”中的LRU Manager。

还有一种简单的页面置换算法是FIFO算法。FIFO算法总是淘汰运存中停留时间最长的页。由于被淘汰的页在运存中的位置呈现周期性，淘汰页面的位置顺序为位置1、位置2……位置N、位置1、位置2……，因而FIFO算法的实现非常简单，只需要将页面循环地进行淘汰即可。完整代码请见“manager.hpp”中的FIFO Manager。



图表 4 FIFO算法

根据上述逻辑编写代码进行测试。测试时，用户的虚拟内存容量为32页，运行内存的大小为变量，从1页逐步增加到32页。由于Trivial算法的命中率总是保持不变，因此对Trivial算法只运行一次测试。对于OPT算法、FIFO算法和LRU算法，总共运行32次测试，每次测试在不同的运行内存大小上运行。

程序的运行框架代码如下：



图表 5 程序框架代码

在程序的一次运行输出参见“output.txt”，下面列出程序的部分输出：



图表 6 自动生成的程序的一部分



图表 7 计算出程序的页面流



图表 8 统计算法的运行结果

将各个算法的命中率统计表和统计曲线在下方列出：



图表 9 各个算法的命中率统计表

图表 10 各个算法的命中率折线图

在上图中，横轴为用户运行内存所能容纳的页面数量，纵轴为命中率。从图表中得知，OPT算法的命中率最高，Trivial算法的命中率最糟糕，FIFO算法和LRU算法不相伯仲。其中，在用户内存较少时，LRU算法的优势较为明显；在用户内存较多时，FIFO算法的优势较为明显。在用户的内存量极多时，FIFO和LRU算法的命中率都接近理想算法OPT。

理论上我们认为LRU的命中率应当高于简单的FIFO，但实际的实验结果表明，在给定的“５０％代码顺序执行、２５％代码取前地址、２５％代码取后地址”的页面走向中，LRU算法实际上与FIFO不相伯仲。造成这一点的原因，我认为与随机生成的页面走向有关。其一，模拟的页面走向中的页面跳转比较频繁，程序顺序执行两条指令后就会跳转到不同序号的页，实际中的页面走向应该比模拟走向页面跳转的频率要小许多。其二，LRU算法是利用过去的信息来预测较近的未来，但随机生成的页面走向中，过去与未来没有联系（完全随机），因此对LRU算法是不利的，这种简单的随机性反而对FIFO算法有利。

# 实验总结

总体而言完成了本次实验的全部要求。页面置换算法原理较为明确，理解起来并不困难，本次实验遇到的困难主要在代码实现上。

本次实验一时兴起采用面向对象的方式进行编程，结果给自己带来了许多麻烦。直观上看体现在代码量比较大。代码量大就增加了理解和维护的难度。代码实现时采用了C++的类模板，初衷是为不同的算法生成不同的DemandPagingMachine对象。例如，OPT算法对应的对象就是DemandPagingMachine<OPTManager>。完成实验之后重新想一想，其实在本次实验中完全没有必要引入类模板的机制，使用C++的继承机制和虚指针就能在保证抽象程度的同时完成所需要的功能。不如说，使用类模板机制反而平添了自己实现代码和别人理解代码的难度。

其二，DemandPagingMachine类和PageReplacementManager类的耦合程度太高，理论上应该用友元实现，但使用友元就遭遇了循环引用陷阱。由于使用了类模板，难以分离定义与实现，不能解决循环引用的问题。又无法使用友元。

应该说C++中的类模板与Java或TypeScript中的泛型机制是很不一样的，在本次实验中不自觉地按照Java和TS的方式进行思考，于是产生了上面所说的问题。为了在今后编写代码的过程中不再出现这样的错误，在切换语言编程时，不能不经思考马上上手编程，而是应该回顾语言设计某种特性的理由，按当前所使用的语言的思维进行构思后再开始编程。

那么回顾自己的代码，目前的代码处于良好的状态吗？我认为，它实现了算法，解决了问题，但远远谈不上优雅或美观。如何处理DemandPagingMachine类和PageReplacementManager类的关系，或者说如何在保证封装性的同时实现良好的代码复用，是一个值得继续研究的问题。

# 实验数据及源代码

实验数据及源代码见“1706300001\_谢金宏\_实验三.zip”。