《操作系统》
实验指导书

南昌大学软件学院

实验 6： 请求分页式存储管理模拟实验

6.1 实验概述

实验目的：

1. 理解页式存储管理的基本原理；
2. 掌握几种常见的页面淘汰算法。

实验语言：c

实验环境：linux、gcc

6.2 实验背景

6.2.1 基本分页式存储管理

在存储器管理中，连续分配方式会形成许多“碎片”，虽然可通过“紧凑”方法将许多碎片拼接成可用的大块空间，但须为之付出很大开销。

如果允许将一个进程直接分散地装入到许多不相邻的分区中，则无须再进行“紧凑”。基于这一思想而产生了离散分配方式。如果离散分配的基本单位是页，则称为分页存储管理方式。在分页存储管理方式中，如果不具备页面对换功能，则称为基本分页存储管理方式，或称为纯分页存储管理方式，它不具有支持实现虚拟存储器的功能，它要求把每个作业全部装入内存后方能运行。

一、页面与页表

（1）页面和物理块

分页存储管理是将一个进程的逻辑地址空间分成若干个大小相等的部分，称为页面或页，并为各页加以编号，从 0 开始，如第 0 页、第 1 页等。相应地，也把内存空间分成与页面相同大小的若干个存储块，称为(物理)块或页框(frame)，也同样为它们加以编号，如 0#块、1#块等等。在为进程分配内存时，以块为单位将进程中的若干个页分别装入到多个可以不相邻接的物理块中。由于进程的最后一页经常装不满一块而形成了不可利用的碎片，称之为“页内碎片”。

（2）页面大小

在分页系统中的页面其大小应适中。页面若太小，一方面虽然可使内存碎片减小，从而减少了内存碎片的总空间，有利于提高内存利用率，但另一方面也会使每个进程占用较多的页面，从而导致进程的页表过长，占用大量内存；此外，还会降低页面换入换出的效率。然而，如果选择的页面较大，虽然可以减少页表的长度，提高页面换入换出的速度，但却又会使页内碎片增大。因此，页面的大

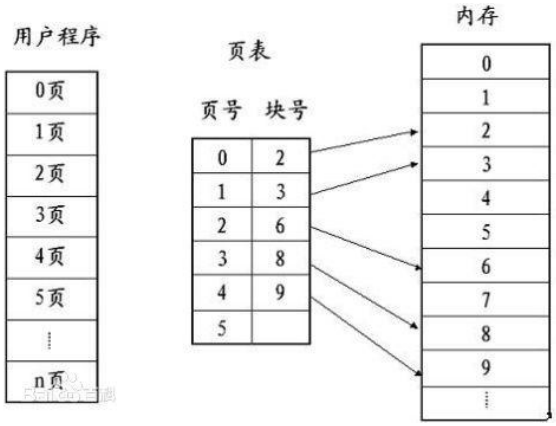
小应选择适中，且页面大小应是 2 的幂，通常为 512 B～8 KB。

二、地址结构

对于某特定机器，其地址结构是一定的。若给定一个逻辑地址空间中的地址为 A，页面的大小为 L，则页号 P 和页内地址 d 可按如下示公式求得：P=(int)[A/L]，d=A%L。

三、页表

在分页系统中，允许将进程的各个页离散地存储在内存不同的物理块中，但系统应能保证进程的正确运行，即能在内存中找到每个页面所对应的物理块。为此，系统又为每个进程建立了一张页面映像表，简称页表。在进程地址空间内的所有页(0～n)，依次在页表中有一页表项，其中记录了相应页在内存中对应的物理块号，见下图的中间部分。在配置了页表后，进程执行时，通过查找该表，即可找到每页在内存中的物理块号。可见，页表的作用是实现从页号到物理块号的地址映射。



6.2.2 请求分页式存储管理

请求分页系统是建立在基本分页的基础上的，为了能支持虚拟存储器功能而增加了请求调页功能和页面置换功能。相应地，每次调入和换出的基本单位都是长度固定的页面，这使得请求分页系统在实现上要比请求分段系统简单。请求分段系统在换进和换出时是可变长度的段，因此，请求分页便成为目前最常用的一种实现虚拟存储器的方式。

一、页表机制

在请求分页系统中所需要的主要数据结构是页表。其基本作用仍然是将用户空间中的逻辑地址变换为内存空间中的物理地址。由于只将应用程序的一部分调入内存，还有一部分仍在盘上，故需在页表中再增加若干项，供程序（数据）在换入、换出时参考。在请求分页系统中的每个页表项如下所示：

页号	物理块号	状态位 P	访问字段 A	修改位 M	外存地址

各字段的说明如下：

①状态位 P：用于指示该页是否已调入内存，供程序访问时参考。

②访问字段 A：用于记录本页在一段时间内被访问的次数，或记录本页最近已有多长时间未被访问，供选择换出页面时参考。

③修改位 M：表示该页在调入内存后是否被修改过。供置换页面时参考。由于内存中的每一页都在外存上有一份副本，因此，若未被修改，在置换该页时就不需要将该页写回到外存上，以减少系统的开销和启动磁盘的次数；若已被修改，则必须将该页重写到外存上，以保证外存中所保留的始终是最新副本。

④外存地址：用于指出该页在外存上的地址，通常是物理块号，供调入该页时参考。

二、缺页中断

在请求分页系统中，每当所要访问的页面不在内存中时，便产生一次缺页中断，请求 OS 将所缺之页调入内存。缺页中断作为中断，同样需要经历诸如保护 CPU 现场、分析中断原因、转入缺页中断处理程序进行处理、恢复 CPU 现场等几个步骤。但缺页中断又是一种特殊的中断，它与一般的中断相比，有着明显的区别，主要表现在下面两个方面：在指令执行期间产生和处理中断信号。

通常，CPU 都是在一条指令执行完成后，才检查是否有中断请求到达。若有，便去响应，否则，继续执行下一条指令。然而，缺页中断是在指令执行期间，发现所要访问的指令或数据不在内存时所产生和处理的。一条指令在执行期间，可能产生多次缺页中断。所以，系统中的硬件机构应能保存多次中断时的状态，并保证最后能返回到中断前产生缺页中断的指令处继续执行。

三、地址转换

请求分页系统中的地址变换机构，是在分页系统地址变换机构的基础上，为实现虚拟存储器而增加了某些功能而形成的，如产生和处理缺页中断，以及从内存中换出一页的功能等等。 在进行地址变换时，首先去检索快表，试图从中找出所要访问的页。若找到，便修改页表项中的访问位。对于写指令，还需将修改位置成“1”，然后利用页表项中给出的物理块号和页内地址形成物理地址。地址变换过程到此结束。如果在快表中未找到该页的页表项时，应到内存中去查找页表，再根据找到的页表项中的状态位 P，了解该页是否已调入内存。若该页已调入内存，这时应将此页的页表项写入快表，当快表已满时，应先调出按某种算法所确定的页的页表项；然后再写入该页的页表项。若该页尚未调入内存，这时应产生缺页中断，请求 OS 从外存把该页调入内存。

6.3 实验内容

在地址映射过程中，若在页面中发现所要访问的页面不在内存中，则产生缺

页中断。当发生缺页中断时，如果操作系统内存中没有空闲页面，则操作系统必须在内存选择一个页面将其移出内存，以便为即将调入的页面让出空间。我们详细介绍先来先服务、最近最少使用法和最佳页面替换法。

首先我们通过随机数产生一个指令序列，共 maxn 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：（produce_inst()函数）

a)在[0，maxn-1]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 m1；

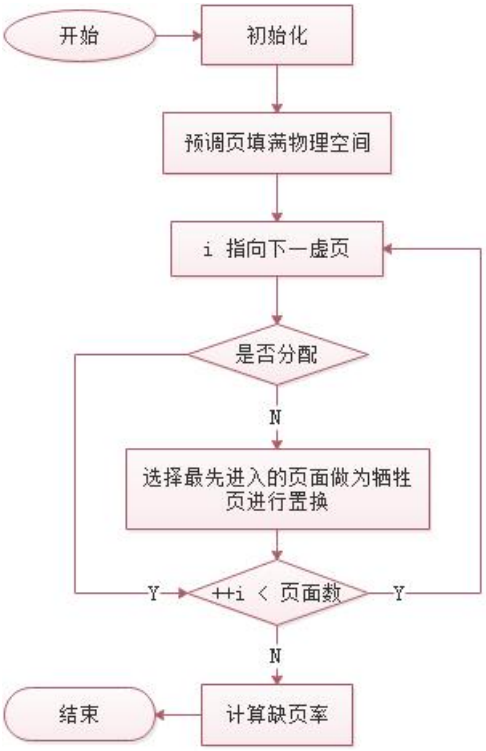
d)顺序执行一条指令，其地址为 m1+1；

e)在后地址[m1+2，maxn-1]中随机选取一条指令并执行；

f)重复上述步骤 a)-f)，直到执行 maxn 次指令。

现在请打开 pagemmuhard.c 文件。现在需要你补全函数并编译运行，验证函数的正确性。

一、先来先服务（FIFO）



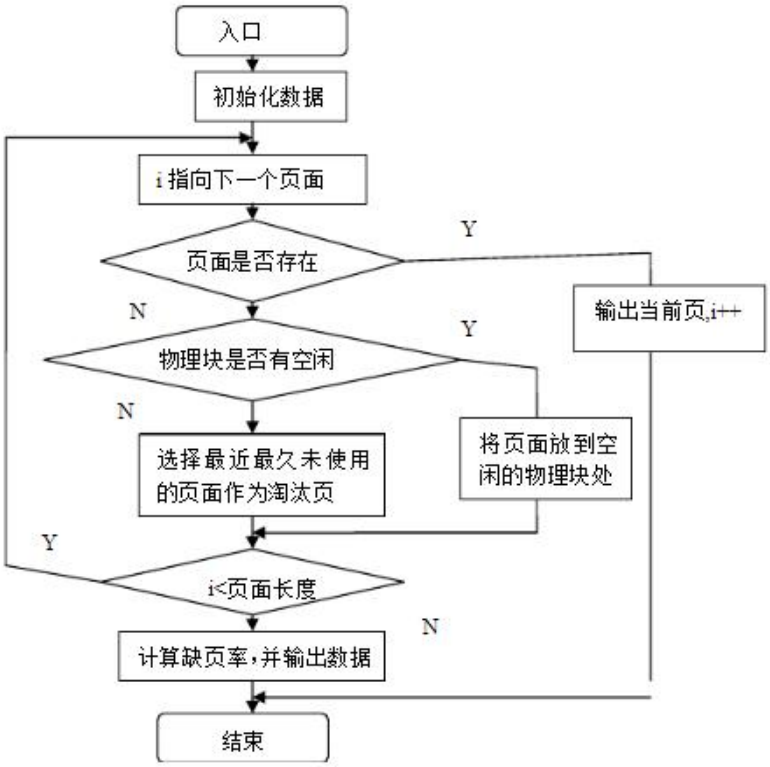
置换最先调入内存的页面，即置换在内存中驻留时间最久的页面。按照进入内存的先后次序排列成队列，从队尾进入，从队首删除。但是该算法会淘汰经常访问的页面，不适应进程实际运行的规律，目前已经很少使用。流程图如左所示。

具体实现过程为：定义变量 ptr。一开始先预调页填满内存。在这一部分，ptr 指向下一个要存放的位置。之后继续执行剩下的指令。此时，ptr 表示队列最前面的位置，即最先进来的位置，也就是下一个要被替换的位置。然后接着对预调页之后的页面进行判断处理，如果未分配，则按算法要求进行置换。结合 ptr，可采用环形队列方式管理。

二、最近最少使用法（LRU）

置换最近一段时间以来最长时间未访问过的页面。根据程序局部性原理，刚被访问的页面，可能马上又要被访问；而较长时间内没有被访问的页面，可能最

近不会被访问。 流程图如下。

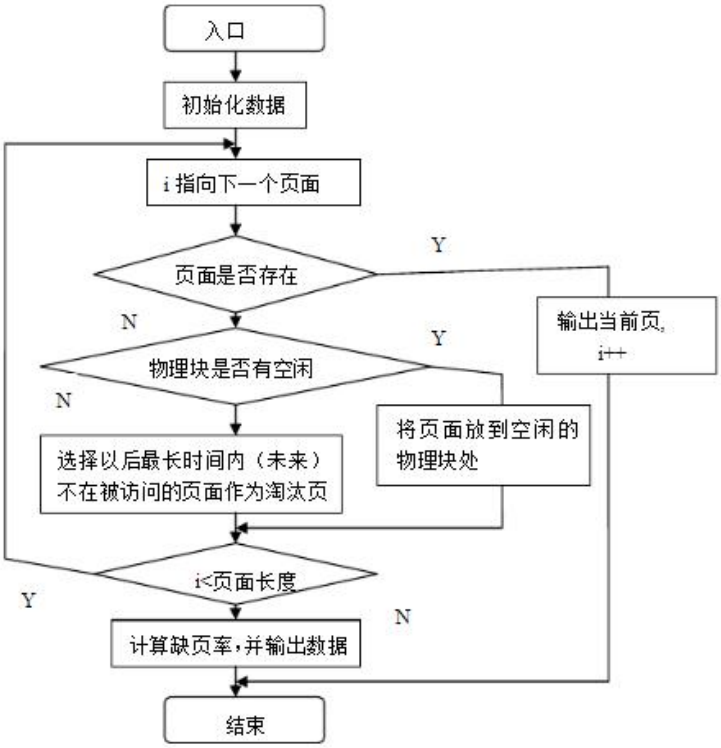


LRU 算法普偏地适用于各种类型的程序，但是系统要时时刻刻对各页的访问历史情况加以记录和更新，开销太大，因此 LRU 算法必须要有硬件的支持。

具体实现如下：定义数组 ltu[],即 last_time_use 来记录该页最近被使用的时间。定义变量 ti 模拟时间的变化，每执行一次加一。本算法实现未预调页，而是直接执行所有指令。若当前需要的页没在内存里，就寻找最近最少使用的页，也就是 ltu[]最小的页，即最近一次使用时间离现在最久的页，然后替换掉它。或者在内存还未满时，直接写入，这个以初始化内存里所有页为-1 来实现。若已经在内存里了，则只遍历内存内的页，把当前页的最近使用时间改一下即可。

三、最佳页面替换法（OPT）

置换以后不再被访问，或者在将来最迟才会被访问的页面，缺页中断率最低。但是该算法需要依据以后各页的使用情况，而当一个进程还未运行完成时，很难估计哪一个页面是以后不再使用或在最长时间以后才会用到的页面。所以该算法在实际情况中几乎是不能实现的。但该算法仍然有意义，作为衡量其他算法优劣的一个标准。流程图如下。



具体实现如下：定义数组 ntu[]，即 next_time_use 来记录下一次被使用的时间，即将来最快使用时间，初始化为-1。开始时预调页填满内存里的页。同样利用变量 ptr 来表示下一个要存放的位置从而控制预调页的过程。接着初始化 ntu 数组为-1。然后求出每一页下一次被使用的指令序号，以此代替使用时间。如果所有剩下的序列都没有用该页时，则还是-1，这种值为-1 的页显然是最佳替换对象。然后执行所有剩下的指令。当该页不在内存里时，遍历 ntu 数组，遇到-1 的直接使用该页，没有则用 ntu[]值最大的，也就是最晚使用的。无论该页在不在内存里，因为这一次已经被使用了，所以都应该更新这个页的 ntu[]，只需往前看要执行的页流，记录下第一个遇到的该页即可。如果没有找到同样添-1 即可。