Broadview®
www.broadview.com.cn

操作系统实验教程

陆松年 主编
訾小超 潘理 龚玲 编著

立足实用·操作性强·覆盖面广

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

操作系统实验教程

陆松年　主编
訾小超　潘理　龚玲　编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本实验教材除了具有传统的操作系统实验基本特色外，还具有通信工程、信息安全和电子工程等非计算机专业特色，并且实验覆盖面很广、实验内容特别丰富。

本书共分 3 部分：第 1 部分是实验相关原理；第 2 部分是实验指导书，共设计了涵盖操作系统绝大部分原理和应用的 28 个实验，可作为学生的上机实验或课程设计；第 3 部分是部分实验参考解答，给出了第 2 部分中的 16 个实验参考答案，其余的实验是希望学生全部独立完成，这些实验也可作为教师布置上机的实验题目。

本书可作为高等院校计算机科学与应用专业，以及通信与信息工程、电子工程、信息安全、自动控制和信息管理类等非计算机专业的实验教材和实验教学参考书，对于计算机软件开发人员也是一本很好的参考书。

图书在版编目（CIP）数据

操作系统实验教程 / 陆松年主编；訾小超，潘理，龚玲编著. —北京：电子工业出版社，2010.3
ISBN 978-7-121-10564-7

Ⅰ. ①操… Ⅱ. ①陆… ②訾… ③潘… ④龚… Ⅲ. ①操作系统－高等学校－教材 Ⅳ. ①TP316

中国版本图书馆 CIP 数据核字（2010）第 048547 号

责任编辑：孙学瑛
印　　刷：北京天宇星印刷厂
装　　订：三河市鹏成印业有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×1092　1/16　印张：18.25　字数：424 千字
印　　次：2010 年 3 月第 1 次印刷
印　　数：4000 册　定价：29.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

操作系统不仅是理论性很强的课程，而且也是技术性和实践性很强的课程。过去在操作系统教学中，教师上课时在课堂上只是放空炮似地讲一些空洞的原理，学生感到学习内容很空洞，这形成了枯燥无味的“空对空”式的脱离实践的教学模式。学生在毕业设计时以及到了工作岗位上后，很多命令、工具不会用，系统程序设计能力很差，常常面对系统开发课题束手无策，很难较快地进入角色和完成研究开发任务。这种轻视实践和技术的思想都将导致学生在今后的工作中付出沉重的代价。

本教材的目的是通过操作系统实验或课程设计，加深学生对操作系统原理、系统与技术的理解和掌握，提高编制大型的系统和应用程序的能力，避免出现高分低能情况。

本实验教材在国内同类实验教材中，除了具有传统的操作系统实验基本特色外，还具有通信工程、信息安全和电子工程等非计算机专业特色，并且实验覆盖面很广、实验内容特别丰富、实用性也很强。

一般操作系统的实验教材较少涉及通信系统部分，而在另一门计算机通信课程中，一般也不结合有关操作系统内核的通信接口进行教学或实验，这使得学生即使学了这两门课程也不会编制与通信有关的系统程序。本实验教材在理论学习后接着安排有关的实验，要求学生编制进程控制、进程间数据通信，进程通信机制的实现，以及进行包括 TCP 和 UDP 的远程进程通信、远程过程调用实验，在此基础上完成远程计算机之间的文件传送 FTP 等功能。

信息安全是实践性很强的课程，同通信与信息工程相似，对掌握操作系统的开发与应用技术要求很高，为此，本教材设计了 Linux 内核模块和系统调用扩展等实验。本实验教材与当前信息安全需求相结合，设计了操作系统安全和安全访问控制实验，促进学生进行安全性增强方面的探索。

鉴于电子科学和自动控制专业与嵌入式系统联系越来越密切，本实验教材能使学生进一步掌握嵌入式操作系统的安装与引导技术。

本课程的实验设计安排分成难度级别不同的实验，供水平和要求不同的学生选择。本书可作为高等院校计算机科学与应用专业，以及通信与信息工程、信息安全、电子工程、自动控制和信息管理类等非计算机专业的实验教材及实验教学参考书，对于计算机软件开发人员也是一本很好的参考书。

全书共分 3 部分，第 1 部分是实验相关原理，讲述了与书中实验密切相关的操作系统，这一部分大多是从本书最后给出的参考文献[1]（《操作系统教程（第 3 版）》，陆松年主编，电子工业出版社出版）摘录整理而来的，这是考虑到使用本书的学生已经或正在学习操作系统，但不是使用参考文献[1]的教材，所以把与实验密切相关的内容整理在这里，其余部分可以使用其他通用的操作系统教材。当然，单单阅读这部分是远远不够的，学生还要掌握操作系统的基本原理才行。个别实验的原理部分可以直接阅读参考文献[1]的其他部分，并可以帮助进行实验设计与编程。

第 2 部分是实验指导书，共设计了涵盖操作系统绝大部分原理和应用的 28 个实验（包

括分实验)。一些实验一般学生在做了适当准备后可以在2～4个学时之内完成，但有些实验比较大，设计、编程和调试都比较复杂，可以作为学生的课程设计，所需学时数约为18～54学时。

第3部分是部分实验参考解答，给出了第2部分中的16个实验参考解答，这些参考解答全部经过了测试。本书其余的实验是希望学生全部独立完成，这些实验也可作为教师布置上机的实验题目。

为了使学生通过实验后，进一步理解与实验相关的原理和理论，我们在每个实验的最后部分给出了若干个思考题，学生在实验前先尝试独立完成这些思考题，必要时可以参考部分实验参考解答中的思考题答案。

在本书的编写过程中，得到了学校和院、系领导的大力支持。由于作者水平有限，实验内容尚需要不断更新，书中难免存在一些错误，恳切希望各位专家、学者和读者批评指教，作者将不胜感谢。

作　者
2010年3月
于上海交通大学
电子信息与电气工程学院 信息安全工程学院

目　录

第 1 部分　实验相关原理

第 2 部分 实验指导书

第 3 部分　部分实验参考解答

第 1 部分　实验相关原理

- 存储管理
- 进程管理
- 进程通信
- 文件系统
- UNIX 系统和网络程序设计
- Linux 的动态内核模块机制

第 1 章　存 储 管 理

1.1　可变分区存储管理

1.1.1　空闲存储区表

可变分区存储管理法并不像固定分区管理那样预先将内存划分成分区，而是等到作业运行需要内存时才向系统申请，从空闲的内存区中“挖”一块出来，其大小等于作业所需内存大小。管理空闲内存区的数据结构可采用链接法和连续线性表格法。下面介绍一下采用连续线性表格管理空闲内存的方法。如果采用链接法，就需要在表项中增加一个指向下一个空闲区的指针。

每一个空闲区用一个 map 结构管理：

```
struct map {
    unsigned m_size;
    char *m_addr;
};
struct map coremap[N];
```

m_size 是空闲区的长度，m_addr 是空闲区的起始地址。各个空闲区按起始地址由低到高的顺次登记在空闲存储区表中，m_size 为 0 的表项是空白表项，它们集中在表的后部。

在系统初始阶段，拥有一块很大的内存空白区，这时空闲存储区表 coremap 中只需有一项登记该空闲区即可，其余的 coremap 表项为空白项，即 m_size 皆为零。此时情况如图 1-1 所示。以后随着很多作业不断申请内存和释放内存，整个存储空间将出现许多大小不等的区域，存储区表和实际的内存空间就可能变成如图 1-2 所示的情况。

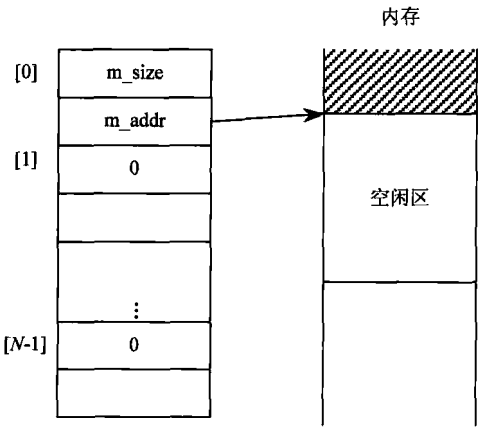

图 1-1　空闲区表的初始状态

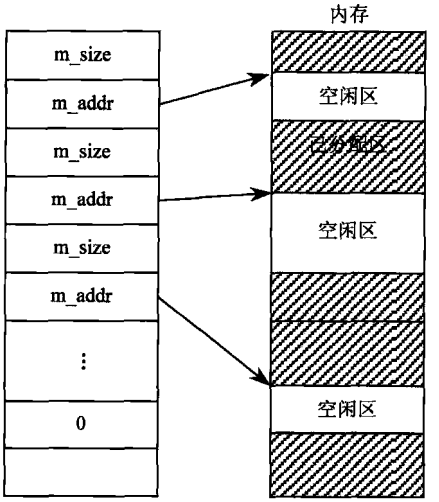

图 1-2　某一时刻空闲区表的状态

如有一个作业需调入主存运行，如何为其选择一个可用的存储分区？下面介绍 4 种分配策略，并对第一种分配策略进行详细介绍。

1.1.2　首次适应法

1．分配算法

采用首次适应法为作业分配大小为 size 的内存空间时，总是从表的起始端的低地址部分开始查找，当第一次找到大于或等于所申请大小的空闲区时，就按所需大小分配给作业。如果分配后原空闲区还有剩余空间，就修改原存储区表项的 m_size 和 m_addr，使它记录余下的“零头”。如果作业所需空间正好等于该空闲区大小，那么该空闲区表项的 m_size 就成为 0，接下来要删除表中这个“空洞”，即将随后的各非零表项依次上移一个位置。

下面用 C 语言给出首次适应法的算法。

```
char *malloc(mp,size)
struct map *mp;
unsigned size;
{
    register char *a;
    register struct map *bp;

    for (bp = mp; bp->m_size; bp++) {
      if(bp->m_size >= size) {
        a = bp->m_addr;
        bp->m_addr += size;
        if((bp->m_size -= size) == 0)
          do   {
              bp++;
              (bp-1)->m_addr = bp->m_addr;
          } while((bp-1)->m_size = bp->m_size);
        return(a);
      }
    }
    return(0);
}
```

2．回收算法

当某一作业释放以前所分配到的内存时，就要将该内存区归还给系统，使其成为空闲区可被其他作业使用。回收时如果释放区与邻近的空闲区相衔接，则要将它们合并成较大的空闲区，否则空闲区将被分割得越来越小，最终导致不能利用；另外，空闲区个数越来越多，也会使空闲区登记表溢出。释放区与原空闲区的相邻情况可归纳为如图 1-3 所示的 4 种情况。下面讨论每一种情况的合并方法。

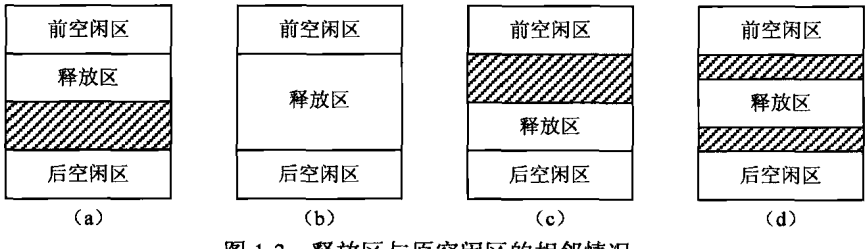

图 1-3　释放区与原空闲区的相邻情况

① 仅与前空闲区相连：合并前空闲区和释放区构成一块大的新空闲区，并修改空闲区表项。该空闲区的 m_addr 不变，仍为原前空闲区的首地址，修改表项的长度域 m_size 为原 m_size 与释放区长度之和。

② 与前空闲区和后空闲区都相连：将三块空闲区合并成一块空闲区。修改空闲区表中前空闲区表项，其起始地址 m_addr 仍为原前空闲区起始地址，其大小 m_size 等于三个空闲区长度之和，这块大的空闲区由前空闲区表项登记。接下来还要在空闲区表中删除后项，方法是将后项以下的非空白表项顺次上移一个位置。

③ 仅与后空闲区相连：与后空闲区合并，使后空闲区表项的 m_addr 为释放区的起始地址，m_size 为释放区与后空闲区的长度之和。

④ 与前、后空闲区皆不相连：在前、后空闲区表项中间插入一个新的表项，其 m_addr 为释放区的起始地址，m_size 为释放区的长度。为此，先要将后项及以下表项都下移一个位置。

下面给出实现该释放算法的 C 函数 mfree(size,aa)，该函数按释放区长度 size 和起始地址 aa 对空闲区表进行相应处理。

```
mfree(size,aa)
unsigned size;
char *aa;
{
    struct map *bp;
    char *a, *t;
    unsigned tt;

    a = aa;
    for (bp=coremap; bp->m_addr<=a && bp->m_size!=0; bp++);
    if (bp>coremap && (bp-1)->m_addr+(bp-1)->m_size == a) {   /* 情况 1,2 */
        (bp-1)->m_size += size;                    /* 情况 1 */
        if (a+size == bp->m_addr) {                /* 情况 2 */
            (bp-1)->m_size += bp->m_size;
            while (bp->m_size) {
                bp++;
                (bp-1)->m_addr = bp->m_addr;
                (bp-1)->m_size = bp->m_size;
            }
        }
    }
    else {
        if (a+size == bp->m_addr && bp->m_size) {          /* 情况 3 */
            bp->m_addr -= size;
            bp->m_size += size;
        }
        else                                       /* 情况 4 */
            if (size)
```

```
                do {
                    t = bp->m_addr;
                    bp->m_addr = a;
                    a = t;
                    tt = bp->m_size;
                    bp->m_size = size;
                    bp++;
                } while (size = tt);
        }
    }
```

若采用首次适应法，则其分配算法简单且合并相邻空闲区也很容易。该算法的缺点是由于查找总是从表首开始，前面的空闲区往往被分割得很小，能满足分配要求的可能性较小，查找次数就较多。系统中作业越多，这个问题就越严重。针对这个问题，对首次适应法稍加改进，就有了循环首次适应法。

1.1.3　循环首次适应法

把空闲区表设计成顺序结构或链接结构的循环队列，各空闲区仍按地址从低到高的顺序登记在空闲区的管理队列中，同时需要设置一个起始查找指针，指向循环队列中的一个空闲区表项。

循环首次适应法分配时总是从起始查找指针所指的表项开始查找，第一次找到满足要求的空闲区时，就分配所需大小的空闲区，修改表项，并调整起始查找指针，使其指向队列中被分配的后面的那块空闲区。下次分配时就从新指向的那块空闲区开始查找。

循环首次适应法的实质是起始查找指针所指的空闲区和其后的空闲区群常为较长时间未被分割过的空闲区，它们已合并成为大的空闲区的可能性较大，比起首次适应法，在没有增加多少代价的情况下却明显地提高了查找分配速度。

释放算法基本同首次适应法一样。释放时当需要在顺序方法实现的空闲队列中插入一个表项或删除一个表项时，根据该表项与起始查找指针之间的相对位置，有可能需要修改指针值，使其仍旧指向原空闲表项。

1.1.4　最佳适应算法

所谓最佳适应算法就是在所有大于或等于要求分配长度的空闲区中挑选一个最小的分区，即该分区对所要求分配的大小来说是最适合的。在分割后，剩余的存储区最小或等于零。因此，该算法的空闲区利用率高，较大的空闲区被保存下来。

空闲存储区管理表的组织方法可以采用顺序结构，也可以采用链接结构。如果采用顺序结构，空闲区按地址由小到大的顺序登记在表中，分配时需要搜索所有的空闲区，以在其中挑出一个满足分配大小的最小分区。此种管理结构的释放算法可采用顺序结构的首次适应法。

当采用链接结构时，空闲区也可按由小到大的非递减顺序排列。分配时总是从最小的第一项开始，这样第一次找到的满足条件的空闲区必定是最合适的。采用链接表，由于空闲区是按大小而不是按地址排序的，因此释放回收空闲区时要在整个链表上寻找地址相邻的前、后空闲区，合并后又要插入到合适的位置，因此释放算法比前两种算法耗时得多。

1.1.5 最差适应法

最差适应法所分割的空闲存储区是所有空闲区中的最大一块。实现最差适应法时的空闲存储区表的组织方法一般都是采用按空闲块由大到小排序的链接表，因为如果采用按地址大小排序的顺序结构，那么该算法与首次适应法和最佳适应法比较起来就没有什么优点可言了。采用按存储区大小顺序排列的链接表的形式虽然释放一个空闲块和合并时速度较慢，但分配效率是一切算法中最高的一种。

1.2 分页存储管理

1.2.1 分页存储管理的基本思想

页式存储管理的基本思想是把作业的虚拟地址空间划分成若干长度相等的页（page），也称虚页，每一个作业的虚页都从 0 开始编号。主存也划分成若干与虚页长度相等的页架（frame），也称页框或实页，主存的页架也从 0 开始编号。程序装入时，每个虚页装到主存中的一个页架中，这些页架可以是不连续的。

页式存储管理又可分为纯页式（静态页式）存储管理和请求分页式（动态页式）存储管理两种。在静态页式管理系统中，要求一个作业在运行前将其所有的虚页都装入主存的页架中，这就要求当时主存中有足够多的空闲页架，否则作业就不能运行。

请求分页式系统在运行一个作业时，不必将所有的虚页都装入主存的页架中，只需要装入当前运行时所必须访问的若干页，其余的虚页仍驻留在辅存中。等到运行到某一时刻需要访问这些虚页时，再将它们调入主存的空闲页架中。有时将请求分页式系统作业运行时所占用的页称为页面。在请求分页式系统中，用户的编程空间就不受系统主存大小的限制，能运行一个虚拟地址空间远大于实际主存空间的程序。

1.2.2 地址变换

在分页系统中，页的大小都是 2 的整数次幂，如 1024、2048 或 4096 等。这样，分页系统的地址结构可表示为：

P（页号）	d（页内偏移量）

虚拟地址字的页内偏移部分占据低 n 个二进制位，使 2^n 刚好等于页的大小，这样安排便于硬件直接截取高位部分的页号和低位部分的页内偏移值。

程序的虚拟地址空间是连续的，但程序的虚页可以分配到主存中不连续的空闲页架中，故分页系统需要在主存中开辟一个页表区域，来建立每一个作业的虚页号到内存的页架号之间的映射关系。系统还需要建立一个总页表来记录各个作业页表的起始地址和长度，并将当前运行进程的页表起始地址和长度存放在页表控制寄存器中。纯分页系统的地址变换机构如图 1-4 所示。

在程序执行时，对于每一条访问内存的指令，纯分页系统的地址变换过程如下。

① 由硬件自动将虚拟地址字分成虚页号和页内偏移两部分。

② 由页表控制寄存器中页表起始地址和虚拟地址字中的虚页号查找页表，对应虚页号的页表表项地址为：页表起始地址+虚页号*页表表项长度。

③ 将页表中取出的页架号和虚拟地址字中的页内偏移值装入地址寄存器，根据地址寄存器中的地址值访问主存。

以上整个地址变换都由硬件自动完成。

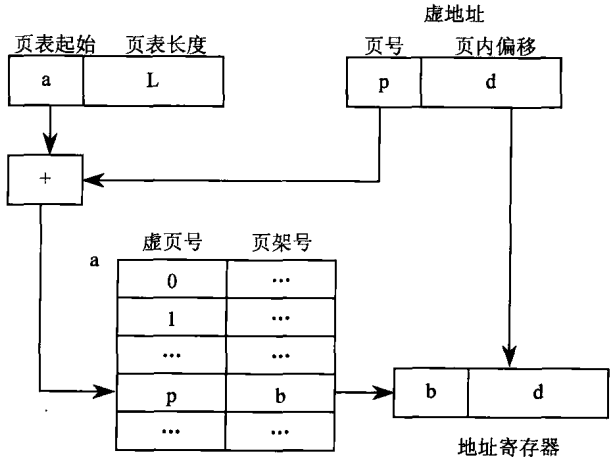

图 1-4　纯分页系统的地址变换机构

1.2.3　空闲内存页的管理

在页式管理系统中，需要一张表记录主存中每个页的使用情况和当前空闲页的总数。由于主存页总数很多，且页的大小相等，故可用位图描述各页的状态。在位图中用一个二进制位对应于主存中的一个页，该位为 0 表示对应页空闲，为 1 表示对应页已分配。位图中每一个字节可表示 8 页的状态，如设页面大小为 4KB，对于 256MB 的主存空间，只需要 8KB 大小的位表。

在分配空闲页时，可一次取出位图中的一个字，如该字内容非全 1，即表示其中必对应有空闲页，接下来就可确定空闲页的位置。为了提高在位图上检索空闲页的速度，可在表头增设一个起始查找位置指针，位图中在起始查找指针之前的所有字（或字节）都不含有空闲位。另外，设置一个循环查找指针的算法也不错。

释放算法很简单，只要在位图中的对应位上置 0 即可，但如果设置起始查找指针，则可能要修改指针值。

也可使用顺序或链接结构的栈来管理空闲页，栈中每一个单元指示一个空闲页。采用这种方法所需的存储空间比位图大许多，但分配空闲页时速度快很多。

1.2.4　请求分页式基本原理

采用请求分页式管理的基本思想是对于每一个运行作业，只装入当前运行所需要的一部分页面集合，该集合又称“工作集”。当作业运行时需要访问其他不在主存中的虚页时，硬件产生“缺页中断”，如主存资源紧张，可在原先装入主存的页面中选择一个或多个页，将其换出到辅存中，再把所需的页调入主存。请求分页式系统将主存和辅存这两级存储器融合成逻辑上统一的整体，故在这种系统中能运行比可用主存更大的作业或在相同容量的主存中并发运行更多的作业。

在请求分页式系统中控制这种能在主存和辅存间自动交换页面、对用户透明的机构称为虚拟存储系统的请求分页机构，该机构的主要功能如下。

① 执行地址变换操作，将程序虚拟地址转换为物理地址，这部分工作由硬件实施。

② 缺页时自动触发页面中断机构。缺页中断与普通的外设中断不同，一般中断只能在一条完整的指令执行结束后才能得到响应；而缺页中断可以发生在一条指令的执行中间，如果一条指令要访问多个页面（如对于间接访问指令和数据传送指令），还能引起多个缺页中断。

③ 缺页中断处理子程序，其中包括页面的调出和调入，这部分工作在软件控制下完成。

在请求分页式系统中，由于作业的虚页可能驻留在主存中，也可能驻留在辅存中，因此在页表中要扩充表项的内容，使之包含辅存地址，以便在发生缺页时请求分页机构可以将辅存中的虚页调入主存。这样，每一个页表项结构为：虚页号、内存页架号、辅存地址和状态。

其中状态字段含有该页当前是在主存还是在辅存的标志。此外，在该字段中还可包括访问位和修改位等。

1.2.5 页面淘汰

请求分页式系统中的程序在运行时，若发现某页的内容未被调入主存，就要通过缺页中断处理程序调入该页。如果这时主存中还有空闲的页架，那么只需要分配给调入页即可；但如果此时主存中所有页架都已分配出去，就只能从已占用的页架中挑选出一个页面，将其淘汰，腾出空页架以装入新页。应当淘汰哪一页呢？最好的策略就是淘汰那些今后不会再被访问或最长时间里不会被访问的页。但操作系统是难以预测程序执行的轨迹的，很有可能出现这样的情况，即刚刚将某一页面淘汰，随后又要访问该页面，从而使系统花费大量的时间用于页面在主存和辅存之间频繁地调入调出，大大降低了作业运行效率。这种现象称为页面“抖动”或“颠簸”。尽量避免页面抖动，是计算机科学家长期以来一直感兴趣的研究课题之一。下面讨论几种页面淘汰算法。

1. 最优淘汰算法（OPT）

程序执行时访问页的页号序列构成了页面流，最优淘汰算法就是淘汰那些从当前时刻起在页面流中不再出现的页，如果没有这类页，则淘汰一个在页面流中最晚出现的页。由于该算法最大限度地推迟了调出的页再调回主存的时间，显然可使页面调入调出的次数达到最小。尽管最优算法是十分诱人的，但由于系统无法预先知道一个作业未来访问页面的情况，故严格意义上的“最优”算法在实际上是无法实现的。不过，最优算法可以作为理论上的评价标准，用以鉴别其他淘汰算法的优劣。

2. 先进先出淘汰算法（FIFO）

这种算法思想很简单，总是淘汰最早调入主存的页面，因为一般可以认为，近期调入的页再次访问的可能性要比早期调入的页大。该算法也很容易实现，可采用一个先进先出的队列，新调入的页进入队尾，淘汰的页从队首取出。

FIFO 算法只在作业是按顺序访问地址空间的情况下才是理想的，否则效率不高，因为该算法很可能淘汰的是经常要访问的页，如调用频繁的子程序段、内循环部分、反复要处理的数组等。

3. 最近最少使用淘汰算法（LRU）

按字面意思，LRU（Least Recently Used）算法要比较最近一段时间里对各个页面的访问频率，淘汰访问频率最低的页面。这样的算法实现起来空间和时间的代价都比较大。

实际上，很多系统都将该算法实现为淘汰“最近一段时间内最久没有访问”过的页。其原理是根据很多事物和现象的延续性原理，在最近一段时间内不曾访问过的页在不久的将来

访问的可能性也较小。NUR（Not Used Recently）是一种低开销的近似于 LRU 的淘汰算法。一些教科书也把它归类为 LRU 算法。其主要思想是淘汰最近一段时间内未曾访问过的某一页面。该算法的实施不仅能考虑最近未曾访问过的页，还能优先挑选页面数据未曾修改过的页，这样可减少将淘汰页写回辅存的开销。

如果采用这种算法，要为每一项增设两个硬件位——访问位和修改位。当该页未访问过或没修改过时对应位为 0，反之为 1。初始时所有页的访问位和修改位都清 0。当访问某页时，该页的访问位置 1；当某页的数据被修改后，将该页的访问位和修改位都置 1。

系统将主存中各个使用页组织成一个循环队列，并设置一个循环检测指针。当需要淘汰一页时，从指针指向的页开始，顺次考察各页的使用情况，如果其访问位为 1，则将该位清 0；如果访问位为 0，但修改位为 1，则将修改位置 0（还应当更新辅存对应页）；不断重复这个过程，直到找到访问位和修改位都为 0 的页，将其作为淘汰页。按照这个算法，最新读过的页在第一轮的循环检测中不会被选中，最近写过的页在第一、二轮循环检测中不会被选中，这样不仅实现了 NUR 算法，而且给修改过的页两次机会。

LRU（或 NUR）的另一种实现方法是设置一个从中间可以出队的非严格意义上的队列，队列中的每一个单元对应于一个内存页，每次访问一个页面时将对应单元从队列中抽出，重新排到队尾去，淘汰的页从队列头取。这样，经常要访问的页总是排在靠近队列的尾部，而排在队首位置上的页就是最长时间没有被访问过的页。实现该算法的主要困难是，如果每执行一条访问主存的指令时都要执行队列的操作，并且该操作是用软件实现的话，其时间开销是不可忍受的。减少算法时间复杂性的方法之一是用硬件实现队列结构和操作，方法之二是每一条访问内存指令不必都执行队列操作，可以每过一个时间间隔（如时钟中断）执行一次。