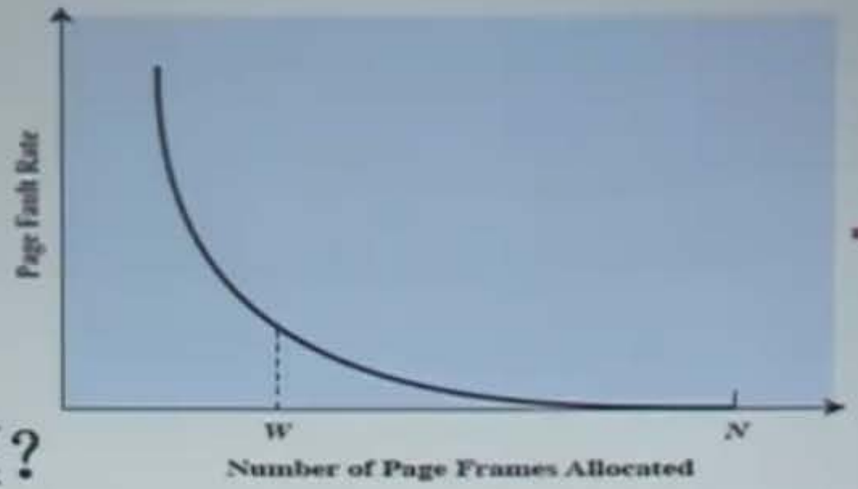
驻留集、置换范围、置换策略、清除策略

软件相关策略

那么下面呢我们就介绍，在虚拟页式存储管理方案当中的
操作系统部分，也就是软件的相关策略

驻留集



- 驻留集大小

  给每个进程分配多少页框？

- 固定分配策略

  进程创建时确定

  可以根据进程类型（交互、批处理、应用类）

或者基于程序员或系统管理员的需要来确定

- 可变分配策略

  根据缺页率评估进程局部性表现

✓ 缺页率高→增加页框数

✓ 缺页率低→减少页框数

✓ 系统开销

置换问题

- 置换范围

  计划置换页面的集合是局限在产生缺页中断的进程，还是所有进程的页框？

- 置换策略

  在计划置换的页框集合中，选择换出哪一个页框？

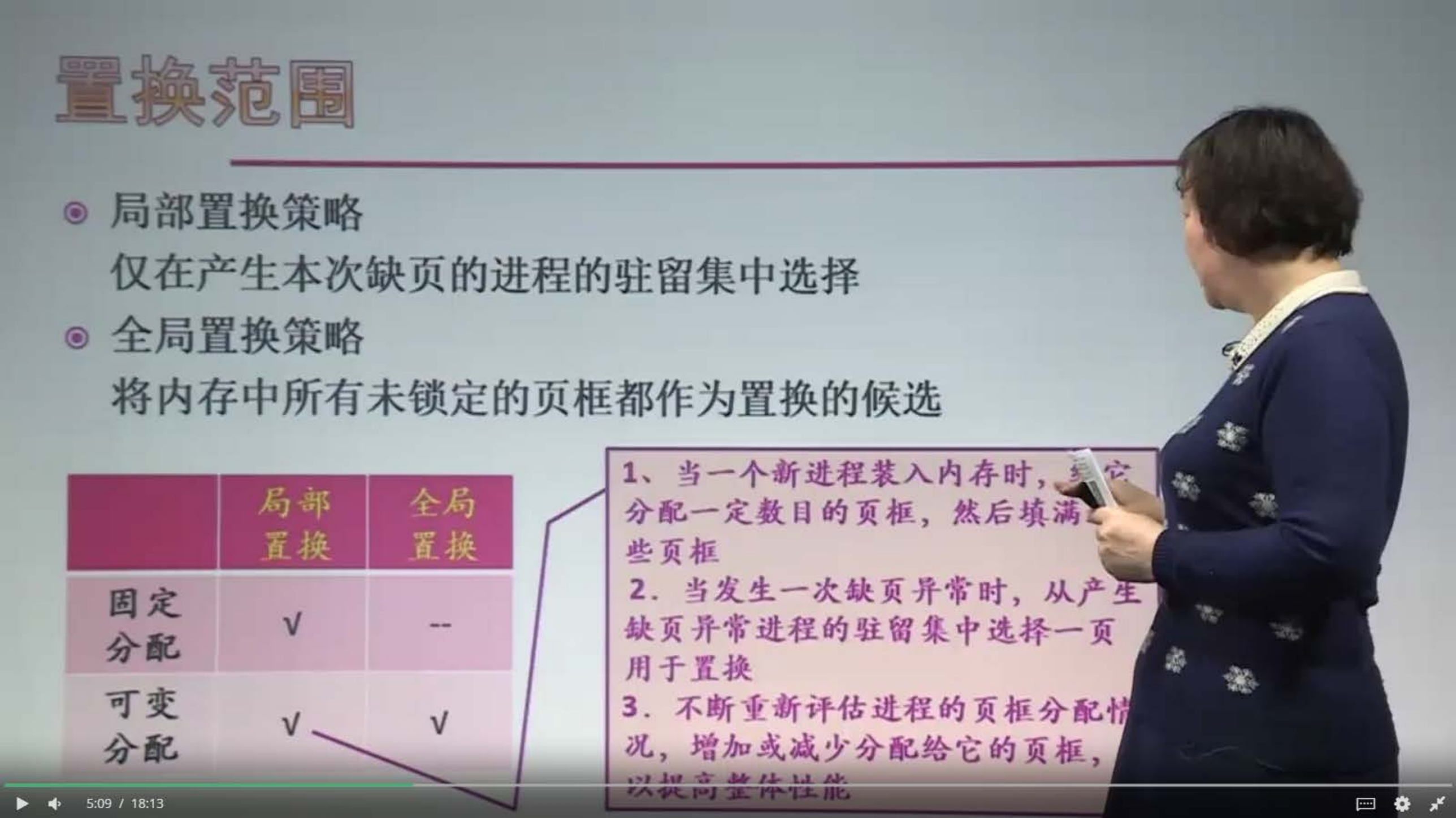
所以这是下面要考虑的策略问题

置换范围

- 局部置换策略
  仅在产生本次缺页的进程的驻留集中选择
- 全局置换策略
  将内存中所有未锁定的页框都作为置换的候选

	局部置换	全局置换
固定分配	√	--
可变分配	√	√

1、当一个新进程装入内存时，给它分配一定数目的页框，然后填满这些页框
2．当发生一次缺页异常时，从产生缺页异常进程的驻留集中选择一页用于置换
3．不断重新评估进程的页框分配情况，增加或减少分配给它的页框，以提高整体性能



置换策略

典型思路

- 决定置换当前内存中的哪一个页框
- 所有策略的目标

  → ***置换最近最不可能访问的页***

- 根据局部性原理，最近的访问历史和最近将要访问的模式间存在相关性，因此，大多数策略都***基于过去的行为来预测将来的行为***
- 注意：置换策略设计得越精致、越复杂，实现的软硬件开销就越大

- 约束：不能置换被锁定的页框

页框锁定

为什么要锁定页面？

- 采用虚存技术后

  开销 → 使进程运行时间变得不确定

- 给每一页框增加一个锁定位
- 通过设置相应的锁定位，不让操作系统将进程使用的页面换出内存，避免产生由交换过程带来的不确定的延迟
- 例如：操作系统核心代码、关键数据结构、I/O缓冲区

特别是正在I/O的内存页面

Windows中的VirtualLock和VirtualUnLock函数

清除策略(1/2)

- 清除：从进程的驻留集中收回页框
- 虚拟页式系统工作的最佳状态：发生缺页异常时，系统中有大量的空闲页框
- ***结论：在系统中保存一定数目的空闲页框供给比使用所有内存并在需要时搜索一个页框有更好的性能***

- 设计一个分页守护进程（paging daemon），多数时间睡眠着，可定期唤醒以检查内存的状态
- 如果空闲页框过少，分页守护进程通过预定的页面置换算法选择页面换出内存
- 如果页面装入内存后被修改过，则将它们写回磁盘
  分页守护进程可保证所有的空闲页框是“干净”的

清除策略(2/2)

- 当进程需要使用一个已置换出的页框时，如果该页框还没有被新的内容覆盖，将它从空闲页框集合中移出即可恢复该页面

页缓冲技术:

- 不丢弃置换出的页，将它们放入两个表之一：如果未被修改，则放到空闲页链表中，如果修改了，则放到修改页链表中
- 被修改的页定期写回磁盘(不是一次只写一个，大大减少I/O操作的数量，从而减少了磁盘访问时间)
- 被置换的页仍然保留在内存中，一旦进程又要访问该页，可以迅速将它加入该进程的驻留集合(代价很小)

0:00 那么下面呢我们就介绍，在虚拟页式存储管理方案当中的 操作系统部分，也就是软件的相关策略 首先我们来看一看一个概念，叫驻留集。 其实驻留集呢指的是 操作系统给每个进程分配多少个页框 通常情况下呢，我们有固定的分配策略和可变的分配策略两种 所谓固定分配策略呢，就指的是，在进程创建的时候就确定好给这个进程多少个页框 当然了，我们在给它分配页框数的时候呢 可以参考这个进程的类型，比如说它是交互式进程 批处理式进程，或者是它是一种应用类型的进程，或者是由 程序员，或者是系统管理员根据它的需要来确定 但是一旦确定了呢，就不要改变了。 那么所谓可变 分配策略呢，就指的是说我分配给一个进程的页框的数量不是 固定一成不变的，是要根据缺页率来评估进程的局部表现，然后做相应的调整 所谓缺页率的意思是说如果在这段时间 里头，那么这个进程的执行缺页率增加，提高了 那么达到了一个预值的话，那么我就要多给它一些页框 那么如果发现一个进程在最近这段时间的这个 执行过程中的缺页率比较低，那么我们可以 减少一部分页框，当然了就看系统的需要，如果系统页框 数不足的话呢，我们可以做这样一些过，过程。 当然这里头，我们要注意，它会带来新的开销 这就是当一个进程进入内存的时候我给它多少个页框比较合适 那这里有一张图，这张图其实呢是给出来了一个缺页率 和分配给进程多少个页框的页框数的这么一个 关系。 那我们可以看到，如果分配给进程的页框数很少 它的缺页率就会很高。 如果我给它更多的页框，它的缺页率就会下降 那如果我把它所需要的所有的都，都要调入内存了，那就相当于 都满足了，那么它就应该讲缺页率就应该是 0 当然了，我们不可能都给它，如果都给它的话呢，实际上呢我们也不符合我们的虚拟存储 这个技术的一个初衷，我们是要给它一部分，让它保持一部分的缺页率 通过这样能够达到我们前面所阐述的目的 这就是第一个概念驻留集 那么第二个就是关于置换问题，首先我们应该说 当一个内存已经用完了，没有 空闲页框了，那我们就需要挑一些页框，把它内容换出去，这就是所谓的置换 那么在置换的过程中，我们需要考虑它的置换的范围 和置换的策略。 所谓置换范围的话就指的是 说，我从哪一个页面的集合，页框的集合当中 来产生换出这个内存的这样的页框 是局限在产生缺页 异常的这个进程当中，还是在整个的系统范围？就是所有进程的页框 所以这是置换范围 那么置换的策略呢是指 你在一个可置换的页框的范围之内，我究竟选择哪一个页框置换出去 所以这是下面要考虑的策略问题 那我们先讨论置换的范围，置换范围的话当然了 我们刚才讲过了，有个局部置换策略和全局置换策略。 所谓局部 置换策略呢就是说产生缺页的这个进程本身，它有一个驻留集

我的置换呢就在这个驻留集当中进行挑选 所谓全局置换策略呢，就是在内存当中所有 没有锁定的页框都可以作为我的一个候选的置换对象 那么这就是两种置换策略 和我们前面所讲的分配策略相比，我们有 固定的页框的分配的策略和可变的页框分配策略，因此呢我们几个组合起来 可以得到三种策略方案 那么固定分配、 局部置换，可变分配、 局部置换 然后是可变分配、 全局置换。 我们以 可变分配、 局部置换为例进行简单的介绍 我们来看一下，当一个新的进程进入 内存了以后，那么我们给它分配一定数目的页框 当然了，我们就要把这些页框填满，当产生一次缺页异常的时候呢 我们就会去从产生缺页异常的这个进程当中的驻留集中，选择一页用于置换 那么我们可以不断地来评估 这个进程的一个页面，页框的分配的一个这个情况 如果缺页率增加了，那么我们可能会适当地增加分配给它的页框 如果缺页率降低了，那么我们就可以减少它的页框，这样的话呢，两个结合起来呢 提高整个系统的性能，性能 那么置换范围呢确定了之后，那我们就来看看 在这个范围当中我们究竟选择哪一个页框把它真正换出去，就要选择 页框的问题。 好，所以这就是我们的置换 策略，就是决定置换当前内存中的哪一个页框 那么我们来看一下 你用什么样的策略，它的目标都是统一的 也就是所有策略的目标都要达到这样一个目标 我选择的要置换出的页框是最近最不可能访问的页框 也就是在近一段时间里头，这个页框我不会访问到，最好是这样 因为根据程序的局部性原理 最近的访问的历史和最近 将要访问的这样一个模式之间呢，是存在着某种相关性的 我们经常会说我要根据你的过去来推断你的将来 我们要选拔人的时候，我们就要去面试他，我们要招聘，我们要面试 那我们要看他之前做了什么事情，之前掌握了什么样的知识 那么我们也是一样，最近的访问历史和最近将要访问的这个模式之间，具有这样一个相关性 因此我们大多数的策略都是基于过去的 行为，这个页面过去的行为来预测这个页面将来的行为 这是一个非常典型的思路。 但是呢 我们反过头来说，我们非常精心地来设置 设计这个置换的策略，设计得越精细 越复杂，那么它实现的过程中，带来的软硬件的开销呢就越大 所以呢我们要在这两者之间进行相应的折中权衡 这也就体现了你不可能只 完成某一个目标，那么还有另外一个，你也要考虑这个因素 但是不管怎么说，我们有这样一个约束 有一些页框是不能用于置换的 哪些页框呢？就是被锁定的页框 那么为什么要对页框进行锁定呢？ 我们来看一下，由于我们采用了虚存技术 可能我要访问的代码数据暂时还不在内存，我要把它临时地调入内存 那么

调入内存需要启动磁盘，需要修改内存的一些数据结构，因此就会带来开销 而这些开销一旦很多的话呢，就使得这个进程运行的时间变得不确定 因此我们需要说，如果我希望 这个进程运行时间相对稳定的话，那么我们就可以通过将 页框锁定的方式来达到我们的目的，我们可以给每一个页框增加一个锁定位 那么通过这样一个设置，把这个锁定位设置了 就不让操作系统将这个进程的这个页面换出内存 因为你换出内存就会带来由于交换带来的 开销，然后导致的一种不确定，不确定。 使它的 运行时间呢延后了，延后了。 所以我们可以通过这样一个设置来做到这一点 那么哪些页框需要锁定呢？比如说操作系统的核心代码 关键的一些数据结构，常用的关键的数据结构，还有一种就是 I/O 的缓冲区，正在进行 I/O 的 我们前面在讲紧缩技术的时候也谈到，正在 I/O 的这样一个页面不应该把它交换出去，应该把它锁定，这就是 这样一些页面我们要需要通过这种方式进行锁定。 那么在 Windows 当中呢会提供了一些 函数，允许用户对自己的一些页面进行相应的锁定。 用 VirtualLock 来锁定，用 VirtualUnlock 来解开。 当然了，不能对更多的页面进行锁定，它只给你提供一个 极少的一些页面你可以做一些锁定工作。 这是关于页面锁定。 那么下面我们来看一下清除策略。 所谓清除呢 指的就是从进程的驻留集当中把页框收回来，收回来。 那么其实置换其实就是选定一个页框，把它的内容交换到磁盘上， 然后呢换进新的一页。但是如果我把这个页框直接收回给系统呢，我们把它称之为叫清除。 我们前面所讲的当缺页以后， 操作系统才把相应的页面从磁盘读入内存， 那么这个时候如果内存中没有空闲页框了呢，再去 实施相应的置换。 但是呢经过了统计实验分析， 会发现虚拟页式系统它工作的最佳状态呢 是在发生了缺页异常的时候，系统中有足够的空闲页框。 也就是不会出现系统当中没有空闲页框的情况。 就不会出现弹尽粮绝的情况。 那么在这种情况下，始终保持 一定数量的空闲页框，使得这个我们的虚拟页式存储管理方案可以更好地去工作，这是它一个- 最佳的状态。 所以为了保持这个最佳的状态呢， 我们实际上呢就得出这样一个结论：我希望 在系统中始终保持一定数目的空闲页框， 那么这样做比 当所有的内存都用完了再去搜索一个页框，再去置换 那么性能要好得多。 这就是一个结论。 在系统中保持一定数量的空闲页框 提供要比使用所有的内存 并在需要的时候才去搜索页框性能更好。 因此 大部分的操作系统都会按这样一个思路来设计清除的策略。 也就是我们前面所讲的当出现了缺页异常以后，再去 寻找页框把它释放，那么这个工作呢

要提前来做了。 那么怎么做呢？我们来看一下。 那么一般系统都会设计这么一样一个进程，叫做 一个守护进程，叫分页守护进程，这是 Linux 一个做法。 那么这个进程大部分时间呢它是没事干的，它是处于睡眠状态， 定期地这个进程被唤醒。 那么唤醒以后它的工作首先去检查内存的状态， 如果发现在内存当中的空闲页框数过少， 有一个阈值，如果太少了，那么这个 进程就会通过预先设定好的 页面置换算法来执行这个算法，去选择 那些要换出内存的页框。 好，这就是 分页守护进程的主要工作。 好，这个工作做完之后，那么当系统中的页框数， 空闲页框数达到了一定的要求，那么这个守护进程就可以 睡眠去。好，那么反过头来看下面的问题。 一个页框它装入内存后如果 被修改了，那你选中它了呢就要把它们写回到磁盘。 所以分页守护进程呢要保证所选的所有的页框都是干干净净的，也就不是 dirty 的。 所以分页守护进程呢在选中页框之后呢，还要把相关的内容写回到磁盘。 那这就是一个典型的做法。 那么我们再强调一下 大部分系统的清除策略 会用这样一种方式来实现。 而不是到了 缺页以后再去寻找相应的页框，提前启动置换策略。 这是清除策略的我们介绍的它的基本思想。 但是呢我们会看到还可能会出现这样一种情况： 这个页框是某个进程的这个驻留集里头的页框， 那么由于分页守护进程根据相应的置换策略选中了这个页框，就 把这个页框从这个进程的驻留集当中收回来了。 收回来之后如果这个进程 还没有结束，没有结束，那么这个进程呢还要使用刚才 你收回去的页框的话，那么这个页框恰好 它的内容还是原来的内容，没有被覆盖掉， 在这种情况下系统呢会将它从 空闲页框的这个集合当中呢移出来，就可以恢复这个页面了。 那么这个进程就可以继续使用这相应的内容，而不需要再从磁盘上把相应内容再读入内存了。 这样的话呢就加快了这个进程的这个速度，执行速度。 这就是当采用清除策略之后，我们还可以利用 这样一个机制来解决收回的页框 再利用的问题，但注意这个进程并没有结束， 我们所讨论的这个清除策略是在这个进程运行过程中， 从它的驻留集当中收回页框。 那如果这个进程已经运行结束了，那所有的页框都还给系统了， 就不存在后面的这个情况出现了。 那么这种技术呢 我们有的地方称之为页缓冲的技术。 我们来简单介绍一下页缓冲技术的思想。 也就是不丢弃 置换出的页框，把它们可以放到两个表当中， 这是一种具体的实现的一个思想。 一个表呢就是没有修改过的， 你就把它放在空闲页的链表当中。 如果修改过的呢，你就放在修改页的链表当中。 那么放了一些 页框之后呢，我们可以再定期地 将

修改过的这些页面呢定期地写回到磁盘。不是一次写一个，而是一批来写。那主要的目的是为了减少 I/O 的操作的数量，减少访盘的时间，提高性能。这是后面也要去讲的，就怎么样提高性能。那这里头我们知道不用一次写，一次写一页，我们可以积攒出一部分修改过的页面，一次写回去。那么被置换的这样的页面呢其实还在 内存，它的内容呢还没有改变，所以一旦进程 再次要访问这个页面，就可以通过把它加入到这个 进程的驻留集当中，是一个迅速加入的方法，那么这时候的代价是最小的，代价是最小的。所以这是关于页缓冲技术的基本思想。