# 第4章 内存管理

这部分主要讨论Linux的内存管理特性，也就是虚拟内存和缓冲区。重点讨论系统管理员需要考虑的内存管理的目的、工作原理和具体操作。

## 4.1 何谓虚拟内存

Linux支持虚拟内存，也就是说，把磁盘当作扩充RAM使用，进而有效增大可用内存空间。内核将把目前未用的内存块中的内容写入磁盘，以便把内存用作其他用途。再次需要原来的内容时，再将它们写回内存。这一切对用户来说，是完全透明的；运行于 Linux系统下的程序只能看见还有大量的内存空间可以使用，而不会注意到有部分内容有时会被写入磁盘。当然，和使用真正的内存相比，读写磁盘的速度是比较慢的（甚至慢上千倍），所以程序的运行不会很快。被用作虚拟内存的那部分磁盘被称作“ 交换空间 ”（swap space）。

Linux即可采用文件系统内的普通文件，又可采用用于交换空间的独立分区。交换分区要快一些，但交换文件的大小更容易更改（没有必要对整个磁盘重新分区，一切都可从头安装）。在得知自己需要多少交换空间时，你应该选择交换分区，但如果你不确定自己需要多少交换空间，可先采用交换文件，暂时选用系统，以便了解自己需要多少交换空间，在确定空间大小之后，再安排交换分区。

此外，你还应该知道Linux允许一个用户同时使用若干个交换分区和／或交换文件。意思是如果你只是偶尔需要一个非常大的交换空间，可为此设置一个特殊的交换文件，而不是一直保留这样大的分配空间。

注意操作系统上的术语：计算机科学中，交换（把整个进程提出，转入交换空间）和页面调度（实现虚拟内存的一种技术。一次只写入固定大小的块，通常只有几 KB。一块称之为一页）之间是有区别的。一般说来，页面调度更为有效，它也是 Linux常采用的方法，但过去的Linux操作系统总称之为交换。

## 4.2 创建交换空间

交换文件是一个普通文件；对内核来说，它没有什么特别之处。对内核而言，唯一值得注意的是，它没有漏洞，而且是为 mkswap的使用而准备的。但是，该文件必须驻留在本地磁盘上；如果由于实施时的具体考虑，文件系统已经通过 NFS得以装入，那么，它就不能存在于这样的文件系统中。

交换文件中绝对不能出现漏洞（即不连续地存储）。交换文件保留了磁盘空间，所以在为文件分配一个磁盘扇区时，内核可以快速交换出一个页，无须重写整个交换文件。内核只采用已经分配给该文件的扇区。由于文件中的漏洞意味着那里没有分配相应的磁盘扇区，所以这将为内核带来不便。

要创建一个没有漏洞的交换文件，最好通过下面的命令来进行：

```
$ dd if=/dev/zero of=/extra-swap bs=1024 count=1024
```

```
1024+0 records in
1024+0 records out
$
```

/extra-swap指的是交换文件名，count=后面则是为其指定的文件大小。这个值最好是4的倍数，因为内核交出的内存页的大小是4K。如果该值不是4的倍数，最后2KB就不能得以采用。

交换分区也没有任何特别之处。你可像对待其他的分区一样，创建它；唯一的区别是它被用作原始分区，也就是说，交换分区内根本不包含任何文件系统。把交换分区标记为82类型（Linux交换）倒是个好办法；这样一来，分区清单将更为清楚（虽然内核并没有这方面的要求）。

在已创建交换文件或交换分区之后，需要在其开始之处，写上签名；签名中应该包含一些管理信息，而且这个签名供内核采用。执行此项任务的命令是\cmd{mkswap}，其用法如下：

```
$ mkswap /extra-swap 1024
Setting up swapspace, size = 1044480 bytes
$
```

注意，交换空间还没有开始使用：它是存在的，但内核还没有用它来提供虚拟内存。

在使用mkswap时，应该非常小心，因为它不会对没有用的交换文件或分区进行检查。利用mkswap，将轻易地改写重要文件或分区！遗憾的是，在你安装系统的时候，必须采用mkswap。

Linux内存管理器将每个交换分区的大小限制在127KB左右（由于技术上的原因，实际上的限制是(4096-10)×8×4096=133890048$字节或127.6875MB）。但是，可同时使用16个交换分区，其总字节数可达2GB（很快，我们就要谈到真正的内存了）。

## 4.3 交换空间的使用

利用swapon，就可开始使用一个经过初始化的交换空间了。这个命令向内核报告可使用的交换空间。到该交换空间的路径作为参数给出，所以对一个临时交换文件开始执行交换时，将采用下面的命令：

```
$ swapon /extra-swap
$
```

在/etc/fstab文件内列出需要的交换空间，就可以自动使用这些交换空间了。

```
/dev/hda8    none   swap   sw 0 0
/swapfile    none   swap   sw 0 0
```

startup（开始）脚本将开始运行swapon -a命令，该命令将开始对/etc/fstab内列出的所有交换空间执行交换。因此，swapon命令通常只有在需要额外交换空间时才使用。

可以用free命令来监视交换空间的使用情况。它会告诉我们总共使用了多大的交换空间。

```
$ free
             total       used       free     shared    buffers
Mem:         15152      14896        256      12404       2528
-/+ buffers:            12368       2784
Swap:        32452       6684      25768
$
```

输出的第一行（Mem:）显示出物理内存的容量。 total列并没有显示出由内核使用的物理内存量，但这通常都在1MB左右。used列显示出已经使用的内存容量（第二行没有把缓冲考虑在内）。free列显示的是完全没有使用的内存容量。 shared列显示出由几个进程共享的内存量：数字越大越好。buffers列显示出当前的磁盘高速缓存容量有多大。

最后一行（Swap:）显示交换空间的一些类似信息。假如这一行全为零，表明你的交换空间尚未激活。

同样的信息也可以通过top命令得到，或使用文件/proc/meminfo内的proc文件系统来得到。目前，很难获得与一个特定交换空间的使用有关的信息。

为了删除一个交换空间，可使用swapoff命令。但通常都没必要这样做，除那些临时交换空间外。交换空间内使用的每个页都会首先得以交换；假如没有足够多的物理内存来容纳它们，它们就会被“交换出去”（到其他的一些交换空间）。假如没有足够多的虚拟内存来容纳全部页，Linux就会出现混乱；经过一段时间，这种混乱状况也许会结束，但在此期间，整个系统是完全无法使用的。因此，删除一个正在使用的交换空间之前，首先应检查是否有足够多的剩余内存（比如用free命令来检查）。

通过swapon -a命令自动使用的所有交换空间都可以用 swapoff -a命令来删除；它会在/etc/fstab文件中检查，判断自己该删除哪些交换空间。注意，任何人工使用的交换空间仍然会继续保持使用状态。

在某些情况下，即使有大量空闲的物理内存，也可能会使用大量的交换空间。例如，假如某个时候需要交换，但在过后的某个时候，一个占用了大量物理内存的进程终止运行，并释放出它占用的所有内存，就会出现这样的情况。交换出去的数据不会自动交换回来，除非有这方面的需要。因此，物理内存可能会长时间保持闲置状态。尽管你不必过多操心这方面的问题，但搞清楚其中的原理，仍然是很有必要的。

## 4.4 与其他操作系统共享交换空间

许多操作系统都内建了虚拟内存机制。由于只有在操作系统运行时，才需要使用虚拟内存，所以，除了当前正在运行的操作系统之外，其他操作系统的交换空间都被浪费了。因此，一种更有效的策略是让它们共享一个独立的交换空间。尽管这种方法理论上是可行的，但事实上需要一些技巧。Tips-HOWTO内包含相关的实用性建议。

## 4.5 交换空间的分配

有些人会告诉你，交换空间的大小应该是物理内存的两倍，但这种说法未免太空泛，正确作法是：

1) 首先估计需要的内存容量。这应该是一次可能需要的最大内存容量，它是你打算同时运行的所有程序需要的内存量的总和。要想计算出这一数字，请实际运行你打算同时使用的全部程序。

例如，如果你想运行X，就应该为其分配8MB，gcc需要为它分配数MB的空间（有些文件甚至需要更多的内存，多达数十 MB），等等。注意，内核本身需要占用约 1MB的内存空间，普通外壳和其他小工具各自需要占用几百 KB的空间。注意，这个数字无须特别精确，估计一下就行了，但宁多勿少。

要记住这一点：如果几个人打算同时使用系统，那么他们都会消耗内存。但是，如果两个人同时运行相同的程序，总共消耗的内存却用不着加一倍，这是由于代码页和共享库只存在着一个副本实例。

free和ps命令特别适合用来估算你需要的内存容量。

2) 根据步骤1算出的结果，我们再增加一些保险系数。这是由于你可能会忘记自己需要运行的某些程序，造成前面的估算有误。这样做也是为了保证你最终有足够的额外空间供你调用。这个保险系数不能太大， 1或2兆就够了（交换空间只能多，不能少，但也不必太多，该不会把整个磁盘都用作交换空间吧！那些闲置的交换空间最终还是被浪费掉了；参考后文，了解如何增加额外的交换空间）。除此以外，注意尽可能采用整数，将你估算出来的值四舍五入，换算成一个整数的MB。

3) 根据上面的计算结果，你已经知道了自己总共需要多少内存空间。所以，为了分配交换空间，你只需从这个所需空间总数内减去物理内存的大小即可，这样就能得出自己需要的交换空间啦（有些Linux版本中，还需要为物理内存映像分配空间，所以步骤 2中计算出来的结果就是你所需的交换空间，不应该减去物理内存大小）。

4) 如果你计算出来的交换空间比自己的物理空间大得多（甚至超过两倍），可能应该分配更多的物理内存，不然，系统性能就会有所损失。

至少得保留交换空间总是错不了，即使你的计算表明你根本不需要它。 Linux会主动采用交换空间，所以可让你的物理内存尽可能地空闲出来。 Linux将交换出没有使用的内存页，即使程序并不需要内存。这样一来，需要内存时，就可不必再等待交换了。交换的工作早在之前的磁盘空闲时进行了。

同一个交换空间可跨越几个磁盘。根据磁盘速度和访问磁盘的方式，这样做有时还可改进性能。你可能想尝试几种方案，但一定要知道各种方案是相当不同的。不要相信某某人声称某方案是最好的，因为这类话通常都不可信。

## 4.6 缓冲区

和访问真正的内存相比，从磁盘上读取数据显然非常之慢（显然，RAM磁盘除外）。另外，在相当短的时间内，多次读取磁盘上的同一部分是常有的事。例如，有人可能会先读取电子邮件消息，然后在答复这封邮件时，把它读入编辑器，再在把它复制到文件夹时，令电子邮件程序再次读取它。或者，以ls命令为例，这个命令非常频繁地运行于一个多用户系统上。只从磁盘上读取一次信息，然后把它保存在内存内，直到不再需要它为止，通过这种方式，除了第一次读取较慢以外，后面的运行速度明显得以加快。这种技术叫作“ 磁盘缓冲 ”，用于该目的的内存叫作“ 缓冲区。”

遗憾的是，由于内存是一个有限的、宝贵的资源，缓冲区通常不可能非常之大（它容纳不下人们想使用的所有数据）。缓冲区充满时，长时间内没有使用的数据就会被丢弃，从而为新数据释放内存空间。

磁盘缓冲对写入数据来说，非常有用。一方面，写入的数据很快又会被再次读取（比如，一个源代码文件被保存为一个文件，然后由编辑器对它进行读取），所以把写入的数据放入缓冲区内是个妙计。另一方面，只把数据放入缓冲区，不立即将其写入磁盘，通过这一方式，程序的运行就要快得多。然后，写入磁盘的操作被移到后台进行，丝毫没有影响其他程序的

运行速度。

许多操作系统都有缓冲区（虽然它们可能也被称为别的什么），但并不表示所有的操作系统都遵循上面的规则。有的“即时写”：数据立即被写入磁盘（当然，同样是保存在缓冲区内）。如果写入操作在稍后进行的话，这个缓冲区就叫作“后写”（write back）。和即时写相比，后写更为有效，但也更容易出错：如果缓冲区内等待写操作的数据在被写入磁盘之前，出现了这类情况：机器崩溃，或电源被切断，或者软盘从软盘驱动器中取走了，缓冲区内的改动通常都会丢失。这意味着文件系统（如果有的话）不再完整，其原因可能是未写入的数据对管理操作员信息来说，是非常关键的。

鉴于此，绝对不应该在没有采用正确的关机进程的情况下，关掉电源，或在未完全装入软盘中数据之前，或它向正在使用它的某某程序发出已完成信号，且软驱指示灯不再闪之前，抽出软盘。sync命令刷新缓冲区，也就是说，迫使所有未写入的数据写入磁盘，供希望确定所有数据都已安全写入的用户使用。在过去的 Unix系统中，有一个程序名为 update，它运行于后台模式，每隔 30s，就执行一次同步处理，所以一般说来，没必要使用 sync命令。Linux系统中有一个额外的后台程序，叫作 bdflush，它更频繁地实行更不完全的同步刷新，以避免由于负荷过大的磁盘I/O导致的宕机，sync有时会出现这样的情况。

Linux系统下，bdflush是由update来启动的。通常情况下，不用过多操心它，但如果出于某种原因，bdflush死了，内核将对此发出警告，而你则应该手工启动它（/sbin/update）。

缓冲区实际上不对文件进行缓冲处理，文件是最小的磁盘 I/O单元（Linux系统下，它们的大小通常是1K）。缓冲处理的对象是目录、超级数据块、其他文件系统管理操作数据和非文件系统磁盘。

缓冲区的效力基本上是由其大小来决定的。小型缓冲区没什么用：其中将容纳极少的数据，以致所有的缓冲数据都是在再次使用它之前，通过缓冲区得以刷新的。临界大小和读写数据的多少有关，与同一个数据的访问频率有关。具体情况只有试过以后，才能知道。

如果缓冲区的大小是固定的，最好不要将它设得很大。因为那样可能令空闲的内存空间太小，导致交换减速。为了能更有效地使用真正的内存，Linux针对缓冲区，自动采用所有空闲RAM，但与此同时，在程序需要更多内存时，也会自动令缓冲区变小。

Linux操作系统中，不必做任何事，就可利用缓冲区，缓冲区的采用完全是自动的。除了遵循正确的关机过程和抽出磁盘外，实在没必要为它操心。