离。下面的讨论基本上是基于Mach的。

一个如何分离策略和机制的简单例子可以参见图3-30。其中存储管理系统被分为三个部分：

1) 一个底层MMU处理程序。

2) 一个作为内核一部分的缺页中断处理程序。

3) 一个运行在用户空间中的外部页面调度程序。

所有关于MMU工作的细节都被封装在MMU处理程序中，该程序的代码是与机器相关的，而且操作系统每应用到一个新平台就要被重写一次。缺页中断处理程序是与机器无关的代码，包含大多数分页机制。策略主要由作为用户进程运行的外部页面调度程序所决定。

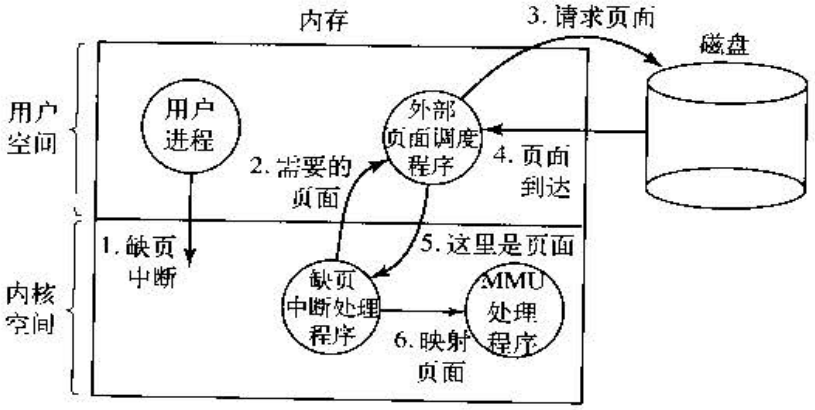

图3-30 用一个外部页面调度程序来处理缺页中断

当一个进程启动时，需要通知外部页面调度程序以便建立进程页面映射，如果需要的话还要在磁盘上分配后备存储。当进程正在运行时，它可能要把新对象映射到它的地址空间，所以还要再一次通知外部页面调度程序。

一旦进程开始运行，就有可能出现缺页中断。缺页中断处理程序找出需要哪个虚拟页面，并发送一条消息给外部页面调度程序告诉它发生了什么问题。外部页面调度程序从磁盘中读入所需的页面，把它复制到自己的地址空间的某一位置。然后告诉缺页中断处理程序该页面的位置。缺页中断处理程序从外部页面调度程序的地址空间中清除该页面的映射，然后请求MMU处理程序把它放到用户地址空间的正确位置，随后就可以重新启动用户进程了。

这个实现方案没有给出放置页面置换算法的位置。把它放在外部页面调度程序中比较简单，但会有一些问题。这里有一条原则就是外部页面调度程序无权访问所有页面的R位和M位。这些二进制位在许多页面置换算法起重要作用。这样就需要有某种机制把该信息传递给外部页面调度程序，或者把页面置换算法放到内核中。在后一种情况下，缺页中断处理程序会告诉外部页面调度程序它所选择的要淘汰的页面并提供数据，方法是把数据映射到外部页面调度程序的地址空间中或者把它包含到一条消息中。两种方法中，外部页面调度程序都把数据写到磁盘上。

这种实现的主要优势是有更多的模块化代码和更好的适应性。主要缺点是由于多次交叉“用户－内核”边界引起的额外开销，以及系统模块间消息传递所造成的额外开销。现在看来，这一主题有很多争议，但是随着计算机越来越快，软件越来越复杂，从长远来看，对于大多数实现，为了获得更高的可靠性而牺牲一些性能也是可以接受的。

3.7 分段

到目前为止我们讨论的虚拟内存都是一维的，虚拟地址从0到最大地址，一个地址接着另一个地址。对许多问题来说，有两个或多个独立的地址空间可能比只有一个要好得多。比如，一个编译器在编译过程中会建立许多表，其中可能包括：

1) 被保存起来供打印清单用的源程序正文（用于批处理系统）。

2) 符号表，包含变量的名字和属性。

3) 包含用到的所有整型量和浮点常量的表。

4) 语法分析树，包含程序语法分析的结果。

5) 编译器内部过程调用使用的堆栈。

前4个表随着编译的进行不断地增长，最后一个表在编译过程中以一种不可预计的方式增长和缩小。在一维存储器中，这5个表只能被分配到虚拟地址空间中连续的块中，如图3-31所示。