合理的页面置换算法：当发生缺页中断时，淘汰一个不在工作集中的页面。为了实现该算法，就需要一种精确的方法来确定哪些页面在工作集中。根据定义，工作集就是最近k次内存访问所使用过的页面的集合（有些设计者使用最近k次页面访问，但是选择是任意的）。为了实现工作集算法，必须预先选定k的值。一旦选定某个值，每次内存访问之后，最近k次内存访问所使用过的页面的集合就是惟一确定的了。

当然，有了工作集的定义并不意味着存在一种有效的方法能够在程序运行期间及时地计算出工作集。设想有一个长度为k的移位寄存器，每进行一次内存访问就把寄存器左移一位，然后在最右端插入刚才所访问过的页面号。移位寄存器中的k个页面号的集合就是工作集。理论上，当缺页中断发生时，只要读出移位寄存器中的内容并排序，然后删除重复的页面。结果就是工作集。然而，维护移位寄存器并在缺页中断时处理它所需的开销很大，因此该技术从来没有被使用过。

作为替代，可以使用几种近似的方法。一种常见的近似方法就是，不是向后找最近k次的内存访问，而是考虑其执行时间。例如，按照以前的方法，我们定义工作集为前1000万次内存访问所使用过的页面的集合，那么现在就可以这样定义：工作集即是过去10ms中的内存访问所用到的页面的集合。实际上，这样的模型很合适且更容易实现。要注意到，每个进程只计算它自己的执行时间。因此，如果一个进程在T时刻开始，在（T+100）ms的时刻使用了40ms CPU时间，对工作集而言，它的时间就是40ms。一个进程从它开始执行到当前所实际使用的CPU时间总数通常称作当前实际运行时间。通过这个近似的方法，进程的工作集可以被称为在过去的τ秒实际运行时间中它所访问过的页面的集合。

现在让我们来看一下基于工作集的页面置换算法。基本思路就是找出一个不在工作集中的页面并淘汰它。在图3-20中读者可以看到某台机器的部分页表。因为只有那些在内存中的页面才可以作为候选者被淘汰，所以该算法忽略了那些不在内存中的页面。每个表项至少包含两条信息：上次使用该页面的近似时间和R（访问）位。空白的矩形表示该算法不需要的其他域，如页框号、保护位、M（修改）位。

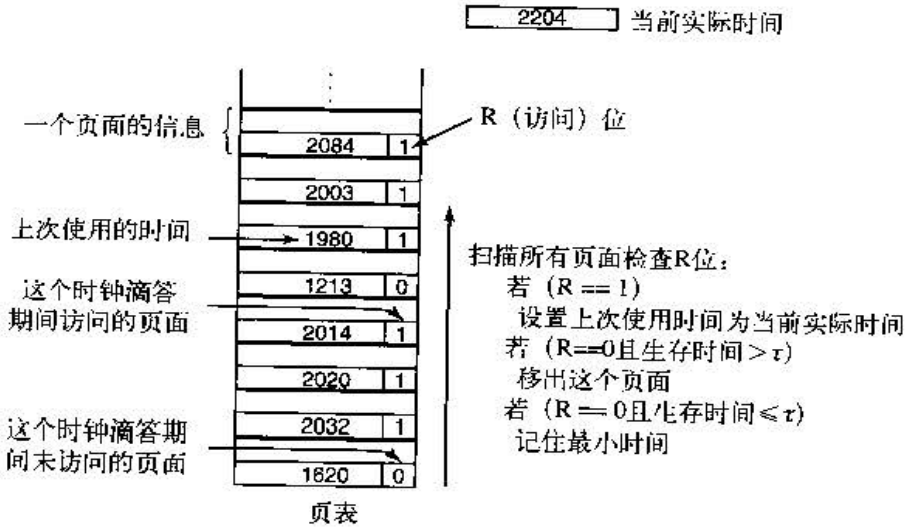

图3-20 工作集算法

该算法工作方式如下。如前所述，假定使用硬件来置R位和M位。同样，假定在每个时钟滴答中，有一个定期的时钟中断会用软件方法来清除R位。每当缺页中断发生时，扫描页表以找出一个合适的页面淘汰之。

在处理每个表项时，都需要检查R位。如果它是1，就把当前实际时间写进页表项的“上次使用时间”域，以表示缺页中断发生时该页面正在被使用。既然该页面在当前时钟滴答中已经被访问过，那么很明显它应该出现在工作集中，并且不应该被删除（假定τ横跨多个时钟滴答）。

如果R是0，那么表示在当前时钟滴答中，该页面还没有被访问过，则它就可以作为候选者被置换。为了知道它是否应该被置换，需要计算它的生存时间（即当前实际运行时间减去上次使用时间），然后与τ做比较。如果它的生存时间大于τ，那么这个页面就不再在工作集中，而用新的页面置换它。扫描会继续进行以更新剩余的表项。