中任意位置的 k 个任意的物理页面。由于页表工作的方式，操作系统没有必要分配 k 个连续的物理内存页面。页面可以随机地分散在物理内存中。

9.5　虚拟内存作为内存保护的工具

任何现代计算机系统必须为操作系统提供手段来控制对内存系统的访问。不应该允许一个用户进程修改它的只读代码段。而且也不应该允许它读或修改任何内核中的代码和数据结构。不应该允许它读或者写其他进程的私有内存，并且不允许它修改任何与其他进程共享的虚拟页面，除非所有的共享者都显式地允许它这么做(通过调用明确的进程间通信系统调用)。

就像我们所看到的，提供独立的地址空间使得区分不同进程的私有内存变得容易。但是，地址翻译机制可以以一种自然的方式扩展到提供更好的访问控制。因为每次 CPU 生成一个地址时，地址翻译硬件都会读一个 PTE，所以通过在 PTE 上添加一些额外的许可位来控制对一个虚拟页面内容的访问十分简单。图 9-10 展示了大致的思想。

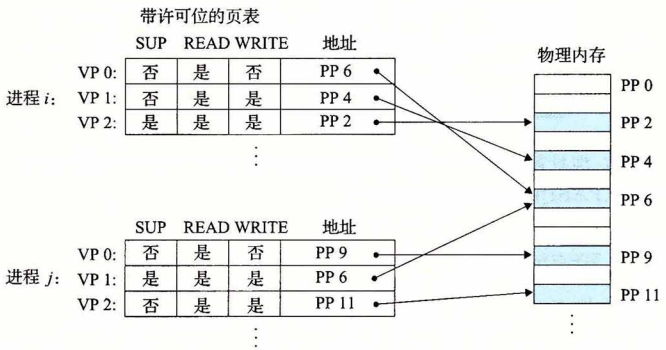

图 9-10　用虚拟内存来提供页面级的内存保护

在这个示例中，每个 PTE 中已经添加了三个许可位。SUP 位表示进程是否必须运行在内核(超级用户)模式下才能访问该页。运行在内核模式中的进程可以访问任何页面，但是运行在用户模式中的进程只允许访问那些 SUP 为 0 的页面。READ 位和 WRITE 位控制对页面的读和写访问。例如，如果进程 i 运行在用户模式下，那么它有读 VP 0 和读写 VP 1 的权限。然而，不允许它访问 VP 2。

如果一条指令违反了这些许可条件，那么 CPU 就触发一个一般保护故障，将控制传递给一个内核中的异常处理程序。Linux shell 一般将这种异常报告为“段错误(segmentation fault)”。

9.6　地址翻译

这一节讲述的是地址翻译的基础知识。我们的目标是让你了解硬件在支持虚拟内存中的角色，并给出足够多的细节使得你可以亲手演示一些具体的示例。不过，要记住我们省略了大量的细节，尤其是和时序相关的细节，虽然这些细节对硬件设计者来说是非常重要的，但是超出了我们讨论的范围。图 9-11 概括了我们在这节里将要使用的所有符号，供读者参考。