大大地简化了内存管理，并提供了一种自然的保护内存的方法。

到目前为止，我们都假设有一个单独的页表，将一个虚拟地址空间映射到物理地址空间。实际上，操作系统为每个进程提供了一个独立的页表，因而也就是一个独立的虚拟地址空间。图 9-9 展示了基本思想。在这个示例中，进程 i 的页表将 V P1 映射到 PP 2，VP 2 映射到 PP 7。相似地，进程 j 的页表将 VP 1 映射到 PP 7，VP 2 映射到 PP 10。注意，多个虚拟页面可以映射到同一个共享物理页面上。

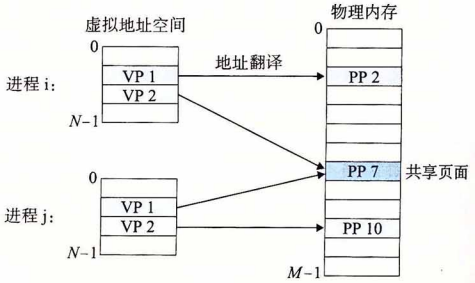

图 9-9　VM 如何为进程提供独立的地址空间。操作系统为系统中的每个进程都维护一个独立的页表

按需页面调度和独立的虚拟地址空间的结合，对系统中内存的使用和管理造成了深远的影响。特别地，VM 简化了链接和加载、代码和数据共享，以及应用程序的内存分配。

- *简化链接*。独立的地址空间允许每个进程的内存映像使用相同的基本格式，而不管代码和数据实际存放在物理内存的何处。例如，像我们在图 8-13 中看到的，一个给定的 Linux 系统上的每个进程都使用类似的内存格式。对于 64 位地址空间，代码段总是从虚拟地址 0x400000 开始。数据段跟在代码段之后，中间有一段符合要求的对齐空白。栈占据用户进程地址空间最高的部分，并向下生长。这样的一致性极大地简化了链接器的设计和实现，允许链接器生成完全链接的可执行文件，这些可执行文件是独立于物理内存中代码和数据的最终位置的。
- *简化加载*。虚拟内存还使得容易向内存中加载可执行文件和共享对象文件。要把目标文件中 .text 和 .data 节加载到一个新创建的进程中，Linux 加载器为代码和数据段分配虚拟页，把它们标记为无效的(即未被缓存的)，将页表条目指向目标文件中适当的位置。有趣的是，加载器从不从磁盘到内存实际复制任何数据。在每个页初次被引用时，要么是 CPU 取指令时引用的，要么是一条正在执行的指令引用一个内存位置时引用的，虚拟内存系统会按照需要自动地调入数据页。

  将一组连续的虚拟页映射到任意一个文件中的任意位置的表示法称作*内存映射*(memory mapping)。Linux 提供一个称为 mmap 的系统调用，允许应用程序自己做*内存映射*。我们会在 9.8 节中更详细地描述应用级内存映射。
- *简化共享*。独立地址空间为操作系统提供了一个管理用户进程和操作系统自身之间共享的一致机制。一般而言，每个进程都有自己私有的代码、数据、堆以及栈区域，是不和其他进程共享的。在这种情况中，操作系统创建页表，将相应的虚拟页映射到不连续的物理页面。

然而，在一些情况中，还是需要进程来共享代码和数据。例如，每个进程必须调用相同的操作系统内核代码，而每个 C 程序都会调用 C 标准库中的程序，比如 printf。操作系统通过将不同进程中适当的虚拟页面映射到相同的物理页面，从而安排多个进程共享这部分代码的一个副本，而不是在每个进程中都包括单独的内核和 C 标准库的副本，如图 9-9 所示。

- *简化内存分配*。虚拟内存为向用户进程提供一个简单的分配额外内存的机制。当一个运行在用户进程中的程序要求额外的堆空间时(如调用 malloc 的结果)，操作系统分配一个适当数字(例如 k)个连续的虚拟内存页面，并且将它们映射到物理内存