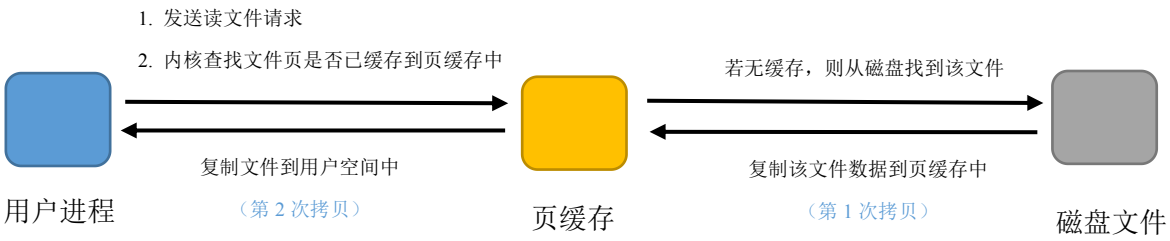
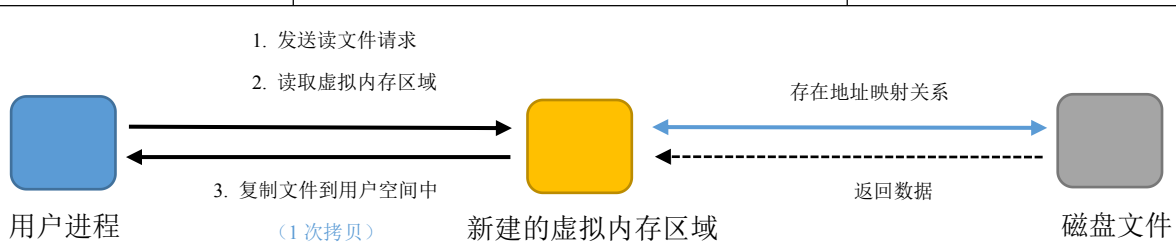
文件读/写操作

传统的 Linux 系统文件操作流程：

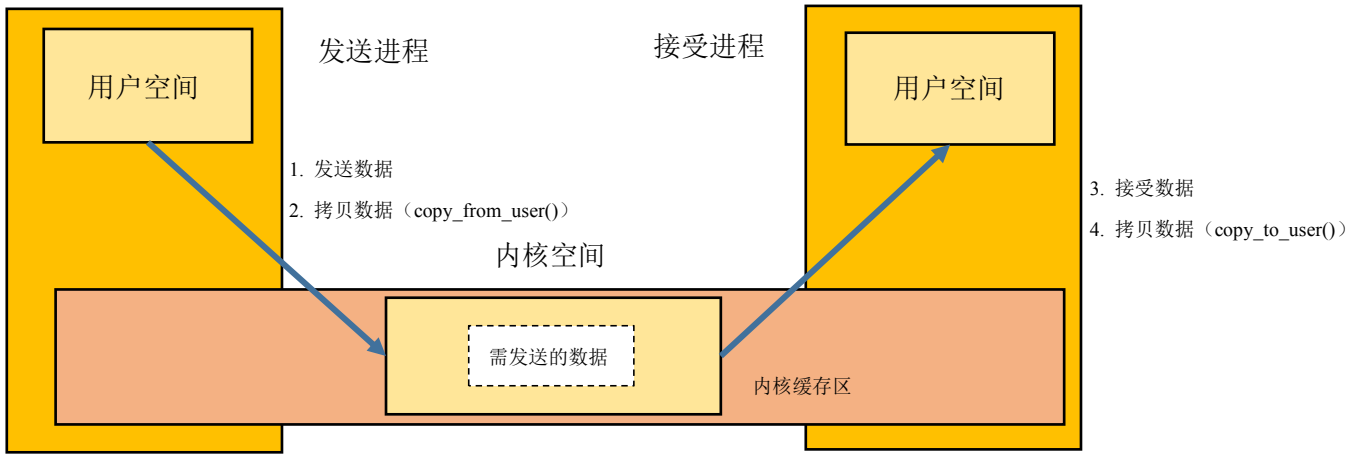
<table>
<tr><td>工作流程</td><td>1. 用户进程发起读文件请求。
2. 内核通过查找进程文件符表，定位到内核已打开文件集上的文件信息，从而找到此文件的 inode。
3. inode 在 address_space 上查找要请求的文件页是否已缓存到页缓存中。
a. 若存在，则直接返回该文件页的内容。
b. 若不存在，则通过 inode 定位到文件磁盘地址，将数据从磁盘复制到页缓存，之后再发起读页面过程，从而将页缓存中的数据返回给用户进程。</td></tr>
<tr><td>示意图</td><td><table><tr><th>用户空间</th><th>内核空间</th><th>磁盘</th></tr><tr><td colspan="3"> 
</td></tr></table></td></tr>
<tr><td>特点</td><td>1. 常规文件操作为了提高读写效率 & 保护磁盘，使用了页缓存机制。
2. 使得读文件时：
(1) 需先将文件页从磁盘拷贝到页缓存中（第 1 次拷贝）
(2) 因页缓存处在内核空间，不能被用户进程直接寻址，故需将页缓存中的数据页再次拷贝到用户空间中（第 2 次拷贝）。
3. 即常规文件的操作需两次数据拷贝能完成。</td></tr>
</table>

使用了内存映射的文件读/写操作：

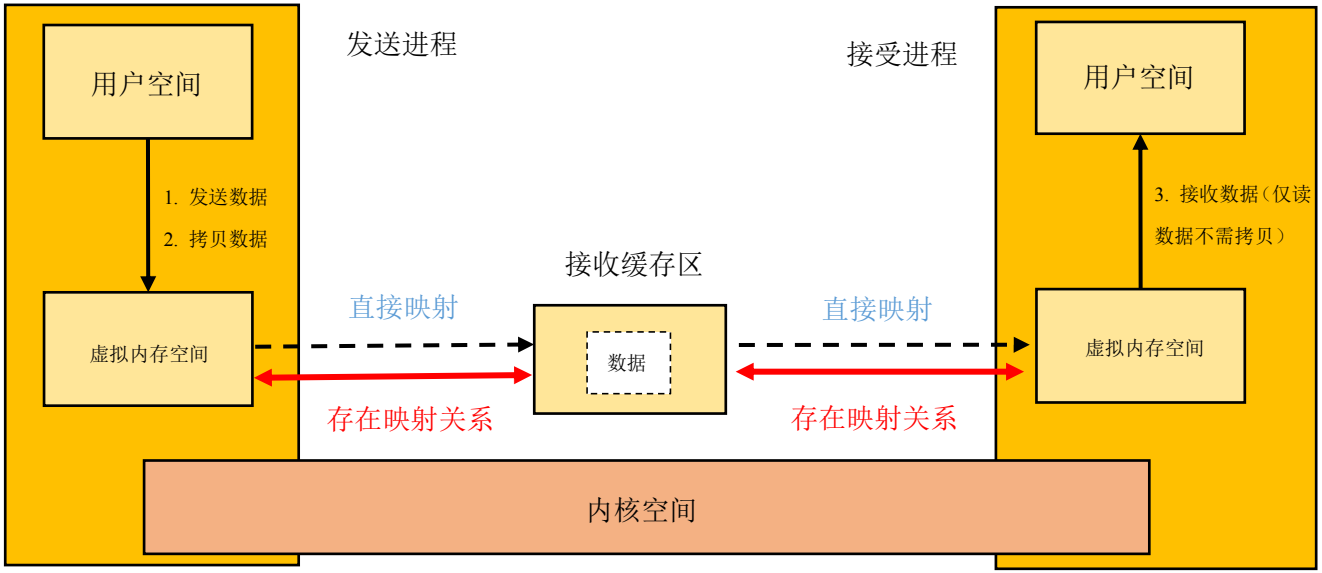
<table>
<tr><td rowspan="4">工作流程
（进程在用户空间调用 mmap()）</td><td>1. 创建虚拟映射区域</td><td>1. 在当前进程的虚拟地址空间中，寻找一段满足大小要求的虚拟地址。
2. 为此虚拟地址分配一个虚拟地址区域（即 vm_area_struct 结构）
3. 初始化该虚拟内存区域
4. 将该虚拟内存区域插入到进程的虚拟内存区域链表/树中</td></tr>
<tr><td>2. 实现地址映射关系
（进程虚拟地址空间 >> 磁盘地址）</td><td>1. 依次通过待映射的文件指针、文件描述符 & 文件结构体，最终调用内核空间的系统调用函数 mmap()。
2. 内核空间的 mmap()通过 VFS 中的 inode 模块定位到文件磁盘物理地址。
3. 通过 remap_pfn_range()建立页表，即实现文件地址和虚拟地址区域的映射关系。（此时该虚拟地址并没有任何数据关联到主存中）</td></tr>
<tr><td colspan="2">注：前两个阶段仅创建了虚拟空间 & 映射地址，但并无任何文件数据拷贝至主存；真正的数据拷贝时刻在进程发起读/写操作时</td></tr>
<tr><td>3. 进程访问该映射空间，实现文件内容到物理内存的数据拷贝</td><td>1. 进程的读/写操作访问虚拟地址空间这一段映射地址。
2. 若进程通过写操作改变了其内容，一定时间后系统自动回写脏页面到对应的磁盘地址，即完成了写入到文件的过程。（修改过的脏页面不会立即更新，而是有延迟，但可通过 msync()来强制同步。此时所写的内容就能立即保存到文件。）</td></tr>
<tr><td>示意图</td><td colspan="2"> 
</td></tr>
<tr><td>特点
（与传统方式相比）</td><td colspan="2">1. 用户空间 & 内核空间的高效交互：通过映射的区域直接交互。
2. 数据拷贝的次数减少：对文件的读取操作跨过了页缓存，减少了数据的拷贝次数（1 次）。
3. 文件的读取效率提高：用内存读写代替 I/O 读写。
4. 可实现高效的大规模数据传输：借助磁盘空间协助大数据操作时，采用 mmap 可提高操作效率。</td></tr>
</table>

跨进程通信

传统的跨进程通信

工作流程	1. 发送进程通过系统调用，将需发送的数据拷贝到 Linux 进程的内核空间的缓存区中。（第一次数据拷贝，通过 copy_from_user()） （注：进程空间分为用户空间 & 内核空间。其中用户空间不可共享 & 直接传输数据；内核空间所有进程共享 & 可传输数据。） 2. 内核服务程序唤醒接受进程的接受线程，通过系统调用将数据发送到接受进程的用户空间中，最终完成数据发送。（第二次数据拷贝，通过 copy_to_user()） 即最终实现了进程间的用户空间的数据交互。
示意图	
缺点	1. 效率低下：两次数据拷贝。 2. 接收数据的缓存要由接收方提供，但接收方却不知需多大的缓存。 （一般做法是，开辟尽量大的空间 or 先调用 API 接收消息头，再开辟适当的空间接收消息体。但前者浪费空间，后者浪费时间）

使用了内存映射的跨进程通信

工作流程	1. 创建一块共享的接收缓存区。 2. 实现地址映射关系：即根据需映射的接收进程信息，实现发送进程的虚拟内存空间和接收进程的虚拟内存空间同时映射到同一个共享接收缓存区中。 （注：前两个阶段仅创建了虚拟区间 & 映射关系，但并无传输数据，真正数据传输的时刻为进程发起读/写操作时） 3. 发送进程将数据发送到自身的虚拟内存区域（数据拷贝一次）。 4. 由于发送进程的虚拟内存区域 & 接收进程的虚拟内存区域存在映射关系（同时映射到一个共享对象中），故相当于也发送到了接收进程的虚拟内存区域中，即实现了跨进程通信。
示意图	
缺点	1. 传输效率高：数据拷贝次数少（一次）、用户空间之间可直接通过共享对象直接交互（直接跨过了内核空间）。 2. 为接收进程分配了不确定大小的接收缓存区。