#### 典型回答

我们期望能够执行逻辑地址空间大于可用物理地址空间的进程。虚拟内存是一种技术，能够将较大的逻辑地址空间映射到较小的物理内存。虚拟内存允许采用允许运行极大的进程，提高多道程度，提高 CPU 利用率。再者，它使程序员不必担心内存可用性。此外，虚拟内存允许多个进程共享系统库和内存。通过虚拟内存，可以采用写入时复制进行高效的进程创建，这里的父进程和子进程共享实际的内存页面。

虚拟内存的实现通过采用请求调页。纯请求调页只有在页面被引用时才会掉入。首次引用会导致操作系统发生缺页错误。操作系统内核查询内部表，以便确定这个页面在后备存储上的位置。然后，它找到空闲帧，并从后备存储中读取页面；更新页表以反映此更改，并重新启动导致缺页错误的指令。这种方法允许进程运行，即使它的整个内存映像不能同时在内存中。只要缺页错误率足够低，那么性能就可接受。

#### 知识延伸

虚拟内存技术允许执行进程不必完全处于内存。这种方案的一个主要优点就是，程序可以大于物理内存。此外，虚拟内存将内存抽象成一个巨大的、统一的存储数组，进而实现了用户看到的逻辑内存与物理内存的分离。这种技术使得程序员不再担忧内存容量的限制。虚拟内存还允许进程轻松共享文件和实现共享内存。此外，它为创建进程提供了有效的机制。然而，虚拟内存的实现并不容易，并且使用不当还可能会大大降低性能。下面会以请求调页为例来讨论虚拟内存。

**请求调页**

下面就要讨论如何采用请求调页来实现虚拟内存。

想一想，如何从磁盘加载可执行程序到内存？

一种选择是，在程序执行时将整个程序加载到物理内存。然而，这个方法的一个问题是，最初可能不需要整个程序都处于内存。假设程序开始时带有一组用户可选的选项。加载整个程序会导致所有选项的执行代码都加载到内存中，而不管这些选项是否最终使用。另一种策略是，仅在需要时才加载页面。这种技术被称为请求调页，常常用于虚拟内存系统。对于请求调页的虚拟内存，页面只有在程序执行期间被请求时才被加载。因此，从未访问的那些页从不加载到物理内存。

当换入进程时，调页程序会猜测在该进程被再次换出之前会用到哪些页。调页程序不是调入整个进程，而是把那些要使用的页调入内存。这样，调页程序就避免了读入那些不适用的页，也减少了交换时间和所需的物理内存空间。

使用这种方案需要一定形式的硬件支持，以区分内存的页面和磁盘的页面。

如果进程试图访问那些尚未调入内存中的页面，情况会如何呢？对标记为无效的页面访问会产生缺页错误。分页硬件在通过页表转换地址时会注意到无效位被设置，从而陷入操作系统。这种陷阱是由于操作系统未能将所需的页面调入内存引起的，处理这种缺页错误的程序很简单：

1. 检查这个进程的内部表，以确定该引用是有效的还是无效的内存访问
2. 如果引用无效，那么终止进程。如果引用有效但尚未调入页面，那么现在就应调入
3. 调度一个空闲帧
4. 调度一个磁盘操作，以将所需页面读到刚分配的空闲帧
5. 当磁盘读取完成时，修改进程的内部表和页表，以指示该页现在处于内存中
6. 重新启动被陷阱中断的指令。该进程现在能访问所需的页面，就好像它总是在内存中

请求调页的关键要求是在缺页错误后重新启动任何指令的能力。

页面置换是请求调页的基础。它完成了逻辑内存和物理内存之间的分离，采用这种机制，较小的物理内存能为程序员提供巨大的虚拟内存。

为实现请求调页，必须解决两个主要问题：应设计帧分配算法和页面置换算法。也就是说，如果有多个进程在内存中，则必须决定要为每个进程分配多少帧；并且当需要页面置换时，必须选择要置换的帧。设计适当的算法来解决这些问题是个重要任务。

通过请求调页可以减少分配给进程的帧数。这种安排可以增加多道程度，并且增加系统 CPU 的利用率。即使进程内存需要超过了总的物理内存，也允许执行。这些进程运行在虚拟内存中。

如果总的内存需求超过物理内存的容量，则可能需要置换内存的页面，以生产新页面的空闲帧。可以使用多种页面置换算法。FIFO 页面置换算法容易编程，但是会遭受 Belady 异常。最优页面置换需要前置信息。LRU 页面置换是最优页面置换的近似，但是仍难以实现。大多数页面置换算法，例如第二次机会算法，是 LRU 置换的近似。

除了页面置换算法之外，还需要帧分配策略。分配可以是固定的，此时建议采用局部页面置换算法；也可以是动态的，此时建议采用全局置换。工作集模型假定进程执行的局部性。工作集是当前局部的所有页面的集合。因此，应当为每个进程的当前工作集分配足够多的帧。如果一个进程没有足够的内存用于工作集，则会发生抖动。为进程提供足够的内存以避免抖动，可能需要进程交换和调度。

大多数操作系统提供内存映射文件的能力，如允许文件 I/O 作为常规内存访问。Win32 API 通过文件的内存映射来实现共享内存。

内核进程通常需要采用物理连续页面来分配内存。伙伴系统允许内核进程按 2 的幂大小来分配，这通常会导致碎片。slab 分配器允许从由 slab 组成的 cache 上分配，每个 slab 由若干物理连续的页面组成。采用 slab 分配，不会因碎片问题而产生内存浪费，并且内存请求可以快速满足。