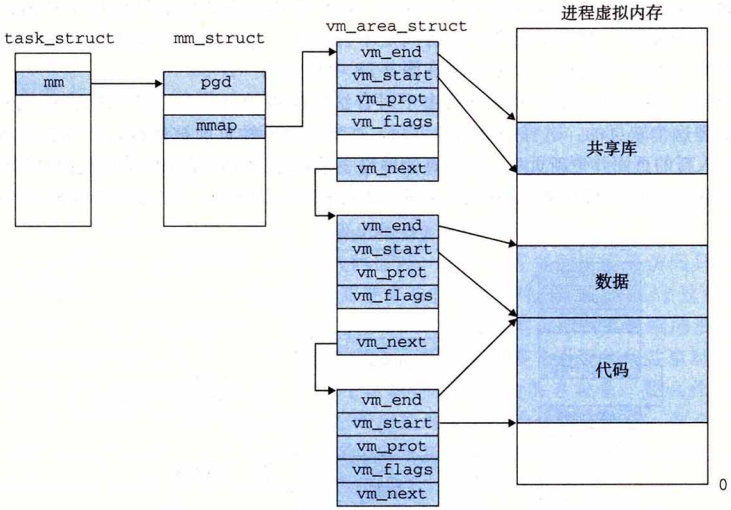

图 9-27　Linux 是如何组织虚拟内存的

任务结构中的一个条目指向 mm_struct，它描述了虚拟内存的当前状态。我们感兴趣的两个字段是 pgd 和 mmap，其中 pgd 指向第一级页表(页全局目录)的基址，而 mmap 指向一个 vm_area_structs(区域结构)的链表，其中每个 vm_area_structs 都描述了当前虚拟地址空间的一个区域。当内核运行这个进程时，就将 pgd 存放在 CR3 控制寄存器中。

为了我们的目的，一个具体区域的区域结构包含下面的字段：

- vm_start：指向这个区域的起始处。
- vm_end：指向这个区域的结束处。
- vm_prot：描述这个区域内包含的所有页的读写许可权限。
- vm_flags：描述这个区域内的页面是与其他进程共享的，还是这个进程私有的(还描述了其他一些信息)。
- vm_next：指向链表中下一个区域结构。

2. Linux 缺页异常处理

假设 MMU 在试图翻译某个虚拟地址 A 时，触发了一个缺页。这个异常导致控制转移到内核的缺页处理程序，处理程序随后就执行下面的步骤：

1) 虚拟地址 A 是合法的吗？换句话说，A 在某个区域结构定义的区域内吗？为了回答这个问题，缺页处理程序搜索区域结构的链表，把 A 和每个区域结构中的 vm_start 和 vm_end 做比较。如果这个指令是不合法的，那么缺页处理程序就触发一个段错误，从而终止这个进程。这个情况在图 9-28 中标识为“1”。

因为一个进程可以创建任意数量的新虚拟内存区域(使用在下一节中描述的 mmap 函数)，所以顺序搜索区域结构的链表花销可能会很大。因此在实际中，Linux 使用某些我们没有显示出来的字段，Linux 在链表中构建了一棵树，并在这棵树上进行查找。

2) 试图进行的内存访问是否合法？换句话说，进程是否有读、写或者执行这个区域内页面的权限？例如，这个缺页是不是由一条试图对这个代码段里的只读页面进行写操作