进程地址空间

  地址空间有进程所使用的的全部线性地址所组成，每个进程看到的线性地址集合是不同的，不同的进程使用的地址之间没什么关系。

**内存描述符**：（mm\_struct）指向线性区的开始，和最后一个引用的线性区。

                         mmap：指向的就是当前内存描述符之下的线性区链表头。

                         mmap\_cache：指向最后一个引用的线性区对象。

                         get\_unmapped\_area：搜索有效线性地址区间

                         unmap\_area：释放线性地址区间

                         free\_area\_cache：内核从此地址开始搜索空闲线性地址

                         map\_cout：线性区个数

                         mmlist：指向相邻元素

                         start\_code,end\_code：可执行代码的起始终止位置

                         start\_data,end\_date：已初始化数据的起始终止位置

                         start\_brk，brk：堆的起始位置,当前最后位置

                         start\_stack：用户态栈的起始位置

                         stack\_vm：用户态堆栈的页数

                         ......

  所有的内存描述符放在一个双向链表中（Linux2.6中放在红黑树）

**线性区（vm\_area\_struct）（程序看到的地址如何转化为线性地址）**

    线性区对象中的字段：vm\_start,vm\_end就是用来标示在线性地址空间的内存段信息的。

      vm\_mm：指回内存描述符

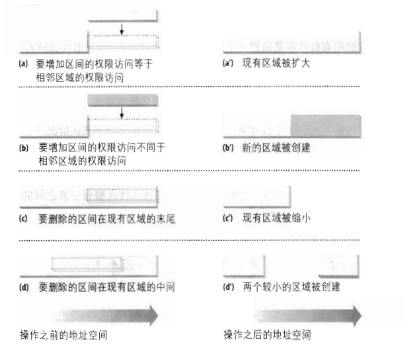
      vm\_next：下一个线性区

      vm\_ops：指向线性区的方法（数据结构）

      shared，anon\_vma\_node,anon\_vma：映射相关

      vm\_flag：叶标志

    每个线性区描述符指向一个线性区间，进程所拥有的线性区从来不重叠，内核会尽量把新分配的线性区和紧邻的线性区合并在一起



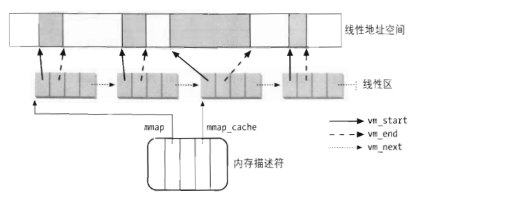
      vm\_ops：open：将线性区增加到进程所拥有的的线性区集合中时使用

                        close：删除时使用

                        nopage：进程访问ram中不存在的叶，但该叶的线性地址属于线性区时触发缺页异常，由缺页处理程序调用

                        populate：设置线性地址所对应的页表项

    进程所拥有的所有线性区通过一个链表链接在一起，通过内存升序排列



    线性区处理：(底层函数)

      find\_vma():查找给定地址的最邻近线性区，参数为mm和给定线性地址addr

      find\_vma\_intersection:查找与给定地址区间重叠的线性区

      get\_unmapped\_area:查找未分配线性区，搜索进程地址空间以获取一个可以使用的线性区

      insert\_vm\_struct：向内存描述符中插入一个vm\_area\_struct