mlock 实现原理及应用

2015/6/16
renyl

1 mlock 简介

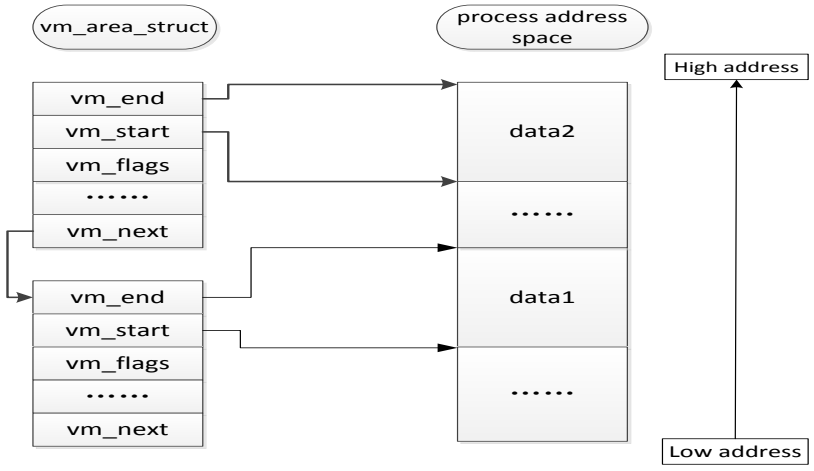
1) mlock（memory locking）是内核实现锁定内存的一种机制，用来将进程使用的部分或全部虚拟内存锁定到物理内存。

2) mlock 机制主要有以下功能：
   a) 被锁定的物理内存在被解锁或进程退出前，不会被页回收流程处理。
   b) 被锁定的物理内存，不会被交换到 swap 设备。
   c) 进程执行 mlock 操作时，内核会立刻分配物理内存。

2 mlock 原理

1) 每个进程都拥有一段连续的虚拟内存，内核并不是以整个虚拟内存为管理单位，而是将整个虚拟内存划分为若干个虚拟内存区域。

2) 内核中使用 vm_area_struct 数据结构来管理虚拟内存区域，简称 vma。vma 管理虚拟内存的方式如下图所示：

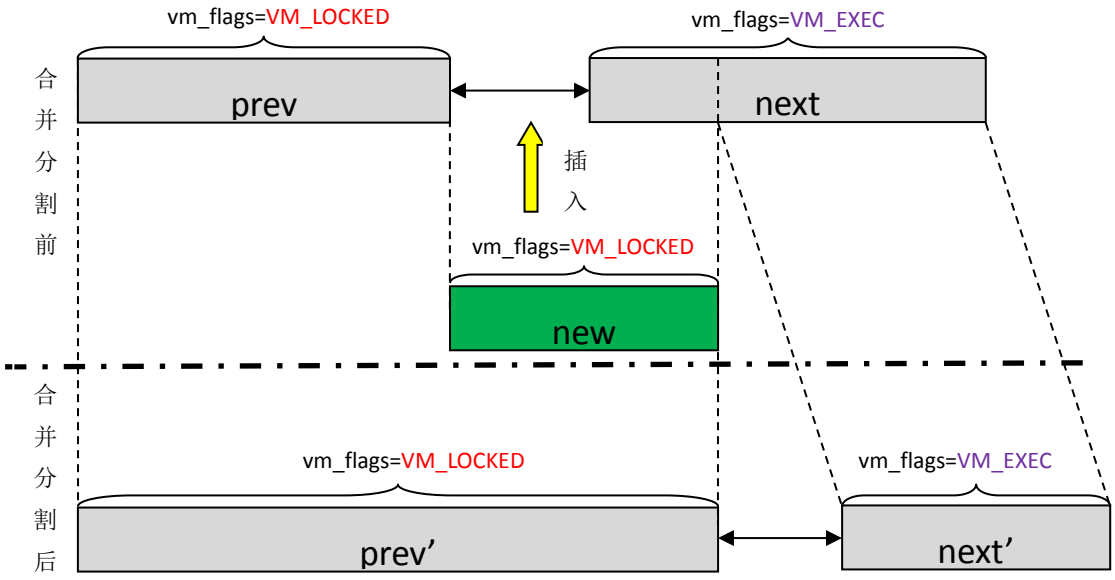


3) 每一个 vma 代表一个已映射的、连续的且属性相同(如可读/写)的虚拟内存区域。内核采用链表和 AVL 树形式管理 vma，其中链表用于遍历，AVL 树用来查找。

4) mlock 操作会给相应的 vma 的 vm_flags 置一个 VM_LOCKED 标记，而这个标记则会影响到物理内存回收和交换。

5) Linux 分配内存到 page 且只能按页锁定内存，所以指定的地址会被 round down 到下一个 page 的边界。

6) 当 mlock 锁定的虚拟内存区域跟现有 vma 管理的虚拟内存区域并不完全重合时，由于同一个 vma 的内存属性要求一致，而 VM_LOCKED 标记也是其属性，因此会导致现有的 vma 被合并或分割，如下图所示：



说明：

a) prev 与 next 是已受链表管理的 vma 结构，new 是将要新加入链表的 vma。

b) 当 new 加入时，如果 new 的起始地址与 prev 的结束地址相同，且 new 属性与 prev 属性均为 VM_LOCKED，则将 prev 和 new 合并成 prev'。

c) 如果 new 的结束地址与 next 的起始地址有重合，但 next 属性是 VM_EXEC，则 next 被分割成两部分，一部分加入 prev'，另一部分变成 next'。

7) 操作系统通过 LRU 算法来管理 Linux 进程的虚拟内存，LRU 算法主要通过两个页面标志符（PG_active 和 PG_referenced）来标识某个页面的活跃程度，从而决定页面如何在

两个链表（active_list 和 inactive_list）之间进行移动。

8) 内核函数 vmscan 会遍历扫描 active_list 和 inactive_list 链表来回收页面。

9) 内核在 LRU 算法中新增了一个 unevictable_list 链表，将不可回收的页面都放在 unevictable_list 中，mlock 的页面就被放在 unevictable_list 中，同时给该页置一个 PG_mlocked 标记。

注：除了 mlock 的页，ramdisk 或 ramfs 页以及共享内存映射的页也被放入 unevictable_list 中。

10) 解锁并不立刻将解锁的页回收，而是将解锁的页放回 active_list 或 inactive_list 链表，然后交由页回收流程处理，所以 mlock 的页不会被页回收流程处理。

11) 由于线程共享进程资源，所以线程的 vma 将继承 VM_LOCKED 标记。但 fork 后的子进程的 vma 并不继承 VM_LOCKED 标记，以及调用 exec 执行其它程序时，不继承 VM_LOCKED 标记。

3 mlock 函数

mlock 机制通过提供四个函数来使用，如下所示：

函数	描述
int mlock(const void *addr, size_t len)	将起始地址 addr ，长度为 len 的虚拟内存锁定到物理内存
int munlock(const void *addr, size_t len)	将起始地址 addr ，长度为 len 的物理内存解锁
int mlockall(int flags)	flags　　当前虚拟内存　　新增加虚拟内存 MCL_CURRENT:　　○　　× MCL_FUTURE:　　×　　○ MCL_CURRENT|MCL_FUTURE:　　×　　○ ○：锁定到物理内存 ×：不锁定到物理内存
int munlockall(void)	将 mlockall()的物理内存解锁

注：

1) 上述函数只能由 root 用户执行。

2) mlock 操作不可叠加，多次调用 mlock 的一段内存会被一次 unlock 解锁。

3) 锁定进程所有映射到地址空间的数据包括：代码、数据、栈片段分页，共享库、用户空间内核数据、共享内存以及内存映射的文件。

4 mlock 应用

1) mlock 机制在虚拟化中的应用

libvirt 已提供对 mlock 支持,可以通过在 Guest 的 xml 文件中添加如下内容来开启 qemu-kvm 进程的 mlock 机制

```
<memoryBacking>
   <locked/>
</memoryBacking>
```

注：默认情况下 qemu-kvm 进程中的 mlock 选项是关闭的。

2) mlock 机制在实时环境下的应用

mlocked	访问内存产生 page fault 次数	访问内存时间开销
on	1	低
off	>1	高

说明：

a) 采用 mlock 机制时,仅在虚拟内存和物理内存建立映射关系时产生一次 page fault。

b) 未采用 mlock 机制时,每当访问不同的虚拟内存单元时,就会产生一次 page fault。、

3) mlock 机制在多线程环境下的应用

mlocked	线程栈空间映射物理内存大小	映射物理内存时间开销
on	10MB	高
off	1MB	低

注：

a) 假设线程整个栈空间大小 10MB，其中 1MB 空间已使用。

b) 假设 10MB 虚拟内存和物理内存建立映射关系耗时 1ms，1MB 虚拟内存和物理内存建立映射关系耗时 0.1ms。

说明：

当进程涉及多线程操作时：

a) 采用 mlock 机制时，线程整个栈空间 10MB 会被映射到物理内存，若一个进程中有 2000 个线程，则线程栈空间和物理内存建立映射关系共耗时 2000ms。

b) 未采用 mlock 机制时，线程栈空间已使用的 1MB 会被映射到物理内存，若一个进程中有 2000 个线程，则线程栈空间和物理内存建立映射关系共耗时 200ms。

5 mlock 代码

```
/*
 *mlock.c
 *
 * check vma's merge and truncation for mlock mechanism
 *
 * renyl 2014/3/24  0.1version
 *
*/

#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>

int main(void)
{

    long size=1024*1024*1024L;  // size=1GB

char * addr_p=(char*)malloc(size);
    memset(addr_p,0,size);
    printf("addr_p=%p --- %p \n",addr_p,addr_p+size);

    int pid=getpid();
    char pid_array[10];
    sprintf(pid_array,"%d",pid);
```

```
    char command[100]="cat /proc/";
    sprintf(command+10,"%s",pid_array);
    sprintf(command+10+strlen(pid_array),"%s","/maps");


    system(command);



    printf("----------------------------------------------------------------\n");
    printf("--------------------------mlock after---------------------------\n");
    printf("----------------------------------------------------------------\n");



    if(mlockall(MCL_FUTURE)==-1)
    {
        perror("mlockall failed:");
        exit(1);
    }


    char *addr2_p =(long)addr_p & 0xfffffffffff0000;  //make sure addr2_p is a
multiple of the page size.
    printf("addr2_p=%p --- %p \n",addr2_p,addr2_p+40960);



    if(mmap((void*)addr2_p,40960,PROT_READ|PROT_WRITE, \
            MAP_PRIVATE|MAP_ANONYMOUS|MAP_FIXED, -1, 0)==(void*)-1)   // note: use
MAP_FIXED  flag
    {
        perror("mmap failed");
        exit(2);
    }
```

```
    system(command);


    munlockall();
    munmap((void*)addr2_p,40960);
    free(addr_p);
    return 0;
}
```