# 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：
3. 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。
4. 被锁定的物理内存，不会被交换到swap设备。
5. 进程执行mlock操作时，内核会立刻分配物理内存。

# 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：



1. 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。
2. 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。
3. 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。
4. 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：

插

入

vm\_flags=VM\_LOCKED

next’

prev’

合并分割前：

prev

next

vm\_flags=VM\_EXEC

vm\_flags=VM\_LOCKED

vm\_flags=VM\_LOCKED

new

合并分割后：

vm\_flags=VM\_EXEC

说明：

1. 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。
2. 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’。
3. 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’。
4. 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。
5. 内核函数vmscan会遍历扫描active\_list和inactive\_list链表来回收页面。
6. 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。

1. 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。
2. 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。

1. 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：

1. 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。
2. 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：

1. 假设线程整个栈空间大小10MB，其中1MB空间已使用。
2. 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：

1. 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。
2. 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 & 0xffffffffffff0000; //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;  } |