

七 Linux 内存管理

实验目的

- 1、理解 Linux 内存地址空间的管理
- 2、理解 VMA 相关操作

实验环境

安装有 Linux 操作系统的计算机

实验步骤

在 32 位的系统上，线性地址空间为 4GB，其中用户进程占有 3GB 线性地址空间，内核占有 1GB 线性地址空间。由于虚拟内存的引入，使的每个进程都可拥有 3GB 的虚拟内存。

用户进程的虚拟地址空间包含若干区域，这些区域的分布方式因体系结构的差异而不同，但所有的方式都包含下列成分：

- (1) 代码段：可执行文件的二进制代码
- (2) 数据段：存储全局变量
- (3) 栈：用于保存局部变量和实现函数调用
- (4) 环境变量和命令行参数
- (5) 程序使用的动态库的代码
- (6) 用于映射文件内容的区域

为便于描述，系统中进程的虚拟内存空间被划分为若干不同的区域，每个区域都有其相关的属性和用途，一个合法的地址总是落在某个区域当中的，这些区域也不会重叠。在 Linux 内核中，这样的区域被称为虚拟内存区域(virtual memory areas, VMA)。一个 VMA 是一块连续的线性地址空间的抽象，它拥有自身的权限(可读，可写，可执行等)，对进程而言，VMA 其实是虚拟空间的内存块，一个进程的所有资源由多个内存块组成。

每一个虚拟内存区域都由一个相关的 `struct vm_area_struct` 结构来描述。

本实验内容编写一个内核模块，遍历一个用户进程中所有的 VMA，并且打印这些 VMA 的属性信息，如 VMA 的大小、起始地址等，并通过与 `/proc/pid/maps` 中显示的信息进行对比验证 VMA 信息是否正确。

1. 编写模块程序

参考代码如下，本示例中代码文件命名“vma_test.c”。

```
1. #include <linux/module.h>
2. #include <linux/init.h>
3. #include <linux/mm.h>
4. #include <linux/sched.h>
5.
6. static int pid;
7. module_param(pid, int, 0644);
8.
9. static void printit(struct task_struct *tsk)
10. {
11.     struct mm_struct *mm;
12.     struct vm_area_struct *vma;
13.     int j = 0;
14.     unsigned long start, end, length;
15.
16.     mm = tsk->mm;
17.     pr_info("mm_struct addr = 0x%p\n", mm);
18.     vma = mm->mmap;
19.
20.     /* 使用 mmap_sem 读写信号量进行保护 */
21.     down_read(&mm->mmap_sem);
22.     pr_info("vmas:          vma          start          end          length\n");
23.
24.     while (vma) {
25.         j++;
26.         start = vma->vm_start;
27.         end = vma->vm_end;
28.         length = end - start;
29.         pr_info("%6d: %16p %12lx %12lx   %8ld\n",
30.             j, vma, start, end, length);
31.         vma = vma->vm_next;
32.     }
33.     up_read(&mm->mmap_sem);
34. }
35.
36. static int __init vma_init(void)
37. {
38.     struct task_struct *tsk;
39.     /* 如果插入模块时未定义 pid 号，则使用当前 pid */
40.     if (pid == 0) {
41.         tsk = current;
```

```

42.     pid = current->pid;
43.     pr_info("using current process\n");
44. } else {
45.     tsk = pid_task(find_vpid(pid), PIDTYPE_PID);
46. }
47. if (!tsk)
48.     return -1;
49. pr_info(" Examining vma's for pid=%d, command=%s\n", pid, tsk->comm);
50. printit(tsk);
51. return 0;
52. }
53.
54. static void __exit vma_exit(void)
55. {
56.     pr_info("Module exit\n");
57. }
58.
59. module_init(vma_init);
60. module_exit(vma_exit);
61.
62. MODULE_LICENSE("GPL");
63. MODULE_AUTHOR("Mr Yu");
64. MODULE_DESCRIPTION("vma test");

```

以上代码中：

38-52 行是内核模块初始化函数 vma_init；

40-46 行目的是获取 pid，可在加载模块时可传递相关参数（即进程 pid）；如果没有传递参数，则使用当前进程，即执行 insmod 命令的进程；

45 行 pid_task() 函数为获取任务的任务描述符信息，其返回值是 struct task_struct 结构体类型的变量；

50 行调用自定义的 printit() 函数打印相关信息；

9-34 行是本实验核心函数；

16 行获取待检查进程的内存描述符 struct mm_struct 数据结构，该结构由 struct task_struct 中的 *mm 指向；

18 行获取 VMA 链表头，即 mm->mmap；

21 行开始遍历 VMA 链表，down_read() 函数用于申请读信号量，因本程序只是读取 VMA 链表，所以申请读者类型即可，若需丢 VMA 链表进行修改，则需申请写者类型信号量；

24-32 行遍历 VMA 链表，并对每个 VMA 打印其起始地址、终止地址和长度信息；

33 行释放读者信号量。

2. 编译内核模块

编写 Makefile 文件，文件名必须为“Makefile”

```

1.  obj-m := vma_test.o
2.
3.  KERNELBUILD := /lib/modules/$(shell uname -r)/build
4.  CURRENT_PATH := $(shell pwd)
5.
6.  all:
7.      make -C $(KERNELBUILD) M=$(CURRENT_PATH) modules
8.
9.  clean:
10.     make -C $(KERNELBUILD) M=$(CURRENT_PATH) clean

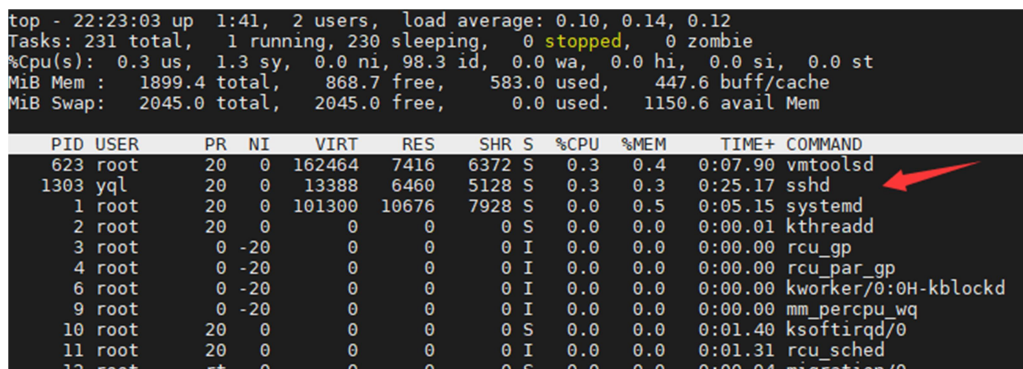
```

3. 编译

使用 make 命令编译即可。

4. 插入模块

先通过 top 命令查看进程，任意获取一个进程 pid，如图所示，本例中获取 ssh 进程的 pid 1303。



```

top - 22:23:03 up 1:41, 2 users, load average: 0.10, 0.14, 0.12
Tasks: 231 total, 1 running, 230 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.3 us, 1.3 sy, 0.0 ni, 98.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 1899.4 total, 868.7 free, 583.0 used, 447.6 buff/cache
MiB Swap: 2045.0 total, 2045.0 free, 0.0 used, 1150.6 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ COMMAND
  623 root        20   0 162464   7416  6372  S   0.3   0.4   0:07.90 vmtoolsd
 1303 yql         20   0  13388   6460  5128  S   0.3   0.3   0:25.17 sshd
    1 root        20   0 101300 10676  7928  S   0.0   0.5   0:05.15 systemd
    2 root        20   0      0      0      0  S   0.0   0.0   0:00.01 kthreadd
    3 root        0 -20      0      0      0  I   0.0   0.0   0:00.00 rcu_gp
    4 root        0 -20      0      0      0  I   0.0   0.0   0:00.00 rcu_par_gp
    6 root        0 -20      0      0      0  I   0.0   0.0   0:00.00 kworker/0:0H-kblockd
    9 root        0 -20      0      0      0  I   0.0   0.0   0:00.00 mm_percpu_wq
   10 root        20   0      0      0      0  S   0.0   0.0   0:01.40 ksoftirqd/0
   11 root        20   0      0      0      0  I   0.0   0.0   0:01.31 rcu_sched
   12 root        rt   0      0      0      0  S   0.0   0.0   0:00.04 migration/0

```

图 7.1 获取 PID

使用 insmod 插入模块，并传参。如图所示，本例中模块名为“vma_test.ko”，pid 为 1303，则插入模块命令为：

```
insmod vma_test.ko pid=1303
```

```
root@bogon:/code/vma# insmod vma_test.ko pid=1303
```

图 7.2 插入模块

5. 查看程序打印信息

通过 `dmesg` 命令查看 VMA 信息。如下图所示为本例内容。

```
[ 1836.985042] Examining vma's for pid=1303, command=sshd
[ 1836.985044] mm_struct addr = 0x00000000cbc3cc8b
[ 1836.985044] vmas:
[ 1836.985045]   1:      ea555f6 55d3f19ca000 55d3f19d5000      45056
[ 1836.985046]   2:      77d3ea30 55d3f19d5000 55d3f1a4c000      487424
[ 1836.985047]   3:      a0b7fdef 55d3f1a4c000 55d3f1a93000      290816
[ 1836.985048]   4:      fb95981 55d3f1a94000 55d3f1a97000      12288
[ 1836.985048]   5:      ae9faf36 55d3f1a97000 55d3f1a98000      4096
[ 1836.985049]   6:      9bb54d72 55d3f1a98000 55d3f1a9c000      16384
[ 1836.985049]   7:      9dcca1db 55d3f241f000 55d3f2482000      405504
[ 1836.985050]   8:      220ca31d 55d3f2482000 55d3f24d2000      327680
[ 1836.985051]   9:      18ee473d 7f2163e08000 7f2163e0e000      24576
[ 1836.985051]  10:      d281b4dd 7f2163e0e000 7f2163e40000      204800
[ 1836.985052]  11:      d2fa51c3 7f2163e40000 7f2163e51000      69632
[ 1836.985052]  12:      9dcb5034 7f2163e51000 7f2163e54000      12288
[ 1836.985053]  13:      4161850d 7f2163e54000 7f2163e55000      4096
```

图 7.3 查看 VMA 信息

从上图可看到 `ssh` 进程包含许多 VMA 区域，以第一个 VMA 区域为例，其起始地址为 `0x55d3f19ca000`，结束地址为 `0x55d3f19d5000`，长度为 45056 字节。

如下图所示为从 `proc` 虚拟文件系统中查看相应进程第一个 VMA 的完整信息。

```
root@bogon:/code/vma# cat /proc/1303/smaps
55d3f19ca000-55d3f19d5000 r--p 00000000 08:01 1063368 /usr/sbin/sshd
Size: 44 kB
KernelPageSize: 4 kB
MMUPageSize: 4 kB
Rss: 44 kB
Pss: 8 kB
Shared_Clean: 44 kB
Shared_Dirty: 0 kB
Private_Clean: 0 kB
Private_Dirty: 0 kB
Referenced: 44 kB
Anonymous: 0 kB
LazyFree: 0 kB
AnonHugePages: 0 kB
ShmemPmdMapped: 0 kB
Shared_Hugetlb: 0 kB
Private_Hugetlb: 0 kB
Swap: 0 kB
SwapPss: 0 kB
Locked: 0 kB
THPeligible: 0
VmFlags: rd mr mw me dw sd
```

图 7.4 查看 VMA 信息

从以上两图中内容对比可知，本程序输出信息正确。

实验内容

- 1、运行本示例内核模块程序；
- 2、查询内核源代码或 vma 数据结构，尝试输出更多 VMA 信息。