操作系统实验一 Linux 进程管理及其扩展

71121117 马骁宇

2023.11.21

一、实验内容

（1）实现系统调用 int hide(pid_t pid, int on)，在进程 pid 有效的前提下，如果 on 置 1，进程被隐藏，用户无法通过 ps 或 top 观察到进程状态；如果 on 置 0 且此前为隐藏状态，则恢复正常状态。

（2）考虑权限问题，只有根用户才能隐藏进程。

（3）设计一个新的系统调用 int hide_user_processes(uid_t uid, char *binname)，参数 uid 为用户 ID 号，当 binname 参数为 NULL 时，隐藏该用户的所有进程；否则，隐藏二进制映像名为 binname 的用户进程。该系统调用应与 hide 系统调用共存。

（4）在/proc 目录下创建一个文件/proc/hidden，该文件可读可写，对应一个全局变量 hidden_flag，当 hidden_flag 为 0 时，所有进程都无法隐藏，即便此前进程被 hide 系统调用要求隐藏。只有当 hidden_flag 为 1 时，此前通过 hide 调用要求被屏蔽的进程才隐藏起来。（选做）

（5）在/proc 目录下创建一个文件/proc/hidden_process，该文件的内容包含所有被隐藏进程的 pid，各 pid 之间用空格分开。（选做）

二、实验目的

通过实验，加深理解进程控制块、进程队列等概念，了解进程管理的具体实施方法。

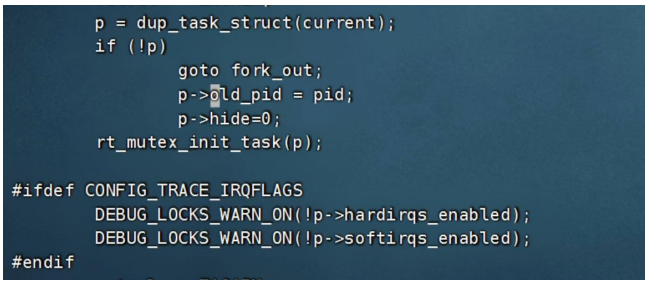
三、实验流程与结果展示

（一）实现系统调用 hide 与 hide_user_processes

（1）修改/usr/src/linux-2.6.21/include/linux/sched.h，在 task_structd 的末尾添加变量 hide 和 old_pid。

```
        struct pipe_inode_info *splice_pipe;
#ifdef  CONFIG_TASK_DELAY_ACCT
        struct task_delay_info *delays;
#endif
#ifdef CONFIG_FAULT_INJECTION
        int make_it_fail;
#endif
        int old_pid;
        int hide;
};
```

（2）修改/usr/src/linux-2.6.21/kernel/fork.c, 在 copy_process 函数加上 p->old_pid = pid; p->hide = 0; 进行初始化。



```
        p = dup_task_struct(current);
        if (!p)
                goto fork_out;
                p->old_pid = pid;
                p->hide=0;
        rt_mutex_init_task(p);

#ifdef CONFIG_TRACE_IRQFLAGS
        DEBUG_LOCKS_WARN_ON(!p->hardirqs_enabled);
        DEBUG_LOCKS_WARN_ON(!p->softirqs_enabled);
#endif
```

（3）修改系统调用 sys.c,在文件结尾添加新的系统调用 hide 和 hide_user_process

```
1.    asmlinkage int sys_hide(pid_t pid, int on){
2.        struct task_struct *p;
3.        struct task_struct * me = NULL;
4.        p = &init_task;
5.        do{
6.            if( pid == p->old_pid ){
7.                me = p;
8.                break;
9.            }
10.       }while((p = next_task(p)) && (p != &init_task));
11.       if(current->uid != 0 || me == NULL)
12.           return 0;
13.       if(on == 1){
14.           me->pid = 0;
15.           me->hide = 1;
16.       }
17.       else{
18.           if(me->hide == 1){
19.               me->pid = me->old_pid;
20.               me->hide = 0;
21.           }
22.       }
23.       return 0;
24.   }
25.
26.   asmlinkage int sys_hide_user_process(uid_t uid, char* binname){
27.       if(current->uid != 0)
28.           return 0;
29.       struct task_struct *p;
30.       p = &init_task;
31.       if(binname == NULL){
32.           do{
33.               if(p->uid == uid){
34.                   p->pid = 0;
35.                   p->hide = 1;
36.               }
37.           }while((p = next_task(p)) && (p != &init_task));
38.       }
39.       else{
40.           do{
41.               if(p->uid == uid){
42.                   int flag = 1;
43.                   int i = 0;
44.                   for(i = 0; (binname[i] != NULL && p->comm[i] !=NULL);
i++){
```

```
                        if(binname[i] != p->comm[i]){
                            flag = 0;
                            break;
                        }
                    }
                    if(flag == 1 && binname[i] == NULL && p->comm[i] == NU
LL){
                        p->pid = 0;
                        p->hide = 1;
                    }
                }
            }while((p = next_task(p)) && (p != &init_task));
        }
        return 0;
    }
```

（4）在/usr/src/linux-2.6.21/include/asm/unistd.h 中加入/usr/include/asm/unistd.h 中多出来的几个系统调用，并且加上自定义的系统调用，同样修改/usr/include/asm 下的 unistd.h，使两者系统调用相同。

```
#define __NR_epoll_pwait        319
#define __NR_utimensat          320
#define __NR_signalfd           321
#define __NR_timerfd            322
#define __NR_eventfd            323
#define __NR_fallocate          324
#define __NR_hide               325
#define __NR_hide_user_process  326
#ifdef __KERNEL__

#define NR_syscalls 327
```

（5）重新修改 sys.c，增加新系统调用

```
asmlinkage int sys_utimensat( void )
{
 return 0;
}
asmlinkage int sys_signalfd( void )
{
 return 0;
}
asmlinkage int sys_timerfd( void )
{
 return 0;
}
asmlinkage int sys_eventfd( void )
{
 return 0;
}
asmlinkage int sys_fallocate( void )
{
 return 0;
}
```

（6）修改 syscall_table，在/usr/src/linux-2.6.21/arch/i386/kernel/syscall_table.S 末尾添加新系统调用。

```
.long sys_utimensat
.long sys_signalfd
.long sys_timerfd
.long sys_eventfd
.long sys_fallocate
.long sys_hide
.long sys_hide_user_process
```

（7）重新编译内核

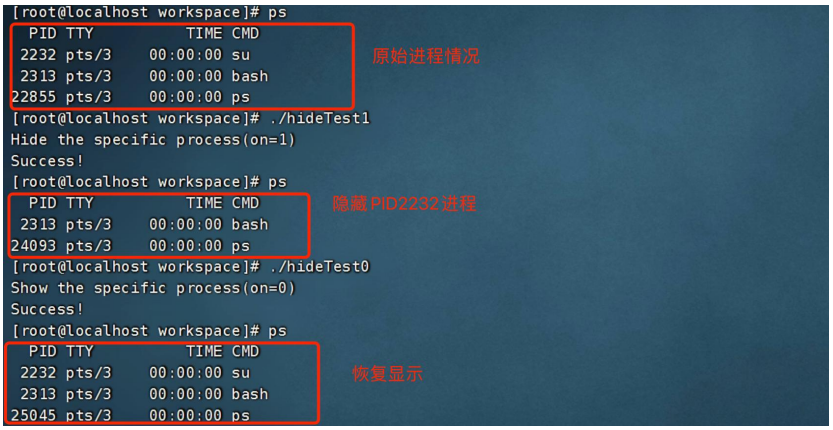
（二）测试 hide

（1）编写测试程序 hideTest1.c 与 hideTest0.c， 分别实现对进程 2232 的隐藏与恢复显示。

```
//hideTest1.c 隐藏某一特定进程，即 on=1
#include<linux/unistd.h>
#include<sys/syscall.h>
#include<stdio.h>
#define __NR_hide 325
main(){
    printf("Hide the specific process(on=1)\n");
    syscall(__NR_hide, 2232, 1 );
    printf( "Success!\n" );
}
```

```
//hideTest0.c 显示某一特定进程，即 on=0
#include<linux/unistd.h>
#include<sys/syscall.h>
#include<stdio.h>
#define __NR_hide 325
main(){
    printf("Show the specific process(on=0)\n");
    syscall(__NR_hide, 2232, 0 );
    printf( "Success!\n" );
}
```

（2）编译后运行测试结果

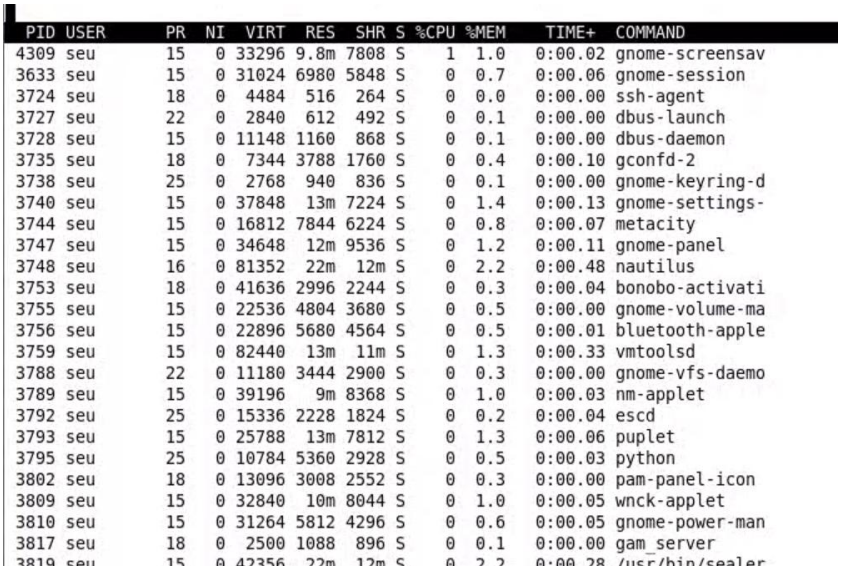


（三）测试 hide_user_processes

（1）编写 hideSeuTest.c 测试程序

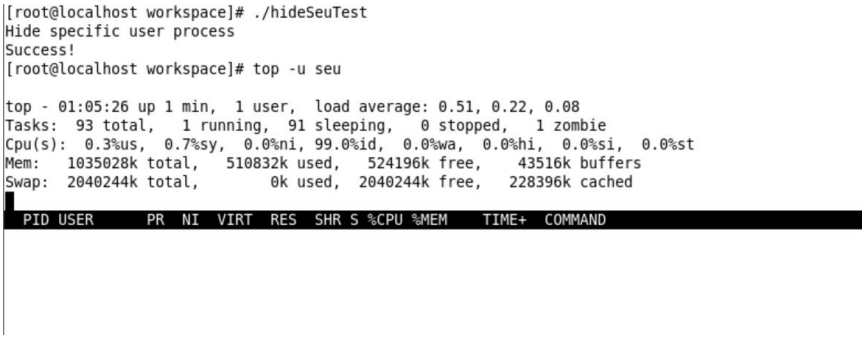
```c
// 隐藏当前用户 seu 的所有进程
#include<linux/unistd.h>
#include<sys/syscall.h>
#include<stdio.h>
#define __NR_hide_user_process 326
main(){
    printf("Hide specific user process\n");
    syscall(__NR_hide_user_process, 500, NULL );
    printf( "Success!\n");
}
```

（2）top -u seu 展示 seu 用户的所有进程



```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 4309 seu       15   0 33296 9.8m 7808 S    1  1.0   0:00.02 gnome-screensav
 3633 seu       15   0 31024 6980 5848 S    0  0.7   0:00.06 gnome-session
 3724 seu       18   0  4484  516  264 S    0  0.0   0:00.00 ssh-agent
 3727 seu       22   0  2840  612  492 S    0  0.1   0:00.00 dbus-launch
 3728 seu       15   0 11148 1160  868 S    0  0.1   0:00.00 dbus-daemon
 3735 seu       18   0  7344 3788 1760 S    0  0.4   0:00.10 gconfd-2
 3738 seu       25   0  2768  940  836 S    0  0.1   0:00.00 gnome-keyring-d
 3740 seu       15   0 37848  13m 7224 S    0  1.4   0:00.13 gnome-settings-
 3744 seu       15   0 16812 7844 6224 S    0  0.8   0:00.07 metacity
 3747 seu       15   0 34648  12m 9536 S    0  1.2   0:00.11 gnome-panel
 3748 seu       16   0 81352  22m  12m S    0  2.2   0:00.48 nautilus
 3753 seu       18   0 41636 2996 2244 S    0  0.3   0:00.04 bonobo-activati
 3755 seu       15   0 22536 4804 3680 S    0  0.5   0:00.00 gnome-volume-ma
 3756 seu       15   0 22896 5680 4564 S    0  0.5   0:00.01 bluetooth-apple
 3759 seu       15   0 82440  13m  11m S    0  1.3   0:00.33 vmtoolsd
 3788 seu       22   0 11180 3444 2900 S    0  0.3   0:00.00 gnome-vfs-daemo
 3789 seu       15   0 39196   9m 8368 S    0  1.0   0:00.03 nm-applet
 3792 seu       25   0 15336 2228 1824 S    0  0.2   0:00.04 escd
 3793 seu       15   0 25788  13m 7812 S    0  1.3   0:00.06 puplet
 3795 seu       25   0 10784 5360 2928 S    0  0.5   0:00.03 python
 3802 seu       18   0 13096 3008 2552 S    0  0.3   0:00.00 pam-panel-icon
 3809 seu       15   0 32840  10m 8044 S    0  1.0   0:00.05 wnck-applet
 3810 seu       15   0 31264 5812 4296 S    0  0.6   0:00.05 gnome-power-man
 3817 seu       18   0  2500 1088  896 S    0  0.1   0:00.00 gam_server
 3819 seu       15   0 42356  22m  12m S    0  2.2   0:00.28 /usr/bin/sealer
```

（3）编译执行 hideSeuTest.c 后再次查看 seu 用户进程，发现全部被隐藏



```
[root@localhost workspace]# ./hideSeuTest
Hide specific user process
Success!
[root@localhost workspace]# top -u seu

top - 01:05:26 up 1 min,  1 user,  load average: 0.51, 0.22, 0.08
Tasks:  93 total,   1 running,  91 sleeping,   0 stopped,   1 zombie
Cpu(s):  0.3%us,  0.7%sy,  0.0%ni, 99.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   1035028k total,   510832k used,   524196k free,    43516k buffers
Swap:  2040244k total,        0k used,  2040244k free,   228396k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
```

四、实验体会

在本次实验中，我加深理解进程的相关概念，同时学习了如何添加系统调用并通过程序进行测试，以实现隐藏进程的功能。在实验中，我主要的困难在于对于设计好的系统调用如何在测试程序中使用并不熟悉，好在经过不断测试与验证成功掌握了相关方法。