# 操作系统专题实践 - 实验1

**09019216 黄启越**



# Linux进程管理及其扩展

## 实验目的、实验内容、具体要求

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。

内容：实现一个系统调用hide，使得可以根据指定的参数隐藏进程，使用户无法使用ps或top观察到进程状态。

具体要求：

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root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。
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。

## 设计思路

此次实验要求在Linux源码上进行修改，因此需要先了解从Linux内核源码进行编译的过程。

实验要求新增两个系统调用`hide`以及`hide_user_processes`。由于需要记录每个进程是否被隐藏，需要为管理进程信息的结构体`task_struct`增加一个标记位`cloak`。这两个系统调用都只有root用户才有权限调用，因此应当检查uid是否为0（即root用户），从而实现权限控制。如果权限允许，则修改对应的`cloak`标记位。`ps`等工具在列举当前进程时访问的是`\proc`目录，需要修改伪文件系统`procfs`枚举当前进程的函数的实现，从而达到过滤被隐藏进程的目的。

要求4、5则在此基础上，在`procfs`中新增了文件，还要修改读、写回调函数来定义对`/proc/hidden`进行读写和对`/proc/hidden_process`读取时的行为。相应地，`hide`以及`hide_user_processes`系统调用的实现也要新增对`hidden_flag`状态的判断。

## 主要数据结构及其说明

增加系统调用的实验使用了结构体`task_struct`，并新增了成员`cloak`用于记录进程当前隐藏的状态。

## 编译 Linux 内核

1. 在VMWare里安装Ubuntu，保证足够的磁盘空间（这里分配的空闲空间约30GB）。
2. Download the latest source code from https://www.kernel.org/ . 我下载的是`linux-5.13.10.tar.xz`。建议这里下载的内核版本应新于第一步安装的Ubuntu的内核版本。
3. 
```
sudo apt-get install vim git fakeroot build-essential ncurses-dev xz-utils libssl-dev bc flex libelf-dev bison
```
4. 解压源代码

```
cd Desktop
xz -d <filename>
tar -xavf <filename>
```

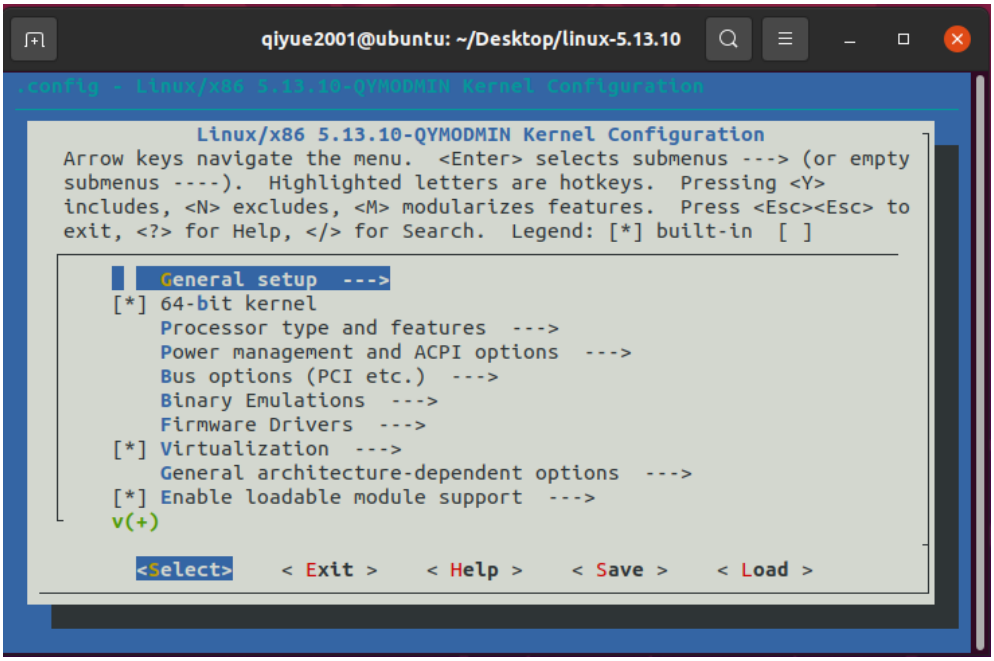
   得到`linux-5.13.10`目录。
5. 编辑配置文件。`make mrproper`可以清理之前编译残留的文件，使编译目录干净。

```
cd linux-5.13.10
make mrproper
```

   编辑`Makefile`，可以定义自己的内核版本号。`vim Makefile`，修改“EXTRAVERSION = <YOUR_EXTRAVERSION>”，这里我设定的是`EXTRAVERSION =-QYMODMIN`。

   再进行配置。由于不知道哪些模块是系统所必需的，可以拷贝正在运行的内核配置文件作为模板。`make oldconfig`或`make menuconfig`等都可以用于编辑配置文件，这里选择带有图形界面的`menuconfig`。有经验的用户可以在这一步进行裁剪，可以大大缩短编译时间。可以裁剪掉部分网络模块、虚拟化模块、文件系统等。

```
sudo cp /boot/config-5.11.0-25-generic .config
make menuconfig
```




5. 内核编译阶段。具体所用时间取决于CPU性能和配置。可以加上`-j#`参数进行多线程编译，`#`为线程数，可从`nproc`知晓。

   我的用时，没有开启多线程编译（仅供参考）：

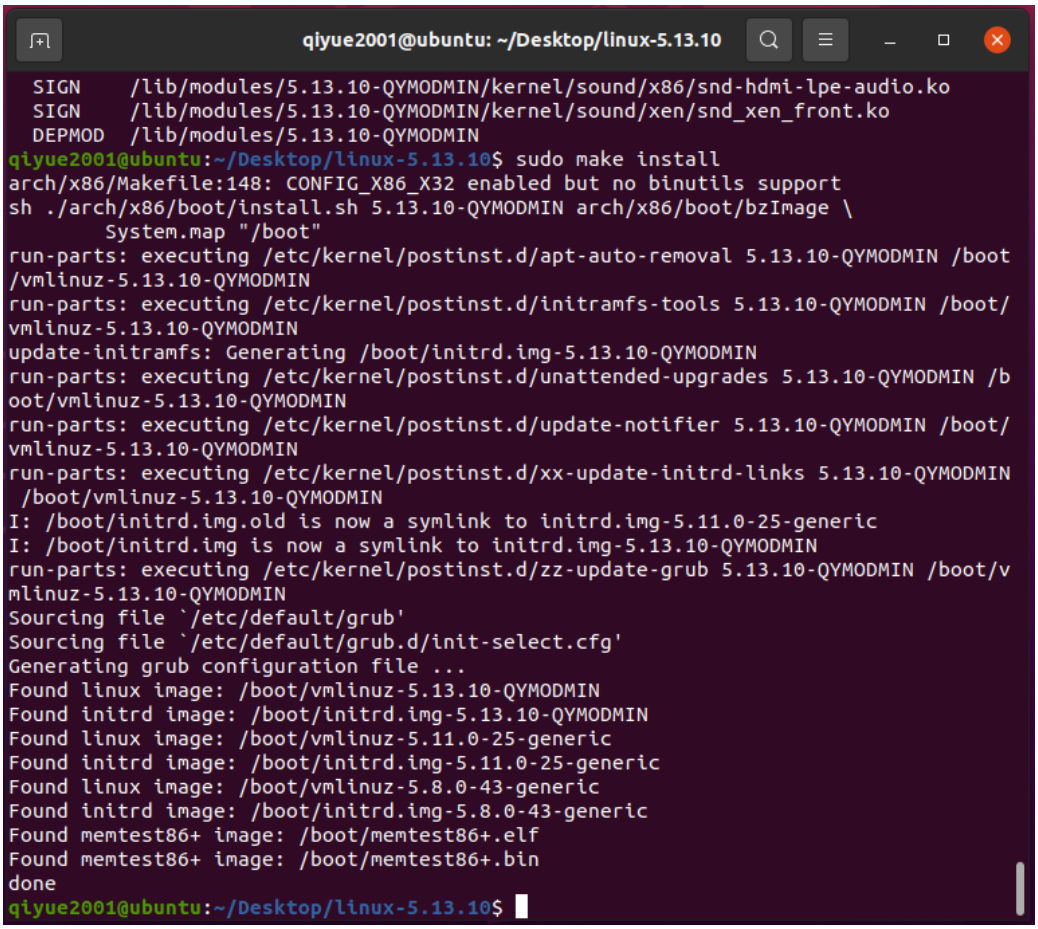
   `make`: 21:03-24:08

   `make modules_install`：约一分钟

   `make install`：不到一分钟

   编译后的目录约11GB。

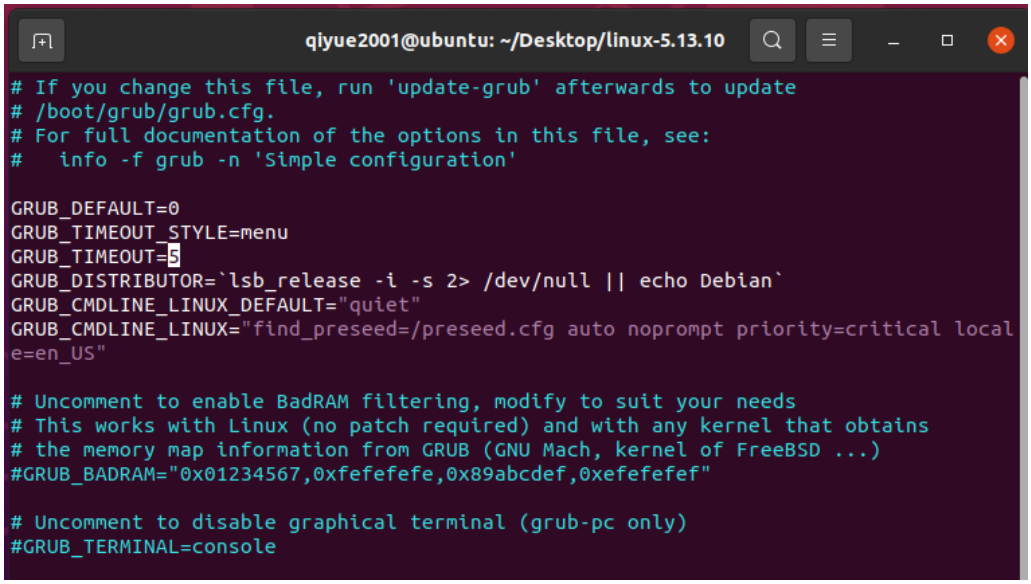
   只要不`make clean`（`make mrproper`），再次运行`make`将会进行增量编译，所用时间会大大缩短。

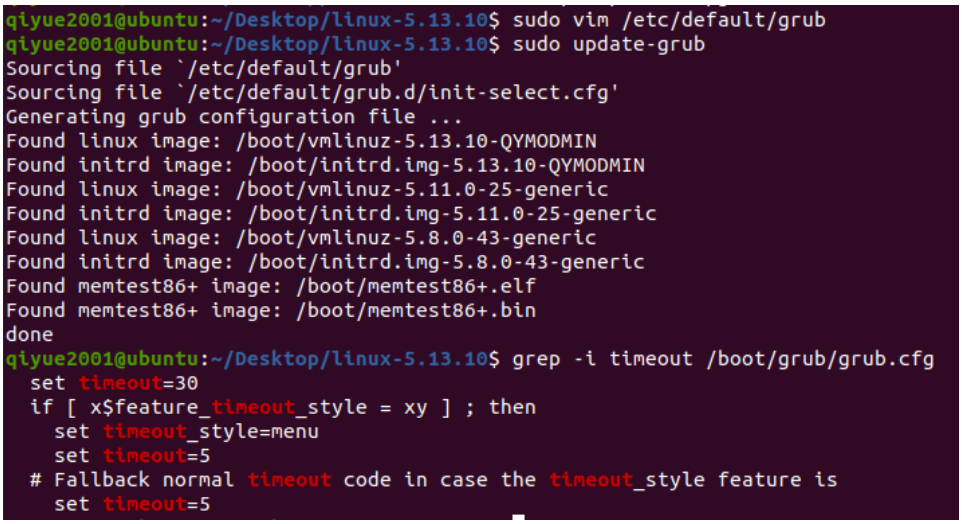


```
sudo make -j8
sudo make modules_install
sudo make install
```

6. 选择新内核启动

   需要编辑grub2选项，开启grub菜单。`sudo vim /etc/default/grub`，改为`GRUB_TIMEOUT_STYLE=menu`和`GRUB_TIMEOUT=5`。



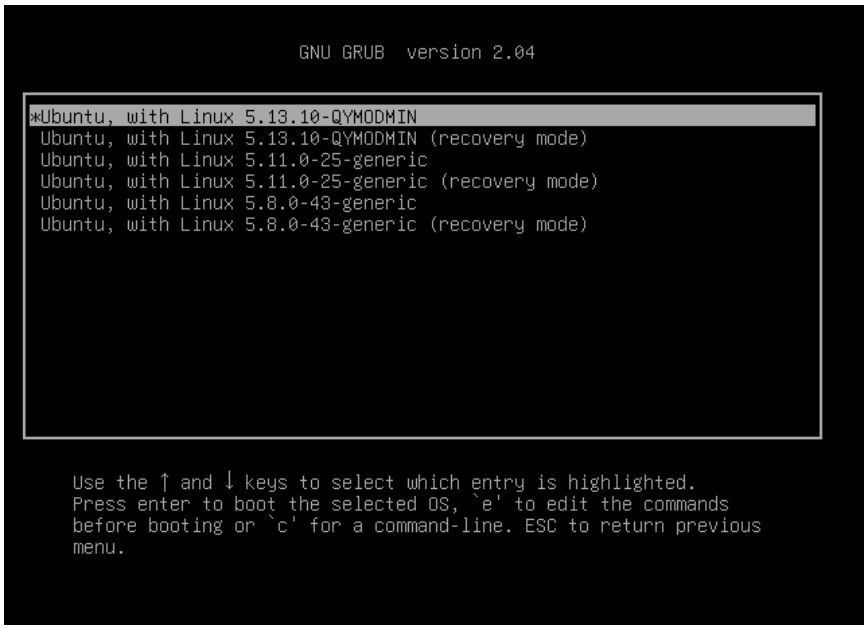




   执行

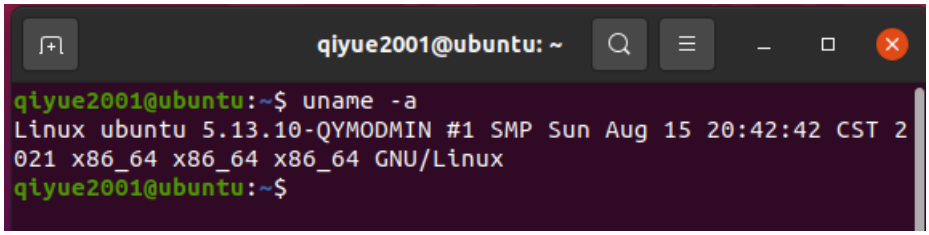
```
sudo update-grub
sudo reboot
```

   在第一步后可以`grep -i timeout /boot/grub/grub.cfg`验证是否更新了`grub.cfg`。

   在grub菜单中选择Advanced options for Ubuntu，在二级菜单中选择新编译的内核，启动。done！



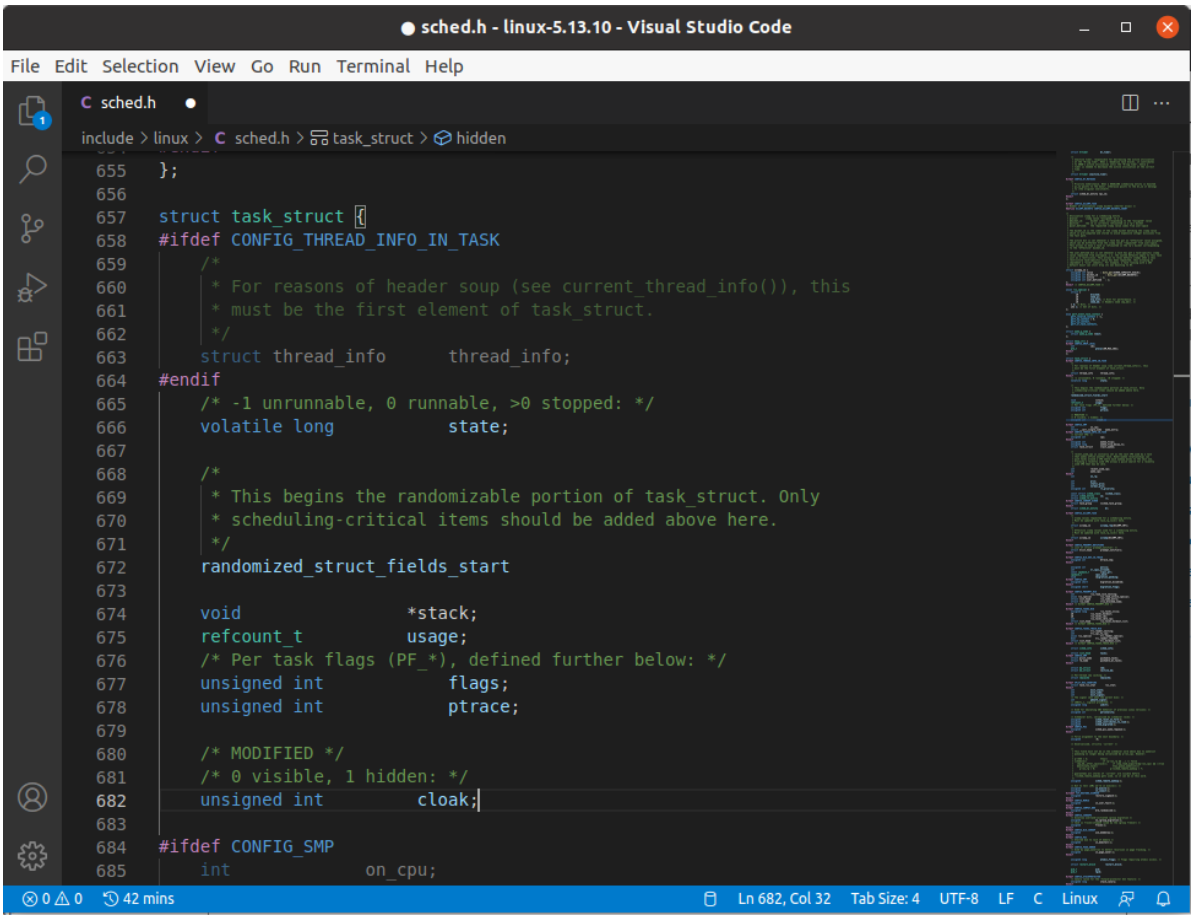



## 新增`hide`系统调用

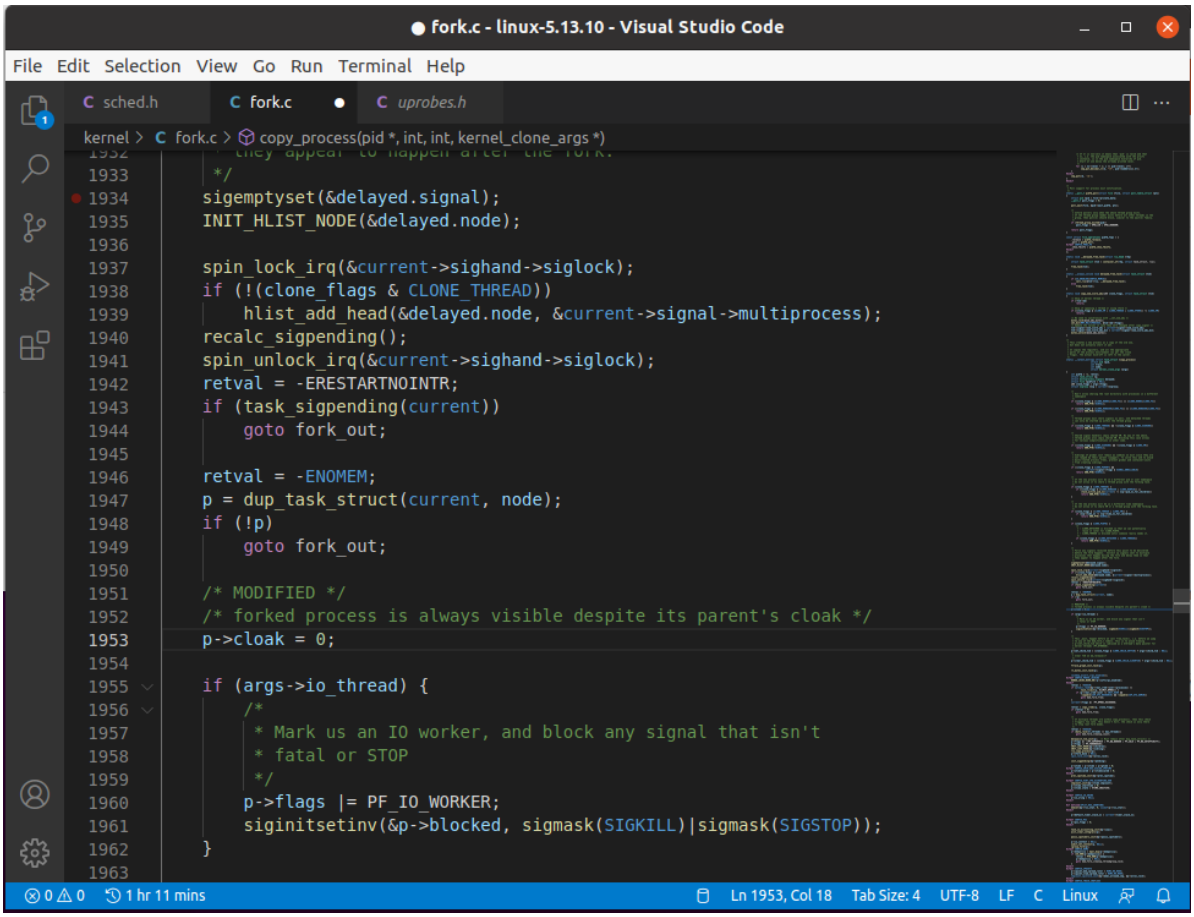
记得先备份源码！

## 修改`task_struct`

在`/include/linux/sched.h`中，为`task_stuct`新增成员变量。规定：0表示显示，1表示隐藏。

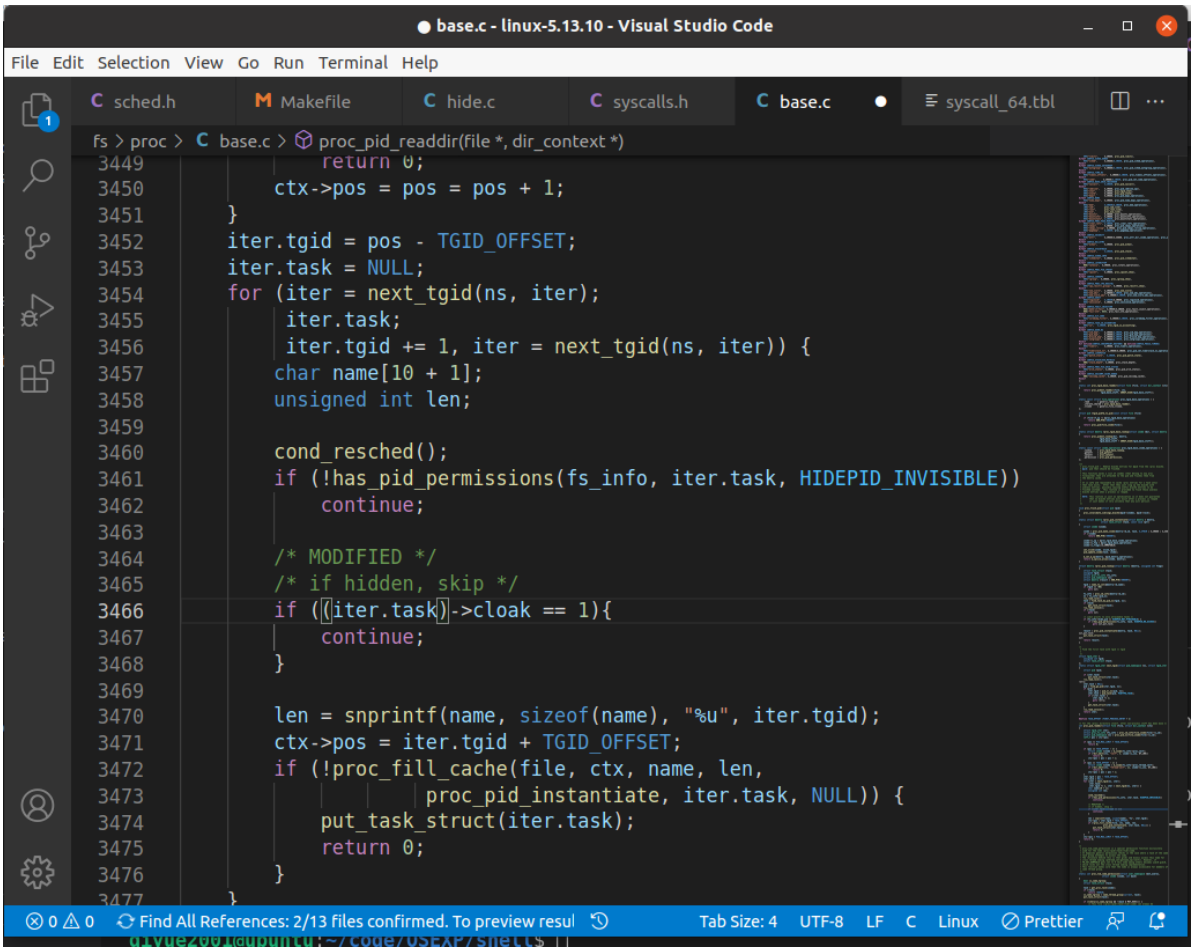


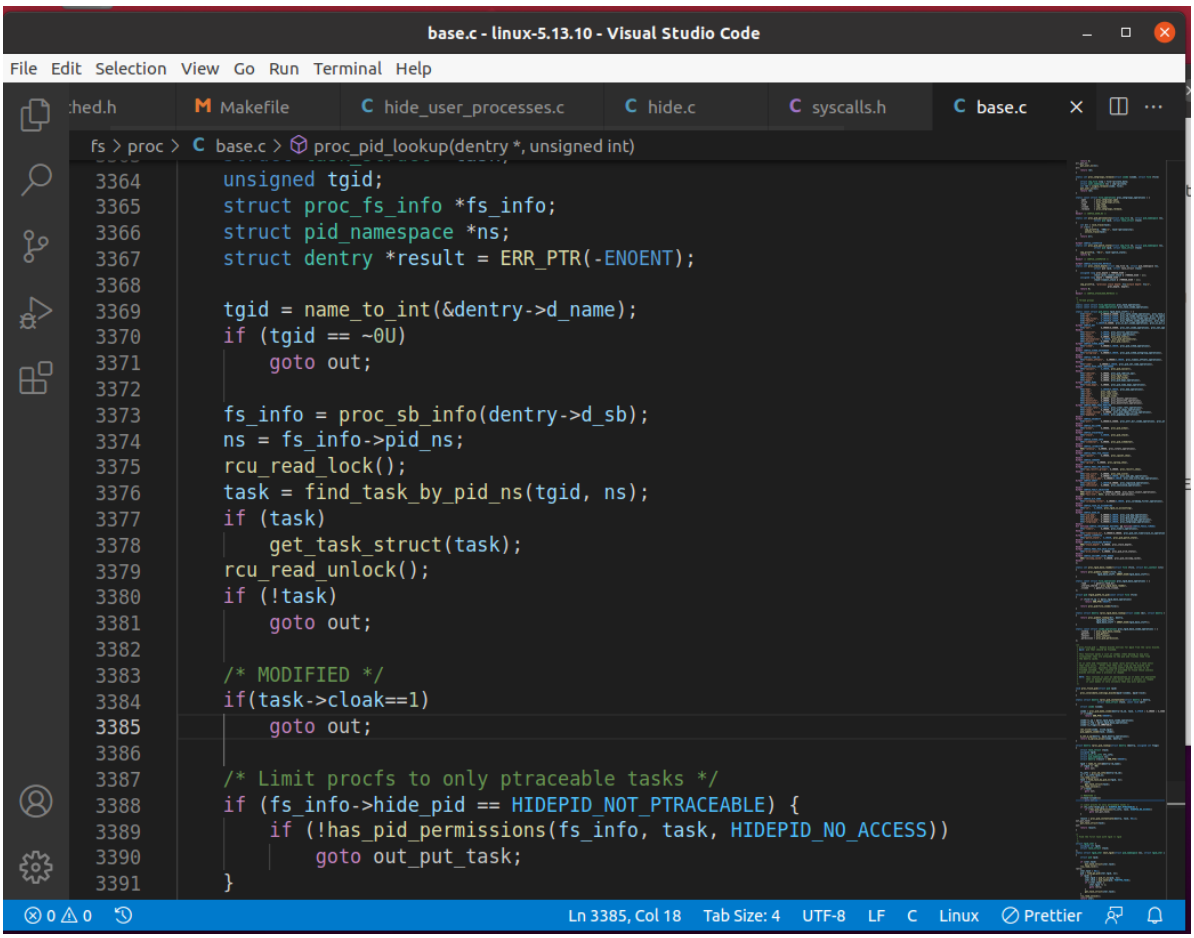

`kernel/fork.c`的`copy_process(pid *, int, int, kernal_clone_args *)`中，为fork的子进程设置clock为0。



## 修改procfs

在`fs\proc\base.c`的`int proc_pid_readdir(struct file *file, struct dir_context *ctx)`以及`struct dentry *proc_pid_lookup(struct dentry *dentry, unsigned int flags)`中，增加针对`cloak`的判断语句。






## 添加系统调用

1. 实现`hide`系统调用内容。

   在`kernel`目录下新增`hide.c`，输入以下内容。

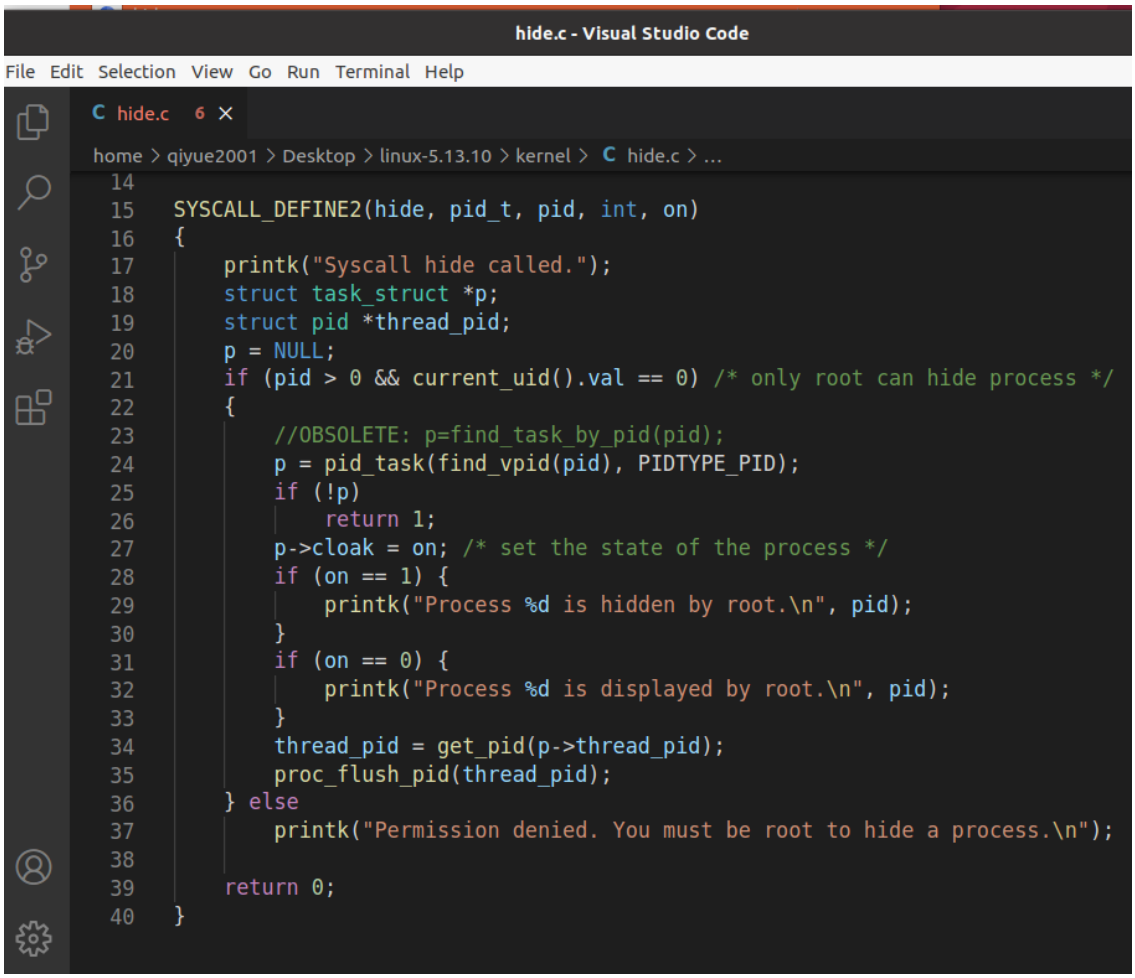
   `pid_task(find_vpid(pid), PIDTYPE_PID)`（较早版本内核为`find_task_by_pid(pid)`）可以通过pid获取进程`task_struct`。函数`proc_flush_pid(pid *)`（较早版本内核为`proc_flush_task(struct task_struct *)`）用于清空VFS层的缓冲，解除已有的dentry项。

   `current_uid().val`（较早版本的内核为`current->uid`）用于获取uid。只有root用户，即uid=0才可隐藏进程。

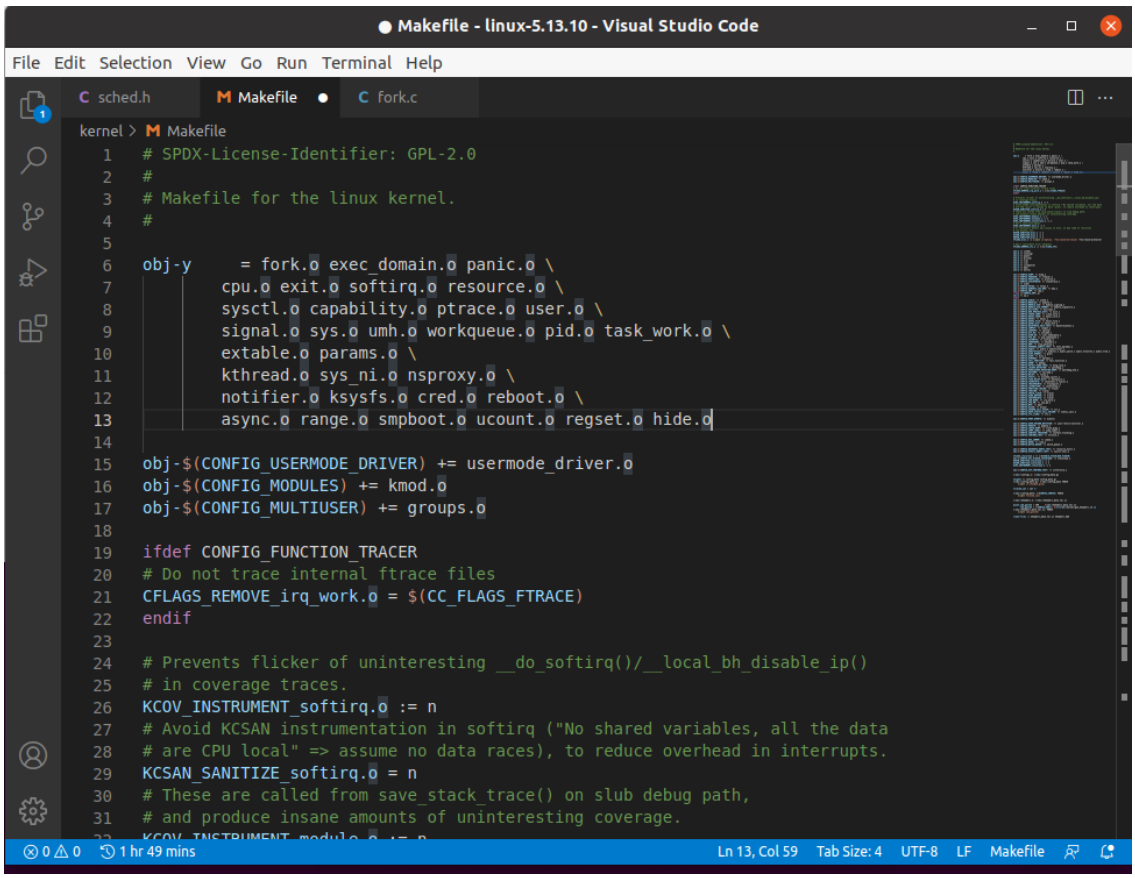
   宏`SYSCALL_DEFINE2`的`2`表示参数个数，参数以`(type, name)`格式呈现。

```
/**
 * MODIFIED
 * Implementation of system call `hide`.
 */

#include <linux/syscalls.h>
#include <linux/kernel.h>
#include <linux/linkage.h>
#include <linux/types.h>
#include <linux/sched.h>
#include <linux/pid.h>
#include <linux/proc_fs.h>
#include <linux/cred.h>

SYSCALL_DEFINE2(hide, pid_t, pid, int, on)
{
    printk("Syscall hide called.");
```

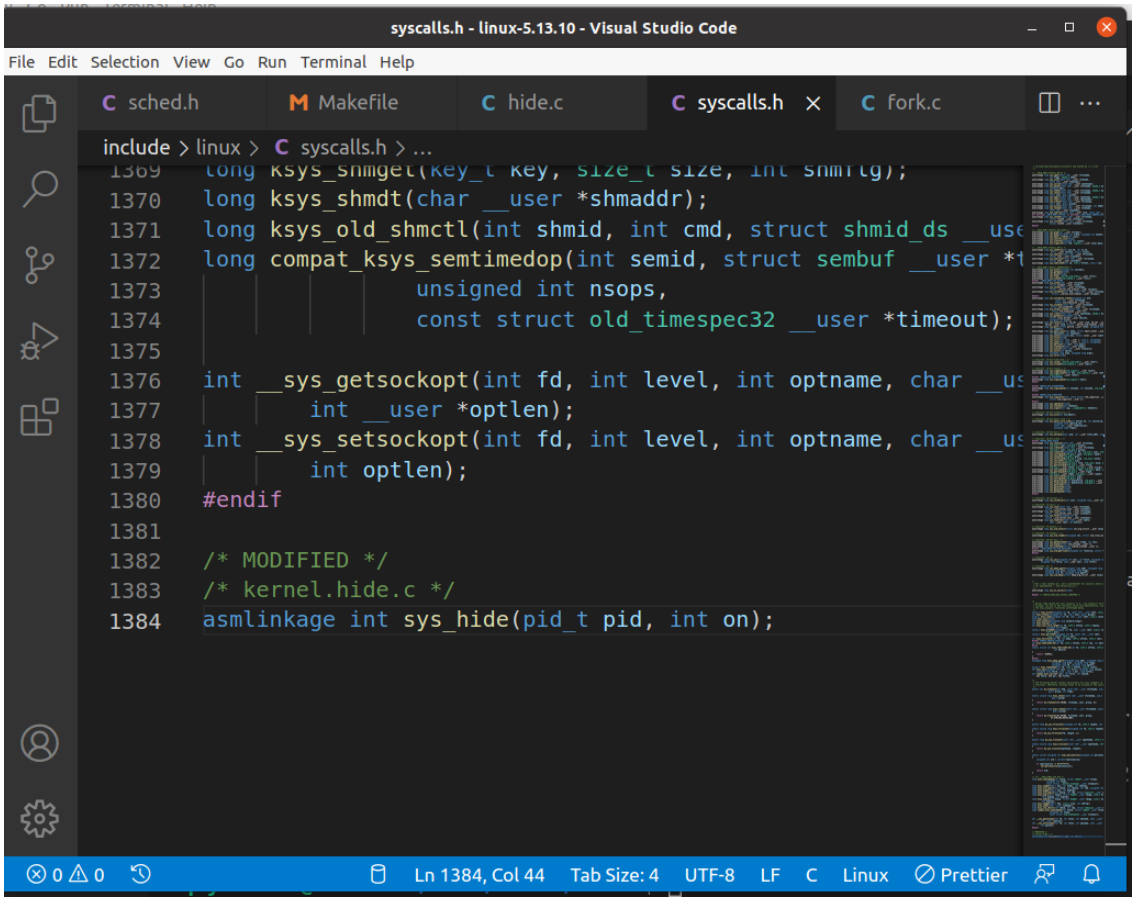
```c
    struct task_struct *p;
    struct pid *thread_pid;
    p = NULL;
    if (pid > 0 && current_uid().val == 0) /* only root can hide process */
    {
        //OBSOLETE: p=find_task_by_pid(pid);
        p = pid_task(find_vpid(pid), PIDTYPE_PID);
        if (!p)
            return 1;
        p->cloak = on; /* set the state of the process */
        if (on == 1) {
            printk("Process %d is hidden by root.\n", pid);
        }
        if (on == 0) {
            printk("Process %d is displayed by root.\n", pid);
        }
        thread_pid = get_pid(p->thread_pid);
        proc_flush_pid(thread_pid);
    } else
        printk("Permission denied. You must be root to hide a process.\n");

    return 0;
}
```




2. `kernel/Makefile`中增加`hide.o`

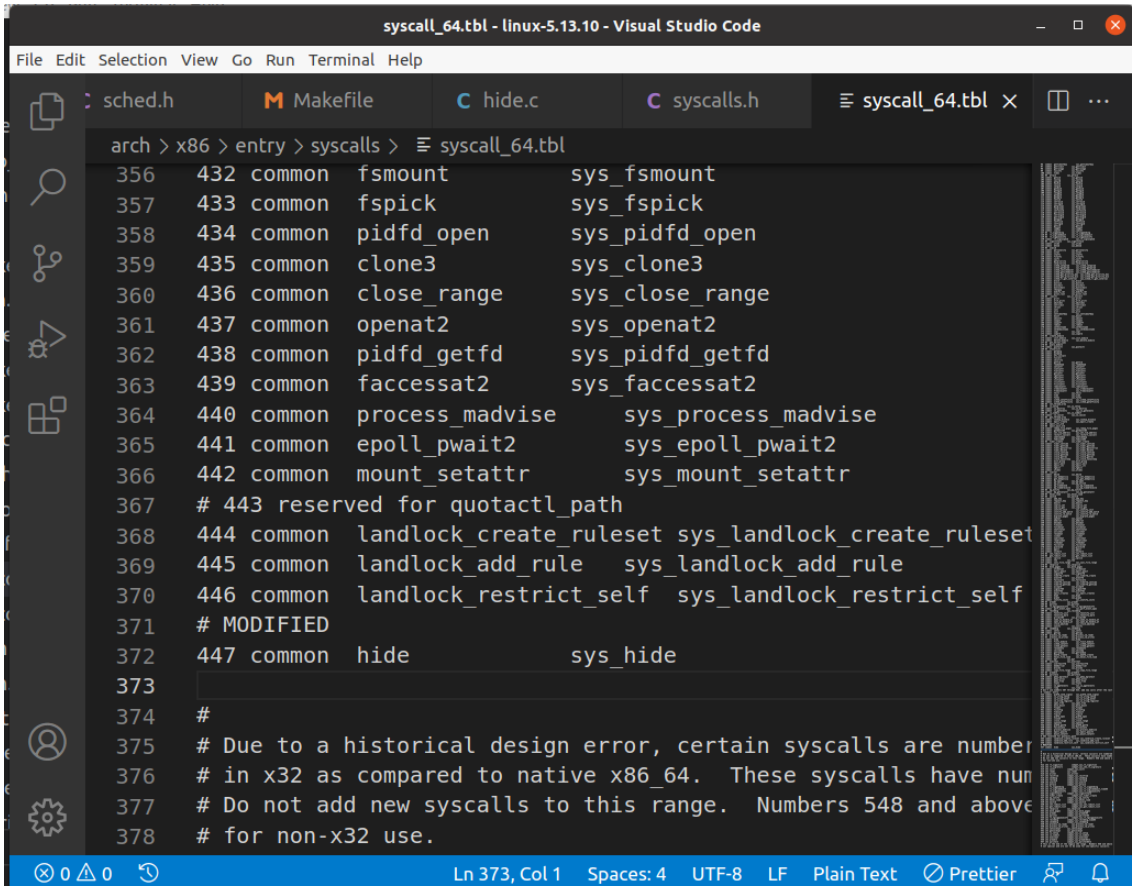



3. 在 `include/linux/syscalls.h` 中添加函数原型 `asmlinkage int sys_hide(pid_t pid, int on)`。




4. 在 `arch/x86/entry/syscalls/syscall_64.tbl` 的系统调用表中添加 `447 common  hide sys_hide`。注意找准位置，如图所示，增加的系统调用应当在x32系统调用之前。

注意：这里仅仅为x86_64架构增加了系统调用表项。如果要为i386架构增加，请在`arch/x86/entry/syscalls/syscall_32.tbl`添加表项。如果要为其他架构添加表项，请前往`include/uapi/asm-generic/unistd.h`。




## 编写测试程序及测试

重新编译安装。增量编译速度会明显快于第一次编译。推荐使用`-s`选项（`sudo make -s -j8`），不会打印正常日志，能更容易地发现编译错误。可以重启后观察`uname -a`是否显示为最近一次编译的结果，确定新内核是否已经成功安装。

编辑`hide_test.c`，输入如下内容（要将宏定义的`SYSCALL_NUM`修改为之前自己定义的系统调用号）：

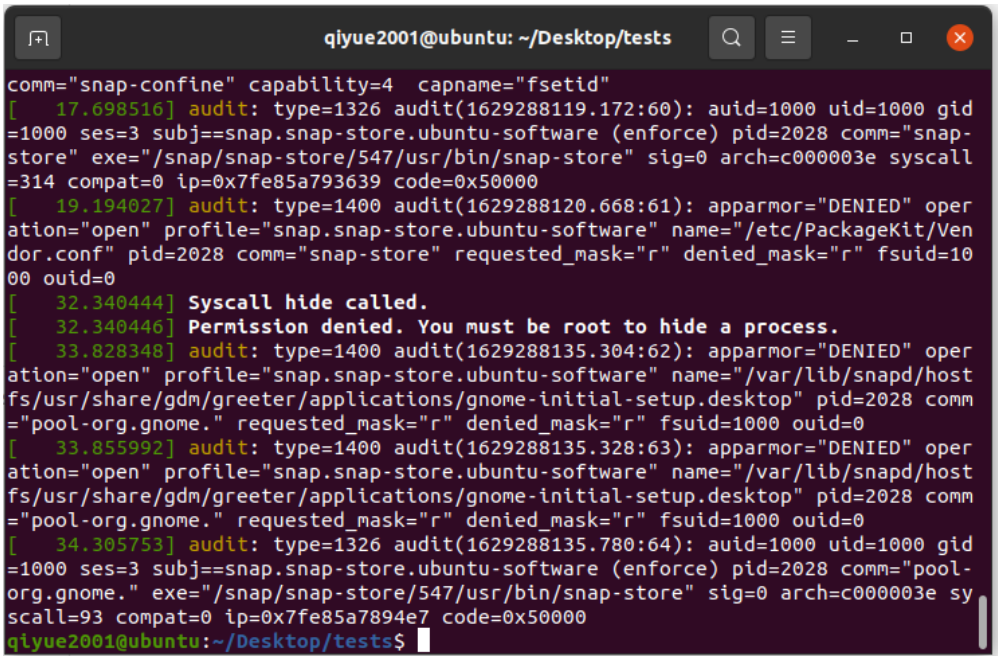
```
/**

This program tests `hide` system call.

**/

#include <stdio.h>
#include <sys/syscall.h>
#include <unistd.h>

#define SYSCALL_NUM 447

int
main()
{
    pid_t pid = 1;
    int on = 1;
```

```
    syscall(SYSCALL_NUM, pid, on);
    return 0;
}
```

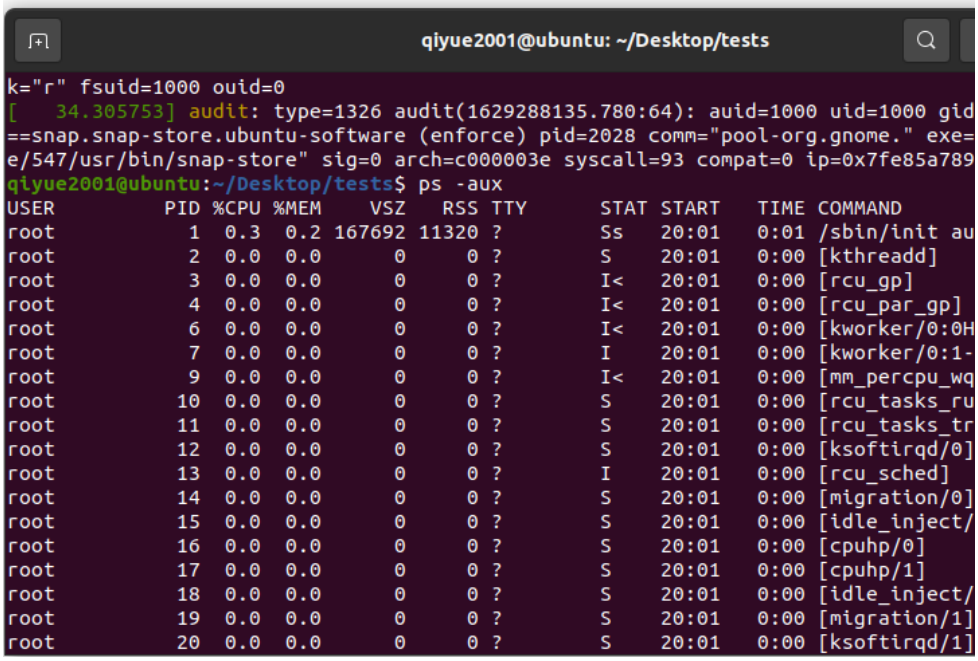
使用gcc编译。首先用非root用户测试。

```
gcc hide_test.c -o hide_test
sudo chmod +x hide_test
./hide_test
dmesg
```

可以看到，`printk`打印了相关内核消息。

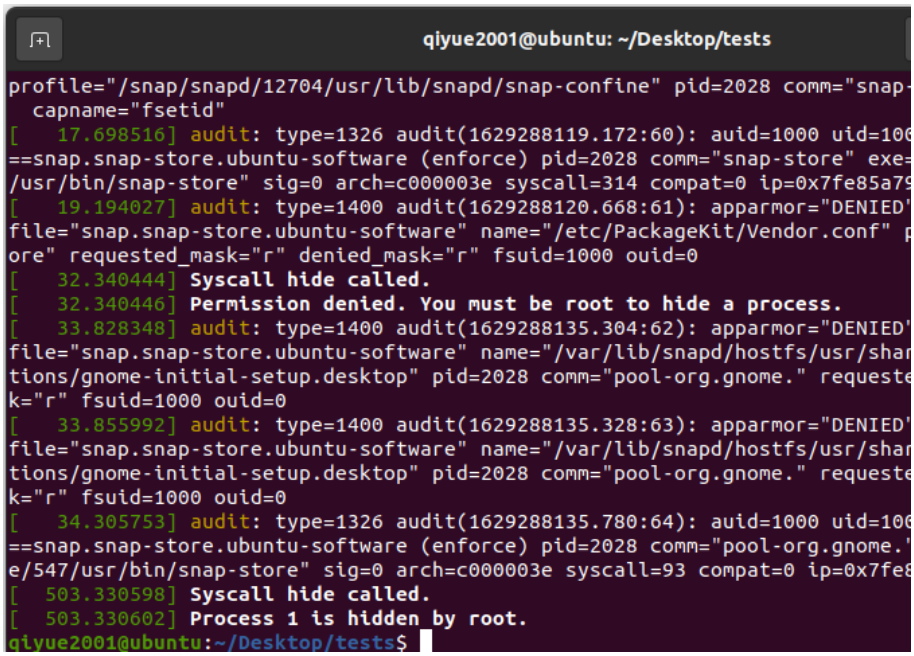


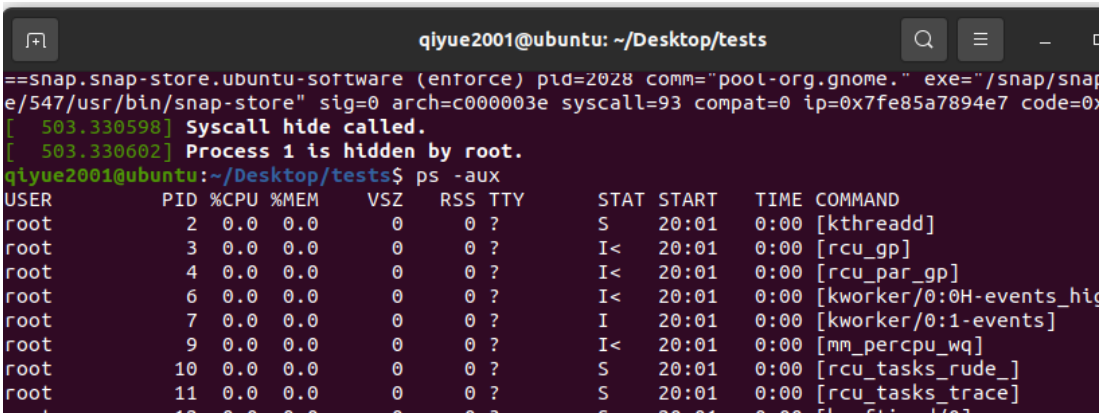

使用`ps -aux`依然可以看到PID为1的进程。




现在，切换到root用户，重复上述实验。

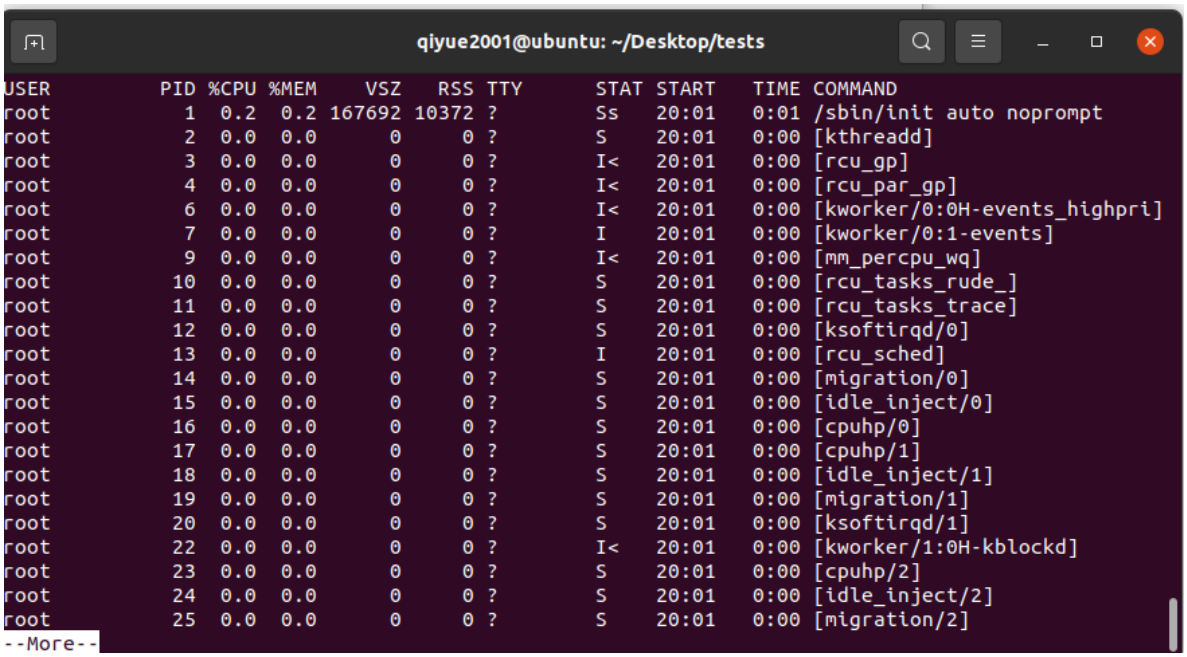
```
sudo ./hide_test
dmesg
ps -aux|more
```




可见打印了操作成功的内核消息，PID为1的进程确实被隐藏了。

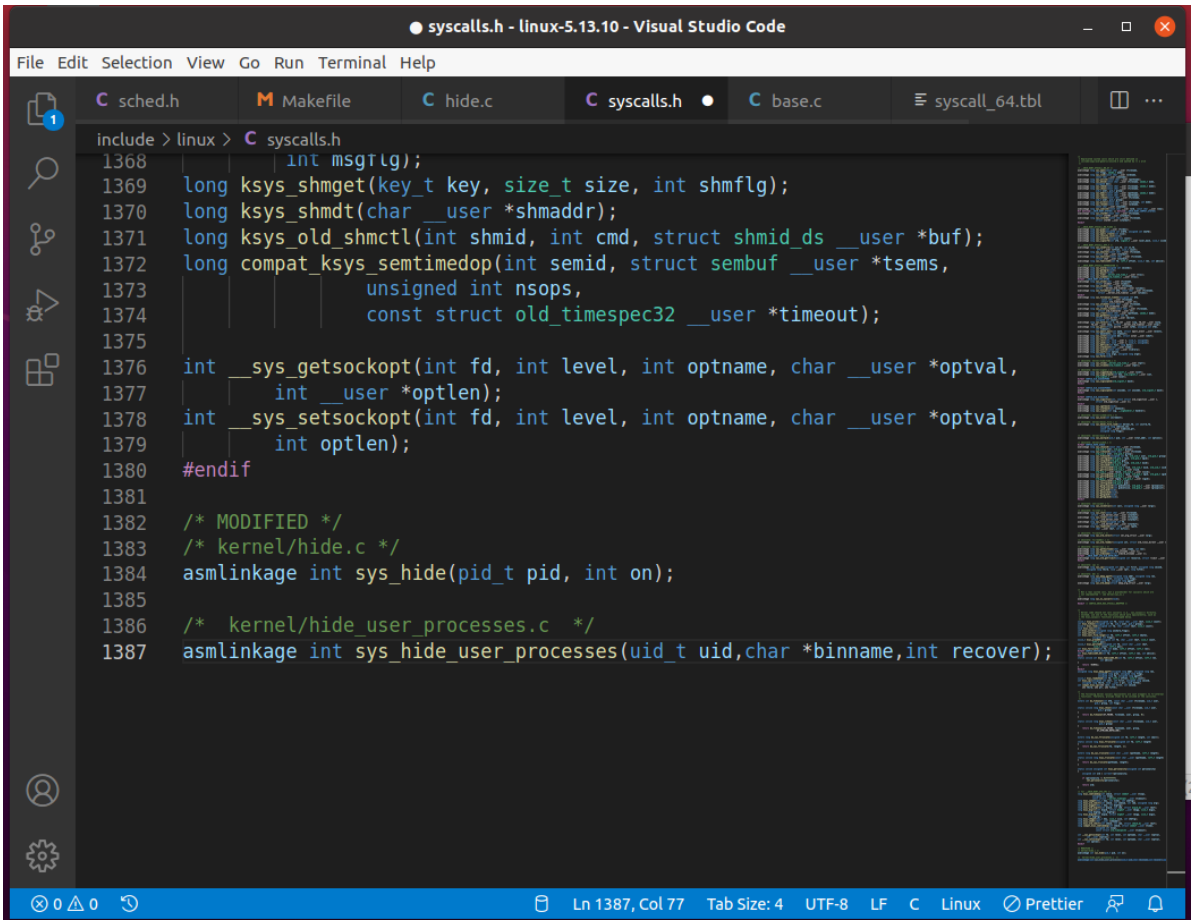
再测试将隐藏的进程恢复显示。将测试程序中的`int on = 1;`改为`int on = 0;`，重新编译，重复测试步骤。




可见被隐藏的PID=1的进程又重新显示了出来。根据START可知，并没有启动一个新的进程，而仅仅改变了原有进程的可见性。实验成功。

## 新增`hide_user_processes`系统调用

新增系统调用同样需要修改`include/linux/syscalls.h`、`arch/x86/entry/syscalls/syscall_64.tbl`、`kernel/Makefile`。



syscall_64.tbl - linux-5.13.10 - Visual Studio Code

File Edit Selection View Go Run Terminal Help

sched.h | Makefile | hide.c | syscalls.h | base.c | syscall_64.tbl

arch > x86 > entry > syscalls > syscall_64.tbl

```
427 common  io_uring_register   sys_io_uring_register
428 common  open_tree           sys_open_tree
429 common  move_mount          sys_move_mount
430 common  fsopen              sys_fsopen
431 common  fsconfig            sys_fsconfig
432 common  fsmount             sys_fsmount
433 common  fspick              sys_fspick
434 common  pidfd_open          sys_pidfd_open
435 common  clone3              sys_clone3
436 common  close_range         sys_close_range
437 common  openat2             sys_openat2
438 common  pidfd_getfd         sys_pidfd_getfd
439 common  faccessat2          sys_faccessat2
440 common  process_madvise     sys_process_madvise
441 common  epoll_pwait2        sys_epoll_pwait2
442 common  mount_setattr       sys_mount_setattr
# 443 reserved for quotactl_path
444 common  landlock_create_ruleset sys_landlock_create_ruleset
445 common  landlock_add_rule   sys_landlock_add_rule
446 common  landlock_restrict_self  sys_landlock_restrict_self
# MODIFIED
447 common  hide                sys_hide
448 common  hide_user_processes   sys_hide_user_processes

#
# Due to a historical design error, certain syscalls are numbered differently
# in x32 as compared to native x86_64.  These syscalls have numbers 512-547.
# Do not add new syscalls to this range.  Numbers 548 and above are available
# for non-x32 use.
```

⊗ 0 ⚠ 0 | Ln 373, Col 58 | Spaces: 4 | UTF-8 | LF | Plain Text | Prettier

● Makefile - linux-5.13.10 - Visual Studio Code

File Edit Selection View Go Run Terminal Help

sched.h | Makefile ● | hide.c | syscalls.h | base.c | syscall_64.tbl

kernel > Makefile

```
# SPDX-License-Identifier: GPL-2.0
#
# Makefile for the linux kernel.
#

# MODIFIED
obj-y     = fork.o exec_domain.o panic.o \
            cpu.o exit.o softirq.o resource.o \
            sysctl.o capability.o ptrace.o user.o \
            signal.o sys.o umh.o workqueue.o pid.o task_work.o \
            extable.o params.o \
            kthread.o sys_ni.o nsproxy.o \
            notifier.o ksysfs.o cred.o reboot.o \
            async.o range.o smpboot.o ucount.o regset.o hide.o \
            hide_user_processes.o

obj-$(CONFIG_USERMODE_DRIVER) += usermode_driver.o
obj-$(CONFIG_MODULES) += kmod.o
obj-$(CONFIG_MULTIUSER) += groups.o

ifdef CONFIG_FUNCTION_TRACER
# Do not trace internal ftrace files
CFLAGS_REMOVE_irq_work.o = $(CC_FLAGS_FTRACE)
endif

# Prevents flicker of uninteresting __do_softirq()/__local_bh_disable_ip()
# in coverage traces.
KCOV_INSTRUMENT_softirq.o := n
# Avoid KCSAN instrumentation in softirq ("No shared variables, all the data
```

⊗ 0 ⚠ 0 | Ln 15, Col 30 | Tab Size: 4 | UTF-8 | LF | Makefile | Pretti

这一实验与新增`hide`系统调用的实验比较相近，故只列出系统调用实现的代码（`kernel/hide_user_processes.c`）。

```
/**
 * MODIFIED
 * Implementation of system call `hide_user_processes`
*/

#include <linux/syscalls.h>
#include <linux/kernel.h>
#include <linux/linkage.h>
#include <linux/types.h>
#include <linux/sched.h>
#include <linux/pid.h>
#include <linux/proc_fs.h>
#include <linux/cred.h>
#include <linux/string.h>

SYSCALL_DEFINE3(hide_user_processes, uid_t, uid, char *, binname, int, recover)
{
    struct task_struct *p = NULL;

    if (current_uid().val != 0) { /* only root can call */
        printk("Permission denied. Only root can call hide_user_processes.\n");
        return 1;
    }

    if (recover == 0) /* recover = 0: allow root to hide processes */
    {
        if (binname == NULL)
        /* if null, hide all processes of the given uid */
        {
            for_each_process (p) {
                if (p->cred->uid.val == uid) {
                    p->cloak = 1;
                    proc_flush_pid(get_pid(p->thread_pid));
                }
            }
            printk("All processes of uid %d are hidden.\n", uid);
        } else /* otherwise, hide the process with the given name */
        {
            char kbinname[TASK_COMM_LEN];
            long len = strncpy_from_user(kbinname, binname, TASK_COMM_LEN);
            kbinname[TASK_COMM_LEN - 1] = '\0';
            if(unlikely(len < 0)){  /* unable to copy from user space */
                printk("Unable to do strncpy_from_user");
                return 2;
            }
            for_each_process (p) {
                char s[TASK_COMM_LEN];
                get_task_comm(s, p);  /* get name("comm") of process */
                if (p->cred->uid.val == uid && strncmp(s, kbinname,
TASK_COMM_LEN) == 0) {
                    p->cloak = 1;
                    printk("Process %s of uid %d is hidden.\n",
                           kbinname, uid);
                    proc_flush_pid(get_pid(p->thread_pid));
```

```
                }
            }
        }
    }

    /* recover != 0: display all of the processes, including previously hidden
ones */
    else {
        for_each_process (p) {
            p->cloak = 0;
        }
    }

    return 0;
}
```

以及测试程序代码`hide_user_processes.c`：

```
/**

This program tests `hide_user_processes` system call.

**/

#include<stdio.h>
#include<sys/syscall.h>
#include<unistd.h>

#define SYSCALL_NUM 448
#define MY_UID 0

int main()
{
    int syscallNum = SYSCALL_NUM;
    uid_t uid = MY_UID;
    char *binname = "init";
    int recover = 0;
    syscall(syscallNum,uid,binname,recover);
    return 0;
}
```

编译测试程序：

```
vim hide_user_processes.c
gcc hide_user_processes.c -o hide_user_processes
sudo chmod +x hide_user_processes
```

测试结果如下：

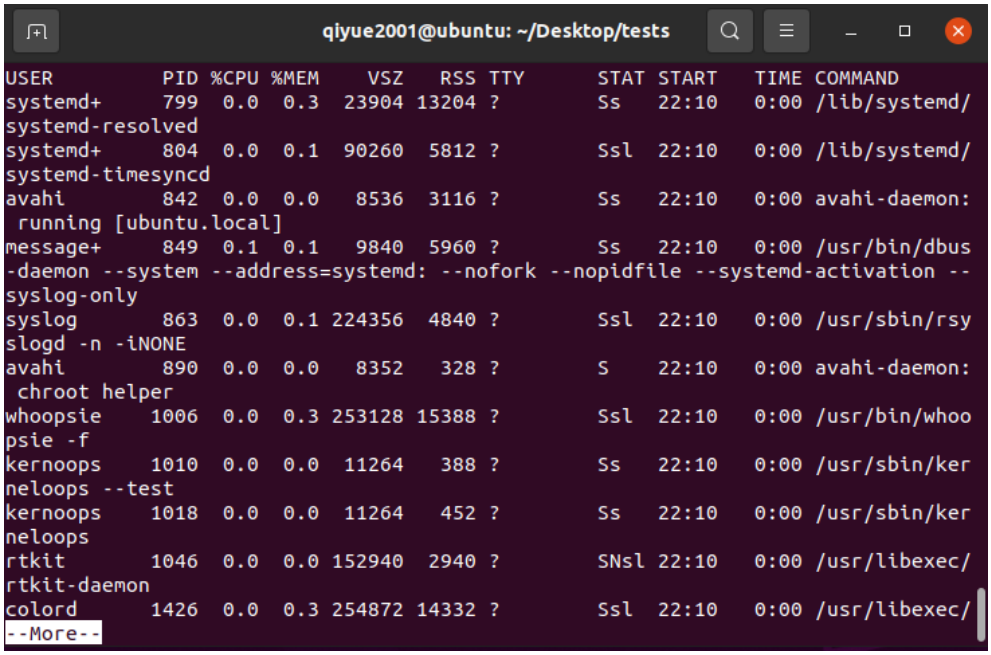
（1）使用非root用户执行，内核消息输出没有权限的提示。

```
qiyue2001@ubuntu: ~/Desktop/tests
ation="capable" profile="/snap/snapd/12704/usr/lib/snapd/snap-confine" pid=1962
comm="snap-confine" capability=4  capname="fsetid"
[   43.736187] audit: type=1326 audit(1629295875.901:46): auid=1000 uid=1000 gid
=1000 ses=3 subj==snap.snap-store.ubuntu-software (enforce) pid=1962 comm="snap-
store" exe="/snap/snap-store/547/usr/bin/snap-store" sig=0 arch=c000003e syscall
=314 compat=0 ip=0x7f9147052639 code=0x50000
[   45.140574] audit: type=1400 audit(1629295877.282:47): apparmor="DENIED" oper
ation="open" profile="snap.snap-store.ubuntu-software" name="/etc/PackageKit/Ven
dor.conf" pid=1962 comm="snap-store" requested_mask="r" denied_mask="r" fsuid=10
00 ouid=0
[   72.311558] audit: type=1400 audit(1629295904.349:48): apparmor="DENIED" oper
ation="open" profile="snap.snap-store.ubuntu-software" name="/var/lib/snapd/host
fs/usr/share/gdm/greeter/applications/gnome-initial-setup.desktop" pid=1962 comm
="pool-org.gnome." requested_mask="r" denied_mask="r" fsuid=1000 ouid=0
[   72.338466] audit: type=1400 audit(1629295904.373:49): apparmor="DENIED" oper
ation="open" profile="snap.snap-store.ubuntu-software" name="/var/lib/snapd/host
fs/usr/share/gdm/greeter/applications/gnome-initial-setup.desktop" pid=1962 comm
="pool-org.gnome." requested_mask="r" denied_mask="r" fsuid=1000 ouid=0
[   72.784869] audit: type=1326 audit(1629295904.821:50): auid=1000 uid=1000 gid
=1000 ses=3 subj==snap.snap-store.ubuntu-software (enforce) pid=1962 comm="pool-
org.gnome." exe="/snap/snap-store/547/usr/bin/snap-store" sig=0 arch=c000003e sy
scall=93 compat=0 ip=0x7f91470484e7 code=0x50000
[  149.628350] Permission denied. Only root can call hide_user_processes.
qiyue2001@ubuntu:~/Desktop/tests$
```

（2）使用root用户执行，隐藏uid为0的所有进程。对比前后`ps -aux`可见，root用户的所有进程都被隐藏了。

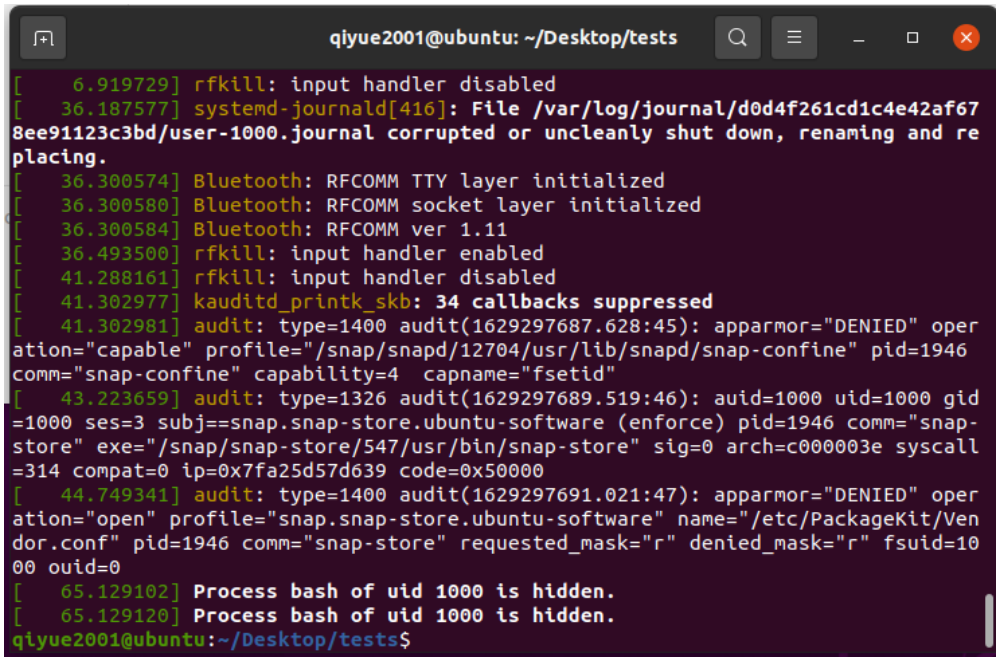
```
qiyue2001@ubuntu: ~/Desktop/tests
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.5  0.2 167692 11440 ?        Ss   22:10   0:01 /sbin/init au
to noprompt
root           2  0.0  0.0      0     0 ?        S    22:10   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   22:10   0:00 [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   22:10   0:00 [rcu_par_gp]
root           5  0.0  0.0      0     0 ?        I    22:10   0:00 [kworker/0:0-
rcu_par_gp]
root           6  0.0  0.0      0     0 ?        I<   22:10   0:00 [kworker/0:0H
-events_highpri]
root           7  0.0  0.0      0     0 ?        I    22:10   0:00 [kworker/0:1-
rcu_par_gp]
root           8  0.1  0.0      0     0 ?        I    22:10   0:00 [kworker/u256
:0-events_unbound]
root           9  0.0  0.0      0     0 ?        I<   22:10   0:00 [mm_percpu_wq
]
root          10  0.0  0.0      0     0 ?        S    22:10   0:00 [rcu_tasks_ru
de_]
root          11  0.0  0.0      0     0 ?        S    22:10   0:00 [rcu_tasks_tr
ace]
root          12  0.0  0.0      0     0 ?        S    22:10   0:00 [ksoftirqd/0]
root          13  0.0  0.0      0     0 ?        I    22:10   0:00 [rcu_sched]
root          14  0.0  0.0      0     0 ?        S    22:10   0:00 [migration/0]
--More--
```

```
qiyue2001@ubuntu: ~/Desktop/tests
oon joydev input_leds serio_raw snd_seq_midi btusb snd_seq_midi_event btrtl btbc
m btintel snd_rawmidi bluetooth snd_seq snd_seq_device snd_timer ecdh_generic ec
c snd soundcore vmw_vmci sch_fq_codel vmwgfx ttm drm_kms_helper cec rc_core fb_s
ys_fops syscopyarea sysfillrect sysimgblt msr parport_pc ppdev drm lp parport ip
_tables x_tables autofs4 hid_generic usbhid hid ahci libahci e1000 psmouse mptsp
i mptscsih mptbase i2c_piix4 scsi_transport_spi pata_acpi
[  312.034475] CR2: 000055ddc1fe7004
[  312.034477] ---[ end trace 1829936e97986445 ]---
[  312.034478] RIP: 0010:strncmp+0x1a/0x30
[  312.034480] Code: 3a 14 06 74 ef 19 c0 83 c8 01 c3 31 c0 c3 66 90 48 85 d2 74
 20 31 c0 eb 0d 84 c9 74 18 48 83 c0 01 48 39 d0 74 0f 0f b6 0c 07 <3a> 0c 06 74
 ea 19 c0 83 c8 01 c3 31 c0 c3 0f 1f 84 00 00 00 00 00
[  312.034481] RSP: 0018:ffffbf1592f1bee0 EFLAGS: 00010246
[  312.034482] RAX: 0000000000000000 RBX: ffff9f5580282050 RCX: 0000000000000073
[  312.034483] RDX: 0000000000000010 RSI: 000055ddc1fe7004 RDI: ffffbf1592f1bef0
[  312.034484] RBP: ffffbf1592f1bf30 R08: ffffbf1592f1bf00 R09: 0000000000000000
[  312.034485] R10: 0000000000000000 R11: 0000000000000000 R12: ffff9f5580281840
[  312.034486] R13: 0000000000000000 R14: 000055ddc1fe7004 R15: 0000000000000001
[  312.034502] FS:  00007f24837bc540(0000) GS:ffff9f55bde80000(0000) knlGS:00000
00000000000
[  312.034503] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[  312.034504] CR2: 000055ddc1fe7004 CR3: 000000011cd06006 CR4: 0000000003706e0
[  558.027691] All processes of uid 0 are hidden.
qiyue2001@ubuntu:~/Desktop/tests$
```

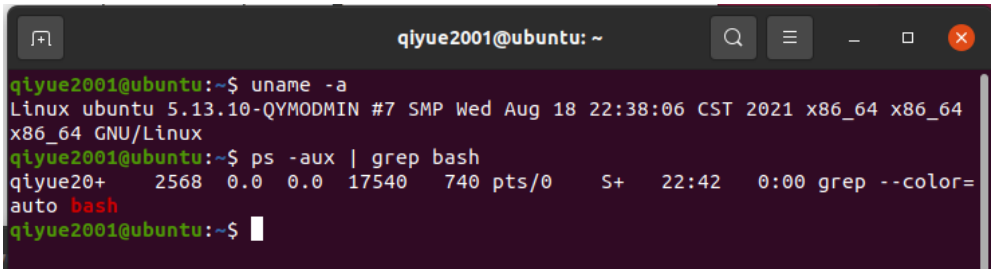


```
qiyue2001@ubuntu: ~/Desktop/tests
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
systemd+     799  0.0  0.3  23904 13204 ?        Ss   22:10   0:00 /lib/systemd/
systemd-resolved
systemd+     804  0.0  0.1  90260  5812 ?        Ssl  22:10   0:00 /lib/systemd/
systemd-timesyncd
avahi        842  0.0  0.0   8536  3116 ?        Ss   22:10   0:00 avahi-daemon:
 running [ubuntu.local]
message+     849  0.1  0.1   9840  5960 ?        Ss   22:10   0:00 /usr/bin/dbus
-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation --
syslog-only
syslog       863  0.0  0.1 224356  4840 ?        Ssl  22:10   0:00 /usr/sbin/rsy
slogd -n -iNONE
avahi        890  0.0  0.0   8352   328 ?        S    22:10   0:00 avahi-daemon:
 chroot helper
whoopsie    1006  0.0  0.3 253128 15388 ?        Ssl  22:10   0:00 /usr/bin/whoo
psie -f
kernoops    1010  0.0  0.0  11264   388 ?        Ss   22:10   0:00 /usr/sbin/ker
neloops --test
kernoops    1018  0.0  0.0  11264   452 ?        Ss   22:10   0:00 /usr/sbin/ker
neloops
rtkit       1046  0.0  0.0 152940  2940 ?        SNsl 22:10   0:00 /usr/libexec/
rtkit-daemon
colord      1426  0.0  0.3 254872 14332 ?        Ssl  22:10   0:00 /usr/libexec/
--More--
```

（3）使用root用户执行，隐藏uid为1000（即用户`qiyue2001`，可用`id`命令获得当前用户的uid）的`bash`进程。对比前后`ps -aux`可见，对应的进程被隐藏了。

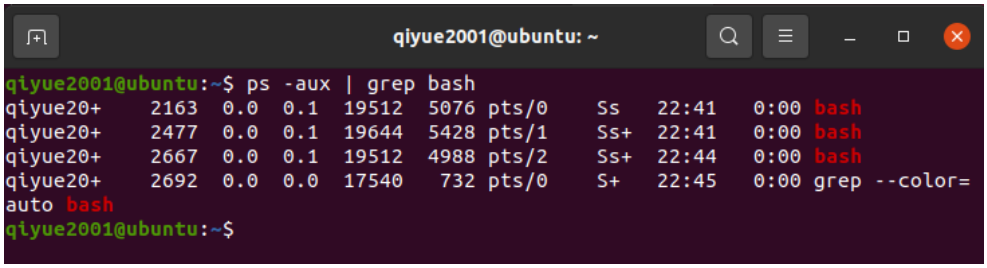


```
qiyue2001@ubuntu: ~/Desktop/tests
[    6.919729] rfkill: input handler disabled
[   36.187577] systemd-journald[416]: File /var/log/journal/d0d4f261cd1c4e42af67
8ee91123c3bd/user-1000.journal corrupted or uncleanly shut down, renaming and re
placing.
[   36.300574] Bluetooth: RFCOMM TTY layer initialized
[   36.300580] Bluetooth: RFCOMM socket layer initialized
[   36.300584] Bluetooth: RFCOMM ver 1.11
[   36.493500] rfkill: input handler enabled
[   41.288161] rfkill: input handler disabled
[   41.302977] kauditd_printk_skb: 34 callbacks suppressed
[   41.302981] audit: type=1400 audit(1629297687.628:45): apparmor="DENIED" oper
ation="capable" profile="/snap/snapd/12704/usr/lib/snapd/snap-confine" pid=1946
comm="snap-confine" capability=4  capname="fsetid"
[   43.223659] audit: type=1326 audit(1629297689.519:46): auid=1000 uid=1000 gid
=1000 ses=3 subj==snap.snap-store.ubuntu-software (enforce) pid=1946 comm="snap-
store" exe="/snap/snap-store/547/usr/bin/snap-store" sig=0 arch=c000003e syscall
=314 compat=0 ip=0x7fa25d57d639 code=0x50000
[   44.749341] audit: type=1400 audit(1629297691.021:47): apparmor="DENIED" oper
ation="open" profile="snap.snap-store.ubuntu-software" name="/etc/PackageKit/Ven
dor.conf" pid=1946 comm="snap-store" requested_mask="r" denied_mask="r" fsuid=10
00 ouid=0
[   65.129102] Process bash of uid 1000 is hidden.
[   65.129120] Process bash of uid 1000 is hidden.
qiyue2001@ubuntu:~/Desktop/tests$
```



```
qiyue2001@ubuntu: ~
qiyue2001@ubuntu:~$ uname -a
Linux ubuntu 5.13.10-QYMODMIN #7 SMP Wed Aug 18 22:38:06 CST 2021 x86_64 x86_64
x86_64 GNU/Linux
qiyue2001@ubuntu:~$ ps -aux | grep bash
qiyue20+    2568  0.0  0.0  17540   740 pts/0    S+   22:42   0:00 grep --color=
auto bash
qiyue2001@ubuntu:~$
```

（4）使用root用户执行，设置recover=1。对比前后`ps -aux`可见，（3）中被隐藏的进程又恢复了。实验成功。



```
qiyue2001@ubuntu: ~
qiyue2001@ubuntu:~$ ps -aux | grep bash
qiyue20+    2163  0.0  0.1  19512  5076 pts/0    Ss   22:41   0:00 bash
qiyue20+    2477  0.0  0.1  19644  5428 pts/1    Ss+  22:41   0:00 bash
qiyue20+    2667  0.0  0.1  19512  4988 pts/2    Ss+  22:44   0:00 bash
qiyue20+    2692  0.0  0.0  17540   732 pts/0    S+   22:45   0:00 grep --color=
auto bash
qiyue2001@ubuntu:~$
```

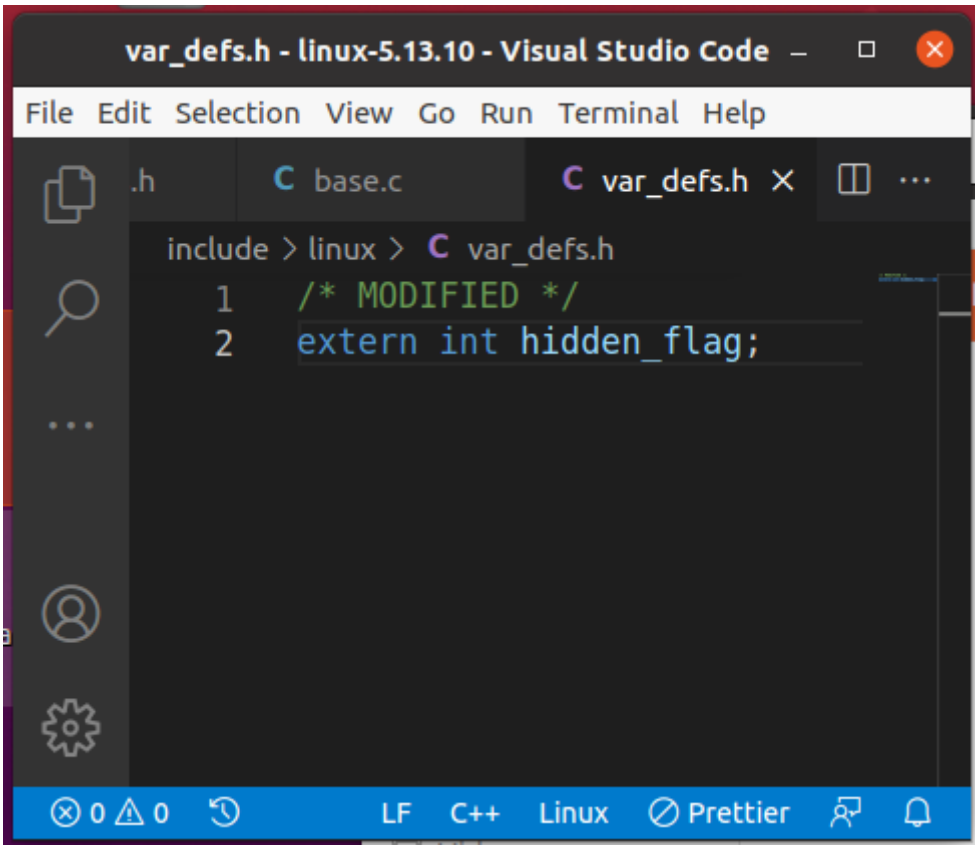
## 在`/proc`目录下创建一个文件`/proc/hidden`

### 定义全局变量`hidden_flag`

在`/include/linux`下新建`var_defs.h`，输入

```
extern int hidden_flag;
```

其他文件在使用这个全局变量时都要include这个头文件。




### 实现`procfs`对`hide`文件的创建及读写

接口`proc_create`（较早版本为`create_proc_entry`）允许新增`proc`项。

完整的接口是`struct proc_dir_entry *proc_create ( const char *name, umode_t mode, struct proc_dir_entry *parent, const struct proc_ops *proc_ops )`。

其中，`name`为文件名。`mode`为访问模式，设置为0644（root可读写，其余用户只可读）。`parent`为父目录的名字，这里设置为`NULL`，让其位于`/proc`。`proc_ops`是一个包含了将要创建的文件项的相关信息的结构体。

新增一个`procfs`项的最好方法是写成一个LKM（Linux内核模块）。但在本次实验中，为了测试的简便（模块要用`sudo insmod <filename>`加载），我并没有这样做。But commented code should give you a hint on how to do that.

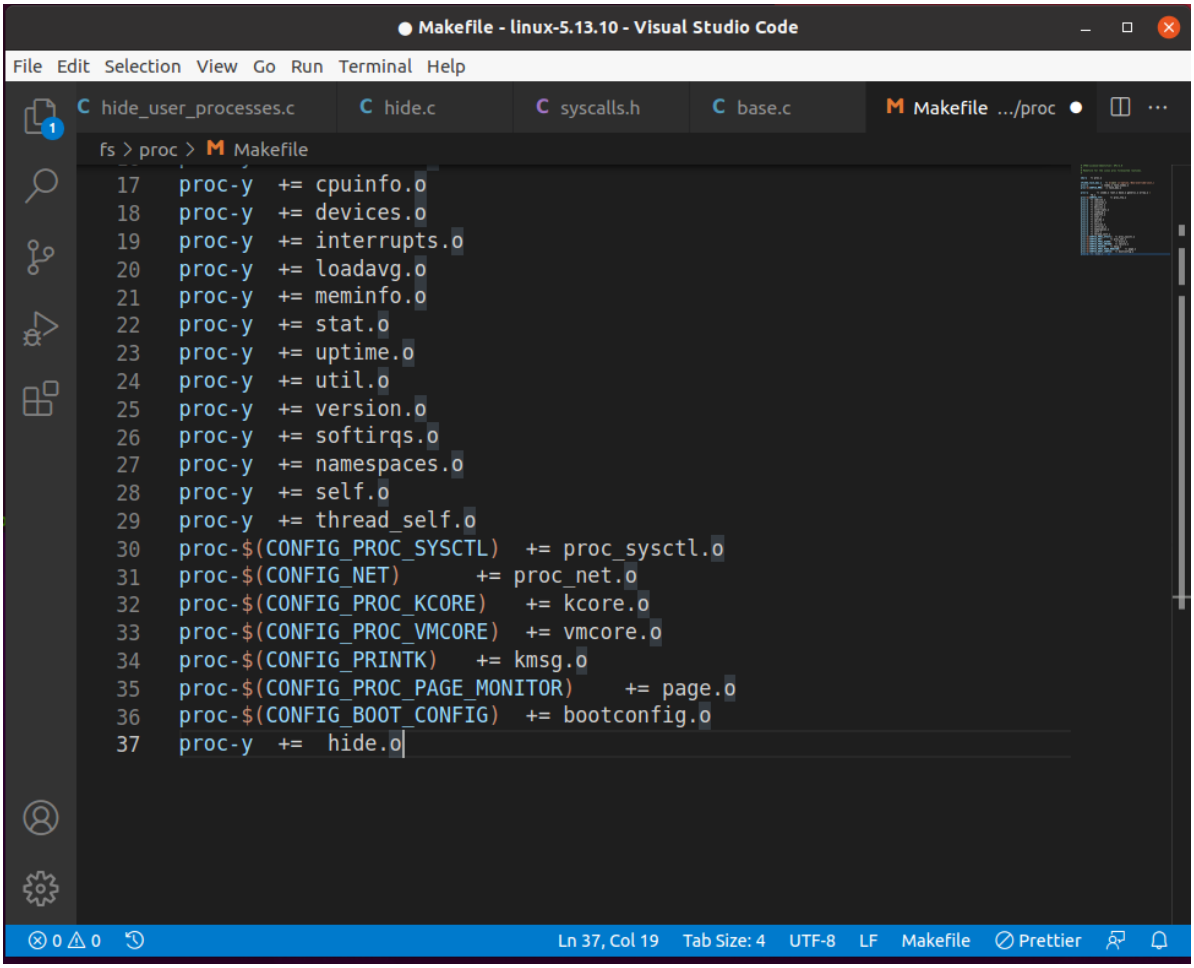
然后，编写`fs/proc/hide.c`代码如下：

```
/**
 * MODIFIED
 * Add procfs entry `hide`
*/

#include <linux/kernel.h>
```

```c
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kdev_t.h>
#include <linux/fs.h>
#include <linux/device.h>
#include <linux/uaccess.h>
#include <linux/ioctl.h>
#include <linux/proc_fs.h>

#include <linux/var_defs.h>

#define PROC_MAX_SIZE 16
#define PRINT_KERNEL_MESSAGE

int hidden_flag = 1;

static struct proc_dir_entry *hide_entry;

static ssize_t proc_read_hidden(struct file *file, char __user *buf,
                size_t count, loff_t *ppos)
{
    char str[16];
    ssize_t cnt;
    ssize_t ret;
#ifdef PRINT_KERNEL_MESSAGE
    printk("In proc_read_hidden.\n");
    printk("hidden_flag: %d\n", hidden_flag);
#endif
    snprintf(str, sizeof(str), "%d\n", hidden_flag);
    cnt = strlen(str);

    /* ret contains the amount of chare wasn't successfully written to `buf` */
    ret = copy_to_user(buf, str, cnt);
    *ppos += cnt - ret;

    /* Making sure there are no left bytes of data to send user */
    if (*ppos > cnt)
        return 0;
    else
        return cnt;
}

static ssize_t proc_write_hidden(struct file *file, const char __user *buf,
                 size_t count, loff_t *ppos)
{
    char temp[PROC_MAX_SIZE];
    int tmp_flag = 0;
#ifdef PRINT_KERNEL_MESSAGE
    printk("In proc_write_hidden.\n");
#endif
    if (count > PROC_MAX_SIZE)
        count = PROC_MAX_SIZE;
    if (copy_from_user(temp, buf, count)) {
        return -EFAULT;
    }
    //temp[count]='\0';
    if (kstrtoint(temp, 10, &tmp_flag)) /* 10: base */
        return -1;
```

```
    hidden_flag = tmp_flag; /* set the value of hidden_flag */
#ifdef PRINT_KERNEL_MESSAGE
    printk("hidden_flag: %d\n", hidden_flag);
#endif
    return count;
}

static const struct proc_ops hide_proc_ops = {
    .proc_write = proc_write_hidden,
    .proc_read = proc_read_hidden,
};

static int __init proc_hide_init(void)
{
    /* 0:oct 6:rw 4:r */
    hide_entry = proc_create("hide", 0644, NULL, &hide_proc_ops);
    return 0;
}

void proc_hide_cleanup(void)
{
    proc_remove(hide_entry);
}
fs_initcall(proc_hide_init);

/** if you'd like to build it as a module
MODULE_LICENSE("GPL");
MODULE_AUTHOR("HUANG Qiyue <qiyue2001@gmail.com>");
MODULE_DESCRIPTION("Simple hide process driver (procfs)");
MODULE_VERSION("1.0");
module_init(proc_hide_init);
module_exit(proc_hide_cleanup);
*/
```

记得在`fs/proc/MakeFile`中增加`proc-y += hide.o`！如果要编译为LKM，则选项应该是`obj-m`。




## 根据hidden_flag显示/隐藏进程

在`fs\proc\base.c`的`int proc_pid_readdir(struct file *file, struct dir_context *ctx)`以及`struct dentry *proc_pid_lookup(struct dentry *dentry, unsigned int flags)`中，增加对`hidden_flag`的判断语句。（记得include！）

base.c - linux-5.13.10 - Visual Studio Code

File Edit Selection View Go Run Terminal Help

:hed.h | Makefile | hide_user_processes.c | hide.c | syscalls.h | base.c

fs > proc > base.c > proc_pid_readdir(file *, dir_context *)

```c
    iter.task = NULL;
    for (iter = next_tgid(ns, iter);
         iter.task;
         iter.tgid += 1, iter = next_tgid(ns, iter)) {
        char name[10 + 1];
        unsigned int len;

        cond_resched();
        if (!has_pid_permissions(fs_info, iter.task, HIDEPID_INVISIBLE))
            continue;

        /* MODIFIED */
        /* if hidden, skip */
        if (hidden_flag==1 && (iter.task)->cloak == 1){
            continue;
        }

        len = snprintf(name, sizeof(name), "%u", iter.tgid);
        ctx->pos = iter.tgid + TGID_OFFSET;
        if (!proc_fill_cache(file, ctx, name, len,
                             proc_pid_instantiate, iter.task, NULL)) {
            put_task_struct(iter.task);
            return 0;
        }
    }
    ctx->pos = PID_MAX_LIMIT + TGID_OFFSET;
    return 0;
}
```

0 ⚠ 0 | Ln 3470, Col 27 | Tab Size: 4 | UTF-8 | LF | C | Linux | Prettier

base.c - linux-5.13.10 - Visual Studio Code

File Edit Selection View Go Run Terminal Help

:hed.h | Makefile | hide_user_processes.c | hide.c | syscalls.h | base.c

fs > proc > base.c > proc_pid_lookup(dentry *, unsigned int)

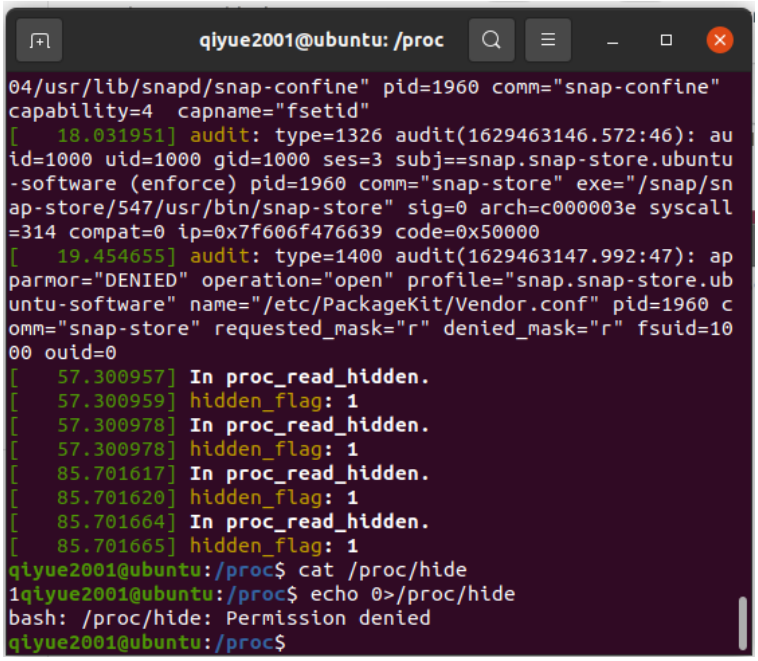
```c
    struct pid_namespace *ns;
    struct dentry *result = ERR_PTR(-ENOENT);

    tgid = name_to_int(&dentry->d_name);
    if (tgid == ~0U)
        goto out;

    fs_info = proc_sb_info(dentry->d_sb);
    ns = fs_info->pid_ns;
    rcu_read_lock();
    task = find_task_by_pid_ns(tgid, ns);
    if (task)
        get_task_struct(task);
    rcu_read_unlock();
    if (!task)
        goto out;

    /* MODIFIED */
    if(hidden_flag==1 && task->cloak==1)
        goto out;

    /* Limit procfs to only ptraceable tasks */
    if (fs_info->hide_pid == HIDEPID_NOT_PTRACEABLE) {
        if (!has_pid_permissions(fs_info, task, HIDEPID_NO_ACCESS))
            goto out_put_task;
    }

    result = proc_pid_instantiate(dentry, task, NULL);
out_put_task:
```

0 ⚠ 0 | Ln 3384, Col 26 | Tab Size: 4 | UTF-8 | LF | C | Linux | Prettier

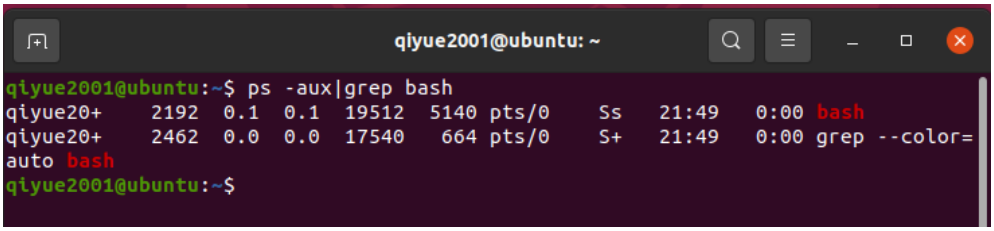
## 测试

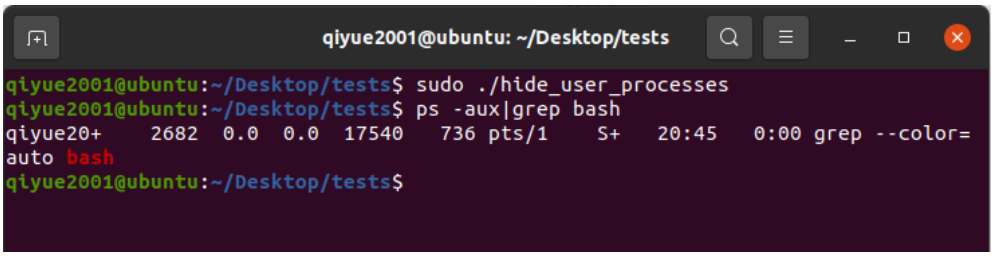
如果编译为LKM，首先需要用`sudo insmod hide.ko`加载模块。我没有编译为LKM，所以不需要这一步。

首先`cat /proc/hide`，可以看到结果`1`，表示初始情况下我们允许进行隐藏操作。同时`dmesg`也会打印相应内容。读取不需要root权限，但修改则需要root权限。



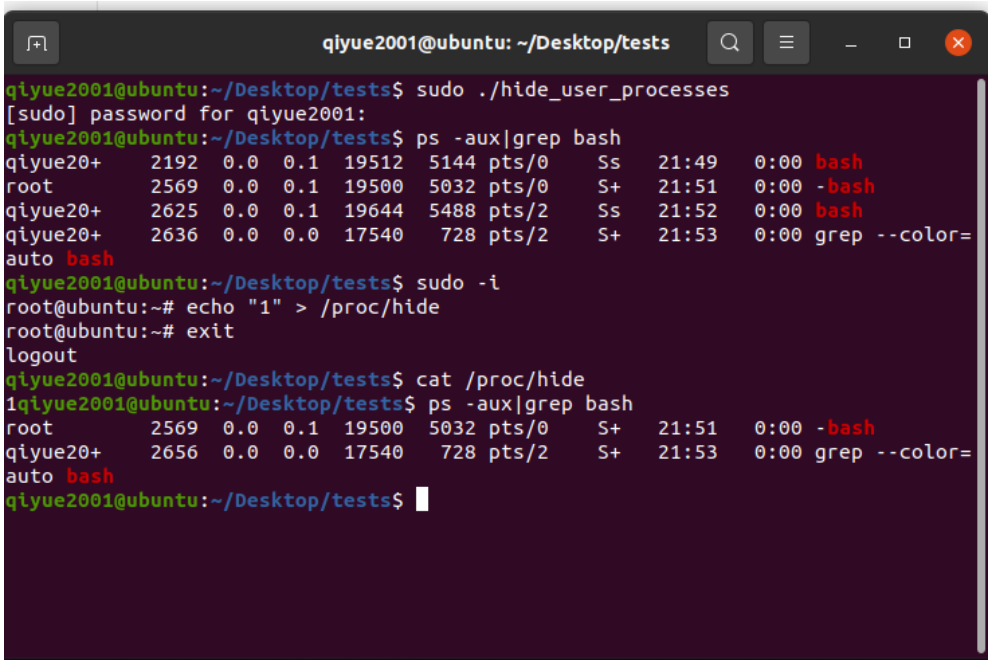

进行`hide`系统调用的测试，隐藏uid为1000的`bash`进程，成功。




然后用root用户`echo "0" > /proc/hide`。再次`ps -aux`，看到刚刚被隐藏的`bash`又显示了。

`echo "1" > /proc/hide`，`bash`又隐藏了。实验成功。




## 在`/proc`目录下创建一个文件`/proc/hidden_process`

该文件用于存储所有被隐藏进程的pid。这个文件不允许从用户空间写入，因此只需要实现读回调函数。

与上一个实验类似，只需要新建`fs/proc/hidden_process.c`。另外，不要忘记修改`MakeFile`！

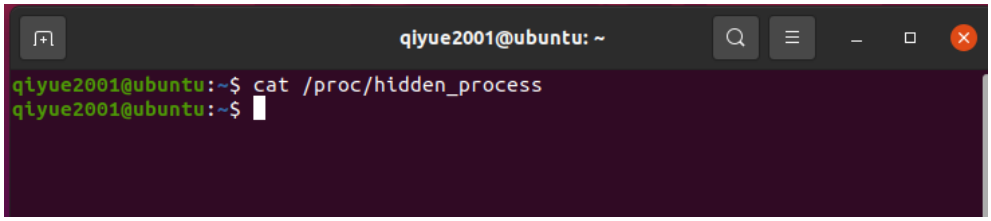
`hidden_process.c`代码如下：

```
/**
 * MODIFIED
 * Add procfs entry `hidden_process`
*/

#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kdev_t.h>
#include <linux/fs.h>
#include <linux/device.h>
#include <linux/uaccess.h>
#include <linux/ioctl.h>
#include <linux/proc_fs.h>

#include <linux/var_defs.h>

#define PRINT_KERNEL_MESSAGE

static struct proc_dir_entry *hidden_process_entry;

static ssize_t proc_read_hidden_process(struct file *file, char __user *buf,
                    size_t count, loff_t *ppos)
{
    ssize_t cnt;
    ssize_t ret;
    char kbuf[1000];
    char tmp[16];
    struct task_struct *p;
#ifdef PRINT_KERNEL_MESSAGE
```

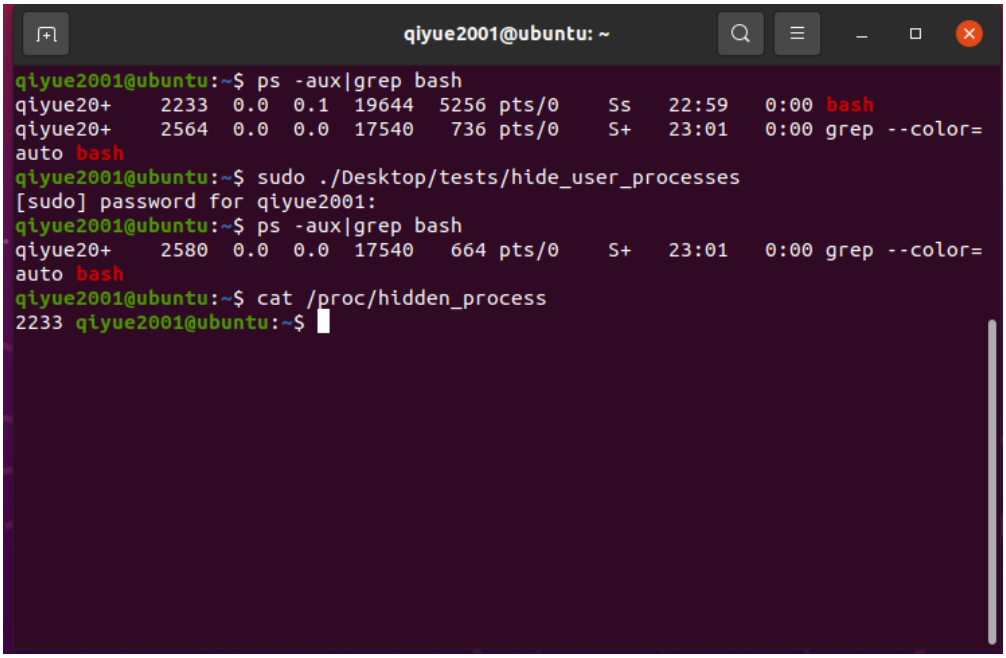
```
    printk("In proc_read_hidden_process.\n");
#endif
    sprintf(kbuf, "%s", "");    /* init buffer */
    for_each_process (p) {
        if (p->cloak == 1) {
            sprintf(tmp, "%ld ", (long)p->pid);
            strcat(kbuf, tmp);
        }
    }
    cnt = strlen(kbuf);

    /* ret contains the amount of chare wasn't successfully written to `buf` */
    ret = copy_to_user(buf, kbuf, cnt);
    *ppos += cnt - ret;

    /* Making sure there are no left bytes of data to send user */
    if (*ppos > cnt)
        return 0;
    else
        return cnt;
}

static const struct proc_ops hidden_process_proc_ops = {
    .proc_read = proc_read_hidden_process,
};

static int __init proc_hidden_process_init(void)
{
    /* 0:oct 6:rw 4:r */
    hidden_process_entry = proc_create("hidden_process", 0444, NULL,
                &hidden_process_proc_ops);
    return 0;
}

void proc_hidden_process_cleanup(void)
{
    proc_remove(hidden_process_entry);
}
fs_initcall(proc_hidden_process_init);

/** if you'd like to build it as a module
MODULE_LICENSE("GPL");
MODULE_AUTHOR("HUANG Qiyue <qiyue2001@gmail.com>");
MODULE_DESCRIPTION("Simple hidden_process process driver (procfs)");
MODULE_VERSION("1.0");
module_init(proc_hidden_process_init);
module_exit(proc_hidden_process_cleanup);
*/
```

测试结果如下：

初始状态，`hidden_process`为空。

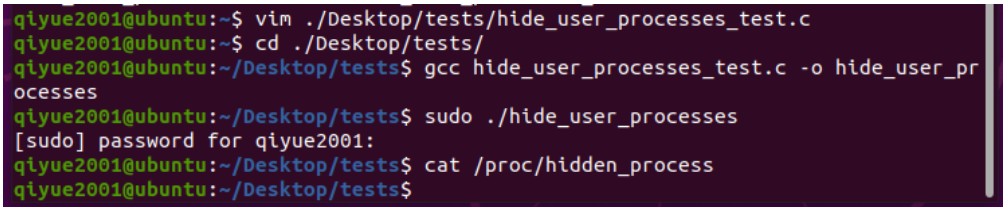



调用`hide_user_processes`系统调用隐藏uid=1000的`bash`进程，查看前后的`ps -aux|bash`结果，并打印此时的`hidden_process`，结果如下：




可见，成功显示了被隐藏的`bash`进程的pid。

此刻我们再次调用`hide_user_processes`，但是令参数`recover`为1，即使之前被隐藏的进程恢复显示（修改`cloak`位），再次查看`hidden_process`，可见此时没有进程被隐藏了。结果正确，实验成功。




## 实验体会

由于我使用的版本较新，并没有直接的教程可供模仿，因此我主要依赖Linux内核源码和文档、Stack Overflow等网站学习内核函数相关用法，在此过程中加深了对内核工作原理的认识。从内核函数的变化，可以看出安全性、通用性等方面的考量，也让我对现代操作系统工程上的复杂性有了更为直观的感受。

# Troubleshooting

（1）编译内核

```
  CC      kernel/user-return-notifier.o
  CC      kernel/padata.o
  CC      kernel/crash_dump.o
  CC      kernel/jump_label.o
  CC      kernel/iomem.o
  CC      kernel/rseq.o
  CC      kernel/watch_queue.o
  AR      kernel/built-in.a
  CHK     kernel/kheaders_data.tar.xz
  GEN     kernel/kheaders_data.tar.xz
  CC [M]  kernel/kheaders.o
  CC      certs/system_keyring.o
make[1]: *** No rule to make target 'debian/canonical-certs.pem', needed by 'certs/x
509_certificate_list'.  Stop.
make: *** [Makefile:1862: certs] Error 2
qiyue2001@ubuntu:~/Desktop/linux-5.13.10$
```

参见： https://askubuntu.com/questions/1329538/compiling-the-kernel-5-11-11

（2）测试`hide_user_processes`，如果BINNAME不为NULL会被直接Killed，查看`dmesg`：

```
qiyue2001@ubuntu: ~/Desktop/tests
[  856.063125] BUG: unable to handle page fault for address: 00005575e86df004
[  856.063130] #PF: supervisor read access in kernel mode
[  856.063131] #PF: error_code(0x0000) - not-present page
[  856.063132] PGD 800000011b8d4067 P4D 800000011b8d4067 PUD 108ab9067 PMD 10ae23067 PTE 0
[  856.063135] Oops: 0000 [#2] SMP PTI
[  856.063137] CPU: 1 PID: 7026 Comm: hide_user_proce Tainted: G      D           5.13.10-QYMODMIN #6
[  856.063140] Hardware name: VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Platform, BIOS 6.00 02/27/2020
[  856.063141] RIP: 0010:strncmp+0x1a/0x30
[  856.063145] Code: 3a 14 06 74 ef 19 c0 83 c8 01 c3 31 c0 c3 66 90 48 85 d2 74 20 31 c0 eb 0d 84 c9 74 18 48 83 c0 01 48 39 d0 74 0f 0f b6 0c 07 <3a> 0c 06 74
 ea 19 c0 83 c8 01 c3 31 c0 c3 0f 1f 84 00 00 00 00 00
[  856.063146] RSP: 0018:ffffbf158472fee0 EFLAGS: 00010246
[  856.063148] RAX: 0000000000000000 RBX: ffff9f559d8f2050 RCX: 0000000000000073
[  856.063149] RDX: 0000000000000010 RSI: 00005575e86df004 RDI: ffffbf158472fef0
[  856.063150] RBP: ffffbf158472ff30 R08: ffffbf158472ff00 R09: 0000000000000000
[  856.063151] R10: 0000000000000000 R11: 0000000000000000 R12: ffff9f559d8f1840
[  856.063152] R13: 00000000000003e8 R14: 00005575e86df004 R15: 0000000000000001
[  856.063153] FS:  00007fda0c12a540(0000) GS:ffff9f55bde40000(0000) knlGS:0000000000000000
[  856.063154] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[  856.063155] CR2: 00005575e86df004 CR3: 0000000111178001 CR4: 00000000003706e0
[  856.063175] Call Trace:
[  856.063177]  ? __x64_sys_hide_user_processes+0x118/0x1d0
[  856.063181]  do_syscall_64+0x40/0xb0
[  856.063184]  entry_SYSCALL_64_after_hwframe+0x44/0xae
[  856.063187] RIP: 0033:0x7fda0c05289d
[  856.063188] Code: 00 c3 66 2e 0f 1f 84 00 00 00 00 00 90 f3 0f 1e fa 48 89 f8 48 89 f7 48 89 d6 48 89 ca 4d 89 c2 4d 89 c8 4c 8b 4c 24 08 0f 05 <48> 3d 01 f0
 ff ff 73 01 c3 48 8b 0d c3 f5 0c 00 f7 d8 64 89 01 48
[  856.063190] RSP: 002b:00007fffeffdacb8 EFLAGS: 00000206 ORIG_RAX: 00000000000001c0
[  856.063191] RAX: ffffffffffffffda RBX: 00005575e86de1a0 RCX: 00007fda0c05289d
[  856.063192] RDX: 0000000000000000 RSI: 00005575e86df004 RDI: 00000000000003e8
[  856.063193] RBP: 00007fffeffdace0 R08: 00007fda0c14dd50 R09: 0000000000000000
[  856.063194] R10: 0000000000000000 R11: 0000000000000206 R12: 00005575e86de060
[  856.063195] R13: 00007fffeffdadd0 R14: 0000000000000000 R15: 0000000000000000
[  856.063196] Modules linked in: rfcomm bnep vsock_loopback vmw_vsock_virtio_transport_common vmw_vsock_vmci_transport vsock nls_iso8859_1 intel_rapl_msr intel
_rapl_common crct10dif_pclmul crc32_pclmul ghash_clmulni_intel snd_ens1371 snd_ac97_codec aesni_intel gameport ac97_bus crypto_simd cryptd rapl snd_pcm vmw_ball
oon joydev input_leds serio_raw snd_seq_midi btusb snd_seq_midi_event btrtl btbcm btintel snd_rawmidi bluetooth snd_seq snd_seq_device snd_timer ecdh_generic ec
c snd soundcore vmw_vmci sch_fq_codel vmwgfx ttm drm_kms_helper cec rc_core fb_sys_fops syscopyarea sysfillrect sysimgblt msr parport_pc ppdev drm lp parport ip
_tables x_tables autofs4 hid_generic usbhid hid ahci libahci e1000 psmouse mptspi mptscsih mptbase i2c_piix4 scsi_transport_spi pata_acpi
[  856.063230] CR2: 00005575e86df004
[  856.063231] ---[ end trace 1829936e97986446 ]---
[  856.063232] RIP: 0010:strncmp+0x1a/0x30
[  856.063234] Code: 3a 14 06 74 ef 19 c0 83 c8 01 c3 31 c0 c3 66 90 48 85 d2 74 20 31 c0 eb 0d 84 c9 74 18 48 83 c0 01 48 39 d0 74 0f 0f b6 0c 07 <3a> 0c 06 74
 ea 19 c0 83 c8 01 c3 31 c0 c3 0f 1f 84 00 00 00 00 00
[  856.063235] RSP: 0018:ffffbf1592f1bee0 EFLAGS: 00010246
[  856.063236] RAX: 0000000000000000 RBX: ffff9f5580282050 RCX: 0000000000000073
[                  DX: 0000000000000010 RSI: 000055ddc1fe7004 RDI: ffffbf1592f1bef0
Show Applications  BP: ffffbf1592f1bf30 R08: ffffbf1592f1bf00 R09: 0000000000000000
[  856.063239] R10: 0000000000000000 R11: 0000000000000000 R12: ffff9f5580281840
```

这是因为SMAP（Supervisor Mode Access Prevention）阻止直接访问用户空间的指针。应当先调用`strncpy_from_user`。

# References

## General

- A complete yet outdated tutorial: https://www.cnblogs.com/hellovenus/p/3967597.html
- An online Linux source viewer: https://elixir.bootlin.com/linux/latest/source
- Linux kernel documentation: https://www.kernel.org/doc/html/latest/

## Building Linux Kernel

- https://zhuanlan.zhihu.com/p/37164435
- https://www.jianshu.com/p/dc5063edcadd
- https://www.cnblogs.com/hellovenus/p/os_linux_core_study.html
- Make 如何配置： https://www.jianshu.com/p/876043f48120
- https://unix.stackexchange.com/questions/71154/only-output-errors-warnings-when-compile-kernel
- https://stackoverflow.com/questions/22900073/compiling-linux-kernel-after-making-changes

## System call

- Adding a system call turotial: https://dev.to/jasper/adding-a-system-call-to-the-linux-kernel-5-8-1-in-ubuntu-20-04-lts-2ga8
- Linux documentation on adding syscalls: https://www.kernel.org/doc/html/latest/process/adding-syscalls.html

## procfs

- How is proc able to list pids: https://ops.tips/blog/how-is-proc-able-to-list-pids/
- Create an entry (new): https://embetronicx.com/tutorials/linux/device-drivers/procfs-in-linux/#Creating_Procfs_Entry
- https://tuxthink.blogspot.com/2017/03/creating-proc-read-and-write-entry.html

## Implementing a file system

- https://sites.cs.ucsb.edu/~trinabh/classes/w19/labs/lab6.html
- http://web.mit.edu/6.033/1997/handouts/html/04sfs.html
- http://web.mit.edu/6.033/1997/handouts/html/04sfs.html
- https://www.cse.iitb.ac.in/~mythili/teaching/cs347_autumn2016/labs/lab8.pdf (Recommended)

## Miscellaneous

- Ubuntu开机启动菜单grub2： https://blog.csdn.net/lu_embedded/article/details/44353499
- https://unix.stackexchange.com/questions/373402/cant-get-grub2-menu-to-display
- https://superuser.com/questions/439511/how-to-save-or-export-a-custom-linux-kernel-configuration
- Changes in new kernel version:
  - Get uid: https://stackoverflow.com/questions/39229639/how-to-get-current-processs-uid-and-euid-in-linux-kernel-4-2/39230936
  - Find process by pid: https://stackoverflow.com/questions/24447841/alternative-for-find-task-by-pidhttps://stackoverflow.com/questions/59772132/how-to-correctly-extract-a-string-from-a-user-space-pointer-in-kernel-space
- Accessing user space pointer: https://stackoverflow.com/questions/59772132/how-to-correctly-extract-a-string-from-a-user-space-pointer-in-kernel-space