



东南大学
SOUTHEAST UNIVERSITY

操作系统实践实验报告

姓名 李宗辉

学号 61522122

教师 张柏礼

2024 年 12 月 7 日

目录

1 下载和编译 Linux 源码	3
2 Linux 进程管理及其扩展	6
2.1 实现系统调用 hide	7
2.1.1 修改 PCB	7
2.1.2 修改 copy_process	7
2.1.3 修改进程显示函数	8
2.1.4 添加 hide 系统调用	8
2.2 实现系统调用 hide_user_processes	13
2.3 创建 fs/proc/hidden 文件和 fs/proc/hidden_process	17
2.3.1 创建头文件	17
2.3.2 完成文件读写函数及修改/proc 初始化函数	17
2.3.3 完善 hide 系统调用的判断逻辑	20
2.3.4 测试	21
3 实验疑难与体会	22
3.1 Linux 源码编译	22
3.2 内核修改开发	23
3.3 体会	23

1 下载和编译 Linux 源码

可以从网上 <https://www.kernel.org/> 下载 Linux 内核源码，虚拟机使用 Ubuntu 版本为 6.8.0-49-generic，我选择的 Linux 内核源码版本为 6.8.1。但由于官网下载速度过慢，选择镜像网站 <https://mirrors.edge.kernel.org/pub/linux/kernel/v6.x/> 找到对应版本下载。

接着下载一些编译所需的工具如下：

```
1 sudo apt-get install vim git fakeroot build-essential ncurses-dev xz-utils bc flex libelf-dev bison
```

解压下载的源代码，得到 linux-6.8.1 目录。

```
1 xz -d linux-6.8.1.tar.xz
2 tar -xavf <linux-6.8.1.tar>
```

切换到 linux-6.8.1 目录下，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 = OSRACTICE。

接着设定使用当前系统的内核配置文件为新内核的配置文件：

```
1 cp /boot/config-6.8.0-49-generic .config
```

如果需要更详细的配置，可以运行以下命令打开图形化配置界面。

```
1 make menuconfig
```

由于 Debian 及其衍生版（即包括了 Ubuntu）为内核模块使用一个签名证书。默认情况下，计算机并不包含这个证书。

因此需要关闭启用模块签名的选项，否则在接下来的内核编译中会报错。

```
1 ./scripts/config --file .config --set-str SYSTEM_TRUSTED_KEYS ""
2 ./scripts/config --file .config --set-str SYSTEM_REVOCATION_KEYS ""
3 scripts/config --disable SYSTEM_TRUSTED_KEYRING
4 scripts/config --disable MODULE_SIG
5 scripts/config --disable MODULE_SIG_ALL
6 scripts/config --disable MODULE_SIG_SHA256
7 scripts/config --disable MODULE_SIG_SHA512
8 scripts/config --disable MODULE_SIG_RSA
```

```

9   scripts/config --disable MODULE_SIG_ECDSA
10  scripts/config --disable X509_CERTIFICATE_PARSER
11  scripts/config --disable CERTS

```

接下来使用多线程加速编译，`make -j$(nproc)` 表示使用所有 CPU 核心并行编译，这里选用参数 `-j4`。`2>&1` 会将 `STDOUT` 和 `STDERR` 重定向到相同的文件描述符，并通过管道传输给 `tee` 命令，这会将输出存储在一个名为 `log` 的文件中，即将终端打印出的内容存储到日志中便于查看。

由于使用了多线程加速，编译的速度很快，大概用了一个半小时完成编译：

```
1 make -j4 2> &1 | tee log
```

然后安装编译好的内核模块：

```
1 sudo make modules_install
```

最后安装新编译的内核：

```
1 sudo make install
```

如图 1 所示 6.8.1-OSPRACTICE 为新编译的内核。

```

TICE
update-initramfs: Generating /boot/initrd.img-6.8.1-OSPRACTICE
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 6.8.1-OSPRACTICE /boot/vmlinuz-6.8.1-OSPRACTICE
run-parts: executing /etc/kernel/postinst.d/update-notifier 6.8.1-OSPRACTICE /boot/vmlinuz-6.8.1-OSPRACTICE
run-parts: executing /etc/kernel/postinst.d/xx-update-initrd-links 6.8.1-OSPRACTICE /boot/vmlinuz-6.8.1-OSPRACTICE
I: /boot/vmlinuz.old is now a symlink to vmlinuz-6.8.0-49-generic
I: /boot/initrd.img.old is now a symlink to initrd.img-6.8.0-49-generic
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 6.8.1-OSPRACTICE /boot/vmlinuz-6.8.1-OSPRACTICE
Sourcing file `/etc/default/grub'
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-6.8.1-OSPRACTICE
Found initrd image: /boot/initrd.img-6.8.1-OSPRACTICE
Found linux image: /boot/vmlinuz-6.8.0-49-generic
Found initrd image: /boot/initrd.img-6.8.0-49-generic
Found linux image: /boot/vmlinuz-6.8.0-48-generic
Found initrd image: /boot/initrd.img-6.8.0-48-generic
Found memtest86+x64 image: /boot/memtest86+x64.bin
Warning: os-prober will not be executed to detect other bootable partitions.
Systems on them will not be added to the GRUB boot configuration.
Check GRUB_DISABLE_OS_PROBER documentation entry.
Adding boot menu entry for UEFI Firmware Settings ...
done
mercurystraw@mercurystraw-VMware-Virtual-Platform:~/Desktop/linux-6.8.1$ 

```

图 1：编译好的内核

为了使用编译的内核，需要编辑 `grub` 选项。输入以下命令打开 `grub` 选项，

```
1 sudo vim /etc/default/grub
```

并将其设置为 GRUB_TIMEOUT_STYLE=menu 和 GRUB_TIMEOUT=5。这是在开机的时候打开一个选用内核的菜单界面并且设置选择时间为 5 秒。

更新 grub 并且重启选择新内核，结果如图 2 所示。

```
1 sudo update-grub  
2 reboot
```

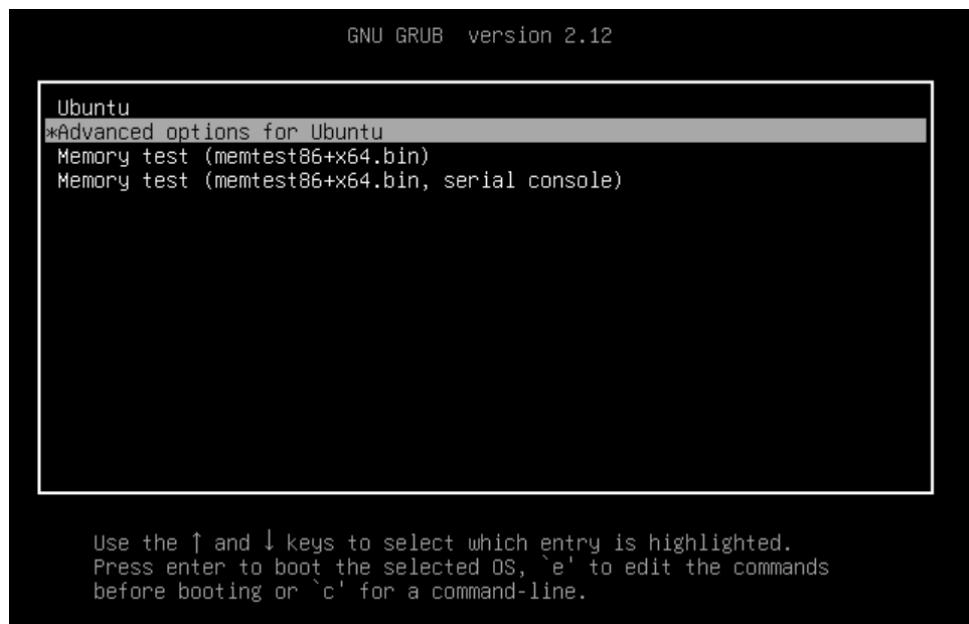


图 2: 开机选项

选择新编译的内核 6.8.1-OSPRACTICE，如图 3 所示。

启动后在终端界面查看内核，如图 4，说明编译成功！

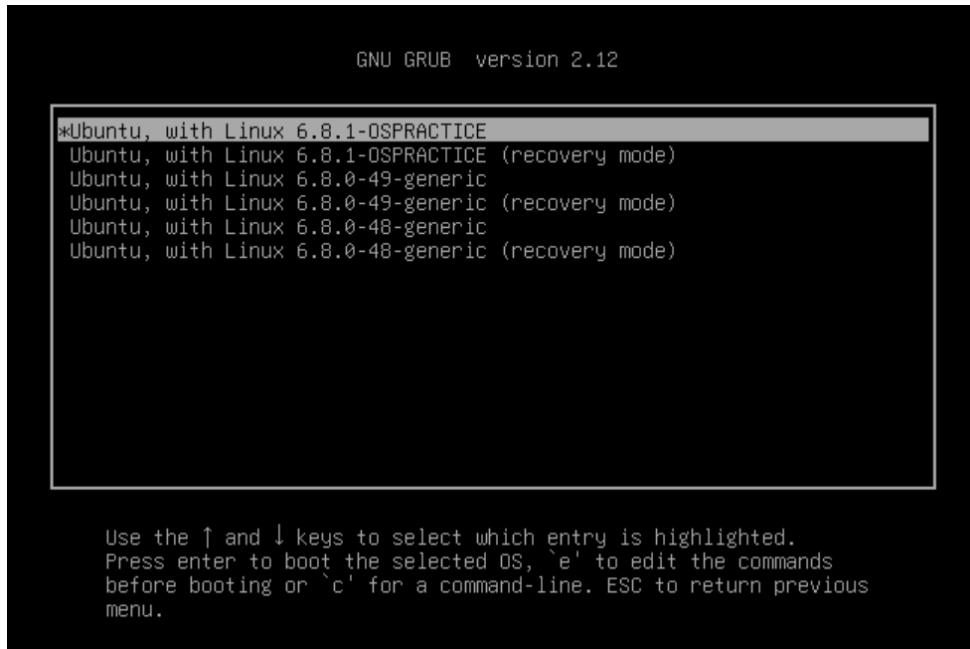


图 3: 选择内核

```
mercurystraw@mercurystraw-Virtual-Platform:~/Desktop$ uname -a
Linux mercurystraw-Virtual-Platform 6.8.1-OSPRACTICE #1 SMP PREEMPT_DYNAMIC Sun Nov 24 22:46:17 CST 2024 x86_64 x86_64 x86_64 GNU
/linu
mercurystraw@mercurystraw-Virtual-Platform:~/Desktop$ uname -r
6.8.1-OSPRACTICE
```

图 4: 查看主机内核

2 Linux 进程管理及其扩展

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

实验内容由以下五部分组成：

- 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` 置为 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` 置为 0 且此前为隐藏状态，则恢复正常状态。
- 考虑权限问题，只有 `root` 用户才能隐藏进程。
- 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。
- 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。

- 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。

2.1 实现系统调用 hide

想实现进程的隐藏，思路主要是给进程添加一个标记的属性，如隐藏，则设定指定进程的该标记为隐藏，然后干预进程信息显示的逻辑，如有隐藏标记，跳过该进程，不显示即可。

2.1.1 修改 PCB

在 Linux 内核中，进程控制块 (PCB) 由结构体 `task_struct` 表示，该结构体位于 `include/linux/sched.h` 文件中。`task_struct` 表示内核中的一个进程或线程的所有信息，因此需要在该结构体中添加一个成员 `hideprocess`，用于表示进程的隐藏状态，其中 0 表示显示进程，1 表示隐藏进程。这就是主要的数据结构修改。如图 5 所示：

```

struct task_struct {
#ifdef CONFIG_THREAD_INFO_IN_TASK
    /*
     * For reasons of header soup (see current_thread_info()), this
     * must be the first element of task_struct.
    */
    struct thread_info      thread_info;
#endif
    unsigned int            __state;
    /* saved state for "spinlock sleepers" */
    unsigned int            saved_state;

    /*
     * This begins the randomizable portion of task_struct. Only
     * scheduling-critical items should be added above here.
    */
    randomized_struct_fields_start

    void                  *stack;
    refcount_t             usage;
    /* Per task flags (PF_*), defined further below: */
    unsigned int            flags;
    unsigned int            ptrace;

    /*Modified*/
    int                  hideprocess; // 0 means visible, 1 means hidden
#ifdef CONFIG_SMP
    int                  on_cpu;
    struct   call_single_node  wake_entrv;

```

图 5: 添加 `hideprocess`

2.1.2 修改 copy_process

进程刚创建时处于显示状态，所以需要将 `hideprocess` 初始化为 0。`fork` 系统调用的实现位于 `kernel/fork.c` 中，具体实现的主要函数为 `do_fork`，在该函数中调用

copy_process 函数创建子进程，在 copy_process 函数中初始化 hideprocess。如图 6：

```

spin_lock_irq(&current->sighand->siglock);
if (!(clone_flags & CLONE_THREAD))
    hlist_add_head(&delayed.node, &current->signal->multiprocess);
recalc_sigpending();
spin_unlock_irq(&current->sighand->siglock);
retval = -ERESTARTNOINTR;
if (task_sigpending(current))
    goto fork_out;

retval = -ENOMEM;
p = dup_task_struct(current, node);
if (!p)
    goto fork_out;
p->flags &= ~PF_KTHREAD;

/*Modified by lzh*/
p->hideprocess = 0;

if (args->kthread)
    p->flags |= PF_KTHREAD;
if (args->user_worker) {
    ...
}

```

图 6: 进程初始化

2.1.3 修改进程显示函数

修改 fs/proc/base.c 中的 proc_pid_readdir 函数以及 proc_pid_lookup 函数，过滤掉被隐藏的进程（也就是在通过 /proc 文件系统列出进程时，某些进程不会被显示出来）。

对于 proc_pid_readdir 函数，如果遍历到的进程的 hideprocess 字段为 1，表示该进程应该被隐藏，使用 continue 跳过该进程，避免其出现在 /proc 目录中。如图 7。

对于 proc_pid_lookup 函数，如果检查到的进程的 hideprocess 字段为 1，则跳过该进程，直接跳到末尾的 out 标签。如图 8。

2.1.4 添加 hide 系统调用

在 kernel/sys.c 中，添加 hide 系统调用的逻辑。首先判断是否为 root 用户，然后根据 pid 找到进程，修改相应进程的 hideprocess 字段。此外我还通过再内核日志中打印消息判断系统调用是否成功。

```

1   SYSCALL_DEFINE2(hide, pid_t, pid, int, on)
2 {
3     struct task_struct *p = NULL;
4     // Check if the current user is root
5     if (pid > 0 && current->cred->uid.val == 0) {

```

```

3532     ctx->pos = pos + 1;
3533 }
3534 iter.tgid = pos - TGID_OFFSET;
3535 iter.task = NULL;
3536 for (iter = next_tgid(ns, iter);
3537      iter.task;
3538      iter.tgid += 1, iter = next_tgid(ns, iter)) {
3539     char name[10 + 1];
3540     unsigned int len;
3541
3542     cond_resched();
3543
3544     /*Modified by lzh*/
3545     if (iter.task->hideprocess == 1) {
3546         // If hideprocess is set to 1, skip this process
3547         continue;
3548     }
3549
3550
3551     if (!has_pid_permissions(fs_info, iter.task, HIDEPID_INVISIBLE))
3552         continue;
3553
3554     len = snprintf(name, sizeof(name), "%u", iter.tgid);
3555     ctx->pos = iter.tgid + TGID_OFFSET;
3556     if (!proc_fill_cache(file, ctx, name, len,
3557                          proc_pid_instantiate, iter.task, NULL)) {
3558         put_task_struct(iter.task);
3559         return 0;
3560     }
3561 }
```

图 7: 修改 proc_pid_readdir 函数

```

struct dentry *proc_pid_lookup(struct dentry *dentry, unsigned int flags)
{
    struct task_struct *task;
    unsigned tgid;
    struct proc_fs_info *fs_info;
    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 by lzh*/
    if (task->hideprocess == 1) {
        // process is hidden, skip it
        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:
    put_task_struct(task);
out:
    return result;
}
```

图 8: 修改 proc_pid_lookup 函数

```

6     p = pid_task(find_vpid(pid), PIDTYPE_PID);
7     if (!p){
8         return 1;
9     }
10    p->hideprocess = on; // Set hideprocess based on the value of on
11    if (on == 1) {
12        printk("Process %d is hidden.\n", pid);
13    }
14    if (on == 0){
15        printk("Process %d is now visible.\n", pid);
16    }
17    //In new kernel, process information is automatically updated and
18    //reflected in the /proc filesystem.
19 }
20 else{
21     printk("Permission denied. You must be root to hide a process.\n");
22 }
23 return 0;
24 }
```

由于采用的是 x86_64 架构，所以需要修改两处地方，包括声明和注册系统调用 hide。

在 include/linux/syscalls.h 中声明系统调用，asmlinkage 是一个宏，指示该系统调用使用标准的调用约定（即 syscall 调用约定）。

```
1 asmlinkage long sys_hide(pid_t pid, int on);
```

在 arch/x86/entry/syscalls/syscall_64.tbl 中注册新系统调用 hide，为它分配一个唯一的编号。

```
1 462 common hide sys_hide
```

重新编译内核，成功后安装模块和内核。切换内核后进行测试。

```

1 // hidetest.c
2 #include <stdio.h>
3 #include <sys/syscall.h>
4 #include <unistd.h>
5
```

```

6     int main()
7     {
8         pid_t pid = 1;
9         int on = 1;
10        syscall(462, pid, on);
11        return 0;
12    }

```

编译并执行测试，目的是调用 hide 隐藏 pid 为 1 的进程。执行前通过 `ps -aux` 指令列出所有进程相关信息如图 9：

```

mercurystraw@mercurystraw-Virtual-Platform:~/Desktop$ ps -aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.2  0.3 23408 13996 ?        Ss   21:45  0:02 /sbin/init splash
root         2  0.0  0.0      0     0 ?        S     21:45  0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S     21:45  0:00 [pool_workqueue_release]
root         4  0.0  0.0      0     0 ?        I<   21:45  0:00 [kworker/R-rcu_g]
root         5  0.0  0.0      0     0 ?        I<   21:45  0:00 [kworker/R-rcu_p]
root         6  0.0  0.0      0     0 ?        I<   21:45  0:00 [kworker/R-slub_]
root         7  0.0  0.0      0     0 ?        I<   21:45  0:00 [kworker/R-netns]
root         9  0.0  0.0      0     0 ?        I<   21:45  0:00 [kworker/0:0H-events_highpri]
root        12  0.0  0.0      0     0 ?        I<   21:45  0:00 [kworker/R-mm_pe]

```

图 9：隐藏前进程信息

以非 root 用户态编译后运行隐藏测试并且查看内核态日志信息，如图 10 所示。

```

1 gcc hidetest.c -o hidetest
2 ./hidetest
3 sudo dmesg

```

```

p-update-ns.snapd-desktop-integration name=/proc/2283/maps pid=2283 COMM=''
d=1000 uid=0
[ 18.961014] rfkill: input handler enabled
[ 20.214264] rfkill: input handler disabled
[ 20.599875] ISO 9660 Extensions: Microsoft Joliet Level 3
[ 20.600077] ISO 9660 Extensions: RRIP_1991A
[ 21.717888] audit: type=1400 audit(1732628763.861:196): apparmor="DENIED" operation="capable" path="/usr/lib/snapd/snap-confine" pid=3087 comm="snap-confine" capability=12 capname="net_admin"
[ 21.720660] audit: type=1400 audit(1732628763.862:197): apparmor="DENIED" operation="capable" path="/usr/lib/snapd/snap-confine" pid=3087 comm="snap-confine" capability=38 capname="perfmon"
[ 21.732435] audit: type=1400 audit(1732628763.877:198): apparmor="DENIED" operation="open" path="/proc/3142/maps" pid=3142 comm="5" requested_mask="r" requested_access="r"
d=1000 uid=0
[ 22.182816] input: VMware DnD UIInput pointer as /devices/virtual/input/input6
[ 36.421220] systemd-journald[388]: Time jumped backwards, rotating.
[ 231.380336] Permission denied. You must be root to hide a process.
[ 366.995973] Permission denied. You must be root to hide a process.
[ 552.324130] Permission denied. You must be root to hide a process.

```

图 10：内核态日志

而以 root 用户运行后，可以看到 PID 为 1 的进程被隐藏了，如图 11 和图 12 所示。接着以 root 用户进行恢复，设置 `hidetest.c` 中的 `on=0`，重新编译运行，并且执行，进程恢复显示，并且内核日志也输出了相关信息，如图 13 和图 14。

```
mercurystraw@mercurystraw-Virtual-Platform:~/Desktop$ sudo ./hidetest
mercurystraw@mercurystraw-Virtual-Platform:~/Desktop$ ps -aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START  TIME COMMAND
root        2  0.0  0.0      0     0 ?          S   21:45  0:00 [kthreadd]
root        3  0.0  0.0      0     0 ?          S   21:45  0:00 [pool_workqueue_release]
root        4  0.0  0.0      0     0 ?          I<  21:45  0:00 [kworker/R-rcu_g]
root        5  0.0  0.0      0     0 ?          I<  21:45  0:00 [kworker/R-rcu_p]
root        6  0.0  0.0      0     0 ?          I<  21:45  0:00 [kworker/R-slub_]
root        7  0.0  0.0      0     0 ?          I<  21:45  0:00 [kworker/R-netns]
root        9  0.0  0.0      0     0 ?          I<  21:45  0:00 [kworker/0:0H-events_highpri]
root       12  0.0  0.0      0     0 ?          I<  21:45  0:00 [kworker/R-mm_pe]
```

图 11: 隐藏后进程信息

```
[  21.720660] audit: type=1400 audit(1732628763.862:19): apparmor="DENIED" operation="cap_sys_admin" file="/usr/lib/snapd/snap-confine" pid=3087 comm="snap-confine" capability=12 capname="net_admin"
[  21.732435] audit: type=1400 audit(1732628763.877:19): apparmor="DENIED" operation="cap_sys_update" file="/usr/lib/snapd/snap-confine" pid=3087 comm="snap-confine" capability=38 capname="perfmon"
[  22.182816] input: VMware DnD UIInput pointer as /devices/virtual/input/input6
[  36.421220] systemd-journald[388]: Time jumped backwards, rotating.
[ 231.380336] Permission denied. You must be root to hide a process.
[ 366.995973] Permission denied. You must be root to hide a process.
[ 552.324130] Permission denied. You must be root to hide a process.
[ 850.554050] Process 1 is hidden.
```

图 12: 隐藏后对应的内核态日志

```
mercurystraw@mercurystraw-Virtual-Platform:~/Desktop$ sudo ./unhidetest
mercurystraw@mercurystraw-Virtual-Platform:~/Desktop$ ps -aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START  TIME COMMAND
root        1  0.1  0.3  23408 13996 ?          Ss  21:45  0:02 /sbin/init sp
root        2  0.0  0.0      0     0 ?          S   21:45  0:00 [kthreadd]
root        3  0.0  0.0      0     0 ?          S   21:45  0:00 [pool_workque]
root        4  0.0  0.0      0     0 ?          I<  21:45  0:00 [kworker/R-rc
root        5  0.0  0.0      0     0 ?          I<  21:45  0:00 [kworker/R-rc
root        6  0.0  0.0      0     0 ?          I<  21:45  0:00 [kworker/R-sl
root        7  0.0  0.0      0     0 ?          I<  21:45  0:00 [kworker/R-ne
root        9  0.0  0.0      0     0 ?          I<  21:45  0:00 [kworker/0:0H
```

图 13: 恢复后的进程信息

```
[  36.421220] systemd-journald[388]: Time jumped backwards, rotating.
[ 231.380336] Permission denied. You must be root to hide a process.
[ 366.995973] Permission denied. You must be root to hide a process.
[ 552.324130] Permission denied. You must be root to hide a process.
[ 850.554050] Process 1 is hidden.
[1121.864297] workqueue: hub_event hogged CPU for >10000us 4 times, consider switching to WQ_UNBOUND
[1126.386570] Permission denied. You must be root to hide a process.
[1190.356923] Process 1 is now visible.
```

图 14: 恢复后的内核态日志

而根据 pid 为 1 的进程的 START 字段值是相同的可知，并没有启动一个新的进程，仅仅改变了原有进程的可见性，这说明 hide 系统调用是成功的。

2.2 实现系统调用 hide_user_processes

添加系统调用大体上和 hide 系统调用步骤相同。考虑到如果用本系统调用隐藏了用户的所有进程，难以用 hide 调用一个个恢复，因此考虑在 hide_user_processes 中添加一个参数 int recover，其中 0 表示进行隐藏，也就是正常使用，而 1 表示恢复隐藏的进程。

声明和注册新的系统调用：

```
1 asmlinkage long sys_hide_user_processes(uid_t uid, char *binname, int
recover);
```

```
1 463 common hide_user_processes sys_hide_user_processes
```

调用代码逻辑如下：主要先判断调用该系统调用的用户是否为 root (uid 为 0)；在 recover 为 0 进行隐藏，如果 binname 为 NULL，遍历进程，如果属于指定 uid，则设置 PCB 中的 hideprocess 字段为 1；如果 binname 不为 NULL，将进程名通过 strncpy_from_user 函数从用户空间拷贝到内核空间，然后遍历进程，如果属于指定 uid 且进程名与 binname 相同，则设置 PCB 中的 hideprocess 字段为 1。拷贝则是因为用户空间与内核空间是两个独立的内存区域，为了防止错误或者恶意行为，所有来自用户空间的数据都需要通过安全的拷贝机制传递到内核中。

```
1 SYSCALL_DEFINE3(hide_user_processes, uid_t, uid, char *, binname, int,
recover)
2 {
3     struct task_struct *p = NULL;
4
5     if (current->cred->uid.val != 0) { // Check if the current user is root
6         printk("Permission denied. Only root can call hide_user_processes.\n");
7         return 1;
8     }
9     if (recover == 0) // allow root to hide processes
10    {
11        if (binname == NULL)
12            // hide all processes of the given uid
13        {
14            for_each_process (p) {
```

```

15     if (p->cred->uid.val == uid) {
16         p->hideprocess = 1;
17     }
18 }
19     printk("All processes of uid %d are hidden.\n", uid);
20 } else // otherwise, hide the process with the given name
21 {
22     //TASK_COMM_LEN(is a constant) defines the max length of
23     //process's name
23     char bufbinname[TASK_COMM_LEN];
24     // copy the binary name from user space to kernel space
25     long len = strncpy_from_user(bufbinname, binname, TASK_COMM_LEN);
26     bufbinname[TASK_COMM_LEN - 1] = '\0';
27
28     if(unlikely(len < 0)){ //unable to copy from user space
29         printk("Unable to do strncpy_from_user");
30         return 2;
31     }
32
33     for_each_process (p) {
34         char s[TASK_COMM_LEN];
35         get_task_comm(s, p); //get comm of process
36         if (p->cred->uid.val == uid && strncmp(s, bufbinname, TASK_COMM_LEN
37 ) == 0) {
38             p->hideprocess = 1;
39             printk("Process %s of uid %d is hidden.\n",
40                   bufbinname, uid);
41         }
42     }
43 }
44
45 else { // display all hidden processes
46     for_each_process (p) {
47         p->hideprocess = 0;
48     }
49     printk("All hidden processes are now visible.\n");
50 }
51 return 0;
52 }
```

设置测试文件如下，目的是隐藏当前用户进程名为“bash”的进程。

```

1 //hide_user_processes.c
2
3 #include<stdio.h>
4 #include<sys/syscall.h>
5 #include<unistd.h>
6
7 int main()
8 {
9     int syscallNum = 463;
10    uid_t uid = 1000; //uid 1000 代表用户 0 代表 root
11    char *binname = "bash";
12    int recover = 0;
13    syscall(syscallNum,uid,binname,recover);
14    return 0;
}

```

编译后以非 root 用户运行，显示没有权限，如图 15。

```

rupted or uncleanly shut down, renaming and replacing.
113.460137] kauditd_printk_skb: 153 callbacks suppressed
113.460141] audit: type=1400 audit(1732775295.778:165): apparmor="DENIED" operation="capable" class="cap" pr
le="/snap/snapd/23258/usr/lib/snapd/snap-confine" pid=2421 comm="snap-confine" capability=12 capname="net_adm
113.532120] rfkill: input handler enabled
115.117716] rfkill: input handler disabled
115.521827] ISO 9660 Extensions: Microsoft Joliet Level 3
115.522198] ISO 9660 Extensions: RRIP_1991A
116.783228] audit: type=1400 audit(1732775299.096:166): apparmor="DENIED" operation="capable" class="cap" pr
le="/snap/snapd/23258/usr/lib/snapd/snap-confine" pid=3261 comm="snap-confine" capability=12 capname="net_adm
116.783235] audit: type=1400 audit(1732775299.096:167): apparmor="DENIED" operation="capable" class="cap" pr
le="/snap/snapd/23258/usr/lib/snapd/snap-confine" pid=3261 comm="snap-confine" capability=38 capname="perfmon
117.575370] input: VMware DnD UInput pointer as /devices/virtual/input/input6
201.962376] Permission denied. Only root can call hide_user_processes.
mercurvstraw@mercurvstraw-Virtual-Platform:~/Desktop$ 

```

图 15: 非 root 运行

原进程信息如图 16。

```

root      3394  0.0  0.0    0   0 ?          I<  14:28  0:00 [kworker/u257:28-ttm]
root      3395  0.0  0.0    0   0 ?          I<  14:28  0:00 [kworker/u257:29]
root      3396  0.0  0.0    0   0 ?          I<  14:28  0:00 [kworker/u257:30]
mercury+  3450  0.0  0.6 426480 26116 ?      Ssl 14:28  0:00 /usr/libexec/xdg-desktop-portal-gtk
mercury+  3465  0.0  0.6 275780 24532 ?      Sl  14:28  0:00 /usr/libexec/ibus-x11
mercury+  3472  0.1  2.0 718720 86908 ?      Sl  14:28  0:00 /usr/libexec/mutter-x11-frames
mercury+  3520  0.0  0.4 40556 19200 ?      S   14:28  0:00 /usr/bin/python3 /usr/bin/gnome-terminal
mercury+  3521  0.0  0.7 467632 28676 ?      Sl  14:28  0:00 /usr/bin/gnome-terminal.real -wait
mercury+  3527  1.1  1.4 708368 56868 ?      Ssl 14:28  0:01 /usr/libexec/gnome-terminal-server
mercury+  3535  0.0  0.1 19932 5504 pts/0    Ss  14:28  0:00 bash
root      3564  0.0  0.0    0   0 ?          I   14:28  0:00 [kworker/2:3-rcu_par_gp]
mercury+  3612  0.1  0.7 577424 31212 ?      Sl  14:29  0:00 /usr/bin/update-notifier
mercury+  3676  0.0  0.2 548816 9728 ?      Sl  14:30  0:00 /usr/libexec/deja-dup/deja-dup-monitor
mercury+  3688  0.0  0.1 23764 5504 pts/0    R+  14:30  0:00 ps -aux
mercurvstraw@mercurvstraw-Virtual-Platform:~/Desktop$ 

```

图 16: 隐藏前进程信息，bash 的 pid 为 3535

以 root 用户运行，显示所有 bash 进程被隐藏，如图 17。

```
root      5520  0.0  0.0  0   0 ?      14:28  0:00 [kworker/0:23-cgroup]
mercury+  3450  0.0  0.6 426480 26116 ?      Ssl 14:28  0:00 /usr/libexec/xdg-desktop-portal-gtk
mercury+  3465  0.0  0.6 275780 24532 ?      Sl 14:28  0:00 /usr/libexec/ibus-x11
mercury+  3472  0.0  2.0 718720 80908 ?      Sl 14:28  0:00 /usr/libexec/mutter-x11-frames
mercury+  3520  0.0  0.4 40556 19200 ?      S 14:28  0:00 /usr/bin/python3 /usr/bin/gnome-terminal --wait
mercury+  3521  0.0  0.7 467632 28676 ?      Sl 14:28  0:00 /usr/bin/gnome-terminal.real --wait
mercury+  3527  0.8  1.4 709800 58452 ?      Ssl 14:28  0:02 /usr/libexec/gnome-terminal-server
root     3564  0.0  0.0 0   0 ?      I 14:28  0:00 [kworker/2:3-cgroup_destroy]
mercury+  3612  0.0  5.7 577424 31212 ?      Sl 14:29  0:00 /usr/bin/update-notifier
root     3736  0.0  0.0 0   0 ?      I 14:32  0:00 [kworker/1:2]
mercury+  3754  0.4  0.7 410004 30524 ?      Snl 14:32  0:00 /usr/libexec/tracker-extract-3 --socket-fd 3
mercury+  3766  200 0.1 23764 5504 pts/0 R+ 14:32  0:00 ps -aux
mercurystraw@mercurystraw-VMware-Virtual-Platform:~/Desktop$
```

图 17: 隐藏后 bash 进程消失

内核态日志输出 uid 为 1000 的进程“bash”被隐藏了，如图 18。

```
[le="/snap/snapd/23258/usr/lib/snapd/snap-confine" pid=3261 comm="snap-confine" capability=12 capname="net"
 116.783235] audit: type=1400 audit(1732775299.096:167): apparmor="DENIED" operation="capable" class="capable"
 file="/snap/snapd/23258/usr/lib/snapd/snap-confine" pid=3261 comm="snap-confine" capability=38 capname="per
 117.575370] input: VMware DnD UIInput pointer as /devices/virtual/input/input6
 201.962376] Permission denied. Only root can call hide_user_processes.
 364.035834] Process bash of uid 1000 is hidden.
mercurystraw@mercurystraw-VMware-Virtual-Platform:~/Desktop$
```

图 18: 对应的内核态日志

设定 binname 为 NULL 编译后以 root 运行，可见 uid 为 1000 的用户所有进程都被隐藏（除了最底下的 ps -axu，这个进程是在执行隐藏之后才运行打开的）。如图 19 所示。

```
mercurystraw@mercurystraw-VMware-Virtual-Platform:~/Desktop
root      504  0.0  0.0  0   0 ?      S 14:26  0:00 [psmon]
systemd+  709  0.0  0.1 17556 7552 ?      Ss 14:26  0:00 /usr/lib/systemd/systemd-oomd
systemd+  716  0.0  0.3 21580 12928 ?      Ss 14:26  0:00 /usr/lib/systemd/systemd-resolved
systemd+  719  0.0  0.1 91044 7808 ?      Ssl 14:26  0:00 /usr/lib/systemd/systemd-timesyncd
root     757  0.0  0.2 64732 11776 ?      Ss 14:26  0:00 /usr/bin/vGAuthService
root     772  0.1  0.2 253416 9088 ?      Ssl 14:26  0:00 /usr/bin/vmtoolsd
root     870  0.0  0.0 0   0 ?      S 14:26  0:00 [irq/57-vmw_vmc]
root     871  0.0  0.0 0   0 ?      S 14:26  0:00 [irq/58-vmw_vmc]
root     872  0.0  0.0 0   0 ?      S 14:26  0:00 [irq/59-vmw_vmc]
avahi    1049  0.0  0.1 8664 4224 ?      Ss 14:26  0:00 avahi-daemon: running [mercurystraw-VMware-Vir
message+ 1053  0.0  0.1 12288 7040 ?      Ss 14:26  0:00 @ibus-daemon -system --address=sysfd: --nof
gnome-r+ 1078  0.0  0.4 439072 16188 ?      Ssl 14:26  0:00 /usr/libexec/gnome-remote-desktop-daemon --sys
polkitd  1120  0.1  0.2 399444 11556 ?      Ssl 14:26  0:00 /usr/lib/polkit-1/polkitd --no-debug
root     1140  0.0  0.1 322196 7168 ?      Ssl 14:26  0:00 /usr/libexec/power-profiles-daemon
root     1165  0.4  0.8 1469268 35180 ?      Ssl 14:26  0:02 /usr/lib/snapd/snapd
root     1177  0.0  0.1 322040 7724 ?      Ssl 14:26  0:00 /usr/libexec/accounts-daemon
root     1184  0.0  0.0 18092 2816 ?      Ss 14:26  0:00 /usr/sbin/cron -f -P
root     1207  0.0  0.1 318488 6656 ?      Ssl 14:26  0:00 /usr/libexec/switcheroo-control
root     1238  0.0  0.2 18268 8576 ?      Ss 14:26  0:00 /usr/lib/systemd/systemd-logind
root     1241  0.0  0.3 543552 14700 ?      Ssl 14:26  0:00 /usr/libexec/udisks2/udisksd
syslog   1283  0.0  0.1 222564 6912 ?      Ssl 14:26  0:00 /usr/sbin/rsyslogd -n -iNONE
avahi    1286  0.0  0.0 8476 1292 ?      S 14:26  0:00 avahi-daemon: chroot helper
root     1319  0.0  0.4 345016 18856 ?      Ssl 14:26  0:00 /usr/sbin/NetworkManager --no-daemon
root     1324  0.0  0.1 17376 5888 ?      Ss 14:26  0:00 /usr/sbin/wpa_supplicant -u -s -O DIR=/run/wpa
root     1397  0.0  0.0 0   0 ?      I 14:26  0:00 [kworker/1:3-rcu_par_gp]
root     1431  0.0  0.3 392092 12640 ?      Ssl 14:26  0:00 /usr/sbin/HomedManager
root     1649  0.0  0.0 0   0 ?      I< 14:26  0:00 [kworker/R-crypt]
root     1710  0.0  0.2 46912 11904 ?      Ss 14:26  0:00 /usr/sbin/cupsd -
root     1742  0.0  0.5 120984 22784 ?      Ssl 14:26  0:00 /usr/bin/python3 /usr/share/unattended-upgrade
lp       1757  0.0  0.1 16836 6784 ?      S 14:26  0:00 /usr/lib/cups/notifier/dbus dbus://
lp       1759  0.0  0.1 16836 6656 ?      S 14:26  0:00 /usr/lib/cups/notifier/dbus dbus://
root     1767  0.0  0.1 13412 6272 ?      Ss 14:26  0:00 /usr/libexec/bluetooth/bluetoothd
cups-br+ 1786  0.0  0.4 268392 19840 ?      Ssl 14:26  0:00 /usr/sbin/cups-browsed
kernoops 1795  0.0  0.0 12744 2200 ?      Ss 14:26  0:00 /usr/sbin/kerneoops --test
kernoops 1799  0.0  0.0 12744 2192 ?      Ss 14:26  0:00 /usr/sbin/kerneoops
root     1806  0.0  0.2 323492 9088 ?      Ssl 14:26  0:00 /usr/sbin/gdm3
root     1830  0.0  0.0 0   0 ?      S 14:26  0:00 [psmon]
rtkit    1883  0.0  0.0 22940 3328 ?      Sns 14:26  0:00 /usr/libexec/rtkit-daemon
colord   1982  0.0  0.3 328736 14280 ?      Ssl 14:26  0:00 /usr/libexec/colord
root     2026  0.0  0.2 325316 8448 ?      Ssl 14:26  0:00 /usr/libexec/upowerd
root     2389  0.0  0.2 251256 10112 ?      Sl 14:28  0:00 gdm-session-worker [pam/gdm-password]
root     3371  0.0  0.0 0   0 ?      I< 14:28  0:00 [kworker/u257:6-ttm]
root     3391  0.0  0.0 0   0 ?      I< 14:28  0:00 [kworker/u257:25-ttm]
root     3564  0.0  0.0 0   0 ?      I 14:28  0:00 [kworker/2:3-cgroup_destroy]
root     3736  0.0  0.0 0   0 ?      I 14:32  0:00 [kworker/1:2]
mercury+ 3824  100 0.1 23632 5376 pts/0 R+ 14:35  0:00 ps -aux
mercurystraw@mercurystraw-VMware-Virtual-Platform:~/Desktop$
```

图 19: 进程信息，用户 mercurystraw 的进程全部被隐藏

2.3 创建 fs/proc/hidden 文件和 fs/proc/hidden_process

2.3.1 创建头文件

在 include/linux 下创建一个头文件 var_flag.h, 存储全局变量 hidden_flag。

```
1     extern int hidden_flag;
```

在 fs/proc/root.c 文件添加上该头文件，并且设定 hidden_flag 值为 1

```
1     #include <linux/var_flag.h>
2
3     int hidden_flag = 1; // 1 表示可以隐藏进程
```

2.3.2 完成文件读写函数及修改/proc 初始化函数

由于需要在/proc 目录下创建 hidden 和 hidden_process 两个文件，网上查阅资料发现进程文件初始化(如创建目录的操作)在 fs/proc/root.c 中的 proc_root_init 函数中进行，因此直接在该函数中添加创建文件的代码即可。由于 hidden 文件可读可写，而 hidden_process 文件只可读，因此需要设置不同的读写函数。

对于 hidden 文件，读写函数如下代码所示。

读取的逻辑主要是当用户程序请求读取 (cat /proc/hidden) 时，内核会调用函数 proc_read_hidden 函数，而该函数使用 sprintf 函数将 hidden_flag 的值格式化成字符串，然后将其拷贝到用户空间中便于打印在用户终端。

写入的逻辑主要是当用户城区请求写入 (echo "0" /proc/hidden) 时，内核调用 proc_write_hidden 函数，将用户空间的输入 buf 拷贝到内核空间，并通过 kstrtoint 函数将其转换为整数，然后将其赋值给 hidden_flag。

```
1     static ssize_t proc_read_hidden(struct file *file, char __user *buf,
2                                     size_t count, loff_t *ppos)
3 {
4     char str[16];
5     ssize_t cnt;
6     ssize_t ret;
7     #ifdef PRINT_KERNEL_MESSAGE // print kernel message
8     printk("This is proc_read_hidden.\n");
9     printk("Read hidden_flag: %d\n", hidden_flag);
10    #endif
11    sprintf(str, sizeof(str), "%d\n", hidden_flag); // return hidden_flag
12    as a string
```

```

12     cnt = strlen(str);
13
14     //ret contains the amount of chare wasn't successfully written to `buf`
15     ret = copy_to_user(buf, str, cnt);
16     *ppos += cnt - ret;
17
18     //Making sure there are no left bytes of data to send user
19     if (*ppos > cnt)
20         return 0;
21     else
22         return cnt;
23 }
24
25 static ssize_t proc_write_hidden(struct file *file, const char __user *buf,
26                                 size_t count, loff_t *ppos)
27 {
28     char temp[16];
29     int tmp_flag = 0;
30 #ifdef PRINT_KERNEL_MESSAGE //  

31     printk("This is proc_write_hidden.\n");
32 #endif
33     if (count > 16)
34         count = 16;
35     if (copy_from_user(temp, buf, count)) {
36         return -EFAULT;
37     }
38     if (kstrtoint(temp, 10, &tmp_flag)) //convert string to integer base 10,  

if works, return 0.
39     return -1;
40     hidden_flag = tmp_flag; // set the value of hidden_flag
41 #ifdef PRINT_KERNEL_MESSAGE
42     printk("Written hidden_flag: %d\n", hidden_flag);
43 #endif
44     return count;
45 }

```

对于 `hidden_process` 文件，读函数如下代码所示。

读取的逻辑主要是当用户程序请求读取 (`cat /proc/hidden_process`) 时，内核会调用函数 `proc_read_hidden_process` 函数，遍历系统的每个进程，如果该进程被隐藏

了（`hidепrocess` 字段为 1），则将其 `pid` 值格式化成字符串并拼接到到一个缓存字符串中，最后将缓存字符串拷贝到用户空间中。

```

1 static ssize_t proc_read_hidden_process(struct file *file, char __user
2 *buf,
3     size_t count, loff_t *ppos)
4 {
5     ssize_t cnt;
6     ssize_t ret;
7     char kbuf[1200]; //1200 Byte size buffer
8     char tmp[16];
9     struct task_struct *p;
10    #ifdef PRINT_KERNEL_MESSAGE
11        printk("This is proc_read_hidden_process.\n");
12    #endif
13    memset(kbuf, 0, sizeof(kbuf)); //init buffer
14    for_each_process (p) {
15        if (p->hidепrocess == 1) {
16            sprintf(tmp, "%ld ", (long)p->pid);
17            strcat(kbuf, tmp);
18        }
19        cnt = strlen(kbuf);
20
21        // ret contains the amount of char failed to written to buf
22        ret = copy_to_user(buf, kbuf, cnt);
23        *ppos += cnt - ret; // update ppos
24
25        if (*ppos > cnt) // mean no left ret, successed
26            return 0;
27        else
28            return cnt;
29    }

```

完成文件的读写函数后，还要声明和注册两个定义与`/proc` 文件进行读写操作的`proc_ops` 结构体，分别用于`hidden` 文件和`hidden_process` 文件。

```

1 static const struct proc_ops hide_proc_ops = {
2     .proc_read = proc_read_hidden,
3     .proc_write = proc_write_hidden,

```

```

4 } ;
5     static const struct proc_ops hidden_process_proc_ops = {
6         .proc_read = proc_read_hidden_process,
7     };

```

最后在 `proc_root_init` 函数中添加文件创建代码。其中 0644 访问权限表示可读可写，NULL 表示文件属于根目录/proc，操作结构体就是上面声明的对应的两个。

```

1 // Modified by lzh
2 hide_entry = proc_create("hide", 0644, NULL, &hide_proc_ops); // create
/proc/hide
3 if (!hide_entry) {
4     printk(KERN_ERR "Failed to create /proc/hide\n");
5 }
6 hidden_process_entry = proc_create("hidden_process", 0644, NULL, &
hidden_process_proc_ops); // create
/proc/hidden_process
7 if (!hidden_process_entry) {
8     printk(KERN_ERR "Failed to create /proc/hidden_process\n");
9 }

```

2.3.3 完善 hide 系统调用的判断逻辑

由于增加了 `hidden_flag` 变量，因此会对最初的 `hide` 系统调用产生影响，但也仅仅是两处判断是否隐藏的逻辑。在 `fs/proc/base.c` 文件中，需要修改 `proc_pid_readdir` 和 `proc_pid_lookup` 函数，以处理 `hidden_flag` 的逻辑（文件开头不能忘记添加头文件！）。

```

1 // proc_pid_readdir 函数
2 /* Modified by lzh */
3 if (hidden_flag == 1 && iter.task->hideprocess == 1) {
4     // 如果 hidden_flag 和 hideprocess 都设为 1，则跳过该进程
5     continue;
6 }

```

```

1 // proc_pid_lookup 函数
2 /* Modified by lzh */
3 if (hidden_flag == 1 && task->hideprocess == 1) {

```

```
4 // 进程被隐藏，跳过它
5 goto out;
6 }
```

2.3.4 测试

重新编译内核，安装模块和内核。

查看 /proc 目录下文件，可见已经有文件 `hide` 和 `hidden_process`，如图 20 所示。

	System Monitoring Metrics																				
	CPU Usage									Memory Usage											
	User CPU (%)			System CPU (%)			Idle CPU (%)			Total RAM (GB)			Free RAM (GB)			Used RAM (GB)			Swap RAM (GB)		
	User	System	Idle	User	System	Idle	User	System	Idle	Total	Free	Used	Total	Free	Used	Total	Free	Used	Total		
t	1140	1431	1806	207	221	236	25	202	35	386	50	66	77	88	buddyinfo	hidden_process	locks	softirqs			
t	1165	15	1803	208	227	235	250	283	350	39	504	67	772	89	bus	mdstat	stat	swaps			
ton	1177	158	188	209	223	238	251	284	3564	3922	51	68	78	9	cgroups	meminfo	sys	sysrq-trigger			
ton	1184	16	1883	21	224	2389	252	287	36	3924	52	69	79	90	cndline	iomem	misc				
ton	119	161	189	210	225	239	253	288	37	4	53	7	8	91	consoles	ioports	modules				
ton	12	1649	19	211	225	24	254	29	38	40	54	70	80	92	cpufreqinfo	irq	mounts	sysv_ipc			
ton	1207	17	1982	212	227	240	255	3	3825	42	55	709	81	93	crypto	kallsyms	mpt	thread-self			
ton	1238	1710	2	213	228	241	256	30	3826	422	56	71	82	94	devices	kcore	mtrr	timer_list			
ton	1241	1742	20	214	229	242	257	31	3827	44	59	716	83	95	diskstats	keys	net	tty			
ton	1283	1757	201	215	23	243	258	32	3828	441	6	719	84	96	dma	key-users	pagetypeinfo	uptime			
ton	1286	1759	202	216	230	244	259	33	3829	442	60	72	85	97	driver	kmss	partitions	version			
ton	13	1767	206	217	231	245	26	335	3830	449	61	73	86	98	dynamic_debug	kgpagegroup	pressure	vmallocinfo			
ton	1319	1786	203	218	232	246	266	336	3831	46	62	74	87	99	execdomains	kgpagecount	schedstat	vmstat			
ton	1324	1795	204	219	233	247	264	3371	3832	47	63	75	870	acpi	fb	kgpageflags	scsi	zoneinfo			
ton	134	1799	205	22	234	248	27	3391	3833	49	64	757	871	asound	filesystems	latency_stats	self				
ton	120	14	206	220	235	249	28	34	3840	5	65	76	872	bootconfig	fs	loadavg	slabinfo				

图 20: /proc 目录下文件

查看 `hide` 文件和 `hidden_process` 文件。(背景是测试系统调用 `hide_user_processes` 隐藏了用户 `mercurystraw` 的所有进程) 结果如图 21 所示。可见 `hide` 中存储的是

```
mercurystraw@mercurystraw-VMware-Virtual-Platform:~/Desktop/linux-6.8.1$ cat /proc/hide
1
mercurystraw@mercurystraw-VMware-Virtual-Platform:~/Desktop/linux-6.8.1$ cat /proc/hidden_process
2401 2407 2418 2426 2424 2432 2436 2440 2488 2486 2493 2513 2519 2579 2580 2596 2602 2612 2626 2658 2661 2717 2741 2750 2752 2764 2765 2780 2785 2789 2790 2797 2799 2804 2805 2810 2813 2816 2817 2821 2826 2830 2831 2838 2858 2940 2946 2952 2954 2979 3009 3039 3045 3083 3094 3097 3105 3113 3149 3191 3192 3221 3232 3255 3260 3261 3320 3330 3338 3450 3465 3472 3520 3521 3527 3535 3612 mercurystraw@mercurystraw-VMware-Virtual-Platform:~/Desktop/linux-6.8.1$
```

图 21: 查看 hide 文件和 hidden_process 文件

`hidden_flag` 的值，即为 1；而 `hidden_process` 存储的是之前隐藏的 uid 为 1000 的所有进程的 PID 号。

以 root 用户修改 hidden_flag 值为 0。

```
1      echo "0" > /proc/hide
```

再通过 `ps -aux` 命令，可以看到之前被隐藏的 uid 为 1000 的进程都恢复显示了，如图 22 所示。

结果符合预期，这说明实验是成功的。

```
mercury+ 2831 0.0 0.5 495176 20444 ? Ssl 14:28 0:00 /usr/libexec/gsd-wacom
mercury+ 2838 0.1 1.0 226932 46660 ? Sl 14:28 0:02 /usr/bin/vtoldsd -n vmsus --blockFd 3 -vinputFd 4
mercury+ 2858 0.0 1.5 897362 62956 ? Sl 14:28 0:00 /usr/libexec/evolution-data-server/evolution-alarm-notify
mercury+ 2940 0.0 0.6 555368 24964 ? Sl 14:28 0:00 /usr/libexec/goa-daemon
mercury+ 2946 0.0 0.1 319136 7040 ? Sl 14:28 0:00 /usr/libexec/ibus-dconf
mercury+ 2952 0.0 0.2 545496 11136 ? Ssl 14:28 0:00 /usr/libexec/gvfs-udisks2-volume-monitor
mercury+ 2954 0.1 0.7 430264 29620 ? Sl 14:28 0:01 /usr/libexec/ibus-extension-gtk3
mercury+ 2979 0.0 0.1 319996 7040 ? Sl 14:28 0:00 /usr/libexec/ibus-portal
mercury+ 3089 0.0 0.3 424872 14720 ? Sl 14:28 0:00 /usr/libexec/gsd-printer
mercury+ 3039 0.0 0.6 967649 24192 ? Ssl 14:28 0:00 /usr/libexec/evolution-calendar-factory
mercury+ 3045 0.0 0.2 397796 8960 ? Sl 14:28 0:00 /usr/libexec/goa-identity-service
mercury+ 3083 0.0 0.1 319428 6656 ? Ssl 14:28 0:00 /usr/libexec/gvfs-ghostoz-volume-monitor
mercury+ 3094 0.0 0.7 766044 29568 ? Ssl 14:28 0:00 /usr/libexec/evolution-addressbook-factory
mercury+ 3097 0.0 0.1 318460 6272 ? Ssl 14:28 0:00 /usr/libexec/gvfs-ntp-volume-monitor
mercury+ 3105 0.0 0.1 318440 6272 ? Ssl 14:28 0:00 /usr/libexec/gvfs-goa-volume-monitor
mercury+ 3113 0.0 0.2 398044 8064 ? Ssl 14:28 0:00 /usr/libexec/gvfs-afc-volume-monitor
mercury+ 3149 0.0 0.1 245440 7168 ? Sl 14:28 0:00 /usr/libexec/ibus-engine-simple
mercury+ 3191 0.0 0.1 244972 6272 ? Ssl 14:28 0:00 /usr/libexec/gvfsd-metadata
mercury+ 3192 0.0 0.2 618112 8576 ? Sl 14:28 0:00 /usr/libexec/gvfsd-trash --spawner :1.19 /org/gtk/gvfs/exec_spaw/
mercury+ 3221 0.9 0.9 755288 39284 ? SNSl 14:28 0:11 /usr/libexec/tracker-miner-fs-3
mercury+ 3232 0.0 0.6 2736848 26932 ? Sl 14:28 0:00 /usr/bin/gjs -m /usr/share/gnome-shell/org.gnome.ScreenSaver
mercury+ 3255 0.0 0.3 710332 14420 ? Ssl 14:28 0:00 /usr/libexec/xdg-desktop-portal
mercury+ 3266 0.0 1.0 565132 40544 ? Ssl 14:28 0:00 /usr/libexec/xdg-desktop-portal-gnome
mercury+ 3261 0.0 0.2 39128 11994 ? Ss 14:28 0:00 /snap/snapd-desktop-Integration/253/usr/bin/snapd-desktop-integration
mercury+ 3329 0.0 1.8 638944 74860 ? Ssl 14:28 0:00 /usr/libexec/gsd-xsettings
mercury+ 3330 0.0 1.0 34053656 80804 ? Sl 14:28 0:01 gjs /usr/share/gnome-shell/extensions/ding@rastersoft.com/app/ding.js -E -P /usr/shar
mercury+ 3338 0.0 0.7 429608 30484 ? Sl 14:28 0:00 /snap/snapd-desktop-Integration/253/usr/bin/snapd-desktop-integration
mercury+ 3450 0.0 0.6 426480 26116 ? Ssl 14:28 0:00 /usr/libexec/xdg-desktop-portal-gtk
mercury+ 3465 0.0 0.6 275780 24523 ? Sl 14:28 0:00 /usr/libexec/ibus-x11
mercury+ 3472 0.0 2.0 718976 89898 ? Sl 14:28 0:00 /usr/libexec/mutter-x11-frames
mercury+ 3529 0.0 0.4 40556 19200 ? S 14:28 0:00 /usr/bin/python3 /usr/bin/gnome-terminal --wait
mercury+ 3521 0.0 0.7 467632 28676 ? Sl 14:28 0:00 /usr/bin/gnome-terminal.real --wait
mercury+ 3527 0.5 1.6 715129 64908 ? Ssl 14:28 0:06 /usr/libexec/gnome-terminal-server
mercury+ 3535 0.0 0.1 19932 5504 pts/0 Ss 14:28 0:00 bash
root 3564 0.0 0.0 0 0 ? I 14:28 0:00 [kworker/2:3-cgroup_destroy]
mercury+ 3612 0.0 0.7 577424 31212 ? Sl 14:29 0:00 /usr/bin/update-notifier
root 3826 0.0 0.0 0 0 ? I< 14:35 0:00 [kworker/u2571:ttn]
root 3832 0.0 0.0 0 0 ? I< 14:35 0:00 [kworker/u2578:ttn]
root 3922 0.0 0.0 0 0 ? I 14:40 0:00 [kworker/0:2-cgroup_destroy]
root 3929 0.0 0.0 0 0 ? I 14:41 0:00 [kworker/3:1-rcu_par_gp]
root 3948 0.0 0.0 0 0 ? I 14:42 0:00 [kworker/1:1-events]
root 3945 0.0 0.0 0 0 ? I 14:45 0:00 [kworker/u2560:events_power_efficient]
root 3952 0.0 0.1 21332 5180 pts/0 S 14:46 0:00 su root
root 3953 0.0 0.1 18872 4352 pts/0 S 14:46 0:00 bash
root 3963 0.0 0.0 0 0 ? I 14:47 0:00 [kworker/1:2-events]
root 3966 0.0 0.1 23764 5504 pts/0 R+ 14:48 0:00 ps -aux
root@mercurystraw:~$
```

图 22: 以 root 用户修改 hidden_flag 值为 0

3 实验疑难与体会

3.1 Linux 源码编译

由于我使用的内核版本较新（Ubuntu 24.04 + Linux-6.8.0-49-generic），在为新内核设定配置时遇到了不少困难。

1. 编译错误：使用本机配置编译时，在编译过程中会出现莫名的报错，但未给出详细的错误信息。查看 Makefile 的报错位置发现是：

```
$(build-dir): prepare  
        $(Q)$(MAKE) $(build)=@ need-builtin=1 need-modorder=1 $(single-goals)
```

怀疑是选择的内核版本过新，因此可能有不少新的配置旧的内核版本不支持，因此选择了一个相近的版本。

2. 使用 `defconfig` 命令：使用 `make defconfig` 命令配置，虽然实现了编译成功，但最终加载新内核时却无法进入 Ubuntu 系统界面，而是进入 BusyBox 命令行界面，这是因为内核缺乏 Ubuntu 相关配置。

3. 证书签名问题：通过查阅编译日志以及网上寻找资料，我发现使用 Ubuntu 时涉及到证书签名的问题，如图 23。经过重新设置后，最终编译成功且正确启动。

4. 编译指令多次修改内核编译记得 make clean，虽然重新编译时间更长了，但是可以避免耗费大量硬盘空间，手动增加虚拟硬盘空间。

```
make[3]: *** No rule to make target 'debian/canonical-certs.pem', needed by 'certs/x509_certificate_list'. Stop.
make[2]: *** [scripts/Makefile.build:485: certs] Error 2
make[1]: *** [/opt/linux/Makefile:1919: .] Error 2
make[1]: *** Waiting for unfinished jobs...
make: *** [Makefile:240: __sub-make] Error 2
```

图 23: 证书签名问题

编译参考资料:

- <https://linux.cn/article-16252-1.html>
- <https://www.cnblogs.com/smlile-you-me/p/18248433#>

3.2 内核修改开发

由于 Linux 版本持续更新，因此在进行内核的修改和开发时，参考的资料也必须注意版本的匹配。尤其是使用较新的内核版本，有很多资料上的函数已经过时了，需要多多查阅官方文档或者关注内核更新日志了解到新版本的函数/API 的使用方法。很多时候编译过程中出现”xxx function is obsolete”的错误信息。

3.3 体会

在实验过程中，我学习到了如何自己给 Linux 内核添加功能。修改内核代码然后编译成功并且完成实验令我兴奋。同时，我也体会到 Linux 内核的复杂性，在修改代码时需要考虑到很多细节，如果不注意就容易白白浪费时间在编译上。

在完成本次实验后，我算是初步认识到了 Linux 内核的开发的流程，也对 Linux 系统的工作原理有了更深入的了解。