

Obadah Abozaid

ID: 5351127

Exploring Kernel Virtual Machine (KVM) and System Call Modifications

Introduction

In this report I will explore a Linux Kernel Virtual Machine (KVM) and how it can be used to simulate modifications made directly to the kernel of an operating system - from the initial boot to a fully functional system. I will also add two of my own system calls to demonstrate the functionality of making modifications to a system kernel, and to implement my own function using native headers.

Understanding System Calls

System calls provide the interface between a running program and the operating system. They are a set of functions that are provided by the operating system's kernel, which applications running in user mode can call. When a system call is invoked, the execution context switches from user mode to kernel mode, allowing the program to access protected system resources.

Examples of system calls include `read()`, `write()`, `open()`, `close()`, `wait()`, `exec()`, `fork()`, `exit()`, among others. These calls allow a program to interact with the system, such as reading from or writing to files, creating new processes, and communicating over a network.

System calls are crucial for several reasons:

Resource Management: System calls allow programs to request resources from the operating system. From requesting memory to creating a new process, the operating system can manage these resources, ensuring that all programs get executed effectively.

Abstraction: System calls provide an abstraction layer between the hardware and the application software, implying a developer does not need to know the specifics of how certain hardware works to interact with it.

Security and Stability: By requiring programs to use system calls to access system resources, the operating system can ensure that programs do not accidentally or maliciously interfere with each other. This helps maintain the stability of the system and protects it from potential security threats.

System calls are a fundamental part of any operating system. They provide an interface for user programs to request services from the operating system's kernel, which manages all the machine's hardware resources.

Modifying System Calls

In this section, I will describe the specific system calls that were modified and the reasons for choosing them. I will also explain testing and implementation.

First System Call: ObadahAbozaid5351127

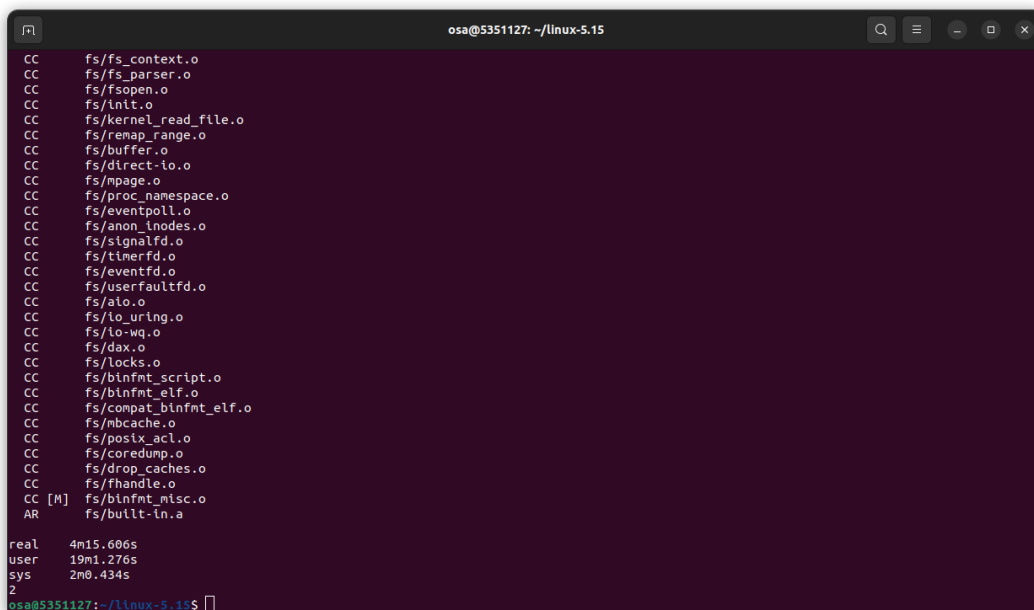
This system call takes no parameters and returns 0 on success. The purpose of this system call is to print a message to the kernel log, containing my name and student ID number. This system call is a simple example of how to create a custom system call and how to use the printk function to write to the kernel log.

To create this system call:

I created a new directory ObadahAbozaid5351127 under the kernel source tree root, and a new file ObadahAbozaid5351127.c in that directory.

I modified the provided code for my system call in ObadahAbozaid5351127.c, using the SYSCALL_DEFINE0 macro and the printk function.

I had initially created a Makefile in the same directory, listing the .c file as an object file (.o).



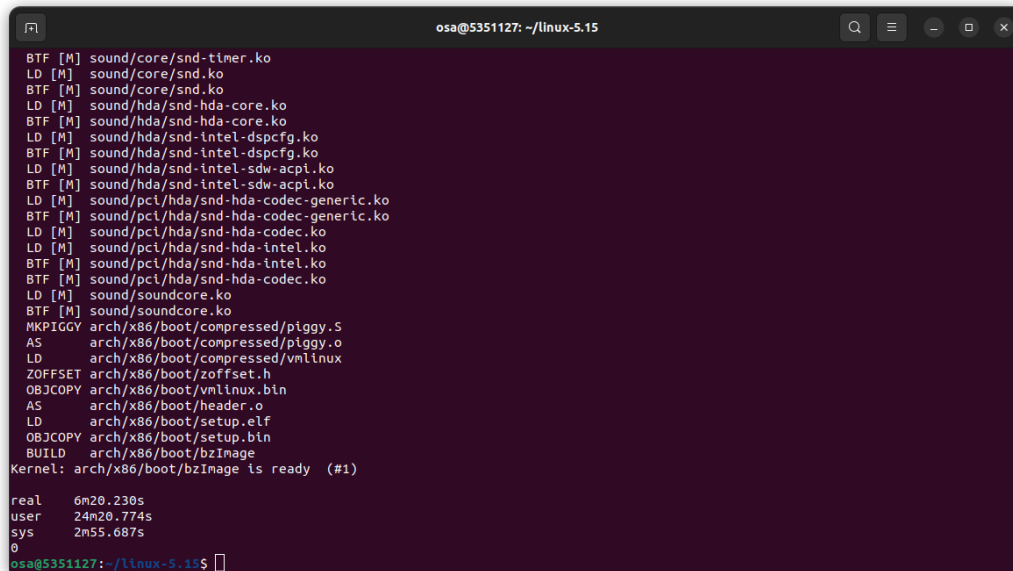
```
osa@5351127: ~/linux-5.15
CC fs/fs_context.o
CC fs/fs_parser.o
CC fs/fsopen.o
CC fs/lnt.o
CC fs/kernel_read_file.o
CC fs/remap_range.o
CC fs/buffer.o
CC fs/direct-io.o
CC fs/mpage.o
CC fs/proc_namespace.o
CC fs/eventpoll.o
CC fs/anon_inodes.o
CC fs/signalfd.o
CC fs/timerfd.o
CC fs/eventfd.o
CC fs/userfaultfd.o
CC fs/ato.o
CC fs/to_uring.o
CC fs/to_wq.o
CC fs/dax.o
CC fs/locks.o
CC fs/binfmt_script.o
CC fs/binfmt_elf.o
CC fs/compat_binfmt_elf.o
CC fs/mbcache.o
CC fs/posix_acl.o
CC fs/coredump.o
CC fs/drop_caches.o
CC fs/rhandle.o
CC [M] fs/binfmt_misc.o
AR fs/built-in.a

real 4m15.606s
user 19m1.276s
sys 2m0.434s
2
osa@5351127: ~/linux-5.15
```

Kernel failed to compile

Shown above by the error code 2, the kernel failed to compile initially, as I later found that I misspelled Makefile as MakeFile. This was one of a few mistakes made along the way.

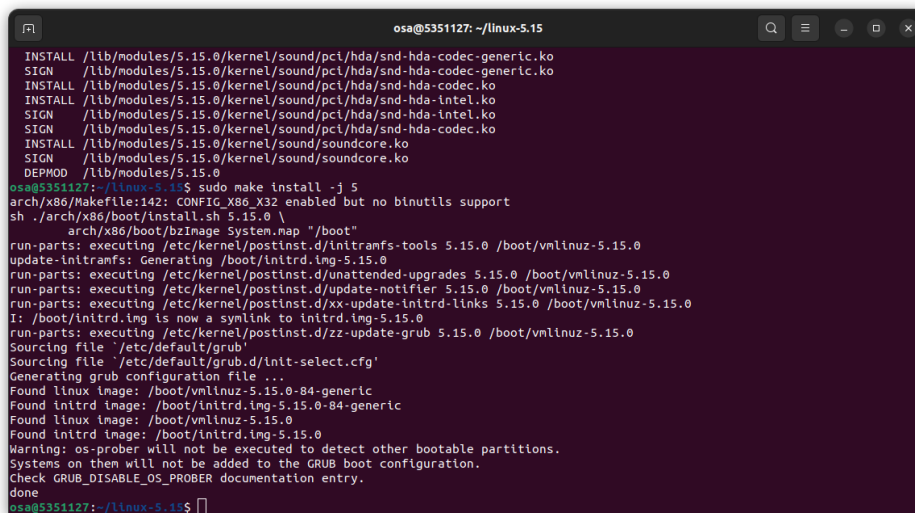
I added my directory to the kernel's Makefile, using the core-y += line, and declared my system call in the syscalls.h file, using the asmlinkage keyword and the long return type. I assigned a unique number, 449, to my system call in the syscall_64.tbl file, using the common qualifier and the sys_ prefix. I then successfully compiled and installed the new kernel, and then rebooted the system.



```
osa@5351127: ~/linux-5.15
BTF [M] sound/core/snd-timer.ko
LD [M] sound/core/snd.ko
BTF [M] sound/core/snd.ko
LD [M] sound/hda/snd-hda-core.ko
BTF [M] sound/hda/snd-hda-core.ko
LD [M] sound/hda/snd-intel-dspcfg.ko
BTF [M] sound/hda/snd-intel-dspcfg.ko
LD [M] sound/hda/snd-intel-sdw-acpi.ko
BTF [M] sound/hda/snd-intel-sdw-acpi.ko
LD [M] sound/pct/hda/snd-hda-codec-generic.ko
BTF [M] sound/pct/hda/snd-hda-codec-generic.ko
LD [M] sound/pct/hda/snd-hda-codec.ko
LD [M] sound/pct/hda/snd-hda-intel.ko
BTF [M] sound/pct/hda/snd-hda-intel.ko
BTF [M] sound/pct/hda/snd-hda-codec.ko
LD [M] sound/soundcore.ko
BTF [M] sound/soundcore.ko
MKPIGGY arch/x86/boot/compressed/piggy.S
AS arch/x86/boot/compressed/piggy.o
LD arch/x86/boot/compressed/vmlinux
ZOFFSET arch/x86/boot/zoffset.h
OBJCOPY arch/x86/boot/vmlinux.bin
AS arch/x86/boot/header.o
LD arch/x86/boot/setup.elf
OBJCOPY arch/x86/boot/setup.bin
BUILD arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready (#1)

real 6m20.230s
user 24m20.774s
sys 2m55.687s
0
osa@5351127: ~/linux-5.15
```

Kernel compiled successfully



```
osa@5351127: ~/linux-5.15
INSTALL /lib/modules/5.15.0/kernel/sound/pct/hda/snd-hda-codec-generic.ko
SIGN /lib/modules/5.15.0/kernel/sound/pct/hda/snd-hda-codec-generic.ko
INSTALL /lib/modules/5.15.0/kernel/sound/pct/hda/snd-hda-codec.ko
INSTALL /lib/modules/5.15.0/kernel/sound/pct/hda/snd-hda-intel.ko
SIGN /lib/modules/5.15.0/kernel/sound/pct/hda/snd-hda-intel.ko
SIGN /lib/modules/5.15.0/kernel/sound/pct/hda/snd-hda-codec.ko
INSTALL /lib/modules/5.15.0/kernel/sound/soundcore.ko
SIGN /lib/modules/5.15.0/kernel/sound/soundcore.ko
DEPMOD /lib/modules/5.15.0
osa@5351127: ~/linux-5.15$ sudo make install -j 5
arch/x86/Makefile:142: CONFIG_X86_X32 enabled but no binutils support
sh ./arch/x86/boot/install.sh 5.15.0 \
  arch/x86/boot/bzImage system.map "/boot"
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 5.15.0 /boot/vmlinuz-5.15.0
update-initramfs: Generating /boot/initrd.img-5.15.0
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 5.15.0 /boot/vmlinuz-5.15.0
run-parts: executing /etc/kernel/postinst.d/update-notifier 5.15.0 /boot/vmlinuz-5.15.0
run-parts: executing /etc/kernel/postinst.d/xx-update-initrd-links 5.15.0 /boot/vmlinuz-5.15.0
I: /boot/initrd.img is now a symlink to initrd.img-5.15.0
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 5.15.0 /boot/vmlinuz-5.15.0
Sourcing file '/etc/default/grub'
Sourcing file '/etc/default/grub.d/init-select.cfg'
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-5.15.0-84-generic
Found initrd image: /boot/initrd.img-5.15.0-84-generic
Found linux image: /boot/vmlinuz-5.15.0
Found initrd image: /boot/initrd.img-5.15.0
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.
done
osa@5351127: ~/linux-5.15
```

Kernel installed successfully

To test the system call, I modified the provided user program that calls the system call using the syscall function. I used the `__NR_ObadahAbozaid5351127` constant to refer to my system call number. I compiled and ran the program and checked the kernel log using the `dmesg` command. I verified that the message was printed correctly.

```
osa@5351127: ~  
osa@5351127: ~/linux-5.15$ sudo reboot  
Connection to 192.168.122.13 closed by remote host.  
Connection to 192.168.122.13 closed.  
obwan@obwan-ubuntu: $ ssh osa@192.168.122.13  
osa@192.168.122.13's password:  
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.0 x86_64)  
  
* Documentation:  https://help.ubuntu.com  
* Management:    https://landscape.canonical.com  
* Support:       https://ubuntu.com/advantage  
  
System Information as of Thu Sep 21 02:41:23 AM UTC 2023  
  
System load:  0.6337890625      Processes:           163  
Usage of /:   72.8% of 11.21GB   Users logged in:    0  
Memory usage: 4%                IPv4 address for enp1s0: 192.168.122.13  
Swap usage:   0%  
  
Expanded Security Maintenance for Applications is not enabled.  
  
1 update can be applied immediately.  
To see these additional updates run: apt list --upgradable  
  
Enable ESM Apps to receive additional future security updates.  
See https://ubuntu.com/esm or run: sudo pro status  
  
Last login: Thu Sep 21 01:39:44 2023 from 192.168.122.1  
osa@5351127: $ uname -r  
5.15.0  
osa@5351127: $
```

'uname -r' output shows newly modified kernel

```
osa@5351127: ~  
[ 5.324878] audit: type=1400 audit(1695264065.032:13): apparmor="STATUS" operation="profile_load" profile="unconfine  
d" name="/usr/lib/snapd/snap-confine" pid=646 comm="apparmor_parser"  
[ 5.325631] audit: type=1400 audit(1695264065.032:14): apparmor="STATUS" operation="profile_load" profile="unconfine  
d" name="/usr/lib/snapd/snap-confine//mount-namespace-capture-helper" pid=646 comm="apparmor_parser"  
[ 10.990609] loop3: detected capacity change from 0 to 8  
[ 11.610491] kauditd_printk_skb: 16 callbacks suppressed  
[ 11.610496] audit: type=1400 audit(1695264071.316:31): apparmor="STATUS" operation="profile_replace" profile="unconf  
ined" name="/snap/snapd/19457/usr/lib/snapd/snap-confine" pid=927 comm="apparmor_parser"  
[ 11.632328] audit: type=1400 audit(1695264071.340:32): apparmor="STATUS" operation="profile_replace" profile="unconf  
ined" name="/snap/snapd/19457/usr/lib/snapd/snap-confine//mount-namespace-capture-helper" pid=927 comm="apparmor_parser"  
[ 11.656796] audit: type=1400 audit(1695264071.364:33): apparmor="STATUS" operation="profile_replace" info="same as c  
urrent profile, skipping" profile="unconfined" name="snap.update-ns.lxd" pid=929 comm="apparmor_parser"  
[ 11.666080] audit: type=1400 audit(1695264071.372:34): apparmor="STATUS" operation="profile_replace" info="same as c  
urrent profile, skipping" profile="unconfined" name="snap.lxd.activate" pid=930 comm="apparmor_parser"  
[ 11.666089] audit: type=1400 audit(1695264071.372:35): apparmor="STATUS" operation="profile_replace" info="same as c  
urrent profile, skipping" profile="unconfined" name="snap.lxd.benchmark" pid=931 comm="apparmor_parser"  
[ 11.670753] audit: type=1400 audit(1695264071.376:36): apparmor="STATUS" operation="profile_replace" info="same as c  
urrent profile, skipping" profile="unconfined" name="snap.lxd.buginfo" pid=932 comm="apparmor_parser"  
[ 11.679753] audit: type=1400 audit(1695264071.384:37): apparmor="STATUS" operation="profile_replace" info="same as c  
urrent profile, skipping" profile="unconfined" name="snap.lxd.check-kernel" pid=933 comm="apparmor_parser"  
[ 11.681660] audit: type=1400 audit(1695264071.388:38): apparmor="STATUS" operation="profile_replace" info="same as c  
urrent profile, skipping" profile="unconfined" name="snap.lxd.daemon" pid=934 comm="apparmor_parser"  
[ 11.685285] audit: type=1400 audit(1695264071.392:39): apparmor="STATUS" operation="profile_replace" info="same as c  
urrent profile, skipping" profile="unconfined" name="snap.lxd.hook.configure" pid=935 comm="apparmor_parser"  
[ 11.689571] audit: type=1400 audit(1695264071.396:40): apparmor="STATUS" operation="profile_replace" info="same as c  
urrent profile, skipping" profile="unconfined" name="snap.lxd.hook.install" pid=936 comm="apparmor_parser"  
[ 287.230697] I am Abozaid Abozaid with ID number 5351127.  
osa@5351127: $
```

System call shown in log with 'dmesg'

```
osa@5351127: ~
1 update can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Last login: Thu Sep 21 01:39:44 2023 from 192.168.122.1
osa@5351127: ~$ uname -r
5.15.0
osa@5351127: ~$ ls
linux-5.15  linux-5.15.tar.xz
osa@5351127: ~$ nano call.c
osa@5351127: ~$ ls
call.c  linux-5.15  linux-5.15.tar.xz
osa@5351127: ~$ gcc -o call call.c
osa@5351127: ~$ ls
call  call.c  linux-5.15  linux-5.15.tar.xz
osa@5351127: ~$ ./call
Your system call is functional. Run the command dmesg in the terminal and find out!
osa@5351127: ~$ sudo dmesg
[sudo] password for osa:
[    0.000000] Linux version 5.15.0 (osa@5351127) (gcc (Ubuntu 11.4.0-1ubuntu1-22.04) 11.4.0, GNU ld (GNU Binutils for
Ubuntu) 2.38) #1 SMP Thu Sep 21 01:59:00 UTC 2023
[    0.000000] Command line: BOOT_IMAGE=/vmlinuz-5.15.0 root=/dev/mapper/ubuntu--vg-ubuntu--lv ro
[    0.000000] KERNEL supported cpus:
[    0.000000] Intel GenuineIntel
[    0.000000] AMD AuthenticAMD
[    0.000000] Hygon HygonGenuine
```

call program functionality

call.c:

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

```
#include <sys/syscall.h>
```

```
#include <stdio.h>
```

```
#include <unistd.h>
```

```
#include <string.h>
```

```
#include <errno.h>
```

```
#define __NR_ObadahAbozaid5351127 449
```

```
long ObadahAbozaid5351127_syscall(void)
```

```
{
```

```
    return syscall(__NR_ObadahAbozaid5351127);
```

```
}
```

```
int main(int argc, char *argv[])
```

```
{
```

```
    long activity;
```

```

activity = ObadahAbozaid5351127_syscall();

if(activity < 0)
{
    perror("Your system call appears to have failed");
}

else
{
    printf("Your system call is functional. Run the command dmesg in the terminal and find
out!\n");
}

return 0;
}

```

Second System Call: getprocessdetails

The second system call created is called `getprocessdetails`. This system call takes a pointer to a struct as a parameter and returns 0 on success or -1 on error. The purpose of this system call is to get and return some information about the calling process, such as its process id, static priority, and name. This information is stored in a struct that is defined in both user space and kernel space. This system call is a more advanced example of how to create a custom system call and how to use the current macro and the `copy_to_user` function to access and transfer process information.

I first defined a struct called `process_info` that can hold the process id, static priority, and name of the calling process. I used types such as `pid_t` and `char[16]` for these fields. Then I declared this struct in a new header file called `process_info.h`, which I placed in the `include/linux/` directory. I also included the necessary header files for the types used in the struct, such as `<linux/types.h>`.

I included this header file in both the kernel code and the user program, so that they can share the same struct definition. I created a new directory `getprocessdetails` under the kernel source tree root, and a new file `getprocessdetails.c` in that directory.

getprocessdetails.c:

```

#include <linux/process_info.h>

#include <linux/kernel.h>

```

```

#include <linux/syscalls.h>
#include <linux/sched.h>
#include <linux/uaccess.h>

SYSCALL_DEFINE1(getprocessdetails, struct process_info *, info)
{
    struct process_info kernel_info; // temporary struct in kernel space

    // get the current task's pid, static_prio, and comm
    kernel_info.pid = task_pid_nr(current);
    kernel_info.static_prio = current->static_prio;
    strncpy(kernel_info.comm, current->comm, 16);

    // copy the kernel_info to the user space pointer info
    if (copy_to_user(info, &kernel_info, sizeof(struct process_info))) {
        return -EFAULT; // return an error if copy fails
    }

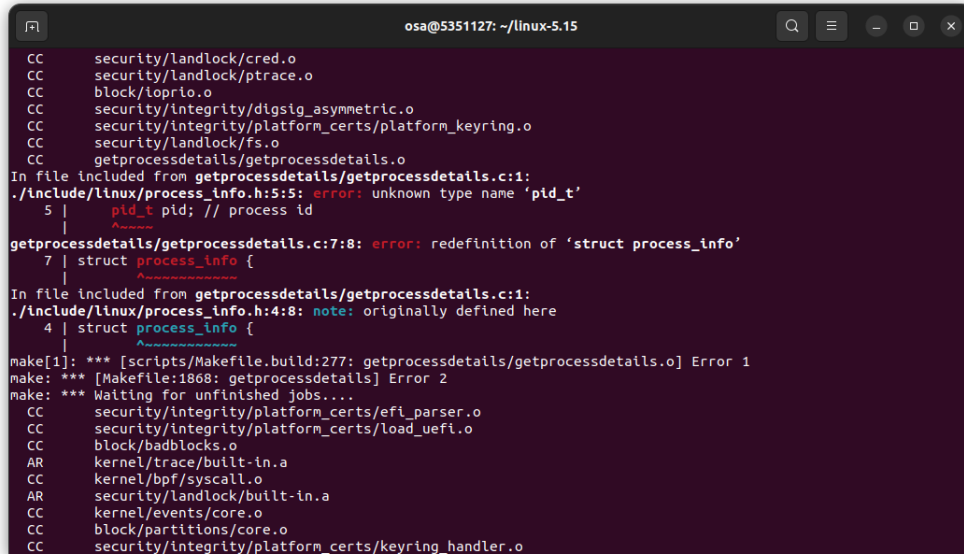
    // print some information to the kernel log
    printk("The process %s with pid %d has static priority %d\n", kernel_info.comm,
kernel_info.pid, kernel_info.static_pr>

    return 0; // return success
}

```

I wrote the code for the system call in `getprocessdetails.c`, using the `SYSCALL_DEFINE1` macro and the `copy_to_user` function. I also used the `current` macro to access the `task_struct` of the calling process, and retrieved its `pid`, `static_prio`, and `comm` fields. I stored these values in a temporary struct in kernel space, and then copied it to the user space pointer passed as a parameter. I also printed some information to the kernel log using the `printk` function.

Initially however, the kernel did not compile correctly when implementing the syscall, because the type `pid_t` wasn't recognized in the `process_info.h` file. This type is defined in the `linux/types.h` header file, so I needed to include this header.

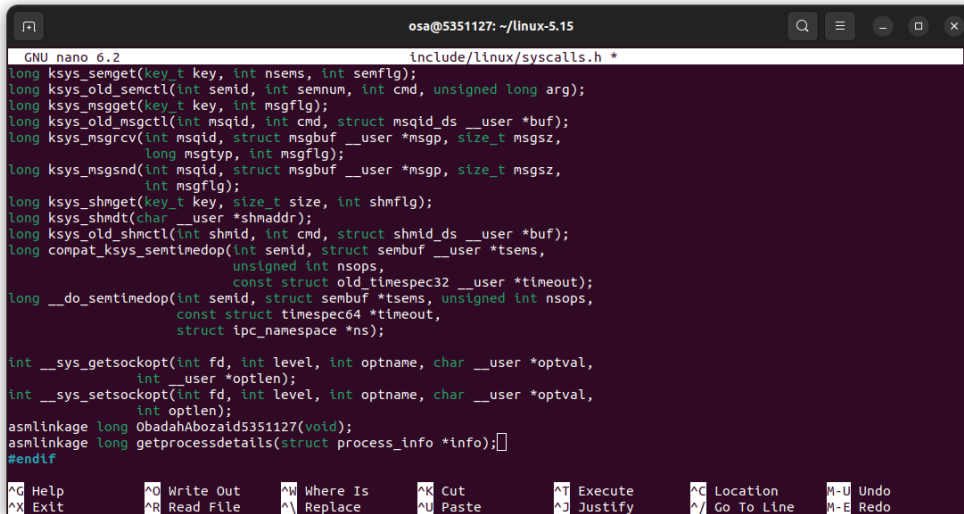


```
osa@5351127: ~/linux-5.15
CC security/landlock/cred.o
CC security/landlock/ptrace.o
CC block/ioprio.o
CC security/integrity/digsig_asymmetric.o
CC security/integrity/platform_certs/platform_keyring.o
CC security/landlock/fs.o
CC getproccsdetails/getproccsdetails.o
In file included from getproccsdetails/getproccsdetails.c:1:
./include/linux/process_info.h:5:5: error: unknown type name 'pid_t'
  5 |     pid_t pid; // process id
    |     ^~~~~
getproccsdetails/getproccsdetails.c:7:8: error: redefinition of 'struct process_info'
  7 |     struct process_info {
    |     ^~~~~
In file included from getproccsdetails/getproccsdetails.c:1:
./include/linux/process_info.h:4:8: note: originally defined here
  4 |     struct process_info {
    |     ^~~~~
make[1]: *** [scripts/Makefile.build:277: getproccsdetails/getproccsdetails.o] Error 1
make: *** [Makefile:1868: getproccsdetails] Error 2
make: *** Waiting for unfinished jobs....
CC security/integrity/platform_certs/efi_parser.o
CC security/integrity/platform_certs/load_uefi.o
CC block/badblocks.o
AR kernel/trace/built-in.a
CC kernel/bpf/syscall.o
AR security/landlock/built-in.a
CC kernel/events/core.o
CC block/partitions/core.o
CC security/integrity/platform_certs/keyring_handler.o
```

process_info.h error

I created a Makefile in the same directory, listing the `.c` file as an object file (`.o`), and added the directory to the kernel's Makefile, using the `core-y +=` line.

I declared the system call in the `syscalls.h` file, using the `asmlinkage` keyword and the long return type, including the struct as a parameter type. I then assigned a unique number, 450, to the system call in the `syscall_64.tbl` file, using the common qualifier and the `sys_` prefix.



```
osa@5351127: ~/linux-5.15
GNU nano 6.2 include/linux/syscalls.h *
long ksys_senget(key_t key, int nsens, int semflg);
long ksys_old_semctl(int semid, int sennum, int cmd, unsigned long arg);
long ksys_msgget(key_t key, int msgflg);
long ksys_old_msgctl(int msqid, int cmd, struct msqid_ds __user *buf);
long ksys_msgrcv(int msqid, struct msgbuf __user *msgp, size_t msgsz,
                long msgtyp, int msgflg);
long ksys_msgsnd(int msqid, struct msgbuf __user *msgp, size_t msgsz,
                int msgflg);
long ksys_shmget(key_t key, size_t size, int shmflg);
long ksys_shmctl(char __user *shmaddr);
long ksys_old_shmctl(int shmid, int cmd, struct shmid_ds __user *buf);
long compat_ksys_semtimedop(int semid, struct sembuf __user *tsens,
                            unsigned int nsops,
                            const struct old_timespec32 __user *timeout);
long __do_semtimedop(int semid, struct sembuf *tsens, unsigned int nsops,
                    const struct timespec64 *timeout,
                    struct ipc_namespace *ns);

int __sys_getsockopt(int fd, int level, int optname, char __user *optval,
                    int __user *optlen);
int __sys_setsockopt(int fd, int level, int optname, char __user *optval,
                    int optlen);
asmlinkage long ObadahAbozaid5351127(void);
asmlinkage long getproccsdetails(struct process_info *info);
#endif
^C
^C Help      ^O Write Out  ^W Where Is  ^R Cut       ^T Execute   ^C Location  ^U Undo
^X Exit      ^B Read File  ^M Replace   ^U Paste     ^J Justify   ^G Go To Line ^R Redo
```

Adding to syscalls.h

```
osa@5351127: ~/linux-5.15
GNU nano 6.2 arch/x86/entry/syscalls/syscall_64.tbl *
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 common quotactl_fd sys_quotactl_fd
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
447 common memfd_secret sys_memfd_secret
448 common process_mrelease sys_process_mrelease
449 common ObadahAbozaid5351127 sys_ObadahAbozaid5351127
450 common getproccsdetails sys_getproccsdetails

# 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.
#
512 x32 rt_sigaction compat_sys_rt_sigaction
513 x32 rt_sigreturn compat_sys_x32_rt_sigreturn
514 x32 ioctl compat_sys_ioctl
515 x32 readv sys_readv
516 x32 writev sys_writev

^O Help ^O Write Out ^O Where Is ^O Cut ^O Execute ^O Location ^O Undo
^X Exit ^R Read File ^N Replace ^U Paste ^J Justify ^G Go To Line ^-B Redo
```

Adding to syscalls_64.tbl

I successfully compiled, installed the new kernel, and rebooted the system.

```
osa@5351127: ~/linux-5.15
BTF [M] arch/x86/kvm/kvm.ko
BTF [M] drivers/md/raid10.ko
BTF [M] drivers/gpu/drm/drm.ko
LD [M] drivers/md/raid456.ko
BTF [M] drivers/md/raid456.ko
LD [M] fs/btrfs/btrfs.ko
LD [M] sound/hda/snd-hda-core.ko
BTF [M] sound/hda/snd-hda-core.ko
BTF [M] fs/btrfs/btrfs.ko
LD [M] sound/pci/hda/snd-hda-codec.ko
BTF [M] sound/pci/hda/snd-hda-codec.ko
LD [M] sound/pci/hda/snd-hda-intel.ko
BTF [M] sound/pci/hda/snd-hda-intel.ko
MKPIGGY arch/x86/boot/compressed/piggy.S
AS arch/x86/boot/compressed/piggy.o
LD arch/x86/boot/compressed/vmlinux
ZOFFSET arch/x86/boot/zoffset.h
OBJCOPY arch/x86/boot/vmlinux.bin
AS arch/x86/boot/header.o
LD arch/x86/boot/setup.elf
OBJCOPY arch/x86/boot/setup.bin
BUILD arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready (#2)

real 7m19.627s
user 28m23.498s
sys 2m54.518s
0
osa@5351127:~/linux-5.15$
```

Successful kernel compilation

```
osa@5351127: ~/linux-5.15
arch/x86/boot/bzImage System.map "/boot"
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 5.15.0 /boot/vmlinuz-5.15.0
update-initramfs: Generating /boot/initrd.img-5.15.0
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 5.15.0 /boot/vmlinuz-5.15.0
run-parts: executing /etc/kernel/postinst.d/update-notifier 5.15.0 /boot/vmlinuz-5.15.0
run-parts: executing /etc/kernel/postinst.d/xx-update-initrd-links 5.15.0 /boot/vmlinuz-5.15.0
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 5.15.0 /boot/vmlinuz-5.15.0
Sourcing file '/etc/default/grub'
Sourcing file '/etc/default/grub.d/init-select.cfg'
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-5.15.0-84-generic
Found initrd image: /boot/initrd.img-5.15.0-84-generic
Found linux image: /boot/vmlinuz-5.15.0
Found initrd image: /boot/initrd.img-5.15.0
Found linux image: /boot/vmlinuz-5.15.0.old
Found initrd image: /boot/initrd.img-5.15.0
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.
done
osa@5351127:~/linux-5.15$ awk -F\ ' '/menuentry / {print $1,$2}' /boot/grub/grub.cfg
menuentry Ubuntu
  menuentry Ubuntu, with Linux 5.15.0-84-generic
  menuentry Ubuntu, with Linux 5.15.0-84-generic (recovery mode)
  menuentry Ubuntu, with Linux 5.15.0
  menuentry Ubuntu, with Linux 5.15.0 (recovery mode)
  menuentry Ubuntu, with Linux 5.15.0.old
  menuentry Ubuntu, with Linux 5.15.0.old (recovery mode)
osa@5351127:~/linux-5.15$
```

Installation of new kernel

To test the system call, I wrote a user program that calls the `getprocessdetails` system call using the `syscall` function. I used the same struct definition as in the kernel code and passed its address as a parameter. I also used an error-checking mechanism to handle any failures. I compiled and ran the program and printed the process details returned by the system call. I also checked the kernel log using the `dmesg` command, verifying that my custom system call worked correctly and returned identical values.

test_getprocessdetails.c:

```
#include <stdio.h>
```

```
#include <sys/syscall.h>
```

```
#include <unistd.h>
```

```
#define SYS_getprocessdetails 450
```

```
struct process_info {
```

```
    pid_t pid; // process id
```

```
    int static_prio; // static priority
```

```
    char comm[16]; // process name
```

```
};
```



```
osa@5351127: ~  
[ 321.147878] audit: type=1400 audit(1695271430.273:39): apparmor="STATUS" operation="profile_repla  
ce" info="same as current profile, skipping" profile="unconfined" name="snap.lxd.hook.configure" pid=  
=1199 comm="apparmor_parser"  
[ 321.150748] audit: type=1400 audit(1695271430.273:40): apparmor="STATUS" operation="profile_repla  
ce" info="same as current profile, skipping" profile="unconfined" name="snap.lxd.hook.install" pid=1  
200 comm="apparmor_parser"  
[ 322.895881] loop4: detected capacity change from 0 to 8  
[ 329.611270] loop4: detected capacity change from 0 to 129976  
osa@5351127:~$ ls  
call call.c linux-5.15 linux-5.15.tar.xz test_getprocessdetails test_getprocessdetails.c  
osa@5351127:~$ ./test_getprocessdetails  
Process ID: 1505  
Static Priority: 120  
Process Name: test_getprocess  
osa@5351127:~$ ./test_getprocessdetails & echo $!  
[1] 1509  
1509  
osa@5351127:~$ Process ID: 1509  
Static Priority: 120  
Process Name: test_getprocess  
[1]+ Done ./test_getprocessdetails  
osa@5351127:~$
```

Final matching process IDs. Success.

Tracing the System Calls

The `strace` command is a powerful command-line tool used in Linux for diagnostic, debugging, and instructional purposes. It intercepts and records the system calls which are called by a process and the signals which are received by a process. The `strace` command can be invoked by prefixing a command with `strace`, and it will print a list of system calls made by that program.

To trace the system calls made by `test_getprocessdetails`, I first compiled the test program with `gcc`:

```
gcc -o test_getprocessdetails test_getprocessdetails.c
```

I ran my program with `strace` and redirected the output to a file:


```
Activities Terminal
osa@5351127:~$ ls
call call.c linux-5.15 linux-5.15.tar.xz results.txt test_getprocessdetails test_getprocessdetails.c
osa@5351127:~$ nano results.txt
osa@5351127:~$ grep getprocessdetails results.txt
execve("./test_getprocessdetails", ["/test_getprocessdetails"], 0x7ffc6bf55d10 /* 13 vars */) = 0
osa@5351127:~$ awk '{print $1}' results.txt | sort | uniq
+++
access("/etc/ld.so.preload",
arch_prctl(0x3001
arch_prctl(ARCH_SET_FS,
brk(0x5607d0985000)
brk(NULL)
close(3)
execve("./test_getprocessdetails",
exit_group(0)
getrandom("\xe4\x79\xd7\x24\xc0\xa",
mmap(0x7f81bf7f5000,
mmap(0x7f81bf98a000,
mmap(0x7f81bf9e2000,
mmap(0x7f81bf9e8000,
mmap(NULL,
mprotect(0x5607d01ee000,
mprotect(0x7f81bf9e2000,
mprotect(0x7f81bfa35000,
munmap(0x7f81bf9f5000,
newfstatat(1,
newfstatat(3,
openat(AT_FDCWD,
pread64(3,
prlimit64(0,
read(3,
rseq(0x7f81bf7cb0e0,
set_robust_llst(0x7f81bf7caa20,
set_tid_address(0x7f81bf7caa10)
syscall_0x1c2(0x7ffd93307c80,
write(1,
osa@5351127:~$ █
```

'awk '{print \$1}' output.txt | sort | uniq'

This command extracted the first word (the system call name) from each line in the results.txt file, sorted them, and removed any duplicates.

By using strace, I was able to see exactly what system calls my program was making, including the custom system call created. This helped to verify that the system call was working correctly and understand how a program interacts with the operating system.

Conclusions

To summarize, I explored the KVM and system call modifications in an Ubuntu Linux Server 22.04 LTS environment. I installed the KVM and created a virtual machine for the guest OS with no difficulty. I learned about the concept and importance of system calls in an operating system by creating two custom system calls: ObadahAbozaid5351127 and getprocessdetails. This was implemented and tested using C code and the strace tool.

I learned that system calls provide a way for user programs to request services from the operating system's kernel, which manages all of the hardware resources. By modifying system calls, developers can customize the behavior and performance of the system, as well as create new functionalities. However, modifying system calls also requires careful planning, testing, and debugging, as well as understanding the kernel source code and structure.