Assignment 2

Advanced Operating Systems (CSC0016)

National Taiwan Normal University
Gábor Szolnok
2023.09.28.

1) Hardware and Software environment

The setup is the same as in the previous assignment. I use a virtualized environment, but do the compilation and coding on my host machine. The compiled code is shared with the guest machine via a shared folder. In the following description, whenever a command starts with “GUEST$” it implies that the command was initiated on the guest machine.

2) Implementation steps

For the implementation I used the instructions of 2 blog posts [1] [2]. I needed to edit the source code in multiple places. I first stepped in the folder of the source code, and then created a folder named “hello” with 2 files under it called “hello.c” and “Makefile”. The former is the source code for the system call, while the latter is a special file that helps in the compilation.

```
1. cd ~/src/ntnu/advanced_os/1assignment/guest/linux-6.6.0
2. mkdir hello
3. vim hello/hello.c
   // Create the hello.c file. The source code can be found in appendix A.
4. vim hello/Makefile
   // Create the Makefile. The source code can be found in appendix B.
```

Next up I modified the Kbuild file to include the hello folder during compilation. The original blog post suggested I should edit the kernel’s makefile, but in the 6.6.0 version the extension of the core-y value has been moved to the Kbuild file [3].

```
5. vim Kbuild
```

```
                                                   //From the 95th line of the file
obj-y                   += sound/
obj-$(CONFIG_SAMPLES)   += samples/
obj-$(CONFIG_NET)       += net/
obj-y                   += virt/
obj-y                   += $(ARCH_DRIVERS)
obj-y                   += hello/
```

Then I added the system call to the system call table. Each record represents a syscall. A record contains the system call identifier, the ABI representer, the descriptive name of the function call and the stub function's name that the build is going to generate. The unique identifier of the new function is going to be "548", since it's the next value coming in the sequence.

```
6. vim arch/x86/entry/syscalls/syscall_64.tbl
```

```
                                        //From the 416th line of the file
543     x32      io_setup        compat_sys_io_setup
544     x32      io_submit       compat_sys_io_submit
545     x32      execveat        compat_sys_execveat
546     x32      preadv2         compat_sys_preadv64v2
547     x32      pwritev2        compat_sys_pwritev64v2
548     common   hello           sys_hello
# This is the end of the legacy x32 range.  Numbers 548 and above are
# not special and are not to be used for x32-specific syscalls.
```

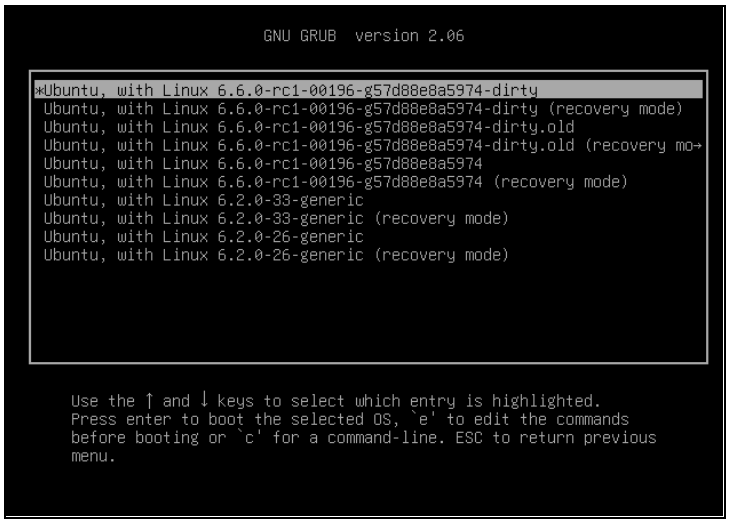
Finally I added the system call's declaration to the system header file as well. This declares the function that I implemented in the hello/hello.c file.

```
7. vim include/linux/syscalls.h
```

```
                                        //From the 1269th line of the file
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 sys_hello(void);
#endif
```

Then I compiled and installed the kernel (the same way as in the previous assignment).

```
8.  GUEST$ cd /media/sf_guest/linux-6.6.0
9.  GUEST$ make defconfig
10. make -j10
11. GUEST$ make modules_install install
```



```
GNU GRUB  version 2.06

*Ubuntu, with Linux 6.6.0-rc1-00196-g57d88e8a5974-dirty
 Ubuntu, with Linux 6.6.0-rc1-00196-g57d88e8a5974-dirty (recovery mode)
 Ubuntu, with Linux 6.6.0-rc1-00196-g57d88e8a5974-dirty.old
 Ubuntu, with Linux 6.6.0-rc1-00196-g57d88e8a5974-dirty.old (recovery mo→
 Ubuntu, with Linux 6.6.0-rc1-00196-g57d88e8a5974
 Ubuntu, with Linux 6.6.0-rc1-00196-g57d88e8a5974 (recovery mode)
 Ubuntu, with Linux 6.2.0-33-generic
 Ubuntu, with Linux 6.2.0-33-generic (recovery mode)
 Ubuntu, with Linux 6.2.0-26-generic
 Ubuntu, with Linux 6.2.0-26-generic (recovery mode)

   Use the ↑ and ↓ keys to select which entry is highlighted.
   Press enter to boot the selected OS, `e' to edit the commands
   before booting or `c' for a command-line. ESC to return previous
   menu.
```

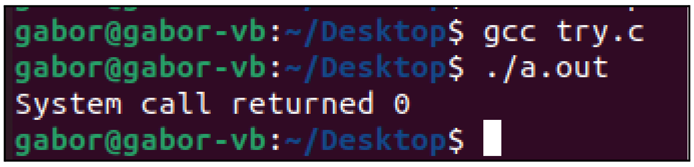
Figure 1: The kernels to choose from after compilation. The freshly modified and compiled kernel received a “dirty” tag.

3) Demonstration

After the installation I rebooted the guest machine and changed the kernel to the newly installed one. In the ~/Desktop folder I created a .c source file which would call the implemented system call. Then I compiled it and ran the compiled code. Finally, I used dmesg to check the kernel log.

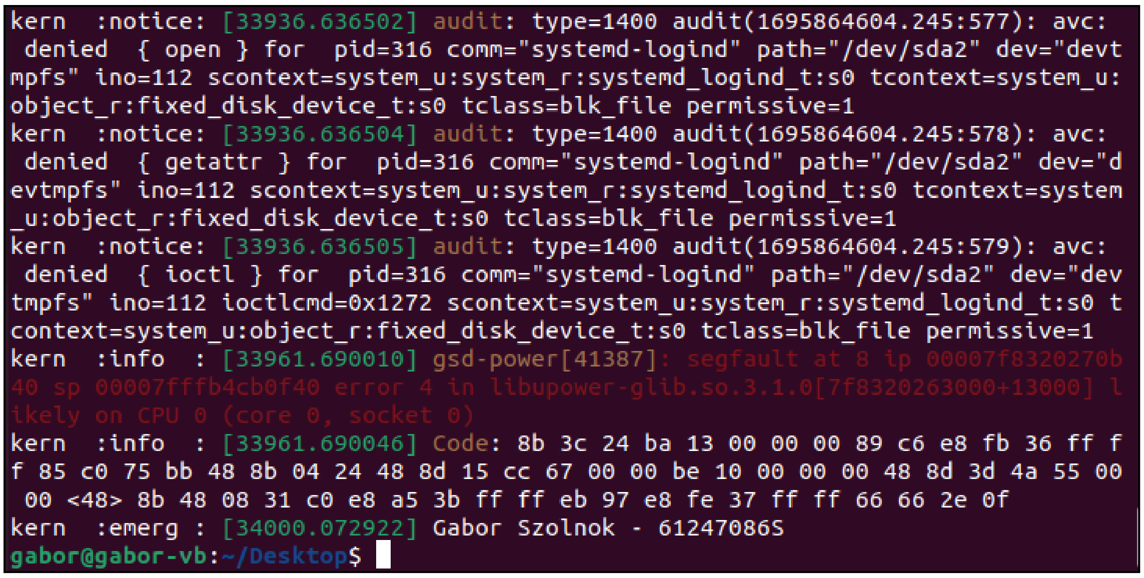
```
12. vim try.c
    // Create the file, which tries the system call. Implementation is in Appendix C.
13. gcc try.c
14. ./a.out
15. dmesg --decode
```

Figure 2 shows the line we receive on stdout after running the a.out file. It indicates that the system call returned 0, meaning that the call was successful. Figure 3 shows the kernel log, where we can find the student identifier printed by the system call on the highest priority level.



```
gabor@gabor-vb:~/Desktop$ gcc try.c
gabor@gabor-vb:~/Desktop$ ./a.out
System call returned 0
gabor@gabor-vb:~/Desktop$ 
```

Figure 2: compiling and running the demonstrating code



```
kern  :notice: [33936.636502] audit: type=1400 audit(1695864604.245:577): avc:
 denied  { open } for  pid=316 comm="systemd-logind" path="/dev/sda2" dev="devt
mpfs" ino=112 scontext=system_u:system_r:systemd_logind_t:s0 tcontext=system_u:
object_r:fixed_disk_device_t:s0 tclass=blk_file permissive=1
kern  :notice: [33936.636504] audit: type=1400 audit(1695864604.245:578): avc:
 denied  { getattr } for  pid=316 comm="systemd-logind" path="/dev/sda2" dev="d
evtmpfs" ino=112 scontext=system_u:system_r:systemd_logind_t:s0 tcontext=system
_u:object_r:fixed_disk_device_t:s0 tclass=blk_file permissive=1
kern  :notice: [33936.636505] audit: type=1400 audit(1695864604.245:579): avc:
 denied  { ioctl } for  pid=316 comm="systemd-logind" path="/dev/sda2" dev="dev
tmpfs" ino=112 ioctlcmd=0x1272 scontext=system_u:system_r:systemd_logind_t:s0 t
context=system_u:object_r:fixed_disk_device_t:s0 tclass=blk_file permissive=1
kern  :info  : [33961.690010] gsd-power[41387]: segfault at 8 ip 00007f8320270b
40 sp 00007fffb4cb0f40 error 4 in libupower-glib.so.3.1.0[7f8320263000+13000] l
ikely on CPU 0 (core 0, socket 0)
kern  :info  : [33961.690046] Code: 8b 3c 24 ba 13 00 00 00 89 c6 e8 fb 36 ff f
f 85 c0 75 bb 48 8b 04 24 48 8d 15 cc 67 00 00 be 10 00 00 00 48 8d 3d 4a 55 00
 00 <48> 8b 48 08 31 c0 e8 a5 3b ff ff eb 97 e8 fe 37 ff ff 66 66 2e 0f
kern  :emerg : [34000.072922] Gabor Szolnok - 61247086S
gabor@gabor-vb:~/Desktop$ 
```

Figure 3: the received logs after initiating the command "dmesg --decode". The dmesg command prints the recent kernel logs, the "--decode" flag helps us see the priority level of them. As it can be seen at the last line, my student identifier was printed by the system call on the highest (emergency) priority.

4) Comments

I had two major challenges within the assignment. I originally went along with one blog post [1], however when I tried to compile the code, it ran on an error (Appendix D). The error indicated that the function declaration was not right somewhere and I spent quite a lot of time trying to figure out how to solve it. I ended up finding another tutorial [2], which - as opposed to the first one,- used the "SYSCALL_DEFINE0" directive from the linux/syscalls.h library. This, and using the "common" ABI representer instead of the 64 one in the syscall_64.tbl file solved the problem.
The other issue I had is related to the 6.6.0. version. I couldn't find the "obj-y" variable in the root Makefile, but both tutorials instructed me to change it there. Since the source code is relatively new, there are not that many resources that show how to solve the issue, so I needed to dig really deep to find where it has been moved to. After a while, I bumped into a Stackoverflow suggestion [3] and it worked.

5) Appendix

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

SYSCALL_DEFINE0(hello) {
   printk(KERN_EMERG "Gabor Szolnok - 61247086S\n");
   return 0;
}
```

Appendix A: The implementation of the hello system call. Printk prints the message to the kernel log with a KERN_EMERG log level. [4]. The SYSCALL_DEFINE0 macro helps other tools use metadata of the function. [5]

```makefile
obj-y := hello.o
```

Appendix B: The Makefile created within the hello/ dir.

```c
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
int main()
{
        long int ret = syscall(548);
        printf("System call returned %ld\n", ret);
        return 0;
}
```

Appendix C: The C file of the demonstrating application that calls the newly created system call.

```
ld: vmlinux.o:(.rodata+0x1300): undefined reference to
`__x64_sys_hello'
make[2]: *** [scripts/Makefile.vmlinux:36: vmlinux] Error 1
make[1]: *** [/abspath-censured/guest/linux-6.6.0/Makefile:1165:
vmlinux] Error 2
make: *** [Makefile:234: __sub-make] Error 2
```

Appendix D: A snippet of the error stack trace I received after trying to compile the kernel using the suggested modifications of the first tutorial[1].

6) References

[1] Blog post: “Adding a Hello World System call to Linux Kernel” Last accessed: 2023.09.28. author: Anubhav Shrimal, Jul. 12. 2018; Link: “https://medium.com/anubhav-shrimal/adding-a-hello-world-system-call-to-linux-kernel-dad32875872”

[2] Blog post: “Adding a System call to the Linux Kernel (Version 5.5.0)” Last accessed: 2023.09.28 Link: “https://redirect.cs.umbc.edu/courses/undergraduate/421/spring21/docs/project0.html”

[3] Blog post: “Adding a system call to Linux kernel 6” Last accessed: 2023.09.28 Link: “https://stackoverflow.com/questions/76262123/adding-a-system-call-to-linux-kernel-6”

[4] “Kernel.org - Message logging with printk” Last accessed: 2023.09.28 Link: “https://www.kernel.org/doc/html/next/core-api/printk-basics.html”

[5] “Kernel.org - Adding a New System Call” Last accessed: 2023.09.28 Link: “https://www.kernel.org/doc/html/latest/process/adding-syscalls.html?highlight=syscall_define”