OS Bonus Project - Project 1 on Arch Linux

Daniel Volya

0.1 Install Arch

I used Vbox. The version I installed used the linux kernel version 4.13.12
`https://wiki.archlinux.org/index.php/installation_guide`
Install SSH and do passwd stuff

```
> passwd
> pacman -S openssh
```

Configure the port forwarding. Each reboot will need to run (or add to file):

```
> dhcpd
> systemctl start sshd.service
```

0.2 Obtain and Install Linux Kernel

Install build dependencies, such as precision numeric processing language:

```
pacman -S bc
```

Get which GCC and Kernel version to get:

```
cat /proc/version
```

```
> cd ~
> mkdir kernelbuild
```

Obtain linux source:

```
> wget https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.13.12.tar.xz
> tar -xvJF linux-4.13.12.tar.xz
```

Build linux by using the current kernel configuration file

```
> cd linux
> make clean && make mrproper
> zcat /proc/config.gz > ~/kernelbuild/linux-4.13.12/.config
```

Set up build system using the current config. This ideally shouldn't actually ask anything

```
> yes '' | make oldconfig
```

Build everything

```
> make
> make modules_install
```

Setup boot

```
> cp -v arch/x86_64/boot/bzImage /boot/vmlinuz-linux413
> cp /etc/mkinitcpio.d/linux.preset /etc/mkinitcpio.d/linux413.preset
```

Apply the following to file: /etc/mkinitcpio.d/linux413.preset

```
ALL_kver="/boot/vmlinuz-linux413"
default_image="/boot/initramfs-linux413.img"
fallback_image="/boot/initramfs-linux413-fallback.img"
```

Run Arch's handy script

```
mkinitcpio -p linux413
```

Since I am using grub:

```
grub-mkconfig -o /boot/grub/grub.cfg
```

0.3 Adding Project 1 into Linux Kernel

There are of course a few ways to do this.
Add syscall to table, by adding the following to: arch/x86/entry/syscalls/syscall_64.tbl

```
333 common setaccesslvl  sys_setaccesslvl
334 common getaccesslvl  sys_getaccesslvl
```

I used the SYSCALL_DEFINE macro to create the syscalls inside: kernel/sys.c
The alternative could be to add them to seperate files and include in Makefile along with prototypes

```
1
2  SYSCALL_DEFINE3(setaccesslvl, int, calling_pid, int, target_pid, int, new_access_lvl)
3  {
4          struct task_struct *calling_task = find_task_by_vpid(calling_pid);
5          struct task_struct *target_task = find_task_by_vpid(target_pid);
6
7          if (target_task == NULL || calling_task == NULL)
8          {
9                  printk("SETACCESSLVL failed => task is NULL\n");
10                 return -1;
11         }
12
13         kuid_t user = calling_task->cred->uid;
14
15         printk("SETACCESSLVL: Got user id %d for task/process", user);
16
17         if (user.val == 0)
18         {
19
20                 printk("SETACCESSLVL: Setting access to %d b/c superuser\n",
                       new_access_lvl);
21                 target_task->access_level = new_access_lvl;
22                 return target_task->access_level;
23
24         }
25         if (calling_pid == target_pid && new_access_lvl <= calling_task->access_level)
26         {
27
28                 printk("SETACCESSLVL: Setting access to %d b/c same process and lower
                       desired\n", new_access_lvl);
29                 target_task->access_level = new_access_lvl;
30                 return target_task->access_level;
31
32         }
33
34         if (target_task->access_level >= calling_task->access_level)
35         {
36                 printk("SETACCESSLVL failed => big access level\n");
37                 return -1;
38         }
39
40         if (target_task->access_level < calling_task->access_level && new_access_lvl <=
               calling_task->access_level)
41         {
42
43
44                 printk("SETACCESSLVL: Setting access to %d b/c higher task access level\n
                       ", new_access_lvl);
45                 target_task->access_level = new_access_lvl;
46                 return target_task->access_level;
47         }
```

```
48          printk("SETACCESSLVL: failed\n");
49
50          return -1;
51
52  }
53
54  SYSCALL_DEFINE1(getaccesslvl, int, target_pid)
55  {
56
57          struct task_struct *target_task = find_task_by_vpid(target_pid);
58          if (target_task == NULL) {
59                  printk("GETACCESSLVL: failed because task is NULL\n");
60                  return -1;
61          }
62          printk("GETACCESSLVL: succeded\n");
63          return target_task->access_level;
64  }
```

Build everything

```
> make
> make modules_install
```

Run Arch's handy script

```
mkinitcpio -p linux413
```

Since I am using grub:

```
grub-mkconfig -o /boot/grub/grub.cfg
```

Create Test Program:

```
1  #include <unistd.h>
2  #include <sys/syscall.h>
3  #include <stdio.h>
4
5
6  int main(int argc, char **argv)
7  {
8
9          // 333 -> setaccesslvl
10         // 334 -> getaccesslvl
11
12         long res = syscall(335, getpid());
13
14         printf("%d\n", res);
15         res = syscall(334, getpid(), getpid(), 5);
16
17         printf("%d\n", res);
18         res = syscall(335, getpid());
19
20         printf("%d\n", res);
21
22         return 0;
23 }
```

Build and run

```
[root@arch test_stuff]# cat test.c
#include <unistd.h>
#include <sys/syscall.h>
#include <stdio.h>

int main(int argc, char **argv)
{

        // 334 -> setaccesslvl
        // 335 -> getaccesslvl

        long res = syscall(335, getpid());

        printf("%d\n", res);
        res = syscall(334, getpid(), getpid(), 5);

        printf("%d\n", res);
        res = syscall(335, getpid());

        printf("%d\n", res);

        return 0;


}
[root@arch test_stuff]# cc -o test test.c
[root@arch test_stuff]# ./test
0
5
5
[root@arch test_stuff]#
```

Figure 1: Showing code, build, and run of test program

```
[21601.380902] GETACCESSLVL: succeded
[21601.382142] SETACCESSLVL: Got user id 0 for task/process
[21601.382145] SETACCESSLVL: Setting access to 5 b/c superuser
[21601.383876] GETACCESSLVL: succeded
[22777.187203] GETACCESSLVL: succeded
[22777.187633] SETACCESSLVL: Got user id 0 for task/process
[22777.187636] SETACCESSLVL: Setting access to 5 b/c superuser
[22777.187717] GETACCESSLVL: succeded
```

Figure 2: Showing the printk message for kernel debug in dmesg (classic programmer misspelling)

0.4 C Library

This part is a little tricky. For one, the default c library installed is glibc. This is one complex beast intended for literally every single architecture.
`https://unix.stackexchange.com/questions/141329/how-to-rebuild-glibc-on-arch-linux`
`https://lwn.net/Articles/655028/`
`https://sourceware.org/glibc/wiki/SyscallWrappers`

Install Arch Linux build source file management tool

```
> asp checkout glibc
> chmod -R a+w glibc
```

Make a new non-root user and then

```
> su myuser
> cd glibc/repos/core-x86_66
```

Build the library... This takes a long time.

```
> makepkg
```

After a lot of trial and error, and lots of google searches, it seems that adding C libraries to glibc is excruciating hard.

I think an easier alternative is to use a different C/POSIX standard library such as musl. However, so far this project has taken way too much time, and I should start worrying about doing the actual project 2 along with studying for exams... But this is an interesting problem to solve at some point in the near future.