Project 3: Linux System Calls

CS 311

For this project, you'll implement two Linux kernel syscalls. You'll also implement a small wrapper shared library to present a cleaner API to users of your syscalls. For each of the syscalls, you'll need to add a unique syscall number to `unistd_32.h` and a syscall table entry to `syscall_table_32.S`, just as you did in Project 1. The two syscalls themselves can be added to your `lab_syscalls.c` file from Project 1.

References

The following are all available on, or linked to from, the course web site:

1. Program Library HOWTO, Chapter 3 Shared Libraries. Recommended reading for those curious about the background and details of shared libraries in Linux.
2. `libmysyscall.h` — Syscall API declaration file. Needed by both your library source code and any user program using your library. The file can be used as-is.
3. `libmysyscall.c` — Syscall API definition file. This will be compiled into the shared library. You'll need to add the code for your two wrapper functions.

`libmysyscall` Shared Library

You can implement this as soon as you've added the two unique syscall numbers to `unistd_32.h`. See the two syscall sections below for the numbers to use. This library will allow users of your syscalls to invoke them like this:

```
myprintint(val);
```

rather than this:

```
syscall(__NR_tpk_sysc1, val);
```

Download `libmysyscall.h` and `libmysyscall.c` into a directory in your kdev account (somewhere under your linux-2.6.27.1 directory would be a good idea, so that you can put the files under revision control). First, follow the recipe below for installing `libmysyscall.h` Then, open `libmysyscall.c` for editing and add the definitions for the two wrapper functions, `myadd()` and `myxtime()`.

Here's the recipe for building and installing the header file and shared library, along with instructions for using the shared library in a user program:

```
# cd to whatever directory contains libmysyscall.c and libmysyscall.h

cd ~/<directory containing library files>
```

```
# Install libmysyscall.h.  Repeat any time changes are made to
# libmysyscall.h.

sudo cp -i libmysyscall.h /usr/include


# _All_ of the following steps have to be repeated any time changes are
# made to either libmysyscall.c or libmysyscall.h.


# Create libmysyscall.o

gcc -fPIC -g -c -Wall libmysyscall.c


# Creates libmysyscall.so.  Note _carefully_ the arguments to the "upper W
lower el" switch and that there are _no_ whitespace characters in this
argument list.

gcc -shared -Wl,-soname,libmysyscall.s0 -o libmysyscall.so libmysyscall.o


# Move libmysyscall.so to /usr/lib

sudo cp -i libmysyscall.so /usr/lib


# Make the shared library file known to the linker

sudo /sbin/ldconfig


# Check that all is fine

/sbin/ldconfig -p | grep libmysyscall

# (The linker's cache entry to your shared library file should be printed.)


# Any source program using this library needs to contain the line:

   #include <libmysyscall.h>

# and be explicitly linked with the library:

gcc -o addtest addtest.c -lmysyscall
```

You won't be able to successfully run any program using your syscalls until you've built and booted a kernel containing the syscalls. Don't forget to edit `syscall_table_32.S` to add your two syscalls to the kernel's syscall table!

Integer Addition Syscall

Implement a syscall which simply takes two integer parameters and returns their sum. Your syscall should have the following signature:

```
asmlinkage int sys_lab_sysc2(int *sum, int op1, int op2)
```

and be numbered 334. The syscall should add `op1` and `op2`, storing the computed sum in the memory location pointed to by `sum`. If `sum` points to memory to which the user process does not have write permission, or the syscall itself can't copy the computed sum back to the user process, the syscall's return value should be -1. Otherwise, the syscall's return value should be 0. Each time the syscall is called, it should log a message to the system's log file. See the section describing userland memory access functions below.

Implement two user programs which demonstrate the use of your syscall. The first program should simply allow the user to input any two integer values, use the syscall to compute the result, and display the result for the user. The second program should pass in a `NULL` pointer as the first parameter and demonstrate that the syscall returns -1.

Current Time Syscall

Implement a syscall which returns the value of the kernel's `xtime` variable to the user process. `xtime` is of type `struct timespec`, defined as:

```
struct timespec {
   time_t   tv_sec;   /* seconds */
   long  tv_nsec;     /* nanoseconds */
};
```

The signature of your syscall should be

```
asmlinkage int sys_lab_sysc3(struct timespec *ct)
```

and this syscall should be numbered 335. If the user process passes in a memory address to which it does not have write permission or the syscall can't copy the value of `xtime` to the user process, the syscall should return -1. Otherwise, it should return 0. The syscall should log each of its invocations to the system log.

`xtime` access requires multiple memory accesses. Because of this, your syscall could be interrupted at the boundary of any of these accesses when it is reading `xtime` from the kernel to its own storage. During this time, the value of `xtime` could be changed by the kernel, thus invalidating the value you have already partially read. Therefore, synchronization primitives need to be used when reading `xtime`. See below.

Write a user program that calls your syscall. Your user program should use the syscall to get the current time and then print the current time as so:

```
The current time since the beginning of the epoch is yyyy seconds
and zzzz nanoseconds.
```

You can check that your syscall is returning the correct value by calling and printing the results from `gettimeofday(2)`.

Userland Memory Access Functions

The following are only necessary for copying memory data between user space and kernel space; i.e., when a pointer is passed to a syscall. These aren't needed for pass-by-value parameters.

1. Validating read or write access to user memory block:

   ```
   access_ok(type, addr, size)
   ```

   Returns true (non-zero) is access is allowed, otherwise returns false (0). `type` should either be `VERIFY_READ` or `VERIFY_WRITE`. `addr` is a pointer to the first byte of the memory block to be tested. `size` is the size, in bytes, of the memory block to be tested. Example:

   ```
   if (!access_ok(VERIFY_READ, ptr, sizeof(double)))
      return -1;
   ```

2. Copying a memory block from user space to kernel space:

   ```
   copy_from_user(to_ptr, from_ptr, size)
   ```

   Copies `size` bytes from user space memory block beginning at `from_ptr` to kernel space memory block beginning at `to_ptr`. Returns number of bytes that could not be copied. On success, this will be 0.

3. Copying a memory block from kernel space to user space:

   ```
   copy_to_user(to_ptr, from_ptr, size)
   ```

   Copies `size` bytes from kernel space memory block beginning at `from_ptr` to user space memory block beginning at `to_ptr`. Returns number of bytes that could not be copied. On success, this will be 0.

Synchronization Code for Reading `xtime`

A sequence lock is used to synchronize readers and writers of various kernel variables, including `xtime`. `xtime_lock` is used to guard `xtime`, which can't be read in a single memory access. The following code should be used when reading `xtime`. Note that `linux/seqlock.h` must be included in your kernel program.

```
unsigned seq;
struct timespec curTime;

do
  {
    seq = read_seqbegin(&xtime_lock);
    curTime = xtime;
  } while (read_seqretry(&xtime_lock, seq));
```

Upon exit from the do/while, `curTime` will contain a coherent value of `xtime`.

Deliverables

The following files should be uploaded to the course GoucherLearn site by 5:00 pm on the due date:

1. Your kernel source file, shared library source file, and user program source files. **Do not include any binary files.**

2. A brief README file. This file should list any functionality missing from your project, and list the names of all the files that you've uploaded, along with a very brief description of each file.

3. You'll need to schedule a demonstration with me. The demonstration will need to occur sometime during the week following the project's due date. The demonstration will count as 30% of your grade and you are responsible for scheduling the demonstration with me.