Project 1: Linux System Calls

CS 325

Due Mar. 4, 2009, 75 pts.

For this project, you'll implement two syscalls. In each case, you'll need to add a unique syscall number to `unistd_32.h` and a syscall table entry to `syscall_table_32.S`. The two syscalls themselves can be added to your `lab_syscalls.c` file. You may work on this project in groups of size one or two.

Integer Addition Syscall, 35 pts.

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, 40 pts.

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.
```

Userland Memory Access Functions

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 Kernel Variables

A sequence lock is used to synchronize readers and writers of various kernel variables, including `xtime`. `xtime_lock` is used to guard `xtime`. The following code should be used when reading `xlock`. 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));
```

Project Turn-In

Email your kernel source file and your user source files to `kelliher[at]goucher.edu` by the beginning of class on the 4th. A 10% penalty will be applied for each day the project is late. Saturday is not a day. After three days I will no longer accept projects. In addition, you will need to schedule a demonstration with me. The demonstration will need to occur sometime during the week of Mar. 2nd. The demonstration will count as 50% of your grade and you are responsible for scheduling this appointment.