Adding a System Call to the Kernel

CS 311

*** Note that all relative paths below are relative to the root of your kernel source tree. ***

Continue using your working files and repo from the "Building a Kernel" exercise.

First, we add a new system call number. Edit include/asm-x86/unistd_32.h. Following the line:

```
#define __NR_inotify_init1      332
```

Add a similar line:

```
#define __NR_lab_sysc1          333
```

Save the file.

Now, we have to add an entry to the syscall table. Edit arch/x86/kernel/syscall_table_32.S. At the end of the file, following the line:

```
.long sys_inotify_init1
```

add:

```
.long sys_lab_sysc1
```

Note that what you just added is the 333rd entry in the syscall table, matching the #define you added to unistd_32.h previously. This is _not_ a coincidence.

Save the file. Now, we need to implement our syscall function, sys_lab_sysc1(). Create the file:

```
kernel/lab_syscalls.c
```

Copy ALL the #include preprocessor directives from kernel/timer.c into kernel/lab_syscalls.c.

At the end of the file, add your sycall function, sys_lab_sysc1(). (Note that this name exactly matches the entry we added to the syscall table. This is EXTREMELY important.) This function should take one int parameter and should have a return type of:

```
asmlinkage int
```

The body of this function should use printk() to log the value of the function's single parameter into the system log file. (printk() behaves similarly to printf(), writing into the system log file. Refer to the documentation for printf in Section 3 of Linux man pages.) The function should return an integer value of 0. Save the file.

Now, we need to edit kernel/Makefile to ensure that our source file gets compiled into the kernel. At the beginning of the Makefile, there is a list of kernel object files corresponding to the kernel source files to be compiled into the kernel. This list is labeled obj-y. At the end of this list, on the same line as the other object files, add an object entry corresponding to the source file you just created. (Object files have a filename extension of ".o" .)

Now, you need to write and compile a userland C program to exercise your system call. (I'd suggest creating a UserLandTests directory immediately under the root of your kernel source tree to hold these programs.) Something like the following should do the trick:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/syscall.h>

#define __NR_lab_sysc1          333

int main(int argc, char *argv[])
{
  int val;

  if (argc != 2)
    {
      printf("One integer argument expected!!!\n");
      return 1;
    }

  val = atoi(argv[1]);

  /* Call syscall 333, the syscall we added to the kernel. */

  printf("syscall returns: %d.\n", syscall(__NR_lab_sysc1, val));
```

```
    return 0;
}
```

Once your finish writing the program, compile it.

Return to the root of your kernel tree, invoke menuconfig to increment your kernel build number, and then build a new kernel:

```
make O=/home/kdev/build menuconfig

make -j 3 O=/home/kdev/build
```

If that goes well, install the new kernel and module files:

```
sudo make O=/home/kdev/build modules_install install
```

Reboot. At the GRUB screen, select the kernel you just built. Open one shell and use it to dynamically view the end of the system log file:

```
sudo tail -f /var/log/messages
```

Run your userland test program in another shell. The integer value you enter should appear in the log file, and the system call should return a value of 0.

If that succeeded, then you are now a kernel programmer!

It would probably be wise to commit your work to your local git repository, and then to push your work up to your remote repository on GitHub. From your kernel source tree run this command to see what files have been changed:

```
git status
```

Use

```
git add <file-to-stage>
```

to stage a modifed file or a new file for a subsequent commit.

If you're impatient, from the root of your repository you can run

```
git add .
```

to stage all modified files, all new files, and all deleted files in one fell swoop.

Use the following command to commit all your staged files to your local repository. Note that no changes are made to the remote repository.

```
git commit -m "Commit comment."
```

Synchronize your local repository with the remote repository before a push (Used to pull revision history pushed by a collaborator).

```
git pull
```

Now, push your changes from your local repository to the remote repository:

```
git push
```