Adding a System Call to the Kernel

CS 411

*** 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.

Open kernel/labsyscalls.c for editing. Edit the stub for labsysc1. 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(), but writes into the system log file. Refer to documentation and examples online for printf() to guide you. The function should return an integer value of 0. It should print this message into the system log:

   labsysc1 called and passed the value: <val>.

where <val> is replaced by the passed value. For example:

   labsysc1 called and passed the value: 42.

Save the file.

Now, you need to write and compile a userland C program to exercise your system call. Create a Userland directory immediately under the root of your kernel source tree to hold these programs. Something like the following should do the trick. Name the file sysc1test.c

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

int main(int argc, char *argv[])
{
  if (argc != 2)
    {
      printf("One integer argument expected!\n");
      return -1;
    }

  printf("labsysc1 returns: %ld.\n", syscall(333, atoi(argv[1])));

  return 0;
}
```

Once your finish writing the program, compile it:

```
gcc -o sysc1test sysc1test.c
```

Now, we need to let git know that it shouldn't include the binary file sysc1test in the repo. Create the file Userland/.gitignore and add the line

```
sysc1test
```

to it. Save this file. Each time you write a new program in Userland, add a line to this .gitignore to keep the program's binary out of your repo.

Return to the root of your kernel source and use the 'git status', 'git add', 'git status', 'git commit', 'git push' sequence to stage, commit, and push your changes.

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
```

Correct any compilation errors that occur during the second 'make' command

and rerun the 'make' command. Repeat until you get a clean kernel build.

Once the kernel builds cleanly, 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/syslog
```

Run your userland test program in another shell. Use 'cd' to enter your Userland directory and run your program:

```
./sysc1test 42
```

The integer value you enter should appear in the log file, and the system call should return a value of 0. Run the program a few times, using different integer values.

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