April 9, 2008
Operating Systems
LIACS
Spring Semester 2008
Assignment #6

Deadline: Tuesday, April 22, 2008
You are encouraged to work in teams of 2 persons.
Try to work on the same PC you used last week.

Goal
Get some hands-on experience with the Linux kernel. Learn how to add and use system calls.

Introduction
In this assignment you will create your own kernel function and the corresponding system call for Linux. A system call is the name of a kernel function that is exported for use by user-space programs. Kernel functions cannot be called directly like ordinary functions. Instead they must be called indirectly via the trap table using the trap instruction. So if you write a new kernel function you need to create a new entry in the kernel trap table to reference your new function.

A system call can be called through the `syscall()` function. This function takes 1+N arguments, where N is the number of arguments to your own function. The first argument to `syscall()` is the number of the system call you wish to call. This number is used to look up the address of the actual function in the `syscall_table`. The precise way this is done may not be immediately clear in this assignment because of changes between different kernel versions and architectures (the machines in room 411 have 64-bit CPUs and run a 64-bit version of Fedora).

Functions already included in the `syscall_table` include commonly used functions like `exit`, `fork`, `getpid`, `open`, etcetera. However, the numbering of these functions may differ from one kernel to another. This isn't a real problem because these system calls are rarely called directly using the `syscall()` function. For most, if not all system calls there are stub functions which perform the `syscall()` for you. These can be generated semi-automatically through macros, but this is once again dependent on the kernel version you are using.

Assignment
Let's start by writing the function itself. We can create new .c files in the appropriate (or a new) directory, or we can add the function to an existing file. Since the Linux kernel is one big program we could place the function almost anywhere, but for this assignment we'll place it in a file which already contains a large number of system calls.

Note: all files and paths in this assignment are relative to the source directory you got when you extracted the source tarball: `/usr/src/linux...../`

Open `kernel/sys.c`. Go down to the very end of the file and add the following function:

```
asmlinkage long sys_multiply (int a, int b) {
      int i, result = 0;
      for (i = 0; i < b; i++) {
            result += a;
      }
      return result;
}
```

This simple function (which multiplies its two integer arguments) will be part of the kernel and at the end of this assignment we will be able to let the kernel run this function. Note that this particular example isn't that useful. System calls are used for standard functionality the kernel provides such as process control (fork, exit, et cetera). If we write a program which utilizes this kernel function and we try to run it on another computer (which doesn't have this kernel function) the program will not work. Because of these compatibility problems system calls have been standardized to a certain degree. For example, the POSIX standard requires that an OS provides an `exit` system call, but it does not prescribe a specific number to use for this system call.

Now our function will be linked into the kernel. If we had created a new file for our function we would also have to edit some Makefiles. To actually make the function available as a system call we need to add an entry to the `syscall_table`. The table is found in the file `arch/i386/kernel/syscall_table.S`. We will add an entry to the bottom of the table:

```
.long      sys_multiply
```

We also need to edit the file `include/asm-x86-64/unistd.h`. Search for the line containing `__NR_syscall_max` and add the following lines before it:

```
#define __NR_multiply 280
__SYSCALL(__NR_multiply, sys_multiply)
```

The number 280 should be one higher than the system call on the lines above it. We also need to change the line containing `__NR_syscall_max` to reflect the new number of system calls. Replace it with:

```
#define __NR_syscall_max __NR_multiply
```

Finally we need to edit the file `include/linux/syscalls.h`. At the very end of the file add the following line:

```
asmlinkage long sys_multiply(int a, int b);
```

Now we are done editing the kernel source. We can `make` the new kernel as we did last week (with a name you can easily identify). While the new kernel is being made you can write the user-space programs which will use the new system call.

Now we'll look at how to use the function we added from a user-space program. As mentioned before, we'll use the `syscall()` function.

```
#include <stdio.h>
int main() {
     printf("6 * 7 = %i\n", syscall(280, 6, 7));
}
```

1) Run this program using your current (unmodified) kernel. **Describe what happens and why.**
2) Now reboot and load your modified kernel. Before rebooting you may wish to edit `/boot/grub/grub.conf` to increase the timeout value for grub. Run the program again once again **describe what happens and why?**
3) Now play around with the numbers a bit (the implementation of multiply is inefficient and slow for high values of b) and `time` the program. **Describe what's noticeable about the output of `time` and explain it.**

Deliverables:

i A README file containing the answers to the three questions on the previous page. Also mention the name of the PC you used, the EXTRA_VERSION string of your kernel and a list of files you modified (including the full path on your machine).
ii All files you created or modified.
iii A one-page lab report detailing what you have done in the lab and more importantly what you have learned from this and what you have learned from your lab partner.

Make sure the lab report contains the names of the authors, student IDs, assignment number and date turned in!

Due date: as stated before April 22st.

Put all files you want to deliver into a separate directory (e.g.,*assignment6*), free of object files and/or binaries, and create a gzipped tar file of this directory:

tar -cvzf assignment6.tgz assignment6/

Mail your README and report to Sjoerd Henstra (e-mail: shenstra at liacs dot nl) and *let the subject field of your e-mail contain the string* `ÒOS Assignment 6".