MP 1 – Implementing a System Call in Linux

CS 423 – Spring 2011
Revision 1.2

Assigned January 26, 2011
Due February 9, 2011, 11:59 PM
Extension 48 hours (penalty 20% of total points possible)

1 Change Log

1.0 Initial Release.

1.1 Correct some mistakes resulting from mixing the way things were handle in older versions of Linux, particularly on the x86 32 bit architecture with the way they are handled now for the x86 64 bit architecture. Cleaned up some garbled sentences. Added some addition hints for Problem 1.

1.2 Fixed the overflow in Problem 3.

2 Objectives and Background

The purpose of this MP is to help the students:

- Add a new system call to Linux and install the modified kernel.
- Setting up a virtual environment (using Qemu) for testing the modified kernel.
- Understand the concept of processes, threads and thread groups, as presented by Linux.

3 Resources

In order to do this assignment and test it, you will need to be able to use QEMU and install a custom kernel in it. Please read Guide to Testing a Kernel and follow its instructions to set up an environment for your use in doing this assignment.

A good place to start learning about Linux kernel programming is Kernel Newbies

The Linux kernel has millions of lines of code and using traditional grep for searching is not advisable in most of the cases. You are advised to use cscope and LXR instead (or at least in addition and before). Csocpe's database can be generated by invoking *make cscope* on the Linux source.

A resource that has been useful to students in past semesters is Robert Love's Linux Kernel Programming. To understand how Linux kernel programming differs from application level programming, you might consult Chapter 2 (Section *A beast of a different nature*).

As the Linux kernel's source code is fairly complex, before reading the source code, try to get an overall picture. Both the above book and Understanding the Linux Kernel by Daniel P. Bovet and Marco Cesati can help.

4 Adding a System Call in Linux

Once you have carried out all the instructions in Guide to Testing a Kernel, you should have created a COW disk image file called `mp1.img` and a directory called `linux`.

Adding a new system call in Linux involves three steps:

1. Define the basic system call in `kernel/sys.c`, i.e. implement the function that'll be called when this system call is invoked. (It can go in another file, but for this MP we will expect you to put it here.)

2. Add an entry to `include/linux/syscalls.h` giving prototype of the system call.

3. Associate a number to the system call in the pertinent library file (`arch/x86/include/asm/unistd_64.h`). You need to define your system call number after the last system call already listed in the file. (You no longer need to increment `NR_syscalls`; this is now computed for you.)

In this assignment, you will be asked to return a value, which requires copying data from kernel space to user space. First, when passing a pointer to receive the data, you will need to add to the function argument type the modifier `__user`, as in `int time( long __user *t )`. This is a macro that tells the compiler not to dereference the pointer, as it will not be meaningful in the kernel address space.

So, how do you get information into user space then? For basic types, you can use `put_user` to write data out to user space (and `get_user` to read data of basic type in from user space). For copying blocks of data such as arrays or structures to or from user space, you should use `copy_to_user` and `copy_from_user`. (Strings have their own special functions.) For a better understanding, you are encouraged to read the article *Adding a Linux system call* here.

5 Thread Groups

Thread groups were implemented in Linux to provide convinience in implementing the POSIX's pthread library in Linux. POSIX defines a multi-threaded process as a collection of related threads, which share all but some resources. It also requires that certain actions can be applied to all threads of a process at once. Linux, on the other hand, deals at thread level, not at process level. Thus, support for thread groups was added in Linux, for grouping together all the threads of a multi-threaded process, as required by POSIX. Interested students can find more details in this paper: `www.kernel.org/doc/ols/2002/ols2002-pages-330-337.pdf`. You should have encountered programming with pthreads in CS241. Additional help in programming with POSIX threads may be found at POSIX Threads Programming.

6 Problems

1. (15 pts) Add the structure

```
struct thread_group_data {
   pid_t tgid; //thread group id
   int num_threads; //number of threads in the thread group
}
```

   to the very end of the file `include/linux/sched.h`. Add to the end of `kernel/sys.c` a system call implementation conforming to the following function prototype:

```
int k_get_thread_group_numbers(pid_t tid, thread_group_data *data)
```

   that, given a thread id `tid` (see `gettid()` system call), returns the associated `thread_group_data`. If the id number given is invalid (fails to correspond to an existing thread), the system call should return `-ESRCH`. If the buffer pointer specified is invalid, the system call should return `-EFAULT`. If it succeeds, it should return `0`.

   NOTE: `gettid()` might not be available on your system. Use `syscall(186)` instead, where 186 is the number associated with `gettid()` for x86-64 systems.

   Hints:

   - To gather information about a thread, you will need to look in its process descriptor, which is an entity of type `task_struct`. The structure `task_struct` is defined in `include/linux/sched.h`. This is the file at the end of which you are to put the definition of `struct thread_group_data`. One entity of type

task_struct that always exists is init_task. It is defined in linux/arch/x86/kernel/init_task.c. Another is current. The structure task_struct makes heavy use of doubly linked lists via the structure list_head. Functions supporting list_head are defined in include/linux/list.h.

- Shared data structures of the kernel need locking to avoid concurrent access, which might result in corruption of the data structure, or inconsistent results.
- Use LXR to find out the existing instances of usage of a shared data structure in kernel. Check what kind of locking is used to protect the data structure elsewhere; you should use the same in your system-call implementation.
- When in kernel, it's OK to use goto, esp. for handling error conditions.

2. (5 pts) Add to the end of kernel/sys.c the code to make the system call get_thread_group_numbers visible to user space with the same prototype as k_get_thread_group_numbers . You should use SYSCALL_DEFINE2 (a macro defined in terms of asmlinkage).

3. (3pts) Add an entry for the new function in include/linux/syscalls.h. Update the file arch/x86/include/asm/unistd_64.h to associate a new system call number (the next unused number) with the newly created system call function.

4. (5 pts) Write a multi-threaded program (using pthread library)

   ```
   void test(int n)
   ```

   that does the following:

   1. Create n number of threads, with identical handler functions.
   2. In the handler function of the threads, make the thread sleep for a random amount of time and then print the results of the newly created system call, passing the current thread's tid (using gettid() or syscall(186)) as an argument.

   Compilation of your program should be done inside qemu running a kernel including your code for the new system call.

5. (5pt) In a file named README-cs423-mp1 put you name at the top, followed by documentation explaining your system call implementation and any design decisions you made. This file should specifically mention each file that was altered and concise description of those changes.

7 Deliverables

To turn in your assignment, you should create a directory mp1. In that directory, you should recreate the directory structure of linux-2.6.37.0 for every directory containing a file you created or altered, and in the corresponding directories you should place those files. You directory should contain a file README-cs423-mp1 containing your name and a description of all the changes / additions you made. Once this directory is complete, you should run

```
tar czf mp1.tgz mp1
```

The resulting file mp1.tgz is what handin will upload for you.