How to Create a System Call

Posted by Alejandro G. Carlstein Ramos Mejia on October 16, 2010 May 14, 2012 About Programming / ANSI/POSIX C / C++ / Linux / Operating Systems / Operative Systems

Note: Please check previous post about How to Build a Custom Kernel.

NOTIFICATION: These examples are provided for educational purposes. Using this code is under your own responsibility and risk. The code is given 'as is'. I do not take responsibilities of how they are used.

1. **Prepare your system call**:
    1. Go to the kernel source code folder: cd /home/your-home-folder/linux
    2. Create a personal folder: mkdir yourcall
    3. Access your folder: cd yourcall
    4. Create your source file: your_sys_call.c
    5. Create a makefile: Makefile
2. **Configure some kernel files**
    1. Add your system call at the end of the file syscall_table_32.S
        1. cd /home/your-home-directory/linux/arch/x86/kernel
        2. gedit syscall_table_32.S
           iii. Add at the end of file: .long sys_your_new_system_call
    2. Add your system call at the end of the file unistd_32.h
        1. cd /home/your-home-directory/linux/x86/include/asm
        2. gedit unistd_32.h
        3. At the end of the file, add (Where XXX is the previous number plus 1)

           ```
           #define __NR_your_new_system_call XXX
           ```

    3. Increase by the number of system calls to the total system calls number: #define __NR_syscalls XXX (Where XXX is the previous number that was plus one.)
    4. Add the declaration of your system call at the end of syscalls.h
        1. cd /home/your-home-directory/linux/include/linux
        2. gedit syscalls.h
        3. asmlinkage long your_new_system_call (parameters you want to pass)
    5. Add the new folder to the kernel compile's Makefile

       ```
       core-y += /kernel ... other folder... /yoursyscall
       ```

3. **Configure your system call**
    1. Write your system call, inside yoursyscall.c

```
asmlinkage long new_system_call (whatever params you want to pass){
// whatever you want to do
}
```

2. Change the makefile by adding the following line: obj-y := yoursyscall.o
3. Compile your kernel (Follow steps at Building a New Custom Kernel

4. **Test your system call**
    1. Create a user level program that calls your system call
    2. Create a header file that the user space program can use:

```
/* header.h */
#include < linux/unistd.h >
#define __NR_new_system_call XXX

/* if you system call returns int and takes no parameter
* use this macro
*/
_syscall0(int,new_system_call)

/* Otherwise, depending on the number of parameters
* being passed use the _syscallN macro, N being the no
* of params, like
_syscall1(int, new_system_call, int)
*/
```

5. Starting at kernel 2.6.18, the _syscallXX macros were removed from the header files to user space, therefore syscall() functions is required to use:
    1. printf (“System call returned %d \n”, syscall (__NR_new_system_call, params_if_any));
    2. or change the header.h file:

```
* header.h */
#include < linux/unistd.h >
#include < sys/syscall.h >
#define __NR_new_system_call XXX

long new_system_call (params_if_any){
   return syscall (__NR_new_system_call, params_if_any);
}
```

6. Test the code:

```
/* test client */
#include 'header.h'

int main (void){
    printf ('System call returned %d \n', new_system_call());
    return 1;
}
```

NEW!!!

Before you do update-grub (if using , do the following command line (Where x.x.xx is the kernel version you compiled):

```
sudo update-initramfs -c -k x.x.xx
```

After Ubuntu 10.04 (or kernel 2.6.32) this seems to be a needed extra step to make it work.

If you encounter any problems or errors, please let me know by providing an example of the code, input, output, and an explanation. Thanks.