Lab 4: System Calls - Reading Material

Introduction

Goals

1. Understanding the interface to the operating system services.
2. Gaining independence from standard support libraries.
3. Basic understanding of code manipulation, through implementation of executable file patches.

Motivation

Aside from the clear advantages implicit in the above goals, the resulting executable files are very small and simple, and require little space to execute. This has clear performance advantages, when such are desired. Additionally, the object and executable files have a very simple structure, and are easier to understand - something that will be very useful in later labs.

Methodology

1. You will be provided with basic interface code in assembly language, (file start.s). The file contains just the "glue" code of two interfaces:
   - Code to start up main() by getting argc and argv and passing them to your main().
   - A function named system_call() to invoke system calls.
   - You should LOOK at the file first, and understand the code! (If you are taking only the SPlab part, no need to understand the assembly language, reading the comments should suffice.)
2. Understand how to generate stand-alone code that uses only your C code and the "glue" from start.s, with no libraries whatsoever.
3. All that needs be done is compiling your code, assembling the glue code (start.s) and linking them together to produce an executable. (Note: in future labs, you will learn how to write a simplified version of ld - the linker - to create your executable from compiled object modules).
4. Exercise with some system calls, mainly file system related.

Reading Material and references

Make sure you read and understand the following **before** attending the lab.

1. System calls: the basic mechanism for accessing OS services

   Note: Faulty Documentation! Since the documentation of the actual system calls is not good, we provide the documentation based on the Linux man (2) of the respective wrapper functions. They are functionally the same as the respective system call except for the **return value**. The actual return value when an error occurs is a negative value that is **not** necessarilty -1 as stated in the documentation and the presentation. That is because the raw system calls have no access to the errno variable. Instead, they return the negative value of the error code. Therefore, you need to check whether the returned value is less than 0 rather than equal to -1.

2. The exit system call
3. Read the "man 2" page for the following system calls: open, read, write, close (or re-read them if you have read them before, just to make sure you understand them). Although these man pages are for standard-library interfaces to the actual system calls, the functionality of these interfaces is the same as the actual system call, except for the issue of returned value and errors, as stated above, and other minor issues.

   It might benefit you to note that the system calls open( ), close( ), read( ), write( ) are also quite similar to some of the functions that you have already used in previous labs: fopen( ), fclose( ), fread( ), fwrite( ), respectively. The difference is in the file descriptor (now int (index into an open file table held managed by the system) rather than FILE* (which is a structure which also contains the above mentioned int index)), as well as some of the argument values, but the functionality is essentially the same. The version with the "f" that you have used before provides an additional level of abstraction/wrapping.

4. Read the manual page for the following Unix system call:
   getdents [141].
   The syntax for reading a system call man page is:
   `man 2 <sys-call>`. For example, `man 2 getdents`.

   Notice that the code in the man page is different to what we expect (they use a different method for system calls - syscall, we expect you to use the system_call function we provide in this lab). Again, make sure you don't include any external files. As always, If you use any code you didn't write yourself, make sure you understand

everything 100%. For more information you can read this document from last year, beware of some inaccuracies. The getdents system call.pdf

5. Bitwise operations in C
6. Linux System Call Table