Machine Problem 1: Familiar with Various Operating Systems and System Calls

The Assignment

For this lab, you will first implement some simple program using standard C library APIs, then learn its system call representations in various operating systems, such as Linux and Windows. You will also implement the same functions using POSIX system calls and compare with the original program. In addition, you will implement the same (and new) functions in some very simple Android app.

1. Write a C program `simpleFile.c` that uses standard C library APIs (e.g., `fopen, fclose, fgetc, putchar`) for simple file operations. You are going to read an existing file (please prepare a small text file by yourself), output to screen, copy to a newly created file, save the file, and finally delete the newly created file.
2. Compile and run in both Linux and Windows. You can use GCC in Linux and LCC-win32 (http://www.cs.virginia.edu/~lcc-win32/) or Visual Studio in Windows to compile the program.
3. Use the `strace` command to trace the system calls of your program in Linux. Use the API Monitor program (http://www.rohitab.com/downloads#APIMonitor) to trace the NT Native APIs in Windows (you can roughly consider it as equivalent version of POSIX system calls in Windows). Try to find and report the mapping between each involved C library call and its corresponding system call(s) in both Linux and Windows.
4. Implement the same functions mentioned above (step 1) using POSIX system calls (e.g., `read, write`) instead of C library APIs. Name your new program as `simpleFile2.c`. You only need to implement the Linux version (Windows version NOT required).
5. Use the time command to measure how fast your two programs are (`simpleFile`, and `simpleFile2`) in Linux (e.g., "`time simpleFile`"). Report the performance on a small text file (e.g., only a few bytes) and a relatively large file (e.g., several MBs).

Explain the difference if any. You can pick any text file as you want by yourself.

6. Implement the same functions (as `simpleFile.c`) in an Android application `simpleFile3.apk` (note: this time you need to write in Java). Trace the system calls at the Linux kernel level when running the Android application and compare with above. Try to find and report the mapping between each involved program/function call in the app and its corresponding system call(s) in the kernel.

   Some resources on Android lab (find more through google by yourself)
   - http://www.makeuseof.com/tag/write-google-android-application/
   - http://developer.android.com/tools/index.html
   - http://www.linux.com/learn/docs/683628-android-programming-for-beginners-part-1
   - How to install Android 4.04 ICS in your PC (in Virtualbox) http://www.redmondpie.com/how-to-install-and-run-android-4.0-on-mac-windows-pc-or-linux-tutorial/
7. Now further implement a simple new Android app (`simpleApp.apk`) with some Android specific functions, e.g., at least including read phone contact, read and send SMS, and make a phone call. Trace their system calls and try to do similar mapping as done in previous steps. Can you understand the Android semantics at the system call level? How can you understand the semantics if given some unknown Android system call sequences? Talk about your ideas/thoughts and findings.

What to Hand In

- The programs: `simpleFile.c`, `simpleFile2.c`, and the corresponding apk file/source for Android (e.g., in the name of `simpleFile3,simpleApp`).
- System call output files in both Linux and Windows (and Android) in pure text.
- A detailed report that describes your results, findings, and thoughts. It should contains answers to all the questions above, detailed procedure with necessary screen captures, issues/problems encountered and how you fix them, understanding of systems after the lab, and any lesson you have learned and thoughts you have about the lab.