CS 3013 Operating Systems — WPI, A Term 2020
Craig E. Wills — Project 2 (15 pts)
Assigned: Friday, September 11, 2020 — Due: Tuesday, September 22, 2020

Note: This project is to be done on an individual basis.

Introduction

An interesting approach for studying the behavior of an application is to trace the set of system calls invoked during the lifetime of the application. The *strace* command in the Linux Operating System executes an application given as arguments to it and prints all system calls and their arguments used by the application. In this project you will use *strace* to analyze the behavior of different applications.

To begin the project, you should use *strace* to execute a few commands and observe the output. For example, try "*strace ls*" or "*strace ls -l*". The system call trace is printed to stderr (standard error) rather than stdout (standard out). This distinction allows it to be redirected to an alternate location. A small shell script in
`https://web.cs.wpi.edu/~cs3013/a20/dostrace`
is available for you to use. This script executes the command given to it with *strace* and redirects the trace of system calls to a log file. Use of the script also allows interactive commands to be more easily traced.

You can use *wget* to retrieve the shell script (the quotes prevent your shell from interpreting the tilde character) and then use *chmod* to change the permission mode for execution. Once done, you can execute any command with the script, such as shown for *ls* in the following:

```
% wget 'https://web.cs.wpi.edu/~cs3013/a20/dostrace'
% chmod 755 dostrace
% ./dostrace ls
<output of ls command>
strace output in ls.slog
```

System Call Counts

The resulting log file is to be analyzed for this project. You need to create a program *traceanal* to analyze a log trace file (or the output of *strace* directly). Your program should read the trace of system calls from stdin so it can be invoked either as "`traceanal < ls.slog`" using the log file generated above or "`strace ls |& traceanal`". Your *traceanal* program should strip out any system call arguments and examine only the system call names. It will also need to ignore command output if you use the latter invocation.

As a starting point, your *traceanal* program should determine and print the total number of system calls invoked as well as the total number of unique system calls used. This information should be preceded with "AAA:" so any sorting will leave it as the first line. Next your program should print the count of times that each system call shows up in the trace. In C or C++, you will likely want to create a hash function to map system call names to a hash table.

A sample execution of your program on a log file should look as follows (your system calls and counts will vary) with the list of invoked system calls and the count for each. You do not need to print system calls in any particular order.

```
% ./traceanal < ls.slog
AAA: 108 invoked system call instances from 21 unique system calls
open  9
mmap 19
read  7
close 11
...
```

Note: you can use the Linux *sort* command and a pipe to sort your output. A couple examples are to sort by system call name or in reverse order by the count:

```
% ./traceanal < ls.slog | sort
AAA: 108 invoked system call instances from 21 unique system calls
close 11
mmap 19
open  9
read  7
...
% ./traceanal < ls.slog | sort -nrk 2
AAA: 108 invoked system call instances from 21 unique system calls
mmap 19
close 11
open  9
read  7
```

System Call Sequence Pairs

Satisfactory completion of the system call count portion of the assignment is worth 8 or the 15 points. For five additional points, your program should be extended to also determine the subsequent system call that is used. Your program should show this information if the command line argument "seq" is added after the command name. Your program should print the count for each system call as well as the count for each subsequent system call pair (indented) as follows:

```
% ./traceanal seq < ls.slog
AAA: 108 invoked system call instances from 21 unique system calls
open 9
    open:fstat 4
    open:read 5
mmap 19
    mmap:mprotect 5
    mmap:mmap 6
```

```
    mmap:close 7
    mmap:arch_prctl 1
...
```

indicating that the *open()* system call is used 9 times in the trace. It is followed 4 times by a call to *fstat()* and 5 times by a call to *read()*. The output shows *mmap()* is followed by calls to four different system calls, which includes itself. Your program should print similar information for all system calls used in a trace. Again, system calls or system call pairs do not need to be written in a particular order, but can be piped to the *sort* command for sorting.

Additional Work

Completion of the program is worth 13 of the 15 points. For the remainder of the project you you need to use it to analyze the behavior of different programs on a system. You should use it to investigate the following issues:

1. Observation of program execution behaviors shows that many system calls are invoked as part of starting up a program. To examine this start-up behavior, construct a simple program that makes no system calls and analyze it using *traceanal*. Do all programs exhibit a similar start-up behavior in terms of which system calls are used and their relative sequence?

2. Researchers have proposed using the system call sequence of a program as a "signature" for that program as a means to detect if a copy of a program is substituted by an intruder. Investigate the validity of this idea by checking if the signature of different executions of the same program are the same. The particular counts of system calls may vary, but are the sequences similar? What if different command line arguments are used for a command? Is there variation in the sequence? Does the sequence change if the amount of data or duration of execution varies for a program?

3. How much variation and commonality do you observe from invocations of different commands? You should try to separate out the start-up behavior common to all commands and the command-specific portion.

Submission of Project

Use InstructAssist to submit your project using the assignment name "proj2". At a minimum you should submit your program code and a file showing execution of your program on test cases for your program. If you do the last portion of the project then you should also submit a short report (at most a page or two) examining the issues in **pdf format**.