COS 333 – Advanced Programming Techniques

Programming Assignment II

Assigned: Monday February 26, 1996

Due: 11:59pm, Thursday March 7, 1996

Program: You are a spy. Your task is to deduce information about the internal operation of existing programs. Unfortunately, you do *not* have access to the program sources. Your *modus operandi* is to first use your system's `trace` command to produce a trace of the system calls made by the program under investigation—`trace`, given an arbitrary Unix program or command, produces a trace (on standard error) of the system calls made by the program or command. For example, in `ksh` the command

```
$ (trace ls -l) 2>ls.trace
```

will trace the system calls made by `ls -l` and will place the trace output in the file `ls.trace`. For more information on `trace`, see your `man` pages.

The only tools at your disposal are a shell[1], and the common Unix utilities (including `grep` and `sed`). You are *not* to use a general-purpose programming language (such as C, C++, or `perl`)—only programs completely written in a shell and using the Unix utilities described in lecture can receive full credit.[2]

Your program will process a trace file supplied as standard input. From the input, your program must extract two pieces of information:

(i) Your program must track basic file operations in the trace. In particular, for each `open`/`close` sequence on file descriptor *fd*, your program should report the number of bytes read (syscall `read`) from and written (syscall `write`) to descriptor *fd*.

For example, upon finding '`close (3) = 0`' in the trace, your program reports

`file descriptor 3: read` *r* `bytes, wrote` *w* `bytes`

where *r* is the sum of the bytes read on this instance of file descriptor 3, and *w* is similarly the number of bytes written.

For descriptors that are already open when the trace commences (*e.g.*, standard input), your program should still correctly accumulate the above information. For descriptors not explicitly closed, print partially-accumulated totals after processing the entire trace.

(ii) If the trace file contains a successful call to `getuid`, your program will examine the accounting information (`ypcat passwd`) and extract the name, location, and phone number of the account matching the uid returned from `getuid` (the uid is the third field in an account entry). You may use the `sed` script presented in lecture for dissecting account entries. If no uid information is in the trace, your program need not display any accounting information.

Submission Procedure: see the course's web page

[1] You may use your favorite shell, although `ksh` programming is strongly encouraged (`ksh` is available on `phoenix` as `~cs333/bin/ksh`).

[2] Check with the instructor or teaching assistants if you want to use other tools in your program.