Due: Wednesday, February 12, 2014. 11:59pm (no extensions).

What to submit: A tar file that should contain the file `answers.txt`, which must be an ASCII file with answers for questions 1 and 2. For question 3, which asks for code, include a file named `dpipe.c` in your tar file.

Use the submit.pl script in ~cs3214/bin/submit.pl to submit your tar file from the command line, or use the submit website. Use the identifier 'ex1.'

The assignment will be graded on the rlogin cluster, so your answers must match this environment.

Understanding Processes and Pipes

In this exercise we consider a program 'dpipe,' which, in connection with the netcat utility program, could be used to turn any ordinary program into a network server.

In this exercise, you're asked to observe, reverse-engineer, and then reimplement the dpipe program. The dpipe binary is provided in ~cs3214/bin/dpipe on our machines. For netcat, use the ~cs3214/bin/gnetcat binary. To allow you to invoke those commands directly, make sure that ~cs3214/bin is in your PATH environment variable.

Here's an example session of how to use it. Say you're logged on to the machine 'locust' and run the command `dpipe wc gnetcat -l 15999`[1]. This will produce output as follows:

```
[cs3214@locust sys1]$ dpipe wc gnetcat -l 15999
Starting: gnetcat
Starting: wc
```

Note that the shell is still waiting for dpipe to complete. The questions listed in part 1 must be answered at this point, after you have started dpipe, but before the next step. You may wish to continue reading and then return to this point, after opening a second terminal on the same machine.[2]

Next, on a different (or the same) machine, run

```
[cs3214@chinkapin ~]$ gnetcat locust 15999 < /etc/passwd
     37      63    1754
[cs3214@chinkapin ~]$
```

This gnetcat instance will connect to the gnetcat instance running on locust, and send its input there (in this case, the content of file /etc/passwd). Gnetcat (running on locust) outputs this data to its standard output stream, which is connected via a pipe to the standard

[1] When you try this out, please replace 15999 with a number that with high likelihood is unique so you do not conflict with other students using this machine, let's agree to using 10000 + last 4 digits of your VT PID here.

[2] Hint: though you cannot predict to which machine you'll be assigned after ssh'ing to rlogin, you can always ssh into the desired machine once you're on any rlogin machine.

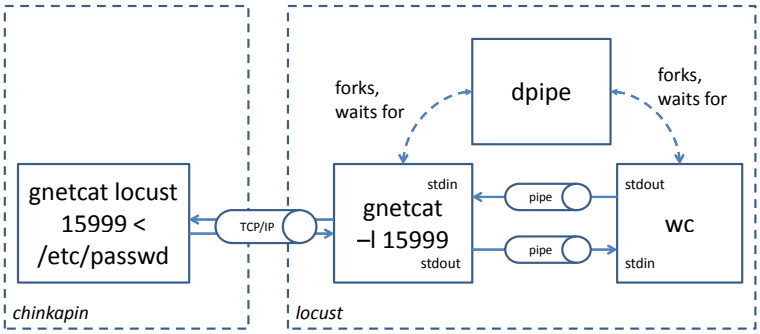

Figure 1: Using dpipe to cross two programs' standard in and out streams.

input stream of the 'wc' program. The standard output stream of 'wc' is connected to the standard input stream of locust's gnetcat instance. Anything output by 'wc' will then be forwarded to chinkapin's gnetcat instance, which outputs it to its standard output, thus appearing on the user's terminal on chinkapin. The entire scenario is shown in Figure 1.

Finally, dpipe exits when both of the processes it started have terminated:

```
[cs3214@locust sys1]$
```

1. **Using /proc** The `/proc` file system, originally introduced in Solaris, is a mechanism by which the Linux kernel provides information about processes. This information is organized in directories and files that can be accessed using ordinary tools such as ls or cat. The kernel creates a directory for each process named for its pid. Within the directory, there are additional files and directories that describe the current status of the process.

   After you have started dpipe with 'wc' and 'gnetcat -l', use the ps(1) command to find the process ids of these three processes, then use /proc to find the parent process id, the process group id, and the status of these processes.

   (a) Provide the following information in table form:

Process	PID	Parent PID	Process Group ID	State
dpipe				
wc				
gnetcat				

   You may find the proc(5) man page useful. Use the single letter Linux short-hand for the process state.

   (b) You should find that all three processes are in the same state. To which of the states in the simplified process state diagram in lecture does this state correspond?

   (c) Now examine the `/proc/nnnnn/fd` directories for each process. Use `ls -l`. For each process's open file descriptors, enter in the table below the device or object to which it refers. Enter N/O if a file descriptor is not currently in use.

File Descriptor	dpipe	wc	gnetcat
(stdin) 0			
(stdout) 1			
(stderr) 2			
3			

Look at Figure 1 and convince yourselves that dpipe indeed creates the arrangement shown there (minus the network connection, which has not yet been created).

2. **Using strace.** Processes use system calls to obtain services from the kernel. The program 'strace' is a utility that can display which system calls a process is executing. strace can trace individual processes from start to end; on request, it can trace child and anchestor processes as well; and it can even trace already running processes. Familiarize yourselves with the 'strace' command (use 'man strace', for instance). Frequently used switches include '-f', '-e', '-p', '-ff', and '-o'.

   Run dpipe under strace, creating again the constellation of processes described in the introduction (wc and gnetcat), and use gnetcat on another machine to send data until dpipe exits. Use the '-ff' and '-o' switches to strace to obtain separate listings for dpipe and its children.

   (a) Identify which system calls the dpipe process made. From the relevant strace output, copy the entries corresponding to the system calls pipe(), close(), fork(), wait(), and exit(). Hint: Linux's fork() system call is called clone(), wait() is implemented using wait4(), and exit() corresponds to exit_group().

   (b) Identify which system calls the process executing 'wc' made. Include the entries corresponding to write(), dup2(), close(), and execve(). Stop at the first execve() call that was successful (listed as returning 0 in strace's output.)

   (c) Like part b), except for the process executing 'gnetcat'.

   Note that the traces for part (b) and (c) correspond to the system calls made by the child processes spawned by dpipe between fork() and exec().

   (d) Now repeat this exercise, running strace with the switches
   `-f -e wait4,accept,read`
   Before you connect the gnetcat client (like in part 1), all three processes will be inside the kernel, in one of these three system calls. Which process is in which system call, and how can each process's state be explained from the semantics of that system call? 3 answers are required: e.g., process 'dpipe' is in the (insert state here) because it is (describe what caused it to be in this state).

3. **Implement dpipe.**

   Armed with the knowledge gained in the previous part, implement dpipe so that it issues exactly the system calls you've observed in the traces.

   Although the OS will use predictable file descriptors in pipe(), your program should use variables and not constants, e.g. 'close(3)' would not be an acceptable solution.

Use the execvp() wrapper for the exec() system call. Make sure you implement argument passing correctly. dpipe should first invoke the program whose name is given as its first argument, without passing any additional arguments to it. Then it should interpret the second argument as the name of a second program and pass the third argument as the first argument to the second program, the fourth argument as the second, and so on.

Note that it is custom to set argv[0] to the name of the program itself, and your invocations of execvp() should do the same. If done correctly, the strace output of your dpipe must be identical to the output obtained from the provided dpipe. Hint: it is possible to achieve this without creating a complete copy of the argv[] array passed to dpipe's main program.

Make sure that your dpipe does not return to the prompt prematurely, or fails to return to the prompt after its children have exited.

Important: We will use the compiler switches `-O -Wall -Werror` when compiling your program. If you ignore any warnings, you run the risk of failing our autograding scripts.