Network Programming, Fall, 2003
Project 1: Remote Access System (ras)

9123571 Chung-Wei Hang
at Learning Lab, CIS, NCTU
gis91571@cis.nctu.edu.tw

2003.10.22

1 Requirement

In this homework, you are asked to design rsh-like access systems, called remote access systems, including both client and server. In this system, the server designates a directory, say ras/; then, clients can only run executable programs inside the directory, ras/.

2 Program Overview

The program contains 4 main C language source files:

- **errexit.c** Definition of errexit function.
- **passivesock.c** Provide socket connection functions used in "passiveTCP.c".
- **passiveTCP.c** Provide TCP protocol functions used in rasd.c.
- **rasd.c** The main part of the program.

rasd.c

The rasd.c contains following important functions[1]:

[1]It only lists important part of functions in "rasd.c".

- **main** The entrance of the program. it creates master socket called "msock" who takes care of the connection from each client. After accept the client connection, it forks a child process and call a function "ras" to handle all the command sent by the cliet.

- **ras** This function uses an infinite loop to read what the client sends until the client disconnects by "exit" command. After each time it reads, the function call "parse" function to parse commands the client sends.

- **parse** The "parse" function do the main job in this program. First, it decomposes the command into arguments, including pipe and redirection. Then, check out the state of pipes kept in the data structure and call proper "dup2" function to organize "stdin" and "stdout" of both the parent and the child process. Finally, it executes the command by calling "execvp" function, and that function will search from the "PATH" environment variable and execute the executable file if it exists.

3 Implementation Issues

I sumerize the main implementation ideas I used in the program. They can be described as follow:

Pipe Management

The management of pipes in the program is maintained by a sorted array called "pipe-list". This list keeps read and write file descriptors of pipes and is sorted by pipe target. This target means the destination of the pipe. Besides, by the behavior of pipes, the commands can be divide into 4 types:

1. **No-Pipe** This kind of commands, such as "ls -l", is the simplest form. The command is executed by child process forked by the parent and the result of the command will be written by a tempory pipe. The parent process reads the pipe and writes to client later.

2. **Pipe-In** These commands output as the "No-Pipe" ones, but input by some pipe written by some command that executes earlier. When every command is going to execute, the program will check "pipe-list" if there is a pipe needs to be read at this command. Then, this command could be "Pipe-In" command or "Pipe-In-Out" command. After this pipe is

determined, it will be "dup2" to the "stdin" of the child process which executes the command.

3. **Pipe-Out**[2] The result of commands in this kind need to be written in some pipe which will be read later, instead of written in "stdout". After these commands are parsed, the program will create a pipe and "dup2" the pipe to "stdout". After the child process starts to execute commands, the parent process will go on read new commands from clients without waiting for the child. Then, the pipe written by the child is added to "pipe-list" except there is already a command with the same pipe out target, and the target of the pipe is decided by the input command. After the child finishes its job, the pipe cannot be closed because there could be some other commands need to be pipe out the same target. Then, the pipe has to be written again. Only before the target command is executed, the program will wait for all the processes output to the pipe. After they finish, the target command can read from the pipe.

4. **Pipe-In-Out** This kind of commands has features decribed in both "Pipe-In" and "Pipe-Out".

Process Control

This program is started by a mater process with the master socket. After it accepts from a client, it forks a child called slave socket to handle the client totally. And the slave socket forks their child if the client want to execute some command.

Environment Variables[3]

The environment variables in the system can be accessed by 2 special commands in the program. The "setenv" can set any variable you want whether it exists or not. And the other "printenv" command can print any environment variable specified by the client. After receiving such commands, the program call "setenv" and "printenv" functions provided by system library and output to the client.

[2]The result of two "Pipe-Out" commands with the same pipe target could be interleave because of the difference of running time of the commands. However, if the parent process always waits for the completion of the child process, the result will be in order.

[3]These two commands and "exit" command is out of the pipe target count. It means that they are not input or output to a pipe.

Constant Definition

- The max number of arguments in a command is set to 20. It means that the number of words in command sent by the client can not exceed 20.
- The buffer size of the program is all set to 1024. It means that "pipe-list" array has the largest entries with 1024. Therefore, there can be 1024 different pipes at the same time.
- The queue length of the master socket is set to 5.

Redirection ">"

The redirection command ">" occurs only when the command is "Pipe-Out" or "Pipe-In-Out" type. The "parse" function maintain the information such as redirection file name or no redirection at all. Then, when the parent process has to read from pipe which is written result of execution by the child process, the parent writes result to a file pointer called "filefd", and calls "fprintf", "fopen" and "fclose" functions to do file operations.

4 Note

- This is the first time for me to write a C program on Unix.
- This is my first time to debug with "gdb"[4].
- There are some mysteries during my implementation. Unix programming interests me, and many bugs confuse me a lot. However, some of the bugs are found out and fixed; some "disappear" automatically!
- The development platform is powered by "Gentoo Linux", "gcc/gdb", and "vi/screen".
- This report is documented by LaTeX.

[4]Console mode.