Project 1
The Voyeur Shell (2.0)

(with apologies to David Beazely)

CMSC 32102

Version 1.2
April 17, 2003

1 Introduction

The Department of Homeland Security launched a research program in the Winter of 2003 to develop a so-called "*Voyeur Shell*," which allows John Ashcroft to keep a lustful eye on everyone's Unix commands. While happy with the resulting prototypes, there is a desire to track additional information. Furthermore, the **vsh** prototypes were written in C and suffered from buffer-overrun vulnerabilities. Therefore, a new call has gone out to develop an enhanced implementation of **vsh** in a safe language.

Your task is to write an implementation of the **vsh** in Concurrent ML. The **vsh** is a simple Unix shell that allows a third party to remotely monitor shell commands. Your shell will have to support the usual shell features of I/O redirection, pipes, and background processes. But it also has to provide this clandestine monitoring feature.

2 Description

The input to the **vsh** is a sequence of commands, each provided on a separate line of input text typed interactively at the keyboard: The **vsh** supports the following command syntax:

Command
::= *Program* (**|** *Command*)^{opt}
| *Program* **&**^{opt}
| **`exit`**

Program
::= *Path Args*^{opt} (**<** *Path*)^{opt} (**>** *Path*)^{opt}

where a *Path* has the following syntax:

Path
::= **/**^{opt} *Filename* (**/** *Filename*)*

Filenames are non-empty sequences of letters, digits, or one of "**.**", "**-**", "**+**", "**=**", "**@**", or "_."

2.1 Background jobs

When a program runs, it normally blocks you from performing any other operations until it has completed. However, you can put a program into the background using the "**&**" operator. For example:

```
progname args &
```

Detaches the program `progname` and runs it in the background. Control is immediately returned to the command shell where additional commands can be executed. Background jobs should continue to run even if you quit the shell before they have finished.

2.2 I/O redirection

In addition to the above commands, your shell must support I/O redirection. I/O redirection is specified using the "**<**" and "**>**" operators at the end of a command line. For example, the command

```
progname args >file.out
```

directs the standard output of `progname` to the file `file.out` and

```
progname args <file.in
```

uses the contents of the file `file.in` as the standard input to program `progname`. Both input and output redirection may be specified for a single command so your shell will have to check for both.

2.3 Pipes

Your shell also needs to support pipes. A pipe is nothing more than a way of hooking up the standard output of one program to the standard input to another. A pipe is indicated using the "**|**" operator as follows:

```
progname1 args | progname2 args
```

Pipes the output of program `progname1` to the input of program `progname2`. For example:

```
vsh % ls | wc
      5      5    28
vsh % foo < infile | bar > outfile
```

Note that attempting to redirect a programs output and also pipe it to another program is not supported. For example, the following produces an error message:

```
vsh % ls > outfile | wc
vsh: illegal command
```

2.4 The Voyeur interface

To allow secret monitoring, the shell should listen for connections on a network port that is equal to the process id (pid) + 10000. Remote users should then be able to watch the shell by simply using the **telnet** command. For example, suppose that a user launches **vsh** and it has a process ID of 11538 (you can use **ps** to find the process ID). The the following command will connect to the **vsh**:

```
% telnet localhost 21538
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
[ Welcome to the vsh shell ]
```

Once a user connects to the **vsh**, the user should see all commands and all output from those commands displayed as long as the telnet connection stays open. Commands should be prefixed with the string "`[vsh]`" and the output from a command `progname` should be prefixed with "`[progname]`." For example, the above command sequence might produce the following trace:

```
[vsh] ls | wc
[ls]bar.h\nbaz.c\nfoo\nfoo.c\nfoo.out
[wc]\t7\t5\t28\n
[vsh]foo < infile | bar > outfile
```

Note that the newlines and tab characters that **wc** uses to format its output are escaped.[1]

Note that the **vsh** should support multiple connections to the voyeur interface.

[1] You can use the function `String.toString` to convert an arbitrary string into a printable string.

3 The project

I recommend that you break the project into four steps. You are required to hand in the result of the first step and the final result.

3.1 Step 1: Design

The first step is to design the architecture of your implementation. For this purpose, I recommend thinking how you want to map the major components of the system onto CML threads. You should also think about how to structure the program into modules. Your design document is due in class on Tuesday April 22.

3.2 Step 2: Command parsing

The second step is to code up a parser for the command language. **Note: you should work on Step 2 in parallel with Step 1.** As a guide, I will supply a scanner that takes a line of input and returns a list of tokens, where a token has the following type:

```
datatype token
  = Path of string
  | Pipe
  | Lt
  | Gt
  | Amp
```

You should write a command-parser module with the following signature:

```
signature PARSER =
  sig
    exception SyntaxError
    val parseLine : string -> Command.cmd
  end
```

The `parseLine` function can be implemented as

```
fun parseLine s = parse (Scanner.scanLine s)
```

where `parse` takes a list of tokens and returns a parsed command. I recommend using a simple recursive descent parser to implement the `parse` function.

3.3 Step 3: Command processing

Once you have a parser for the shell commands, you should implement the shell interpreter. Start with basic command execution and then add background jobs, redirection, and pipes.

3.4 Step 4: Voyeur interface

Once you have the basic shell working, add the voyeur mechanism. The complete project is due on Thursday, May 8th.

4 Document history

1.2 (4/17/03) Changed syntax of file names and the PARSER signature to agree with the sample implementation.

1.1 (4/15/03) Fixed spelling errors and the example. Also added clarification on combining output redirection and pipes.

1.0 (4/15/03) Initial release.