Umeå University
Department of Computing Science

System Level Programming 7.5 p
5DV088

Minimal Shell - MISH

Submitted	2016-10-07
Author:	Lorenz Gerber (`dv15lgr@cs.umu.se` `lozger03@student.umu.se`)
Instructor:	Mikael Ränner / Filip Åberg / Jonathan Westin / Mattias Åsander

Contents

1 Problem Description

The aim of this laboration was to develop a minimal unix shell. The shell had to implement input and output stream redirection, stream piping between commands and two internal commands, ‘cd’ and ‘echo’. The main techniques to be used for implementing the shell were ‘pipes’, ‘forking’ and execution of programs by the ‘exec’ function family. Further, the interrupt signal (SIGINT) had to be reassigned to a function that allows to stop all child processes and then returns control back to the shell prompt. A function for parsing the shell input was provided.

2 Compilation and Invocation

A makefile is provided for compiling the code: ‘make all’. It was tested on clang 800.0.38 (OSX 10.11.6), gcc 4.9.2 (Debian) and gcc 5.4.0 (Ubuntu). Some issues where encountered regarding functionality defined in the POSIX standard. A #define was required in the files that contain the respective functions. Another solution would have been to compile with the -std=gnu11 flag. Makefile also provides a function for cleaning up: ‘make clean’.

3 Detailed Usage Description

‘mish’ does not take any command line argument. The shell can be aborted by sending EOF to the prompt (Ctrl-D). SIGINT (Ctrl-C) is ignored in the prompt main loop and assigned to a custom function during execution of external commands. The custom implementation stops all started child processes and returns back to the command prompt. Child pid’s are stored in a global integer arrray.

4 Algorithm Description

Below follows a description of the function that executes external commands and takes care of piping and redirection. Here, it was chosen to implement a solution where pipes are created one by one while iterating over the commands to be piped. Another solution would have been to create all pipes in advance and then just spawn processes one by one while solving the pipe routing. The current implementation always keeps the stream end for the next command ’remembered’ in the parent process. On the second last command, the iteration over commands is abondend and as the last command does not need another pipe.

```
loop over all but the last commands
    create a pipe
    spawn a child process

    Child process code:
        if requested
            setup redirect
        else
            close pipe read_end
        if there is a remembered file descriptor from the last iteration
```

```
            duplicate it to STDIN and close the remembered file descriptor
        duplicate the pipes write end to STDOUT and close the pipes write end
        run command

    Parent process code:
        if there is a remembered file descriptor from the last iteration,
            close it.
        remember the the read end of the pipe
        close the pipes write end

spawn a child process for last command

Child process code:
    if there is only one command and this command has
        to redirect input from a file
        then redirect in_file file descriptor to STDIN

    if redirect of output to a file is requested
        redirect out_file file descriptor to STDOUT

    redirect pipe read_end to STDIN

    clean up

    run command

Parent process code:
    wait for child processes to report finished
    cleaning up
```

5 System Description

5.1 General structure

The general structure of the implementation is first documented by a call graph in figure 1. As it can be seen, a rather flat organization was chose where most commands are called directly by the main process.

5.2 Piping and Redirection

The basic structure for starting commands in a shell is:

1. open a pipe to allow inter process communication
2. fork a child processs
3. execute command in child process
4. monitor termination of the child process in the parent process

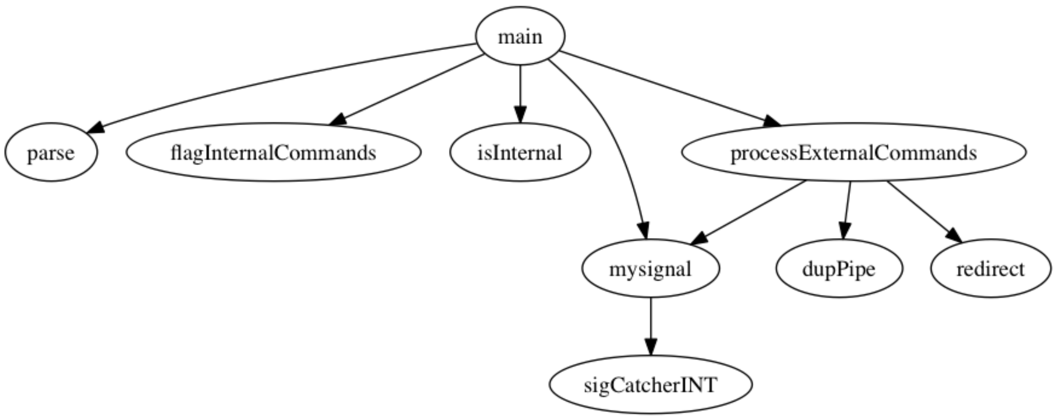

Figure 1: *This figure shows the call graph of mish.*

Below follows a more detailed account for the chosen implementation of the piping and redirection functionality. Figure 2 exemplifies the sequence of commands for the piping setup with three commands, hence two pipes.

6 Known Limits

Currently the number of command commands to be processed on one command line entry is set to 8.

7 Testing

7.1 Debugging

Debugging was found to be rather difficult with the child processes. Not all versions of GDB allow switching to the child process. The default GDB version available on my home setup (Ubuntu 16.04) didn't seem to react on the setting within GDB that should allow switching to the child process. Neither did it work on a custom installed GCC/GDB setup from Homebrew on a OSX 10.11.6. Hence the obvious available solution for some insight were trace printouts to stderr.

7.2 Testing for closing of file descriptors

A number of trace print statements of active file descriptors was setup to monitor whether a long chain of piped commands does not result in the increase of lowest availble file descriptors value.

7.3 Testing for correct finishing of child processes

Process control using the ‘ps’ command was used to monitor the correct termination of child processes.

References

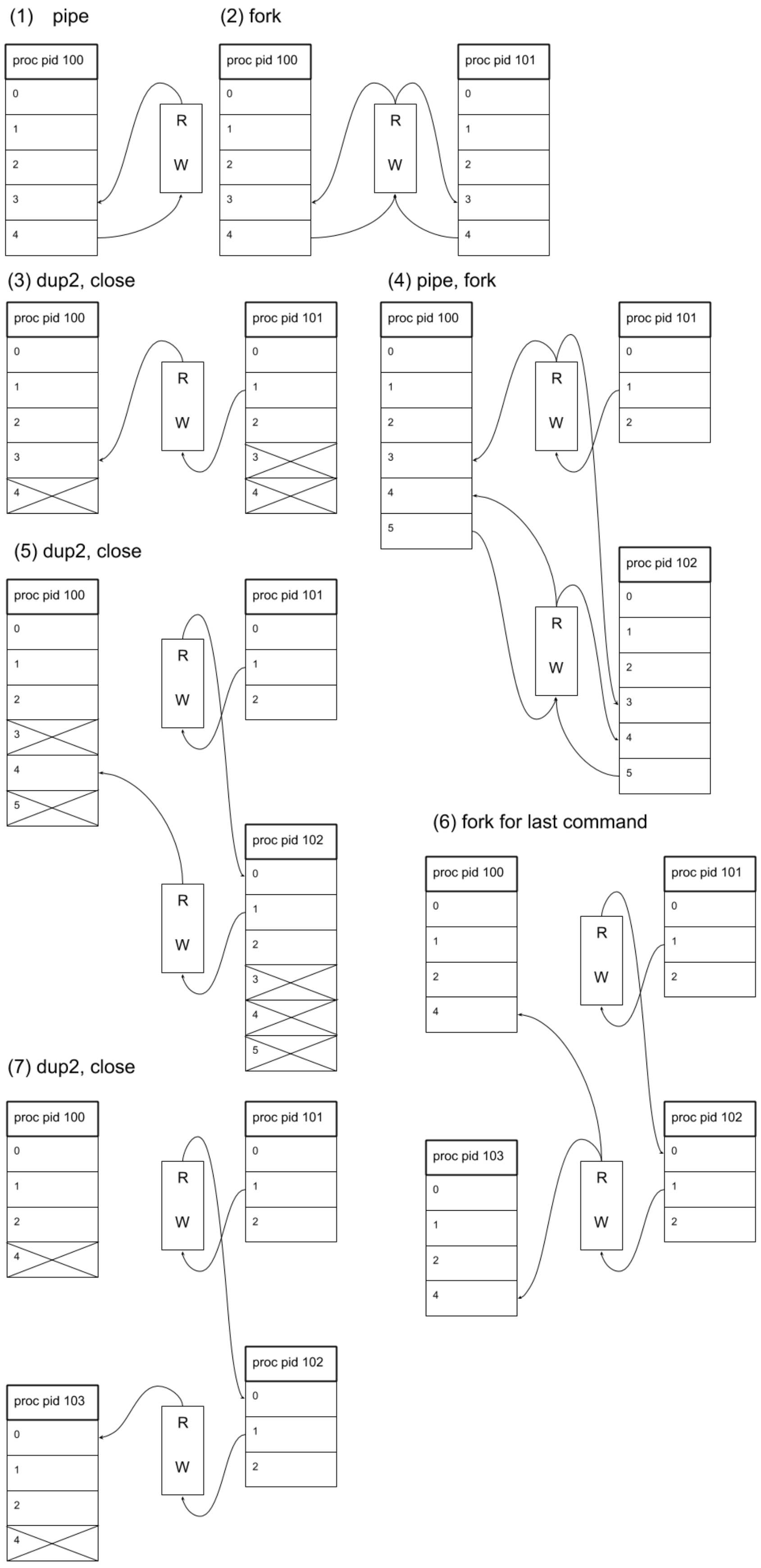

Figure 2: *This figure shows the sequence of commands for the piping setup, here with three commands, hence two pipes.*