Assignment 1 Internet Programming

Marc Went (2507013) and Ferry Avis (1945653)

September 16, 2015

1 Programming the shell

1.1 mysh1

The structure of the program follows the description of the assignment and will thus not be repeated.

The only problem encountered was at implementing the `exit` command. Before executing each command by `execlp`, the `strcmp` function is used to check whether the exit command is typed in and the program thus needs to exit. It appeared that the new line character, that is typed in by the user to send the request of the program, is included in the input. A possible solution is to include the new line character in the `strcmp` function, but an other solution was chosen instead. All white space characters, i.e. spaces, tabs and new lines, are are stripped from the command. Those characters are never part of the command and can safely be removed if they are entered by accident.

1.2 mysh2

To implement this program, the `execvp` system call is chosen because it allows multiple parameters to be passed as a single (dynamically) allocated array while taking the PATH variable into account. The user input is split into an array by splitting on a space. The `strtok` system call is used repeatedly to obtain each chunk of the split input and added to an array.

1.3 mysh3

In the following question A from section 1.2 of the assignment will be answered. The shell needs to create a process for each command in the input, i.e. for `ls /tmp | wc -l` two processes need to be created. Only one pipe is needed per pair of commands in the input. The shell needs to wait for accepting the next command after the last command in the input is finished.

If the shell would need to support chained pipes, we think another pipe must be created in the code. A command that is not the first or the last in line, must read from a pipe and write to it. It is either not possible to do this with one pipe or causes mangled input and output. We think the problem is different from communicating

between processes that run simultaneously, which is not possible to do with one pipe, as pipes are only read after the previous process is finished.

The structure of the program is as follows. First, the input is split on the — character and the pipe is created. Then, the custom function `execute` is called that trims the first command that needs to be executed, exits the program if requested and creates a process to execute the requested program. The standard output is changed to the write end of the pipe. After the program is finished, it is checked whether the input contained a pipe. If so, the function `execute` is called again from the parent. By using this recursive call, duplicate code for checking whether `exit` is entered and forking is avoided. This structure is also better suited if in future chained pipes would be implemented.

Question B of section 1.2 asks whether the shell program could be implemented using threads instead of processes. The answer is no. The functions from the `exec` family replace the complete contents of the program. No code from the shell program after this call will be executed. After executing one requested program, no additional piped commands can be executed, nor will be asked for new input.

To answer question C of section 1.2: the `cd` command cannot be used. This is a shell command, just like exit. The command `cd` is not implemented. It can be seen as memory or string that will be prepended to each requested program that is not called by using the absolute path of the program.