Report

# Description

The simple shell program is a small shell that implements a subset of the Bash shell. The program continually prompts the user to enter a command line to be executed. After parsing it, it executes every command in the command line as separate processes and redirects their outputs and inputs if the commands are given as a series of piped commands. The simple shell also supports redirection for any piped command. The number of piped commands that can be handled by the simple shell is arbitrary. A command line with a terminating ampersand is considered a background job and is executed without interrupting the input of more commands in the shell.

# Techniques

The shell follows the process described in the next paragraphs to execute the input command line.

The shell is a loop that displays conditionally a prompt, reads an input line, parses it and executes the parsed commands. The crucial part of this loop is the command line parsing; this is done by the program in a few steps:

First, it looks for a terminating ampersand to determine if the commands will execute in the background. Then, it splits the command line using the pipe character “|” as the delimiter. The splitting generates an array with all the commands that are going to be executed and pipelined.

The next step is to go through every single command and parse the file redirections by looking for the “>” and “<” characters, the filenames and redirection types are saved in an array. The program has no restriction on the number of redirections allowed but it only considers the last redirection of a given type as the redirection target. The following step is to use the command without the redirections and split it by spaces, this will result in another array, this time called the arguments array, since it contains the command to be executed as the first element and the arguments as the remaining elements. This series of steps results in an array of redirections and an array of arguments for each piped command. The next step is to execute the commands found.

The command execution is started by creating the required pipes, connecting them in a pipeline and feeding a special function with the redirections, the pipes and the arguments. The function then creates a new process and handles the standard input and output redirection of it using the pipe and the file redirection information. It then tries to execute the command contained in the arguments array.

After all processes have been forked, the program can either wait for the termination of all forked processes or simply let them run on the background without waiting for them.

In order to avoid having zombie processes a signal handler is installed at the start of the shell execution, this handler waits for the SIGCHLD signal, which is the signal that is generated from a terminating process. By using this handler, the shell can avoid having defunct or zombie processes while the shell is running.