Process exercise

Daniel Hagimont

Daniel.Hagimont@enseeiht.fr

USTH

March 2018

Objectives

The objective of this exercise is to use Unix process management APIs and Unix pipes for implementing a simple shell (such as bash). This shell should allow the execution of binaries (such as ls, pwd, grep or your shell itself) and it should also allow to cascade such executions with pipes. In a further step, this shell can implement the cd command to enable changing the current directory.

Reading the shell input

You are given a module which implement the reading and analysis of the shell input. This module (readcmd.c/readcmd.h/tst.c) provides a **readcmd** function which returns a pointer to a **struct cmdline** data structure. A *struct cmdline* structure has the following fields:

char *err – error message to display, else *null*

char * in – file name for input redirection, else *null*

char *out – file name for output redirection, else *null*

char ***seq – a command line is a sequence of commands. Each command's output is linked to the input of the next command by a pipe. A command is an array of strings (*char* **), whose last item is a null pointer. A sequence is an array of commands (*char* ***), whose last item is a null pointer. When a *struct cmdline* is returned by *readcmd*, *seq[0]* is never *null*. This structure may appear complex, but it is suited to be used with the *execvp* system call.

The *tst.c* program gives an example of use of the *readcmd* module.

Instructions

You have to implement a shell program which will basically rely on the *readcmd* module, and the fork, execvp, pipe and dup2 system calls.

In a second step, you can implement the *cd* command, which cannot be implemented as an external binary as other commands. You can use the *getcwd, chdir* and *getenv* system calls.