https://folk.ntnu.no/michaeng/tdt4186_22/
michael.engel@ntnu.no

Practical exercises
Spring 2022

Practical Exercise 3

Unix Shell

Please submit solutions on Blackboard by Tuesday, 19.4.2022 12:00h

In this assignment, you will combine a number of topics you have learnt about in this course so far – such as process and memory management, I/O handling and redirection.

Your task is to implement a simple Unix shell called the "famous little unix shell" *flush*. *flush* should be an interactive shell with minimal functionality that is nevertheless doing some useful things.

You should build *flush* in several steps as detailed below.

3.1 Basic functionality

flush prompts the user for input by printing the current working directory followed by a colon (:) character. The entered text is split into command name and arguments. Arguments are separated from each other and from the command name by space (ASCII 0x20) or tab (0x09) characters.

Implement the command line parsing, create a new process (using `fork(2)`) and execute the entered command with the given parameters (using a variant of `exec(3)`).

Hint: For this task, you do **not** need to handle parameters with quotes or backslash to escape whitespace characters (e.g. `ls /my/home\ dir` or `ls "/my/home dir"`). You can also ignore search paths for commands and always assume that a complete absolute (starting with /) or relative path is given for the command name.

flush waits for the termination of the started (foreground) process (using `waitpid(2)`) and prints the exit status of the command along with the command line like this:

```
/home/user/shelldev: /bin/echo test
test
Exit status [/bin/echo test] = 0
```

After printing the status, the shell is ready to accept a new command.

flush terminates when control-D (ASCII 0x04) is entered on the command line.

3.2 Changing directories

Implement the `cd` command to change directories (using `chdir(2)`). You can ignore a `cd` command without a parameter (a real Unix shell would change the path to the user's home directory).

Remember that `cd` has to be an internal shell command (think about why this needs to be internal).

3.3 I/O redirection

Implement simple I/O redirection for `stdin` and `stdout` when this is indicated on the command line using the < or > characters followed by a file name.

You can simplify this task by only accepting input and output redirections as the last parameters of a command line (excluding the "&" character for background processes, see below), e.g.:

```
/home/user/shell: ls > /tmp/foo
Exit status: 0
/home/user/shell: head -1 < /etc/passwd
root:*:0:0:System Administrator:/root:/bin/sh
Exit status: 0
/home/user/shell: head -1 < /etc/passwd > /tmp/foo2
Exit status: 0
```

3.4 Background tasks

If a command line ends with the character "&", the entered command is to be executed as a background process, i.e., *flush* does not wait for the termination of the process but directly prompts the user to input a new command.

Before printing a new prompt, *flush* should collect all background processes that have terminated (zombies) and print their exit status like for the foreground processes of the first subtask. For this, you need to store the PID of each created background process and its command line, e.g., in a linked list.

3.5 Background task status

Implement an internal shell command `jobs` to print all running background processes (PID and command line, one line per process).

3.6 Optional bonus task: Pipelines

Implement command pipelines using the `pipe(2)` syscall, e.g.:

```
/home/user/shell: ls -l | grep root | cat > /tmp/file
```