Grading script for CS2106 AY2122S1 Lab 2

Quickstart

1. Ensure that you are on the xcne servers
2. Place your `myshell.c` file in the same folder as this README
3. Run `./grade_single.sh`

After a few minutes, the console will output a summary of your grades for each (non-bonus) component of this lab, along with specific comments for each testcase if your code did not pass it.

To grade the bonus component, you should run `./grade_single.sh 1`. This will run tests for only the bonus component.

Explanation of Grading Script

The writers of the grading script will like to thank Haowei for the codebase which was originally used for grading in AY2021S1

All testcases are stored in the `testcases` folder, with a README within the folder to explain the purpose of each testcase. Each test is a bash script, and the system calls you used in your code are tracked using `strace` and logged into a `.trace` file. For the chained and redirection commands, temporary files may be used.

Some testcases make use of executables that we have specifically written for testing purposes; these are compiled from the various `.c` files in this folder.

With the `.trace` files, we then use `Node.js` to read through the output of each file to ensure that they conform to our specifications as mentioned in the lab document. These files are found in the `./grader` folder. You can also manually look through the `.trace` files to see how your code behaves under each test.

NOTE: Due to the stochastic nature of operating systems, it is not possible for the grading script to deterministically evaluate your code. As such, we have run the grading script multiple times and awarded you the highest grade among them.

Breakdown of testcases

Testcases are broken down in the following format `NC`, where `N` is the corresponding exercise number, and `C` is the component being tested according to the order below:

ex1: (a) foreground program, (b) background program, (c) info\ **ex2**: (a) wait, (b) terminate, (c) chained programs, (d) info, (e) quit\ **ex3**: (a) one redirection, (b) two redirections, (c) three redirections, (d) redirection for chained programs\ **ex4**: (a) \<Ctrl-Z> + fg (b) \<Ctrl-C>

Exercise 1 (3 marks)

1a (1.2 marks)

1. Test `{program}`
    - Check if program is run
2. Test `{program} (args...)`
    - Check if program is run with correct args
3. Check that program waits for ongoing command before running another command
4. Check program runs properly with multiple args

1b (0.9 marks)

1. Test `{program} &` and `{program} (args...) &`
    - Check if program is run in background (with correct args if present)
2. Check that program runs next command without waiting for background command
3. Tests non-existing program files for `&` and non-`&` versions
    - Check that a process is not spawned
    - Check for correct output

1c (0.9 marks)

1. Test `info` for background commands
    - Check if `info` gives correct status for currently running program
    - Check if `info` gives correct status for exited program
2. Test `info` for foreground commands
    - Check if `info` gives correct nonzero exit codes
    - Check if `info` gives correct status and exit code when a running process exits
3. Test `info` for commands with improper args
    - Check that processes still spawn, but with appropriate exit code

Exercise 2 (2 marks)

2a (0.3 marks)

1. Test `wait {PID}`
    - Check if running programs are waited for
2. Check that exited programs are not waited for again

2b (0.3 marks)

1. Test `terminate {PID}`
    - Check if running programs are sent SIGTERM
2. Test `terminate {PID}` for multiple background commands
    - Check that already killed programs are not sent SIGTERM

2c (0.6 marks)

1. Test `{program1} && {program2}`
    - Check that program2 runs after program1
    - Check that both `wait(program1)` and `wait(program2)` are done
2. Test `{program1} (args...) && {program2} (args...)`
    - Check if both programs run with the correct args
    - Check that program2 runs after program1
    - Check that both `wait(program1)` and `wait(program2)` are done
3. Test a long chained command of 9 commands in total, with and without args
    - Check that the programs run in order
    - Check that the programs are waited in order
4. Test a long chained command of 9 commands in total, with and without args. One of the middle commands will return non-zero exit status.
    - Check that the rest of the chain is not executed
5. Test a long chained command of 9 commands in total, with and without args. One of the middle commands will point to a non-existing program.
    - Check that the rest of the chain is not executed, and a child process is not spawned for the non-existing program

2d (0.5 marks)

1. Test `info` for chained command without errors
   - Check that all programs have the correct status and exit code
2. Test `info` for chained command with errors
   - Check that `info` only lists programs without errors
   - Check that `info` does not list extra programs
3. Test `info` for terminating
   - Check that `info` updates with the correct "Terminating" status

2e (0.3 marks)

1. Test `quit` with a number of (exited) foreground + background commands
   - Check if all running processes (and only these processes) are sent the SIGTERM signal
   - Check if all running processes (and only these processes) are waited for before the shell exits
2. Test `quit` with a mix of exited and running background commands

Exercise 3 (2 marks)

3a (0.6 marks)

1. Test input redirection in foreground/background
   - Test `{program} (arg) < {file} (&)` with existing and non-existing file
2. Test output redirection in foreground/background
   - Test `{program} (arg) > {file} (&)` with existing and non-existing file, check permissions of created file
3. Test error redirection in foreground/background
   - Test `{program} (arg) 2> {file} (&)` with existing and non-existing file, check permissions of created file

3b (0.6 marks)

1. Test in/out redirections in foreground/background
   - Test `{program} (arg) < {file1} > {file2} (&)`
2. Test in/err redirections in foreground/background
   - Test `{program} (arg) < {file1} 2> {file2} (&)`
3. Test out/err redirections in foreground/background
   - Test `{program} (arg) > {file1} 2> {file2} (&)`

3c (0.2 marks)

1. Test in/out/err redirection in foreground/background
   - Test `{program} < {file1} > {file2} 2> {file3} (&)`

3d (0.6 marks)

1. Test a long chained command of 9 commands in total, with and without args, with and without redirection
2. Test a long chained command of 9 commands in total, with and without args, with and without redirection, one of the middle commands will have a non-existing input file.
   - Check that the rest of the chain is not executed, and a child process is not spawned for the program with the non-existing input file
3. Test a long chained command of 9 commands in total, with and without args, with and without redirection, one of the middle commands will exit with non-zero exit status + error output redirected into a file.
   - Check that the rest of the chain is not executed, and the error output is correctly captured into the file

Exercise 4 (BONUS - 2 marks)

4a (1 mark)

1. Test if SIGTSTP signal is sent correctly, test if fg works
2. Test if SIGTSTP (and fg) works with redirection
3. Test if SIGTSTP (and fg) works with chained commands

4b (1 mark)

1. Test if SIGINT signal is sent correctly
2. Test if SIGINT works with redirection
3. Test if SIGINT works with chained commands