OSC Project 1

Dieks Scholten - s1098110

September 2024

1 Implementation

1.1 Code traversal of expressions

First, the code checks if the first part of the first command is cd or exit. If that is the case it executes that command and proceeds.

Then the program creates a vector of pipes for each command in the expression. In a for-loop children are created and children PIDs are stored. Each child is linked to the next child or the std-in/std-out with pipes. The child gives the command to a function called execute_composite_command. This function creates a new child for each & and executes all these separate atomic commands simultaneously.

The systemcalls used are:

1. exit, to exit from child processes when the command has executed or exit the parent process when the *exit* command is called.
2. chdir, to change the current directory
3. fork, to create child processes
4. open, to create a file descriptor to set as the std-in or std-out if needed
5. dup2, to set the standard-in or -out to the ends of pipes or to file descriptors
6. close, to close pipes and file descriptors
7. abort, to abort child processes if there is an error waitpid, to wait as a parent on their child processes

1.2 Pipes and &'s

I choose to allow for &'s in pipes commands but the standard-in of all programs is the read-side of the pipe from the previous command. Similarly the standard-out of all programs is the write-side of the pipe to the next part of the pipe. This is a little different from the Linux implementation. In Linux the command: date

| tail -c 5 & date | tail -c 6 outputs "CEST" and " CEST". In my implementation the output is " CEST". This is since the output of the tail -c 5 is added to the output of date. Since the programs are run simultaneously this adds indeterminacy but this means the command should be changed. The upside of this implementation is that it is more intuitive and easier to implement.

1.3 exit and cd in Pipes

Here I tried to stick to the Linux implementation. Linux only executes exit or cd when placed in the last part of piped commands where there is atleast 1 command preceding it. This is done before executed before the program starts on the rest of the program. I decided to do it like this since all commands are handled in seperate threads so it would be unclear when the commands would be executed eitherway. For this reason there is no reason to not execute them before. This made implementation easier.

2 Tests

1. **Testing exit and cd.**

Command	Expected	Result
exit	exited	exited
cd ..	current dir one up	current dir one up
cd build	current dir to ../build	current dir to ../build

2. **Testing basic commands.**

Command	Expected	Result
ls	[files in current dir]	[files in current dir]
date	gives the date	gives the date

3. **Testing simple piping.**

Command	Expected	Result
date | tail -c 5	CEST	CEST
date | tail -c 5 | head -c 3	CES	CES

   This is to test one and multiple pipes

4. **Testing &'s.**

Command	Expected	Result
date & date	the date twice	the date twice

5. **Testing pipes with &'s.**

Command	Expected	Result
date | tail -c 5 & date | tail -c 6	\nCEST	\nCEST
date | tail -c 5 & date | tail -c 10	CEST\nCEST	CEST\nCEST
date | tail -c 5 & exit & date | tail -c 6	An error from the exit (the input to exit is not valid) and \nCEST	No such file or directory \nCEST
date | tail -c 5 & date | tail -c 6 & exit	'CESTCEST' and at sometime during the commands exit the terminal	exits the terminal immediately (is not really different from the user's side).
date | tail -c 5 & date | tail -c 6 & cd ..	change the current directory to one above and print "CEST\nCEST "	change the current directory to one above and print "CEST\nCEST "
date | tail -c 5 & date | tail -c 6 & cd .. & cd build	change the current directory to one above and back to ../build and print "CEST\nCEST "	change the current directory to one above and back to ../build and print "CEST\nCEST "

6. **Tesing infinite outputs (with piping).**

Command	Expected	Result
yes	Infinite amount of y's	Infinite amount of y's
yes | tr y n	Infinite amount of n's	Infinite amount of n's
yes | date	The date prompt for next command.	The date prompt for next command.

7. **Testing file input and output.** (I added a test-file "test.txt"):

Command	Expected	Result
date ¿ test.txt	Wrote the date to test.txt and replaced the old content.	Wrote the date to test.txt and replaced the old content.
tail -c 5 ¡ test.txt	CEST	CEST

For all these tests, the shell provided the expected results. The tests are quite comprehensive, despite that they do not proof the shell is bug/error free.

Infinite Buffer

As far as my tests conclude my shell implementation does not require an infinite buffer. I ran a test: yes | date. This executes fine which leads me to believe that my implementation does not suffer from the Infinite buffer problem.