NATHAN-ANDRE SHELL (NASH)

PROJECT 1

October 21, 2019

Nathan Nichols && Andre Kurait

EECS 678

Operating Systems

Contents

Features & Implementation 2
Testing 5

```
bash> ./nash

 ___________________
< Welcome to NAsh >
 -------------------
          \    ^__^
           \   (oo)\_______
               (__)\       )\/
                   ||----w |
                   ||     ||
NAsh@/home/username/NAsh-Shell>
```

Features & Implementation

1. Run executables without arguments (10)
   This is the feature that allows us to execute any executable by just specifying the absolute directory to it. For example, if you wanted to execute the /bin/ls executeable, you would just enter that as a command. We implemented this using the execlp function with the arguments as the file location followed by NULL for termination.

2. Run executables with arguments (10)
   This is similar to above except that we can add one or multiple arguments. For example, if you wanted to execute the /bin/ls executeable with the -a argument, you would just enter /bin/ls -a as a command. We implemented this using split method we created in main.cpp which uses a stringstream with getline to split the command across the ' ' delimiter and passing the tokens to the execlp with NULL termination.

3. set for HOME and PATH work properly (5) We implemented set for HOME and PATH (and actually any environment variable desired) by the putenv method in the parent process. This adds it to the process's environment which is duplicated to the child

process which carry out the execution.

4. exit and quit work properly (5)
   Typing exit or quit as a command toggle a boolean flag in the shell object indicating that it is not active which since we take in commands while our shell is active, our input is triggered to end and "Exiting..." is displayed

5. cd (with and without arguments) works properly (5)
   cd with no arguments attempts to change directory to whatever is in the HOME environoment variable. If the HOME is not set then "NAsh: cd: HOME not set" is displayed. With an argument (the path given, or HOME if not specified) NAsh displayes an error if the directory is not found. If the directory begins with a / then it is treated as an absolute path and otherwise a relative path. This was implemented using getenv("HOME") to retrieve the HOME environment variable and then using chdir to change directory to either the HOME or the specified directory.

6. PATH works properly. Give error messages when the executable is not found (10)
   The PATH environment variable allows us to execute executables without specifying the full absolute path of the PATH in the command. Our shell searches all directories in the PATH (separated by :) and executes the first match found. If no match is found, NAsh displayes "NAsh: [command]: No such file or directory". We implemented this using execlp to search the path environment variable that the process has.

7. Child processes inherit the environment (5) Our child process and executable will inherit the environment variables that were set using the set command or inherited from the process that started NAsh. Because our set used putenv and we are executing our child process with execlp, we were given this feature for free from implementing the other features

8. Allow background/foreground execution (&) (5) Foreground execution is supported as

expected with traditional commands with concurrent printing to output if not piped to another command. Background execution (command ends in &) also support concurrent printing and allow other processes to be started. Background processes are implemented by allowing the process to become a zombie process.

9. Printing/reporting of background processes, (including the jobs command) (10) Background process starts with "[JOBID] PID running in background" printed and end with "[JOBID] PID finished COMMAND" with PID being the OS assigned PID and JOBID being the NAsh assigned id starting at 1 and counting up without recycling. This is achieved by using a map to store all the JobIDs, PIDs, and Commands and checking which background process ended every-time a SigChild exception is thrown to do the proper printing.

10. Allow file redirection (> and <) (5) Any command superseded by either/both > for file output and < for file input followed by the filename. This was implemented by parsing the tokens and then feeding the file input into a pipe which was fed into the executable for input or creating the file then piping the output of the executable into a pipe and then redirecting that into the file. Likewise with bash, if you pipe ls into a file in the current directory, the filename itself will appear in the file itself.

11. Allow (1) pipe (|) (10) Two commands can be joined by | which feeds the output of the first command into the input of the second command. This was implemented by detecting when there was a pipe following a command and then passing in the read side of the pipe into the next command to duplicate as input.

12. Supports reading commands from prompt and from file (10)
    NAsh can support reading commands from prompt as expected or from a file using the "< filename" in bash to feed into it. The file MUST end in a newline to avoid unexpected behavior. This was implemented using standard input in main which was identical for both prompt and the file, but detecting for input end of file to detect the

end of commands read in from a file.

13. Report (10)
    This is the report. It was implemented in LaTeX on the overleaf platform.

14. Support multiple pipes in one command. (10) Piped commands can be piped into other commands example: "ls | grep main | grep .o". This feeds the output of the first command into the second command, and that output into the third command. This was implemented by tokenizing based off of the | and then keeping track of the input and output pipes to keep feeding in until the last command which printed using standard output. This allows us to have arbitrarily long piped chains.

15. kill command delivers signals to background processes. (5)
    The kill command has the format: kill SIGNUM JOBID, where SIGNUM is an integer specifying the signal number, and JOBID is an integer that specifies the job that should receive the signal. This was implemented by using the map that stores all the background tasks to find the PID associated with the respective JOBID and then kill using the kill function in C++.

16. working directory printed on command prompt (EXTRA)
    NAsh prints the working directory on the prompt for commands in the format "NAsh@CWD>". This was implemented with the getcwd method in C++ to print the current working directory each time the prompt was printed.

Testing

1. Run executable without arguments (10)

```
NAsh@/home/n545n545/NAsh-Shell> ls
commands.txt
main.cpp
main.o
```

```
Makefile
nash
NAsh.cpp
NAsh.h
NAsh.o
README.md
NAsh@/home/n545n545/NAsh-Shell>
```

2. Run executable with arguments (10)

```
NAsh@/home/n545n545/NAsh-Shell> cat commands.txt
yes &
yes &
yes &
jobs
exit
NAsh@/home/n545n545/NAsh-Shell>
```

3. set for HOME and PATH work properly (5) (Note: env must be in PATH in this case)

```
NAsh@/home/n545n545/NAsh-Shell> set HOME=/usr/
NAsh@/home/n545n545/NAsh-Shell> set PATH=/usr/bin/
NAsh@/home/n545n545/NAsh-Shell> env
..........
HOME=/usr/
PATH=/usr/bin/
..........
NAsh@/home/n545n545/NAsh-Shell>
```

4. exit and quit work properly (5)

```
NAsh@/home/n545n545/NAsh-Shell> exit
```

```
Exiting...

NAsh@/home/n545n545/NAsh-Shell> quit

Exiting...
```

5. cd (with and without arguments) works properly (5)

```
NAsh@/home/n545n545/NAsh-Shell> cd
NAsh@/home/n545n545> cd /
NAsh@/>
NAsh@/> set HOME=notavaliddir
NAsh@/> cd
NAsh: cd: notavaliddir: No such file or directory
NAsh@/>
```

6. PATH works properly. Give error messages when the executable is not found (10)

```
NAsh@/home/n545n545/NAsh-Shell> set PATH=/bin/
NAsh@/home/n545n545/NAsh-Shell> env
NAsh: env: No such file or directory
NAsh@/home/n545n545/NAsh-Shell>
```

7. Child processes inherit the environment (5)

```
NAsh@/home/n545n545/NAsh-Shell> set HELLO=WORLD
NAsh@/home/n545n545/NAsh-Shell> ./nash
 -----------------
< Welcome to NAsh >
 -----------------
        \   ^__^
         \  (oo)\_______
```

```
            (__)\       )\/
                ||----w |
                ||     ||
NAsh@/home/n545n545/NAsh-Shell> env
.....
HELLO=WORLD
NAsh@/home/n545n545/NAsh-Shell>
```

8. Allow background/foreground execution (&) (5)

```
NAsh@//home/n545n545/NAsh-Shell> sleep 5 &
[1] 4443 running in background
NAsh@/home/n545n545/NAsh-Shell> ls
Makefile
NAsh.cpp
NAsh.h
NAsh.o
README.md
commands.txt
main.cpp
main.o
nash
output.txt
quash
NAsh@/home/n545n545/NAsh-Shell>
[1] 4443 finished sleep 5
NAsh@/home/n545n545/NAsh-Shell>
```

9. Printing/reporting of background processes, (including the jobs command) (10)

```
NAsh@/Users/n545n545/Documents/NAsh-Shell> sleep 10 &
[1] 4383 running in background
NAsh@/Users/n545n545/Documents/NAsh-Shell> sleep 10 &
[2] 4384 running in background
```

```
NAsh@/Users/n545n545/Documents/NAsh-Shell> sleep 10 &
[3] 4385 running in background
NAsh@/Users/n545n545/Documents/NAsh-Shell> jobs
[1] 4383 sleep 10
[2] 4384 sleep 10
[3] 4385 sleep 10
NAsh@/Users/n545n545/Documents/NAsh-Shell>
[1] 4383 finished sleep 10
NAsh@/Users/n545n545/Documents/NAsh-Shell>
[2] 4384 finished sleep 10
NAsh@/Users/n545n545/Documents/NAsh-Shell>
[3] 4385 finished sleep 10
NAsh@/Users/n545n545/Documents/NAsh-Shell>
```

10. Allow file redirection (> and <) (5)

```
NAsh@/home/n545n545/NAsh-Shell> ls > output.txt
NAsh@/home/n545n545/NAsh-Shell> cat output.txt
Makefile
NAsh.cpp
NAsh.h
NAsh.o
README.md
commands.txt
main.cpp
main.o
nash
output.txt
quash
NAsh@/home/n545n545/NAsh-Shell> grep main < output.txt
main.cpp
main.o
NAsh@/home/n545n545/NAsh-Shell>
```

11. Allow (1) pipe (|) (10)

```
NAsh@/home/n545n545/NAsh-Shell> ls | grep main
main.cpp
main.o
NAsh@/home/n545n545/NAsh-Shell>
```

12. Supports reading commands from prompt and from file (10)

```
Nathans-iMac-2:NAsh-Shell n545n545$ cat commands.txt
echo "Hello grader"
echo "Please give us 115/100"
quit
Nathans-iMac-2:NAsh-Shell n545n545$ ./nash < commands.txt
 -----------------
< Welcome to NAsh >
 -----------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/
                ||----w |
                ||     ||
NAsh@/Users/n545n545/Documents/NAsh-Shell> "Hello grader"
NAsh@/Users/n545n545/Documents/NAsh-Shell> "Please give us 115/100"
NAsh@/Users/n545n545/Documents/NAsh-Shell>
Exiting...
Nathans-iMac-2:NAsh-Shell n545n545$
```

13. Report (10)
    This LaTeX masterpiece

14. Support multiple pipes in one command. (10)

```
NAsh@/home/n545n545/NAsh-Shell> ls | grep main | grep .o
main.o
NAsh@/home/n545n545/NAsh-Shell>
```

15. kill command delivers signals to background processes. The kill command has the format: kill SIGNUM JOBID, where SIGNUM is an integer specifying the signal number, and JOBID is an integer that specifies the job that should receive the signal. (5)

```
NAsh@/home/n545n545/NAsh-Shell> sleep 100 &
[1] 4396 running in background
NAsh@/home/n545n545/NAsh-Shell> sleep 100 &
[2] 4397 running in background
NAsh@/home/n545n545/NAsh-Shell> jobs
[1] 4396 sleep 100
[2] 4397 sleep 100
NAsh@/home/n545n545/NAsh-Shell> kill 2 1
[1] 4396 finished sleep 100
NAsh@/home/n545n545/NAsh-Shell> jobs
[2] 4397 sleep 100
NAsh@/home/n545n545/NAsh-Shell>
```