Luis Medina

CS 4375

Minishell

a. The program shell.py is able to run **ls, cat, grep** and a number of other commands through the use of the **PATH** environment variable. It appends the given command name to each possible path to attempt to find the corresponding file. Shown in *images 1, 7*

b. The shell stops if the user decides by entering "exit" into the shell. Shown in *image 2*

c. The minishell program makes sure to set the environment variable **PS1** to **$$$$** during its start and prints it to the screen before reading each line through the use of **input()**. Shown more clearly in *images 1 – 3.*

d. By using an ampersand, the minishell is able to recognize multiple commands strung together (no output redirect) and will execute them in the background, immediately prompting the user for the next command. This is achieved through string manipulation and the use of process forking without instructing the parent process to wait. Shown in *images 4, 6.*

e. Not implemented.

f. The minishell is able to recognize other scripts specified anywhere in the read line and runs them. Shown in *image 8.*

g. When the minishell cannot find a given command, it will raise an exception and specify which command was not found. Shown in *image 5.*

h. If a process fails, the program will specify which process ID failed and the code it exited with. Shown in *image 5.*

i. The minishell is able to change directories by using **cd**. It accepts all the usual command parameters. An example shown in *image 6.*

j. By specifying with the character '>', the user can instruct the minishell to redirect output to a specified file. This is achieved by closing the file descriptor for standard out and opening the specified file in its place. I created a method called **redirect_output()** for this section to separate and reuse this functionality. Shown in *image 7.*

```
Disra@DESKTOP-OBU1TKK ~/minishell
$ python shell.py
$$$ ls
Parent: My pid=696.  Child's pid=697
aa-chatbot.py  nw-chatbot.py  shell.py  spinner.py  test.py  testMinishell.shx
Parent: Child 697 terminated with exit code 0
$$$ ls /
Parent: My pid=696.  Child's pid=698
Cygwin-Terminal.ico  Cygwin.ico  cygdrive  etc   lib   sbin  usr
Cygwin.bat           bin         dev       home  proc  tmp   var
Parent: Child 698 terminated with exit code 0
$$$ ls /home
Parent: My pid=696.  Child's pid=699
Disra
Parent: Child 699 terminated with exit code 0
$$$
```

Image 1

```
Disra@DESKTOP-OBU1TKK ~/minishell
$ python shell.py
$$$ exit

Disra@DESKTOP-OBU1TKK ~/minishell
$
```

Image 2

```
Disra@DESKTOP-OBU1TKK ~/minishell
$ python shell.py
$$$
```

Image 3

```
$$$ ls & cat test.txt
Parent: My pid=703.  Child's pid=706
aa-chatbot.py  nw-chatbot.py  shell.py  spinner.py  test.py  test.txt  testMinishell.shx
Parent: Child 706 terminated with exit code 0
Parent: My pid=703.  Child's pid=707
"hello world"
Parent: Child 707 terminated with exit code 0
$$$
```

Image 4

```
Disra@DESKTOP-OBU1TKK ~/minishell
$ shell.py
$$$ hello
Parent: My pid=105.  Child's pid=106
Error: Command not found: hello
Parent: Child 106 terminated with exit code 256
$$$
```

Image 5

```
$$$ ls & cd / & ls
Parent: My pid=703.  Child's pid=709
aa-chatbot.py  nw-chatbot.py  shell.py  spinner.py  test.py  test.txt  testMinishell.shx
Parent: Child 709 terminated with exit code 0
Parent: My pid=703.  Child's pid=710
Parent: My pid=703.  Child's pid=711
Cygwin-Terminal.ico  Cygwin.bat  Cygwin.ico  bin  cygdrive  dev  etc  home  lib  proc  sbin  tmp  usr  var
Parent: Child 711 terminated with exit code 0
$$$
```

Image 6

```
Disra@DESKTOP-OBU1TKK ~/minishell
$ python shell.py
$$$ echo "hello world" > test.txt
Parent: My pid=703.  Child's pid=704
Parent: Child 704 terminated with exit code 0
$$$ cat test.txt
Parent: My pid=703.  Child's pid=705
"hello world"
Parent: Child 705 terminated with exit code 0
$$$
```

Image 7

```
Disra@DESKTOP-OBU1TKK ~/minishell
$ shell.py
$$$ python spinner.py 10000
Parent: My pid=710.  Child's pid=711
10000
Parent: Child 711 terminated with exit code 0
$$$
```

Image 8

Conclusion

To begin with, one thing that really stood out to me was the use of child processes in an endless loop while handling them with proper logic. Previously we had played with the idea of creating one child and managing both it and its parent before they both terminated. In this project, we not only had to make sure the system could handle a number of child processes properly, but then had to expand on that initial configuration to allow for further functionality. Every time I implemented the next feature, such as supporting '&', 'cd' and '>', I had to retest my code to ensure the new feature didn't create unexpected issues with previous sections. As I write code, I tend to keep that foresight in mind, and I am glad I made a habit of it as each iteration went fairly smoothly.

Besides that, I'd like to note that I also tried to keep my code "elegant" as I wrote it. Namely by attempting to separate repeated functionality by creating methods, creating comments and accounting for certain edge cases. Despite being happy with the resulting program, I'm not the most content on the elegance of the code. As this was the first time I worked on such a system, I would have preferred to spend more time reviewing the code and optimizing it as personal preference. Although of course, that comes second to understanding the OS and being able to work with it.

Also, please note that for the screenshots provided, I had the system print more information on the running and terminating processes to help illustrate required functionality. I removed the parts that were not requested in the instructions from the final code.