CSE 344 - Homework 2
1901042601 - Sena Özbelen

We are asked to implement a terminal emulator which is capable of executing commands with “|” and “<,>”.

Firstly, I initialized an array of char with the size 2500. In “read_input” function, it gets the input from the user and stores it to this array. “split_commands” function tokenizes the input by using “|” as a token. The number of commands separated by pipes is the return value of this function. Using this number, it executes a for loop to execute each command one by one. Before execution, it checks whether the command is equal to “:q”. If it is, then it terminates the program. Otherwise, it continues to execute.

To implement pipe mechanism and redirections, we need to store original stdin and stdout using dup so I have 2 global variables to store them, def_stdin and def_stdout. I use dup2 to convert stdin and stdout to default versions at the beginning of each execution. Also, there is a variable called “prev_fd” which stores the read end of the previous pipe. Since it is equal to default stdin at the beginning, we can use dup to duplicate the default stdin and assign it to prev_fd.

In the “execute” function, pipe mechanism requires a pipe at first. Before initializing pipe, we need to check if it is the last command or not. If it is the last command, then we don’t need to have a pipe.
For both cases, it creates a child process using fork and when it is in child process, it changes stdin and stdout.
If it is the last command, we don’t need to change stdout so it duplicates stdin with prev_fd which is the read end of the previous pipe and closes prev_fd because we don’t need it anymore.
If it is not the last command, additionally, it changes stdout. It duplicates the write end of the pipe with stdout so that the output will be read from the read end. After duplication, it closes the write end of the pipe. Also, we don’t need to read end of this pipe since we use the previous one, we should also close it as well.
After all of these steps, execl is called by child process and the command is executed. When it finishes, it is terminated and the parent closes prev_fd and the write end of the pipe and it returns the read end of the current pipe. Parent also prints the pid and the executed command in the log file.

For redirections, we need to tokenize the command for “<“ and “>”. In “is_redirection”, it checks if there is a redirection. After checking, it splits the command and stores separately and sends it to the “execute” function. In “execute”, if it is “<“, it changes stdin to the file descriptor of the given file. If it is “>”, it changes stdout to the file descriptor of the given file. Before that, if there are spaces, it removes them because we give the file name by extracting from the command.

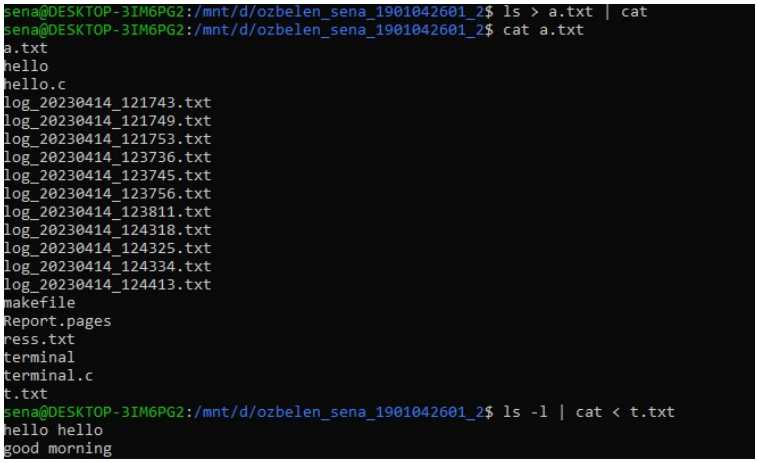
For signal handling, signals are declared at first : SIGINT, SIGTERM. There is a function called “handler” and it handles the signal and prints info about the signal. If it is SIGINT, then it kills the current process whose pid is stored in a global variable and waits for the new command.

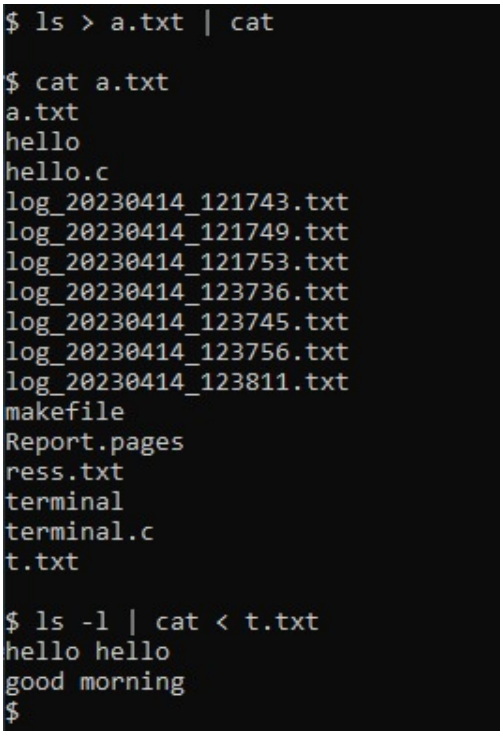
Test Cases

!!! Redirections change the input and output so it is not a good idea to use them inside pipes. “<“ should be used at the beginning and “>” should be used at the end in order to get a proper result. This program only supports file redirections such as “cat > text.txt” !!! Also, when a redirection exists, it only changes input and output by using redirection sources so pipe is disabled for the given direction.

“ ls -l > a.txt | cat” —> This redirects the output to the given file but pipe is disabled for that direction so cat doesn’t print anything
“ ls -l | cat < t.txt ” —> This redirects the input but pipe is disabled that for that direction so cat prints redirected file.

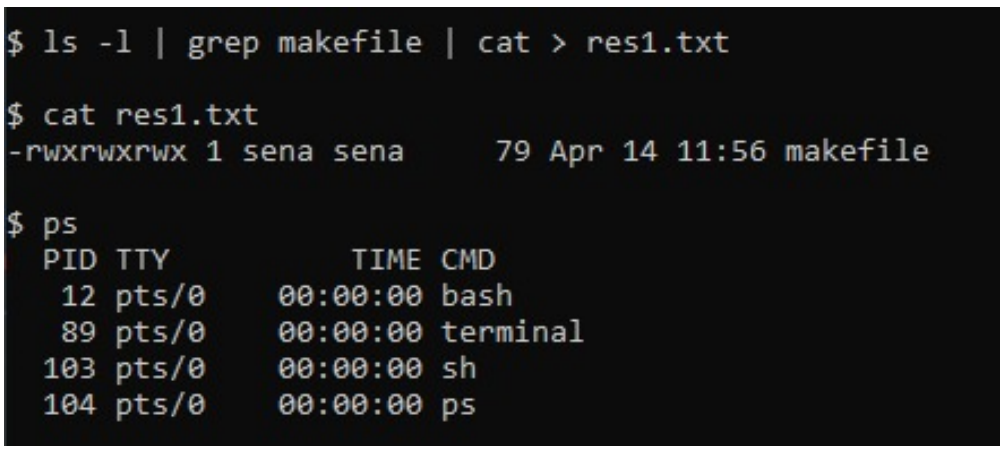
The left output is from Ubuntu terminal and the right output is from my terminal



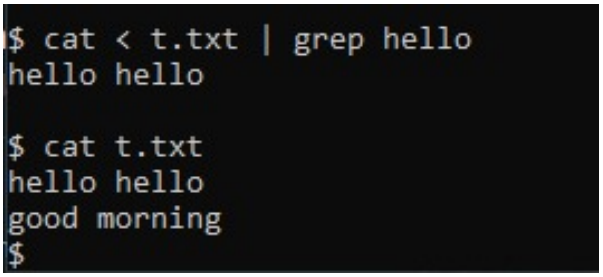


!!! Sometimes, commands might give unexpected errors. (Like cat: command not found)

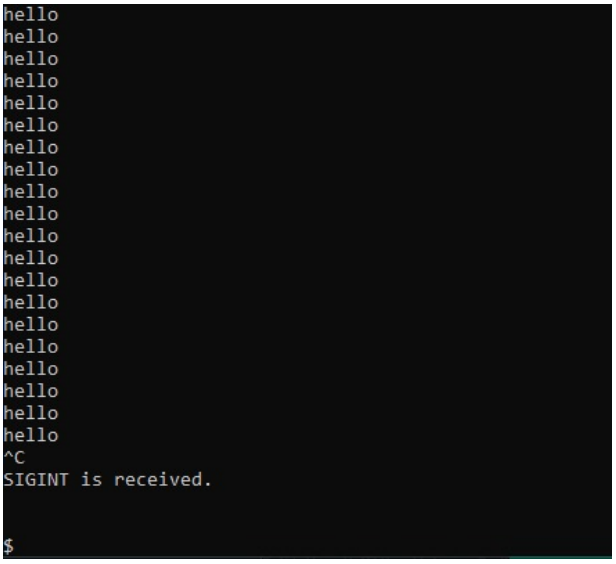
- ls -l l grep makefile l cat > res1.txt (test case for both pipes and redirection >)



- cat < t.txt | grep hello (test case for both pipes and redirection <)



- ./hello (which produces an infinite loop to test SIGINT)
- (Compile it before executing the shell “gcc -o hello hello.c)



- ps -ef | grep -c root > res3.txt
- ps -ef | grep root | cat