CSC 345-02 Project 1

# Project #1 User Interface

Chris Jenson

02/26/21

**Program Requirements:**

Child process creation and executing commands: Implemented

Child process creation through forking and executing commands using the exec() family of commands was fully implemented. The forking occurs after the command is read in. The child then executes the command using the execvp(char *command, char *params[]) prototype. The program does check whether the user included an '&' to determine if the parent process should wait for the child process to exit or not. An exit request is also checked for, if an exit is entered, the program breaks out of its while loop ending the execution.

Command history management: Implemented

Command history is handled by checking the read in command for the "!!" string. The program first verifies that there is a previous command by checking if there is a previous argument stored. If there is no previous command, the user is alerted. If there is a previous argument, the function to read in commands exits back to the main function without changing the stored arguments. This causes the program to run again with the same command as the previous request.

Add support of input and output redirection: Implemented

Support of input and output redirection is included in the program. After the command is read in and is returned to main, the code iterates through the arguments. As it iterates, it checks for the redirection characters '<' and '>'. If one of those characters is found, the specified file is accessed with error handling for bad files. The dup() function is then used to get the file descriptor for reading or writing. The files are closed at the end. The program does check whether the user included an '&' to determine if the parent process should wait for the child process to exit or not.

Allow the parent and child process to communicate via a pipe: Implemented

The parent and child process can communicate via a pipe. After the command is read in and returned to main, the code iterates through the arguments. As it iterates, it checks for the pipe character '|'. When this occurs the two commands are obtained. The code then forks for the pipe, the child executes the first command and then the parent waits for the child to finish before executing the second command. The pipe() function is used to connect the output of the first command to the input of the second. The program does check whether the user included an '&' to determine if the parent process should wait for the child process to exit or not.

Keep working directory information (change directory by [cd] command): Implemented

The user can execute change directory [cd] commands. The program checks for the “cd” token. If the token is found, a flag is set. When the flag is high, the program will then grab the cd request and pass into the chdir() function. This function changes the working directory of the program. The code constantly prints the current working directory for the user and shows the updated directory in the event of a cd request. There is error handling for a bad ccd request.