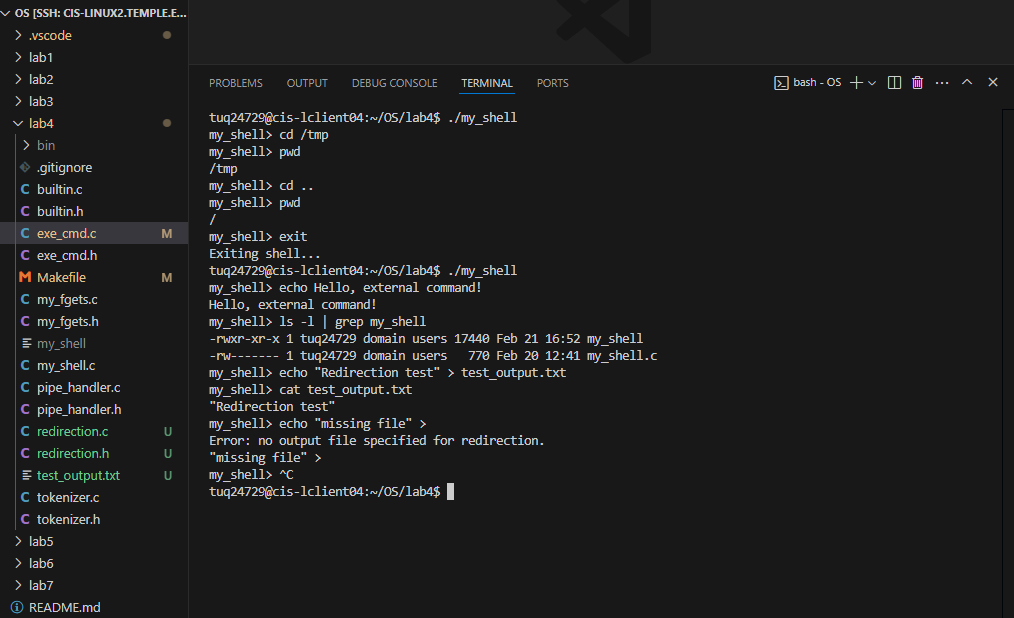
Kenneth Angelikas   
Feb 18, 2025

Operating Systems

Lab 4

***Code–*** <https://github.com/KennyAngelikas/OS/tree/main/lab4>

***All Tests***  


***Explanation of each function***

### **builtin.c**

* \**do\_cd(char args[])*
  + **Purpose:** Implements the cd (change directory) built-in.
  + **How it Works:**
    - Checks if a target directory is provided; if not, prints an error message.
    - Uses the chdir() system call to change the current directory.
    - Returns a status value (1) to indicate that a built-in was executed.
* \**do\_exit(char args[])*
  + **Purpose:** Implements the exit built-in to terminate the shell.
  + **How it Works:**
    - Prints an exit message.
    - Calls exit(0) to stop the shell.
* \**builtin(char args[])*
  + **Purpose:** Determines if the given command is a built-in.
  + **How it Works:**
    - Iterates through a lookup table (builtins[]) mapping command names to functions.
    - If a match is found, it calls the corresponding function.
    - Returns 1 if a built-in was executed or 0 if not.

### **exe\_cmd.c**

* \**exe\_cmd(char input)*
  + **Purpose:** Central function to process and execute the command line entered by the user.
  + **How it Works:**
    1. **Tokenization:**
       - Calls tokenize\_input() to split the input string into an array of tokens (args).
    2. **Pipe Detection:**
       - Scans for the pipe operator (|).
       - If found, passes control to pipe\_handler() (see below) and returns.
    3. **Output Redirection:**
       - Calls check\_redirection() (from redirection.c) to look for a > operator and extract the filename (if any).
       - This function also modifies the token list to remove redirection tokens.
    4. **Built-in Check:**
       - Calls builtin() to see if the command is built-in (e.g., cd or exit).
       - If it is, the built-in is executed in the parent process and the function returns.
    5. **External Command Execution:**
       - Forks a child process using fork().
       - In the child, if output redirection was requested, opens the file and uses dup2() to redirect stdout.
       - Executes the command using execvp().
       - In the parent, waits for the child to complete.

### 

### 

### **my\_fgets.c**

* \**my\_fgets(char buffer, int size)*
  + **Purpose:** Custom implementation to read a line from standard input.
  + **How it Works:**
    - Reads characters one at a time with getchar().
    - Stops reading when a newline or EOF is encountered or when the buffer is full.
    - Null-terminates the string.

### **my\_shell.c**

* **main()**
  + **Purpose:** The main loop of your shell.
  + **How it Works:**
    - Displays a prompt (my\_shell> ).
    - Uses getline() to read user input dynamically.
    - Removes the trailing newline from the input.
    - Skips empty commands.
    - Calls exe\_cmd() to process and execute the input command.
    - Frees allocated memory and exits when EOF is reached or the user types exit.

### **pipe\_handler.c**

* \*\*pipe\_handler(char **args, int pipe\_index)**
  + **Purpose:** Handles commands that include a pipe (|).
  + **How it Works:**
    1. **Splitting:**
       - Replaces the pipe token with NULL to split the arguments into two arrays: one for the command before the pipe (left\_cmd) and one for after (right\_cmd).
    2. **Pipe Creation:**
       - Calls pipe() to create a unidirectional communication channel.
    3. **Forking Processes:**
       - Forks a child process for the left command:
         * Redirects stdout to the write end of the pipe using dup2().
         * Executes the left command (if built-in, calls the built-in function).
       - Forks another child process for the right command:
         * Redirects stdin to the read end of the pipe.
         * Executes the right command.
    4. **Cleanup:**
       - The parent process closes both ends of the pipe and waits for both child processes to finish.

### **redirection.c**

* \*\*check\_redirection(char **args, int arg\_count)**
  + **Purpose:** Scans the command arguments for output redirection using >.
  + **How it Works:**
    - Iterates over the args array looking for >.
    - If found, ensures that a filename follows; if not, prints an error.
    - Saves the filename and replaces the > token with NULL (truncating the argument list for execution).
    - Returns the filename (or NULL if no redirection is specified).

### **tokenizer.c**

* \*\*tokenize\_input(char \*input, char *args[], int arg\_count)*
  + **Purpose:** Splits a string (user input) into tokens for processing.
  + **How it Works:**
    - Uses strtok() to split the input string by spaces.
    - Fills the args array with tokens and updates arg\_count accordingly.
    - Ensures that the argument list is terminated with a NULL pointer.