Overview

My goals of the assignment is to construct a shell that can be run on MINIX. Furthermore, it can support the basic command like “ls”, “cd”, inredirect, outredirect, and the pipe.

To reach the goals, I have decompose the assignment into the following major parts:

- Identifying processes
  Every processes can be created by the function fork(). Furthermore every processes have its own PID. When a new command is executed, a new child process is created using the fork() function. In the parent process, use the waitpid() function to wait for the termination of the child process, thereby ensuring that the parent process continues execution after the child process completes.
- fork
  Used to create a new process (child process). Its function is to copy the current process to form a completely independent new process. The new process (child process) obtains all the contents of the parent process, including program code, data space, stack, file descriptors, etc. The content of these two processes are the same when they are created, but they are independent during subsequent execution. They run independently and do not interfere with each other.
- execvp
  Used to execute program files under the specified path. It is a member of the exec() function family. exec stands for "execute" and is responsible for loading a new program into the current process and executing it.
- waitpid
  It allows a process to take appropriate action while waiting for a child process to finish or receive a signal.
- exit
  It is the function that is used to end program execution.

- chdir
  It is used to change the current working directory. If successful, it returns 0; Else it return -1.

Major data structures

Major data structures in my program is Array. But I have defined a struct called ParseInfo to store information about each command.

Synchronization

If the type of command is PIPE, then we should do the synchronization.

We should create a process first, and then create the second process. We can achieve this strategy by if-else statements.

The challenges are that I have to realize that what the pipe is and how it works; besides, I have to write the code that can accomplish the pipe function.

Major algorithms

- Read the command (read_cmd)
- Split the command into parameters (split_cmd)
- Construct a define structure that can store the information that has been parsed (ParseInfo)
- Execute the command (execute_cmd)

How the different parts of the design interact together ?

- First, the function read_cmd read the input from the user, and it will call remove_space() to remove the space of the command.
- The remove_space() will remove the space in front of the commands and the space at the end.
- After processing the space, the program will call split_cmd() split the commands into parameters that stores in an Array called “tokens”.

- Then, the program will call cmd_info to parse the commands to gain the information, and to determine which type of the command is. (No special symbol is assigned NORMAL, “>” will be assigned OUT_REDIRECT, “<” will be assigned IN_REDIRECT, and “|” will be assigned PIPE.
- Finally, the function execute_cmd() will execute the commands according to their type.

In-depth analysis and implementation

The functions to be implemented

- read_cmd ()
  It will read the commands that the user inputs. It uses an array called “cmd”. After receiving the input from the stdin, it then pass to the function called remove_space() to remove the space that is in front or at the end of the commands.
- remove_space()
  It uses the pointer that is the beginning of the array called “str” and the pointer that points to the end called “last”. I uses these two pointers to remove the space in front and at the end of the commands.
- split_cmd()
  I uses an array called tokens with initialized with all NULL, and then extract the parameters from the commands that have been removed spaces to store into tokens.
- parse_cmd()
  It will parse the commands to get the informations. For example, if there exists a “&” symbol, then it will set the background to be 1, indicating that this commands should run in the background.
- execute_cmd()
  It will execute the commands. If the commands is “cd”, it will change the current directories. If the type of commands is NORMAL, it will simply execute it. If the type of commands is PIPE, it will deal with it. Otherwise, it will call execute_inredirect() and execute_outredirect() to execute the

redirect commands.

- execute_inredirect()

  It will execute the inredirect.
- execute_outredirect()

  It will execute the outredirect.

The existing functions to be modified

The function execute_cmd() has too many if-else statements, maybe it can be optimized.

Corner cases

- fgets()

  fgets returns NULL if it encounters EOF or an error while reading the command.
- MAX_TOKEN_NUM

  I define the MAX_TOKEN_NUM. If the parameters is more than 50, the process will print out "Error: Command is too long."
- int check

  In parse_cmd, I set a variable called "check" to check if there exists strange symbols, it will execute check++. If the check value is more than 1, it will print "Error: error command.".
- execute_cmd()

  If the directory is not exist, it will print the error messages "Error! Unknown directory."
- fork()

  First will call pid = fork(), if the value of pid is smaller than 0, it means that it create the child process, and it will print out the error message.
- pipe_fd[2]

  pipe_fd[2] is used to implement the pipe function. I will test `if(pipe(pipe_fd) < 0)`, if it is true, it will print out "Error: Unable to pipe."
- command not found

  If users input the strings that are not commands, the program will detect it and print out "Error: command not found".

- execute_inredirect()
  If the file can not open, it will print the error message “Error: Unable to open file for input redirection."
- execute_outredirect()
  First of all, it will open the file with 0666, which means that the file will be set with read-write permissions, which means that all users can read and write to the file. If the file can not open, it will print the error message.

Test plan

Input Handling

- Test 1: Enter a simple command without any redirection or pipes.
  The command should be executed successfully.
- Test 2: Enter a command with excessive whitespace.
  The excessive whitespace should be trimmed, and the command should be executed successfully.
- Test 3: Enter a command longer than the maximum allowed length.
  The shell should report an error indicating that the command is too long and exit gracefully.
- Test 4: Enter an empty command.
  The shell should prompt for a new command without executing anything.

Command Execution

- Test 5: Execute a command in the foreground.
  The command should be executed, and the shell should wait for its completion before prompting for a new command.
- Test 6: Execute a command in the background.
  The command should be executed asynchronously, and the shell should immediately prompt for a new command without waiting for its completion.
- Test 7: Execute a command with input redirection.
  The command should read input from the specified file instead of stdin and execute successfully.
- Test 8: Execute a command with output redirection.

The command should write output to the specified file instead of stdout and execute successfully.

- Test 9: Execute a command with a pipe.
  The output of the first command should be passed as input to the second command via the pipe, and both commands should execute successfully.

Error Handling:

- Test 10: Enter a command with incorrect syntax (e.g., multiple redirection symbols).
  The shell should report an error indicating that the command has incorrect syntax and exit.
- Test 11: Attempt to execute a nonexistent command.
  The shell should report an error indicating that the command was not found and exit.
- Test 12: Attempt to open a nonexistent input or output file for redirection.
  The shell should report an error indicating that the file could not be opened and exit
- Test 13: Attempt to create a pipe when the system resources are exhausted.
  The shell should report an error indicating that the pipe could not be created and exit gracefully.

Risks

Have been handled

- Imperfect parameter parsing:
  The function parse_cmd can correctly parse command parameters and determine the command type based on the redirection and pipe symbols in the command, thus avoiding the risk of command parsing errors.
- File opening error handling:
  In the functions execute_inredirect and execute_outredirect, error handling is performed on file opening failures, and corresponding error messages are output to avoid risks related to file operations.

Haven’t been handled

- Imperfect handling of command execution failures:
  In the function execute_cmd, when the command execution fails, although an error message will be output, there is no mechanism to clean up resources, which may lead to resource leaks or inconsistent system status. Resources should be released and status restored when command execution fails.
- Buffer overflow risk:
  In the function read_cmd, the fgets function is used to read the command entered by the user, but there is no check whether the input length exceeds the buffer size. If the command entered by the user exceeds the length of the cmd array, a buffer overflow problem will occur, which may cause the program to crash or be exploited by attackers.

Estimates for how long the project should take

- Best case (1 week):
  Have already known what the structure of shell is, knowning how to use the system calls, go search to find how other people implement the shell.
  In this case, I can implement all the parts of the assignment.
- Average case(2 weeks):
  I should spend a lot time searching for what the structure of shell would be looked like and how to use the system calls. Furthermore, I should spend a lot of time to figure out how to design a shell perfectly.
  In this case, I can implement 90% of the assignment, maybe the pipe function can not be implemented.
- Worst case(1 month):
  If I don’t go to the OS class, I can’t even know what the system call is.
  In this case, I can only finish the basic commands, including “ls”, “cd”.

In the above situations, if any parts of the assignment can not be implemented, then I will make sure that when the user input commands like in/out redirection or pipe function, it will print out that “The commands are not found”, to avoid the program being crushed.

Assignment questions

- How will the general program do if the user only input the commands “cd” without any parameters ? What will the minix shell deal with this situation?
- In minix shell, when I type “ls”, the shell will execute directly without waiting, so does my assignment, so is that test case suitable ?

Final

There is a readme file indicating how to compile and run my code. You can read that file before testing my code.