Code Report: Linux Shell

The given task required implementing a shell with a certain number of functionalities. The solution implements all the required functionalities, the autocomplete feature and the command history. It is also capable of creating other directories. Unfortunately, the $ expansion feature was not implemented due to time constraints.

The shell implementation follows the sequence:

1. Clean any zombie processes, if any
2. Print prompt text
3. Get user input
4. Check whether the program should exit
5. Save the input to the history
6. Tokenize the input
7. Execute

which presents a comfortable interface for the user. Step 1 is required so that any program executing with ‘&’ that runs in the background will not be left “unwaited”, thus removing all zombie processes generated by this process.

Step 7 must execute the required logic, considering whether the user has included many pipelined commands. For example, the sequence “ps | sort | head -n 10 | tee” results in 4 commands, requiring at least three pipes to connect all the processes. To do so, the following algorithm is followed by the solution:

```
for each command
      if there is a command after
            new_fds = pipe()
      fork()
      if in child
            if there is a command before
                  dup2(old_fds[0], stdin)
                  close old_fds
            if there is a command after
                  dup2(new_fds[1], stdout)
                  close new_fds
            execvp(command)
      else
            if there is a previous command
                  close old_fds
            if there is a next command
                  old_fds = new_fds
if there is more than 1 command
      close old_fds
```

The algorithm keeps track of which pipes to create between processes and ensures that no opened pipes are left open after execution. To handle I/O redirection, the program uses the *dup2* system call, which duplicates a file descriptor in a specific location. The algorithm also supports this feature and closes the file descriptors when appropriate. After extensive experimentation, it was noticeable that this algorithm works correctly. For *N* commands, a total of *N-1* pipes are created. Special attention was given to the output redirection phase - besides the flag to write to the file. It is also required to use the *O_CREAT* flag and pass the file permissions so the writing can happen. The mask 0777 was used in the proposed solution.

The mkdir feature is automatically implemented, given that a child process creates the directory with this functionality.

For the command history and autocomplete features, some extra effort was required. Given that *bash* uses buffered input, the arrow keys and tab hits were not reaching the program before the enter key was pressed. Therefore, it was required to set the terminal into raw mode by communicating with the *termios* library from C.

For the command history, a simple vector is used, which stores the raw string input by the user. These strings are accessed in a read-only fashion (any changes to a selected history entry will not be stored in the history unless the command is executed and a new entry is generated). Then, this raw string is tokenized and fed into the algorithm as if a new entry was run.

The autocomplete feature requires understanding where the user is trying to autocomplete (which directory/directories) and filtering the valid options. In the solution, the autocomplete feature makes a best-effort approach to finding a valid option, adding the required text if only one option is left and listing all valid options in case there are more.
Furthermore, the search must be aware of which directories to navigate. In the case of a non-local search, the *PATH* environment variable is searched by splitting the directories on the *:* separator.

Due to the input of the user showing up in the terminal, some cleaning is done after these functionalities, which includes the main keys used in the scope of this task (up/down arrows, tabs). Some special keys can result in multiple keystrokes into the terminal, so the result might not be perfect.