PA2 Report

Parsing:

Tokenizer data structure is used to parse commands. This data structure is a singleton design. At first, I read user command input as a whole string and then I call Tokenizer constructor that takes in a string. In this constructor, it will go char by char in order to parse the command. Whenever it sees a ' ', it will create a new token as a string and store it as an element of "tokens". It also checks for special characters such as '|', '>','<','&'. Whenever it sees '&', it turns "isBackground" to true. Whenever it sees '>' or '<', it turns "isIORedirect" to true. If it sees " or ', it will consider everything inside as 1 token. Whenever it sees '|', this means a new command is created so I store the previous command (or everything in current "tokens" with "isIORedirect" value) as an element of "cmdTokens" and reset "isIORedirect" to false for next command.

When it comes to process each command, vector "cmdTokens" will be iterated to process command by command. For every command to be processed, it will parse vector of tokens (string type) into char *[] (array of pointers). However, whenever it sees '>' or '<', it stops parsing and starts IO redirection process. After parsing into char *[], it is parsed into execvp.

IO redirection:

Whenever IO redirection command is detected, a file is open accordingly.

If it is '>' , fd = open(fileName, O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);

If it is '<', fd = open(fileName, O_RDONLY,0);

Before execvp is called, I call dup2 to redirect std::in or std::out to file input/output:

If it is '>', dup2(fd, STDOUT_FILENO);

If it is ' <', dup2(fd, STDIN_FILENO);

Pipe:

For piping, before execvp is called, I create a pipe. Then, I perform a fork to create a child process. In the child, I redirect stdout to pipeout and close pipein. In the parent process, I wait until the child finishes, close pipeout, and redirect stdin to pipein. For example, this means that output of command 1 will serve as input of command 2, and output of command 2 will serve as input of command 3 and so on until the last command.

Background:

Whenever there is an indication of background process, I push pid_t of the child into vector of pid processes and then let parent continue its process (no wait for child). Then, I process the vector of pid processes by using waitpid to see if any child finishes and reap it (no zombies allowed).