Assignment 1: Design (Revised)

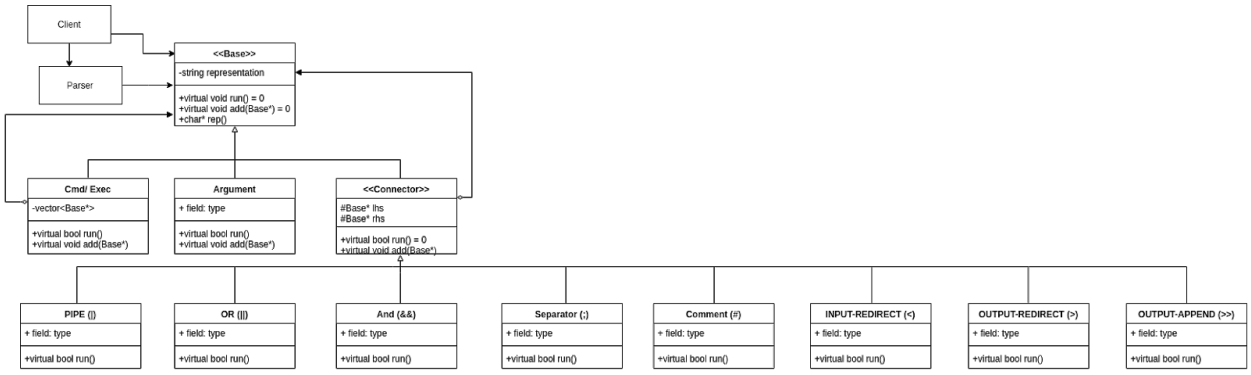
Nathan Brennan
Abdelrahim Hentabli
Ronan Todd

12/13/18
Fall 2018

Introduction:

We will be creating a command shell using C++. It will be able to read commands and execute them. We will also be allowing the use of connectors ";", "&&", and "||". We will use syscalls "fork", "execvp", and "waitpid" to execute the commands the program receives.

UML Diagram:



Classes/ Class Group:

Parser:

The parser will display the machine and username as well as a "$", then take a string as an input. It will first input the first "word" as a cmd and every subsequent word as an argument in that command. If it reaches an operator, it will create an operator, then add the cmd before and after it as its children. Finally, the parser will return a base* to the head of the "tree" which contains our full command. Additionally, the parser is now able to handle both () and [ ] commands. The parser now holds an additional pre-parser that will check if the input contains brackets and sets up a test command that gets stored into the command object. A stack has been implemented to correctly iterate and build the tree of commands to properly follow precedence rules. The stack only affects the way the tree is built and keeps all previous functionality intact.

Base:

This will just have a char array representation of its command name, argument name, or operator representation. And is an abstract parent class for all the remaining classes, with the functions run(), and add().

Argument:

Just has empty run() and add() functions that do nothing.

Cmd/Exec:

Will add in arguments into the vector with add. And will execvp in run(). A new case has been made to handle the test command and will run the necessary test with -e being the default. Test does not use execvp() but has been implemented so that the client remains unaffected.

Operator:

Will add into lhs than rhs in add() while run() is still virtual.

OR:

Will fork in run(), if the lhs.run() fails, it will run the rhs.run(). If lhs.run() doesn't fail however, it will not run the rhs.

AND:

Will fork in run(), will only run rhs if lhs runs.

Separator:

Will fork in run(), will run the lhs than rhs. rhs may be null, in that case it will only run lhs.

Comment:

Will fork in run(), will run the lhs, and skip the rhs.

Pipe:

Will use the output of the lhs and use it as the input for the rhs.

Input-Redirect:

Takes the command as an argument on the right of the '<' and uses it as an input for the left command

Output-Redirect:

Takes the output from the left of the '>' and outputs overwrite the contents to a file.

Output-Append:

Takes the output from the left of the '>' and outputs overwrite the contents to a file.

Coding Strategy:

One member will create a parser, while the other creates the base class and cmd/Exec class. This will insure that we have a framework to test using googletest. Then we each work on either Argument or Operator. Finally, we will each create 2 out of the 4 operators, this will insure that we both know how to work with fork() and waitpid() for fixing any issues we may have find in google testing our project.

Roadblocks:

One of the first roadblocks that we foresee is the usage of fork(), waitpid(), and execvp() in our Operator and Execute classes. We have already looked through tutorials and shell documentation, and haven't completely understood the usage of it. We plan to overcome this through trial and error while using the functions in our program.

Another roadblock we may encounter is with the Argument class, we may need to restructure the classes because Argument has to implement the run() function as well as the add(Base*) function, but it does not do or add anything and is just a character array.

Known Bugs:

We have used destructors to get rid of allocated memory. However, we did not create assignment operators and copy constructors, this may result in undefined behavior.