Assignment 1: Design

11/1/18

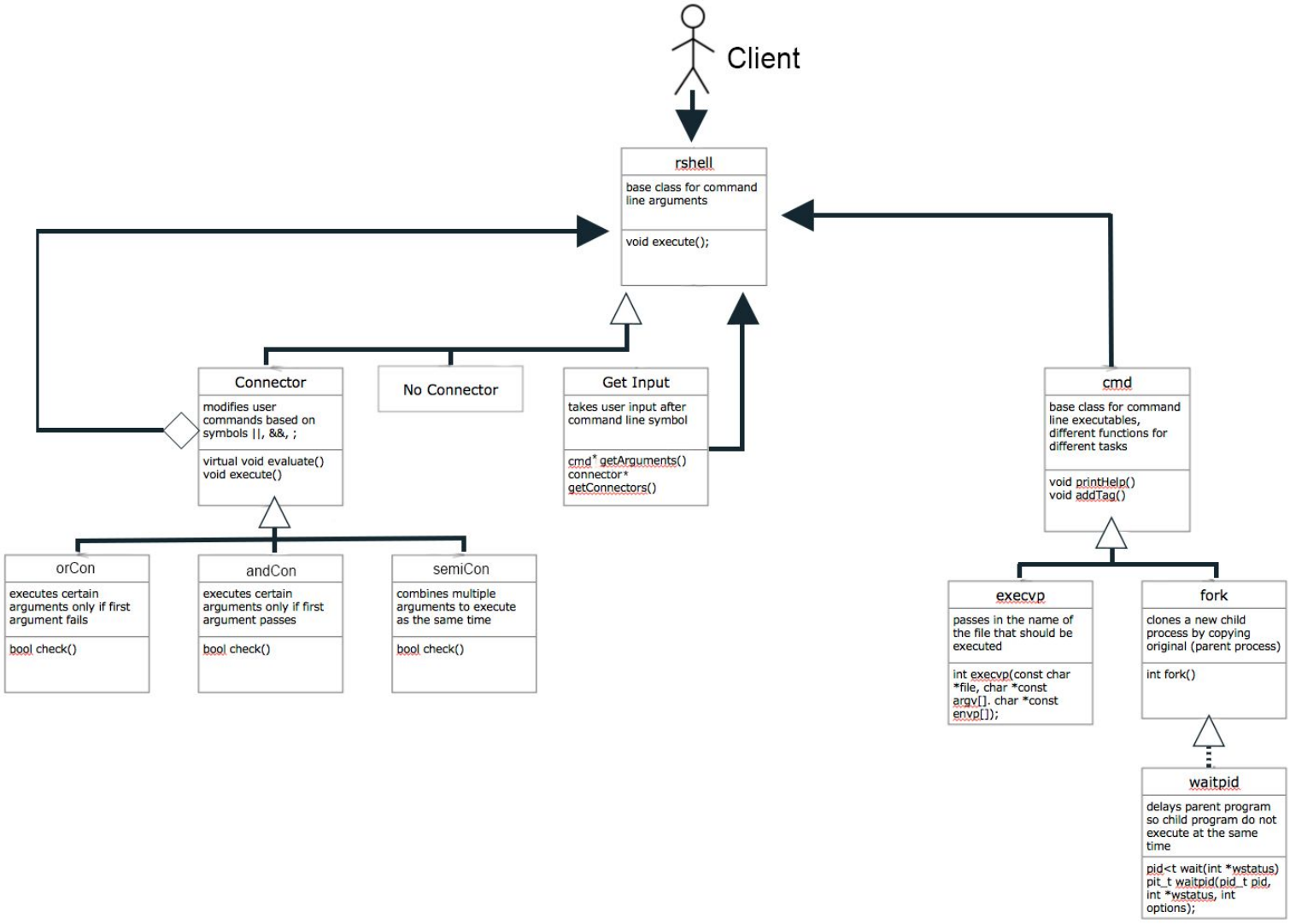
Fall 2018

Raymond Kim, Lucas Song

Introduction:

For this project, we will be designing a basic command shell. This command shell will be able to print a command prompt, read in a line of user input, segment this input into an executable, argument list, connector, and more, and use said input to execute the proper commands using fork, execvp , and waitpid. The shell should be able to run multiple commands at once, utilizing symbols to determine how multiple commands should be run in unison. These include "| |, &&, or ;."

Diagram:



Classes/Class Groups:

rshell

- Acts as a base class
- Other classes inherit from it or interact with it
- **void execute();** Executes commands based on user input command line arguments

Connector

- Inherits from rshell base class
- Modifies user commands in that it is used to run multiple commands at once
- Only acceptable symbols: || (or), && (and), ; (semicolon)
- **virtual void evaluate();**
- **void execute();** Executes next command

orCon

- Inherits from Connector class
- Executes next command if the first argument fails
- **bool check();** Checks if the first argument fails, returns a boolean value to indicate whether or not to execute next command

andCon

- Inherits from Connector class
- Executes next command if the first argument passes
- **bool check();** Checks if the first argument fails, returns a boolean value to indicate whether or not to execute next command

semiCon

- Inherits from Connector class
- Executes next command
- **bool check();** Checks if there is a follow-up command, returns a boolean value to indicate whether or not to execute next command

NoConnector

- If the user inputs a single command, simply execute

GetInput

- Takes user input
- **cmd* getArguments();** Returns pointer to argument
- **Connector* getConnectors();** Returns pointer to connector

cmd

- Acts as a base class for commands to run executable programs
- **void printHelp();** Prints list of helpful code for clients to use

tag

- Acts as a base class for tags that will be used to modify command line arguments
- **Virtual void tag();**

execvp

- Passes in the name of the file that should be executed
- **int execvp(const char *fil**
- **0e, char *const argv[], char *const envp[]);** Duplicates action of shell in searching for executable file

fork

- Clones original process (parent) and produces a child process
- **int fork();** Creates new process by duplicating current process, returns PID of new process

waitpid

- Used to delay parent process
- Functions: All system calls are used to wait for state changes of child process

Coding Strategy:

We intend to split the work in such a way that we are both able to familiarize ourselves with the program. We will both implement the classes and their respective functions, however, we will test each others' code so that we are exposed to all aspects of the program and its design.

Roadblocks:

An issue that we anticipate is in parsing the user input. Since there are many ways for spaces and stray symbols to introduce problems in the program, we must ensure that the data is properly compartmentalized.

In addition, coding out the logic of connectors seems a little confusing. We are not completely sure as to how to approach the case of multiple connectors with multiple arguments that must execute in different cases.

Not only this, we infer that there will be many changes to the structure of our design. As such, keeping an accurate account of changes that have been made and will be made should allow for smoother transitions between adding new features.