**Overview**

This project aims to implement a simple Linux shell interface that will allow the user to run processes both in the foreground and in the background. It will also support auto recovery mode for background processes and allow querying of statistics about background processes. The project will be written in C++ and heavily utilize the concepts we learned in class such as forking and process creation.

The shell will support the following builtin commands:

· ‘cd’

· ‘exit’

· ‘echo’

· Pipe (|) command

The shell will support the following I/O redirection:

· file-to-standard input

· Standard-output-to-file

**Architectural Description**

The general event loop of the project will be:

1. Receive input from user

2. Parse the command and execute it

3. If foreground process started, block until it finishes.

4. Run waitpid with WNOHANG to check for finished background processes.

5. Repeat

We will have the following classes:

1. Main class that contains the event loop of the program

2. Process class: Holds information about running process

3. Parser class: Parses user input, extracts commands and sends information to Builder

4. Builder class: Builds the command object

5. Command class: Executes the user’s commands

The project will utilize the following patterns for different aspects:

1. Command Pattern:

It turns the user input into different object that contain the method calls representing individual commands that the user provides.

2. Builder Pattern:

It is used to create command objects step by step by passing each initialization parameter when available.

We intend to use Boost.Spirit and C++ Standard Library. Boost.Spirit will be used for creating a grammar for our program that supports the above commands.

We can hold background and foreground Process objects in std::unordered\_map or concurrent queue.

Error conditions:

· What if the string is too long(buffer overflow)? Will the program crash?

· What if parser doesn’t recognize the command? Is proper error shown to the user?

· If exception thrown by program, throw to user and exit program.

· Does the program automatically restart if killed by signal other than SIGSEGV and SIGKILL?

· If run a program that doesn’t exist, show error.

· Check and remove newline and tab characters from user input

· Check state that should not be shared across multiple shells or across multiple executions such as current working directory

· While querying status of background process, throw error if process already dead.

Milestones:

- Parsing user input correctly

- Executing processes in the foreground

- Executing processes in the background

- Add auto recovery mode support

- Handling error codes sent by process

- Keep track of running processes and allow user to query this info

Timeline:

- March 21 - Finish the design aspect

- March 24 – Submit a report about the design.

- March 29 - Finish the parser

- April 5 - Have foreground and background process execution ready

- April 9 – status update and submit a set of slides describing what you have done till now.

- April 18 - Test and finish the project

- April 21 – final report