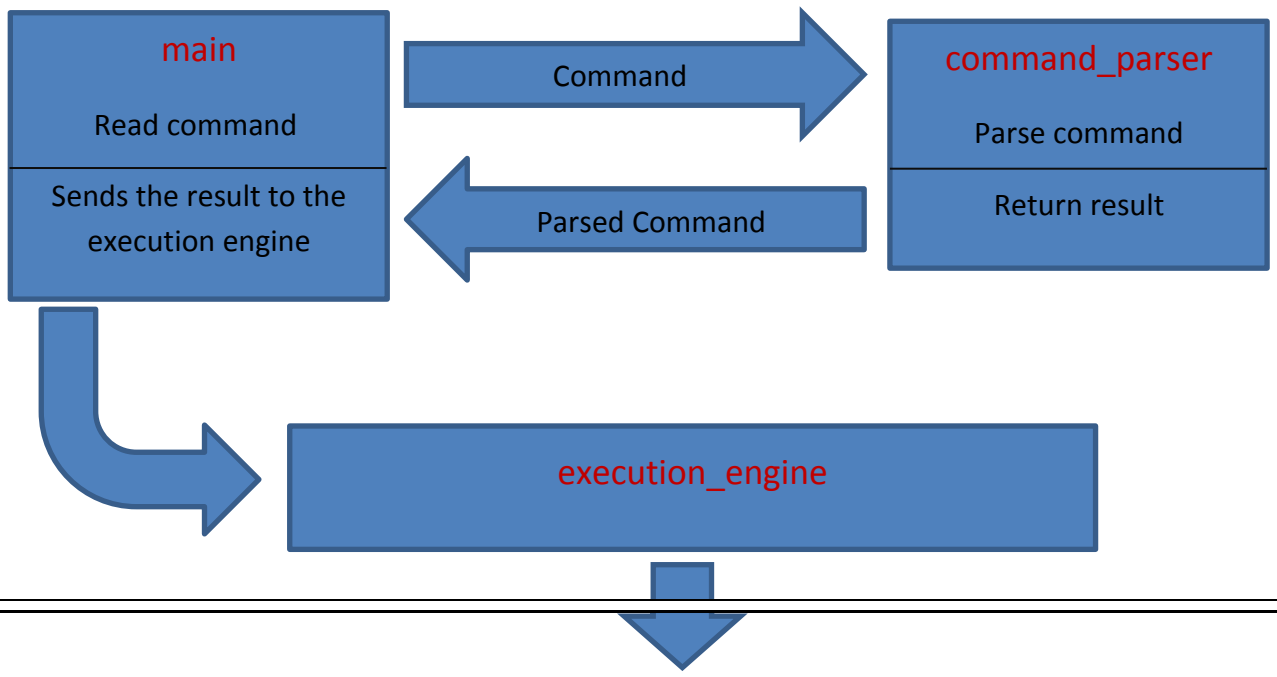
Objectives

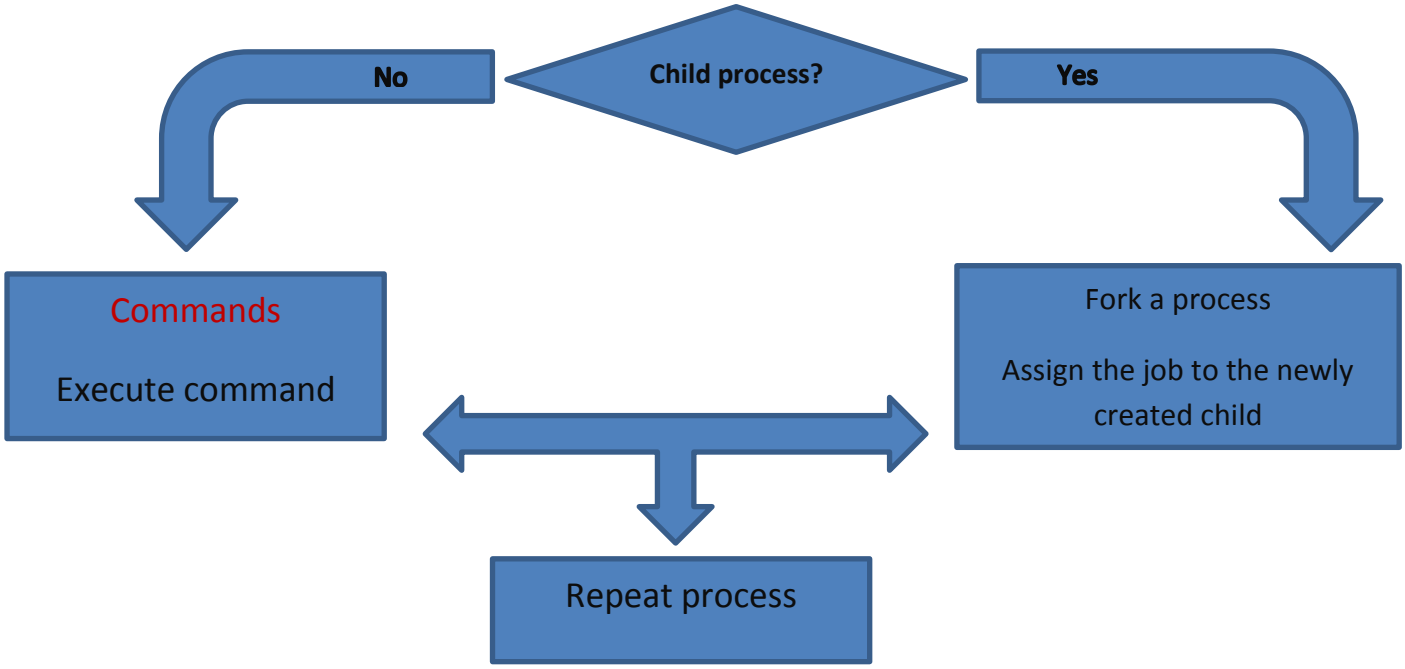
- To get familiar with Linux and its programming environment.
- To understand the relationship between OS command interpreters (shells), system calls, and
- The kernel.
- To learn how processes are handled (i.e., starting and waiting for their termination).
- To learn robust programming and modular programming.

Overview

You are required to implement a command line interpreter (i.e., shell).
The shell should display a user prompt, for example: Shell>, at which a user can enter for example, ls -l command, as follows: Shell> ls -l.
Next, your shell creates a child process to execute this command. Finally, when its execution is finished, it prompts for the next command from the user.

Design





Main Functions

main(): the main function that setups the environment, open files and defining which mode to enter (i.e. interactive or batch mod).

setup_environment(): sets the environment variables.

shell_loop(): the main loop that's running all the time waiting for user input.

trimwhitespace(): trims the leading and trailing white spaces.

parse_command(): a wrapper function that parses the command line input by calling the following functions:

- **expand_variables():** expands every variable to its corresponding value.
- **tokenize():** returns an array of tokens from the expanded command line.

execute(): checks if a command should be done in the parent process,if so, calls **external_command()** , if not, it forks the parent process,creates a new child and calls **child_process**() to do the job.

Guidance

-To run the shell; head to the shell files location and run the "**makfile"** (type: **make** in the terminal). A file named **shell** will be created, type:

./shell command_line_arguments[optional]

-log and history files are created in the same shell location.