# Command Line Interpreter

## Operating Systems Lab Project

121046 Saumya Shah

121060 Varsha Teckchandani

# Project overview

Command line interpreter or shell. The shell should operate in this basic way: when one type in a command (in response to its prompt), the shell creates a child process that executes the command entered and then prompts for more user input when it has finished.

The shells student implement will be similar to, but much simpler than, the one run every day in Linux. They can find out which shell; they are running by typing echo $SHELL at a prompt. For this project, one do not need to implement much functionality; but one will need to be able to handle running multiple commands simultaneously.

This shell can be run in two ways: interactive and batch.

# Abstract

## Batch and Interactive execution

Batch execution involves execution of all the commands in the batch file

For example: If we run a shell script which has a set of commands then all those commands should be serially executed. Say running shell [Batch\_file\_name] will execute the commands in the file “Batch\_file\_name” until it sees the exit command.

Interactive mode is something similar to the terminal that we use. An interactive console that will allow us to run commands just as one does on the command prompt. Also we need to show proper error messages as and when required. The exit command concept holds in this case too.

## Built in commands

We will be implementing the basic unix built-in commands such as cd, exit, echo etc

## Running multiple commands at a time

Multiple commands execution should be permitted separated by a special character for example;

Next command should be executed only when the previous one is executed successfully for example

> echo Hello; echo Bye

This will execute both commands. Output will be

Hello

Bye

## & separator

In addition, programs can be launched asynchronously with the "&" separator. This causes the all preceding commands since the beginning of the line or the last '&' operator to execute without waiting for the result. This should be implemented by calling fork() and then executing all the commands since the previous "&" separator in sequence.

For example:

> ls -l &

> echo Hello

will execute "echo Hello" without waiting for "ls -l" to complete.