# Assignment 3

## 1 Introduction

After doing an assignment on command shell interface and learning the nuances of process creation and loading, it now becomes imperative as a systems programmer to understand how pattern matching and pipes work. This assignment will give you a deeper knowledge of process relationships and management.

## 2 Objective

To understand Linux process management and how pattern matching using 'grep' and pipes work.

## 3 Description

Augment your mini command shell to include the facility for pattern matching using grep. Additionally, your shell should also support the use of pipes i.e “|”.

### 3.1 ‘mygrep’ Command (15 marks)

**SYNOPSIS**

- mygrep [OPTIONS] PATTERN [FILE...]

**DESCRIPTION:**

mygrep searches the named input FILEs or lines containing a match to the given PATTERN. By default, mygrep prints the matching lines.

**Options to be implemented :**

- -v, --invert-match :
  Invert the sense of matching, to select non-matching lines.

- -m NUM, --max-count=NUM
  Stop reading a file after NUM matching lines.

**Regular expression** :
It should support the following :

A regular expression may be simple string or followed by one of several repetition. Operators:

| | |
|---|---|
| ? | The preceding item is optional and matched at most once. |
| * | The preceding item will be matched zero or more times. |
| + | The preceding item will be matched one or more times. |

<u>Example :</u>

- mygrep [option] abc [FILE...]
- mygrep [option] abc?xyz [FILE...]
- mygrep [option] xyz*abc [FILE...]

## 3.2 <u>Pipes</u> (20 marks)

Modify your shell to accept the names of two or more executable programs and redirect the output of the first program to the input of the second program, output of the second program to the input of the third program, and so on. The executable files will be separated by “|“.
<u>Example:</u>

(i) ls | sort | wc
(ii) myls | sort |wc

## 3.3 <u>Background Process</u> (20 marks)

On many occasions a process can be executed in the background i.e without it being attached to any terminal. This requires the process to detach itself from the terminal and start a new session and become a session leader.
Your mini shell should facilitate the user to run any process in the background by appending a ‘&’ at the end of the input. Once a process goes into background, the shell prompt should appear and wait for the next input. The background process may send its output (if any) to the stdout.
Example:

```
[/usr/home/]$ gedit &
```

## 3.4 Command History (25 marks)

It is often handy for users to have a shortcut key for their recently used commands. Incorporate such a logger functionality for maintaining the history of commands and bind it to the Up and Down arrow keys. The Up and Down arrow key should allow the user to iterate through the most recent commands.
The memory for the logger should be dynamically allocated. Initially, memory for only one command should be allocated. Whenever the buffer is full more memory should be dynamically reallocated in powers of 2.

## 3.5 Complete Shell (20 marks)

Along with all the above functionalities and commands, your mini shell should execute any binary file that is either in the current path or any path (absolute/relative) given in the $PATH environment variable.

Example:

```
myls | sort r | mygrep ‘xyz’
```

myls and mygrep is from your own program sort is an external command,piping again from your own function.

# 4 Deliverables

It is mandatory that your code should follow proper indentation and commenting. The C programs with name as “myprogram_rollno.c” along with the Makefile should be submitted in a compressed tar file format with the name **rollno_a3.tar.gz**. There will be deductions in the awarded marks, if you fail to do so. All the source c files should remain in the same directory.