Lab 2
Writing, Compiling, Running C Programs in LINUX, Creating Static Libraries and Exploring Redirections, Pipes and Advanced LINUX Commands for Scripting

Muhammad Bilal Qureshi
Department of Computer Science
School of Engineering

Report Submission Date: 11th February 2025

Objective

- To practice writing and compiling programs using the `vi` editor.
- To understand static library creation and linking in C programming.
- To explore redirection, piping, and advanced Linux commands for scripting and file management.

Task 1

Write a simple C program using the `vi` editor to print:

- Your **ID**, **Name**, and **marks** of all subjects in an array.
- Ensure that all data is stored in appropriate variables or arrays.
- Familiarize yourself with `vi` commands from Lecture 1 and Lecture 2 slides.

Saving, Compiling, and Running

1. Press the `Esc` key to exit insert mode.
2. Save and exit the file using: `:wq`.
3. To compile the code, use the command: `gcc hello.c`.
4. Run the program using: `./a.out`.
5. Alternatively, compile the code with a custom output name: `gcc -o hello hello.c`.
6. Run the program with the custom name: `./hello`.

Task 2

Repeat the task of creating a static library and linking it with a program as mentioned in the Lecture Slides (pages 43–45). Ensure you understand each step before proceeding to Task 3.

Linking, Compiling, and Running

1. Compile the source files into object files:

```
gcc -c fred.c bill.c
```

2. Create a static library called `libfoo.a`:

```
ar crv libfoo.a bill.o fred.o
```

3. Use this library to link it to the program:

```
gcc -o program program.o libfoo.a
```

4. Run the program: `./program`

Task 3

Create a static library named `libYourLastName.a`. This library must include the following matrix functions:

1. `void YourName_Get (double Mat[][3])`
   Reads data for a 3x3 matrix.
2. `void YourName_Show (double Mat[][3])`
   Displays a 3x3 matrix.
3. `void YourName_Add (double Mat_A[][3], double Mat_B[][3], double Mat_C[][3])`
   Adds two 3x3 matrices (`Mat_A` and `Mat_B`) and saves the result in `Mat_C`.
4. `void YourName_Mul (double Mat_A[][3], double Mat_B[][3], double Mat_C[][3])`
   Multiplies two 3x3 matrices (`Mat_A` and `Mat_B`) and saves the result in `Mat_C`.
5. `void YourName_Inv (double Mat_A[][3], double Mat_B[][3])`
   Computes the inverse of a 3x3 matrix (`Mat_A`) and saves the result in `Mat_B`.
6. `double YourName_Mod (double Mat_A[][3])`
   Calculates and returns the modulus of a 3x3 matrix (`Mat_A`).

Write the steps to compile and use the library, then call these functions in your main program to demonstrate their functionality.

Task 4

Work with redirection and file manipulation using the following commands:

- `$ ls -al | more`
- `$ man bash | col -b | lpr` (If a printer is not added, redirect the output to a file.)
- `$ bash --version`
- `$ /bin/bash --version`
- `$ /bin/zsh --version`

- `$ ls -l > lsoutput.txt`
- `$ ps >> lsoutput.txt`
- `$ kill -HUP 1234 > killout.txt 2> killerr.txt`
- `$ kill -1 1234 > killouterr.txt 2>&1`
- `$ kill -1 1234 > /dev/null 2>&1`
- `$ more < lsoutput.txt`
- `$ ps > psout.txt`
- `$ sort psout.txt > pssort.out`
- `$ ps | sort > pssort.out`
- `$ ps | sort | more`
- `$ ps -xo comm | sort | uniq | grep -v sh | more`
- `$ cat mydata.txt | sort | uniq > mydata.txt`

Task 5

Installation related commands for Ubuntu MacOs:

- `$ sudo apt update` (To update the list of packages.)
- `$ sudo apt install bash` (On personal systems only.)
- `$ brew --version` (For macOS, check if Homebrew is installed.)
- If Homebrew is not installed, use:
  `$ /bin/bash -c "$(curl {fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`
- To update Homebrew:
  `$ brew update` and install Bash using: `$ brew install bash`.

Online Resources

- Access Linux terminal online: `https://www.labex.io/learn`
- Access Linux using CoCalc: `https://www.cocalc.com`

Notes for macOS Users

- Some commands like `apt`, `dpkg`, and `rpm` are not available on macOS. Use `brew` for package management instead.

Conclusion

Through this lab, students will develop essential skills in system programming by writing and executing C programs using the `vi` editor, creating and linking static libraries, and exploring advanced Linux commands such as redirection and piping. These activities will serve as a foundation for understanding program compilation, efficient code reuse through libraries, and Linux system-level operations. This lab will prepare students for shell scripts writing, advanced topics in system programming and efficient software development in projects.