National University of Computer and Emerging Sciences

Lab Manual 01
Operating System Lab

Course Instructor	Mr. Ibrahim Nadir
Lab Instructor (s)	Zumirrah Khalid
Section	E
Semester	Spring 2021

Department of Computer Science

FAST-NU, Lahore, Pakistan

Objectives

In this lab, students will:

1. practice Basic commands on terminal

2. develop a small program in C for reading/writing files
3. create a makefile program for compilation

Basic Commands

• Clear the console: **clear**
• Changing working Directory: **cd Desktop**
cd Home
• List all files in directory: **ls**
• Copy all files of a directory within the current work directory: **cp dir/***
• Copy a directory within the current work directory: **cp -a tmp/dir1**
• Look what these commands do
cp -a dir1 dir2
cp filename1 filename2

Compiling C and C++ Programs on the Terminal:

For C++:
Command: g++ source_files… -o output_file

For C:
Command: gcc source_files… -o outputfiles

Example:

- g++ main.cpp lib.cpp –o output
- ./output

MakeFile

- Make is Unix utility that is designed to start execution of a makefile. A makefile is a special file, containing shell commands that you create and name makefile (or Makefile depending upon the system). While in the directory containing this makefile, you will type *make* and the commands in the makefile will be executed. If you create more than one makefile, be certain you are in the correct directory before typing make.

- Make keeps track of the last time files (normally object files) were updated and only updates those files which are required (ones containing changes) to keep the sourcefile up-to-date. If you have a large program with many source and/or header files, when you change a file on which others depend, you must recompile all the dependent files. Without a makefile, this is an extremely time-consuming task.

- As a makefile is a list of shell commands, it must be written for the shell which will process the makefile. A makefile that works well in one shell may not execute properly in another shell.
- However, for a large project where we have thousands of source code files, it becomes difficult to maintain the binary builds. The **make** command allows you to manage large programs or groups of programs.

- The **make** program is an intelligent utility and works based on the changes you do in your source files. If you have four files main.cpp, hello.cpp, factorial.cpp and functions.h, then all the remaining files are dependent on functions.h, and main.cpp is dependent on both hello.cpp and factorial.cpp. Hence if you make any changes in functions.h, then the **make** recompiles all the source files to generate new object files. However, if you make any change in main.cpp, as this is not dependent of any other file, then only main.cpp file is recompiled, and help.cpp and factorial.cpp are not.
- While compiling a file, the **make** checks its object file and compares the time stamps. If source file has a newer time stamp than the object file, then it generates new object file assuming that the source file has been changed.

Structure of Makefile:

```
Target: dependencies
        Action
```

Naming of Makefile:

By default, when make looks for the makefile, it tries the following names, in order: `GNUmakefile', `makefile' and `Makefile'. You can give any of the three names to your makefile. The convention is to us the name “Makefile” (capital M).

Running the Makefile:

Simply run the command “make”. The current working directory should be where the intended makefile is placed.

Benefits of Makefile:

Makefile checks the last modified time of both the source file and the output file. If the output file’s last modified time is later, then it will not compile the source files since the outputfile is already latest. However, if any of the source files is modified after the creation of output file, then it will run the command since the output file is outdated.

Example:

Suppose we have two cpp files: main.cpp, lib.cpp, and and a heade file lib.h. Suppose the main function in main.cpp makes use of several functions from lib.cpp. In order to compile our program, we will create the makefile as follows:

```
main.out: main.cpp lib.cpp
        g++ main.cpp lib.cpp -o main.out
```

In Lab Tasks

Question 1:

See the usage of the following commands online. Also run them on the terminal.

1. pwd
2. ls
3. cd
4. cp
5. mkdir & rmdir
6. man
7. sudo
8. apt-get
9. kill
10. ping
11. grep
12. mount
13. unmount

Question 2:

a. Create a function removeNonAlphabets(char * inputFileName, char * outputFileName) in C or C++ that is passed as parameters: an input file name and an output file name. The function then reads the input file using read system call and removes all non-alphabets. It then writes the data to output file using write system call. You will need to see open, read, write, and close system calls. https://www.geeksforgeeks.org/input-output-system-calls-c-create-open-close-read-write/

b. Place the signature of the function in a header file named util.h, and place the implementation of the function in a cpp file named util.cpp

c. Now create a main.cpp file containing the main function which calls the function created previously. (For input file, create a random text file containing alphanumeric characters)