National University of Computer and Emerging Sciences

Laboratory Manual # 02
Operating Systems

Course Instructor	Mubashar Hussain
Lab Instructor	Muhammad Hashir Mohsineen
Section	BCS-4E
Date	04-Feb-2025
Semester	Spring 25

Instructions:

- Submit a world/LibreOffice file containing screenshots of terminal commands/ Output
- Submit your .c (Code files)
- In case of any explanation you can add a multiline comment.

Objectives:

- Basic Linux commands
- Makefile Utility

Reading Material

Makefile:

- Make is a 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 source file 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.
- While compiling a file, the make checks its object file and compares the time stamps. If the source file has a newer timestamp than the object file, then it generates a new object file assuming that the source file has been changed.

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 use 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 output file is already latest. However, if any of the source files is modified after the creation of the output file, then it will run the command since the output file is outdated.

Suppose we have two cpp files: main.cpp, lib.cpp, and a header 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:

touch Makefile // this will create a makefile, open Makefile and write:
main.out: main.cpp lib.cpp
    g++ main.cpp lib.cpp -o main.out

Open Makefile and paste the following code there.
hello: hello.c main.c
  gcc -o hello hello.c main.c -I.

In line 1, the hello before ':' symbol is the target name. You can run make hello in the terminal to compile your code. In line 1, two .c files (hello.c and main.c) after ':' sign shows the dependencies for building a target.

In line 2, we tell the make utility how to build a target. Here, we are telling the make utility to build the code using gcc compiler. The -o flag tells about the output binary name. In our case, the output binary file name would be hello. After that, we have told the compiler to compile hello.c and main.c. In the end of line 2, -I flag tells the compiler to find the header files in the present working directory.
In the terminal, run the following command in the sample_project directory.
 make hello

Now, run the below command. Your program will get executed.
./hello

Further details with example =>
https://docs.google.com/document/d/1Db9h-HPnGnap81bnfZFhEdwSJL6J5e8OoUE3b60Vs3A/edit?usp=sharing

GNU make manual: https://www.gnu.org/software/make/manual/html_node/

1. Exercise: [10]

Create the following C files:

- main.c: The main function that calls a function from math_operations.c.
- math_operations.c: A function that adds two numbers.
- math_operations.h: A header file declaring the function from math_operations.c.

Makefile: Write a Makefile that compiles these files into an executable called calculator.

2. Exercise: [10]

Create 3 files

- main.c
- functions.c
- Header.h

header.h file contains following function prototypes:

```
void sort(int array(), bool order)
void findHighest(int array[], int nth)
void print(int array[])
```

functions.c file contains following 3 functions along with their logic

```
void sort(int array[], bool order) {
> sort in ascending order if order is true
> sort in descending order if order is false
}
void findHighest(int array(), int nth) {
> find nth highest value
if nth 2 find 2nd highest value from the array
}
void print(int array[]){
> print all elements in the array
}
```

In main.c you will accept command line arguments including 3 things

- an array of integers
- order of sort (1 for ascending order and 0 for descending order)
- nth position to get the nth highest number from the array

Use makefile to execute all these files.
Example Input: ./main 11 15 13 12 16 14 18 19 20 17 1 4