OS Lab 2 - Introduction to Makefile

TAs of OS Lab

August 14, 2020

1 Introduction

Make is a program building tool that runs on Linux, Unix and their flavours. Generation of program executables may need multiple modules. If number of such modules are huge then manually typing the names of such modules every time during compilation is not possible. Make provides a simple yet systematic way of code organisation. It also helps in building and managing projects automatically. The user has to create a Makefile where some specific rules have to be defined to notify the make utility what are the tasks that are needed to be executed.

2 Rules

A file named Makefile has to be created in which rules are to be defined. Given below is an example of a rule that can be defined in a make file

```
hellomake : hellomake.c hellofunc.c
    gcc -o hellomake hellomake.c hellofunc.c -I
```

hellomake, present on the left-hand side of the : is the name of the rule, make will know that the rule hellomake has to be executed whenever there is a change in any one of the file defined in the right-hand side of the :. A tab is mandatory before beginning of any command in a rule.

3 Macros

```
CC=gcc
CFLAGS=-I
DEPS = hellomake.h
%.o: %.c $(DEPS)
    $(CC) -c -o $@ $< $(CFLAGS)
hellomake : hellomake.c hellofunc.c
    $(CC) -o hellomake hellomake.c hellofunc.c -I
```

Here the macro CC is used to define the compiler that is to be used for compiling, CFLAGS are the flags that are to be used during compilation, DEPS has the list of header files which is required to generate the executables.
The rule says that the .o filesa re dependant on the .c files and the .h files defined in DEPS.

The rule further says to compile .c files using the comipler defined in the CC macro. $ refers to the left handside of :. It is ued to say that use gcc to compile the .c files and produce .o files with name same as left of the :.

4 Maintaining Directory Structure

Directory structure can be also maintained for the project by defining constants which contains the location of the target folders respectively. For example constant
OPT = ../output_files can be defined which can be used in a rule as $(OPT) later to specify the path of a particular output file.

```
hellomake : hellomake.c hellofunc.c
    $(CC) -o $(OPT)/hellomake hellomake.c hellofunc.c -I
```

Similarly, one can maintain separate constants for input files, object files and so on.

5 Links to further read about make

For further reading please look into https://www.gnu.org/software/make/manual/make.html