Make(software) & G++ Compiler

LinhTo

June 14, 2014

1 Introduction

In software development, **Make** is a utility that automatically builds executable programs (*.exe) and libraries from source code (*.cc, *.h) by reading files called **makefiles** which specify how to derive the target program. Though IDE and some compiler features can also be used to manage a build process, Make remains widely used, especially in **Unix**.

In OMNeT++ and inet, "make makefiles" and "make" will create makefiles which tell how to build the target executable program and make will create according to how-to from makefiles.

- **Compiler**: source files (*.cc) → object files (*.o)
- **Linker**: object files (*.o) → executable files (*.exe in windows)

The trivial (shortcut) way to compile all the files and obtain an executable is by running the following command

```
$ g++ main_test.cc function.cc -o main_test
```

We will learn how to write makefile to do the above job in automatic and faster way.

2 Tutorial-Make

```
Make's used to build executable programs & libraries from source.
Make searches the current directory for the makefile to use.
```

Compiling your source code files can be tedious, especially when you want to include several source files and have to type the compiling command everytime you want to do it. Compile → Build almost over. Noew you will learn how to write Makefiles. Makefiles are special format files that together with the *make* utility will help you to automatically build and manage your projects.

```
Note: I use g++ for compiling.
Note: You can change to others.
```

What-if-run-make: If you run **make**, this program will look for a file named *makefile* in your current directory, and then execute that *makefile*.

2.1 Syntax

```
target: dependencies
[tab] system command
```

Example: we wrote a makefile with content as

```
all:
     g++ main_test.cc function.cc -o main_test
```

Then put it in the directory with *.cc files and call it from terminal with

```
$ make -f makefile
or
$ make all -f makefile
```

If you only type MAKE, Unix searches for default file name MAKEFILE.

If you add the option -F and MAKEFILE_NAME, then unix only executes commands of the MAKEFILE_NAME. -f means file

- Target is ALL, default target for all kind of makefiles.
- No dependencies for target *all*
- MAKE executes the system command specified.

2.2 Target: Clean

```
clean:
     rm -rf *o test
```

Then call it from terminal by

```
$ make clean
```

The final argument "test" is just the name of executable file.

2.3 Using dependencies

Dependencies can be useful when you want to track, cooperate all the files. In the section 2.1, the result is only executable file, but you did not notice any object (*.o) files as output. What happened? The reason is the difference between

```
$ g++ *.cc -o test --> create *.exe
and
$ g++ *.o -o test --> create *.exe
and
$ g++ -c *.cc --> create *.o
```

Depend on what you want to track, you can use different commands.

3 Tutorial-G++

G++ is a script to call gcc with options to recognize C++. GCC processes input files through one or more of four stages

- pre-processing
- compilation
- assembly
- linking

3.1 Syntax

```
g++ [option | filename] ...
```

3.2 Options

4 Conclusions