PH502: Scientific Programming Concepts

Irish Centre for High End Computing (ICHEC)

September 23, 2020

Overview

- There are some topics that are not directly related to coding but are important to know as a programmer.
- One of these is Makefiles and the utility *make*.
- So far we have been using only single programs which are easy to compile.
- Some applications have so much code that it is split into different files to make it easy to read and maintain.
- In this case compiling can be a bother, using Makefiles smooths the process.

Makefiles

- Large software packages can be installed semi-automatically.
- A typical process would be “configure”, “make” and “make install”.
- The configure step generates the “Makefile”s that are appropriate for your system. The make step build all the executable programs which are then installed into a publicly accessible directory.
- Here we shall focus on the “make” phase. Below is an example.

```
VARS = something

section: dependencies
<tab>        statement1; \
<tab>        statement2
```

- When a user types “make section” the dependencies are checked. Then the statement in that section are executed within a shell.

Example

- Below is an example Makefile. Variables are defined at the top. There are several sections which are interdependent. Prior dependencies are executed first.

```
CC = icc
CCFLAGS = -g -ansi
LDFLAGS = -lm
ROOTDIR = $(shell pwd)
CTARGS = prog1 prog2

install: prog
    cd /usr/bin; cp $(ROOTDIR)/prog .

prog: prog1.o prog2.o
    echo "Linking"; \
    $(CC) -o $@ prog1.o prog2.o $(LDFLAGS)

$(CTARGS):
    echo "Compiling"; $(CC) -c $(CFLAGS) $@.c
```

Variables

- So from the example above we can see that variables are used to customize the compile.
- The configure step can set these variables. Here is a simple example.

```
CC = icc
```

- We can also use the Linux shell commands to set variables.

```
ROOTDIR = $(shell pwd)
```

- Then there are two special variables $@ and $ˆ. The first corresponds to the section name and the second the section dependencies.

Example

- Here is another example using more complex variables.

```
CC = gcc
CPROGS = $(shell ls *.c)
OBJ = $(CPROGS:.c=.o)

all: prog

prog: $(OBJ)
        $(CC) -o $@ $^ -lm


$(OBJ):
        $(CC) -c $(@:.o=.c)

clean:
        /bin/rm -f *.o
```

Clean

- In the last example we had a *clean* section.
- Make is a smart utility in that it will only update files that have been changed.
- You could have an application that takes hours to compile. If you make a small change then only that part of the code is recompiled.
- However if you have a stable code base then you may want to remove the intermediate files (.o).
- It is normal practice to have a *clean* section which removes these files.