Make, Processes, and the Environment

Compilation

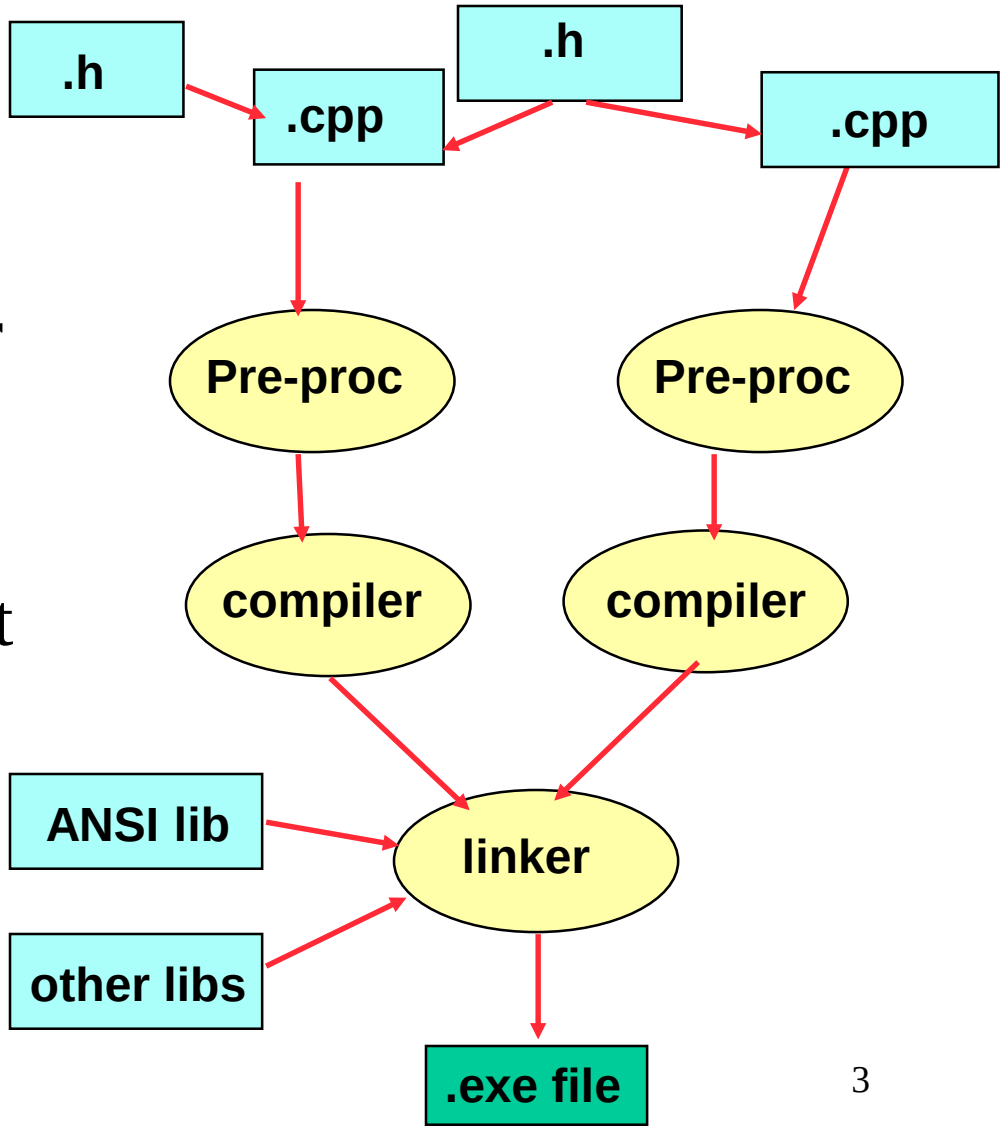
- To compile a program:
  - `g++ <options> <source files>`
  - Recommended:

```
g++ -Wall –ansi -pedantic –g
    –o <executable_name> *.cpp
```

    - `-Wall` : *W*arnings: *ALL*
    - `-ansi` : Strict ANSI compliance
    - `-pedantic`: reject non-ANSI compliant code
    - `-g` : Add debugging symbols to the executable (i.e., make it debuggable!)

“Compilation” or “The Big Lie”

- The *discrete* steps to program “compilation”
- Typing “`g++ *.cpp`” to *build* (not “compile”) your program hides all these separate steps.
- Question: would you want to do this entire process (i.e., pre-process and compile *every* file) each time you want to generate a new executable?



Selective Recompilation and Makefiles

- Answer: NO!
  - You only want to compile those files which were changed (or were affected by a change in another file. We can reuse the .o/.obj files for files which weren't modified.
- You could do this yourself…

```
g++ <options> <changed files>
g++ *.o
```

Selective Recompilation and Makefiles

- You can also use the **`make`** command and a **`Makefile`**!
  - Create a **`Makefile`** to keep track of file dependencies and build options
  - The **`make`** command will read the **`Makefile`** and compile (and build) those files which have ***dependencies on modified files!***

Makefile Syntax

- Makefiles consists of *variables* and *rules*.
- Rule Syntax:

  `<target>: <requirements>`

  `<command>`

  - The `<requirements>` may be files and/or other targets
  - There ***must*** be a tab (not spaces) before `<command>`
  - The first rule in a `Makefile` is the default `<target>` for `make`

Makefile Syntax

- Variable Syntax:

  `<variable> = <string value>`

  - All variable values default to the shell variable values
  - Example:
    - BUILD_FLAGS = -Wall -g -ansi

Make

- Make
  - make [-f makefile][option] target
  - A tool to update files that are derived from other files. Great for software development.
  - The default files for make are ./makefile, ./Makefile, ./s.makefile, ……in order
  - The default files can be overwritten with the –f option
    - Make –f myprog.mk

Make

- Make
  - The makefile has three components:
    - Macros: define constants
    - Target rules: tell how to make targets
    - Inference rules: also tell how to make targets, make will first check if a target rule can apply before it checks the inference rules.

Make

- Macros:
  - String1 = string2
  - E.g. CC=gcc
  - CFLAG=-Wall –ansi –pedantic
- Target rules:
  - Target [target…] : [prerequisite…]
  - <tab> command
  - <tab> command
- Example:
  - a.out: myprog1.c myprog2.c myprog3.c
  - $(CC) myprog1.c myprog2.c myprog3.c

Example Makefile

```
# Example Makefile
CXX=g++
CXXOPTS=-g –Wall -ansi -DDEBUG

foobar: foo.o bar.o
     $(CXX) $(CXXOPTS) –o foobar foo.o bar.o

foo.o: foo.cc foo.hh
     $(CXX) $(CXXOPTS) –c foo.cc

bar.o: bar.cc bar.hh
     $(CXX) $(CXXOPTS) –c bar.cc

clean:
     rm -f foo.o bar.o foobar
```

Processes 1: suspend, background, foreground

- When you run a command, it starts a new "process"
- Each process has a unique number called the PID (Proccess ID)
- Unix is a multitasking operating system, so you can run multiple processes simultaneously
- After you start a process, it is usually in what is called the "foreground." That is, it takes over your shell.

Processes 1: suspend, background, foreground

- You can suspend processes in the foreground with Ctrl-Z
- The process is now frozen. You can pull it to the foreground again by typing “fg”.
- Alternately, you can make it keep running, but put it in the background so can still use the same shell by typing “bg”.
- You can also start a task in the background by putting a & at the end of the command.

Processes 2: ps, kill, kill -9

- You can list all the processes that have been run from the current shell with “ps”
- To list all the processes on the system, do “ps awux”

```
root     3723 0.0 0.1 3092 988 pts/20
  S 16:21 0:00 –bash

awong    3724 0.0 0.1 1406 712 pts/17
  R 16.35 0:00 –bash

awong    3725 0.0 0.1 2334 716 pts/17
  R 16.36 0:00 ps awux
```

Processes 2: ps, kill, kill -9

- If you want to end a process, you can do
  "kill <pid>"
  eg. kill 3724

- If that doesn't work, do the "super kill,"
  "kill -9 <pid>"
  eg. kill –9 3724

If a process of yours freezes, you can login to the same machine again, do a "ps -awux" and find the process number and then kill it.

Environment Variables

- What are environment variables? Think of them as parameters that get passed to your programs. All operating systems have them. They give information about the context within which a program runs.
- The most commonly used environment variable is the **PATH** variable which tells the shell where to look for programs.

Environment Variables

- To create an environment variable in bash, type: export VARNAME=value
- To create an environment variable in tcsh type: setenv VARNAME value
- Don’t forget the export or the setenv, otherwise you create a “Shell variable” instead of an environment variable.