1 How to use Linux

Hongik University
Eunsung Jung

Disclaimer: The slides are borrowed from many sources!

Standards: ANSI, ISO, POSIX, etc.

- *Why standards are required? Portability*

Header	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10	Description
<assert.h>	•	•	•	•	verify program assertion
<complex.h>	•	•	•	•	complex arithmetic support
<ctype.h>	•	•	•	•	character classification and mapping support
<errno.h>	•	•	•	•	error codes (Section 1.7)
<fenv.h>	•	•	•	•	floating-point environment
<float.h>	•	•	•	•	floating-point constants and characteristics
<inttypes.h>	•	•	•	•	integer type format conversion
<iso646.h>	•	•	•	•	macros for assignment, relational, and unary operators
<limits.h>	•	•	•	•	implementation constants (Section 2.5)
<locale.h>	•	•	•	•	locale categories and related definitions
<math.h>	•	•	•	•	mathematical function and type declarations and constants
<setjmp.h>	•	•	•	•	nonlocal goto (Section 7.10)
<signal.h>	•	•	•	•	signals (Chapter 10)
<stdarg.h>	•	•	•	•	variable argument lists
<stdbool.h>	•	•	•	•	Boolean type and values
<stddef.h>	•	•	•	•	standard definitions
<stdint.h>	•	•	•	•	integer types
<stdio.h>	•	•	•	•	standard I/O library (Chapter 5)
<stdlib.h>	•	•	•	•	utility functions
<string.h>	•	•	•	•	string operations
<tgmath.h>	•	•	•	•	type-generic math macros
<time.h>	•	•	•	•	time and date (Section 6.10)
<wchar.h>	•	•	•	•	extended multibyte and wide character support
<wctype.h>	•	•	•	•	wide character classification and mapping support

Figure 2.1 Headers defined by the ISO C standard

Standards: ANSI, ISO, POSIX, etc.

Header	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10	Description
<aio.h>	•	•	•	•	asynchronous I/O
<cpio.h>	•	•	•	•	cpio archive values
<dirent.h>	•	•	•	•	directory entries (Section 4.22)
<dlfcn.h>	•	•	•	•	dynamic linking
<fcntl.h>	•	•	•	•	file control (Section 3.14)
<fnmatch.h>	•	•	•	•	filename-matching types
<glob.h>	•	•	•	•	pathname pattern-matching and generation
<grp.h>	•	•	•	•	group file (Section 6.4)
<iconv.h>	•	•	•	•	codeset conversion utility
<langinfo.h>	•	•	•	•	language information constants
<monetary.h>	•	•	•	•	monetary types and functions
<netdb.h>	•	•	•	•	network database operations
<nl_types.h>	•	•	•	•	message catalogs
<poll.h>	•	•	•	•	poll function (Section 14.4.2)
<pthread.h>	•	•	•	•	threads (Chapters 11 and 12)
<pwd.h>	•	•	•	•	password file (Section 6.2)
<regex.h>	•	•	•	•	regular expressions
<sched.h>	•	•	•	•	execution scheduling
<semaphore.h>	•	•	•	•	semaphores
<strings.h>	•	•	•	•	string operations
<tar.h>	•	•	•	•	tar archive values
<termios.h>	•	•	•	•	terminal I/O (Chapter 18)
<unistd.h>	•	•	•	•	symbolic constants
<wordexp.h>	•	•	•	•	word-expansion definitions
<arpa/inet.h>	•	•	•	•	Internet definitions (Chapter 16)
<net/if.h>	•	•	•	•	socket local interfaces (Chapter 16)
<netinet/in.h>	•	•	•	•	Internet address family (Section 16.3)
<netinet/tcp.h>	•	•	•	•	Transmission Control Protocol definitions
<sys/mman.h>	•	•	•	•	memory management declarations
<sys/select.h>	•	•	•	•	select function (Section 14.4.1)
<sys/socket.h>	•	•	•	•	sockets interface (Chapter 16)
<sys/stat.h>	•	•	•	•	file status (Chapter 4)
<sys/statvfs.h>	•	•	•	•	file system information
<sys/times.h>	•	•	•	•	process times (Section 8.17)
<sys/types.h>	•	•	•	•	primitive system data types (Section 2.8)
<sys/un.h>	•	•	•	•	UNIX domain socket definitions (Section 17.2)
<sys/utsname.h>	•	•	•	•	system name (Section 6.9)
<sys/wait.h>	•	•	•	•	process control (Section 8.6)

Figure 2.2 Required headers defined by the POSIX standard

Editors

- *vi, pico, emacs*, etc
- What is good about *emacs*?
  - *emacs* is more than just an editor
  - You can compile and edit within *emacs*
  - If you know *lisp*, you can expand its functionality
  - Some useful commands to get started:
    - C-h t get tutorial
    - C-g cancel command
    - C-x C-c quit *emacs*
    - C-h help
- *The most recent editor: Visual Studio Code.*

C compilers

- gcc, cc
- Using ANSI C, the code must pass with flags

  –Wall –ansi –pedantic

  with no warning message
- Some examples
  - gcc –g –Wall –ansi –pedantic example1.c
  - gcc –g –c –Wall –ansi –pedantic example1.c
  - gcc –g example1.o
  - gcc –g example.o -lm

Debugger

- ddd, xxgdb, gdb
- The code must be compiled with –g option.
- The power of a debugger:
  - Finding the line that causes coredump.
  - See example:
    - Break point, show value, change value, step, next, continue, print
  - Very efficient in debugging sequential code
  - Not very effective in debugging concurrent code (multiple threads, multiple processes)
- *Good software development practice: You must have seen each line of your code execute in the debugger, at least once*

Make

- make [-f makefile][option] target
  - A tool to update files derived from other files
  - The default files for make are ./makefile, ./Makefile, ./s.makefile
  - Use the *–f* option to specify some other file
    - make –f makefile1
  - The makefile has three components
    - Macros: define constants
    - Target rules: Specify how targets are made
    - Inference rules: Specify how targets can be made, implicitly. *make* will first check if a target rule applies, before using inference rules.

make … continued

- Macros:
  - String1 = string2.
    - Example

      CC=gcc

      CFLAG=-Wall –ansi –pedantic
- Target rules:
  - Target : [prerequisite…]
  - <tab> command
  - <tab> command
    - Example

      a.out : myprog1.c myprog2.c myprog3.c
      $(CC) $(CFLAG) myprog1.c myprog2.c myprog3.c

make … continued

- Inference rules
  - Target:
  - <tab> command
  - <tab> command
  - Target must be of the form .s1 or .s1.s2 where .s1 and .s2 must be prerequisites of the .SUFFIXES special target.
    - .s1.s2 ➔ make *.s2 from *.s1
    - .s1 ➔ make * from *.s1
  - Example:

```
.c:
    $(CC) –o $@  $<
.c.o:
     $(CC) –c $<
```

make … continued

- $@
  - The file name of the target of the rule. If the target is an archive member, then ‘$@’ is the name of the archive file. In a pattern rule that has multiple targets (see Introduction to Pattern Rules), ‘$@’ is the name of whichever target caused the rule's recipe to be run.
- $<
  - The name of the first prerequisite. If the target got its recipe from an implicit rule, this will be the first prerequisite added by the implicit rule.
- $^
  - all prerequisites.

make example

```
CC=g++
# CC=gcc
#CC=mpicc
#NVCC=nvcc
all:TEST_load_xml
TEST_load_xml: Test_load_xml.o
	$(CC) -o $@ $^
Test_load_xml.o:Test_load_xml.cpp
$(CC) -c Test_load_xml.cpp

.PHONY:clean
clean:
	rm -f *.o
```

Review some features of C

- **Header files**
- **Macros**
- Command line arguments
- Utilities

Header files

- Usually define interfaces between separately compiled modules
- May contain macro definitions, preprocessor directives, declarations of types, and function prototypes
- Should *not* contain variable definitions or executable code

Some header file errors

- Improper header file use can cause problems
  - Try compiling *example2.c*
  - Including a header file multiple times may cause *redefinition* errors
  - Why does including *stdio.h* twice not cause any problem?
    - Look at /usr/include/stdio.h

Conditional Code in Headers

- Preprocessor directives are used to prevent the body of a header file from being used multiple times.

```
#ifndef MYHEADER
#define MYHEADER
/* the body of the header file */
#endif
```

Macros with and without Parameters

- #define MAX_LENGTH 256
  - ... for (i = 0; i < MAX_LENGTH; i++) ...
- Macros can have parameters
  - #define max(a,b) (a > b) ? a : b
- What is wrong with the following?
  - #define sum(a, b) a + b
  - #define product(a, b) a*b

Lab/Assignment #2

- Program a “Hello world!” C program with Makefile.
  - C compiler options: *Wall –ansi –pedantic*
  - No warning messages should be displayed.
- Submit hello.c, Makefile, and output file (output messages of make)