Ateneo de Manila University

Introduction to GNU Development Tools

Ateneo High Performance Computing Group
1st Semester 2001-2002
http://www.math.admu.edu.ph/ahpc/
william.s.yu@ieee.org
Section I

Introduction
Programming in the Unix Environment

★ several tools available to aid programming and development

★ the most common of these tools are the GNU tools

★ can be divided into subgroups:
  - compilers and linkers: cpp, gcc, as
  - source code management: make
  - troubleshooting: gdb and strace
  - library manipulation: ar, ranlib, nm

★ many other tools
Section II

Compilers and Linkers
Stages of Compilation

★ Pre-processing
★ Compiling
★ Assembling
★ Linking
Pre-processing

★ **cpp** is the gcc pre-processor

★ this step is usually skipped in modern systems development

★ options:
  ★ **-Wall**: flags all errors and warnings
  ★ **-IDIRECTORY**: flags a directory to search for header files
  ★ **-DMACRO**: defines a macro instruction
  ★ **-DMACRO=value**: defines a macro instruction and its value
  ★ **-M**: enables dependencies associated with files to be generated

★ examples:
  ★ `cpp example.c`
  ★ `gcc -E example.c`
Compile

★ **gcc** is the gcc C compiler
★ generates assembly code
★ options:
  ★ **-On**: level of optimization where n is from 0 (no optimization) to 3 (maximum optimization)
  ★ **-g**: used for debugging
  ★ **-Wall**: indicates all errors and warnings
  ★ **-Werror**: indicates all warnings as errors
★ examples:
  ★ gcc -S example.c
Assembly

★ this step is typically combined with the compile stage in modern systems

★ example: `as -o example.o example.s`

★ this example assembles example.s and generates object code example.o

★ example: `gcc -c example.c -o example.o`

★ this example compiles and assembles example.c and generates example.o
Linker

★ this is the final stage of compilation and is useful when dealing with multiple libraries and object files

★ linking is still done with gcc

★ example: gcc -o main example.o -lm

★ this example links the example.o object file and the libm.so library and produces the program main
Section III

Troubleshooting
The GNU Debugger

- useful tool for tracing program flow
- monitors variable and memory locations
- set breakpoint during runs
- make changes on run time
- example: `gdb main`
- this example starts the debugger on the program main
Commands

★ **help** - displays the gdb help screen

★ **set args** - sets the arguments to be passed to the program

★ **break** sets breakpoint on specified location

★ **run** - start program execution until a breakpoint or error occurs

★ **list** - lists the program code about the current program pointer location

★ **print** - displays the contents of a variable

★ **step** - executes the program one line at a time

★ **next** - same as step but functions are not stepped into

★ **continue** - continues program execution

★ **where** - finding where the function calls are located

★ **quit** - stops debugger
Strace

★ debugging and diagnostics tool
★ tracing system calls and signal
★ quick tool for running a specified applications until it exits
★ records system calls which are called by a process and the signals which are received by such process
Section IV

Source Code Management
GNU Make

★ is to determine automatically which pieces of a large program need to be recompiled, and issue the commands to recompile them

★ ideal for maintaining source code bases with a large number of files

★ looks for a file `makefile` or `Makefile` in the current directory
Example Makefile

CC=gcc
RM=/bin/rm

CFLAGS=-g -Wall

SRC=example.c extra.c extra2.c
OBJ=$(SRC:.c=.o)

PROG=main
LIB=-lm

$(PROG): $(OBJ)
$(CC) $(CFLAGS) $(OBJ) -o $(PROG) $(LIB)
clean:
$(RM) -f $(OBJ) $(PROG)

example.o : example.c example.h
extra.o : extra.c example.h
extra2.o : extra2.c example.h
Section V

Library Creation
lists symbols in an object file

useful for determining function in a library and their state

states are defined as:

- **U** undefined symbol
- **W** probably overloaded symbol
- **T** symbol defined normally

example: `nm /lib/libm.so.6`

this example lists all symbols in the libm.so.6 library
Static Libraries

★ linked with actual object files to generate executable during compile time
★ handled like object files on compile

1. compile and assemble code
   ★ example: gcc -c complex.c -o complex.o
   ★ creates an object file complex.o from source

2. insert object file into library or create library
   ★ example: ar cru libcomplex.a complex.o
   ★ creates a new library if not existent and appends to the existing library if existent

3. generate index to archive
   ★ example: ranlib libcomplex.a
   ★ generates the library index
Dynamic Libraries

- linked with actual program files during runtime
- enables other programs to share the same libraries
- also known as shared objects (so)

1. compile and assemble object code using the -fPIC option
   - example: gcc -c -fPIC complex.c -o complex.o
   - generates object code for creating a dynamic library

2. use link edits to generate library
   - example: gcc -shared -Wl -soname libcomplex.so.1.0 -o libcomplex.so.1.0 complex.o
   - links a dynamic library with proper name libcomplex.so.1 and physical name libcomplex.so.1.0
   - this is to allow the existence of multiple existence of these libraries
Copyright © 2000-2001 by William Emmanuel S. Yu. This material may be distributed only subject to the terms and conditions set forth in the Open Content License, v1.0 or later (the latest version is presently available at http://opencontent.org/opl.shtml).