gcc & gdb

Mohammed Junaid

GCC

- Introduction
- Commonly used GCC options
- Compiling C program
- Creating a shared object(.so file)
- Examining memory map
- Compiling Linux Kernel and running it in a Virtual Machine(at the end of presentation)

GCC

- The GNU Compile Collection (GCC) is a compiler system produced by the GNU Project supporting various programming languages.
- Latest version - GCC 4.8.0 released
- Languages supported: C, C++,Objective-C, Java, etc
- Architectures supported: Arm, x86_64, PowerPC, Sparc, etc
- Official site: http://gcc.gnu.org/

Source code to running program

- **C Preprocessor:** This stage processes all the preprocessor directives. Basically, any line that starts with a #, such as #define and #include.

- **Compilation Proper:** Once the source file has been preprocessed, the result is then compiled. Since many people refer to the entire build process as compilation, this stage is often referred to as "compilation proper". This stage turns a .c file into an .o (object) file.

Source code to Running program cont..

- **Linker:** Here is where all of the object files and any libraries are linked together to make final program. Now, you have a complete program that is ready to run. When you launch it from the shell, the program is handed off to the loader.

- **Loading:** This stage happens when program starts up. Program is scanned for references to shared libraries. Any references found are resolved and the libraries are mapped into the program.

Commonly Used GCC options

- -o: output file
- -static: Static compilation
- -shared: Produce a shared object
- -Wall: Enable all the warning
- -Werror: Make all warnings into hard errors
- -l: link a library
- -g: enable debugging
- -c: compile the source files, but do not link.
- -fPIC: Position independent code
- And others...

Compiling C program

- Command to compile a C program
  $ gcc demo.c -o demo

- A static linking is a compiled version of a program which has been statically linked against libraries.
  $ gcc -static demo.c -o demo-static

- file command to print the type of binary
  $ file demo

Shared Objects(.so)

- A Shared object is a binary file that contains a set of callable C functions. The file is designed so that the code is position-independent, meaning that the code may be loaded anywhere in memory.

- Steps to create a .so file
  - compiling with position independent code(pic)
    $ gcc -c -fPIC -o print.o print.c

  - Creating a shared object from an object file
    $ gcc -shared -o libprint.so print.o

Shared Objects(.so) cont..

- Linking the shared library

  $ gcc -o test test.c -lprint -L <pre>/shared/lib -I <pre>/shared/lib

- Making the library available runtime

  $export LD_LIBRARY_PATH=

  <pre>/share/lib:$LD_LIBRARY_PATH

- Run the program

  $ ./test

Examining memory map

- Print shared library dependencies
  $ ldd demo
  $ ldd demo-static

- Proc filesystem:
  $ less /proc/<pid>/maps
  $ less /proc/<pid>/smaps
  $ less /proc/<pid>/cmd

GDB - The GNU Debugger

- What is gdb?
- Compile and run program with gdb
- Basic gdb Commands
- Advanced gdb Commands
- Scripting with gdb
- Examining a Core file
- GDB with GNU-Emacs

What is gdb?

- “GNU Debugger”
- A debugger for several languages, including C and C++
- It allows you to inspect what the program is doing at a certain point during execution.
- Errors like segmentation faults may be easier to find with the help of gdb

Compile and run program with gdb

- -glevel: Produce debugging information in the OS native format(default is -g2)
- -g3: Produce debugging information for macros
- -O0: Unoptimize the binary
  -O2: Optimized binary code
- Run program with gdb
  $ gdb <binary>

Basic gdb commands

- For help, run the **help** command in the interactive gdb shell
  (gdb) help [<command>]

- To **run** the program, just use:
  (gdb) run

- You can single-step (execute just the next line of code) by typing "**step**." This gives you really fine-grained control over how the program proceeds.
  $ step [num]

- Similar to "step," the "**next**" command single-steps as well, except this one doesn't execute each line of a subroutine, it just treats it as one instruction
  (gdb) next [num]

  This brings us to the next set of commands. . .

Basic gdb commands cont..

- **Breakpoints** can be used to stop the program run in the middle, at a designated point. The simplest way is the command “break.” This sets a breakpoint at a specified file-line pair:
  (gdb) break file1.c:6

  You can also tell gdb to break at a particular function. Suppose you have a function my_func:

  int my_func(int a, char *b);

  You can break anytime this function is called:

  (gdb) break my_func

- **continue:** Once you have set a breakpoint, you can try using the run command again. This time, it should stop where you tell it to (unless a fatal error occurs before reaching that point).
  You can proceed onto the next breakpoint by typing “continue”
  (gdb) continue

Basic gdb commands cont..

- The print command prints the value of the variable specified, and print/x prints the value in hexadecimal:
  (gdb) print my_var
  (gdb) print/x my_var

- Other useful commands
  - backtrace: produces a stack trace of the function calls
  - finish: runs until the current function is finished
  - delete: deletes a specified breakpoint
  - info breakpoints: shows information about all declared breakpoints
  - start: similar to run, but implicitly adds a breakpoint on main function
  - file: load a binary to run with gdb
  - list: list specified function or line.

Advanced gdb commands

- set -> Evaluate expression EXP and assign result to variable VAR
- info frame -> Print information on the current frame
- x -> examine memory
- disassemble -> disassemble a section of memory
- macro -> command dealing with C preprocessor
- Thread - info, switch, examine
- Examining pointers and structures
- Attaching gdb to a running process

Scripting with GDB

- Yes, that's right. Like shell scripting, we can write scripts to run under gdb
- Contents of script - list of gdb commands one per line.
- pass the script to gdb
  $ gdb -x <script> <binary>

Examining a Core file

- Segmentation Fault ??
- Enable core dump on your machine
  $ ulimit -c unlimited
- Attach the core to gdb
  $ gdb <binary> <core-file>

GDB with GNU-Emacs

Compiling Linux Kernel and running it in a Virtual Machine

- Steps to compile a kernel
  $ make defconfig or make menuconfig
  $ make
  $ make modules_install
  $ make install

- Print the version of the Kernel built
  $ make kernelversion

- Command to generate initrd:
  $ mkinitrd <initrd-name> <kernel-version>

Questions??