Why Study GCC

Understand how machine code gets generated

Source code-based program analysis
- Compiler parses the source code
- Compiler knows exactly the behavior of the program
- Compiler has to know the machine details (when generating the code)
- Compiler-based security solutions
- Source code auditing (Integer overflow, buffer overflow, format string)

Understanding the tool-chain
- Compiler, linker, loader
Overview Internals, Architecture USAGE DEMO Summary

Understanding the tool-chain

GNU Compiler Collection – GCC

- **Developer(s)** GNU Project
- **Initial release** May 23, 1987
- **Stable release** 4.6.3 / March 1, 2012; 19 days ago
- **Written** in C, C++
- **Operating system** Cross-platform
- **Platform** GNU
- **Type** Compiler
- **License** GNU General Public License (version 3 or later)
- **Website** gcc.gnu.org

History of GCC

- **GCC 1.xx**
- **GCC 2.xx**
- **EGCS**
  - Enhanced (Experimental) GNU Compiler System
  - Fortran, Chill front-end.
  - C and C++ standard libraries.
    - libc, stdlibc++
  - Java, Ada, more optimization.
- **GCC 4.0**, Apr. 2005
- **GCC 4.6.3**, March 2012

**History of GCC (cont’d)**

- The newest version is not the best.
  - Compiler needs correct, stable and fast code.
- more stable and popular version:
  - gcc 2.6.3, 2.7.2.3
  - egcs-1.1.2
  - gcc 2.95.3
  - gcc 3.3
To prevent any particular individual, group or organization from getting control over the project.

13 members
1. Joe Buck (Synopsys)
2. David Edelsohn (IBM)
3. Kaveh R. Ghazi
4. Jeffrey A. Law (Red Hat)
5. Marc Lehmann (nethype GmbH)
6. Jason Merrill (Red Hat)
7. David Miller (Red Hat)
8. Mark Mitchell (CodeSourcery / Mentor Graphics)
9. Toon Moene (Koninklijk Nederlands Meteorologisch Instituut)
10. Gerald Pfeifer (SUSE)
11. Joel Sherrill (OAR Corporation)
12. Jim Wilson (Cisco)
13. Richard Stallman (Free Software Foundation)
Code Size of GCC

- For GCC 2.7.2:
  - Distributed front ends: 109,380 lines.
    - C: 22,415
    - C++: 72,112
    - Objective-C: 7,973 + 6,880 (library)
  - Base compiler: 226,057 lines.
  - Optimizer: 56,581
  - Total size: 566,556 lines.
- GCC 3.2: 133Mbytes

Compilation System

- Compilation system includes the phases
  - Preprocessor
  - Compiler
  - Optimizer
  - Assembler
  - Linker

- Compiler Driver coordinates these phases.
The Structure of GCC

Outline

NAME
gcc - GNU project C and C++ compiler

SYNOPSIS
Options

--help Display this information
(Use '-v --help' to display command line options of sub-processes)
--dumpspecs Display all of the built in spec strings
--dumpversion Display the version of the compiler
--dumpmachine Display the compiler's target processor
--print-search-dirs Display the directories in the compiler's search path
--print-libgcc-file-name Display the name of the compiler's companion library
--print-prog-name=<prog> Display the full path to compiler component <prog>
--print-multi-directory Display the root directory for versions of libgcc
--print-multi-lib Display the mapping between command line options and multiple library search directories
-Wa,<options> Pass comma-separated <options> on to the assembler
-Wp,<options> Pass comma-separated <options> on to the preprocessor
-Wl,<options> Pass comma-separated <options> on to the linker
-Wl <arg> Pass <arg> on to the linker
--save-temps Do not delete intermediate files
-std=<standard> Assume that the input sources are for <standard>
-B <directory> Add <directory> to the compiler's search paths
-b <machine> Run gcc for target <machine>, if installed
-V <version> Run gcc version number <version>, if installed
-g Display the programs invoked by the compiler
-x <language> Specify the language of the following input files
Ex: gcc -x java test.java
-O0~-O1: optimization level.
default is -O1.
-S: output assembly.
-E: preprocessor output.
-g: debug information.
-std: : optimization level.
-x <language>: input file's language.
Ex: gcc -x java test.java
-S: output assembly.
-m???: special for target's option.
  - Ex1: gcc -m68000 test.c
  - Ex2: gcc -mcpu=i686 test.c
-ldir: header files will be searched in dir.
  - Ex: gcc -I/home/pschen/include test.c
-da: output all RTL files.

At the prompt, type
```
gcc -ansi -Wall pgm.c
```
where pgm.c is the C program source file.
-ansi is a compiler option that tells the compiler to adhere to the ANSI C standard.
-Wall is an option to turn on all compiler warnings (best for new programmers).

The Result: a.out
If there are no errors in pgm.c, this command produces an executable file, which is one that can be executed (run).
The gcc compiler names the executable file a.out.
To execute the program, at the prompt, type ./a.out
Although we call this process "compiling a program", what actually happens is more complicated.
Using GCC

Editor

Source File pgm.c

Preprocessor

Modified Source Code in RAM

Compiler

Program Object Code File pgm.o

Other Object Code Files (if any)

Linker

Executable File a.out

Preprocessing

- Performed by a program called the **preprocessor**
- Modifies the source code (in RAM) according to the **preprocessor directives (preprocessor commands)** embedded in the source code
- Strips comments and white space from the code
- The source code as stored on disk is **not** modified.

```
#include <stdio.h>
int main()
{
 printf("pid=%d\n",getpid());
 return 0;
}
```

gcc -E pgm.c >pgm.i

Compilation

- Performed by a program called the **compiler**
- Translates the preprocessor-modified source code into **object code (machine code)**
- Checks for **syntax errors** and **warnings**
- Saves the object code to a disk file, if instructed to do so (we will not do this).
- If any compiler errors are received, no object code file will be generated.
- An object code file will be generated if only warnings, not errors, are received.

```
#include <stdio.h>
int main()
{
 printf("pid=%d\n",getpid());
 return 0;
}
```
Linking

- Combines the program object code with other object code to produce the executable file.
- The other object code can come from the Run-Time Library, other libraries, or object files that you have created.
- Saves the executable code to a disk file. On the Linux system, that file is called `a.out`.
  - If any linker errors are received, no executable file will be generated.

**Example Usage**

```bash
gcc -v

gcc -v -o test0 test0.c
```

```
Reading specs from /usr/lib/gcc-lib/i386-redhat-linux/3.2.3/specs
Configured with: ../configure –prefix=/usr –mandir=/usr/share/man
   –infodir=/usr/share/info –enable-shared –enable-threads=posix
   –disable-checking –with-system-zlib –enable-__cxa_atexit –host=i386-redhat-linux
Thread model: posix
```

```
gcc version 3.2.3 20030502 (Red Hat Linux 3.2.3-20)
/usr/lib/gcc-lib/i386-redhat-linux/3.2.3/cc1 -lang-c -v -D__GNUC__=3
-D__GNUC_MINOR__=2 -D__GNUC_PATCHLEVEL__=3
-D__GXX_ABI_VERSION=102 -D_ELFFUNC -Dunix -D__gnu_linux__ -Dlinux
-D__ELF__ -D__unix__ -D__gnu_linux__ -D__linux
-Asystem=posix -D__NO_INLINE__ -D__STDC_HOSTED__=1 -Acpu=i386
-Amachine=i386 -Dقيقة -D__NO_INLINE__ -D_STDC_HOSTED__=1 -Acpu=i386
-Amachine=i386 -Dقيقة -D__NO_INLINE__ -D_STDC_HOSTED__=1 -Acpu=i386

GNU assembler version 2.14.90.0.4 (i386-redhat-linux) using BFD version 2.14.90.0.4
```

```
20030523
/usr/lib/gcc-lib/i386-redhat-linux/3.2.3/collect2 –eh-frame-hdr -m elf_i386
-dynamic-linker /lib/ld-linux.so.2 -o test0
```

```
GNU C version 3.2.3 20030502 (Red Hat Linux 3.2.3-20) (cpplib) (i386 Linux/ELF)
```

```
GNU CPP version 3.2.3 20030502 (Red Hat Linux 3.2.3-20) (cpplib) (i386 Linux/ELF)
```

```
compiled by GNU C version 3.2.3 20030502 (Red Hat Linux 3.2.3-20).
```

```
pgm.i
1 .file "pgm.c"
2 .section .rodata
3 .LC0:
4 .string "pid=%d
"
5 .text
6 .globl main
7 .type main, @function
8 main:
9 .LFB0:
10 .cfi_startproc
11 pushq %rbp
12 .cfi_def_cfa_offset 16
13 movq %rsp, %rbp
14 .cfi_offset 6, -16
15 .cfi_def_cfa_register 6
16 movl $0, %eax
17 call getpid
18 movl %eax, %edx
19 movl $.LC0, %eax
20 movl %edx, %esi
21 movq %rax, %rdi
22 movl $0, %eax
23 call printf
24 movl $0, %eax
25 leave
26 .cfi_def_cfa_register 6
27 ret
28 .cfi_endproc
29 .LFE0:
30 .size main, .-main
31 .ident "GCC: (Ubuntu/Linaro 4.5.2-8ubuntu4) 4.5.2"
```
**Overview**

**Internals, Architecture**

**USAGE**

**DEMO**

**Summary**

---

Compiler is a key system component

GCC

- Overview
- Architecture
- USAGE
## References

- [http://vmakarov.fedorapeople.org/spec/comparison.html](http://vmakarov.fedorapeople.org/spec/comparison.html)