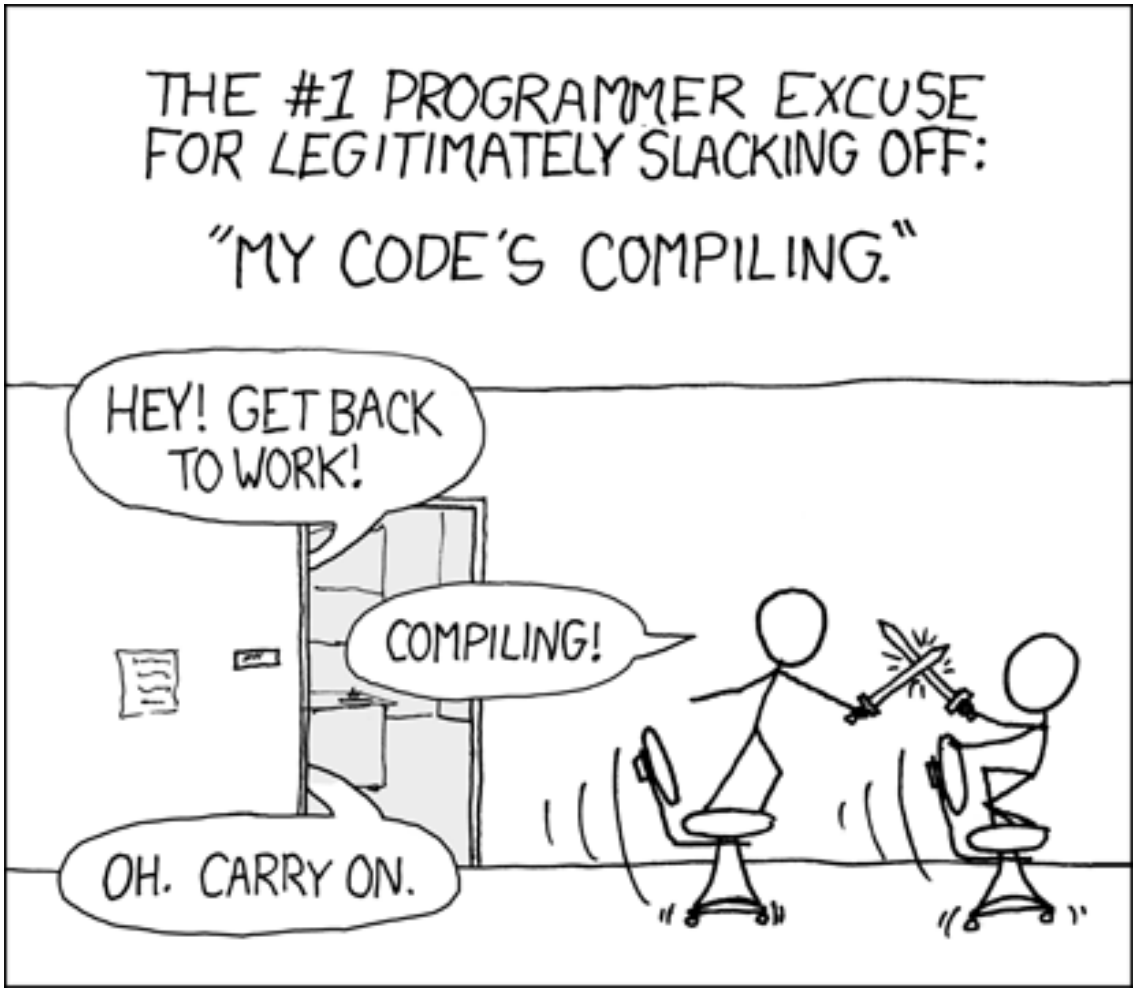
COMP20200 Unix Programming

Lecture 08

CS, University College Dublin, Ireland



http://xkcd.com/303/

Compilation process in GCC

- 4 phases of compilation:
  - Preprocessing
  - Compilation
  - Assembly
  - Linking

```
#include <stdio.h>
#define STRING "Hello World"

int main (void) {
/* Example program */
int var = 8;
printf ("%s\n", STRING);
return 0;
}
```

Compilation process in GCC: The Preprocessor

- Text Substitution
- Stripping of Comments
- File Inclusion
- Lines beginning with "#" are preprocessor directives
- Stand alone tool: C preprocessor `cpp`
- Default for processed code: `*.i`

```
$ cpp hello.c > hello.i
```

or

```
$ gcc -E hello.c -o hello.i
```

More than 900 lines code for hello.c, selection on next slide.

More than 900 lines code for hello.c. Selection:

```
# 1 "hello.c"
# 1 "<command-line>"
# 1 "hello.c"
# 1 "/usr/include/stdio.h" 1 3 4
# 28 "/usr/include/stdio.h" 3 4
# 1 "/usr/include/features.h" 1 3 4
__extension__ typedef unsigned int __uid_t;
# 267 "/usr/include/stdio.h" 3 4
extern FILE *fopen (__const char *__restrict __filename,
      __const char *__restrict __modes) ;
extern int printf (__const char *__restrict __format, ...);
# 940 "/usr/include/stdio.h" 3 4
# 2 "hello.c" 2
  int main (void) {

  printf ("%s\n","Hello World");
  return 0;
  }
```

Compilation process in GCC: The Preprocessor

Use

```
$ gcc -E -P hello.c -o hello.i
```

to get rid of lines for debugger

Preprocessor Directives

Directive:	Description:
`#include`	Inserts a particular header from another file
`#define`	Defines a preprocessor macroname
`#undef`	Undefines a preprocessor macro
`#ifdef`	Returns true if this macro is defined
`#ifndef`	Returns true if not defined
`#if`	Tests compile time condition
`#else`	Alternative for #if
`#elif`	#else an #if in one statement
`#endif`	Ends #if
`#error`	Prints error message on stderr
`#pragma`	Issues special commands to the compiler.

Preprocessor Example

```
#include <stdio.h>

#define DEBUG

#define SUM(a,b) (a + b)

int main(){
#ifdef DEBUG
   printf("Debug message!\n");
#endif
   printf("Sum:%d\n", SUM(4, 3));
   return 0;
}
```

Compilation process in GCC: The Compiler

- Compile processed code into assembly for a specific processor.
- Can have a different target architecture than current machine: cross-compilation

```
gcc -Wall -S hello.i -o hello.s
```

- Assembly `hello.s` on next slide

```
    .file   "hello.c"
    .section    .rodata
.LC0:
    .string "Hello World"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushl   %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl    %esp, %ebp
    .cfi_def_cfa_register 5
    andl    $-16, %esp
    subl    $16, %esp
    movl    $.LC0, (%esp)
    call    puts
    movl    $0, %eax
    leave
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
```

Compilation process in GCC: The Assembler

- Converts assembly code into machine code.
- Assembler was one of first software tools developed after the invention of computer.
- Assembler leaves the addresses of the external functions undefined.
  - Filled in later by the Linker.
- as is assembler invoked by gcc creates .o object files.

```
$ as hello.s -o hello.o
```

or

```
$ gcc -c hello.c
```

```
$ hexdump hello.o
0000000 457f 464c 0101 0001 0000 0000 0000 0000
0000010 0001 0003 0001 0000 0000 0000 0000 0000
0000020 0120 0000 0000 0000 0034 0000 0000 0028
0000030 000d 000a 8955 83e5 f0e4 ec83 c710 2404
0000040 0000 0000 fce8 ffff b8ff 0000 0000 c3c9
0000050 6548 6c6c 206f 6f57 6c72 0064 4700 4343
0000060 203a 5528 7562 746e 2f75 694c 616e 6f72
0000070 3420 372e 322e 322d 6275 6e75 7574 2931
0000080 3420 372e 322e 0000 0014 0000 0000 0000
0000090 7a01 0052 7c01 0108 0c1b 0404 0188 0000
00000a0 001c 0000 001c 0000 0000 0000 001c 0000
00000b0 4100 080e 0285 0d42 5805 0cc5 0404 0000
00000c0 2e00 7973 746d 6261 2e00 7473 7472 6261
00000d0 2e00 6873 7473 7472 6261 2e00 6572 2e6c
00000e0 6574 7478 2e00 6164 6174 2e00 7362 0073
00000f0 722e 646f 7461 0061 632e 6d6f 656d 746e
0000100 2e00 6f6e 6574 472e 554e 732d 6174 6b63
0000110 2e00 6572 2e6c 6865 665f 6172 656d 0000
0000120 0000 0000 0000 0000 0000 0000 0000 0000
*
0000140 0000 0000 0000 0000 001f 0000 0001 0000
0000150 0006 0000 0000 0000 0034 0000 001c 0000
0000160 0000 0000 0000 0000 0004 0000 0000 0000
```

Compilation process in GCC: The Linker

- Links object files and libraries.
- Resolve dependencies
- Puts actual addresses of functions
- Adds code for setting up environment, command-line arguments and return values.

```
$ ld -dynamic-linker hello.o -L/usr/lib/gcc/i686-linux-gnu/4.4/
                                        -lgcc <and more>
```

or

```
$ gcc -o hello hello.o
```

Libraries in C

- Libraries are files of ready-compiled code.
- Some provided as standard: `libc`, `libm`
- Or you can write your own: libfoo.a
- Standard C library `glibc` is linked automatically by `gcc`
- Other libs must be explicitly linked, eg. maths library libm.so

```
gcc program.c -lm
```

- `ldd` to list linked libs.
- `nm` list symbols from object files.

Static Libraries

- Linked at compile time and becomes part of binary executable.
- An archive of object files.
- `lib*.a`

```
$ gcc -c foo_src1.c foo_src2.c
$ ar rcs libfoo.a foo_src1.o foo_src2.o
$ gcc -o foobar bar.c -L/path/to/library-directory -lfoo
$
```

Shared Libraries

- Also known as dynamically linked libraries
- `lib*.so`
- Not included in executable
- Loaded at execution initialization or as a module during execution

```
$ gcc -c -fPIC src1.c src2.c
$ gcc -shared -o libfoo.so  src1.o src2.o
```

Install into /usr/local/lib:

```
$ mv libfoo.so /usr/local/lib
```

Dynamically link:

```
$ gcc -o foobar bar.c -L/usr/local/lib -lfoo
```