Compilation, Linking, Loading

Abhijit A M

Review of last few lectures

Boot sequence: BIOS, boot-loader, kernel

Boot sequence: Process world

kernel->init -> many forks+execs() -> ....

Hardware interrupts, system calls, exceptions

Event driven kernel

System calls

Fork, exec, ... open, read, ...

What are compiler, assembler, linker and loader, and C library

System Programs/Utilities

Most essential to make a kernel really usable

Standard C Library

A collection of some of the most frequently needed functions for C programs

scanf, printf, getchar, system-call wrappers (open, read, fork, exec, etc.), ...

An machine/object code file containing the machine code of all these functions

Not a source code! Neither a header file. More later.

Where is the C library on your computer?

/usr/lib/x86_64-linux-gnu/libc-2.31.so

Compiler

application program, which converts one (programming) language to another

Most typically compilers convert a high level language like C, C++, etc. to Machine code language

E.g. GCC /usr/bin/gcc

Usage: e.g.

$ gcc main.c -o main

Here main.c is the C code, and "main" is the object/machine code file generated

Input is a file and output is also a file.

Other examples: g++ (for C++), javac (for java)

Assembler

application program, converts assembly code into machine code

What is assembly language?

Human readable machine code language.

E.g. x86 assembly code

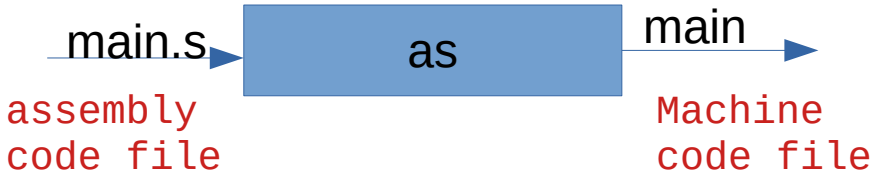
mov 50, r1

add 10, r1

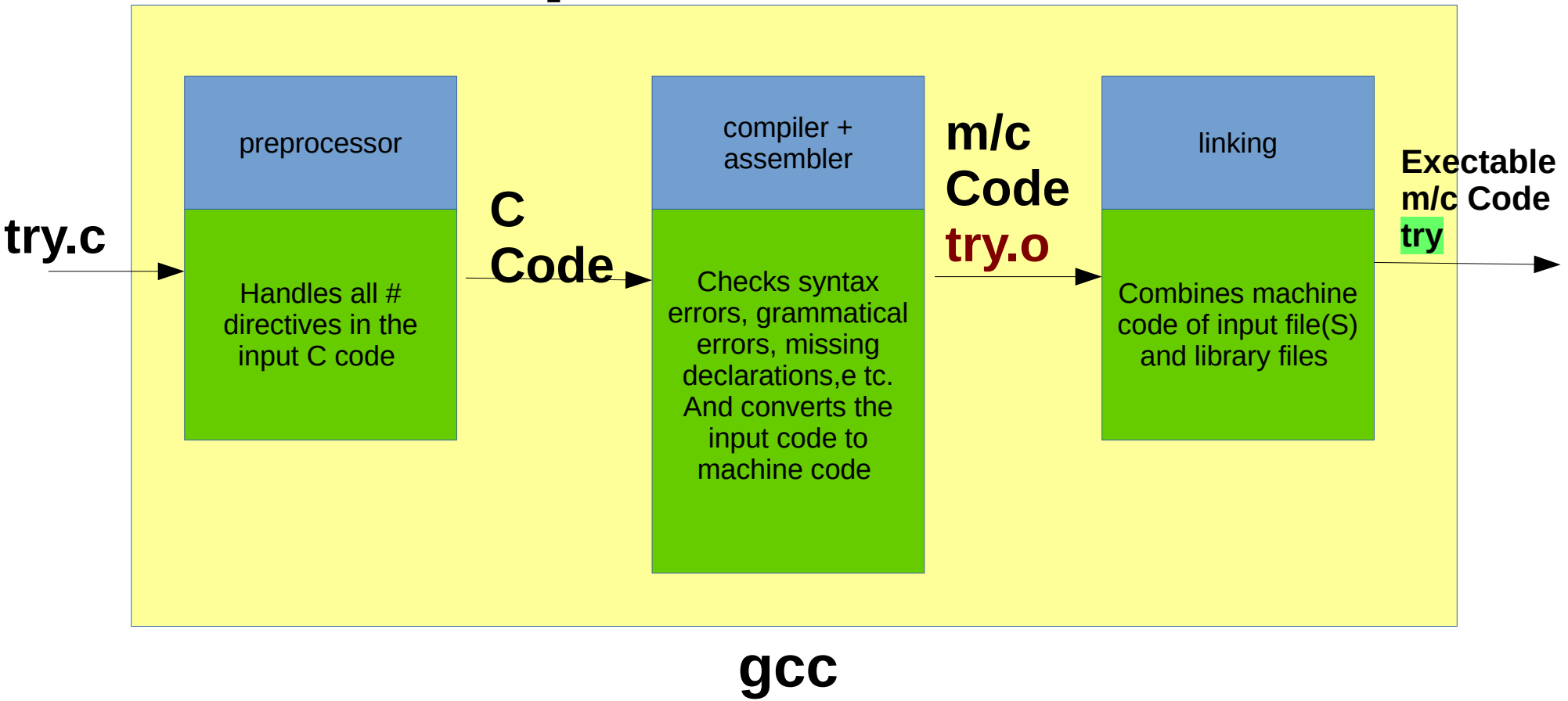
mov r1, 500

Usage. eg..

$ as something.s -o something



Compilation Process



Example

try.c

```
#include <stdio.h>
#define MAX 30
int f(int, int);
int main() {
        int i, j, k;
        scanf("%d%d", &i, &j);
        k = f(i, j) + MAX;
        printf("%d\n", k);
        return 0;
}
```

f.c

```
int g(int);
#define ADD(a, b) (a + b)
int f(int m, int n) {
        return ADD(m,n) + g(10);
}
```

g.c

```
int g(int x) {
        return x + 10;
}
```

Try these commands, observe the output/errors/warnings, and try to understand what is happening

```
$ gcc try.c
$ gcc -c try.c
$ gcc -c f.c
$ gcc -c g.c
$ gcc try.o f.o g.o -o try
$ gcc -E try.c
$ gcc -E f.c
```

More about the steps

Pre-processor

#define ABC XYZ

cut ABC and paste XYZ

include <stdio.h>

copy-paste the file stdio.h

There is no CODE in stdio.h, only typedefs, #includes, #define, #ifdef, etc.

Linking

Normally links with the standard C-library by default

To link with other libraries, use the -l option of gcc

```
cc main.c -lm -lncurses -o main  # links with libm.so and libncurses.so
```

Using gcc itself to understand the process

Run only the preprocessor

cc -E test.c

Shows the output on the screen

Run only till compilation (no linking)

cc -c test.c

Generates the “test.o” file , runs compilation + assembler

gcc -S main.c

One step before machine code generation, stops at assembly code

Combine multiple .o files (only linking part)

cc test.o main.o try.o -o something

Linking process

Linker is an application program

On linux, it's the "ld" program

E.g. you can run commands like $ ld a.o b.o -o c.o

Normally you have to specify some options to ld to get a proper executable file.

When you run gcc

$ cc main.o f.o g.o -o try

the CC will internally invoke "ld" . ld does the job of linking

The resultatnt file "try" here, will contain the codes of all the functions and linkages also.

What is linking?

"connecting" the call of a function with the code of the function.

What happens with the code of printf()?

The linker or CC will automatically pick up code from the libc.so.6 file for the functions.

Executable file format

An executable file needs to execute in an environment created by OS and on a particular processor

Contains machine code + other information for OS

Need for a structured-way of storing machine code in it

Different OS demand different formats

Windows: PE, Linux: ELF, Old Unixes: a.out, etc.

ELF : The format on Linux.

Try this

$ file /*bin/l*s

$ file /usr/lib/x86_64-linux-gnu/libc-2.31.so

$ file a.out # on any a.out created by you

Exec() and ELF

When you run a program

$ ./try

Essentially there willl be a fork() and exec("./try", ...")

So the kernel has to read the file "./try" and understand it.

So each kernel will demand it's own object code file format.

Hence ELF, EXE, etc. Formats

ELF is used not only for executable (complete machine code) programs, but also for partially compiled files e.g. main.o and library files like libc.so.6

What is a.out?

"a.out" was the name of a format used on earleir Unixes.

It so happened that the early compiler writers, also created executable with default name 'a.out'

Utilities to play with object code files

objdump

$ objdump -D -x /bin/ls

Shows all disassembled machine instructions and “headers”

hexdump

$ hexdump /*bin*/ls

Just shows the file in hexadecimal

readelf

Alternative to objdump

ar

To create a “statically linked” library file

$ ar -crs libmine.a one.o two.o

Gcc to create shared library

$ gcc hello.o -shared -o libhello.so

To see how gcc invokes as, ld, etc; do this

$ gcc -v hello.c -o hello

/* https://stackoverflow.com/questions/1170809/how-to-get-gcc-linker-command */

Linker, Loader, Link-Loader

Linker or linkage-editor or link-editor

The “ld” program. Does linking.

Loader

The exec(). It loads an executable in the memory.

Link-Loader

Often the linker is called link-loader in literature. Because where were days when the linker and loader’s jobs were quite over-lapping.

Static, dynamic / linking, loading

Both linking and loading can be

Static or dynamic

More about this when we learn memory management

An important fundamental:

memory management features of processor, memory management architecture of kernel, executable/object-code file format, output of linker and job of loader, are all interdependent and in-separable.

They all should fit into each other to make a system work

That’s why the phrase “system programs”

Cross-compiler

Compiler on system-A, but generate object-code file for system-B (target system)

- E.g. compile on Ubuntu, but create an EXE for windows

Normally used when there is no compiler available on target system

- see gcc -m option

See https://wiki.osdev.org/GCC_Cross-Compiler