Linux System Programming

张海宁

贵州大学

hnzhang1@gzu.edu.cn

April 4, 2018

Overview

1. System Programming

2. GCC
    - Using GCC
    - 库文件
    - 补充工具

3. make

4. GDB

System Programming

System programming is the art of writing system software.System software lives at a low level, interfacing directly with the kernel and core system libraries.

There are three cornerstones to system programming in Linux:

1. system calls
2. the C library
3. the C compiler

System Calls

System programming starts with system calls.System calls (often shorted to syscalls) are function invocations made from user space—your text editor, favorite game, and so on—into the kernel (the core internals of the system) in order to request some service or resource from the operating system.

Invoke system calls

It is not possible to directly link user-space applications with kernel space. For reasons of security and reliability, user-space applications must not be allowed to directly execute kernel code or manipulate kernel data. Instead, the kernel must provide a mechanism by which a user-space application can ”signal” the kernel that it wishes to invoke a system call. The application can then trap into the kernel through this well-defined mechanism, and execute only code that the kernel allows it to execute. The exact mechanism varies from architecture to architecture.

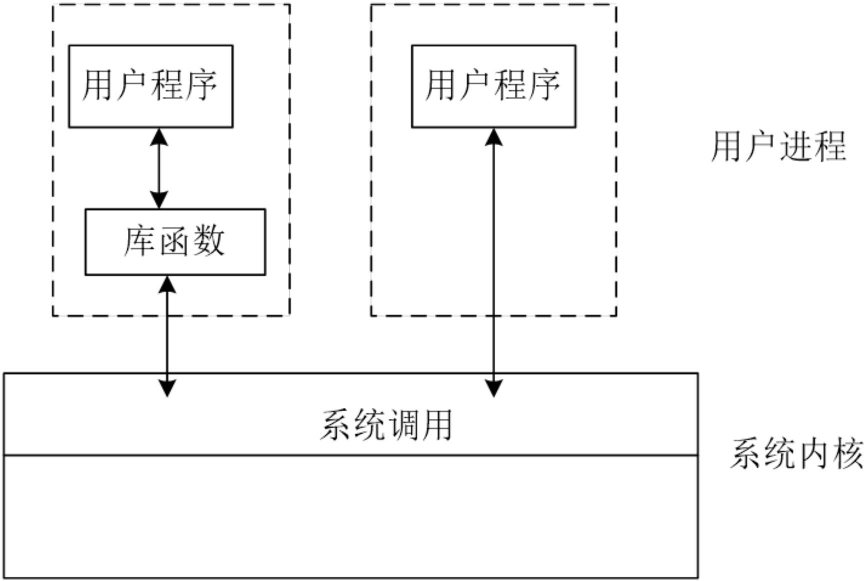
The C Library

The C library (libc) is at the heart of Unix applications. Even when you're programming in another language, the C library is most likely in play, wrapped by the higher-level libraries, providing core services, and facilitating system call invocation. On modern Linux systems, the C library is provided by GNU libc, abbreviated glibc, and pronounced gee-lib-see or, less commonly, glib-see.
The GNU C library provides more than its name suggests. In addition to implementing the standard C library, glibc provides wrappers for system calls, threading support, and basic application facilities.

The C Compiler

In Linux, the standard C compiler is provided by the GNU Compiler Collection (gcc). Originally, gcc was GNU's version of cc, the C Compiler. Thus, gcc stood for GNU C Compiler. Over time, support was added for more and more languages. Consequently, nowadays gcc is used as the generic name for the family of GNU compilers. However, gcc is also the binary used to invoke the C compiler. In this course, when we talk of gcc, we typically mean the program gcc, unless context suggests otherwise.

用户程序 vs 库函数 vs 系统调用



GCC

The GNU Compiler Collection includes front ends for C, C++, Objective-C, Fortran, Ada, and Go, as well as libraries for these languages (libstdc++,...). GCC was originally written as the compiler for the GNU operating system.

The GNU system was developed to be 100% free software, free in the sense that it respects the user's freedom.

The four essential freedoms

A program is free software if the program's users have the four essential freedoms:

- The freedom to run the program as you wish, for any purpose (freedom 0).
- The freedom to study how the program works, and change it so it does your computing as you wish (freedom 1). Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help others (freedom 2).
- The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.

`https://www.gnu.org/philosophy/free-sw.html`

GNU Toolchain

GCC is a key component of "GNU Toolchain", for developing applications, as well as operating systems. The GNU Toolchain includes:

- GNU Compiler Collection (GCC): a compiler suit that supports many languages, such as C/C++ and Objective-C/C++
- GNU Make: an automation tool for compiling and building applications
- GNU Binutils: a suit of binary utility tools, including linker and assembler
- GNU Debugger (GDB)
- GNU Autotools: A build system including Autoconf, Autoheader, Automake and Libtool
- GNU Bison: a parser generator (similar to lex and yacc)

Platform

GCC is portable and run in many operating platforms.
GCC (and GNU Toolchain) is currently available on all Unixes. They are also ported to Windows (by MinGW and Cygwin). GCC is also a cross-compiler, for producing executables on different platform.

Install gcc

- Unix/Linux/Mac GCC(GNU Toolchain) is included.
- Windows
  - Cygwin GCC: Cygwin is a Unix-like environment and command-line interface for Microsoft Windows. Cygwin is huge and includes most of the Unix tools and utilities. It also included the commonly-used Bash shell.
  - MinGW: MinGW (Minimalist GNU for Windows) is a port of the GNU Compiler Collection (GCC) and GNU Binutils for use in Windows. It also included MSYS (Minimal System), which is basically a Bourne shell (bash).

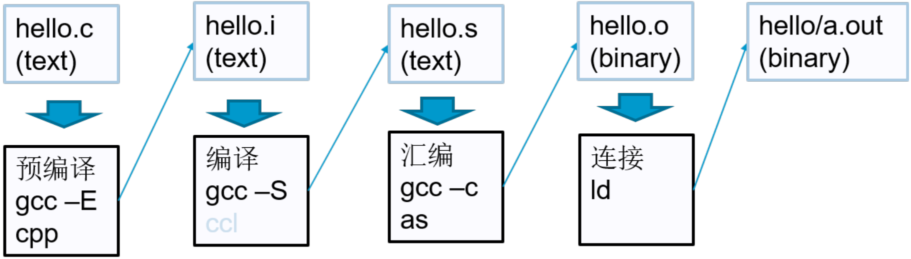
Versions and Help

We can display the version of GCC with –version/-v option, and get help with –help option:

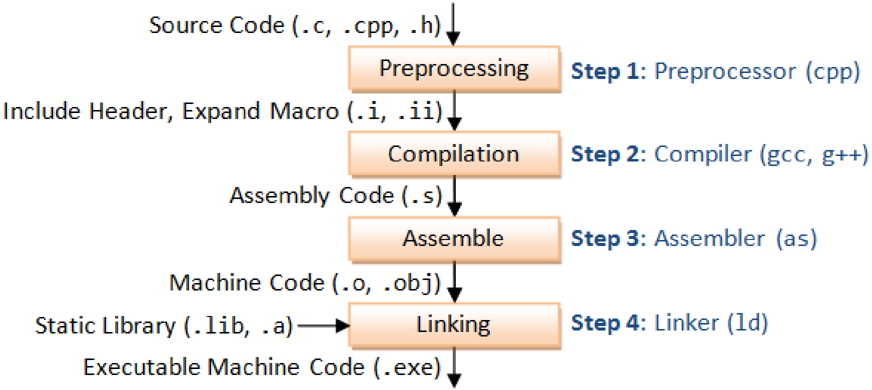
Example (show version and help of GCC)

```
$ gcc -v
$ gcc --version
$ gcc --help
```

编写 c 程序的流程


hello.c
(text)
预编译
gcc –E
cpp
hello.i
(text)
编译
gcc –S
ccl
hello.s
(text)
汇编
gcc –c
as
hello.o
(binary)
连接
ld
hello/a.out
(binary)

编写 c 程序的流程



Source Code (.c, .cpp, .h)

Preprocessing

Step 1: Preprocessor (cpp)

Include Header, Expand Macro (.i, .ii)

Compilation

Step 2: Compiler (gcc, g++)

Assembly Code (.s)

Assemble

Step 3: Assembler (as)

Machine Code (.o, .obj)

Static Library (.lib, .a)

Linking

Step 4: Linker (ld)

Executable Machine Code (.exe)

hello.c

Example

```
// hello.c
#include <stdio.h>

int main(){
        printf("Hello, C.\n");
        return 0;
}
```

从.c 文件到可执行文件

预处理

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

编译

```
gcc -S hello.i
```

汇编

```
gcc -c hello.s -o hello.o
```

链接

```
gcc hello.o -o hello.out
```

all in one

```
gcc hello.c -o hello.out
```

静态库和动态库

库文件是一组被预先编译的目标文件的集合，可以被链接到用户所编写的程序中。

- 静态库
  通常以“.a”结尾 (archive file)，静态库的代码在编译时就会连接到用户的程序中
- 动态库
  通常以“.so”结尾 (shared objects)，用户的程序在运行时，按需要载入

几个有用的工具

1. file
   判断文件的类型
2. nm
   显示目标文件的符号表，通常用于查看目标文件中是否定义了某个特定的函数
3. ldd
   检查一个可执行文件并显示出其需要的动态库文件
4. mtrace
   内存溢出检测

make

依照 makefile 文件，make 程序可以自动确定一个软件包的哪些部分需要重新编译。
makefile 文件描述了建立可执行程序的一些规则。

makefile

hello.c

```c
// hello.c
#include <stdio.h>

int main(){
    printf("Hello,C.\n");
    return 0;
}
```

makefile

```make
all:hello

hello:hello.o
	gcc -o hello hello.o

hello.o:hello.c
	gcc -c hello.c
clean:
	rm hello.o
```

makefile 的规则

```
target: pre-req-1 pre-req-2 ...
^^Icommand
```

The target and pre-requisites are separated by a colon (:). The command must be preceded by a tab (NOT spaces).

make 和 makefile

```
$ make
gcc -c hello.c
gcc -o hello hello.o
$ make clean
rm hello.o
$ ls
hello  hello.c  makefile
```

修改源文件与 make 的关系

```
$ make
gcc -c hello.c
gcc -o hello hello.o
$ make
make: Nothing to be done for 'all'.
$ vim hello.c
$ make
gcc -c hello.c
gcc -o hello hello.o
$ ./hello
Hello, C.
test make.
```

GDB

GDB(GNU Source-Level Debugger)，是 GNU 的一个命令行调试工具。它主要可以完成以下功能：

- 设置运行环境和参数运行指定程序
- 让程序在指定的条件下停止
- 当程序停止时，检查发生了什么
- 改变正在调试的程序，以修正某个 bug，然后继续调试

gdb 部分命令

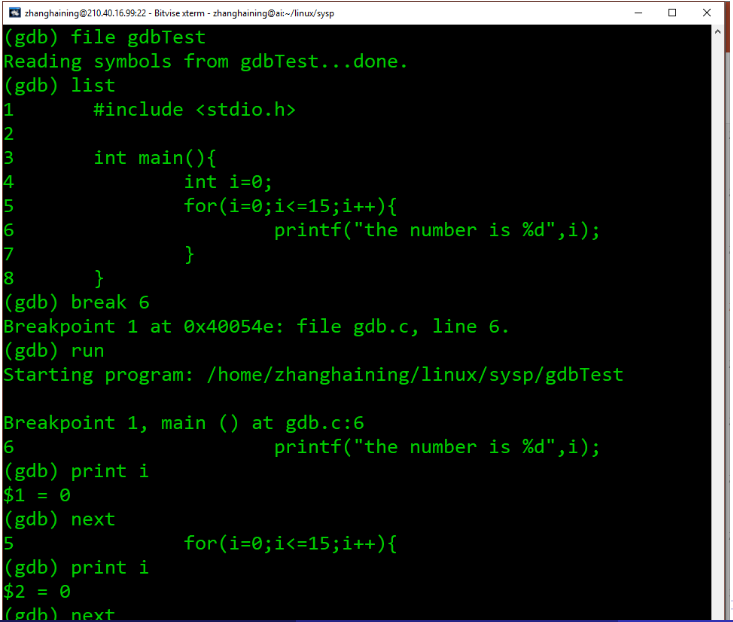
命令	作用	命令	作用
file	载入可执行文件	list	显示源代码
run	执行	info local	显示当前函数中变量值
kill	停止执行	info break	显示断点列表
break	设置断点	info func	显示所有函数名
delete	删除断点	watch	监视表达式的变化
print	显示表达式的值	next	执行下一条代码，但不进入
shell 命令	执行 shell 命令	step	执行下一条代码，进入函数

Table: GDB 部分命令

gdb 示例

```
$ vim gdb.c
$ cat gdb.c
#include <stdio.h>
int main(){
        int i=0;
        for(i=0;i<=15;i++){
                printf("the number is %d",i);
        }
}
$ gcc -ggdb -o gdbTest gdb.c
```

gdb 示例



zhanghaining@210.40.16.99:22 - Bitvise xterm - zhanghaining@ai:~/linux/sysp

```
(gdb) file gdbTest
Reading symbols from gdbTest...done.
(gdb) list
1       #include <stdio.h>
2
3       int main(){
4               int i=0;
5               for(i=0;i<=15;i++){
6                       printf("the number is %d",i);
7               }
8       }
(gdb) break 6
Breakpoint 1 at 0x40054e: file gdb.c, line 6.
(gdb) run
Starting program: /home/zhanghaining/linux/sysp/gdbTest

Breakpoint 1, main () at gdb.c:6
6                       printf("the number is %d",i);
(gdb) print i
$1 = 0
(gdb) next
5               for(i=0;i<=15;i++){
(gdb) print i
$2 = 0
(gdb) next
```

总结

1. System Call,C Compiler,C Library
2. GCC
3. make,makefile
4. GDB

作业

按以下要求进行编程练习：

1. 2 个 cpp 文件：prog.cpp，aux.cpp
2. 1 个 h 文件：aux.h
3. aux.h 头文件定义函数 Max 和 Min，它们分别计算四个数（参数）的最大值和最小值；aux.cpp 实现这两个函数
4. prog.c 中定义主函数，循环输入 4 个随机数，输出他们的最大和最小值
5. 编译该项目，调试、跟踪程序执行过程；并在控制台界面运行编译的可执行文件。
6. 编写类似功能的 c 语言程序也可。

The End