第 1 章　Linux C 语言程序设计

本章学习目标

- 了解 C 语言的发展历史和特点；
- 理解 Linux 应用编程、系统编程和内核编程的含义；
- 掌握 Ubuntu Linux 虚拟机的安装；
- 掌握运行 C 程序的流程；
- 掌握 gcc、make、Makefile 的使用；
- 理解 cmake 和 CMakeLists.txt 的优点；
- 了解完整的编译过程。

本章简要介绍了 C 语言的发展历史和特点以及使用 C 语言可以在 Linux 操作系统中进行哪些方面的编程。考虑到有些读者之前没有接触过 Linux，因此，本章介绍了 Linux C 语言编程环境的搭建方法，读者可以根据视频 1-1 安装 Linux 虚拟机。然后通过示例介绍了 Linux C 语言的编程方法和过程。

1.1　C 语言

1.1.1　C 语言简介

计算机程序设计语言的发展大致经历了四个过程，即机器语言、汇编语言、面向过程的程序设计语言和面向对象的程序设计语言。编程语言分为低级语言和高级语言。机器语言和汇编语言属于低级语言，直接用机器指令编写程序；而 C、C ++ 、Java、C ♯ 、Objective-C、Python 等属于高级语言，用语句编写程序。语句是机器指令的抽象表示，分为输入/输出、基本运算、测试分支和循环等几种。

C 语言是一门面向过程的通用计算机编程语言。C 语言兼具高级编程语言和低级编程语言的特点，广泛用于系统软件与应用软件的开发。C 语言可以通过软件工程、模块化编程构建几千万行的超大型软件项目，如 Linux 内核。C 语言描述问题比汇编语言迅速、工作量小、可读性好以及易于调试、修改和移植，而代码质量与汇编语言相当。C 语言生成的目标程序效率一般只比汇编语言低 10％～20％。目前市面上绝大多数操作系统是用 C 语言编写的。很多基础软件，如编译器、数据库、虚拟机、多媒体库、图形库等，都是用 C 语言实现的。很多流行的编程语言是用 C 语言实现的，如 Lua、Python 脚本语言等。在嵌入式系统开发中，比如固件、BSP、内核驱动等，除了少量的汇编代码，大部分也是用 C 语言开

发的。

C 语言具有高效、灵活、功能丰富、表达力强和较高的可移植性等特点。目前,C 语言编译器普遍存在于各种不同的操作系统中。C 语言的设计影响了众多后来的编程语言,例如 C++ 、Java、C# 等。

1.1.2 C 语言发展历史

C 语言的发展历史大致上分为三个阶段,即老格式 C、C89 和 C99。

早期的系统软件几乎都是由汇编语言编写的。汇编语言过分依赖硬件,可移植性很差。一般高级语言又难以实现汇编语言的某些功能,不能很方便地对底层硬件进行灵活的控制和操作,所以急需一种既有高级语言特点又有低级语言功能的中间语言。

1960 年问世的 ALGOL(algorithmic language,算法语言)是所有结构化语言的先驱,它有丰富的过程和数据结构,语法严谨。1963 年剑桥大学将其发展为 CPL(combined programming language)。1967 年剑桥大学的 Martin Richards 对 CPL 做了适当简化后推出了 BCPL(basic combined programming language)。

1970 年,美国贝尔实验室的 Ken Thompson 以 BCPL 为基础,设计出很简单且很接近硬件的 B 语言(取 BCPL 的首字母),并且用 B 语言写了最初的 UNIX 操作系统。

1972 年,美国贝尔实验室的 Dennis Ritchie 在 B 语言的基础上设计出了一种新的语言,取 BCPL 的第二个字母作为这种语言的名字,这就是 C 语言(老格式 C)。

1973 年年初,C 语言的主体完成。Ken Thompson 和 Dennis Ritchie 用它完全重写了 UNIX。随着 UNIX 的发展,C 语言自身也在不断地完善。此后,C 语言开始快速流传,广泛用于各种操作系统和系统软件的开发。

1982 年,很多专家学者和美国国家标准协会(ANSI)为了避免各开发厂商用的 C 语言语法产生差异以及使 C 语言健康地发展下去,决定成立 C 语言标准委员会,建立 C 语言的标准。委员会由硬件厂商、编译器及其他软件工具生产商、软件设计师、顾问、学术界人士、C 语言作者和应用程序员组成。1988 年,美国国家标准协会(ANSI)正式将 C 语言标准化,标志着 C 语言开始稳定和规范化。1989 年,ANSI 发布了第一个完整的 C 语言标准 ANSI X3.159-1989,简称 C89,不过人们也习惯称其为 ANSI C。C89 是最早的 C 语言规范。C89 在 1990 年被国际标准化组织(international standard organization,ISO)原样采纳,ISO 给的名称为 ISO/IEC 9899,所以 ISO/IEC 9899:1990 也常被简称为 C90,ANSI C 又称 C89 或 C90。Ken Thompson 和 Dennis Ritchie 最初发明的 C 语言有很多语法和现在最常用的写法并不一样,但为了向后兼容,这些语法仍然在 C89 和 C99 中保留了下来。之后,C 语言标准委员会不断地对 C 语言进行改进。

1.1.3 C 语言特点

C 语言是一种普适性极强的结构化编程语言,有着清晰的层次,可按照模块的方式对程序进行编写。C 语言依靠全面的运算符和多样的数据类型,可以构建各种数据结构。C 语言可通过指针对内存直接寻址以及对硬件进行直接操作,因此 C 语言既能用于开发系统程序,也可用于开发应用程序。C 语言的特点如下。

1. 低级语言

C 语言能够直接操作硬件和管理内存，因此可以实现汇编语言的主要功能，这使得它是一种非常接近底层的语言，非常适合写需要跟硬件交互以及有极高性能要求的程序。C 语言不但具备高级语言所具有的良好特性，又包含了许多低级语言的优势，故在系统软件编程领域有着广泛的应用。

2. 可移植性

机器语言和汇编语言都不具有移植性，为 x86 开发的程序不可能在 Alpha、SPARC 和 ARM 等机器上运行，而 C 语言非常注重可移植性，C 程序可运行在任意架构的处理器上，只要那种架构的处理器具有对应的 C 语言编译器和库即可。C 语言还是嵌入式系统的首选编程语言，这也是因为 C 语言良好的可移植性。

3. 简单性

C 语言的语法相对简单，很贴近操作系统。一般来说，如果两个语法可以完成几乎相同的功能，C 语言就只提供一种，这样大大减少语言的复杂性。9 类控制语句和 32 个关键字是 C 语言所具有的基础特性，使其在计算机应用程序编写中具有广泛的适用性。

4. 灵活性

C 语言的哲学是“信任程序员，不要妨碍他们做事”。因此，C 语言对程序员的限制很少。C 语言让程序员自己管理内存，不提供内存自动回收功能，也不提供类型检查、数组的负索引检查、指针位置的检查等保护措施。C 语言假设程序员知道自己在干什么，不会限制程序员做各种危险的操作，干什么都可以，后果也由程序员自己负责。这表面上看似乎很危险，但是对于高级程序员来说，却有了更大的编程自由。

1.2　Linux 简介

Linux 是一款诞生于网络、成长于网络并且成熟于网络的操作系统。Linux 最早由一位名叫 Linus Torvalds 的芬兰赫尔辛基大学计算机科学系的学生开发，然后由世界各地的成千上万的程序员设计和实现。Linux 可在 GNU(GNU's Not Unix)公共许可权限下免费获得，是一个符合 POSIX 标准的操作系统。Linux 是一个不受任何商品化软件版权制约、全世界都能自由使用且遵循 GNU 通用公共许可证(GPL)的操作系统。

Linux 版本有两种，即内核版本和发行版本。

对于 Linux 初学者来说，经常分不清内核版本与发行版本。实际上，Linux 操作系统的内核版本是指在 Linus Torvalds 领导下的开发小组开发的 Linux 内核的版本。Linux 操作系统的核心就是它的内核，Linus Torvalds 和他的小组在不断地开发和推出新内核。内核的主要作用包括进程调度，内存管理，配置管理虚拟文件系统，提供网络接口以及支持进程间通信。像所有软件一样，Linux 内核也在不断升级。

一个完整的操作系统不仅只有内核，还包括一系列为用户提供各种服务的外围程序。所以，许多个人、组织和企业，开发了基于 GNU/Linux 的 Linux 发行版本，他们将 Linux 系统的内核与外围应用软件和文档包装起来，并提供一些系统安装界面和系统设置与管理工

具,这样就构成了一个发行版本。实际上,Linux 的发行版本就是 Linux 内核和外围实用程序组成的一个大软件包而已。相对于操作系统内核版本,发行版本的版本号是随发布者的不同而不同,与 Linux 系统内核的版本号是相对独立的。

Linux 的发行版本大体可以分为两类,一类是商业公司维护的发行版本,另一类是社区组织维护的发行版本。前者以著名的 Red Hat Linux 为代表,后者以 Debian 为代表。目前 Red Hat 系的 Linux 发行版本主要包括 RHEL、Fedora、CentOS、CentOS Stream、Rocky Linux、OEL 和 SL。Debian 系的 Linux 发行版本主要包括 Debian、Ubuntu、Kali 和 Deepin。

Ubuntu(乌班图)由开源厂商 Canonical 公司开发和维护。Ubuntu 是基于 Debian 的 unstable 版本加强而来,拥有 Debian 的所有优点。本书所有示例都运行在 Ubuntu 22.04 上。

1.3 Linux C 语言程序设计简介

1.3.1 Linux 应用编程、系统编程和内核编程

在 Linux 系统中可以使用 C 语言进行内核编程与应用程序开发。内核程序运行在内核态,应用程序主要运行在用户态,当需要内核服务时会通过系统调用切换到内核态下运行,内核相关代码执行完后再返回用户态。

内核模块是具有独立功能的程序,可以被单独编译,但不能单独运行,必须被链接到 Linux 内核,作为内核的一部分在内核空间运行。模块编程也称为内核编程,通常是开发各种驱动程序。

Linux 应用编程和系统编程都是指在用户空间程序进行的开发,涉及多方面的编程。Linux 系统编程就是编写各种函数库。Linux 应用编程就是利用写好的各种函数库来编写具有某种功能的应用程序,包括各种用户应用程序、各种工具软件、各种系统软件、网络程序、图形界面程序等。

1.3.2 Linux 图形界面编程

在 Linux 中除了能够开发基于字符界面的应用程序,也可以开发出美观的图形界面应用程序。图形界面编程也叫 GUI(graphical user interface,图形用户界面)编程。目前在 Linux 中已经有多种用于开发 GUI 程序的开发库,其中最常用的是 Qt 和 GTK。

Qt 是一个跨平台的 GUI 开发库,它不仅支持 Linux,还支持所有类 UNIX 以及 Windows。Linux 的桌面环境 KDE 就是使用 Qt 作为其底层库开发出来的。Qt 使用 C++ 语言作为其开发语言。

GTK 是用 C 语言编写的用于开发 GUI 程序的开发库。GTK 几乎可以在任何操作系统上使用。与 Qt 不同,GTK 支持使用纯 C 语言进行开发。Linux 的桌面环境 GNOME 就是建立在 GTK 基础上的。

1.4　Linux C 语言编程环境

1.4.1　安装 Ubuntu Linux 虚拟机

读者可以通过 VirtualBox 安装 Ubuntu，进而学习 Linux C 语言程序设计。假设读者计算机的配置为：内存 4GB，CPU 为 2 核 4 线程，则可以给 Ubuntu 虚拟机分配 1GB 内存，2 个逻辑 CPU。如果读者计算机的硬件配置更高，则可以多分配些内存给 Ubuntu 虚拟机。


视频 1-1
安装
Ubuntu

读者可以从本书配套资源下载 VDI（virtual disk images，虚拟硬盘镜像）文件。在 VirtualBox 中导入 VDI 文件即可。步骤为：①单击新建按钮，新建一个虚拟系统。建议起一个有意义的名字，类型选择 Linux，版本选择 Ubuntu（64bit）。②给 Ubuntu 虚拟机分配 1GB 内存或更多。③在虚拟硬盘页面选择“使用已有的虚拟硬盘文件”，找到 VDI 文件。④保存配置，然后启动 Ubuntu 虚拟机。

启动 Ubuntu 虚拟机后，在登录窗口以普通账号 ztg 或管理员账号 root 登录，在输入密码的界面选择 Xfce 会话，如图 1-1 所示，输入密码（111111）后按 Enter 键，即可进入 Xfce 桌面环境，如图 1-2 所示。

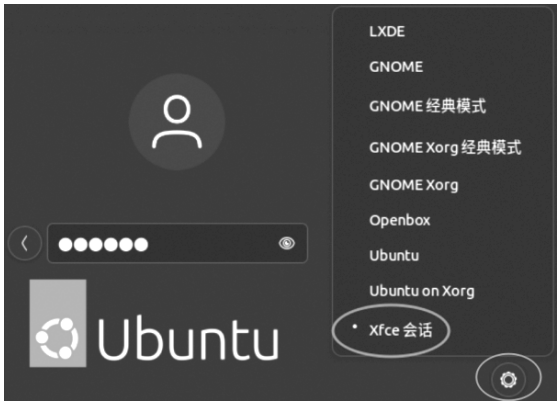

图 1-1　选择 Xfce 会话

Xfce 是使用率仅次于 KDE 与 GNOME 的 Linux 桌面环境。如果给虚拟机分配的内存大于 2GB，则建议使用 GNOME 桌面环境，否则使用 Xfce 桌面环境。

1.4.2　gedit、vim 和 nano

Linux 中一个常用的文本编辑器是 gedit。gedit 是一个简单的文本编辑器，用户可以用它完成大多数的文本编辑任务，如修改配置文件等。

vi 是 visual interface 的简称，它为用户提供了一个全屏幕的窗口编辑器，窗口中一次可以显示一屏的编辑内容，并可以上下屏地滚动。vi 是 Linux 和 UNIX 系统中标准的文本编辑器，可以说几乎每一台 Linux 或 UNIX 机器都会提供这套软件。vi 可以工作在字符模式下。由于不需要图形界面，使它成为效率很高的文本编辑器。尽管在 Linux 上也有很多图

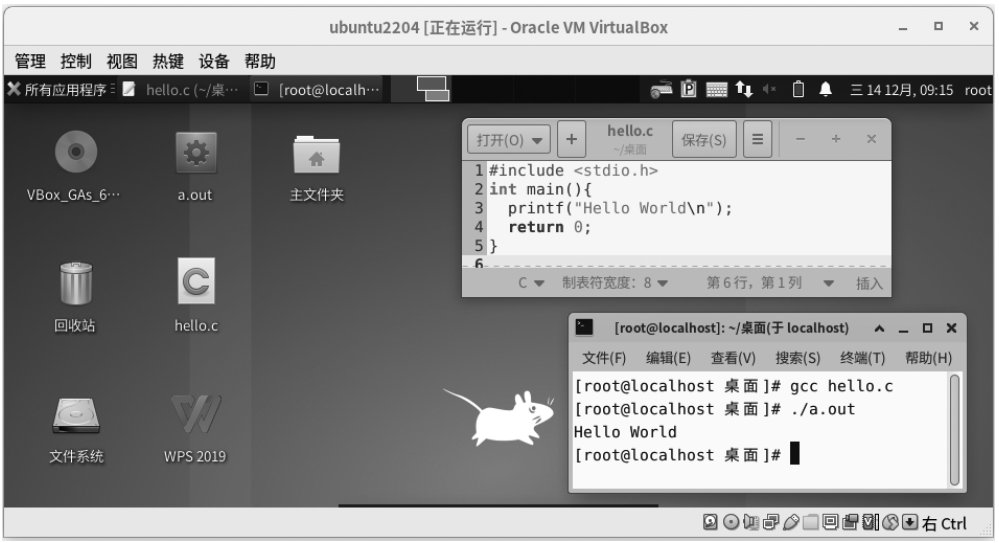

图 1-2 Xfce 桌面环境

形界面的编辑器可用,但 vi 在系统和服务器管理中的能力是其他图形编辑器所无法比拟的。

vim 是 vi 的增强版,即 vi improved。执行 apt install vim 命令安装 vim。在后面的实例中将介绍 vim 的使用。

nano 是一个用于字符终端的文本编辑器,比 vi/vim 简单,比较适合 Linux 初学者使用。nano 命令可以对打开的文件进行编辑。

1.4.3 C 语言编译器及集成开发环境

CPU 只能识别二进制指令,无法直接执行源代码。因此,在程序真正运行之前必须将源代码转换成二进制指令。根据不同语言转换时机的不同,将高级编程语言分为编译型语言和解释型语言。

编译型语言在程序执行之前需要通过编译器把程序编译成可执行文件,然后由机器运行这个可执行文件。以后再次运行该文件时不需要重新编译。解释型语言是边执行边转换,不会由源代码生成可执行文件,而是先转换成中间代码,再由解释器对中间代码进行解释运行,以后每次执行时都要再次转换。Python 语言属于典型的解释型语言。

C 语言是一种编译型语言,源代码都是文本文件,本身无法执行。必须通过编译器生成二进制的可执行文件才能执行。

C 语言编译器主要有 GCC、MinGW、Clang 和 cl.exe。

C 语言的集成开发环境主要有 Code::Blocks、CodeLite、Dev-C++、C-Free 和 Visual Studio。

目前,最常见的 C 语言编译器是自由软件基金会推出的 GCC(GNU compiler collection)编译器。Linux 和 Mac 操作系统可以直接安装 GCC,Windows 操作系统可以安装 MinGW。本书的所有例子都使用 GCC 编译器在命令行进行编译。

1.4.4　编写 Hello World 程序

使用 C 语言编写的第一个程序的源代码如下，功能是向显示器输出字符串"Hello World"。


视频 1-2
运行 Hello
World 程序

```
1: #include <stdio.h>
2: /*
3:  * 多行注释
4:  */
5: int main(){
6:   printf("Hello World\n");          //在屏幕上输出字符串。单行注释
7:   return 0;
8: }
```

C 语言源代码文件通常以.c 为后缀。将上面源代码保存在文件 hello.c 中。hello.c 就是一个普通的文本文件。

```
#gcc hello.c
#./a.out
Hello World
```

执行上面的 gcc 命令将源文件 hello.c 编译成二进制代码。注意，# 是命令行提示符，需要输入的是 # 后面的部分。运行 gcc 命令编译后，会在当前目录下生成可执行文件a.out。直接在命令行输入 a.out 的路径就可以执行它，会在屏幕上显示字符串 Hello World。

```
#gcc -o hello hello.c
#./hello
Hello World
```

也可以执行上面的 gcc 命令将源文件 hello.c 编译成二进制代码。gcc 的-o 选项指定生成的可执行文件名 hello。

下面简要介绍一下上面的源代码。

C 语言程序的基本组成单位是函数。每个 C 语言程序都是由若干个函数组成的，其中至少应该包括一个主函数 main(本小节程序实例的第 5～8 行)。函数是由若干条语句组成的，每条语句后面都要加上分号。C 程序总是从 main 函数里的第一条语句开始执行，在这里就是 printf 这条函数调用语句。printf 是一个函数名，功能是将括号中双引号引起的字符串原样输出到屏幕上。为了提高程序的可读性，最好添加注释，有单行注释(//)和多行注释(/*...*/)两种方法。第 1 行使用预处理指令 include 将头文件 stdio.h 包含在 hello.c 文件中，这样就可以调用 stdio.h 头文件中声明的 printf 库函数。

1.5　使用 gcc 编译程序

Linux 上常用的编程工具套件是 GCC。GCC 原名为 GNU C Compiler，后来逐渐支持更多的编程语言，如 C++ 、Fortran、Pascal、Objective-C、Java、Ada、Go 以及各类处理器架构

上的汇编语言等,所以改名为 GNU Compiler Collection(GNU 编译器套件)。在 Debian/Ubuntu 操作系统命令行执行 apt install build-essential 命令,安装 GCC。

GCC 套件中的 C 语言编译器是 gcc,对应的命令也是 gcc。下面通过示例简要介绍 gcc 的使用。编写一个简单的程序,包含 3 个.c 文件和 1 个.h 文件,文件名及其内容如下所示。

```
① main.c
1: #include "hello.h"
2: int main(void){
3:   hello1("world", i);
4:   hello1("world",++i+j);
5:   hello2(++j);
6: }
③ hello2.c
1: #include <stdio.h>
2: int j=10;
3: void hello2(int j){
4:   printf("j: %d\n",j);
5: }
② hello1.c
1: #include <stdio.h>
2: int i=5;
3: void hello1(char * s, int i){
4:   int j=6;
5:   printf("hello %s, i+j=%d\n", s, i+j);
6: }
④ hello.h
1: extern int i, j;
2: void hello1(char *,int);
3: void hello2(int);
```


视频 1-3
单步编译
和多步编译

可以采用单步编译或多步编译的方法编译该程序,如下所示。相关源代码文件在本书配套资源的“src/第 1 章/hello”目录中。

① 单步编译及运行结果如下:

```
#gcc main.c hello1.c hello2.c -o main
#./main
hello world, i+j=11
hello world, i+j=22
j: 11
```

② 多步编译如下:

```
#gcc -c main.c
#gcc -c hello1.c
#gcc -c hello2.c
#gcc main.o hello1.o hello2.o -o main
#./main
```

使用 gcc 编译器时需要给出必要的选项和文件名。gcc 命令的语法如下:

```
gcc [options] [filenames]
```

其中,filenames 是文件名,多个文件名之间由空格隔开。options 是编译器所需的选项,常用选项如下。

-c:只编译汇编生成目标文件,不链接生成可执行文件。gcc 编译器只是将.c 源代码文件编译汇编生成以.o 为后缀的目标文件。该选项通常用于编译不包含主程序的子程序文件。

-E:只进行预处理而不编译。结合使用-o 选项,gcc 编译器可将.c 源代码文件预处理成以.i 为后缀的源文件,如 gcc -E main.c -o main.i。cpp 命令也可以达到同样的效果。

-S:只编译生成汇编代码。gcc 编译器只是将.c 源代码文件编译生成以.s 为后缀的汇

编语言源程序文件。

-o outfile：指定输出文件名为 outfile，这个文件名不能和源文件重名。不带-c、-E、-S 选项时，gcc 编译器能将.c 源代码文件编译成可执行文件。如果没有使用选项-o 给出可执行文件名，gcc 将生成一个名为 a.out 的文件。在 Linux 系统中，可执行文件没有特定的后缀名，Linux 系统根据文件的属性来区分可执行文件和不可执行文件。不过 gcc 命令通常根据后缀来识别文件的类别。

-O?：各种编译优化选项。-O0 为默认优化选项。-O/-O1 这两个都会尝试减少代码段大小和优化程序的执行时间。相比于-O1，-O2 打开了更多的编译优化开关。-O3 在-O2 的基础上进行更高级别优化。-Os 优化生成的目标文件的大小。-Ofast 为了提高程序的执行速度而进行优化。为了能够生成更好的调试信息，-Og 关闭很多优化开关。如果同时使用多个不同级别的-O 优化选项，编译器会根据最后一个-O 选项的级别决定采用哪种优化级别。

-g：在生成的目标文件中添加调试信息，在 gdb 调试和 objdump 反汇编时要用到这些信息。-g0 不生成调试信息，相当于没有使用选项-g。-g1 生成最小的调试信息，最小调试信息包括函数描述、外部变量、行数表，但不包括局部变量信息。-g2 是默认的调试级别，即为-g。-g3 相对-g 生成额外的信息，如所有的宏定义。如果多个级别的-g 选项同时存在，最后的-g 选项会生效。如果没有使用其他优化选项，可以将-Og 与-g 选项一起使用。

-D macroname[=definition]：定义一个宏，macroname 为宏名，definition 为宏值。

-U macroname：取消已经定义的宏 macroname。

-Idir：指定头文件所在的目录路径。

-Ldir：指定库文件所在的目录路径。

-print-search-dirs：打印库文件的默认搜索路径。

-v：打印详细的编译链接过程。

-Wall：打印所有的警告信息。

1.6　使用 make 和 Makefile 构建程序

如果一个软件项目只有一个源文件，则可以直接使用 gcc 命令进行编译。但是真正实用的软件项目中都会包含多个源文件，并且将它们分类放在不同文件夹中，如果用 gcc 命令逐个文件编译，工作量大且易出错。此时可以使用 make(GNU Make)命令，它是一个自动化编译工具，使用一条 make 命令即可对多文件软件项目进行完全编译和链接。但是 make 命令本身并没有编译和链接功能。make 命令需要一个规则文件 Makefile，该文件中描述了整个软件项目的编译规则和各个文件之间的依赖关系。make 命令通过调用 Makefile 文件中的命令来进行编译和链接，从而实现自动化编译，极大地提高了软件开发效率。

使用 make 命令可以最小化重新编译次数。如果不使用 make 命令，而是使用 gcc 命令编译程序，若对一个源文件做了修改，则所有源文件都要重新编译一遍，如 1.5 节的单步编译。这肯定不是个好办法，如果编译之后又对 hello1.c 做了修改，又要把所有源文件编译一遍，即使其他文件都没有修改也要重新编译。一个大型软件项目通常包含成千上万个源文件，全部编译一遍耗时很长，上述编译过程是不合理的。可以采用 1.5 节的多步编译，如果

编译之后又对 hello1.c 做了修改,要重新编译只需要执行第 2 行和第 4 行的命令即可。但是当文件很多时这种手动编译方法也很容易出错。因此更好的办法就是写一个 Makefile 文件,和源代码放在同一个目录下,make 命令会自动读取当前目录下的 Makefile 文件,完成相应的编译步骤。即使有个别文件做了修改,执行 make 命令时也只对修改的文件进行编译。


视频 1-4
Makefile

① Makefile 文件如下:

```
main: main.o hello1.o hello2.o
    gcc main.o hello1.o hello2.o -o main
main.o: main.c hello.h
    gcc -c main.c
hello1.o: hello1.c
    gcc -c hello1.c
hello2.o: hello2.c
    gcc -c hello2.c
clean:
    -rm main *.o
    @echo "clean completed"
.PHONY: clean
```

② 使用 make 命令编译程序如下:

```
#make
gcc -c main.c
gcc -c hello1.c
gcc -c hello2.c
gcc main.o hello1.o hello2.o -o main
#make
make: “main”已是最新
#touch hello1.c
#make
gcc -c hello1.c
gcc main.o hello1.o hello2.o -o main
#make clean
rm main *.o
clean completed
```

1. Makefile 规则

Makefile 文件由一组规则(rule)组成,每条规则的格式如下:

```
target ... : prerequisites ...
    command1
    command2
    ...
```

Makefile 文件中,第 1、第 2 行是一条规则,main 是这条规则的目标(target),main.o、hello1.o、hello2.o 是这条规则的条件。目标和条件之间的关系是:如果要更新目标,则必须先更新它的所有条件;所有条件中只要有一个条件被更新,则目标也必须随之被更新。更新的含义就是执行一遍规则中的命令列表,命令列表中的每条命令必须以一个 Tab 开头,注意不能是空格。对于 Makefile 文件中的每个以 Tab 开头的命令,make 都会创建一个 Shell