# Linux下C#编程基础

## 3.1 vi（vim）编辑器（vim是vi升级版）

### 3.1.1 vi的模式

Vi有三种模式，分别为命令行模式、插入模式、底行（末行）模式。

1. 命令行模式

在用vi编辑文件时，最初进入的为一般模式。在改模式下用户可以通过上下移动光标进行“删除字符”或者“整行删除”等操作，也可以进行“复制”、“粘贴”等操作，但是无法编辑文字。

1. 插入模式

在此模式下可以进行文字编辑输入，按[ESC]键可以回到命令行模式。

1. 底行模式

在此模式下，光标位于屏幕的底行。用户可以进行文件的保存或者退出操作，也可以设置编辑环境，如寻找字符串、列出行号等。

### 3.1.2 vi各模式功能键（设行号，查字符串）

（1）命令行模式常见功能键

|  |  |
| --- | --- |
| 功能键 | 功 能 |
| /name | 在光标之后查找一个名为name的字符串 |
| ?name | 在光标之前查找一个名为name的字符串 |

（2）插入模式的功能键只有一个，即按“[ESC]”键回到命令行模式

（3）底行模式常见功能键

|  |  |
| --- | --- |
| 功能键 | 功 能 |
| :set nu | 显示行号，设定之后，会在每一行的前面显示对应行号 |
| :set nonu | 取消行号显示 |

## 3.2 gcc编译器

### 3.2.1 gcc编译流程和各功能

hello.c

#include <stdio.h>

int main()

{

printf("Hello! This is our embedded world!\n");

return 0;

}

gcc编译流程分为了四个步骤：

1. 预处理

对包含(#include)和(#define、#ifdef)等进行处理。编译器将包含的头文件stdio.h编译进来，并且用户可以使用gcc的选项”-E”进行查看，此选项是让gcc在预处理结束后停止编译过程。

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

“-o”是指目标文件，”.i”是为已经预处理过的C程序，把”stdio.h”的内容插入到hello.i文件中。

1. 编译

随后是编译阶段，首先gcc要检查代码的规范性、是否有语法错误等，用以确定代码实际要做的工作，检查无误后，gcc把代码翻译成汇编语言。并且可以用”-S”选项来进行查看，该选项只进行编译不进行汇编，结果生成汇编代码。

**gcc -S hello.i -o hello.s**

1. 汇编

把编译阶段生成的”.s”文件转成目标文件，在此使用”-c”选项就可以看到汇编代码已转化为”.o”的二进制目标代码了。

**gcc -c hello.s -o hello.o**

1. 链接

成功编译后进入链接阶段。重要概念：函数库

在没有特别指定时候，gcc会到系统默认的搜索路径”/usr/lib”下进行查找，链接到libc.so.6函数库中去，这样就能调用函数”printf”了，这就是链接的作用。

函数库有静态库和动态库两种。

静态库是指编译链接时，将库文件的代码全部加入可执行文件中，因此生成的文件比较大，但在运行时也就不再需要库文件了。其后缀名通常为”.a”。

动态库与之相反，在编译链接时并没有将库文件的代码加入可执行文件中，而是在程序执行时加载库，这样可以节省系统开销。一般动态库后缀名为”.so”，如前面所述的libc.so.6就是动态库。gcc在编译时默认使用动态库。

完成链接后，gcc就可以生成可执行文件，如下

**gcc hello.o -o hello**

运行该可执行文件：

**./hello**

## 3.3静态库与动态库区别

动态库只有当使用它的程序执行时才被链接使用，而不是将需要的部分直接编译入可执行文件中，并且一个动态库可以被多个程序使用故可称为共享库，而静态库将会整合到程序中，因此在程序执行时不用加载静态库。从而可知，链接到静态库会使用户的程序臃肿，并且难以升级，但是可能会比较容易部署。而连接到动态库会使用户的程序轻便，并且易于升级，但是会难以部署。

## 3.4 gdb调试器

### 3.4.1 gdb使用流程

test.c为例

|  |  |
| --- | --- |
| 命令 | 注释 |
| **gcc -g test.c -o test** | 生成可执行文件test |
| **gdb test** | 对可执行文件进行调式（进入调式页面） |

1. 查看文件

在gdb中键入”l(list)”就可以查看所载入的文件，并且列出的源代码还给出行号，方便定位。

1. 设置断点

使程序运行到一定位置时暂停。因此，在暂停处可以方便的查看变量值、堆栈情况，从而找出代码症结所在。

**b 6**

上述命令的意思是运行到第六行之前暂停（并没有运行到第六行）。

1. 查看断点情况

在设置完断点过后，用户可以键入”info b”来查看设置断点情况，在gdb中可以设置多个断点。

可以在断点键入”backrace”（只输入”bt”即可）可以查到调用函数（堆栈）的情况，经常用于排除错误或者监视调用堆栈的情况。

1. 运行代码

gdb默认从首行开始运行代码，键入”r”（run）即可，如果想指定某行开始运行，在r后面加上行号即可。直到运行到断点处停止。

1. 查看变量值

在程序停止运行之后，在gdb中键入”p”+变量值即可。

**p n**

上述命令的意思是查看变量n的值。

1. 单步运行

可以用命令”n”（next）或”s”（step）其区别在于，若有函数调用的时候，”s”会进入函数而”n”不会进入函数。”s”就类似于Visual C++等工具中的”step in”，”n”就类似于Visual C++等工具中的”step over”。”n”为逐函数运行，”s”为逐步运行。

1. 恢复程序运行

可以使用”c”（continue）恢复程序的正常运行，把剩余的未执行的程序执行完。

### 3.4.2 gdb基本命令

|  |  |
| --- | --- |
| set args 运行时的参数 | 指定运行时的参数，如set args 2 |

**set n=4**

# 3.5 make工程管理器

3.5.1 makefile基本结构

（1）需要由make工具创建的目标体（target），通常是目标文件或可执行文件；

（2）要创建的目标体所依赖的文件（dependency\_file）；

（3）创建每个目标体时需要运行的命令（command），这一行必须以制表符（Tab键）开头。

格式为：

target：dependency\_files

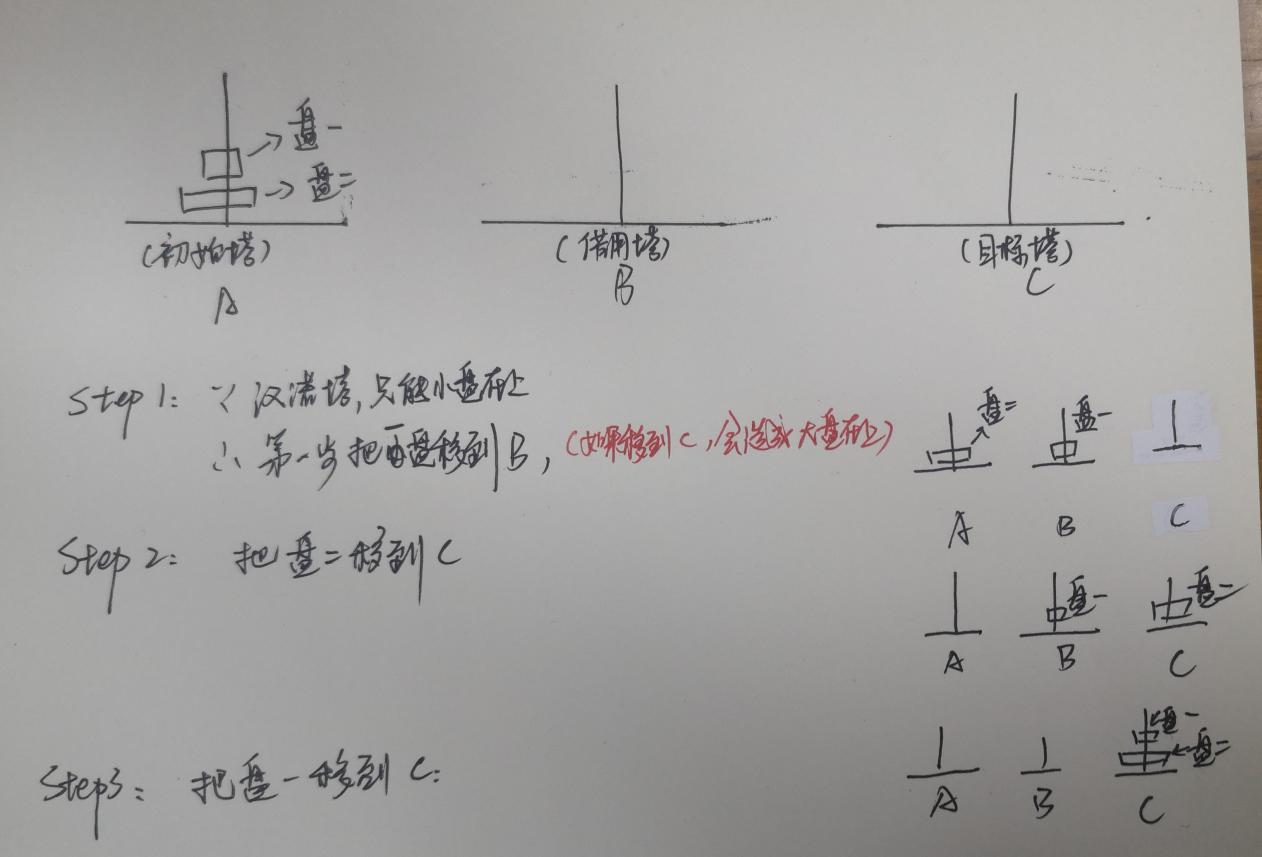
command

例如，有两个文件分别为hello.c和hello.h，创建的目标体为hello.o，执行的命令为gcc编译指令：gcc -c hello.c，那么对应的makefile就可以写成如下：

Hello.o：hello.c hello.h

gcc -c hello.c -o hello.o

3.6 汉诺塔



# 文件I/O编程

## 6.1 Linux系统调用及用户编程接口（API）

（1）系统调用

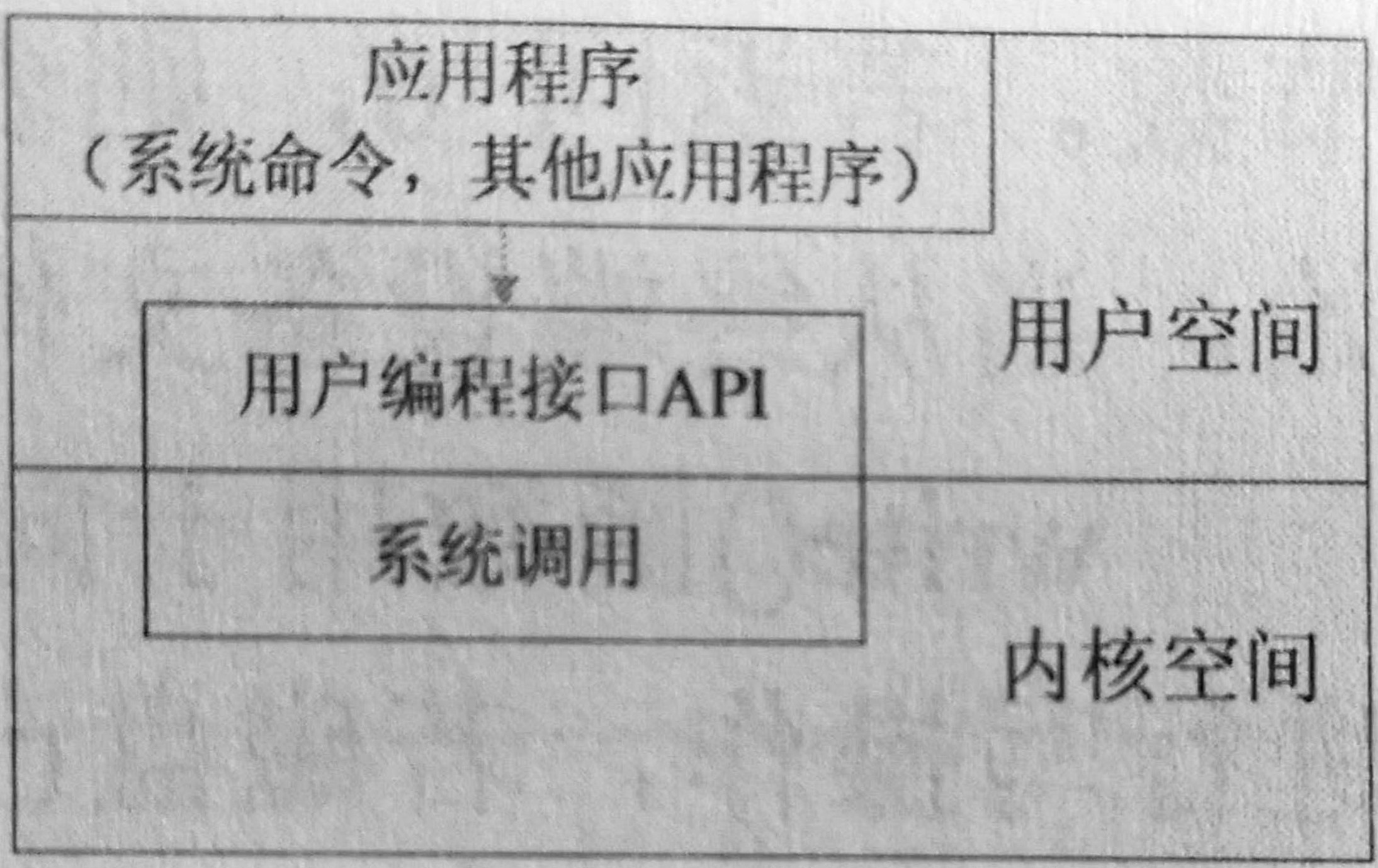
所谓系统调用是指操作系统提供给用户程序调用的一组特殊接口，用户程序可以通过这组特殊接口来获得操作系统内核提供的服务。

（2）用户编程接口

API实现系统调用途径

（3）系统命令

系统命令相对于API高了一层，实际上是一个可执行程序，他的内部引用了用户编程接口API来实现相应的功能



系统调用、API及系统命令之间的关系

## 6.2 copy\_file.c 书上P157页 看懂

## 6.3 文件锁

fcntl( )能对文件的某一记录上锁，也就是记录锁。

记录锁分为读取锁和写入锁，其中读取锁又称为共享锁，它能够使多个进程都能在文件的同一部分建立读取锁。而写入锁又称排斥锁，在任何时刻只能有一个进程在文件的某个部分上建立写入锁。在文件的同一部分不能同时建立读取锁和写入锁。

## 6.4 multiplex\_select.c 书上P166页 看懂