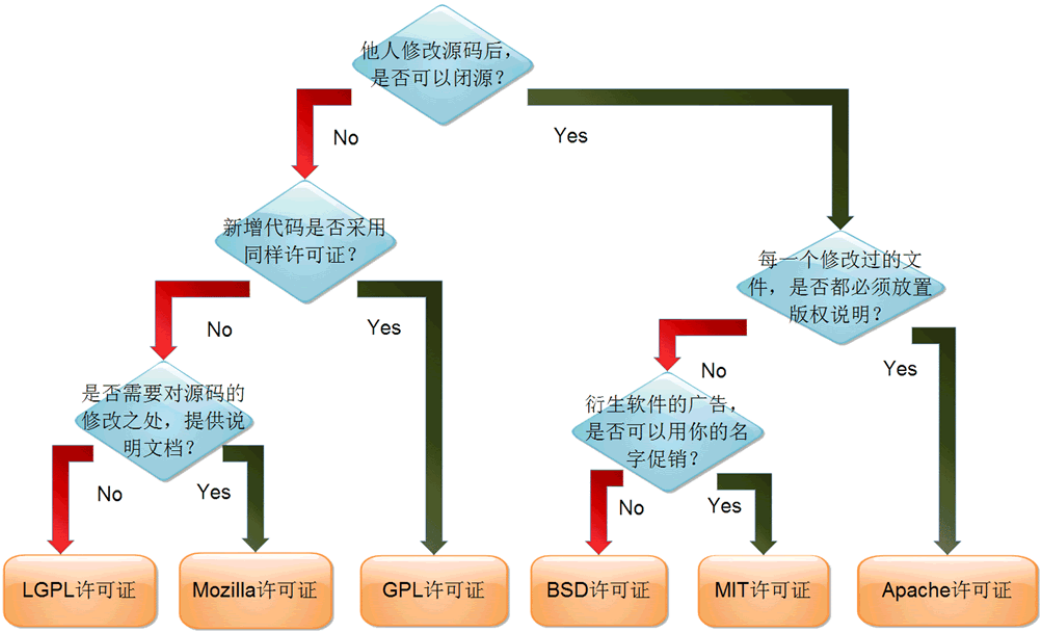
知识

1 综合

1.1 ++

——

开源协议



——

MinGW与MSYS（Minimal SYStem）

原本 GNU 工具只在 Linux/Unix 系统里才有，随着 Windows 系统的广泛使用， 为了在 Windows 系统里可以使用 GNU 工具，诞生了 MinGW（Minimalist GNU for Windows） 项目，利用 MinGW 就可以生成 Windows 里面的 exe 程序和 dll 链接库。

需要注意的是，MinGW 与 Linux/Unix 系统里 GNU 工具集的有些区别：

- MinGW 里面工具带有扩展名 .exe， Linux/Unix 系统里工具通常都是没有扩展名的。
- MinGW 里面的生成器文件名为 mingw32-make.exe，Linux/Unix 系统里就叫 make。
- MinGW 在链接时是链接到 *.a 库引用文件，生成的可执行程序运行时依赖 *.dll，而 Linux/Unix 系统里链接时和运行时都是使用 *.so 。

另外 MinGW 里也没有 ldd 工具，因为 Windows 不使用 .so 共享库文件。如果要查看 Windows 里可执行文件的依赖库，需要使用微软自家的 Dependency Walker 工具。Windows 里面动态库扩展名为 .dll，MinGW 可以通过 dlltool 来生成用于创建和使用动态链接库需要的文件，如 .def 和 .lib。

MinGW 原本是用于生成 32 位程序的，随着 64 位系统流行起来， 从 MinGW 分离出来了 MinGW-w64 项目，该项目同时支持生成 64 位和 32 位程序。Qt 的 MinGW 版本库就是使用 MinGW-w64 项目里面的工具集生成的。

另外提一下，由于 MinGW 本身主要就是编译链接等工具和头文件、库文件，并不包含系统管理、文件操作之类的 Shell 环境， 这对希望用类 Unix 命令的开发者来说还是不够用的。 所以 MinGW 官方又推出了 MSYS（Minimal SYStem），相当于是一个部署在 Windows 系统里面的小型 Unix 系统环境， 移植了很多 Unix/Linux 命令行工具和配置文件等等，是对 MinGW 的扩展。

MSYS 对于熟悉 Unix/Linux 系统环境或者要尝试学习 Unix/Linux 系统的人都是一种便利。MSYS 和 MinGW 的安装升级都是通过其官方的 mingw-get 工具实现，二者是统一下载安装管理的。

对于 MinGW-w64 项目，它对应的小型系统环境叫 MSYS2（Minimal SYStem 2），MSYS2 是 MSYS 的衍生版，不仅支持 64 位系统和 32 位系统，还有自己的独特的软件包管理工具，它从 Arch Linux 系统里移植了 pacman 软件管理工具，所以装了 MSYS2 之后，可以直接通过 pacman 来下载安装软件，而且可以自动解决依赖关系、方便系统升级等。装了 MSYS2 之后，不需要自己去下载 MinGW-w64，可以直接用 pacman 命令安装编译链接工具和 git 工具等。

MinGW 项目主页（含 MSYS）： http://www.mingw.org/

MinGW-w64 项目主页： https://sourceforge.net/projects/mingw-w64/

MSYS2 项目主页： https://sourceforge.net/projects/msys2/

--

CMake

CMake（Cross platform Make）是一个开源的跨平台自动化构建工具，可以跨平台地生成各式各样的 makefile 或者 project 文件，支持利用各种编译工具生成可执行程序或链接库。

CMake 自己不编译程序，它相当于用自己的构建脚本 CMakeLists.txt，叫各种编译工具集去生成可执行程序或链接库。

一般用于编译程序的 makefile 文件比较复杂，自己去编写比较麻烦，而利用 CMake，就可以编写相对简单的 CMakeLists.txt，由 CMake 根据 CMakeLists.txt 自动生成 makefile，然后就可以用 make 生成可执行程序或链接库。

本教程里面是使用 Qt 官方的 qmake 工具生成 makefile 文件，没有用 CMake。这里之所以提 CMake，是因为整个 KDE 桌面环境的茫茫多程序都是用 CMake 脚本构建的，另外跨平台的程序/库如 Boost C++ Libraries、OpenCV、LLVM、Clang 等也都是用 CMake 脚本构建的。以后如果接触到这些东西，是需要了解 CMake 的。

CMake 项目主页：https://cmake.org/
KDE 项目主页：https://www.kde.org/

--

GNU工具集

工具	说明
gcc	GNU C 语言编译器。
g++	GNU C++语言编译器。
ld	GNU 链接器，将目标文件和库文件链接起来，创建可执行程序和动态链接库。
ar	生成静态库 .a，可以编辑和管理静态链接库。
make	生成器，可以根据 makefile 文件自动编译链接生成可执行程序或库文件。
gdb	调试器，用于调试可执行程序。
ldd	查看可执行文件依赖的共享库（扩展名 .so，也叫动态链接库）。

--

字符编码

ANSI编码： 是一种对ASCII码的拓展，ANSI码仅在前128（0-127）个与ASCII码相同，之后的字符全是某个国家语言的所有字符。最多可以存储的字符数目是2的16次方，即65536个字符。
GBK 扩展中文GB编码（兼容了GB2312）更全： 编码方式：两个字节表示一个汉字 英文字母或半角标点占用一个字节
Unicode编码： 最常用的是用两个字节表示一个字符（如果要用到非常偏僻的字符，就需要4个字节）。 问题在于，原本可以用一个字节存储的英文字母在Unicode里面必须存两个字节（规则就是在原来英文字母对应ASCII码前面补0），这就产生了浪费。
UTF-8： 用1到4个字节来表示一个字符（比较节省空间）
UTF-16编码： 用1到2个short来表示一个字符
UTF-32编码： 用一个int来表示 abcd 需要占用4个int

--

库函数缺陷

系统相关 - 库函数缺陷（strcpy, 线程不安全函数等） - 举例：strcpy

THUNDERWORLD 中科

• strcpy的第1个参数可以出现缓冲区溢出。
• 比如下面的程序会出现缓冲区溢出：

```
#include <stdio.h>
#include <string.h>
int main()
{
   char strings[] = "123\0abcdefg\0";
   char *str1 = strings;
   char *str2 = strings + 4;
   printf("str1=%s,str2=%s\n", str1, str2);
   strcpy(str1, str2);
   printf("str1=%s,str2=%s\n", str1, str2);
   return 0;
}
```

• 此程序的运行结果为，str2被篡改：
str1=123,str2=abcdefg
str1=abcdefg,str2=efg

系统相关 - 库函数缺陷（strcpy, 线程不安全函数等） - 举例：strncpy

THUNDERWORLD

• `man strncpy`命令中的说明如下：

```
The strncpy() function is similar, except that at most n bytes of src are copied.
Warning: If there is no null byte among the first n bytes of src, the string placed in
dest will not be null-terminated.
```

• 看如下程序：

```
#include <stdio.h>
#include <string.h>
int main()
{
   char dest[5];
   memset(dest, 0xff, sizeof(dest));
   strncpy(dest, "1234567", sizeof(dest) - 1);  /* "- 1" 是考虑'\0' */
   printf("%s\n", dest);
   printf("%d, %d, %d, %d, %d\n", dest[0], dest[1], dest[2], dest[3], dest[4]);
   return 0;
}
```


系统相关 - 库函数缺陷（strcpy, 线程不安全函数等） - 举例：strncpy

• 上面程序的输出结果是：


```
1234◆
49, 50, 51, 52, -1
```

• 可以看到乱码，且dest没有以'\0'结尾，这是比较危险的，使用“%s”打印时不好说什么时候会遇到'\0'。

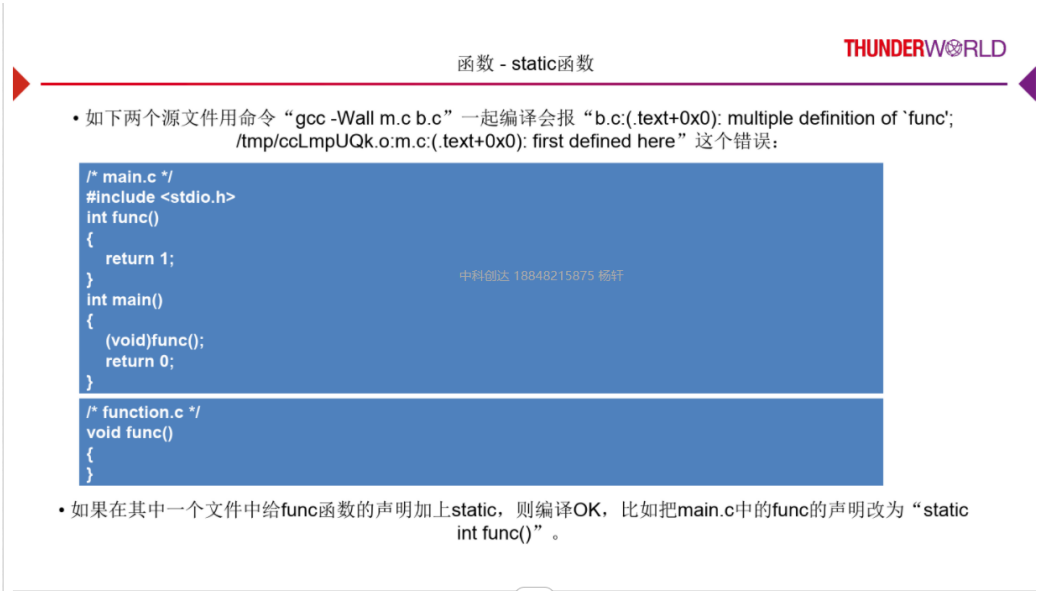
系统相关 - 库函数缺陷（strcpy, 线程不安全函数等） - 举例：strerror

THUNDERWORLD

• strerror是非线程安全函数，其返回值指向多个线程可共享的静态存储区，多线程调用此函数时，会有一定概率产生log字符串掺杂在一起的混乱情况。
• 但strerror不会导致程序crash。

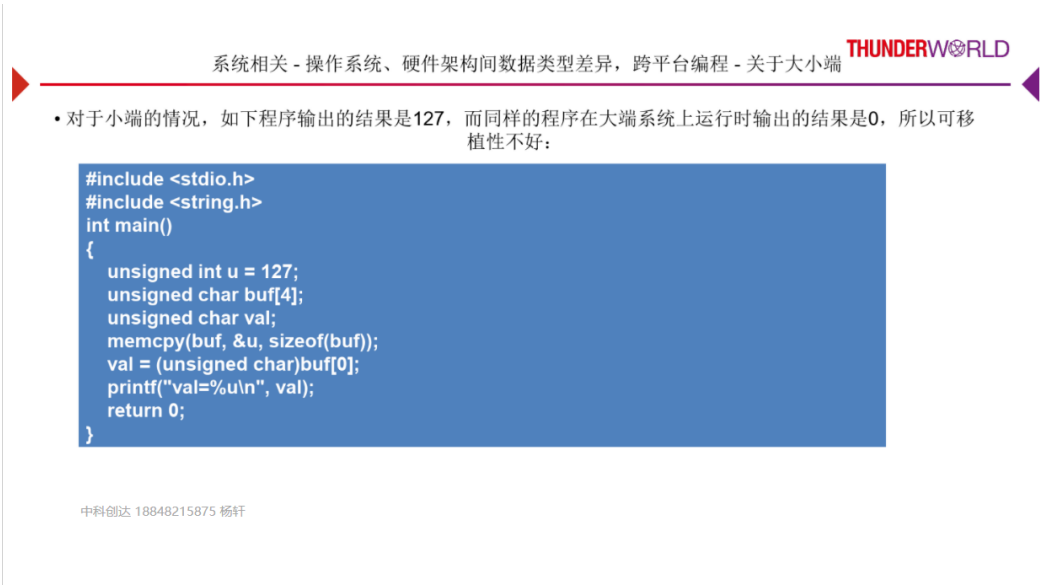
1.2 报错

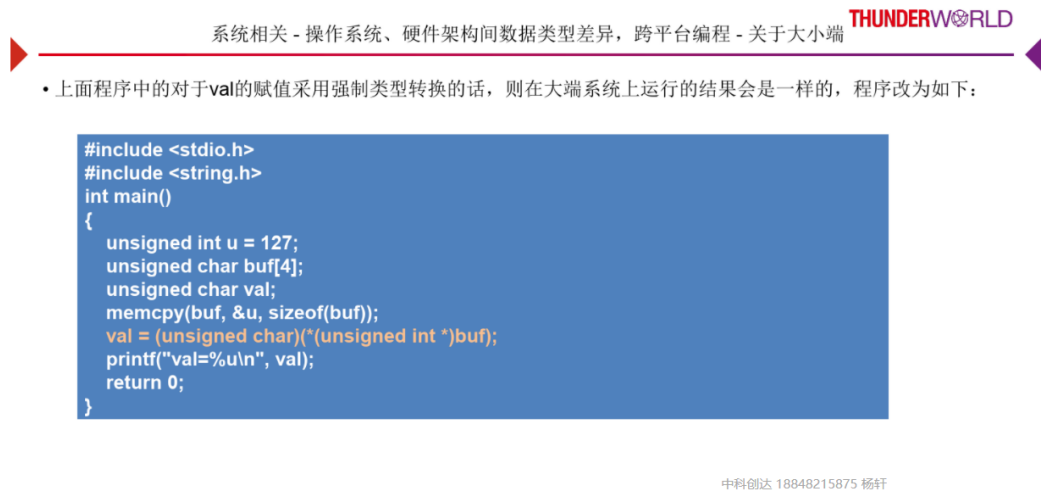
--


THUNDERWORLD
函数 - static函数
• 如下两个源文件用命令“gcc -Wall m.c b.c”一起编译会报“b.c:(.text+0x0): multiple definition of `func'; /tmp/ccLmpUQk.o:m.c:(.text+0x0): first defined here”这个错误：
/* main.c */
#include <stdio.h>
int func()
{
return 1;
}
int main()
{
(void)func();
return 0;
}
/* function.c */
void func()
{
}
• 如果在其中一个文件中给func函数的声明加上static，则编译OK，比如把main.c中的func的声明改为“static int func()”。

1.3 移植

——


THUNDERWORLD
系统相关 - 操作系统、硬件架构间数据类型差异，跨平台编程 - 关于大小端
• 对于小端的情况，如下程序输出的结果是127，而同样的程序在大端系统上运行时输出的结果是0，所以可移植性不好：
#include <stdio.h>
#include <string.h>
int main()
{
unsigned int u = 127;
unsigned char buf[4];
unsigned char val;
memcpy(buf, &u, sizeof(buf));
val = (unsigned char)buf[0];
printf("val=%u\n", val);
return 0;
}
中科创达 18848215875 杨轩


THUNDERWORLD
系统相关 - 操作系统、硬件架构间数据类型差异，跨平台编程 - 关于大小端
• 上面程序中的对于val的赋值采用强制类型转换的话，则在大端系统上运行的结果会是一样的，程序改为如下：
#include <stdio.h>
#include <string.h>
int main()
{
unsigned int u = 127;
unsigned char buf[4];
unsigned char val;
memcpy(buf, &u, sizeof(buf));
val = (unsigned char)(*(unsigned int *)buf);
printf("val=%u\n", val);
return 0;
}
中科创达 18848215875 杨轩

2 规范

2.1 C++

2.1.1 变量命名

--

命令的整体原则

1. 同一性 在编写一个子模块或者派生类的时候，需要遵循其基类或整体模块的命令风格，保持命令风格在整体模块的同一性。
2. 标识符组成--单词 标识符采用英文单词或其组合，应当直观且可以拼读，可望文知意，用词应当准确。
3. 最小化长度&&最大化信息量原则 在保持一个标识符明确意思的同时，应该尽量缩短其长度。
4. 避免过于相似 不要出现紧靠大小写区分的标识符，例如”i”与”I”,”function”与”Function”等。
5. 避免在不同级别的作用域的重名 程序中不要出现名字完全相同的局部变量和全局变量，尽管两者的作用域不同而不会发生语法的错误，但是容易使人误解。
6. 正确命名具有互斥意义的标识符 用正确的反义词命名具有互斥意义的标识符，如：”nMinValue”和”nMaxValue”,”GetName()”和”SetName()”
7. 避免名字出现数字编号 尽量避免名字出现数字的编号，如Value0、Value1、Value2等，除非逻辑上的确需要编号。这是为了防止程序员偷懒，不肯为命名动脑筋从而导致了产生无意义的名字（因为用数字编号最省事）。

--

类/结构的命名

除了异常类等个别情况（不希望用户把类看作一个普通的、正常类的情况）外，C++类结构的命名一般应该遵循以下准则

1. C++类/结构的命名 类的名字都要以大写的字母”C”开头，后跟一个或多个单词。为便于界定，每个单词的首字母需要大写。
2. 推荐的组织形式 类的命名推荐用”名词”或者”形容词+名词”的形式，例如：”CAnalyzer”、”CFVecteor”…
3. 类的名字都要以大写的字

--

C语言结构体的命名

1. 传统的C的命名规则 传统C结构体的名称全部由大写字母组成，单词间使用下划线界定，例如:”SERVICE_STATUS”,”DRIVER_INFO”…

--

函数名称

1. 函数的命名 函数的名称由一个或者多个单词组成。便于界定，每个单词的首字母要大写。 推荐的组织形式 函数名应该使用”动词”或者”动词+名词”（动宾词组）形式。 例如:”GetName()”、”SetName()”、”Erase()”、”Reserve()”…
2. 保护成员函数 保护成员函数的开头应当加上一个下划线以示区分。 例如：”_SetState()”… 私有成员函数
3. 私有成员函数的开头应当加上两个下划线。 例如：”__DestroyImp()”…
4. 虚函数 虚函数习惯用”Do”开头。 例如：”DoRefresh()”…
5. 回调和事件处理函数 回调和事件处理函数习惯以单词”On”开头、 例如：”OnTimer()”…

--

变量命名

..

匈牙利命名法

1. 变量的命名 变量名由作用域前缀+类型的缀+变量标识符组成。为了便于界定，每个单词的首字母要大写。 变量的名字应当使用”名词”或者”形容词+名词”。 例如:”nCode1”,”nMaxWidth1”... 对于某些用途简单明了的局部变量，也可以使用简化的方式。 例如：i,j,k,x,y,z...
2. 类实例的命名 类实例名由作用域前缀+类型的缀+变量标识符组成。 例如:”sTarget1”
3. 作用域前缀 作用域前缀标明了一个变量的可见范围。作用域可以有以下几种： 作用域前缀 说明 无 局部变量 m_ 类的成员变量（member） sm_ 类的静态成员变量（static member） s_ 静态变量（static） g_ 外部全局变量（global） sg_ 静态全局变量（static global） sg_ 静态全局变量（static global） gg_ 进程间共享的数据段全局变量(global global) 除非不得已，否则应该尽可能少用全局变量。
4. 类型的缀 变量前缀 说明 n 整型和位域变量(number) e 枚举型变量(enumeration) c 字符型变量(char) b 布尔型变量(bool) f 浮点型变量(float) p 指针型变量和迭代子(pointer) pfn 特别针对指向函数指针变量和函数指针(pointer of function) g 数组(grid) 一般类实例前缀 i(install) 数据类实例前缀 s
5. ..... 组合使用 例如: List sl

..

驼峰命名法

第一个单词以小写字母开始，从第二个单词开始以后的每个单词的首字母都采用大写字母，例如：myName、myAge，这样的变量名看上去就像骆驼峰一样此起彼伏，因此被称为驼峰命名法。

..

帕斯卡（Pascal）命名法

也叫大驼峰法，与驼峰命名法类似，不过骆驼命名法是首字母小写，而帕斯卡命名法是首字母大写。

..

下划线命名法

与驼峰命名法相似，通过一种方式将不同单词区分开，方便读懂变量含义。与驼峰命名法不同的是，驼峰命名法采用的是首字母大写区分，下划线命名法是在不同单词之间添加下划线。

..

首选匈牙利命名法

Qt选驼峰命名法

--

常量命名

常量名由类型前缀+全大写字母组成，单词间通过下划线来界定， 如 cDELIMITER,nMAX_BUFFER…类型前缀的定义和变量命名规则中的相同。

--

枚举、联合、typedef

枚举、联合以及typdef的命名 枚举、联合以及typedef语句生成的类型名全都是大写字母组成，单词间通过下划线界定， 如：FAR_PROC,ERROR_TYPE…

--

宏、枚举值

宏、枚举值的命名 宏和枚举值全大写字母组成，单词通过下划线界定，如:ERROR_UNKNOWN,OP_STOP…

2 头文件

2.1 快捷

--

Linux

```
//Linux
#include <fcntl.h>            //文件控制
#include <unistd.h>           //符号常量
#include <string.h>           //字符串操作
#include <time.h>             //定义关于时间的函数
#include <utime.h>            //文件时间
#include <syslog.h>           //系统出错日志记录

#include <pthread.h>          //线程控制
#include <signal.h>           //信号机制支持

#include <arpa/inet.h>        //INTERNET定义
#include <net/if.h>           //套接字本地接口
#include <netinet/in.h>       //INTERNET地址族
#include <netinet/tcp.h>      //传输控制协议定义

#include <sys/mman.h>         //内存管理声明
#include <sys/select.h>       //Select函数
#include <sys/socket.h>       //套接字借口
#include <sys/stat.h>         //文件状态
#include <sys/times.h>        //进程时间
#include <sys/types.h>        //基本系统数据类型
#include <sys/un.h>           //UNIX域套接字定义
#include <sys/utsname.h>      //系统名
#include <sys/wait.h>         //进程控制
#include <sys/ipc.h>          //IPC(命名管道)
#include <sys/msg.h>          //消息队列
#include <sys/resource.h>     //资源操作
#include <sys/sem.h>          //信号量
#include <sys/shm.h>          //共享存储
#include <sys/statvfs.h>      //文件系统信息
#include <sys/time.h>         //时间类型
#include <sys/timeb.h>        //附加的日期和时间定义
#include <sys/uio.h>          //矢量I/O操作
```

--

C

```
#include <errno.h>            //定义错误码
#include <float.h>            //浮点数处理
#include <math.h>             //定义数学函数
#include <stddef.h>           //常用常量
#include <signal.h>           //信号机制支持
```

```
#include <stdio.h>            //定义输入 / 输出函数
#include <stdlib.h>           //定义杂项函数及内存分配函数
#include <string.h>           //字符串处理
#include <time.h>             //定义关于时间的函数
```

C++

```
#include <iostream>           //数据流输入 / 输出
#include <fstream>            //文件输入 / 输出
#include <sstream>            //基于字符串的流
#include <cfloat>             //浮点数处理
#include <cmath>              //定义数学函数
#include <csignal>            //信号机制支持

#include <cstddef>            //常用常量
#include <cstdio>             //定义输入 / 输出函数
#include <cstdlib>            //定义杂项函数及内存分配函数

#include <cstring>            //字符串处理
#include <string>             //字符串类
#include <stdbool.h>          //布尔环境
#include <stdint.h>           //整型环境

#include <ctime>              //定义关于时间的函数

#include <deque>              //STL 双端队列容器
#include <list>               //STL 线性列表容器
#include <map>                //STL 映射容器
#include <iterator>           //STL迭代器
#include <queue>              //STL 队列容器
#include <set>                //STL 集合容器
#include <stack>              //STL 堆栈容器
#include <vector>             //STL 动态数组容器
```

2.2 Linux

概览

头文件目录中总共有32个.h头文件。其中主目录下有13个，asm子目录中有4个，Linux子目录中有10个，sys子目录中有5个。

```
#include <a.out.h>                    //a.out头文件，定义了a.out执行文件格式和一

#include <const.h>                    //常数符号头文件，目前仅定义了i节点中i_mod

#include <ctype.h>                    //字符类型头文件，定义了一些有关字符类型判断

#include <errno.h>                    //错误号头文件，包含系统中各种出错号。(Lin

```

```
#include <fcntl.h>                    //文件控制头文件，用于文件及其描述符的操作
#include <signal.h>                   //信号头文件，定义信号符号常量，信号结构以
#include <stdarg.h>                   //标准参数头文件，以宏的形式定义变量参数列
#include <stddef.h>                   //标准定义头文件，定义了NULL, offsetof(T
#include <string.h>                   //字符串头文件，主要定义了一些有关字符串操
#include <termios.h>                  //终端输入输出函数头文件，主要定义控制异步
#include <time.h>                     //时间类型头文件，主要定义了tm结构和一些有
#include <unistd.h>                   //Linux标准头文件，定义了各种符号常数和类型
#include <utime.h>                    //用户时间头文件，定义了访问和修改时间结构
```

--

体系结构相关头文件子目录include/asm

这些头文件主要定义了一些与CPU体系结构密切相关的数据结构、宏函数和变量。共4个文件。

```
#include <asm/io.h>                   //I/O头文件，以宏的嵌入汇编程序形式定义对I
#include <asm/memory.h>               //内存拷贝头文件，含有memcpy()嵌入式汇编宏
#include <asm/segment.h>              //段操作头文件，定义了有关段寄存器操作的嵌
#include <asm/system.h>               //系统头文件，定义了设置或修改描述符/中断门
```

--

Linux内核专用头文件子目录include/linux

```
#include <linux/config.h>             //内核配置头文件，定义键盘语言和硬盘类型（H
#include <linux/fdreg.h>              //软驱头文件，含有软盘控制器参数的一些定义。
#include <linux/fs.h>                 //文件系统头文件，定义文件表结构（file,buf
#include <linux/hdreg.h>              //硬盘参数头文件，定义访问硬盘寄存器端口、
#include <linux/head.h>               //head头文件，定义了段描述符的简单结构，和
#include <linux/kernel.h>             //内核头文件，含有一些内核常用函数的原形定
#include <linux/mm.h>                 //内存管理头文件，含有页面大小定义和一些页面
#include <linux/sched.h>              //调度程序头文件，定义了任务结构task_struc
#include <linux/sys.h>                //系统调用头文件，含有72个系统调用C函数处理
```

```
#include <linux/tty.h>                    //tty头文件，定义了有关tty_io，串行通信方
```

--

系统专用数据结构子目录include/sys

```
#include <sys/stat.h>                     //文件状态头文件，含有文件或文件系统状态结构

#include <sys/times.h>                    //定义了进程中运行时间结构tms以及times()函

#include <sys/types.h>                    //类型头文件，定义了基本的系统数据类型。

#include <sys/utsname.h>                  //系统名称结构头文件。

#include <sys/wait.h>                     //等待调用头文件，定义系统调用wait()和wait
```

--

Linux常用头文件

..

POSIX标准定义的头文件

```
#include <dirent.h>                       //目录项

#include <fcntl.h>                        //文件控制

#include <fnmatch.h>                      //文件名匹配类型

#include <glob.h>                         //路径名模式匹配类型

#include <grp.h>                          //组文件

#include <netdb.h>                        //网络数据库操作

#include <pwd.h>                          //口令文件

#include <regex.h>                        //正则表达式

#include <tar.h>                          //TAR归档值

#include <termios.h>                      //终端I/O

#include <unistd.h>                       //符号常量

#include <utime.h>                        //文件时间

#include <wordexp.h>                      //字符扩展类型

\-------------------------

#include <arpa/inet.h>                    //INTERNET定义

#include <net/if.h>                       //套接字本地接口
```

```
#include <netinet/in.h>              //INTERNET地址族

#include <netinet/tcp.h>             //传输控制协议定义

\------------------------

#include <sys/mman.h>                 //内存管理声明

#include <sys/select.h>               //Select函数

#include <sys/socket.h>               //套接字借口

#include <sys/stat.h>                 //文件状态

#include <sys/times.h>                //进程时间

#include <sys/types.h>                //基本系统数据类型

#include <sys/un.h>                   //UNIX域套接字定义

#include <sys/utsname.h>              //系统名

#include <sys/wait.h>                 //进程控制

POSIX定义的XSI扩展头文件

#include g`"<dlfcn.h>                //动态链接
#include <fmtmsg.h>                   //消息显示结构
#include <ftw.h>                      //文件树漫游
#include <iconv.h>                    //代码集转换使用程序
#include <langinfo.h>                 //语言信息常量
#include <libgen.h>                   //模式匹配函数定义
#include <monetary.h>                 //货币类型
#include <ndbm.h>                     //数据库操作
#include <nl_types.h>                 //消息类别
#include <poll.h>                     //轮询函数
#include <search.h>                   //搜索表
#include <strings.h>                  //字符串操作
#include <syslog.h>                   //系统出错日志记录
#include <ucontext.h>                 //用户上下文
#include <ulimit.h>                   //用户限制
#include <utmpx.h>                    //用户帐户数据库
\----------------------------
#include <sys/ipc.h>                   //IPC(命名管道)
#include <sys/msg.h>                   //消息队列
#include <sys/resource.h>              //资源操作
#include <sys/sem.h>                   //信号量
#include <sys/shm.h>                   //共享存储
#include <sys/statvfs.h>               //文件系统信息
#include <sys/time.h>                  //时间类型
```

```
#include <sys/timeb.h>                  //附加的日期和时间定义
#include <sys/uio.h>                    //矢量I/O操作
```

..

POSIX定义的可选头文件

```
#include <aio.h>                        //异步I/O

#include <mqueue.h>                     //消息队列

#include <pthread.h>                    //线程

#include <sched.h>                      //执行调度

#include <semaphore.h>                  //信号量

#include <spawn.h>                      //实时spawn接口

#include <stropts.h>                    //XSI STREAMS接口

#include <trace.h>                      //事件跟踪
```

2.3 C

--

C

```
#include <assert.h>         //设定插入点

#include <ctype.h>          //字符处理

#include <errno.h>          //定义错误码

#include <float.h>          //浮点数处理

#include <iso646.h>         //对应各种运算符的宏

#include <limits.h>         //定义各种数据类型最值的常量

#include <locale.h>         //定义本地化C函数

#include <math.h>           //定义数学函数

#include <setjmp.h>        //异常处理支持

#include <signal.h>        //信号机制支持

#include <stdarg.h>        //不定参数列表支持

#include <stddef.h>        //常用常量

```

```
#include <stdio.h>          //定义输入 / 输出函数

#include <stdlib.h>         //定义杂项函数及内存分配函数

#include <string.h>         //字符串处理

#include <time.h>           //定义关于时间的函数

#include <wchar.h>          //宽字符处理及输入 / 输出

#include <wctype.h>         //宽字符分类
```

2.4 C++

——

传统C++

```
#include <fstream.h>        //改用<fstream>

#include <iomanip.h>        //改用<iomainip>

#include <iostream.h>       //改用<iostream>

#include <strstrea.h>       //该类不再支持，改用<sstream>中的stringstream
```

——

标准C++

```
#include <algorithm>        //STL 通用算法

#include <bitset>           //STL 位集容器

#include <cctype>           //字符处理

#include <cerrno>           //定义错误码

#include <cfloat>           //浮点数处理

#include <ciso646>          //对应各种运算符的宏

#include <climits>          //定义各种数据类型最值的常量

#include <clocale>          //定义本地化函数

#include <cmath>            //定义数学函数

#include <complex>          //复数类

#include <csignal>          //信号机制支持

#include <csetjmp>          //异常处理支持
```

```
#include <cstdarg>          //不定参数列表支持

#include <cstddef>          //常用常量

#include <cstdio>           //定义输入/输出函数

#include <cstdlib>          //定义杂项函数及内存分配函数

#include <cstring>          //字符串处理

#include <ctime>            //定义关于时间的函数

#include <cwchar>           //宽字符处理及输入/输出

#include <cwctype>          //宽字符分类

#include <deque>            //STL 双端队列容器

#include <exception>        //异常处理类

#include <fstream>          //文件输入/输出

#include <al>               //STL 定义运算函数(代替运算符)

#include <limits>           //定义各种数据类型最值常量

#include <list>             //STL 线性列表容器

#include <locale>           //本地化特定信息

#include <map>              //STL 映射容器

#include <memory>           //STL通过分配器进行的内存分配

#include <new>              //动态内存分配

#include <numeric>          //STL常用的数字操作

#include <iomanip>          //参数化输入/输出

#include <iOS>              //基本输入/输出支持

#include <iosfwd>           //输入/输出系统使用的前置声明

#include <iostream>         //数据流输入/输出

#include <istream>          //基本输入流

#include <iterator>         //STL迭代器
```

```
#include <ostream>        //基本输出流

#include <queue>          //STL 队列容器

#include <set>            //STL 集合容器

#include <sstream>        //基于字符串的流

#include <stack>          //STL 堆栈容器

#include <stdexcept>      //标准异常类

#include <streambuf>      //底层输入/输出支持

#include <string>         //字符串类

#include <typeinfo>       //运行期间类型信息

#include <utility>        //STL 通用模板类

#include <valarray>       //对包含值的数组的操作

#include <vector>         //STL 动态数组容器
```

3 函数

3.1 Linux

3.1.1 用户

> include <unistd.h>

1--

chdir

原型:

```
int chdir(const char * path);
```

功能:

chdir（）用户将当前的工作目录改变成以参数路径所指的目录。

参数:

返回:

实例:

```
main()
{
    chdir("/tmp");
    printf("当前工作目录：%s\n", getcwd(NULL , NULL));
}
```

3.1.2 内核

> #include <linux/module.h>

1--

模块

模块加载函数(必需)：安装模块时被系统自动调用的函数，通过module_init宏来指定，以上例子，指定的加载的函数就是hello_init

模块卸载函数(必需)：卸载模块时被系统自动调用的函数，通过module_exit宏来指定，以上例子，指定的加载的函数就是hello_exit

实例：

```
#include <linux/module.h>
#include <linux/init.h>

static int __init hello_init(void) /* 通过__init声明的函数会在链接的时候被放在.i
{
    printk("<1>Hello,world!n");
    return 0;
}

static void __exit hello_exit(void)
{
    printk("<I>Goodbye,cruel world!n");
}

module_init(hello_init); //表示内核模块入函数是hello_init
module_exit(hello_exit); //表示内核模块退出函数是hello_exit

#if 0 //把这里改成0/1开关下面代码然后编译、加载看是否有报错或者警告？
MODULE_AUTHOR("Brad");
MODULE_LICENSE("Dual BSD/GPL"):
MODULE_DESCRIPTION("Kernel module");
#endif
ifneq ($(KERNELRELEASE),)
        obj-m := hello.o
else
        KERNELDIR ?=/lib/modules/$(shell uname -r)/build
        PWD := $(shell pwd)
default:
        $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
clean:
        rm-rf *~ *.o *.ko *.mod.c .*.cmd modules.order Module.markers Module
endif
```

> #include <linux/module.h>

2--

printk

printk函数和printf的函数实现的功能是一样的，只不过一个运行在内核态，一个运行在用户态。
用printk函数打印时候，内核会根据日志级别，可能把消息打印到当前的控制台上， 这个控制台通常是一个字符模式的终端、一个串口打印机或是一个并口打印机。这些消息正常输出的前提是：日志输出级别小于控制台日志级别(在内核中数字越小优先级越高)。如果没有指定打印级别，默认的基本是<4>,即是KERN_WARNING级别，其定制可以再/kernelprintk.c中找到。
日志级别一共有8个级别，printk的日志级别定义如下(在include/linux/kernel.h中)：
define KERN_EMERG 0
define KERN_ALERT 1
define KERN_CRIT 2
define KERN_ERR 3
define KERN_WARNING 4
define KERN_NOTICE 5
define KERN_INFO 6
define KERN_DEBUG 7
通过读写/proc/sys/kernel/printk文件可读取和修改控制台的日志级别
$ cat /proc/sys/kernel/printk

4 CV代码

4.1 快捷

--

```
int main(int argc,char *argv[]);
int main(int argc,char **argv);
int system(const char *command);

#ifndef XX_H
#define XX_H

#endif

using namespace std;
class A
{
public:

private:

};

int main(int argc, char const *argv[])
{
    /* code */
    return 0;
}
```

文件头部应进行注释，注释必须列出：版权说明、版本号、生成日期、作者、内容、功能、修改日志等。

```
/*********************************************************************************
Copyright: 1988-1999, Huawei Tech. Co., Ltd.
File name: 文件名
Description: 用于详细说明此程序文件完成的主要功能，与其他模块或函数的接口，输出值、取
Author: 作者
Version: 版本
Date: 完成日期
History: 修改历史记录列表， 每条修改记录应包括修改日期、修改者及修改内容简述。
*********************************************************************************
```

函数头部应进行注释，列出：函数的目的/功能、输入参数、输出参数、返回值、调用关系(函数、表)等。

```
/*********************************************
Function: //函数名称
Description: //函数功能、性能等的描述
Calls: //被本函数调用的函数清单
Called By: //调用本函数的函数清单
Table Accessed: //被访问的表(此项仅对于牵扯到数据库操作的程序)
Table Updated: //被修改的表(此项仅对于牵扯到数据库操作的程序)
Input: //输入参数说明，包括每个参数的作// 用、取值说明及参数间关系。
Output: //对输出参数的说明。
Return: //函数返回值的说明
Others: //其它说明
*********************************************/
```