第九章：杂散问题

系统调用和库函数

- 操作系统既是管理者又是服务者，通过系统调用向操作系统发出申请
- C库函数是对系统调用的二次封装，是为了让其更加好用
- 一个功能，库函数可以实现，也有系统调用可以完成，用哪个都行
  - 读写文件API接口：open/write/read/close
  - 读写文件库函数：fopen/fwrite/fread/fclose
  - fopen对open做了封装，为了添加缓冲机制
- 不同平台下（linux、windows、裸机）的API和库函数
  - 完成一个任务调用的API不同，但都可以完成所有任务
  - 库函数在不同os下不同，但相似度更高（人为封装时使其尽可能相似）
  - 一个os上的应用程序不可能直接在另一个os上运行（Qt和Java是跨平台的）

main函数

- main函数是整个程序的入口，其它函数直接间接被main函数调用才能执行，main函数结束=整个程序结束
- main函数三种写法

```
int main(void)
int main(int argc, char **argv)
//argc(argument count表示传几个参数)，argv(argument value是字符串数组)，存多个字符串，每个字符串是给main传的一个参数，argv[0]是传给main的第一个参数...
int main(int argc, char *argv[])

void main(void)//有些情况下会出错，有些情况下不会
```

- 参数是函数的输入，返回值是函数的输出

- 返回值：main函数返回给父进程，表示任务执行成功还是失败（0为成功，负数表示失败，不同负数表示不同失败）
- 传参：main函数可以不传参，传参是为了不重新编译程序就改变程序运行结果的效果

示例

```
#include<stdio.h>
int main(int argc,char**argv)
{
    printf("feel better\n");
    printf("%d\n",argc);
    for(int i=0;i<argc;i++)
        printf("第%d个参数为%s\n",i,argv[i]);
    return 0;
}

结果:
feel better
1
第0个参数为./test
```

- main函数传参通过字符串传入，即使./test 0传入的是“0”，如果操作需要使用字符串函数
- 传多个参数，各个参数之间通过空格间隔（空格数目任意，几个都行）
- 使用argv之前要检验argc有几个（使得程序在错误操作时也可以正常运行）

示例

```
//执行时传参
yuki@YukisPC:~/exercise/C_advance$ ./test z y q
feel better
4
第0个参数为./test
第1个参数为z
第2个参数为y
第3个参数为q
```

程序执行本质

- linux中程序调用的方式：命令行中 ./xx；shell脚本调用执行；程序中调用执行另一个程序（fork、exec）
- linux中新程序执行的本质：父进程fork一个子进程，使用exec把程序丢进进程中执行直到结束（本质上是一个进程的创建、加载、运行、消亡）

- ./xx执行程序的原理：命令行本身是一个进程，命令行中输入./xx，新程序作为命令行进程的子进程执行

void类型

- C语言属于强类型语言，所有变量有自己固定的类型、内存占用、解析方法，弱类型语言所有变量一个类型（一般是字符串类型），使用的时候程序根据需要处理（makefile等）
- void类型变量：a是一个变量，但该变量类型还无法确定（void其实是描述还没具体使用的内存），定义void类型的变量没有意义
- void类型指针：一般用法（C语言不可以有没有类型的变量，但可以有没有类型的内存）
- 典型应用：malloc
  - malloc向堆管理器申请一段内存给当前函数使用，返回指针
  - 指针指向申请的内存，不知道内存将来存什么类型的数据，无法返回具体类型的指针，返回void *类型
  - 最终会被强制类型转换成具体类型

C语言中的NULL

- NULL不是关键字，本质是宏定义

NULL的定义

```
#ifdef _cplusplus /*有时候C和C++混合使用，\
C++编译环境中预先定义了一个宏，程序中用该宏判定编译环境 */
#define NULL 0
#else
#define NULL (void*)0/*C语言中NULL为强制转成void*类型的0，本质是0，但不是数字，是
#endif

//定义不同的原因：C语言中会做严格的类型检查
int *p;
p=0;错误
p=(int*)0正确
p=(void*)0正确
```

- 使用NULL的场景：安全使用指针
  - 指针初始化为NULL，销毁后再指向NULL，使用前判断是否为NULL

示例

```c
int *p=NULL;//定义时立即初始化
p=xx;
if(NULL!=p)
{
  *p//确认p不为NULL时才去解引用
}
p=NULL;//用完p再次等于NULL
```

  - 指针初始化为NULL的原因？大部分CPU中，内存0地址不可以随意访问，保证野指针不会误伤（若指针被无意识解引用，会触发段错误，可以帮助定位错误）

- NULL、 '\0'、'0' 、0区分
  - '\0'是转义字符，ASCII编码值为0，本质是0，字符串结尾
  - '0'是字符，ASCII编码值为48，本质是48，用来获取0的ASCII码
  - 0是数字，没有ASCII编码，本质是0，用于比较数字
  - NULL是表达式，是void*强制类型转换后的0，本质是0，用于确保指针的安全使用

高级语言&低级语言

- 汇编语言（低级语言）是机器码的助记符，不同CPU指令集差异大，因此汇编编程差异大；高级语言封装低级语言，人类思考模式
- 低级语言好几步才能完成的运算，高级语言可以一步完成，i++需要三步（三个周期）：从内存中读取 i 到寄存器 → 对寄存器里的 i 加1 → 把加1后的 i 写回内存中的 i
- 越底层的语言越没有限制，越上层的语言人为限制越多，越接近人，编程难度越低
- 人与机器总劳动量守恒，语言越简单，编译器设计越复杂，语言越复杂，编译器越简单，CPU执行效率越高→快速开发用高级语言，追求效率用低级语言（C语言编写内核）

临时变量

- int b=(int) a; //三个步骤，构建临时变量x→a整数部分赋值给x→x赋值给b→销毁x

▾ 示例

```c
int b;
float a;
b = 10;
a = b/3;//得到的结果为3.0000而不是3.3333
/*步骤：计算b/3
-> 运算得到的结果强制类型转换成float，存到临时变量x
-> x赋值给a
-> 销毁x */
```

```
9
```

顺序结构

- 编译与顺序结构：一个c程序多个.c文件，编译时每个.c文件按照从前往后的顺序编译，也因此函数/变量要先定义/声明才可以使用
- 链接与顺序结构：链接顺序由链接脚本指导完成，若链接脚本没有指定则链接器按照内部规则排布（可能按照读入的顺序，可能按照字母顺序）

程序调试的debug宏

- 常见方案：单步调试、裸机LED调试、打印信息调试、log文件调试
  - 单步调试：IDE中使用调试器模拟调试，或者使用Jlink（限制性大，速度慢）
  - 裸机使用LED、蜂鸣器等硬件调试，适合单片机
  - printf打印调试，在合适的地方加上printf函数，常用且推荐
  - 使用log文件（日志文件），系统运行过程中会打印一些调试信息，以便后续查找问题，适合大型程序和系统级调试（调试信息少找不到问题，太多会淹没有用信息）
- 调试（DEBUG）版本和发行（RELEASE）版本
  - 调试（DEBUG）版本：包含了调试信息输出的版本，运行时打印调试信息/log文件，辅助判断问题所在；输出调试信息占用系统资源，拖慢了系统运行的速度，性能低于发行（RELEASE）版本
  - 发行（RELEASE）版本：功能和调试（DEBUG）版本相同，但去掉了所有调试信息，无法调试，适合最终发布程序，效率高
  - 其实是一套代码，靠debug宏控制版本

  示例

```
#ifdef DEBUG
#define dbg() printf()
#else
#define dbg()//把debug定义成空，编译时生成没有任何调试信息的代码
#endif
```

- **debug宏分析**

  应用程序中的debug

```c
#ifdef DEBUG
#define DBG(...) fprintf(stderr, " DBG(%s, %s(), %d): ", __FILE__,
__FUNCTION__, __LINE__); fprintf(stderr, __VA_ARGS__)
/*
输出到stderr标准错误中
" DBG(%s, %s(), %d): "是输出的字符串
__FILE__, __FUNCTION__, __LINE__是预定义宏
（C语言自己定义的宏，有特殊含义，__FILE__代表当前正在编译程序的文件名）
__VA_ARGS__:将左边宏中 ... 的内容原样抄写在右边 __VA_ARGS__ 所在的位置

这条输出当前这条调试信息在哪个文件、哪个函数、第多少行
*/
#else
#define DBG(...)
#endif
```

内核中的debug

```c
#ifdef DEBUG_S3C_MEM
#define DEBUG(fmt, args...)     printk(fmt, ##args)
#else
#define DEBUG(fmt, args...)     do {} while (0)
//使用do {} while (0)，而不是空，为了保证后面有、无分号时都是对的
#endif
```

其它

- 数据类型决定变量的内存占用和解析方法
- 编译器将源代码转成二进制，学习编程的关键为学习编译器的习性，反映在编程中就是编程语言的语法
- C语言学习不应当把重心放在做题上，而是在项目实践中体会C语言的规则，对规则理解的越深入，编程能力越强