目录

[1-1 ※写一个"标准"宏MIN ，这个宏输入两个参数并返回较小的一个 2](#_Toc485373666)

[1-2 ※用预处理指令#define 声明一个常数，用以表明1年中有多少秒 2](#_Toc485373667)

[1-3 ※把上面的define.c 写成一个Makefile文件并编译出可执行文件 2](#_Toc485373668)

[1-4 ※交换两个数的值 3](#_Toc485373669)

[1-5 ※int main（int argc char \*argv[]）的用法 3](#_Toc485373670)

[1-6 ※fork（）的用法 4](#_Toc485373671)

[1-7 ※exit（）用法 6](#_Toc485373672)

[1-8 ※sleep（）用法 6](#_Toc485373673)

[1-9 ※用fprintf()/fscanf() 6](#_Toc485373674)

[1-10 ※用extern 存储类 7](#_Toc485373675)

[1-11 ※大端小端判断 8](#_Toc485373676)

[1-12 ※Linux命令 set 9](#_Toc485373677)

[1-12 ※C语言字符操作 9](#_Toc485373678)

[1-13※struct的初始化 10](#_Toc485373679)

[1-14※关键字volatile有什么含意?并给出三个不同的例子 10](#_Toc485373680)

[1-15※unsigned int a=1; int b=-20 (a+b)>0 11](#_Toc485373681)

[1-16※Linux I/O操作 open read write open（系统） 11](#_Toc485373682)

[1-17※Linux socket 本地通信 13](#_Toc485373683)

# 1-1 ※写一个"标准"宏MIN ，这个宏输入两个参数并返回较小的一个

#include<stdio.h>

**#define MIN(x,y) x<y? x:y //后面的比较：输出小的输 x,Y的值顺序不要反**

int main()

{

static int a=5;

static int b=9;

int c;

scanf("plate 输入数字：%d%d\n",&a,&b);

printf("a=%d\n b=%d\n",a,b);

**c=MIN(a,b); //使用**

printf("c= %d\n",c);

}

# 1-2 ※用预处理指令#define 声明一个常数，用以表明1年中有多少秒

#include<stdio.h>

**#define COUNT\_YEAR\_MIN ((unsigned long)(60 \* 60 \* 24 \* 365))//注长度**

int main()

{

int c;

**c=COUNT\_YEAR\_MIN; //使用**

printf("c=%d\n",c);

}

# 1-3 ※把上面的define.c 写成一个Makefile文件并编译出可执行文件

edit: define.o

**cc -o define define.o @ 必须tab键开头 @**

define.o:define.c

cc -c define.c

clean:

rm **define** define.o

**带变量的makefile**

**objects = define.o**

edit:$(objects)

cc -o define $(objects)

define.o: define.c

cc -c define.c

clean:

rm define $(objects)

# 1-4 ※交换两个数的值

#include<stdio.h>

**void swap(int \*a,int \*b)**

{

int c;

c=\*a;

\*a=\*b;

\*b=c;

}

int main()

{

int d=10;

int f=55;

swap(&d,&f);

printf("%d\n%d\n",d ,f);

}

# 1-5 ※int main（int argc char \*argv[]）的用法

#include <stdio.h>

int main(int argc,char\* argv[])

{

int i;

for (i=0;i<argc;i++)

{

printf("%s\n",argv[i]);

printf("%d\n",i);

printf("%d\n",argc);

}

system("pause");

return 0;

}

**解释：**

argc：表示执行可运行程序后的参数

argv[]: 此可执行文件的存储路径、程序变量

**如：**执行./main 无参数

argc: 为1

argv[i]：为工程的路径和名称如：C:\Qt\Qt5.3.2\Tools\QtCreator\bin\build-untitled2-Desktop\_Qt\_5\_3\_MinGW\_32bit-Debug\debug\debug\untitled2.exe

运行结果：

C:\Qt\Qt5.3.2\Tools\QtCreator\bin\build-untitled2-Desktop\_Qt\_5\_3\_MinGW\_32bit-Debug\debug\debug\untitled2.exe

i=0

argc=1

**如：**执行./main sss.c(什么都行)

argc: 为2

argv[i]：如下

运行结果：

C:\Qt\Qt5.3.2\Tools\QtCreator\bin\build-untitled2-Desktop\_Qt\_5\_3\_MinGW\_32bit-Debug\debug\debug\untitled2.exe

i=0

argc=2

ssss.c

i=1

argc=2

# 1-6 ※fork（）的用法

#include<sys/types.h>

#include<unistd.h>

#include<stdio.h>

int main(int argc,char\* argv[])

{

pid\_t pid;

pid = fork();

if (pid < 0 ) {

printf("error!");

} else if( pid ==0 ) {

printf("This is the child process!\n");

} else{

printf("This is the parent process!\n child process id = %d\n", pid);

}

return 0;

}

疑问：为什么每次执行结果不一样

无固定顺序是因为子程序与父程序并发导致的，因为在用fork（）函数创建的3个进程同时在cpu中运行，因为3个进程的本质是一样的，所以就会抢占资源，因此形成一个队列，因为每次的排序不同，故此输出的结果也会不同。

**●fork的用法二**

#include<sys/types.h>

#include<unistd.h>

#include<stdio.h>

int main(int argc,char\* argv[])

{

pid\_t pid;

pid = fork();

if(pid < 0)

printf("error\n");

else if(pid == 0){ //等于0是子进程

printf("The child pid = %d\n",getpid());

}

else{

printf("The parent pid = %d\n",getpid());

}

}

Fork 返回值：**0 成功创建子进程**

>0 成功创建父进程，返回进程PID

-1 失败返回父进程

**运行结果：**

**The parent pid = 16938**

**The child pid = 16939**

# 1-7 ※exit（）用法

主要用到子函数中

exit（0）：正常运行程序并退出程序；

exit（1）：非正常运行导致退出程序；

* + main函数调用return
  + 调用exit
  + 调用\_exit
  + 调用abort
  + 被一个信号终止

区别 abort与exit

exit会做一些释放工作**：释放所有的静态的全局的对象，缓存，关掉所有的I/O通道**，然后终止程序。如果有函数通过atexit来注册，还会调用注册的函数。不过，如果atexit函数扔出异常的话，就会直接调用terminate。  
abort：立刻terminate程序，**没有任何清理工作**。

# 1-8 ※sleep（）用法

Sleep（1）：1s 秒

Usleep( 1000) 1000 毫秒

# 1-9 ※用fprintf()/fscanf()

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#define FILE\_PATH "./demo.txt" // 文件路径

int main(){

int i=20;

char name[20]="yang"; // 姓名

int age=20; // 年龄

int studNo=45454; // 学号

float score=222; // 平均分

FILE \*fp; // 文件指针

// 判断文件是否能够正确创建/打开

if( (fp=fopen(FILE\_PATH,"wt+")) == NULL ){

perror(FILE\_PATH);

exit(1);

}

// 从控制台输入学生信息并写入文件

printf("(please enter以空格分隔)：\n");

// scanf("%s %d %d %f", name, &age, &studNo, &score);

while(1)

{

static age=555;

studNo--;

score--;

fprintf(fp,"%s\t%d\t%d\t%f\n", name, age, studNo, score);

// 刷新缓冲区，将缓冲区的内容写入文件

fflush(fp);

// 重置文件内部位置指针，让位置指针指向文件开头

rewind(fp);

// 从文件中读取学生信息

//printf("read file:\n");

while(fscanf(fp, "%s\t%d\t%d\t%f", name, &age, &studNo, &score) != EOF){

printf("%s %d %d %f\n", name, age, studNo, score);

}

if(i<0) break;

i--;

}

fclose(fp);

return EXIT\_SUCCESS;

}

# 1-10 ※用extern 存储类

extern 修饰符通常用于当有**两个或多个文件共享相同的全局变量或函数**的时候。

**如 extern\_1.c**

#include <stdio.h>

**extern void swap(int \*a,int \*b)**

{

int c;

c=\*a;

\*a=\*b;

\*b=c;

}

**如extern\_2.c**

#include <stdio.h>

int main()

{

int i=0,h=99999;

printf("before i= %d\t h= %d\n",i,h);

**swap(&i,&h);//能使用其他文件的函数**

printf("after i= %d\t h= %d\n",i,h);

return 0;

}

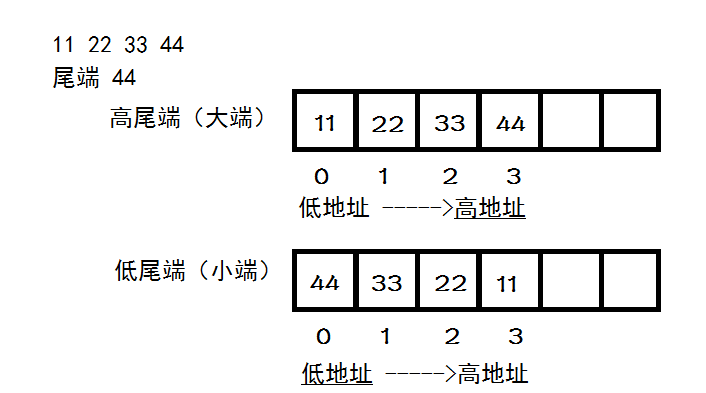
输出：before i= 0 h= 99999

after i= 99999 h= 0

# 1-11 ※大端小端判断

大端模式：是指数据的高字节保存在内存的低地址中，而数据的低字节保存在内存的高地址端。

小端模式，是指数据的高字节保存在内存的高地址中，低位字节保存在在内存的低地址端。

intel芯片是小端(修改分区表时要注意)，**单片机一般为大端**

编程检测：

#include <stdio.h>

int check()

{

union UN //union是共享内存的

{

char c;

int i;

}un;

un.i = 1;

return un.c;

}

int main(void)

{

if(check()==1)

printf("little\n");

else

printf("big\n");

return 0;

}

# 1-12 ※Linux命令 set

作用主要是**显示系统中已经存在的shell变量**，以及**设置shell变量的新变量值。**使用set更改shell特性时，符号"+"和"-"的作用分别是打开和关闭指定的模式  
来自: <http://man.linuxde.net/set>

**set 用来显示本地变量**  
env 用来显示环境变量  
export 用来显示和设置环境变量s

set 显示当前shell的变量，包括当前用户的变量  
env 显示当前用户的变量  
export 显示当前导出成用户变量的shell变量

每个shell有自己特有的变量（set）显示的变量

# 1-13 ※Linux命令 man 1,2,3..

1、standard command （标准命令）

**2、System calls （系统调用）  
3、Library functions （库函数）**  
4、Special devices （设备说明）  
5、File formats （文件格式）

# 1-15 ※C语言字符操作

|  |  |
| --- | --- |
| 1 | **strcpy(s1, s2);** 复制字符串 s2 到字符串 s1。 |
| 2 | **strcat(s1, s2);** 连接字符串 s2 到字符串 s1 的末尾。 |
| 3 | **strlen(s1);** 返回字符串 s1 的长度。 |
| 4 | **strcmp(s1, s2);** 如果 s1 和 s2 是相同的，则返回 0；如果 s1<s2 则返回小于 0；如果 s1>s2 则返回大于 0。 |
| 5 | **strchr(s1, ch);** 返回一个指针，指向字符串 s1 中字符 ch 的第一次出现的位置。 |
| 6 | **strstr(s1, s2);** 返回一个指针，指向字符串 s1 中字符串 s2 的第一次出现的位置。 |

# 1-16※struct的初始化

#include <stdio.h>

#include <string.h>

struct stud //定义一个结构体

{

int age;

char \*name;

int score;

};

void out(struct stud \*s) //**定义一个指向结构体的指针**

{

printf("age=%d\n",s->age);//取出初始化结构体的值

printf("name=%s\n",s->name);/\*用指向该结构的指针,访问结构的成员，必须使用 -> 运算符\*/

printf("score=%d\n",s->score);

}

int main()

{

struct stud std= //初始化结构体stud 类型std

{ //**声明 std，类型为 stud**

.age = 18,

.name = "yang",

.score = 99,

};

out(&std); //**通过传 std 的地址来输出 std 信息**

}

# 1-17※关键字volatile有什么含意?并给出三个不同的例子

# 1-18※unsigned int a=1; int b=-20 (a+b)>0

#include <stdio.h>

int main(void)

{

unsigned int a=2;

int b=-20;

if((a+b)>10000000)**//如果无符号的变量和有符号的变量相运算时会是一个非常大的数 [除了-运算]**

{

printf("a+b=%u\n",a+b);

printf("a-b=%u\n",a-b);

printf("a\*b=%u\n",a\*b);

printf("b/a=%u\n",b/a);

}else

{

printf("a+b<0\n");

}

return 0;

}

a+b=4294967278

a-b=22

a\*b=4294967256

b/a=2147483638

%d 有符号10进制整数 %u 无符号10进制整数

# 1-19※Linux I/O操作 open read write open（系统）

flags：（falgs参数可通过“|”组合构成，只读、只写、读写 这三种方式是互斥的，不可同时使用）  
        O\_RDONLY（只读方式打开），  
        O\_WRONLY（只写方式打开），  
        O\_RDWR（读/写方式打开），  
        O\_CREAT（如果文件不存在，就创建新的文件），  
        O\_EXCL（如果使用O\_CREAT时，文件存在，则可返回错误信息），  
        O\_TRUNC（如果文件已存在，且以只读或只写方式打开，则先删除文件中的原有数据），  
        O\_APPEND（以添加方式打开文件，在打开文件的同时，文件指针指向文件末尾）

 为了防止对文件的意外操作，往往要以合适的方式打开文件（只读，只写），每个文件只负责一个特定的用  
    途，有利于提高这些文件的重复利用

#include <unistd.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <fcntl.h>

#include <stdio.h>

#define NAME "file.log"

int main(void){

int fd,size,w;//fd：文件描述符，用于判断打开|新建文件是否成功，size：读取到的字节数

char s[] = "hellossssssssss\n----->\n52661314ll";//需要写入的字符串

char **buffer**[80];//存储读出数据的缓冲区

fd = **open**(NAME, **O\_WRONLY | O\_CREAT**);//以只写方式打开文件，若不存在则创建文件

if (**fd == -1**){

printf("Open or create file failed.\n");

return -1;

}

w=**write**(fd, s, sizeof(s));//向该文件写入字符串

printf("w==%d\n",w);

close(fd);

fd = **open**(NAME, O\_RDONLY);//以只读方式打开文件

if (fd == -1){

printf("Open file failed.\n");

return -1;

}

size = **read**(fd, buffer, sizeof(buffer));//读取文件内容到buffer并返回读取的字符个数

printf("size==%d\n",size);/**/w=size 字符长度**

**close**(fd);

printf("%s", buffer);

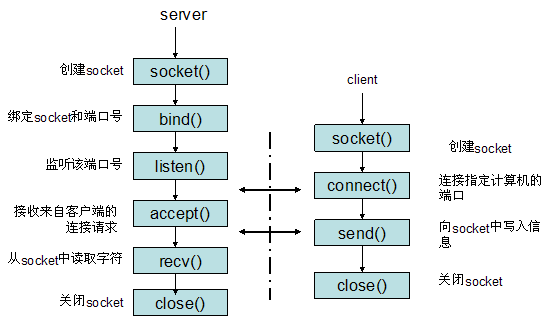
return 0;

}

# 1-20※Linux socket 本地通信

* **理论篇**

两个进程有唯一的PID，在本地进程通讯中我们可以使用PID来唯一标示一个本地进程，但PID只在本地唯一，网络中的两个进程PID冲突几率很大。IP层的ip地址可以唯一标示主机，而TCP层协议和端口号可以唯一标示主机的一个进程，这样我们可以利用ip地址＋协议＋端口号唯一标示网络中的一个进程。socket起源于Unix，而Unix/[Linux](http://lib.csdn.net/base/linux)基本哲学之一就是“一切皆文件”，都可以用“打开open –> 读写write/read –> 关闭close”模式来操作。Socket文件。



* **Socket用到的函数**

**int socket(int domain, int type, int protocol)**

eg:int socket(AF\_INET,SCOK\_STREAM,0) 【常用】

 -**domain**：即协议域，又称为协议族（family）。常用的协议族有，**AF\_INET（PF\_INET没有区别）**、AF\_INET6、AF\_LOCAL（或称AF\_UNIX，Unix域socket）、AF\_ROUTE等等。协议族决定了socket的地址类型，在通信中必须采用对应的地址，如AF\_INET决定了要用ipv4地址（32位的）与端口号（16位的）的组合、**AF\_UNIX决定了要用一个绝对路径名作为地址**。

  **-type：**指定socket类型。常用的socket类型有，SOCK\_STREAM、SOCK\_DGRAM、SOCK\_RAW、SOCK\_PACKET、SOCK\_SEQPACKET等等（socket的类型有哪些？）。

 -**protocol**：就是指定协议。常用的协议有，IPPROTO\_TCP、IPPTOTO\_UDP、IPPROTO\_SCTP、IPPROTO\_TIPC等，它们分别对应TCP传输协议、UDP传输协议、STCP传输协议、TIPC传输协议（这个协议我将会单独开篇讨论！）。

注意：并不是上面的type和protocol可以随意组合的，如SOCK\_STREAM不可以跟IPPROTO\_UDP组合。**当protocol为0时（常用），会自动选择type类型对应的默认协议**。

当我们调用**socket**创建一个socket时，返回的socket描述字它存在于协议族（address family，AF\_XXX）空间中，但没有一个具体的地址。如果想要给它赋值一个地址，就必须调用**bind()函数（绑定一个具体的地址）**，否则就当调用connect()、listen()时系统会自动随机分配一个端口。

# 1-17※同步和异步区别