软件工程师丛书

C 语言函数大全

张　翔　裘　岚　张晓芸　等编著

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

出 版 说 明

随着我国加入 WTO，现代化建设也将以前所未有的步伐向前迈进。我们面临更大的挑战，也面临更多的机遇。一个不争的事实是计算机的应用普及将更加深入，将需要数量更多、水平更高的软件工程师。

我国的软件工程师队伍已有了长足的发展，软件开发水平已有了长足的进步。作为中国人，我们期盼的是中国软件业走自主创新之路，在世界上的地位越来越高。作为出版工作者，为发展我国的软件事业尽最大努力，是我们义不容辞的责任，这正是我们于 1999 年底推出《软件工程师》丛书的初衷。

目前这套丛书已出版了 40 多种。从市场销售和读者反馈的情况看，这套丛书已经得到了读者的首肯和厚爱，这也是对我们下一步工作的激励。

可以说，计算机应用系统的多样化、规模化和复杂化对软件工程师提出了更高的要求，同时也为软件工程师提供了更多的施展个人才华的机会。

针对这种形势，我们正在扩充《软件工程师》丛书的选题范围，进一步界定这套丛书的特色，并把丛书按如下类型整合。

一是开发类，通过大量实例说明如何使用各种流行的高级语言、工具类软件开发不同的应用系统，说明开发思想、开发过程、难点及其解决方案。为了适应我国软件工程师开发综合软件系统的需求，我们把包含编程功能在内的高级应用软件的开发应用也纳入到丛书中。

二是技巧类，通过大量实例说明在不同应用系统开发过程中，有关缩短开发周期、提高开发质量、解决开发中的疑难问题的各种技巧。

三是技术类，介绍软件开发的有关理论和技术，以及在实践中的应用，如系统分析与系统设计、软件测试和系统安全等。

四是手册类，即每个软件工程师必备的案头书。

在新的一年开始之际，这套丛书从内容、开本、印刷及装帧等方面都将以全新的面貌与广大读者见面，目的在于使其更受读者的欢迎，每本书能容纳更多的信息。

我们以为软件工程师提供图书信息服务为宗旨，坚持以图书质量为生命。我们希望《软件工程师》丛书能对读者有所帮助，希望读者提出更多的宝贵建议和意见，包括工作中遇到的技术难点、疑点和问题。希望更多的作者加入我们的专家行列，推介自己的实践经验和累累硕果。我们的网址是 www.phei.com.cn，请和我们联系。

为了我国软件业的更加美好的明天，让我们共同努力。

電子工業出版社

前　言

C 语言是多年来国内外得到迅速推广使用的一种现代语言。C 语言功能丰富、表达能力强、使用灵活方便、应用面广、可移植性好，既具有高级语言的优点，又具有低级语言的许多特点。现在，C 语言已不仅为计算机专业人员所使用，而且也受到了广大计算机软件编程爱好者的喜爱。

随着操作系统的不断发展和演变，目前广泛使用的 C 语言可以大概分为 Microsoft C、Turbo C 和 Unix C 等 3 种类型。这 3 类 C 语言之间既有相同点，也存在差异。为了帮助大家更好地学习和使用 C 语言，本书按照功能分类介绍了这 3 类 C 语言函数的功能和使用方法，逐条给出了其语法格式、功能、使用说明等内容，对一些重要函数还给出了使用样例。同时，为了使读者在查找函数时更加方便快捷，本书还在说书的末尾给出了函数名称索引。

近年来随着 Internet 的普及，Unix 操作系统日趋流行，在 Unix 平台上编写应用程序的需求也越来越大。为了满足读者的需要，本书在编写时收录了 GNU 函数库中的全部内容，相信完全能够作为 Unix 开发人员的参考工具使用。

本书第 1 章内容为 Unix 的 C 函数库 GNU C，其中包括错误报告、内存分配、字符处理、字符串和数组工具、流输入/输出、底层输入/输出、文件系统接口、管道和队列、套接口、底层终端接口、数学函数、初等数学函数、搜索和排序、匹配、日期和时间、扩展字符集、本地和国际化、非本地退出、信号处理、进程的启动和终止、进程、任务控制、用户和群组、系统信息、系统配置参数等。第 2 章为 Borland 公司的 Turbo C 函数库，其中包括 Turbo C 中 21 个头文件的所有函数说明。第 3 章为 Microsoft 公司的 MSC 中 20 个头文件中的所有函数说明。

本书第 1 章由张翔编写，第 2 章由裘岚编写，第 3 章由张晓芸编写。此外参与本书编写和整理工作的还有吴世祥、许进等同志。

由于编者的水平有限，书中可能存在不当之处，请读者提出宝贵意见。

作　者

2002 年 3 月

内 容 提 要

本书介绍了包括 Unix C，Turbo C 和 Microsoft C 共 3 类 C 语言函数，列出了每个函数的功能、语法格式和使用说明等，涉及到了当今 C 语言使用的各个层面。

本书适用于使用 C 语言进行软件开发的人员，是一本覆盖面大、用途广泛的工具书。本书也可作为 C 语言初学者的参考书。

图书在版编目（CIP）数据

C 语言函数大全 / 张翔等编著．—北京：电子工业出版社，2002.4
（软件工程师丛书）
ISBN 7-5053-7538-5

Ⅰ.C... Ⅱ.张... Ⅲ.C 语言－程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2002）第 017051 号

责任编辑： 段来盛
印　　刷： 北京市天竺颖华印刷厂
出版发行： 电子工业出版社 http://www.phei.com.cn
北京市海淀区万寿路 173 信箱 邮编 100036
经　　销： 各地新华书店
开　　本： 787×1092 1/16 印张：26.5 字数：626 千字
版　　次： 2002 年 4 月第 1 版 2002 年 4 月第 1 次印刷
印　　数： 6000 册 定价： 40.00 元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系。联系电话：（010）68279077

目　　录

第 1 章

Unix C 函数

本章分类介绍每个 Unix C 函数的功能、原型、原型所在的包含文件、使用说明等内容，对一些重要函数还给出了样例。

1.1 错 误 报 告

perror

〖功能〗该函数在 stderr 流中输出错误消息。

〖原型〗void perror (const char *message)

〖位置〗stdio.h (ISO)

〖说明〗如果调用 perror 时使用的 message 参数是一个空指针或者一个空字符串，则 perror 函数将打印与 errno 相对应的错误信息，以及一个换行符。

如果提供的 message 参数非空，则 perror 函数将作为前缀首先输出该字符串的内容，然后添加一个冒号和空格字符，最后是 errno 相对应的错误信息。

strerror

〖功能〗将由参数 errnum 指定的错误代码映射到描述错误信息的串，返回值是指向串的指针。

〖原型〗char * strerror (int errnum)

〖位置〗string.h (ISO)

〖说明〗errnum 的值通常来自 errno，不应修改 strerror 所返回的串，以后再次调用 strerror 时会重写该串。

strerror_r

〖功能〗该函数与 strerror 基本相同，只是使用该函数时错误信息写入的缓存区由用户提供，并且从 buf 开始，长度为 n 字节，而不是程序中所有线程共享的静态分配缓存区。

〖原型〗char * strerror_r (int errnum, char *buf, size_t n)

〖位置〗string.h (GNU)

〖说明〗至多写 n 个字符，除非用户选择足够大的缓存。由于无法保证返回的串属于当前线程的最后调用，所以常用于多线程程序。

该函数是 GNU 扩展，其声明在 string.h 中。

1.2 内 存 分 配

alloca

〖功能〗分配存储块。

〖原型〗void * alloca (size_t size)

〖位置〗stdlib.h (GNU, BSD)

〖说明〗该函数的返回值是 size 字节长的存储块地址，该存储块在调用函数的栈帧中分配。

不要在函数调用的参数中使用 alloca 函数，这将产生不可预知的结果。因为用于 alloca 的栈空间在堆栈中将处在函数参数使用空间的中间位置，所以要避免出现像 foo(x, alloca (4), y)这样的调用。

calloc

〖功能〗分配内存并且将内存清 0。

〖原型〗void * calloc (size_t count, size_t eltsize)

〖位置〗malloc.h, stdlib.h (ISO)

〖说明〗该函数分配一块包含 count 个元素的内存，其中每个元素的大小为 eltsize。其内容在 calloc 返回之前清 0。

例如，可以定义 calloc 如下。

```
void * calloc (size_t count, size_t eltsize)
{
  size_t size = count * eltsize;
  void *value = malloc (size);
  if (value != 0)
    memset (value, 0, size);
  return value;
}
```

但是通常不能保证 calloc 在内部调用 malloc。如果某个应用程序在 C 函数库之外提供自己的 malloc/realloc/free，则也应当同时提供 calloc。

cfree

〖功能〗释放存储空间。

〖原型〗void cfree (void *ptr)

〖位置〗stdlib.h (Sun)

〖说明〗与 free 函数的功能类似。提供该函数是为了与 SUN OS 兼容，因此应当尽量使用 free 函数。

free

〖功能〗释放由指针 ptr 指向的存储空间。

〖原型〗void free (void *ptr)

〖位置〗malloc.h, stdlib.h (ISO)

mallinfo

〖功能〗该函数返回结构中当前的动态内存使用情况，该结构为 struct mallinfo 类型。

〖原型〗struct mallinfo mallinfo (void)

〖位置〗malloc.h (SVID)

malloc

〖功能〗该函数用于分配一个新的存储块。

〖原型〗void * malloc (size_t size)

〖位置〗malloc.h, stdlib.h (ISO)

〖说明〗该函数返回一个指针，指向新分配的 size 字节长的存储块，如果返回空指针，则说明没有执行分配动作。

存储块的内容没有定义，必须自己进行初始化(或者使用 calloc 函数实现)。通常需要将返回值强制转换类型。以下提供一些样例，说明如何使用 memset 库函数进行初始化。

```
struct foo *ptr;
...
ptr = (struct foo *) malloc (sizeof (struct foo));
if (ptr == 0) abort ();
memset (ptr, 0, sizeof (struct foo));
```

事实上不强制转换类型也可以将 malloc 的结果保存在任何一个指针变量中，因为 ISO C 在必要时自动将 void *类型转换为其他类型的指针。但是如果不是使用赋值操作符或者需要在传统 C 环境中运行代码，则必须强制转换类型。

为字符串分配空间时，malloc 的参数必须是字符串长度加 1。这是因为尽管字符串结尾的空字符不计算在字符串的长度内，但是该字符同样需要一个字节长的空间。例如：

```
char *ptr;
...
ptr = (char *) malloc (length + 1);
```

mcheck

〖功能〗调用 mcheck 函数将通知 malloc 执行临时的一致性检查，诸如写操作超过 malloc 分配的存储块结尾等错误情况。

〖原型〗int mcheck (void (*abortfn) (enum mcheck_status status))

〖位置〗malloc.h (GNU)

〖说明〗abortfn 参数是发现不一致情况之后调用的函数。如果该参数提供一个空指针，则 mcheck 使用一个默认函数来打印一条消息，然后调用 abort 函数。所提供的函数在调用时带有一个参数，说明所检查到的不一致类型。

如果在使用 malloc 执行分配动作之后再进行分配检查，则为时已晚。此时 mcheck 不会做任何动作，并返回-1；否则返回 0(表示成功完成)。

为了尽早安排调用 mcheck，最简单的方法是在链接程序时使用-lmcheck 选项，之后将完全不需要再改动源程序。

memalign

〖功能〗该函数在 boundary 的倍数位置分配一块 size 大小的字节。

〖原型〗void * memalign (size_t boundary, size_t size)

〖位置〗malloc.h, stdlib.h (BSD)

〖说明〗boundary 必须是 2 的阶乘的幂。函数执行时首先分配一个较大的块，然后返回块中一个位于指定边界的地址。

obstack_alloc

〖功能〗该函数在对象堆栈中分配 size 个字节的存储块，不进行初始化，然后返回其地址。

〖原型〗void * obstack_alloc (struct obstack *obstack-ptr, int size)

〖位置〗obstack.h (GNU)

〖说明〗这里的 obstack-ptr 用于指定在哪个对象堆栈中分配块，即对象堆栈中 struct obstack 对象的地址。每个 obstack 函数或宏都要求指定一个 obstack-ptr 作为第 1 参数。

如果需要分配一部分新的内存，则该函数调用对象堆栈的 obstack_chunk_alloc 函数；如果 obstack_chunk_alloc 返回，则该函数返回一个空指针。这种情况下，对象堆栈中分配的内存量没有改变。如果所提供的 obstack_chunk_alloc 函数在内存不足时调用 exit 或者 longjmp，那么 obstack_alloc 不返回空指针。

例如，下面的函数在一个指定的对象堆栈中分配字符串 str 的备份。

```
struct obstack string_obstack;
char *
copystring (char *string)
{
  size_t len = strlen (string) + 1;
  char *s = (char *) obstack_alloc (&string_obstack, len);
  memcpy (s, string, len);
  return s;
}
```

obstack_base

〖功能〗该函数返回对象堆栈 obstack-ptr 中当前增长对象的起始地址。

〖原型〗void * obstack_base (struct obstack *obstack-ptr)

〖位置〗obstack.h (GNU)

〖说明〗如果立即结束对象则可以即刻得到返回地址；如果对象还需要增长，则可能超出当前的块，其地址也将会发生改变。

如果没有正在增长的对象，则该函数的返回值说明了下一个分配对象的起始位置(再次假设其符合当前块的大小)。

obstack_blank_fast

〖功能〗该函数为对象堆栈 obstack-ptr 中的增长对象添加 size 个字节，但不进行初始化。

〖原型〗void obstack_blank_fast (struct obstack *obstack-ptr, int size)

〖位置〗obstack.h (GNU)

〖说明〗使用 obstack_room 函数检查空间时，如果没有足够的空间可添加，那么使用快速增长函数将不够安全。在这种情况下，只需使用相应的普通增长函数，即立即将对象拷贝至新块，然后再次提供大量可用空间。

因此，每次使用普通增长函数之前，都需要使用 obstack_room 函数检查空间是否充足。一旦对象被拷贝至新块，就可以再次提供充足的空间，从而程序可以再次使用快速增长函数。

以下是一个样例。

```
void
add_string (struct obstack *obstack, const char *ptr, int len)
{
  while (len > 0)
    {
      int room = obstack_room (obstack);
      if (room == 0)
        {
          /* Not enough room. Add one character slowly,
             which may copy to a new chunk and make room.  */
          obstack_1grow (obstack, *ptr++);
          len--;
        }
      else
        {
          if (room > len)
            room = len;
          /* Add fast as much as we have room for. */
          len -= room;
          while (room-- > 0)
            obstack_1grow_fast (obstack, *ptr++);
        }
    }
}
```

obstack_blank

〖功能〗该函数是为增长对象添加字节的最基础函数，添加空间之后不进行初始化。

〖原型〗void obstack_blank (struct obstack *obstack-ptr, int size)

〖位置〗obstack.h (GNU)

obstack_copy0

〖功能〗该函数与 obstack_copy 类似，但是 obstack_copy0 将追加一个额外的字节，其中包含一个空字符。

〖原型〗void * obstack_copy0 (struct obstack *obstack-ptr, void *address, int size)

〖位置〗obstack.h (GNU)

〖说明〗额外追加的字节不计算在参数 size 的大小中。

在将字符序列作为以空字符结尾的字符串拷贝到对象堆栈中时，使用该函数非常便捷。以下是一个样例。

```
char *
obstack_savestring (char *addr, int size)
{
  return obstack_copy0 (&myobstack, addr, size);
}
```

obstack_copy

〖功能〗该函数通过拷贝从 address 开始的 size 个字节的数据来分配一个数据块，并进行初始化。

〖原型〗void * obstack_copy (struct obstack *obstack-ptr, void *address, int size)

〖位置〗obstack.h (GNU)

〖说明〗在遇到与 obstack_alloc 相同的情形时，该函数将返回一个空指针。

obstack_finish

〖功能〗对象增长完毕时，使用该函数关闭对象并返回其最终地址。

〖原型〗void * obstack_finish (struct obstack *obstack-ptr)

〖位置〗obstack.h (GNU)

〖说明〗一旦对象增长完毕，对象堆栈就可以用来完成普通的分配操作，或者增长另一个对象。

该函数在遇到 obstack_alloc 相同的情况时返回一个空指针。

增长并构建一个对象时，可能需要了解该对象最终的大小。此时无需跟踪对象的增长过程，因为可以在对象增长完毕之前使用 obstack_object_size 函数得到对象堆栈的长度。

obstack_free

〖功能〗如果 object 是一个空指针，则释放对象堆栈中分配的所有对象。

〖原型〗void obstack_free (struct obstack *obstack-ptr, void *object)

〖位置〗obstack.h (GNU)

〖说明〗如果 object 不是一个空指针，则必须是对象堆栈中分配的一个对象的地址。随后函数释放 object 及对象堆栈中从 object 之后分配的所有对象。

注意，如果 object 是一个空指针，则结果将得到一个未初始化的对象堆栈。要释放对象堆栈中的所有内容，且允许进行下一次分配，必须使用对象堆栈中分配的第 1 个对象的地址调用 obstack_free：

```
obstack_free (obstack_ptr, first_object_allocated_ptr);
```

对象堆栈中的对象被组合成组块。当某个组块中的所有对象都被释放时，obstack 将自动释放组块。随后，其他对象堆栈或非对象堆栈分配时可以重新使用组块的空间。

obstack_grow

〖功能〗需要添加一块初始化空间时使用该函数，该函数是增长对象的 obstack_copy 函数。

〖原型〗void obstack_grow (struct obstack *obstack-ptr, void *data, int size)

〖位置〗obstack.h (GNU)

〖说明〗该函数为增长对象添加 size 个字节的数据，从 data 中复制内容。

obstack_grow0

〖功能〗该函数是增长对象的 obstack_copy0 函数。

〖原型〗void obstack_grow0(struct obstack *obstack-ptr, void *data, int size)

〖位置〗obstack.h (GNU)

〖说明〗该函数添加从 data 中复制来的 size 个字节，随后添加一个额外的空字符。

obstack_init

〖功能〗初始化对象堆栈 obstack-ptr，以准备分配对象。

〖原型〗int obstack_init (struct obstack *obstack-ptr)

〖位置〗obstack.h (GNU)

〖说明〗该函数调用 obstack_chunk_alloc 函数。如果 obstack_chunk_alloc 函数返回一个空指针，则该函数返回 0，说明内存不足；否则返回 1。如果在内存不足时提供 obstack_chunk_alloc 函数调用 exit 或者 longjmp，则可以忽略 obstack_init 的返回值。

以下两个样例说明如何为对象堆栈分配空间，并进行初始化。首先声明 obstack 是一个静态变量。

```
static struct obstack myobstack;
...
obstack_init (&myobstack);
```

然后由对象堆栈自己完成动态分配。

```
struct obstack *myobstack_ptr
  = (struct obstack *) xmalloc (sizeof (struct obstack));
obstack_init (myobstack_ptr);
```

obstack_int_grow

〖功能〗该函数为对象堆栈 obstack-ptr 中的增长对象添加一个 int 类型的数值。

〖原型〗void obstack_int_grow_fast (struct obstack *obstack-ptr, int data)

〖位置〗obstack.h (GNU)

〖说明〗函数添加 sizeof 个 int 类型的字节，并且将其初始化为 data。

obstack_int_grow_fast

〖功能〗该函数为对象堆栈 obstack-ptr 中的增长对象添加 sizeof 个字节，其中包含 data

值。

〖原型〗void obstack_int_grow_fast (struct obstack *obstack-ptr, int data)

〖位置〗obstack.h (GNU)

obstack_lgrow

〖功能〗使用该函数为对象堆栈的增长对象添加字符，每次添加一个字节。添加的字节为字符 c。

〖原型〗void obstack_lgrow (struct obstack *obstack-ptr, char c)

〖位置〗obstack.h (GNU)

obstack_1grow_fast

〖功能〗该函数为对象堆栈 obstack-ptr 中的增长对象添加一个字节，添加的字节为字符 c。

〖原型〗void obstack_1grow_fast (struct obstack *obstack-ptr, char c)

〖位置〗obstack.h (GNU)

obstack_next_free

〖功能〗该函数返回 obstack-ptr 中当前组块的第 1 个空闲字节的地址。

〖原型〗void * obstack_next_free (struct obstack *obstack-ptr)

〖位置〗obstack.h (GNU)

〖说明〗如果没有增长对象，则该函数的返回值与 obstack_base 相同。

obstack_object_size

〖功能〗该函数返回当前增长对象的字节数。

〖原型〗int obstack_object_size (struct obstack *obstack-ptr)

〖位置〗obstack.h (GNU)

〖说明〗该函数等效于如下表达式。

```
obstack_next_free (obstack-ptr) - obstack_base (obstack-ptr)
```

obstack_ptr_grow

〖功能〗该函数为指针值加 1。

〖原型〗void obstack_ptr_grow(struct obstack *obstack-ptr, void *data)

〖位置〗obstack.h (GNU)

〖说明〗添加 sizeof 个字节，其中包含 data 值。

obstack_ptr_grow_fast

〖功能〗该函数为 obstack-ptr 中的增长对象添加 sizeof 个字节，其中包含 data 值。

〖原型〗void obstack_ptr_grow_fast (struct obstack *obstack-ptr, void *data)

〖位置〗obstack.h (GNU)

obstack_room

〖功能〗该函数返回可以使用快速增长函数为对象堆栈的当前增长对象安全添加的字节数。

〖原型〗int obstack_room (struct obstack *obstack-ptr)

〖位置〗obstack.h (GNU)

〖说明〗当确认有足够空间时，可以使用快速增长函数为增长对象添加数据。

r_alloc

〖功能〗该函数用于分配一块可重定位块，大小为 size 个字节。

〖原型〗void r_alloc (void **handleptr,size_t size)

〖位置〗malloc.h (GNU)

〖说明〗该函数在*handleptr 中保存块地址，并且返回一个空指针说明函数执行成功。如果函数得不到所需的空间，则在*handleptr 中保存一个空指针，并返回一个空指针。

r_alloc_free

〖功能〗该函数用于释放一块可重定位块。

〖原型〗void r_alloc_free (void **handleptr)

〖位置〗malloc.h (GNU)

〖说明〗释放*handleptr 指向的块，并在*handleptr 中保存一个空指针，说明它不再指向某个分配块。

r_re_alloc

〖功能〗该函数用于调整*handleptr 指向的存储块的大小，并且令其长度等于 size 个字节。

〖原型〗void * r_re_alloc (void **handleptr, size_t size)

〖位置〗malloc.h (GNU)

〖说明〗将调整大小之后的存储块地址保存在*handleptr 中，并返回一个非空指针说明执行成功。

如果没有足够的内存，则函数返回一个空指针，并且不改变*handleptr 的内容。

realloc

〖功能〗该函数将地址为 ptr 的存储块大小更改为 newsize。

〖原型〗void * realloc (void *ptr, size_t newsize)

〖位置〗malloc.h, stdlib.h (ISO)

〖说明〗因为存储块结尾后面的空间可能已经被使用，因此 realloc 可能必须将存储块拷贝到可用空间充足的新地址中。realloc 的值是存储块的新地址。如果需要移动存储块，则 realloc 将复制旧内容。

如果向 ptr 传递一个空指针，则 realloc 的动作类似 malloc (newsize)。这样可能很方便，但必须记住，旧版本(在 ISO C 之前)可能不支持这一动作，因此向 realloc 传递空指针可能导致系统崩溃。

该函数与 malloc 类似，如果没有足够的内存空间来扩大存储块，将返回一个空指针。此时不更改或重新定位原来的存储块。

在多数情况下，realloc 函数执行失败时不会改变原有存储块，因为内存不足时应用程序无法继续执行，惟一可做的动作就是发送致命错误消息。可以编写一个称为 xrealloc 的子程序，用来负责错误消息。

```
void *
xrealloc (void *ptr, size_t size)
{
  register void *value = realloc (ptr, size);
  if (value == 0)
    fatal ("Virtual memory exhausted");
  return value;
}
```

也可以根据自己的需要使用 realloc 函数缩小存储块的大小。在执行某些分配动作时，缩小存储块的大小必须进行拷贝，因此如果没有其他可用的空间，该动作将可能失败。

如果指定的新尺寸与原有尺寸相同，则 realloc 函数将不会改变任何内容，并返回原有地址。

valloc

〖功能〗该函数类似于 memalign 第 2 个参数为页面大小时的情况。

〖原型〗void * valloc (size_t size)

〖位置〗malloc.h, stdlib.h (BSD)

1.3　字 符 处 理

isalnum

〖功能〗如果 c 是字母或者数字，则返回真。

〖原型〗int isalnum (int c)

〖位置〗ctype.h (ISO)

〖说明〗如果某个字符的 isalpha 函数为真或者 isdigit 函数为真，则该字符的 isalnum 函数也为真。

isalpha

〖功能〗如果 c 是字母则返回真。

〖原型〗int isalpha (int c)

〖位置〗ctype.h (ISO)

〖说明〗如果某个字符的 islower 或者 isupper 函数返回真，则该字符的 isalpha 函数也返回真。

在某些场合下，可能存在一些例外的字符，其 isalpha 函数为真，而该字符既不是大写也不是小写。但是在标准 C 中不存在这样的例外字符。

isascii

〖功能〗如果 c 是符合 US/UK ASCII 字符集的 7 位无符号 char 值，则返回真。

〖原型〗int isascii (int c)

〖位置〗ctype.h (SVID, BSD)

〖说明〗该函数为 BSD 扩展，也是 SVID 扩展。

isblank

〖功能〗如果 c 为空白字符(即空格或者制表符)，则返回真。

〖原型〗int isblank (int c)

〖位置〗ctype.h(GNU)

〖说明〗该函数为 GNU 扩展。

iscntrl

〖功能〗如果 c 为控制字符(也就是非打印字符)，则返回真。

〖原型〗int iscntrl (int c)

〖位置〗ctype.h (ISO)

isdigit

〖功能〗如果 c 为十进制数字(从 0 到 9)，则返回真。

〖原型〗int isdigit (int c)

〖位置〗ctype.h (ISO)

isgraph

〖功能〗如果 c 为图形字符(即有相关轮廓的字符)，则返回真。

〖原型〗int isgraph (int c)

〖位置〗ctype.h (ISO)

〖说明〗空白字符不作为图形字符处理。

islower

〖功能〗如果 c 为小写字母，则返回真。

〖原型〗int islower (int c)

〖位置〗ctype.h (ISO)

isprint

〖功能〗如果 c 为可打印字符，则返回真。
〖原型〗int isprint (int c)
〖位置〗ctype.h (ISO)
〖说明〗可打印字符包括所有图形字符，以及空格(' ')字符。

ispunct

〖功能〗如果 c 为标点字符，则返回真。
〖原型〗int ispunct (int c)
〖位置〗ctype.h (ISO)
〖说明〗除了字母和文字字符或者空格字符之外的所有可打印字符，该函数都返回真。

isspace

〖功能〗如果 c 为空白字符，则返回真。
〖原型〗int isspace (int c)
〖位置〗ctype.h (ISO)
〖说明〗在标准 C 中，只有 c 为如下的标准空白字符时 isspace 返回真。

' '	空格
'\f'	进纸符
'\n'	换行符
'\r'	回车符
'\t'	水平制表符
'\v'	垂直制表符

isupper

〖功能〗如果 c 为大写字母，则返回真。
〖原型〗int isupper (int c)
〖位置〗ctype.h (ISO)

isxdigit

〖功能〗如果 c 为十六进制数字，则返回真。
〖原型〗int isxdigit (int c)
〖位置〗ctype.h (ISO)
〖说明〗十六进制数字包括从 0 到 9 的十进制数字和从 A 到 F 或从 a 到 f 的字母。

toascii

〖功能〗通过清除高位将 c 转换成符合 US/UK ASCII 字符规定的 7 位无符号字符。

〖原型〗int toascii (int c)

〖位置〗ctype.h (SVID, BSD)

〖说明〗该函数是 BSD 扩展，也是 SVID 扩展。

tolower

〖功能〗如果 c 是大写字母，tolower 返回相应的小写字母。如果 c 不是大写字母，则返回 c 而不改变。

〖原型〗int tolower (int c)

〖位置〗ctype.h (ISO)

_tolower

〖功能〗与 tolower 相同，提供与 SVID 的兼容性。

〖原型〗int _tolower (int c)

〖位置〗ctype.h (SVID)

toupper

〖功能〗如果 c 是小写字母，toupper 返回相应的大写字母。如果 c 不是小写字母，则返回 c 而不改变。

〖原型〗int toupper (int c)

〖位置〗ctype.h (ISO)

_toupper

〖功能〗与 toupper 相同，提供与 SVID 的兼容性。

〖原型〗int _toupper (int c)

〖位置〗ctype.h (SVID)

1.4 字符串和数组处理

bcmp

〖功能〗该函数是从 BSD 沿袭下来的函数，是 memcmp 的过时版本。

〖原型〗int bcmp (const void *a1, const void *a2, size_t size)

〖位置〗string.h (BSD)

bcopy

〖功能〗该函数是从 BSD 沿袭下来的函数，是 memmove 部分功能的过时版本。请注意它与 memmove 不完全等价，因为两者的参数顺序不同。

〖原型〗void * bcopy (void *from, const void *to, size_t size)

〖位置〗string.h (BSD)

bzero

〖功能〗该函数是从 BSD 沿袭下来的函数，是 memset 部分功能的过时版本。请注意它与 memset 不等价，因为它只能保存 0。

〖原型〗void * bzero (void *block, size_t size)

〖位置〗string.h (BSD)

index

〖功能〗该函数是 strchr 函数的另一个名称，两者的功能完全一致。

〖原型〗char * index (const char *string, int c)

〖位置〗string.h (BSD)

memccpy

〖功能〗该函数从 from 拷贝最多 size 个字节到 to，直到遇到与 c 匹配的字节时为止。

〖原型〗void * memccpy (void *to, const void *from, int c, size_t size)

〖位置〗string.h (SVID)

〖说明〗返回值为一个指针，指向 to 中拷贝 c 之后的一个字节。如果 from 的前 size 个字节中没有发现与 c 匹配的字节，则返回一个空指针。

memchr

〖功能〗该函数在从 block 开始的前 size 个字节中查找第 1 次出现的 c(转换为 unsigned char 类型)。

〖原型〗void * memchr (const void *block, int c, size_t size)

〖位置〗string.h (ISO)

〖说明〗返回一个指针，指向找到的字节，如果没有匹配字符，则返回一个空指针。

memcmp

〖功能〗该函数比较 a1 的前 size 个字节和 a2 的前 size 个字节的内容。

〖原型〗int memcmp (const void *a1, const void *a2, size_t size)

〖位置〗string.h (ISO)

〖说明〗返回值的符号与找到的第 1 对不同字节(解释为 unsigned char 对象，然后转换为 int)之间的差异相同。

如果两个块的内容相同，则 memcmp 返回 0。

对于任意长度的数组而言，使用 memcmp 函数检查相等性非常有效。但是对除了字节之外的其他对象组成的数组进行字节方式的顺序比较通常没有意义。例如，对组成浮点数的字节进行字节方式的比较则无法说明两个浮点数值之间的关系。

使用 memcmp 比较可能包含“洞”的对象需要更加小心，例如为了强制对齐而在结构对象中插入的填充内容，合并结尾的剩余空间，以及长度小于分配大小的字符串结尾的剩余字符。这些“洞”中的内容不确定，执行字节方式的比较时可能导致奇怪的动作发生。

为了得到可预测的结果，建议执行明确的成分方式比较。

例如，定义一个结构类型如下。

```
struct foo
  {
    unsigned char tag;
    union
      {
        double f;
        long i;
        char *p;
      } value;
  };
```

最好自己编写一个专用的比较函数来比较 struct foo 对象，而不要使用 memcmp 函数进行比较。

memcpy

〖功能〗该函数从 from 开始的对象中拷贝 size 个字节到从 to 开始的对象中。

〖原型〗void * memcpy (void *to, const void *from, size_t size)

〖位置〗string.h (ISO)

〖说明〗如果两个数组 to 和 from 相互重叠，则该函数的行为无法确定，如果可能出现重叠则应当使用 memmove 函数。

memcpy 函数返回 to 的值。

以下样例说明如何使用 memcpy 拷贝一个数组的内容。

```
struct foo *oldarray, *newarray;
int arraysize;
...
memcpy (new, old, arraysize * sizeof (struct foo));
```

memmem

〖功能〗该函数与 strstr 类似，但是 needle 和 haystack 是字节数组而不是以空字符结尾的字符串，needle-len 表示 needle 的长度，而 haystack-len 表示 haystack 的长度。

〖原型〗void * memmem (const void *haystack, size_t haystack-len,const void *needle, size_t needle-len)

〖位置〗string.h (GNU)

〖说明〗该函数为 GNU 的扩展。

memmove

〖功能〗该函数将 from 的 size 个字节拷贝到 to 的 size 个字节中，即使两个存储块空间相互重叠也没有影响。

〖原型〗void * memmove (void *to, const void *from, size_t size)

〖位置〗string.h (ISO)

〖说明〗如果出现重叠，memmove 将拷贝 from 存储块中的源值，其中包括同时属于 to 存储块的那些字节。

memset

〖功能〗该函数将 c(转换为 unsigned char)拷贝到从 block 开始的对象的前 size 个字节中。

〖原型〗void * memset (void *block, int c, size_t size)

〖位置〗string.h (ISO)

〖说明〗返回 block 值。

rindex

〖功能〗该函数是 strrchr 函数的另一个名称，两者的功能完全相同。

〖原型〗char * rindex (const char *string, int c)

〖位置〗string.h (BSD)

stpcpy

〖功能〗从字符串 from(直到并包括终止字符)中拷贝字符到字符串 to。正如 memcpy 一样，如果字符串重叠，则将会产生不确定的结果。该函数的返回值为 to 的值。

〖原型〗char * stpcpy (char *to, const char *from)

〖位置〗string.h

stpncpy

〖功能〗与函数 stpcpy 类似，但只向 to 中拷贝 size 个字符。

〖原型〗char * stpncpy (char *to, const char *from, size_t size)

〖位置〗string.h (GNU)

〖说明〗如果 from 的长度大于 size，stpncpy 只拷贝开始 size 个字符，并返回指向紧接在最后一个拷贝的字符之后的字符型指针。

如果 from 的长度小于 size，stpncpy 拷贝整个 from，并且在其后缀以足够的空字符以填满 size 个字符。这种动作没有什么用处，但这是 ISO C 的标准要求。

如果字符串重叠，则 strncpy 的结果不确定。

strcasecmp

〖功能〗该函数与 strcmp 类似，但是忽略大小写差异。

〖原型〗int strcasecmp (const char *s1, const char *s2)

〖位置〗string.h (BSD)

strcat

〖功能〗该函数与 strcpy 类似，但是来自于 from 的字符将连接或附加到 to 的结尾处，

而不是覆盖 to，即 from 的首字符覆盖 to 的结尾标志空字符。

〖原型〗char * strcat (char *to, const char *from)

〖位置〗string.h(ISO)

〖说明〗与 strcat 等效的定义如下。

```
char *
strcat (char *to, const char *from)
{
  strcpy (to + strlen (to), from);
  return to;
}
```

如果字符串重叠，则函数的结果不确定。

strchr

〖功能〗该函数在从指针 string 开始的以 0 字符结束的字符串中寻找第 1 次出现的字符 c。

〖原型〗char * strchr (const char *string, int c)

〖位置〗string.h (ISO)

〖说明〗返回值是指向所找到的字符的指针，找不到时返回空指针。

例如：

```
strchr ("hello, world", 'l')
   => "llo, world"
strchr ("hello, world", '?')
   => NULL
```

标志结束的空指针是字符串的一部分，所以可以通过将 c 定义为空指针来得到指向字符串结尾的指针。

strcmp

〖功能〗该函数比较字符串 s1 和 s2。

〖原型〗int strcmp (const char *s1, const char *s2)

〖位置〗string.h (ISO)

〖说明〗如果两个字符串相同，则返回 0。如果 s1 是 s2 起始部分的子串，则认为 s1“小于”s2。

strcoll

〖功能〗该函数与 strcmp 相似，但该函数是通过比较 LC_COLLATE 的顺序来实现。

〖原型〗int strcoll (const char *s1, const char *s2)

〖位置〗string.h (ISO)

〖说明〗以下是一个使用 strcoll 为字符串数组排序的样例。实际的排序算法 qsort 没有在这里给出，这里代码的任务是说明在对字符串排序时如何进行比较。

```
/* This is the comparison function used with qsort. */
int
compare_elements (char **p1, char **p2)
{
  return strcoll (*p1, *p2);
}
/* This is the entry point---the function to sort
   strings using the locale's collating sequence. */
void
sort_strings (char **array, int nstrings)
{
  /* Sort temp_array by comparing the strings. */
  qsort (array, sizeof (char *),
         nstrings, compare_elements);
}
```

strcpy

〖功能〗从 from 中向 to 中拷贝字符(直到并包括终止空字符)。

〖原型〗char * strcpy (char *to, const char *from)

〖位置〗string.h (ISO)

〖说明〗与 memcpy 一样，如果串重叠，则没有确定的结果。该函数的返回值是 to。

strcspn

〖功能〗返回 string 中第 1 个 stopset 中的字符的偏移量。

〖原型〗size_t strcspn (const char *string, const char *stopset)

〖位置〗string.h (ISO)

strdup

〖功能〗该函数将以空字符结束的字符串 s 拷贝到一个新分配的字符串中。

〖原型〗char * strdup (const char *s)

〖位置〗string.h (GNU)

〖说明〗字符串使用 malloc 函数分配。如果 malloc 不能为新的字符串分配空间，则 strdup 函数返回一个空指针；否则该函数将返回指向新的字符串的指针。

strdupa

〖功能〗该函数与 strdup 类似，但是使用 alloca 函数代替 malloc 函数分配新的字符串。因此返回字符串存在与使用 alloca 函数分配的所有内存块相同的限制。

〖原型〗char * strdupa (const char *s)

〖位置〗string.h (GNU)

〖说明〗strdupa 只能作为一个宏来执行，原因很明显，用户无法得到该函数的地址。除此之外，该函数是一个非常有用的函数。例如，在如下代码中，使用 malloc 函数的开销过高。

```
#include <paths.h>
#include <string.h>
#include <stdio.h>
const char path[] = _PATH_STDPATH;
int
main (void)
{
  char *wr_path = strdupa (path);
  char *cp = strtok (wr_path, ":");
  while (cp != NULL)
    {
      puts (cp);
      cp = strtok (NULL, ":");
    }
  return 0;
}
```

注意，直接使用 path 调用 strtok 函数是非法的。

只有使用 GNU C 函数库时，才能使用该函数。

strlen

〖功能〗返回由空字符结束的字符串 s 的长度(即返回数组中结束空字符的偏移量)。

〖原型〗size_t strlen (const char *s)

〖位置〗string.h (ISO)

〖说明〗例如：

```
strlen ("hello, world")
 =>12
```

应用到字符数组上，strlen 函数返回保存的字符串的长度，而不是其分配大小。可以使用 sizeof 操作符得到保存字符串的字符数组的分配长度：

```
char string[32] = "hello, world";
sizeof (string)
   => 32
strlen (string)
   => 12
```

strncasecmp

〖功能〗该函数与 strncmp 类似，只是该函数将忽略大小写差异。

〖原型〗int strncasecmp (const char *s1, const char *s2, size_t n)

〖位置〗string.h (BSD)

〖说明〗strncasecmp 是 GNU 扩展。

strncat

〖功能〗将 from 中不超过 size 个的字符附加到 to 结尾处其余与 strcat 一样。一个单独

的空字符也常附加到 to，因此分配给 to 的长度至少比其原长度多 size+1 个字节。

〖原型〗char * strncat (char *to, const char *from, size_t size)

〖位置〗string.h (ISO)

〖说明〗如果两个串重叠，strncat 的影响不确定。

strncmp

〖功能〗该函数与 strcmp 类似，但是最多只比较 size 个字符。即如果两个字符串的前 size 个字符相同，则返回值为 0。

〖原型〗int strncmp (const char *s1, const char *s2, size_t size)

〖位置〗string.h (ISO)

strncpy

〖功能〗该函数与 strcpy 类似，但是只拷贝 size 个字符到 to 块中。

〖原型〗char * strncpy (char *to, const char *from, size_t size)

〖位置〗string.h (ISO)

〖说明〗如果 from 的长度大于 size，则只拷贝开始的 size 个字符。注意，此时没有将终止符写入 to 块。如果 from 的长度小于 size，则拷贝 from 的全部内容，并且在后面添加足够的空字符，直到总长度达到 size 个字符为止。此功能非常有用，但仅在 ISO C 标准中提供。

如果两个串重叠，则 strncpy 的结果不确定。

相对于 strcpy 函数而言，strncpy 函数可以避免写入的内容超过分配给 to 块的存储空间的末尾。然而，它也可导致程序在一个普通的情形下运行缓慢，例如，将一个可能很小的字符串拷贝到较大的缓存时。此时，由于 size 可能太大，strncpy 将浪费相当多的时间拷贝空字符。

strndup

〖功能〗该函数与 strdup 函数类似，但是通常仅向新分配的字符串中拷贝至多 size 个字符。

〖原型〗char * strndup (const char *s, size_t size)

〖位置〗string.h (GNU)

〖说明〗如果 s 的长度大于 size，则 strndup 函数仅拷贝前 size 个字符并添加空结束符。否则将拷贝所有字符且终止字符串。与 strncpy 函数不同的是，该函数总是终止目的字符串。

strndupa

〖功能〗该函数与 strndup 类似，但正如 strdupa 函数一样该函数利用 alloca 分配新的串。

〖原型〗char * strndupa (const char *s, size_t size)

〖位置〗string.h (GNU)

〖说明〗strdupa 的优势和局限性对 strndupa 也有效。

该函数仅作为宏来执行，即无法得到它的地址。strndupa 仅在使用 GUN C 时有效。

strpbrk

〖功能〗该函数和 strcspn 有关，但是返回一个指针，指向 string 中第 1 个 stopset 成员的字符，而不是第 1 个子串的长度。

〖原型〗char * strpbrk (const char *string, const char *stopset)

〖位置〗string.h (ISO)

〖说明〗如果未发现 stopset 中的成员，则返回空指针。

例如：

```
strpbrk ("hello, world", " \t\n,.;!?")
   => ", world"
```

strrchr

〖功能〗该函数和 strchr 类似，但它是从串 string 的尾部反向搜索(而不是从前面开始正向搜索)。

〖原型〗char * strrchr (const char *string, int c)

〖位置〗string.h (ISO)

〖说明〗例如：

```
strrchr ("hello, world", 'l')
   => "ld"
```

strsep

〖功能〗该函数与 strtok 类似，但是可重入。

〖原型〗char * strsep (char **string_ptr, const char *delimiter)

〖位置〗string.h (BSD)

〖说明〗第 2 个可重入的方法是避免使用多余的第 1 个参数，移动指针的初始化必须由用户来完成。后续的 strsep 调用将沿着 delimiter 分隔标记移动指针，返回下一个标记的地址，并将 string_ptr 更改为指向下一个标记的开始。

以下是使用 strsep 的样例。

```
#include <string.h>
#include <stddef.h>
...
char string[] = "words separated by spaces -- and, punctuation!";
const char delimiters[] = " .,;:!-";
char *running;
char *token;
...
running = string;
token = strsep (&running, delimiters);    /* token => "words" */
token = strsep (&running, delimiters);    /* token => "separated" */
```

```
token = strsep (&running, delimiters);    /* token => "by" */
token = strsep (&running, delimiters);    /* token => "spaces" */
token = strsep (&running, delimiters);    /* token => "and" */
token = strsep (&running, delimiters);    /* token => "punctuation" */
token = strsep (&running, delimiters);    /* token => NULL */
```

strspn

〖功能〗返回 string 中第 1 个完全由 skipset 定义的字符组中的成员所组成的子串的长度。注意，skipset 中字符的顺序并不重要。

〖原型〗size_t strspn (const char *string, const char *skipset)

〖位置〗string.h (ISO)

strstr

〖功能〗与 strchr 函数一样，但该函数在 haystack 中搜索一个子串 needle 而不是单个字符。

〖原型〗char * strstr (const char *haystack, const char *needle)

〖位置〗string.h (ISO)

〖说明〗返回指向 haystack 中子串的第 1 个字符的指针，或者当未发现匹配时返回空指针。如果 needle 是空串，则该函数返回 haystack。

strtok

〖功能〗连续调用该函数可以将一个字符串分割为多个标记。

〖原型〗char * strtok (char *newstring, const char *delimiters)

〖位置〗string.h (ISO)

〖说明〗第 1 次调用时需要分割的字符串作为 newstring 参数传递，strtok 函数用来建立一些内部状态信息。随后的调用可以为 newstring 参数传递一个空指针。使用其他的非空 newstring 参数调用 strtok 将重新初始化状态信息，这样可以保证不会有其他库函数在后台调用 strtok(从而破坏内部状态信息)。

参数 delimiters 是一个字符串，指定包含将被分解标记的分隔符集合。属于这一集合成员的初始字符都将被丢弃。第 1 个不属于该集合的字符标志着下一个标记的开始。找到下一个分割符集合的成员时，就找到了标记的结尾。原字符串 newstring 中的该字符被空字符覆盖，并且返回指向 newstring 中标记开始位置的指针。

下一次调用 strtok 时，搜索将从上一个标记末尾字符的下一个开始。注意，每次调用 strtok 时，无需保持分割符集合 delimiters 不变。

如果到达字符串 newstrin 的末尾，或者字符串的剩余部分全部由分割字符组成，则 strtok 返回一个空指针。

以下是使用 strtok 的一个简单样例。

```
#include <string.h>
#include <stddef.h>
```

```
...
char string[] = "words separated by spaces -- and, punctuation!";
const char delimiters[] = " .,;:!-";
char *token;
...
token = strtok (string, delimiters);  /* token => "words" */
token = strtok (NULL, delimiters);    /* token => "separated" */
token = strtok (NULL, delimiters);    /* token => "by" */
token = strtok (NULL, delimiters);    /* token => "spaces" */
token = strtok (NULL, delimiters);    /* token => "and" */
token = strtok (NULL, delimiters);    /* token => "punctuation" */
token = strtok (NULL, delimiters);    /* token => NULL */
```

strtok_r

〖功能〗该函数与 strtok 一样，将字符串分割为多个标记。

〖原型〗char * strtok_r (char *newstring, const char *delimiters, char **save_ptr)

〖位置〗string.h (POSIX)

〖说明〗调用该函数时下一个标记的有关信息没有建立在内部状态信息中，调用方提供了另外一个参数 save_ptr，该函数是一个指针，指向另外一个字符串指针。使用等于空指针的 newstring 调用 strtok_r，并且多次调用时保持 save_ptr 不变即可保证任务可重入。

strxfrm

〖功能〗该函数使用选择的当前场所所确定的比较转换来实现字符串转换，并将转换后的字符串保存在数组 to 中，保存字符数不超过 size 个(包括结束空字符)。

〖原型〗size_t strxfrm (char *to, const char *from, size_t size)

〖位置〗string.h (ISO)

〖说明〗如果字符串 to 和 from 相互重叠，那么函数的动作将不能确定。

返回值是整个被转换字符串的长度，它不受 size 值的影响，但是如果该值大于或者等于 size，则意味着转换后的字符串没有完全存入数组 to。这时仅仅保存 size 个字节的内容。要得到整个字符串，需要使用较大的输出数组再次调用 strxfrm。

转换后字符串可能比初始字符串长，也可能比初始字符串短。

如果 size 为 0，则 to 中没有保存任何字符。这时，strxfrm 仅仅返回转换后字符串的字符长。可以用该值来确定分配的字符串长度。如果 size 等于 0，那么与 to 的内容无关，to 甚至可以是一个空指针。

以下样例说明在准备执行多次比较时，如何使用 strxfrm。该样例完成的工作与上一个样例类似，但是速度更快，因为无论需要与其他字符串比较多少次，每个字符串都只需转换一次。

```
struct sorter { char *input; char *transformed; };
/* This is the comparison function used with qsort
   to sort an array of struct sorter. */
int
```

```
compare_elements (struct sorter *p1, struct sorter *p2)
{
  return strcmp (p1->transformed, p2->transformed);
}
/* This is the entry point---the function to sort
   strings using the locale's collating sequence. */
void
sort_strings_fast (char **array, int nstrings)
{
  struct sorter temp_array[nstrings];
  int i;
  /* Set up temp_array.  Each element contains
     one input string and its transformed string. */
  for (i = 0; i < nstrings; i++)
    {
      size_t length = strlen (array[i]) * 2;
      char *transformed;
      size_t transformed_lenght;
      temp_array[i].input = array[i];
      /* First try a buffer perhaps big enough.  */
      transformed = (char *) xmalloc (length);
      /* Transform array[i].  */
      transformed_length = strxfrm (transformed, array[i], length);
      /* If the buffer was not large enough, resize it
         and try again.  */
      if (transformed_length >= length)
        {
          /* Allocate the needed space. +1 for terminating
             NUL character.  */
          transformed = (char *) xrealloc (transformed,
                                           transformed_length + 1);
          /* The return value is not interesting because we know
             how long the transformed string is.  */
          (void) strxfrm (transformed, array[i], transformed_length + 1);
        }
      temp_array[i].transformed = transformed;
    }
  /* Sort temp_array by comparing transformed strings. */
  qsort (temp_array, sizeof (struct sorter),
         nstrings, compare_elements);
  /* Put the elements back in the permanent array
     in their sorted order. */
  for (i = 0; i < nstrings; i++)
    array[i] = temp_array[i].input;
  /* Free the strings we allocated. */
  for (i = 0; i < nstrings; i++)
    free (temp_array[i].transformed);
}
```

1.5 流输入/输出

asprintf

〖功能〗该函数产生格式化输出内容，并将结果置于动态分配内存中。

〖原型〗int asprintf (char **ptr, const char *template, ...)

〖位置〗stdio.h (GNU)

〖说明〗该函数动态地分配内存(正如 malloc 一样)来保存输出，而不是将输出存入事先分配的缓冲器中，除此之外该函数类似于 sprintf。ptr 参数应该是一个 char * 对象的地址，同时 asprintf 存储在其位置被重新分配的字符串的指针。

下面是如何使用 asprintf 得到和 snprintf 相同结果的例子，但是该例子实现起来更容易。

```
/* Construct a message describing the value of a variable
   whose name is name and whose value is value. */
char *make_message (char *name, char *value)
{
  char *result;
  asprintf (&result, "value of %s is %s", name, value);
  return result;
}
```

clearerr

〖功能〗清除 stream 流中的文件结尾和错误指示符。

〖原型〗void clearerr (FILE *stream)

〖位置〗stdio.h (ISO)

〖说明〗文件定位函数也将清除流的文件结尾指示符。

fclose

〖功能〗关闭流 stream，并中断与相应文件之间的连接。

〖原型〗int fclose (FILE *stream)

〖位置〗stdio.h (ISO)

〖说明〗输出所有的缓存数据，丢弃所有的输入缓存。如果成功关闭所有文件，则 fclose 函数返回 0，如果出现错误则返回 EOF。

调用 fclose 关闭一个输出流时，检查错误非常重要。这是因为此时可能出现实时常见错误。例如当 fclose 输出缓存数据时，可能由于磁盘空间不足出现错误。即便缓存已经清空，当使用 NFS 关闭文件时仍然可能出现错误。

fcloseall

〖功能〗关闭进程打开的所有流，并中断与相应文件之间的连接。

〖原型〗int fcloseall (void)

〖位置〗stdio.h (GNU)

〖说明〗输出所有的缓存数据，丢弃所有的输入缓存。如果成功关闭所有文件，则 fcloseall 函数返回 0，如果出现错误则返回 EOF。

该函数只能在某些特殊情况下使用，例如出现某个错误并且程序必须中断时。通常每个流应当单独关闭，以便于识别各个流中存在的问题。此外，由于同时关闭了标准流，也容易出现问题。

当程序中的 main 函数返回，或者调用 exit 函数时，将自动关闭所有打开的流。如果程序以其他方式中断，例如调用 abort 函数或者出现某个致命信号，则打开的流可能无法正常关闭。此时，缓存输出可能无法清空，文件可能不完整。

feof

〖功能〗只有设置 stream 流的文件结尾标志时，该函数才返回非 0 值。

〖原型〗int feof (FILE *stream)

〖位置〗stdio.h (ISO)

ferror

〖功能〗只有设置 stream 流的错误标志时，该函数才返回非 0 值，表明在上一次流操作中出现了错误。

〖原型〗int ferror (FILE *stream)

〖位置〗stdio.h (ISO)

〖说明〗除了设置与流相关的错误标志之外，流操作函数也可以使用与相应的低级函数中对文件描述符操作的同样方式来设置 errno。例如，执行流输出的所有函数如 fputc、printf 和 fflush 等，都是按照 write 的标准执行的，因此 write 定义的所有 errno 错误代码在这些函数中都有意义。

fflush

〖功能〗将 stream 中的所有缓存输出传递到文件中。

〖原型〗int fflush (FILE *stream)

〖位置〗stdio.h (ISO)

〖说明〗如果 stream 是一个空指针，那么 fflush 将清空所有打开的输出流中的缓存输出数据。

如果出现写错误，则函数返回 EOF；否则返回 0。

fgetc

〖功能〗从 stream 流中按照不带符号的格式读取下一个字符，并返回该字符转换为 int 类型之后的数值。

〖原型〗int fgetc (FILE *stream)

〖位置〗stdio.h (ISO)

〖说明〗如果遇到文件结尾或者出现读错误，则返回 EOF。

fgetpos

〖功能〗保存 fpos_tstream 流中由 position 指向对象的文件位置指示符。

〖原型〗int fgetpos (FILE *stream, fpos_t *position)

〖位置〗stdio.h (ISO)

〖说明〗成功则返回 0，否则返回一个非 0 值，并且在 errno 中保存一个运行时定义的正数数值。

fgets

〖功能〗在 stream 中从上至下读入字符直至遇到换行符，将读入内容保存在字符串 s 中，并且在字符串末尾添加一个空字符。

〖原型〗char * fgets (char *s, int count, FILE *stream)

〖位置〗stdio.h (ISO)

〖说明〗字符串 s 中必须有 count 个字符的空间，但是最多只能读入 count-1 个字符。剩余的字符空间用来保存字符串末尾的空字符。

如果调用 fgets 时系统已经位于文件结尾，那么数组 s 的内容不会改变，返回一个空指针。如果出现读入错误，则同样返回一个空指针。否则返回值为指针 s。

警告：如果输入数据中包含有空字符，则系统不会通知程序员。因此除非确定数据中没有空字符，否则不要使用 fgets。不要使用该函数读入由用户编辑的文件，如果用户插入一个空字符，程序应当正确处理或者打印出明确的错误信息。这种情况下建议使用 getline 代替 fgets。

fmemopen

〖功能〗该函数打开一个流，允许由 opentype 参数指定的访问类型对由参数 buf 指定的缓存区进行读写。数组的最小长度为 size 个字节。

〖原型〗FILE * fmemopen (void *buf, size_t size, const char *opentype)

〖位置〗stdio.h (GNU)

〖说明〗如果 buf 参数指定一个空指针，则 fmemopen 将动态分配一个 size 字节长的数组(使用 malloc 函数)。实际上在需要将数据写入缓存区并随后立即读出的情况下，只能使用该函数，因为没有什么其他办法可以真正得到一个指向缓存区的指针(可使用 open_memstream 测试一下)。打开流之后缓存区就被释放。

opentype 参数与 fopen 中的参数相同。如果 opentype 指定追加模式，那么初始文件位置将设置为缓存区中的第 1 个空字符。否则初始文件位置将位于缓存区的起始位置。

当某个等待写入的流被清空或者关闭时，将在大小许可的情况下在缓存区末尾写入一个空字符(字节 0)。考虑到这一点，应当为 size 参数增加一个额外的字节。如果在缓存区中写入的字节数超过 size，则将出现错误。

对于等待读出的流而言，缓存区中的空字符将不作为文件结尾符。只有当文件位置超过 size 个字节时，读操作才会指出文件结束。因此，如果需要从某个以空字符结尾的字符

串中读出字符，则应当将字符串长度作为 size 参数值。

下面是一个使用 fmemopen 创建流，从字符串中读出数据的样例。

```
#include <stdio.h>
static char buffer[] = "foobar";
int
main (void)
{
  int ch;
  FILE *stream;
  stream = fmemopen (buffer, strlen (buffer), "r");
  while ((ch = fgetc (stream)) != EOF)
    printf ("Got %c\n", ch);
  fclose (stream);
  return 0;
}
```

该程序产生的输出如下。

```
Got f
Got o
Got o
Got b
Got a
Got r
```

fopen

〖功能〗该函数打开一个对文件 filename 执行 I/O 操作的流，并返回指向该流的指针。

〖原型〗FILE * fopen (const char *filename, const char *opentype)

〖位置〗stdio.h (ISO)

〖说明〗opentype 参数是一个字符串，用来控制如何打开文件并说明结果流的属性。该参数必须以如下字符开始。

r 打开一个已存在的文件进行只读操作。

w 打开一个文件进行只写操作。如果文件已经存在，则截短为 0 长度；否则创建一个新文件。

a 打开一个文件执行追加访问，也就是只允许在文件末尾写入。如果文件已经存在，则不允许更改原有内容，流的输出内容将追加到文件结尾；否则创建一个新的空文件。

r+ 打开一个已存存在的文件进行读写操作。不更改文件的原有内容，初始文件位置位于文件起始位置。

w+ 打开一个文件执行读写操作。如果文件已经存在，则截短为 0 长度；否则创建一个新文件。

a+ 打开或者创建一个文件，执行读操作和追加操作。如果文件存在，则不更改原有内容；否则创建一个新文件。读操作的初始文件位置位于文件起始位置，但是输出通常追加在文件末尾。

正如读者所见，+要求流同时进行输入和输出操作。ISO 标准要求使用这样的流在读操作和写操作之间切换时，必须调用 fflush 或者类似 fseek 的文件定位函数；否则内部缓存可能无法彻底清空。GNU C 函数库没有这一条限制，可以按照任意顺序执行读操作和写操作。

这些参数之后可能还有额外的字符来说明调用的标志位。通常首先说明模式(r，w+等)，可以保证所有的系统都兼容这部分参数。

GNU C 函数库为 opentype 的使用定义了一个额外字符，字符 x 要求创建一个新文件——如果文件 filename 已经存在，则 fopen 不会打开该文件，而会出现错误。只有确定不会与已存在的文件冲突时才能使用 x。这一点与 open 函数的 O_EXCL 选项一致。

opentype 中字符 b 的含义很标准，请求一个二进制流，而不是一个文本流。但是这与 POSIX 系统(包括 GNU 系统)没有区别。如果同时指定+和 b 参数，两者可以按照任意顺序出现。

opentype 中出现的其他字符都将忽略不计，但是在其他系统中可能会有意义。

如果打开失败，fopen 将返回一个空指针。

可以有多个流(或者文件描述符)指向同时打开的同一个文件。如果只是输入流则非常简单，但是如果还有其他输出流，则必须小心谨慎。无论多个流位于一个程序(这种情况不常见)或者位于多个程序中(这种情况经常发生)，都必须小心。使用文件锁定机制来避免对文件的同时访问是非常有效的。

fopencookie

〖功能〗该函数使用 io-functions 参数中的函数创建一个与 cookie 通信的流。

〖原型〗FILE * fopencookie (void *cookie, const char *opentype, cookie_io_functions_t io-functions)

〖位置〗stdio.h (GNU)

〖说明〗opentype 参数的解释与 fopen 相同(但是这里忽略了 truncate on open 选项)。

fopencookie 函数返回新创建的流，如果出现错误则，返回一个空指针。

fprintf

〖功能〗该函数与 printf 类似，但是输出写入 stream 流，而不是 stdout 流。

〖原型〗int fprintf (FILE *stream, const char *template, ...)

〖位置〗stdio.h (ISO)

fputc

〖功能〗将字符 c 转换为不带符号的 char 类型，并写入 stream 流。

〖原型〗int fputc (int c, FILE *stream)

〖位置〗stdio.h (ISO)

〖说明〗如果出现写错误，则返回 EOF；否则返回字符 c。

fputs

〖功能〗将字符串 s 写入 stream 流。

〖原型〗int fputs (const char *s, FILE *stream)

〖位置〗stdio.h (ISO)

〖说明〗不写终止空字符。该函数也不会添加换行字符，仅仅输出字符串中的字符。

如果出现写错误，则该函数返回 EOF；否则返回一个非负数值。

例如：

```
fputs ("Are ", stdout);
fputs ("you ", stdout);
fputs ("hungry?\n", stdout);
```

以上语句输出文本 Are you hungry 及一个换行符。

fread

〖功能〗从 stream 流中将最多 count 个 size 大小的对象读入数组 data。

〖原型〗size_t fread (void *data, size_t size, size_t count, FILE *stream)

〖位置〗stdio.h (ISO)

〖说明〗返回实际读入的对象数目，如果遇到文件结尾或者出现读错误，该数目可能小于 count。如果 size 或者 count 等于 0，则函数返回数值 0(并且不会读入任何对象)。

如果 fread 在某个对象中遇到文件结尾，则返回读入完整对象的数目，并抛弃不完整的对象。因此流停留在实际的文件结尾位置。

freopen

〖功能〗该函数类似 fclose 和 fopen 函数的结合。

〖原型〗FILE * freopen (const char *filename, const char *opentype, FILE *stream)

〖位置〗stdio.h (ISO)

〖说明〗该函数首先关闭 stream 指向的流，忽略进程中检测到的任何错误(由于忽略错误，因此如果使用流执行了任何输出操作，则名为 filename 的文件已经打开)。然后按照 fopen 的 opentype 模式打开名为 filename 的文件，并且与同一个流对象 stream 相关联。

如果操作失败，则返回一个空指针；否则 freopen 返回 stream。

freopen 原本用来连接标准流和自己选择的文件。在标准流固定为某种用途使用的程序中该函数非常有用。在 GNU C 函数库中，可以关闭标准流，然后使用 fopen 打开新的标准流。但是其他系统没有这种能力，因此使用 freopen 的可移植性较强。

fscanf

〖功能〗该函数类似 scanf，但是输入从 stream 流读入，而不是从 stdin 读入。

〖原型〗int fscanf (FILE *stream, const char *template, ...)

〖位置〗stdio.h (ISO)

fseek

〖功能〗用来改变 stream 流中的文件位置。

〖原型〗int fseek (FILE *stream, long int offset, int whence)

〖位置〗stdio.h (ISO)

〖说明〗whence 的数值必须是 SEEK_SET，SEEK_CUR 及 SEEK_END 常量之一，用来表示 offset 是否与文件开始、当前文件位置或者文件结尾相关。

如果操作成功则函数返回数值 0，返回非 0 值表示操作失败。成功的调用还会清除 stream 的文件结尾标志，并且抛弃使用 ungetc 返回的字符。

fseek 将在设置文件位置之前清空输出缓存，或者保存缓存内容以便于随后在文件的恰当位置输出。

可移植性提示：在非 POSIX 系统中，ftell 和 fseek 可能只在二进制流中稳定工作。

fsetpos

〖功能〗将流 stream 的文件位置指示符设置为 position 位置，该位置必须通过之前调用 fgetpos 才能得到。

〖原型〗int fsetpos (FILE *stream, const fpos_t position)

〖位置〗stdio.h (ISO)

〖说明〗如果成功则清除流中的文件结尾指示符，清除所有使用 ungetc 返回的字符，并且返回数值 0。否则函数返回一个非 0 值，并且在 errno 中保存一个运行时指定的整数数值。

ftell

〖功能〗返回 stream 流的当前文件位置。

〖原型〗long int ftell (FILE *stream)

〖位置〗stdio.h (ISO)

〖说明〗如果流不支持文件定位，或者文件位置无法使用长整数表示，或者出现其他可能的原因，该函数将可能执行失败。如果失败，则返回数值-1。

fwrite

〖功能〗将数组 data 中最多 count 个 size 大小的对象写入 stream 流。

〖原型〗size_t fwrite (const void *data, size_t size, size_t count, FILE *stream)

〖位置〗stdio.h (ISO)

〖说明〗如果调用成功，返回值通常为 count。任何其他返回值都表示出现某一类错误，例如空间不足。

getc

〖功能〗与 fgetc 类似，但是该函数允许作为宏执行，即允许多次为 stream 参数赋值。

〖原型〗int getc (FILE *stream)

〖位置〗stdio.h (ISO)

〖说明〗该函数的自由化程度很高，因此通常是作为读入单个字符使用的最佳函数。

getchar

〖功能〗与 stream 等于 stdin 时的 getc 函数等价。

〖原型〗int getchar (void)

〖位置〗stdio.h (ISO)

getdelim

〖功能〗与 getline 类似，但是用来说明终止读入的字符不一定是换行符。

〖原型〗ssize_t getdelim (char **lineptr, size_t *n, int delimiter, FILE *stream)

〖位置〗stdio.h (GNU)

〖说明〗参数 delimiter 说明分隔字符，getdelim 函数将在遇到该字符(或者遇到文件结尾)时停止读入。

文本保存在 lineptr 中，包括分隔字符和一个终止空字符。与 getline 类似，如果 getdelim 函数发现 linptr 长度不够，可以加长 lineptr。

getline 实际上是按照 getdelim 的规则执行的，如下所示。

```
ssize_t
getline (char **lineptr, size_t *n, FILE *stream)
{
  return getdelim (lineptr, n, '\n', stream);
}
```

getline

〖功能〗从 stream 读入一整行，将文本(包括换行符和终止空字符在内)保存在缓存中，然后在*lineptr 中保存缓存的地址。

〖原型〗ssize_t getline (char **lineptr, size_t *n, FILE *stream)

〖位置〗stdio.h (GNU)

〖说明〗在调用 getline 之前，应当在*lineptr 中存放缓存区地址，该缓存区有*n 个字节长，通过 malloc 分配。如果该缓存的长度可以保存读入数据，则 getline 将读入数据保存在缓存区中。否则 getline 将使用 realloc 扩大缓存区，然后将新的缓存地址放入*lineptr 中，并且将增加后的长度放入*n。

如果在调用该函数之前设置*lineptr 为空指针，并且*n 为 0，则 getline 将调用 malloc 分配初始缓存区。

任何情况下，getline 返回时的*lineptr 都是一个字符串指针，指向读入文本行。

getline 执行成功时，返回读入的字符数(包括换行符，但是不包括终止空字符)。这一数值可以用来区分读入数据中的空字符和作为终止符插入的空字符。

该函数为 GNU 的扩展，但是从流中读入数据时建议使用该函数。相对应的标准函数性能不够稳定。

如果出现错误或者遇到文件结尾，则 getline 返回-1。

getservbyname

〖功能〗返回使用 proto 协议且名为 name 的服务信息。

〖原型〗struct servent * getservbyname (const char *name, const char *proto)

〖位置〗netdb.h (BSD)

〖说明〗如果无法找到这样的服务，则返回一个空指针。

该函数同时适用于服务器和客户端，服务器用它来确定应当监听哪一个端口。

getwd

〖功能〗与 getcwd 类似，但是不能指定缓存区的大小。

〖原型〗char * getwd (char *buffer)

〖位置〗unistd.h (BSD)

〖说明〗GNU 函数库提供 getwd 函数只是为了与 BSD 兼容。

buffer 参数应当是一个指针，指向至少 PATH_MAX 字节长的一个数组。在 GNU 系统中对文件名称的长度没有限制，因此对保存目录名称的最小空间无法限定。这也正是该函数受到忽视的原因。

obstack_printf

〖功能〗该函数与 asprintf 类似，但是该函数使用对象堆栈分配空间。

〖原型〗int obstack_printf (struct obstack *obstack, const char *template, ...)

〖位置〗stdio.h (GNU)

〖说明〗字符输出到当前对象的末尾，必须使用 obstack_finish 结束对象之后才能得到这些字符。

obstack_vprintf

〖功能〗如果像 vprintf 那样指定变量参数列表，则该函数等效于 obstack_printf。

〖原型〗int obstack_vprintf (struct obstack *obstack, const char *template, va_list ap)

〖位置〗stdio.h (GNU)

open_memstream

〖功能〗该函数打开一个流，写入缓存区。

〖原型〗FILE * open_memstream (char **ptr, size_t *sizeloc)

〖位置〗stdio.h (GNU)

〖说明〗缓存区是动态分配的，并且可以根据需要扩大。

当使用 fclose 关闭流或者使用 fflush 清空流时，更新 ptr 和 sizeloc 的位置，令指针指向缓存区及其大小。仅在流不再执行输出操作时，保存的值才有效。如果继续输出则必须再次清空流，以便于在使用之前保存新值。

缓存区末尾写入一个空字符，该空字符不包括在 sizeloc 中保存的长度值中。

使用 fseek 可以移动流的文件位置。如果移动的文件位置超过数据尾端，则将在插入

空间中填充 0。

以下的样例说明如何使用 open_memstream。

```
#include <stdio.h>
int
main (void)
{
  char *bp;
  size_t size;
  FILE *stream;
  stream = open_memstream (&bp, &size);
  fprintf (stream, "hello");
  fflush (stream);
  printf ("buf = '%s', size = %d\n", bp, size);
  fprintf (stream, ", world");
  fclose (stream);
  printf ("buf = '%s', size = %d\n", bp, size);
  return 0;
}
```

该程序产生的输出如下。

```
buf = 'hello', size = 5
buf = 'hello, world', size = 12
```

open_obstack_stream

〖功能〗该函数打开一个流，等待将数据输出到对象堆栈中。

〖原型〗FILE * open_obstack_stream (struct obstack *obstack)

〖位置〗stdio.h (GNU)

〖说明〗这一动作将在对象堆栈中启动一个对象，并且在输出数据的过程中增长对象。

在流中调用 fflush 可以更新对象的当前大小，以便与写入数据量相匹配。调用 fflush 之后，可以临时检查对象。

使用 fseek 可以改变对象堆栈流的文件位置。如果移动的文件位置超过写入数据的末尾，则插入空间将填充 0。

要使对象永久存在，应当使用 fflush 更新对象堆栈，然后使用 obstack_finish 完成对象并得到其地址。随后的流输出将启动对象堆栈中的一个新对象，并且将后续的输出内容不断添加到该对象中，直至再次执行 fflush 和 obstack_finish 为止。

但是如何得到对象的长度呢？可以调用 obstack_object_size 得到以字节为单位的长度，或者使用如下语句终止对象。

```
obstack_lgrow (obstack, 0);
```

无论采取哪一种方法，都必须在调用 obstack_finish 之前完成(如果愿意也可以同时采用两种方法)。

以下样例说明了如何使用 open_obstack_stream。

```
char *
make_message_string (const char *a, int b)
{
  FILE *stream = open_obstack_stream (&message_obstack);
  output_task (stream);
  fprintf (stream, ": ");
  fprintf (stream, a, b);
  fprintf (stream, "\n");
  fclose (stream);
  obstack_1grow (&message_obstack, 0);
  return obstack_finish (&message_obstack);
}
```

parse_printf_format

〖功能〗该函数返回 printf 的模板字符串 template 所需要的参数种类和个数。

〖原型〗size_t parse_printf_format (const char *template, size_t n, int *argtypes)

〖位置〗printf.h (GNU)

〖说明〗返回内容保存在数组 argtypes 中，该数组的每个元素都描述一个参数，其信息使用不同的 PA_宏进行编码。

参数 n 说明数组 argtypes 中的元素个数，是该函数可以写入的最多元素个数。

该函数返回 template 所需的参数总量。如果该数目大于 n，则返回信息只描述前 n 个参数。如果需要的信息远多于这些参数，则需要分配一个更大的数组并重新调用 parse_printf_format。

printf

〖功能〗该函数在模板字符串 template 的控制下向 stdout 流输出可选参数。

〖原型〗int printf (const char *template, ...)

〖位置〗stdio.h (ISO)

〖说明〗返回打印的字符个数，如果出现输出错误则返回一个负值。

putc

〖功能〗该函数与 fputc 类似，但是多数系统都作为宏执行，以提高执行速度。

〖原型〗int putc (int c, FILE *stream)

〖位置〗stdio.h (ISO)

〖说明〗与通常的宏规则不同，可以多次为 stream 参数赋值。putc 通常是输出单个字符的最佳函数。

puts

〖功能〗将字符串 s 写入 stdout 流，并添加一个换行符。

〖原型〗int puts (const char *s)

〖位置〗stdio.h (ISO)

〖说明〗该函数不写入字符串终止空字符(请注意 fputs 函数不写入换行符)。

puts 函数是打印简单消息时最简便实用的函数。例如：

```
puts("This is a message.");
```

putw

〖功能〗将字 w(一个 int 值)写入 stream 流。

〖原型〗int putw (int w, FILE *stream)

〖位置〗stdio.h (SVID)

〖说明〗提供该函数是为了与 SVID 保持兼容，但是建议读者使用 fwrite 函数代替。

putchar

〖功能〗当 stream 参数的值等于 stdout 时，该函数等价于 putc。

〖原型〗int putchar (int c)

〖位置〗stdio.h (ISO)

register_printf_function

〖功能〗该函数用于定义转换说明符 spec。

〖原型〗int register_printf_function (int spec, printf_function handler-function, printf_arginfo_function arginfo-function)

〖位置〗printf.h (GNU)

〖说明〗如果 spec 为 z，则定义的转换为%z。可以重新定义%s 这样的内置转换说明，但是#标志字符和 l 类型转换符不能用做转换说明，使用这些字符调用该函数无效。

handle-function 是发现转换说明出现在字符串中时，printf 调用的函数。如果指定一个空指针，则删除 spec 已有的全部处理函数。

arginfo-function 是发现转换说明出现在字符串中时，parse_printf_format 调用的函数。

注意，在 GNU C 函数库的 2.0 版本之前，除非用户使用 parse_printf_format 函数，否则无需安装 arginfo-function。现在情况已经不同，如果发现该格式说明出现在格式字符串中，则调用任何 printf 函数都将调用该函数。

调用成功返回 0，调用失败返回-1(当 spec 超出范围时才会出现)。

可以重新定义标准的输出转换说明，但是因为有可能产生混淆，所以不建议这样做。如果这样做，那么其他人编写的库程序可能被破坏。

rewinddir

〖功能〗该函数用来重新初始化目录流 dirstream，因此如果调用 readdir 将会再次返回目录中第 1 项的有关信息。

〖原型〗void rewinddir (DIR *dirstream)

〖位置〗dirent.h (POSIX.1)

〖说明〗如果自从使用 opendir 打开目录之后曾经添加或者删除过文件，则函数也将

加以说明(如果在最后一次调用 opendir 或者 rewinddir 之后这些文件被添加或者删除，那么其项目内容可能返回，也可能不返回)。

scanf

〖功能〗按照模板字符串 template 的要求从 stdin 流读入格式化的输入内容。

〖原型〗int scanf (const char *template, ...)

〖位置〗stdio.h (ISO)

〖说明〗可选的参数是指向接收结果值位置的指针。

返回值通常是成功读入字符的数目。如果在匹配实现(包括匹配空白字符和匹配模板中的文字字符)之前出现文件结尾，则返回 EOF。

setbuf

〖功能〗如果 buf 是一个空指针，则该函数的作用等同于调用以_IONBF 为 mode 参数的 setvbuf。否则，该函数等同于调用以 buf、_IONBF 为 mode 和 BUFSIZ 为 size 参数的 setvbuf。

〖原型〗void setbuf (FILE *stream, char *buf)

〖位置〗stdio.h (ISO)

〖说明〗提供该函数是为了与旧版 BSD 代码兼容，建议使用 setvbuf 代替。

setbuffer

〖功能〗如果 buf 是空指针，该函数使 stream 无缓冲。否则，使 stream 把 buf 当缓存区进行缓存。参数 size 指定 buf 的长度。

〖原型〗void setbuffer (FILE *stream, char *buf, size_t size)

〖位置〗stdio.h (BSD)

〖说明〗提供该函数是为了与旧版 BSD 代码兼容，建议使用 setvbuf 代替。

setlinebuf

〖功能〗该函数设置 stream 为行缓存模式，并且为用户分配缓存区。

〖原型〗void setlinebuf (FILE *stream)

〖位置〗stdio.h (BSD)

〖说明〗提供函数是为了与旧的 BSD 代码兼容，建议使用 setvbuf 代替。

setvbuf

〖功能〗该函数用于指定流 stream 的缓存模式 mode，包括_IOFBF(完全缓存)、_IOLBF(线性缓存)或者_IONBF(非缓存输入/输出)。

〖原型〗int setvbuf (FILE *stream, char *buf, int mode, size_t size)

〖位置〗stdio.h (ISO)

〖说明〗如果指定一个空指针作为 buf 参数，则该函数将使用 malloc 自己分配一个缓存区，关闭流时将释放该缓存区。

否则 buf 就应当是可以保存至少 size 个字符的字符数组。在流打开并且该数组保留其缓存时不能释放数组的空间。通常该缓存应当静态分配，或者使用 malloc 分配。使用自动数组不是一个好建议，除非在退出声明数组的代码块之前关闭文件。

因为数组中保存流缓存区，因此流 I/O 函数会因为内部原因使用缓存区。当流使用数组缓存时，不要直接访问数组中的值。

该函数成功则返回 0，如果 mode 无效或者请求无法完成则返回一个非 0 值。

snprintf

〖功能〗该函数除参数 size 定义了输出的最大字符数外，与函数 sprintf 类似。后缀的空字符也计算在限制之内，因此必须为字符串 s 分配至少 size 个字节。

〖原型〗int snprintf (char *s, size_t size, const char *template, ...)

〖位置〗stdio.h (GNU)

〖说明〗该函数的返回值是所存储的字符数，不包括终止空字符。如果返回值等于 size-1，则表示在 s 中没有足够的空间用于输出。必需使用一个更大的输出字符串。以下是一个样例。

```
/* Construct a message describing the value of a variable
   whose name is name and whose value is value. */
char *
make_message (char *name, char *value)
{
  /* Guess we need no more than 100 chars of space. */
  int size = 100;
  char *buffer = (char *) xmalloc (size);
  while (1)
    {
      /* Try to print in the allocated space. */
      int nchars = snprintf (buffer, size,
                         "value of %s is %s",
                         name, value);
      /* If that worked, return the string. */
      if (nchars < size)
        return buffer;
      /* Else try again with twice as much space. */
      size *= 2;
      buffer = (char *) xrealloc (size, buffer);
    }
}
```

实际操作过程中使用 asprintf 更为简单。

sprintf

〖功能〗该函数与 printf 类似，但是输出保存在字符数组 s 中，而不是写入一个流。

〖原型〗int sprintf (char *s, const char *template, ...)

〖位置〗stdio.h (ISO)

〖说明〗写入空字符表示字符串结束。

该函数返回保存在数组 s 中的字符个数，但是不包括终止空字符在内。

如果函数在迭代对象之间复制，则结果不可预料，例如，s 同时也是需要按照%s 转换符输出的参数。

警告：由于 sprintf 函数可以输出超过字符串 s 大小的字符个数，因此该函数可能非常危险。谨记在转换说明符中给定的字段宽度只是一个最小值。

为了避免出现以上问题，可以使用 snprintf 或者 asprintf 代替。

sscanf

〖功能〗该函数与 scanf 类似，但是其字符来自以空字符结束的字符串 s，而不是来自某个流。

〖原型〗int sscanf (const char *s, const char *template, ...)

〖位置〗stdio.h (ISO)

〖说明〗遇到字符串结尾时按照文件结尾情况处理。

如果在迭代对象之间复制，则该函数的动作不可预料。

ungetc

〖功能〗该函数将字符 c 退回输入流 stream 中，因此来自 stream 的下一次读取必然首先读入 c。

〖原型〗int ungetc (int c, FILE *stream)

〖位置〗stdio.h (ISO)

〖说明〗如果 c 等于 EOF，则 ungetc 函数不做任何动作，只是返回 EOF。这样就可以使用 getc 的返回值来调用 ungetc，同时不必检查 getc 出现的错误。

退回的字符不一定要求与从流中实际读入的最后一个字符相同。实际上，在使用 ungetc 拒绝之前甚至可以不从流中读入任何字符。但是这种写程序的方法比较奇怪，通常只有在退回一个刚刚从同一个流中读入的字符时，才使用 ungetc。

GNU C 库仅仅支持退回一个字符，也就是说不能在没有输入的情况下连续调用 ungetc 两次。其他系统可能允许退回多个字符，然后再从流中读入时，将得到与退回顺序相反的字符。

退回字符不会更改文件，只影响流的内部缓存。退回流中位于文件末尾的字符将清除流的文件结束标记。

以下的样例说明使用 getc 和 ungetc 跳过空白字符的方法。当函数遇到非空白字符时，字符将被退回，并通过下一次读操作再次浏览。

```
#include <stdio.h>
#include <ctype.h>
void
skip_whitespace (FILE *stream)
{
  int c;
  do
```

```
    /* No need to check for EOF because it is not
       isspace, and ungetc ignores EOF.  */
    c = getc (stream);
  while (isspace (c));
  ungetc (c, stream);
}
```

vasprintf

〖功能〗该函数等效于 asprintf，其变量参数列表与 vprintf 完全一致。

〖原型〗int vasprintf (char **ptr, const char *template, va_list ap)

〖位置〗stdio.h (GNU)

vfprintf

〖功能〗该函数等效于 fprintf，其可变参数列表与 vprintf 完全一致。

〖原型〗int vfprintf (FILE *stream, const char *template, va_list ap)

〖位置〗stdio.h (ISO)

vfscanf

〖功能〗该函数等效于 fscanf，其可变参数列表与 vscanf 完全一致。

〖原型〗int vfscanf (FILE *stream, const char *template, va_list ap)

〖位置〗stdio.h (GNU)

vprintf

〖功能〗该函数与 printf 类似，但是没有直接使用可变数目的参数，而是使用参数列表指针 ap。

〖原型〗int vprintf (const char *template, va_list ap)

〖位置〗stdio.h (ISO)

vsnprintf

〖功能〗该函数等效于 snprintf，其可变参数列表与 vprintf 完全一致。

〖原型〗int vsnprintf (char *s, size_t size, const char *template, va_list ap)

〖位置〗stdio.h (GNU)

vscanf

〖功能〗该函数与 scanf 类似，但是没有直接使用可变数目的参数，而是使用 va_list 类型的参数列表指针 ap。

〖原型〗int vscanf (const char *template, va_list ap)

〖位置〗stdio.h (GNU)

vsprintf

〖功能〗该函数等效于 sprintf，其可变参数列表与 vprintf 完全一致。
〖原型〗int vsprintf (char *s, const char *template, va_list ap)
〖位置〗stdio.h (ISO)

vsscanf

〖功能〗该函数等效于 sscanf，其可变参数列表与 vscanf 完全一致。
〖原型〗int vsscanf (const char *s, const char *template, va_list ap)
〖位置〗stdio.h (GNU)

1.6 底层输入/输出

closedir

〖功能〗关闭目录流 dirstream。
〖原型〗int closedir (DIR *dirstream)
〖位置〗dirent.h (POSIX.1)
〖说明〗函数调用成功时返回 0，失败时返回-1。
该函数定义了如下 errno 错误代码。

EBADF　　表示参数 dirstream 无效。

creat

〖功能〗打开文件。
〖原型〗int creat (const char *filename, mode_t mode)
〖位置〗fcntl.h (POSIX.1)
〖说明〗该函数已经过时。

```
creat(filename, mode)
```

等效于

```
open(filename, O_WRONLY | O_CREAT | O_TRUNC, mode)
```

dup2

〖功能〗该函数将 old 描述符复制到描述符号 new。
〖原型〗int dup2 (int old, int new)
〖位置〗unistd.h (POSIX.1)
〖说明〗如果 old 为无效描述符，则 dup2 不做任何动作，也不会关闭 new。否则，假

定 new 已经被关闭，使用 old 的拷贝将覆盖描述符 new 原来代表的任何意义。

如果 old 和 new 为不同的数字，并且 old 为有效的描述符号，则 dup2 等效于：

```
close (new);
fcntl (old, F_DUPFD, new)
```

但是 dup2 自动完成以上的动作。在调用 dup2 的过程中，关闭 new 和复制 old 这两个动作之间没有间隔。

dup

〖功能〗将描述符 old 复制到第 1 个可用的描述符中(第 1 个描述符号目前没有打开)。

〖原型〗int dup (int old)

〖位置〗unistd.h (POSIX.1)

〖说明〗该函数等价于 fcntl(old, F_DUPFD, 0)。

fclean

〖功能〗清除流 stream，以便清空其缓存。

〖原型〗int fclean (FILE *stream)

〖位置〗stdio.h (GNU)

〖说明〗如果 stream 执行输出任务，则强制其输出。如果 stream 执行输入任务，则将缓存中的数据返回给系统，并安排重新读取。

在其他系统中，多数情况下可以使用 fflush 来清除流。

如果已知流已经被清除，则可以跳过 fclean 或者 fflush。只要缓存为空，则流已经清除。例如一个没有缓存内容的流通常都是被清除的。位于文件结尾位置的输入流也是被清除的。当输出的最后一个字符为换行符时，则该行缓存流也被清除。

在多数系统中，有一种情况无法清除流，那就是当流从一个无法随机访问的文件输入时。这类流通常需要提前读入，而且当文件无法随机访问时，无法恢复已经读入的过剩数据。当某个输入流从一个随机访问文件中读取数据时，fflush 负责清除流，但是文件指针将停留在某个不定位置，必须在进行其他 I/O 操作之前设置文件指针。在 GNU 系统中，使用 fclean 可以避免出现以上的所有问题。

fflush 也可以关闭一个只允许输出的流，因此这是一个清除输出流的有效方法。在 GNU 系统中使用 fclean 关闭输入流。

在使用流的描述符完成类似设置终端模式这样的控制操作之前，无需清除流，这些操作不会影响文件位置，同时也不会受到影响。完成这些操作时可以使用任何描述符，所有的通道将同时受到影响。但是清空之后，已经输出到流中并且仍然缓存的文本将隶属于新的终端模式。

fcntl

〖功能〗对文件描述符 filedes 指定的文件执行 command 指定的命令。

〖原型〗int fcntl (int filedes, int command, ...)

〖位置〗fcntl.h (POSIX.1)

〖说明〗有些命令需要提供额外的参数，这些额外的参数和返回值，以及错误情况将在各条命令的详细说明中解释。

下面简要介绍几个不同的命令。

F_DUPFD	复制文件描述符(返回指向同一个打开文件的另一个文件描述符)。
F_GETFD	得到与文件描述符相关的标记。
F_SETFD	设置与文件描述符相关的标记。
F_GETFL	得到与打开文件相关的标记。
F_SETFD	设置与打开文件相关的标记。
F_GETLK	得到文件锁。
F_SETLK	设置或者清除文件锁。
F_SETLKW	与 F_SETLK 类似，但是需要等待完成。
F_GETOWN	得到接收 SIGIO 信号的进程或者进程组 ID。
F_SETOWN	设置接收 SIGIO 信号的进程或者进程组 ID。

fdopen

〖功能〗返回为文件描述符 filedes 准备的新的流。

〖原型〗FILE * fdopen (int filedes, const char *opentype)

〖位置〗stdio.h (POSIX.1)

〖说明〗参数 opentype 的解释方法与 fopen 函数相同，但是不允许使用 b 选项，这是因为 GNU 无法区分文本和二进制文件。同样 w 和 w+也无法切断文件，这两个选项只有在打开文件时才能发挥作用，而此时文件已经被打开。必须确认 opentype 参数与打开文件描述符所处的实际模式相匹配。

返回值是一个新的流。如果无法创建流(例如，如果文件描述符指定的文件模式不允许实现 opentype 参数指定的访问)，则返回一个空指针。

在某些其他系统中，fdopen 可能无法检测出不允许执行 opentype 指定访问的文件描述符模式。

fileno

〖功能〗该函数返回与 stream 流相关联的文件描述符。

〖原型〗int fileno (FILE *stream)

〖位置〗stdio.h (POSIX.1)

〖说明〗如果出现错误(例如 stream 无效等)，或者 stream 没有对某个文件执行 I/O 操作，则返回-1。

lseek

〖功能〗该函数用来改变使用 filedes 描述符指定的文件的位置。

〖原型〗off_t lseek (int filedes, off_t offset, int whence)

〖位置〗unistd.h (POSIX.1)

〖说明〗whence 参数说明如何解释 offset，具体方法与 fseek 函数相同，该参数必须是 SEEK_SET、SEEK_CUR 或者 SEEK_END 等 3 个符号常量之一。

SEEK_SET	说明 whence 为从文件开始位置计算的字符个数。
SEEK_CUR	说明 whence 为从当前文件位置开始计算的字符个数，计数值可以为正数或者负数。
SEEK_END	说明 whence 为从文件结束位置计算的字符个数。负的计数值说明位于文件范围之内，正的计数值说明位置已经超出了当前的结尾。

如果设置的位置超出当前结尾，并且正在写数据，那么文件将扩充大于等于该计数值的长度。lseek 的返回值通常等于最终得到的文件位置，使用从文件头开始的字节数衡量。可以利用这一特性与 SEEK_CUR 相结合读出当前的文件位置。如果需要追加文件，则仅仅使用 SEEK_END 将文件位置设置为当前文件结尾还不够。可能会有其他的进程在你的程序找到文件位置之后执行写操作之前写入更多数据，从而扩大文件长度，使得你的写入位置发生错误。这种情况下应当使用 O_APPEND 操作模式。设置的文件位置可以超过当前文件结尾，而这一动作并不会扩大文件长度，lseek 永远不会改动文件。但是以后在该位置上的输出则将扩大文件长度。位于原有文件结尾和新位置之间的字符将填充为 0。这种扩大文件的方法将出现一个“洞”：填充 0 的这些块并没有真正分配在硬盘上，因此文件真正占有的空间没有表面看起来那么大，这类文件被称为“稀疏文件”。如果文件位置无法更改，或者操作无效，则 lseek 返回值-1。该函数定义的 errno 错误代码如下。

EBADF	filedes 不是有效的文件描述符。
EINVAL	whence 参数值无效，或者最终的文件偏移值无效。
ESPIPE	filedes 对应的对象无法定位，可能是一个管道、FIFO 或者终端设备(POSIX.1 仅仅为管道和 FIFO 规定了这类错误，但是在 GNU 系统中如果对象无法定位，通常会出现 ESPIPE 错误)。

lseek 函数其实是 fseek、ftell 和 rewind 函数的原始形式，这 3 个函数都是对流执行操作，而不再对文件描述符执行操作。

如果多次打开一个文件或者使用 dup 复制文件描述符，同一个文件可能得到多个描述符。分别从不同 open 调用得到的描述符，其文件位置也相互独立。对一个描述符使用 lseek 时，对其他描述符将不发生作用。例如：

```
{
  int d1, d2;
  char buf[4];
  d1 = open ("foo", O_RDONLY);
  d2 = open ("foo", O_RDONLY);
  lseek (d1, 1024, SEEK_SET);
  read (d2, buf, 4);
}
```

将读取文件 foo 的前 4 个字符(对于真实的程序而言，错误检查代码是非常必要的。但是为了内容更加简洁这里只能省略该部分代码)。相对而言，通过复制得到的描述符将与被复制的源操作符共享相同的文件位置。更改其中一个文件位置的任何操作，包括读数据或者写数据，都将同时影响两者。例如：

```
{
  int d1, d2, d3;
  char buf1[4], buf2[4];
  d1 = open ("foo", O_RDONLY);
  d2 = dup (d1);
  d3 = dup (d2);
  lseek (d3, 1024, SEEK_SET);
  read (d1, buf1, 4);
  read (d2, buf2, 4);
}
```

将读入文件 foo 中从第 1024 个字符开始的 4 个字符，然后从第 1028 个字符开始读入 4 个以上的字符。

opendir

〖功能〗该函数打开并返回一个文件名为 dirname 的目录流，以供读取。流的类型为 DIR *。

〖原型〗DIR * opendir (const char *dirname)

〖位置〗dirent.h (POSIX.1)

〖说明〗如果函数执行失败，则返回一个空指针。除了常见的文件名错误之外，该函数还定义了如下 errno 错误代码。

EACCES	名为 dirname 的目录没有读权限。
EMFILE	进程打开的文件过多。
ENFILE	整个系统或包含该目录的文件系统，此时无法再打开额外的文件(这一问题不可能在 GNU 系统中发生)。

readdir

〖功能〗该函数从目录中读入下一项。

〖原型〗struct dirent * readdir (DIR *dirstream)

〖位置〗dirent.h (POSIX.1)

〖说明〗该函数通常返回一个指针，指向包含文件信息的结构。该结构是静态分配的，下一次调用时将覆盖其内容。

可移植性说明：在某些系统上，readdir 可能不会返回.和..，尽管这些内容在任何目录中都是有效的文件名。

如果目录中没有更多内容或者检测到错误，则 readdir 将返回一个空指针。该函数还定义了如下 errno 错误代码。

EBADF　　dirstream 参数无效。

readdir 不是线程安全的。多个线程对同一个 dirstream 使用 readdir 时，返回值将被覆盖。重要情况下应当使用 readdir_r。

select

〖功能〗阻塞调用过程，直至指定的文件指示符集合中的任意一项开始活动，或者超出了规定时间要求。

〖原型〗int select (int nfds, fd_set *read-fds, fd_set *write-fds, fd_set *except-fds, struct timeval *timeout)

〖位置〗sys/types.h (BSD)

〖说明〗read-fds 参数指定的文件描述符需要检查是否已经做好读取准备，write-fds 文件描述符则需要检查是否已经做好写入准备，同时还应当检查 except-fds 文件描述符是否出现例外的情况。如果不需要检查以上任何一种情况，可以为相应参数传递一个空指针。

文件描述符位于文件末尾时认为已经做好读取准备。服务器套接口存在一个可以使用 accept 接收的挂起连接时，认为已经做好读取准备。而客户端套接口的连接完全建立时，则认为已经做好写入准备。

“例外情况”所指的并非错误——执行错误的系统调用时将立即报错，并且不能构成描述符的状态。而是包括类似套接口中出现一条紧急信息的情况。

函数仅仅检查第 1 个 nfds 文件描述符，通常将 FD_SETSIZE 传递给该参数。

timeout 参数指定最长的等待时间。如果为该参数传递一个空指针，则意味着在第 1 个文件描述符准备好之前可能发生阻塞。否则提供的时间就应当按照 struct timeval 格式给出。如果需要立即检查出已经准备好的描述符，则指定时间为 0(struct timeval 中包含所有的 0)。

函数的正常返回值是所有集合中准备好的文件描述符的总量。每个参数集合都将覆盖为准备执行相应操作的描述符信息。因此要了解某个描述符 desc 是否输入，需要在 select 函数返回之后调用 FD_ISSET (desc, read-fds)。

如果函数由于超时返回，则将返回 0 值。

任何信号都将导致 select 函数立即返回。因此如果程序使用信号，则不能依靠 select 函数等待指定的时间。如果需要确认等待一段指定的时间，则必须检查 EINTR 错误并且使用根据当前时间计算的超时时间量重复执行 select。可以参见下面的样例。

如果出现错误，函数返回-1，并且不会更改参数文件描述符集合。该函数定义的 errno 错误代码如下。

EBADF	指定的文件描述符集合中有一个无效文件描述符。
EINTR	操作被信号中断。
EINVAL	timeout 参数无效，其内容为负值或者值过大。

可移植性说明：select 函数是 BSD Unix 的特性。

下面的样例讲解如何使用 select 建立从文件描述符读入的超时时间量，函数 input_timeout 在文件描述符可以输入或者超时之前将阻塞调用进程。

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/time.h>
int
input_timeout (int filedes, unsigned int seconds)
{
  fd_set set;
  struct timeval timeout;
  /* Initialize the file descriptor set. */
  FD_ZERO (&set);
  FD_SET (filedes, &set);
  /* Initialize the timeout data structure. */
  timeout.tv_sec = seconds;
  timeout.tv_usec = 0;
  /* select returns 0 if timeout, 1 if input available, -1 if error. */
  return TEMP_FAILURE_RETRY (select (FD_SETSIZE,
                                     &set, NULL, NULL,
                                     &timeout));
}
int
main (void)
{
  fprintf (stderr, "select returned %d.\n",
           input_timeout (STDIN_FILENO, 5));
  return 0;
}
```

write

〖功能〗该函数从 buffer 向描述符为 filedes 的文件输出最多 size 个字节。buffer 中的数据不一定是字符型字符串，空字符将和其他字符一样输出。

〖原型〗ssize_t write (int filedes, const void *buffer, size_t size)

〖位置〗unistd.h (POSIX.1)

〖说明〗返回值为实际输出的字节数。可能等于 size，但是通常会小于 size。通常程序应当在循环中调用 write 函数，在输出所有数据之前不断重复。

write 函数一旦返回，等待输出的数据可以立即读回，但是无需立即输出到永久存储空间中。可以使用 fsync 验证数据是否在继续操作前已经永久保存起来(系统可以执行批量的连续输出，然后在方便时一次完成，这样效率将更高。通常将在一分钟之内或者更短的时间里写入磁盘)。write 函数可以使用 O_FSYNC 打开模式，在返回之前将数据保存到磁盘上。

如果出现错误，函数将返回-1。该函数定义的 errno 错误代码如下。

EAGAIN	通常在写操作完成之前将阻塞 write 函数。但是如果文件设置了 O_NONBLOCK 标志，则不会输出任何数据，将立即返回并报错。一种情况下可能导致进程阻塞输出，就是输出的终端设备支持流控制，而输出由于接收到 STOP 字符被挂起。 兼容性说明：多数 BSD Unix 版本都使用不同的错误代码 EWOULDBLOCK。在 GNU 库中，EWOULDBLOCK 就是 EAGAIN 的别名，因此使用什么名称并不重要。在有些系统中，如果内核没有足够的物理内存锁定用户页面，那么从字符文件输出大量数据也可能失败并报错 EAGAIN。这就要求传递直接内存的设备访问用户内存，也就是说不包括终端在内，因为它们通常都使用内核中独立的缓存区。这一问题不会在 GNU 系统中出现。
EBADF	参数 filedes 是无效的文件描述符，或者无法按照写模式打开。
EFBIG	文件的大小超过当前版本可以支持的范围。
EINTR	write 操作在等待完成之前的阻塞过程中被某个信号中断。信号不一定导致 write 返回 EINTR，但是将使得成功的 write 输出少于请求数量的字节。
EIO	对于许多设备和磁盘文件而言，这种错误代码表示硬件错误。
ENOSPC	保存文件的设备已满。
EPIPE	当试图写入的管道或者 FIFO 不是由某个进程按照读模式打开时，返回该错误。发生这一错误时，同时向进程发送一个 SIGPIPE 信号。

除非已经安排了预防 EINTR 失败，否则用户需要在每次调用 write 失败之后检查 errno，如果错误为 EINTR，则应当重复调用。最简单的方法是使用宏 TEMP_FAILURE_RETRY，如下所示。

```
nbytes = TEMP_FAILURE_RETRY(write (desc, buffer, count));
```

write 函数是所有对流写入的函数的潜在原型。

1.7　文件系统接口

access

〖功能〗确定文件的访问许可权限。

〖原型〗int access (const char *filename, int how)

〖位置〗unistd.h (POSIX.1)

〖说明〗该函数检查名为 filename 的文件是否可以被由 how 参数所指定的方式访问。how 参数既可是标志 R_OK、W_OK、X_OK 的逐位 OR，也可以是存在测试 F_OK。

该函数使用调用进程的真实用户或组 ID，而不是使用其有效 ID 来检查访问许可。造成的结果是，如果在 setuid 或者 setgid 程序中使用该函数，将会给出实际运行该程序的用户的相关信息。

如果该函数返回 0 值表示访问被许可，否则将返回-1。(换句话说，作为一个判定函数，如果请求的访问被拒绝，则 access 返回真值。)

另外对于通常的文件名错误，该函数定义的 errno 错误代码如下所示。

EACCES how	参数指定的访问被拒绝。
ENOENT	文件不存在。
EROFS	在一个只读文件系统中对某文件请求写操作许可。

access 函数中的 how 参数所使用的宏在头文件 unistd.h 中定义，其值为整型常量。

宏	含义
int R_OK	测试读许可。
int W_OK	测试写许可。
int X_OK	测试执行/搜索许可。
int F_OK	测试文件是否存在。

chdir

〖功能〗设置进程的工作目录。
〖原型〗int chdir (const char *filename)
〖位置〗unistd.h (POSIX.1)
〖说明〗该函数设置进程的工作目录为 filename。

该函数调用成功则返回值为 0，返回值为-1 表示出现错误。该函数定义的错误代码 errno 都是常见的文件名语法错误，如果 filename 不是一个目录，则也可能出现 ENOTDIR 错误。

chmod

〖功能〗设置文件的访问权限位。
〖原型〗int chmod (const char *filename, mode_t mode)
〖位置〗sys/stat.h (POSIX.1)
〖说明〗该函数将文件 FILENAME 的访问权限位设置为 mode。

如果 filename 是一个符号链接，那么 chmod 将改变该链接指向的文件权限，而不改变链接本身。

如果调用成功则函数返回 0，否则返回-1。除了常见的文件名称错误之外，该函数还定义了如下 errno 错误代码。

ENONET	命名文件不存在。
EPERM	该进程无权更改当前文件访问权限，只有文件所有者(由进程的有效用户 ID 判定)或者特许用户可以更改。
EROFS	文件驻留在只读文件系统中。

EFTYPE MODE 设置 S_ISVIX 位(二进位制方程)，而命名文件不是一个目录。有些系统不允许在非目录文件中设置二进位制方程，有些系统则允许其中一部分为非目录文件的

位指定含义。在那些非目录文件的二进位制方程没有含义的系统上，只能得到 EFTYPE，因此清除 MODE 位之后再调用 CHMOD 较为安全。

chown

〖功能〗改变文件的所有者为 owner，所有组为 group。

〖原型〗int chown (const char *filename, uid_t owner, gid_t group)

〖位置〗unistd.h (POSIX.1)

〖说明〗在某些系统中改变文件的所有者将清除文件权限中的 SET-USER-ID 和 SET-GROUP-ID 位(因为这些位可能不适用于新的所有者)。其他的文件权限位不会发生改变。

函数调用成功则返回 0，失败则返回-1。除了常见的文件名错误之外，该函数还定义了如下 errno 错误代码。

EPERM	该进程没有做出相应改变的权限。只有特许用户或者文件的所有者才能改变文件的组。在多数文件系统中，只有特许用户才能改变文件所有者，有些文件系统允许所有者本人进行更改。访问一个远程文件系统时，由实际保存文件的系统来决定你的行为，而不是由你的运行程序所在的系统决定。
EROFS	文件位于只读文件系统中。

closedir

〖功能〗关闭目录流 dirstream。

〖原型〗int closedir (DIR *dirstream)

〖位置〗dirent.h (POSIX.1)

〖说明〗该函数调用成功则返回 0，失败则返回-1。

该函数定义了如下 errno 错误代码。

EBADF	参数 dirstream 无效。

fchmod

〖功能〗与 chmod 类似，但是将改变目前通过描述符 filedes 打开的文件权限。

〖原型〗int fchmod (int filedes, int mode)

〖位置〗sys/stat.h (BSD)

〖说明〗调用成功则返回 0，失败则返回-1。该函数定义了如下 errno 错误代码。

EBADF	参数 filedes 是无效的文件描述符。
EINVAL	参数 filedes 对应于一个管道、套接口、或者其他没有实际访问权限的对象。
EPERM	该进程没有权限更改当前文件的访问权限，只有文件所有者(通过进程的有效用户 ID 判断)或者特许用户才能更改。

EROFS　　文件驻留在一个只读文件系统中。

fchown

〖功能〗与 chown 类似，但是改变打开文件描述符为 filedes 的文件所有者。

〖原型〗int fchown (int filedes, int owner, int group)

〖位置〗unistd.h (BSD)

〖说明〗调用成功则返回 0，失败则返回-1。该函数定义了如下 errno 错误代码。

EBADF　　参数 filedes 是无效的文件描述符。

EINVAL　　参数 filedes 对应于一个管道或者套接口，而不是一个普通文件。

EPERM　　该进程没有权限来完成请求的更改。详细内容参见 chmod。

EROFS　　文件驻留在一个只读文件系统中。

fstat

〖功能〗与 stat 类似，但是使用打开文件描述符作为参数代替文件名称。

〖原型〗int fstat (int filedes, struct stat *buf)

〖位置〗sys/stat.h (POSIX.1)

〖说明〗与 stat 相同，fstat 调用成功时返回 0，失败时返回-1。fstat 定义了如下 errno 错误代码。

EBADF　　参数 filedes 是无效的文件描述符。

getcwd

〖功能〗返回提供的字符数组 buf，其中保存有表示当前工作目录的绝对文件名称。

〖原型〗char * getcwd (char *buffer, size_t size)

〖位置〗unistd.h (POSIX.1)

〖说明〗参数 size 向系统说明需要为 buffer 分配的大小。

该函数的 GNU 库版本还允许为 buffer 参数指定空指针。然后 getcwd 负责使用 malloc 自动分配缓存区。如果 size 大于 0，则 buffer 分配的大小为 size，否则 buffer 的大小按照结果的需要来分配。

调用成功则返回 buffer，失败则返回一个空指针。该函数还定义了如下 errno 错误代码。

EINVAL　　参数 size 等于 0，并且 buffer 不是空指针。

ERANGE　　参数 size 小于工作目录名称的长度，需要再次分配一个较大的数组。

EACCES　　拒绝读取或者搜索文件名的某个部分。

下面的样例说明了如何使用 getcwd 标准操作实现 GNU 的 getcwd(NULL, 0)操作。

```
char *
gnu_getcwd ()
```

```
{
  int size = 100;
  char *buffer = (char *) xmalloc (size);
  while (1)
    {
      char *value = getcwd (buffer, size);
      if (value != 0)
        return buffer;
      size *= 2;
      free (buffer);
      buffer = (char *) xmalloc (size);
    }
}
```

getumask

〖功能〗返回当前进程中文件创建模版的当前值。

〖原型〗mode_t getumask (void)

〖位置〗sys/stat.h (GNU)

〖说明〗该函数为 GNU 扩展特性。

getwd

〖功能〗与 getcwd 类似，但是不能指定缓存区的大小。

〖原型〗char * getwd (char *buffer)

〖位置〗unistd.h (BSD)

〖说明〗GNU 库提供 getwd 函数只是为了与 BSD 兼容。

buffer 参数应当是一个指针，指向至少 PATH_MAX 字节长的一个数组。在 GNU 系统中对文件名称的长度没有限制，因此对保存目录名称的最小空间无法限定。这也正是该函数受到忽视的原因。

link

〖功能〗该函数为名为 oldname 的文件建立一个新链接，新的名称为 newname。

〖原型〗int link (const char *oldname, const char *newname)

〖位置〗unistd.h (POSIX.1)

〖说明〗调用成功则返回 0，失败则返回-1。除了 oldname 和 newname 出现的常见文件名错误之外，该函数还定义了如下 errno 错误代码。

EACCES	在写入新链接的目录中没有写权限。
EEXIST	已经存在名为 newname 的文件。如果希望使用新链接替换旧链接，则首先必须强制删除旧链接。
EMLINK	名为 oldname 的文件中已经存在的链接过多(文件的最大链接数为 LINK_MAX)。
ENOENT	名为 oldname 的文件不存在，不能为不存在的文件创建链接。

ENOSPC	保存新链接的目录或者文件系统已满，并且不能扩大。
EPERM	在 GNU 系统和一些其他系统中，不能为目录创建链接。许多系统只允许特权用户执行这类操作。
EROFS	保存新链接的目录位于只读文件系统中，因此无法更改。
EXDEV	newname 中指定的目录与已有文件所处的文件系统不同。
EIO	读写文件系统时出现硬件错误。

lstat

〖功能〗类似 stat，但是没有符号链接。

〖原型〗int lstat (const char *filename, struct stat *buf)

〖位置〗sys/stat.h (BSD)

〖说明〗如果 filename 是一个符号链接的名称，则 lstat 返回链接本身的有关信息，否则 lstat 的功能与 stat 一致。

mkdir

〖功能〗该函数创建一个名为 filename 的新的空目录。

〖原型〗int mkdir (const char *filename, mode_t mode)

〖位置〗sys/stat.h (POSIX.1)

〖说明〗参数 mode 规定新目录文件的文件权限。

返回 0 说明执行成功，返回-1 说明失败。除了常见的文件名语法错误之外，该函数还定义了如下 errno 错误代码。

EACCES	添加新目录的父目录没有写权限。
EEXIT	文件 filename 已经存在。
EMLINK	父目录的链接过多。设计合理的文件系统不会出现该错误，因为其中允许的链接数超过了磁盘可以承受的数目。但是仍然必须考虑到这种错误的可能性，因为通过网络访问其他机器上的文件系统也可能造成这一结果。
ENOSPC	文件系统没有足够的空间创建新目录。
EROFS	创建目录的父目录位于只读文件系统中，并且无法更改。

使用该函数时，程序应当包括头文件 sys/stat.h。

mknod

〖功能〗该函数创建一个特殊文件 filename。

〖原型〗int mknod (const char *filename, int mode, int dev)

〖位置〗sys/stat.h (BSD)

〖说明〗mode 说明文件模式，其中可能包括一些特殊文件位，例如 S_IFCHR(适用于字符特殊文件)或者 S_IFBLK(适用于块特殊文件)。

参数 dev 说明特殊文件引用的设备，确切的解释根据创建的特殊文件种类而定。

成功时返回 0，错误返回-1。除了常见的文件名错误之外，该函数还定义了如下 errno 错误代码。

EPERM	调用进程没有特许权限，只有超级用户才可以创建特殊文件。
ENOSPC	包含新文件的目录或者文件系统已满，无法扩展。
EROFS	包含新文件的目录位于只读文件系统中，无法更改。
EEXIST	文件 filename 已经存在。如果需要替换该文件，则必须首先明确删除旧文件。

mkstemp

〖功能〗该函数如同 mktemp 一样生成一个惟一的文件名，但同时使用 open 打开文件。

〖原型〗int mkstemp (char *template)

〖位置〗unistd.h (BSD)

〖说明〗如果成功，则更改 template 并且返回读写该文件的文件描述符。如果无法创建一个名称惟一的文件，则 template 赋值为一个空字符串，并且返回-1。如果 template 没有以 XXXXXX 结尾，则返回-1 并且不更改 template。

mkstemp 与 mktemp 函数不同，mkstemp 实际上可以确保创建一个惟一的文件，并且不会与其他任何试图创建临时文件的程序发生冲突。这是因为该函数调用 open 时使用了 O_EXCL 标志位，说明需要创建一个新文件，如果文件已经存在则提示错误。

mktemp

〖功能〗该函数更改 template 并生成一个惟一的文件名。

〖原型〗char * mktemp (char *template)

〖位置〗unistd.h (Unix)

〖说明〗如果成功，则返回更改后的 template。如果找不到惟一的文件名，则返回的 template 为一个空字符串。如果 template 不是以 XXXXXX 结尾，则函数返回一个空指针。

opendir

〖功能〗该函数打开并返回一个文件名为 dirname 的目录流，以供读取。流的类型为 DIR*。

〖原型〗DIR * opendir (const char *dirname)

〖位置〗dirent.h (POSIX.1)

〖说明〗如果函数执行失败，则返回一个空指针。除了常见的文件名错误之外，该函数还定义了如下 errno 错误情况。

EACCES	名为 dirname 的目录没有读权限。
EMFILE	进程打开的文件过多。
ENFILE	整个系统或包含该目录的文件系统，此时无法再打开额外的文件(这一问题不可能在 GNU 系统中发生)。

readdir

〖功能〗该函数从目录中读入下一项。

〖原型〗struct dirent * readdir (DIR *dirstream)

〖位置〗dirent.h (POSIX.1)

〖说明〗该函数通常返回一个指针，指向包含文件信息的结构。该结构是静态分配的，下一次调用时将覆盖其内容。

可移植性说明：在某些系统上，readdir 可能不会返回.和..，尽管这些内容在任何目录中都是有效的文件名。

如果目录中没有更多内容或者检测到错误，则 readdir 将返回一个空指针。该函数还定义了如下 errno 错误代码。

EBADF　　dirstream 参数无效。

readdir 不是线程安全的。多个线程对同一个 dirstream 使用 readdir 时，返回值将被覆盖。重要情况下，应当使用 readdir_r。

readdir_r

〖功能〗该函数是 readdir 的可重入版本。

〖原型〗int readdir_r (DIR *dirstream, struct dirent *entry, struct dirent **result)

〖位置〗dirent.h (GNU)

〖说明〗该函数象 readdir 一样，返回目录的下一项。但是为了避免同时运行的线程之间的冲突，其结果没有保存在内部内存中。而是使用参数 entry 说明保存结果的位置。

如果成功读取下一项则返回值为 0，此时指向结果的指针在*result 中返回，*result 无需与 entry 相同。如果在执行 readdir_r 时发生某些错误，则函数返回-1，所设置的 errno 变量与 readdir 相同。

可移植性说明：在某些系统中，即便 struct dirent 中没有可用的 d_reclen 元素并且文件名符合允许的最大长度，readdir_r 也可能不会返回结束字符串作为文件名称。现代系统都拥有 d_reclen 字段，而在旧系统中多线程也并不危险。在任何一种情况下，使用 readdir 函数都不会出现这类问题，因此即便在没有 d_reclen 字段的系统上，也可以利用外部锁定来使用多线程。

readlink

〖功能〗该函数从符号连接 filename 中获取值。

〖原型〗int readlink (const char *filename, char *buffer, size_t size)

〖位置〗unistd.h (BSD)

〖说明〗链接指向的文件名称被拷贝到 buffer 中。该文件名字符串不是以空字符结尾的，通常 readlink 返回拷贝的字符个数。size 参数指定拷贝字符的最大数目，通常就是 buffer 的分配大小。

如果返回值等于 size，则无法分辨是否有空间返回整个名称。因此可以创建一个更大的缓存区，然后再次调用 readlink，样例如下。

```
char *
readlink_malloc (char *filename)
{
  int size = 100;
  while (1)
    {
      char *buffer = (char *) xmalloc (size);
      int nchars = readlink (filename, buffer, size);
      if (nchars < size)
        return buffer;
      free (buffer);
      size *= 2;
    }
}
```

如果出现错误则该函数返回-1。除了常见的文件名错误之外，该函数还定义了如下 errno 错误代码。

EINVAL	指定文件不是一个符号链接。
EIO	对磁盘读写数据时发生了硬件错误。

remove

〖功能〗该函数为删除文件的 ISO C 函数，与对文件操作的 unlink 和对目录操作的 rmdir 类似。

〖原型〗int remove (const char *filename)

〖位置〗stdio.h (ISO)

rename

〖功能〗该函数使用 newname 重新命名文件 oldname。

〖原型〗int rename (const char *oldname, const char *newname)

〖位置〗stdio.h (ISO)

〖说明〗原来使用名称 oldname 访问的文件随后应当使用 newname 进行访问(如果文件还有除了 oldname 之外的其他别名，那么这些别名可以继续使用)。

包含 newname 文件的目录必须位于文件原来的文件系统中(与 oldname 一致)。

该函数存在一种特殊情况，就是当 oldname 和 newname 是同一个文件的两个不同名称时，为了保持一致性，必须删除 oldname。但是 POSIX 要求在这种情况下 rename 不要做任何动作，并且报告执行成功——尽管存在不一致性。而用户自己的操作系统可能有各自不同的特点。

如果 oldname 不是目录，则任何名为 newname 的已知文件都应当在重命名的操作过程中删除。但是如果 newname 是目录名称，则 rename 函数将执行失败。

如果 oldname 是目录，则 newname 必须不存在或者必须命名一个空目录。在后一种情况下，将首先删除名为 newname 的目录。newname 这个名称不能用于指定被重命名的 oldname 目录的子目录。

如果在操作过程中系统崩溃，那么可能两个名称都仍然存在，但是如果两者同时存在，则 newname 一定保持原样不变。

如果函数操作失败，则返回-1。除了常见的文件名错误之外，该函数还定义了如下 errno 错误代码。

EACCES	保存 newname 或者 oldname 的目录之一拒绝写权限，或者 newname 和 oldname 是目录，而其中一个拒绝写权限。
EBUSY	名为 oldname 或者 newname 的目录所处的系统不允许重命名，其中包括文件系统的安装目录和进程的当前工作目录。
ENOTEMPTY 或 EEXIST	目录 newname 非空。在这种情况下 GNU 系统通常返回 ENOTEMPTY，但是其他一些系统返回 EEXIST。
EINVAL	oldname 是一个目录，并且 newname 位于该目录中。
EISDIR	newname 命名的是一个目录，而 oldname 不是目录。
EMLINK	newname 的父目录链接过多。
ENOENT	文件 oldname 不存在。
ENOSPC	将要包含 newname 的目录没有足够的自由空间，并且文件系统也没有可以扩展的空间。
EROFS	操作涉及到在只读文件系统上对目录执行写操作。
EXDEV	名为 newname 和 oldname 的两个文件位于不同的文件系统中。

rewinddir

〖功能〗该函数用来重新初始化目录流 dirstream，因此如果调用 readdir 将会再次返回目录中第 1 项的有关信息。

〖原型〗void rewinddir (DIR *dirstream)

〖位置〗dirent.h (POSIX.1)

〖说明〗如果目录自从使用 opendir 打开之后曾经添加或者删除过文件，则函数也将加以说明(如果在最后一次调用 opendir 或者 rewinddir 之后这些文件被添加或者删除，那么其项目内容可能返回，也可能不返回)。

rmdir

〖功能〗该函数用于删除一个目录。该目录在删除之前必须为空，也就是说其中只能包含.和..两项。

〖原型〗int rmdir (const char *filename)

〖位置〗unistd.h (POSIX.1)

〖说明〗在多数方面，rmdir 的行为与 unlink 类似。rmdir 函数额外定义的两个 errno 错误代码如下。

ENOTEMPTY 或 EEXIST	删除的目录非空。

这两个错误代码是同义词，有些系统使用 ENOTEMPTY，有些系统使用 EEXIST，GNU 系统通常使用 ENOTEMPTY。

seekdir

〖功能〗把 dirstream 流目录的位置指向 pos。
〖原型〗void seekdir (DIR *dirstream, off_t pos)
〖位置〗dirent.h (BSD)
〖说明〗pos 值是在指定的流中先前调用 telldir 的结果。关闭和重新打开目录都将使 telldir 返回值无效。

stat

〖功能〗该函数在 buf 指向的结构中返回 filename 文件的有关属性信息。
〖原型〗int stat (const char *filename, struct stat *buf)
〖位置〗sys/stat.h (POSIX.1)
〖说明〗如果 filename 是符号链接的名称，则得到的属性信息描述链接指向的文件。如果链接指向一个不存在的文件名称，则 stat 函数将出现错误，同时报告文件不存在。

如果操作成功则返回 0，如果操作失败则返回-1。除了常见的文件名称错误之外，该函数还定义了如下 errno 错误代码。

ENOENT	名为 filename 的文件不存在。

symlink

〖功能〗给 oldname 一个名为 newname 的符号链接。
〖原型〗int symlink (const char *oldname, const char *newname)
〖位置〗unistd.h (BSD)
〖说明〗返回值通常为 0，返回值-1 表明有错误。除了常见的文件名语法错误外，以下是为该函数定义的 errno 错误代码。

EEXIST	名为 newname 的文件已经存在。
EPROFS	文件 newnname 在只读文件系统中。
ENOSPC	目录或文件系统不可扩展为新的链接。
EIO	在磁盘上读写数据时出现硬件错误。

telldir

〖功能〗该函数返回文件流 dirstream 的文件位置，可以使用含有该值的 seekdir 在该位置存储文件流。

〖原型〗off_t telldir (DIR *dirstream)

〖位置〗dirent.h (BSD)

tempnam

〖功能〗该函数产生一个惟一的临时文件名。如果 prefix 不是空指针，最多有 5 个串的字符用做文件名的前缀。返回值是使用 malloc 重新分配的字符串。当不再需要时应使用 free 释放存储空间。

〖原型〗char * tempnam (const char *dir, const char *prefix)

〖位置〗stdio.h (SVID)

〖说明〗由于串是动态分配的，因此该函数是可重入的。

临时文件名的目录前缀由顺序检测以下每一项来决定，目录必须存在且可写。

• 环境变量 TMPDIR，如果定义了该环境变量的话，出于安全原因仅当程序是非 SUID 或 SGID 时才发生。

• 参数 dir，如果不是空指针的话。

• P_tmpdir 宏的值。

• 目录‘/tmp’。

该函数用于提供与 SVID 的兼容性。

tmpfile

〖功能〗为更新模式创建一个临时二进制文件，以 wb+模式调用 fopen 一样。

〖原型〗FILE * tmpfile (void)

〖位置〗stdio.h (ISO)

〖说明〗文件在其关闭或程序结束时自动删除。(在其他 ISO C 系统中，如果程序非正常结束，则文件不会被删除。)

该函数为可重入函数。

tmpnam

〖功能〗该函数构造并返回一个有效的文件名，且不命名任何现有的文件。

〖原型〗char * tmpnam (char *result)

〖位置〗stdio.h (ISO)

〖说明〗如果参数 result 是空指针，则返回值是指向一个可被后续调用改写的内部静态串，因此也使函数不可重入。否则，result 是指向一个至少有 L_tmpnam 个字符的一个数组，并且将返回结果写入该数组。

如果不移走先前创建的文件而多次调用 tmpnam 时会失败，其原因是：临时文件名的固定长度仅向有限个不同的文件名提供空间。如果 tmpnam 失败，则返回空指针。

tmpnam_r

〖功能〗与函数 tmpnam 基本相同，但该函数不允许 result 为空指针。

〖原型〗char * tmpnam_r (char *result)

〖位置〗stdio.h (GNU)

〖说明〗该函数可重入。

umask

〖功能〗该函数将当前进程的文件创建屏蔽设置为 mask，并且返回文件创建屏蔽的原值。

〖原型〗mode_t umask (mode_t mask)

〖位置〗sys/stat.h (POSIX.1)

〖说明〗以下样例说明如何使用 umask 读取屏蔽，同时不做永久性改变。

```
mode_t
read_umask (void)
{
  mask = umask (0);
  umask (mask);
}
```

但是，如果只需要读入屏蔽值，建议使用 getumask，因为该函数可重入(至少在使用 GNU 操作系统时是这样)。

unlink

〖功能〗删除文件名 filename，如果这是一个文件惟一的名字，也删除该文件。(如果此时任何一个进程打开了该文件，删除会延迟到所有的进程关闭该文件)。

〖原型〗int unlink (const char *filename)

〖位置〗unistd.h (POSIX.1)

〖说明〗unlink 函数在头文件 unistd.h 中声明。

该函数成功完成时返回 0，否则返回-1。除通常的文件名错误之外，以下是为该函数定义的 errno 错误代码。

EACCES	被删除文件所处的目录拒绝写入，或者目录已有相应设置且用户不拥有该文件。
EBUSY	该错误说明系统使用文件的方式导致文件无法删除。例如，当文件名指定根目录时会出现这一错误。
ENOENT	删除的文件名不存在。
EPERM	在有些系统中，unlink 不能用来删除目录名称，或者只有特许用户才能这样做。为了避免发生这类问题，请使用 rmdir 删除目录(在 GNU 系统中，unlink 永远不能删除目录名)。
EROFS	删除文件所在的目录位于只读文件系统中，不能更改。

utime

〖功能〗该函数用来修改与 filename 文件相关联的文件时间。

〖原型〗int utime (const char *filename, const struct utimbuf *times)

〖位置〗time.h (POSIX.1)

〖说明〗如果 times 是一个空指针，那么文件的访问时间和更改时间都设置为当前时间。否则设置为 times 指向的 utimbuf 结构中的 actime 和 modtime 成员的值。

无论在哪一种情况下，文件的更改时间属性都将被设置为当前时间(因为修改时间戳本身就是修改文件属性)。

该函数调用成功返回 0，失败则返回-1。除了常见的文件名称错误之外，该函数还定义了如下 errno 错误代码。

EACCES	当 time 参数传递一个空指针时，出现了权限问题。要更新文件的时间戳，必须是文件所有者、对文件具有写权限、或者是特许用户。
ENOENT	文件不存在。
EPERM	如果 times 参数不是空指针，则必须是文件所有者或者特许用户。
EROFS	文件驻留在只读文件系统中。

3 个时间戳都有各自相应的微秒部分来提高精度。3 个字段称为 st_atime_usec、st_mtime_usec 和 st_ctime_usec，每个字段的数值范围从 0 到 999 999，以微秒为单位表示时间。它们对应于 timeval 结构中的 tv_usec 字段。

utimes

〖功能〗该函数与 utime 类似，但是允许指定文件时间格式的小数部分。

〖原型〗int utimes (const char *filename, struct timeval tvp[2])

〖位置〗sys/time.h (BSD)

〖说明〗该函数设置 filename 文件的访问和更改时间。新的文件访问时间由 tvp[0]指定，新的文件更改时间由 tvp[1]指定，该函数源自 BSD。

函数的返回值和错误情况与 utime 函数相同。

1.8 管道和队列

mkfifo

〖功能〗该函数创建一个特殊的 FIFO 文件 filename。

〖原型〗int mkfifo (const char *filename, mode_t mode)

〖位置〗sys/stat.h (POSIX.1)

〖说明〗参数 mode 用来设置文件的权限。

该函数成功时返回值为 0，如果出现错误则返回-1。除了常见的文件名错误之外，该

函数还定义了如下 errno 错误代码。

EEXIT	同名文件已经存在。
ENOSPC	目录或者文件系统无法扩展。
EROFS	保存文件的目录驻留在只读文件系统中。

pclose

〖功能〗该函数用来关闭由 popen 函数创建的流。

〖原型〗int pclose (FILE *stream)

〖位置〗stdio.h (POSIX.2, SVID, BSD)

〖说明〗函数将等待子进程来中断执行，并返回其状态值，与 system 函数类似。

以下样例说明如何使用 popen 和 pclose 这两个函数如何使用其他程序过滤输出，这里使用的是页面调用程序 more。

```
#include <stdio.h>
#include <stdlib.h>
void
write_data (FILE * stream)
{
  int i;
  for (i = 0; i < 100; i++)
    fprintf (stream, "%d\n", i);
  if (ferror (stream))
    {
      fprintf (stderr, "Output to stream failed.\n");
      exit (EXIT_FAILURE);
    }
}
int
main (void)
{
  FILE *output;
  output = popen ("more", "w");
  if (!output)
    {
      fprintf (stderr, "Could not run more.\n");
      return EXIT_FAILURE;
    }
  write_data (output);
  pclose (output);
  return EXIT_SUCCESS;
}
```

pipe

〖功能〗该函数创建一个管道，并且将读写管道的文件描述符分别存放在 filedes[0]和

filedes[1]中。

〖原型〗int pipe (int filedes[2])

〖位置〗unistd.h (POSIX.1)

〖说明〗两上参数之中的前者为输入端，要记住这一点有一个简便的方法，就是文件描述符 0 是标准输入，而文件描述符 1 是标准输出。

如果执行成功，则函数返回 0；如果失败则返回-1。该函数还定义了如下错误代码。

EMFILE	进程打开的文件过多。
ENFILE	整个系统中打开的文件过多，该错误不可能出现在 GNU 系统中。

下面有一个简单的创建管道程序。该程序使用 fork 函数创建子进程，父进程负责将数据写入管道，由子进程负责读取。

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
/* Read characters from the pipe and echo them to stdout. */
void
read_from_pipe (int file)
{
  FILE *stream;
  int c;
  stream = fdopen (file, "r");
  while ((c = fgetc (stream)) != EOF)
    putchar (c);
  fclose (stream);
}
/* Write some random text to the pipe. */
void
write_to_pipe (int file)
{
  FILE *stream;
  stream = fdopen (file, "w");
  fprintf (stream, "hello, world!\n");
  fprintf (stream, "goodbye, world!\n");
  fclose (stream);
}
int
main (void)
{
  pid_t pid;
  int mypipe[2];
  /* Create the pipe. */
  if (pipe (mypipe))
    {
      fprintf (stderr, "Pipe failed.\n");
```

```
        return EXIT_FAILURE;
      }
    /* Create the child process. */
    pid = fork ();
    if (pid == (pid_t) 0)
      {
        /* This is the child process. */
        read_from_pipe (mypipe[0]);
        return EXIT_SUCCESS;
      }
    else if (pid < (pid_t) 0)
      {
        /* The fork failed. */
        fprintf (stderr, "Fork failed.\n");
        return EXIT_FAILURE;
      }
    else
      {
        /* This is the parent process. */
        write_to_pipe (mypipe[1]);
        return EXIT_SUCCESS;
      }
  }
```

popen

〖功能〗该函数与 system 函数联系紧密，它将命令行 command 作为子进程执行。但是该函数不会等待命令完成，而是创建一个指向子进程的管道，并返回与管道相对应的流。

〖原型〗FILE * popen (const char *command, const char *mode)

〖位置〗stdio.h (POSIX.2, SVID, BSD)

〖说明〗如果指定 mode 参数为 r，则可以读取流，得到子进程标准输出频道中的数据。子进程从父进程那里继承标准的输入频道。

同样，如果指定 mode 参数为 w，则可以写入流，将数据发送给子进程的标准输入频道。子进程从父进程那里继承标准的输出频道。

一旦出现错误，则 popen 返回一个空指针。如果管道或者流无法创建，子进程无法得到，或者程序无法执行，则可能出现这种情况。

1.9　套 接 口

accept

〖功能〗该函数用来在服务器套接口 socket 上接受连接请求。

〖原型〗int accept (int socket, struct sockaddr *addr, socklen_t *length-ptr)

〖位置〗sys/socket.h (BSD)

〖说明〗如果没有新的连接，则 accept 函数将一直等待，直到套接口 socket 设置为非阻塞模式为止(可以使用 select 函数在非阻塞套接口中等待新连接)。

参数 addr 和 length-ptr 用来返回初始化连接的客户端套接口名称的有关信息。

接受连接之后，socket 并不成为连接的一部分，而是创建一个新的套接口完成连接。accept 的正常返回值就是这个新套接口的文件描述符。

执行 accept 之后，原有的套接口 socket 继续打开并且保持未连接的状态，并在被关闭之前始终负责监听。再次调用 accept 可以接受其他连接。

如果出现错误 accept 将返回-1。该函数定义的 errno 错误代码如下所示。

EBADF	socket 参数为无效的文件描述符。
ENOTSOCK	描述符参数 socket 不是一个套接口。
EOPNOTSUPP	描述符 socket 不支持该操作。
EWOULDBLOCK	socket 设置为非阻塞模式，并且目前没有立即可用的未定连接。

使用无连接通信方式的套接口不允许使用 accpet 函数。

bind

〖功能〗该函数为套接口 socket 分配地址。

〖原型〗int bind (int socket, struct sockaddr *addr, socklen_t length)

〖位置〗sys/socket.h (BSD)

〖说明〗addr 和 length 参数用来说明地址，地址的详细格式根据 NAMESPACE 而定。地址的第 1 部分通常是说明 NAMESPACE 的格式指示符，表明地址按照该 NAMESPACE 的格式给出。

函数成功返回 0，失败返回-1。该函数定义的 ERRNO 错误代码如下。

EBADF	参数 socket 不是一个有效的文件描述符。
ENOTSOCK	描述符 socket 不是一个套接口。
EADDRNOTAVAIL	本机的指定地址不可用。
EADDRINUSE	其他套接口正在使用指定地址。
EINVAL	套接口 socket 已经有一个地址。
EACCES	没有权限访问请求的地址(在 Internet 中，只有超级用户才允许指定从 0 到 IPPORT_RESERVED-1 范围内的端口号)。

根据套接口的 NAMESPACE 不同还可能出现其他的不同情况。

connect

〖功能〗该函数用于初始化连接。

〖原型〗int connect (int socket, struct sockaddr *addr, socklen_t length)

〖位置〗sys/socket.h (BSD)

〖说明〗该函数在使用文件描述符 socket 指定的套接口与使用 addr 和 length 参数指定地址的套接口之间初始化一个连接(通常后者位于另外一台机器上，并且已经被设置成一台服务器)。

通常 connect 函数将等待服务器响应请求之后才返回。可以在套接口 socket 上设置非阻塞模式，使得 connect 函数可以无需等待响应就立即返回。

connect 的正常返回值为 0，如果出现错误则返回-1。该函数定义了如下 errno 错误代码。

EBADF　套接口 socket 不是一个有效的文件描述符。

endhostent

〖功能〗该函数用于关闭主机数据库。
〖原型〗void endhostent ()
〖位置〗netdb.h (BSD)

endnetent

〖功能〗该函数用于关闭网络数据库。
〖原型〗void endnetent (void)
〖位置〗netdb.h (BSD)

endprotoent

〖功能〗该函数用于关闭协议数据库。
〖原型〗void endprotoent (void)
〖位置〗netdb.h (BSD)

endservent

〖功能〗该函数用于关闭服务数据库。
〖原型〗void endservent (void)
〖位置〗netdb.h (BSD)

gethostbyaddr

〖功能〗该函数用于返回 Internet 地址为 addr 的主机信息。
〖原型〗struct hostent * gethostbyaddr (const char *addr, int length, int format)
〖位置〗netdb.h (BSD)
〖说明〗参数 length 为 addr 的地址长短(以字节为单位)。参数 format 指定地址格式，对于 Internet 地址则还需要指定 AF_INET 的值。

如果查询失败，gethostbyaddr 将返回一个空指针。

检查变量 h_errno 的数值可以找出 gethostbyname 参数或者 gethostbyaddr 参数查询失败的原因(设置 errno 可能会更加清楚明了，但是使用 h_errno 可以与其他系统兼容)。在使用 h_errno 之前，必须使用如下语句声明。

```
extern int h_errno;
```

h_errno 中的错误代码如下。

HOST_NOT_FOUND	数据库中没有这样的主机。
TRY_AGAIN	在无法联络到名称服务程序时将出现这一情况。如果再试可能会成功。
NO_RECOVERY	出现一个无法恢复的错误。
NO_ADDRESS	主机数据库中有这个名称项，但是没有相关联的 Internet 地址。

也可以同时使用 sethostent、gethostent 和 endhostenv 函数搜索整个主机数据库。但是使用这些函数时必须小心，因为它们都是不可重入的。

gethostbyname

〖功能〗该函数返回名为 name 的主机的有关信息。如果查询失败，则返回一个空指针。

〖原型〗struct hostent * gethostbyname (const char *name)

〖位置〗netdb.h (BSD)

gethostbyname2

〖功能〗该函数与 gethostbyname 类似，但是它允许调用方为返回结果指定地址家族(例如 AF_INET 或者 AF_INET6)。

〖原型〗struct hostent * gethostbyname2 (const char *name, int af)

〖位置〗netdb.h (IPv6 Basic API)

gethostent

〖功能〗该函数返回主机数据库中的下一项。

〖原型〗struct hostent * gethostent ()

〖位置〗netdb.h (BSD)

〖说明〗如果不存在下一项，则返回一个空指针。

getnetbyaddr

〖功能〗该函数返回 type 类型并且网络号为 net 的网络信息。

〖原型〗struct netent * getnetbyaddr (long net, int type)

〖位置〗netdb.h (BSD)

〖说明〗对于互联网络而言，需要指定参数 type 为数值 AF_INET。

如果该网络不存在，则 getnetbyaddr 返回一个空指针。

也可以使用 setnetent、getnetent 和 endnetent 函数扫描网络数据库。使用这些函数时应小心，它们都是不可重入的。

getnetbyname

〖功能〗该函数返回名为 name 的网络的有关信息。
〖原型〗struct netent * getnetbyname (const char *name)
〖位置〗netdb.h (BSD)
〖说明〗如果该网络不存在，函数则返回一个空指针。

getnetent

〖功能〗该函数返回网络数据库中的下一项。
〖原型〗struct netent * getnetent (void)
〖位置〗netdb.h (BSD)
〖说明〗如果没有下一项，函数则返回空指针。

getpeername

〖功能〗该函数返回连接在 socket 上的套接口地址。
〖原型〗int getpeername (int socket, struct sockaddr *addr, size_t *length-ptr)
〖位置〗sys/socket.h (BSD)
〖说明〗在 addr 和 length-ptr 两个参数指定的内存空间中保存地址。地址长度则保存在*length-ptr 中。
在某些操作系统中，函数 getpeername 仅仅适用于互联网中的套接口。
函数执行成功返回 0，失败则返回-1。该函数定义了如下 errno 错误代码。

EBADF	参数 socket 不是有效的文件描述符。
ENOTSOCK	描述符 socket 不是一个套接口。
ENOTCONN	套接口 socket 没有连接。
ENOBUFS	没有足够的内部缓存以供使用。

getprotobyname

〖功能〗该函数返回名为 name 的网络协议的有关信息。
〖原型〗struct protoent * getprotobyname (const char *name)
〖位置〗netdb.h (BSD)
〖说明〗如果该协议不存在，则函数返回一个空指针。

getprotobynumber

〖功能〗该函数返回号码为 protocol 的网络协议的有关信息。
〖原型〗struct protoent * getprotobynumber (int protocol)
〖位置〗netdb.h (BSD)
〖说明〗如果该协议不存在，则函数返回一个空指针。

getprotoent

〖功能〗该函数返回协议数据库中的下一项。
〖原型〗struct protoent * getprotoent (void)
〖位置〗netdb.h (BSD)
〖说明〗如果没有下一项，则函数返回一个空指针。

getservbyname

〖功能〗该函数返回使用 proto 协议，并且名为 name 的服务信息。
〖原型〗struct servent * getservbyname (const char *name, const char *proto)
〖位置〗netdb.h (BSD)
〖说明〗如果无法找到这样的服务，则函数返回一个空指针。
该函数同时适用于服务器和客户端，服务器可以用它来确定应当监听哪一个端口。

getservbyport

〖功能〗该函数返回在 port 端口中使用协议 proto 的服务信息。
〖原型〗struct servent * getservbyport (int port, const char *proto)
〖位置〗netdb.h (BSD)
〖说明〗如果无法找到这样的服务，则函数返回一个空指针。

getservent

〖功能〗该函数返回服务数据库中的下一项。
〖原型〗struct servent * getservent (void)
〖位置〗netdb.h (BSD)
〖说明〗如果没有下一项，则函数返回一个空指针。

getsockname

〖功能〗该函数返回套接口 socket 的地址信息，其地址由 addr 和 length-ptr 参数指定。
〖原型〗int getsockname (int socket, struct sockaddr *addr, socklen_t *length-ptr)
〖位置〗sys/socket.h (BSD)
〖说明〗注意，参数 length-ptr 是一个指针，应当进行初始化并根据参数 addr 分配大小，返回结果中包含地址数据的实际大小。

地址数据的格式根据套接口的名称空间而定。每个名称空间的信息长度通常是不变的，因此可以准确地知道需要多少空间以及可以提供多少空间。通常的做法是使用恰当的数据类型分配空间，然后将其地址放在 struct sockaddr *中，并传递给 getsockname。

函数成功返回 0，错误则返回-1。该函数定义了如下 errno 错误代码。

EBADF	socket 参数是无效的文件描述符。
ENOTSOCK	描述符 socket 不是一个套接口。

ENOBUFS	没有足够的内部缓存，无法完成操作。

在文件名称空间中无法读取套接口的地址。这一点与系统的其他部分一致，即通常是不能从某个文件的描述符中找到文件名称的。

getsockopt

〖功能〗该函数用于得到套接口 socket 在 level 级别上 optname 选项的值。

〖原型〗int getsockopt (int socket, int level, int optname, void *optval, socklen_t *optlen-ptr)

〖位置〗sys/socket.h (BSD)

〖说明〗选项值保存在 optval 指向的缓存中。在调用之前，应当在*optlen-ptr 中提供该缓存区的大小，返回后，相应位置中的数值表示缓存区中实际保存的信息的字节数。

多数选项都将 optval 缓存区解释成一个单一的 int 值。

getsockopt 函数成功返回 0，失败返回-1。定义的 errno 错误代码如下。

EBADF	socket 参数是无效的文件描述符。
ENOTSOCK	描述符 socket 不是一个套接口。
ENOPROTOOPT	optname 在指定的 level 级别上没有意义。

htonl

〖功能〗该函数将长整数类型的参数 hostlong 从主机字节顺序转换为网络字节顺序。

〖原型〗unsigned long int htonl (unsigned long int hostlong)

〖位置〗netinet/in.h (BSD)

htons

〖功能〗该函数将短整数类型的参数 hostshort 从主机字节顺序转换为网络字节顺序。

〖原型〗unsigned short int htons (unsigned short int hostshort)

〖位置〗netinet/in.h (BSD)

inet_addr

〖功能〗该函数将互联网主机地址 name 从标准的数字和点组成的标记符号转换为二进制数据。

〖原型〗unsigned long int inet_addr (const char *name)

〖位置〗arpa/inet.h (BSD)

〖说明〗如果输入无效，则 inet_addr 函数返回 INADDR_NONE。这是与 inet_aton 之间已经废弃的旧接口，因为 INADDR_NONE 本身就是一个有效地址(255.255.255.255)，并且 inet_aton 提供了一种更加明确的方法来说明返回错误。

inet_aton

〖功能〗该函数将互联网主机地址 name 从标准的数字和点组成的符号标记转换为二进制数据，并且将结果保存在 addr 指向的结构 struct in_addr 中。

〖原型〗int inet_aton (const char *name, struct in_addr *addr)

〖位置〗arpa/inet.h (BSD)

〖说明〗如果地址有效则 inet_aton 返回非 0 值，否则返回 0。

inet_lnaof

〖功能〗该函数返回互联网主机地址 addr 中的网内本地地址部分。

〖原型〗int inet_lnaof (struct in_addr addr)

〖位置〗arpa/inet.h (BSD)

inet_makeaddr

〖功能〗该函数将网络号 net 和网段内本地地址号 local 合并成为一个互联网主机地址。

〖原型〗struct in_addr inet_makeaddr (int net, int local)

〖位置〗arpa/inet.h (BSD)

inet_netof

〖功能〗该函数返回互联网主机地址 addr 的网络号部分。

〖原型〗int inet_netof (struct in_addr addr)

〖位置〗arpa/inet.h (BSD)

inet_network

〖功能〗该函数从地址 name 中分解出网络号，并且按照标准的数字和点组成的符号标记格式返回。

〖原型〗unsigned long int inet_network (const char *name)

〖位置〗arpa/inet.h (BSD)

〖说明〗如果输入无效，则 inet_network 返回-1

inet_ntoa

〖功能〗该函数将互联网主机地址 addr 转换为字符串，字符串格式为标准的数字和点组成的标记符号。

〖原型〗char * inet_ntoa (struct in_addr addr)

〖位置〗arpa/inet.h (BSD)

〖说明〗返回一个指针，指向静态分配的缓存。下一次调用该函数将覆盖同一缓存区，因此如果需要保存其内容就必须拷贝字符串。

在多线程的程序中，每个线程都拥有自己的静态分配缓存。但是在同一个线程中多次调用 inet_ntoa 仍然将覆盖上一次调用的结果。

inet_ntop

〖功能〗该函数将互联网地址(IPv4 或者 IPv6 格式)从网络格式(二进制)转换为陈述格式(文本)。

〖原型〗char * inet_ntop (int af, const void *cp, char *buf, size_t len)

〖位置〗arpa/inet.h (IPv6 basic API)

〖说明〗af 可以是 AF_INET 或者 AF_INET6，根据需要进行选择。cp 是指向被转换地址的指针，buf 则是指向结果缓存的指针，len 表示该缓存区的长度。该函数的返回值应当是缓存区地址。

inet_pton

〖功能〗该函数将互联网地址(IPv4 或者 IPv6 格式)从陈述格式(文本)转换为网络格式(二进制)。

〖原型〗int inet_pton (int af, const char *cp, void *buf)

〖位置〗arpa/inet.h (IPv6 basic API)

〖说明〗af 可以是 AF_INET 或者 AF_INET6，与转换地址的类型相适应。cp 是指向输入字符串的指针，buf 则是指向结果缓存的指针。调用方应当负责确保缓存的空间足够大。

listen

〖功能〗该函数令套接口 socket 接受连接，从而成为服务器套接口。

〖原型〗int listen (int socket, unsigned int n)

〖位置〗sys/socket.h (BSD)

〖说明〗参数 n 指定等待连接的队列长度。当队列排满时，在服务器调用 accept 接受队列中的下一个连接之前，新客户端将出现 ECONNREFUSED 连接失败。

调用成功则函数返回 0，失败返回-1。该函数定义的 errno 错误代码如下。

EBADF	参数 socket 为无效的文件描述符。
ENOTSOCK	参数 socket 不是套接口。
EOPNOTSUPP	套接口 socket 不支持该操作。

ntohl

〖功能〗该函数将长整数类型的 netlong 参数从网络字节顺序转换为主机字节顺序。

〖原型〗unsigned long int ntohl (unsigned long int netlong)

〖位置〗netinet/in.h (BSD)

ntohs

〖功能〗该函数将短整数类型的 netshort 参数从网络字节顺序转换为主机字节顺序。

〖原型〗unsigned short int ntohs (unsigned short int netshort)

〖位置〗netinet/in.h (BSD)

recvfrom

〖功能〗该函数从套接口 socket 中读取一个数据包，并保存在缓存区 buffer 中。

〖原型〗int recvfrom (int socket, void *buffer, size_t size, int flags, structsockaddr *addr, socklen_t *length-ptr)

〖位置〗sys/socket.h (BSD)

〖说明〗参数 size 指定可以读取的最大字节数。

如果数据包的长度大于 size 个字节，则读取前 size 个字节，而剩余字节将丢失，并且无法读取。因此使用数据包协议时，必须知道请求的数据包有多长。

参数 addr 和 length-ptr 是用来返回包的源地址的。对于位于文件域中的套接口，其地址信息是没有意义的，因为这类套接口地址是无法读取的。如果对这一信息没有兴趣，可以指定 addr 参数为一个空指针。

flags 的解释方法与 recv 函数相同，返回值和错误情况也与 recv 相同。

如果不需要了解发送包的源地址(可能已经知道其源地址或者对待所有的发送方都一样处理)，该函数就可以使用 recv 代替。如果不需要指定 flags，甚至也可以使用 read 代替该函数。

recv

〖功能〗该函数与 read 函数类似，但是使用补充标志位 flags。

〖原型〗int recvfrom (int socket, void *buffer, size_t size, int flags)

〖位置〗sys/socket.h (BSD)

〖说明〗如果 socket 设置为非阻塞模式，并且没有可供读入的数据，则 recv 函数将立即退出执行，而不会等待。

该函数将返回接收的字节数，如果失败则返回-1。定义的 errn 错误代码如下。

EBADF	参数 socket 是无效的文件描述符。
ENOTSOCK	描述符 Socket 不是套接口。
EWOULDBLOCK	套接口设置为非阻塞模式，而且续操作可能阻塞通常 recv 将保持阻塞状态，直至有可供续取的输入为止。
EINTR	在续入数据之前，操作被某个信号中断。
ENOTCONN	该套接口没有连接。

send

〖功能〗该函数与 write 函数类似，但是存在一个额外的标志位 flags。

〖原型〗int send (int socket, void *buffer, size_t size, int flags)

〖位置〗sys/socket.h (BSD)

〖说明〗该函数返回传输的字节数，失败时的返回值为-1。如果套接口没有阻塞，则

send 函数(如同 write)能在仅发送部分数据后返回。

但是需要注意的是，一个成功的返回值仅仅表明信息发送时没有出错，不能说明接收也没有错误。

该函数定义的 errno 错误代码如下。

EBADF	socket 参数不是一个有效的文件描述符。
EINTR	在没有发送任何数据之前，一个信号中断了这次操作。
ENOTSOCK	描述符 socket 不是套接口。
EMSGSIZE	套接口的类型要求发送很小的信息，但是信息超出了允许的大小范围。
EWOULDBLOCK	套接口设置了 nonblocking 模式，但是写操作可能被阻塞(通常在操作执行完毕之后 send 才会阻塞)。
ENOBUFS	内部可用的缓存空间不足。
ENOTCONN	没有连接到该套接口。
EPIPE	该套接口被连接，但是目前连接已中断。这种情况下，send 函数首先产生一个 SIGPIPE 信号，如果该信号被忽略或者阻塞，又或者其句柄被返回，则 send 函数失败，并报错 EPIPE。

sendto

〖功能〗该函数通过 socket 套接口把 buffer 中的数据传送到由 addr 和 length 参数指定的目的地址中。

〖原型〗int sendto (int socket, void *buffer. size_t size, int flags, struct sockaddr *addr, socklen_t length)

〖位置〗sys/socket.h (BSD)

〖说明〗参数 size 指定传输的字节数。

flags 的解释方法与 send 函数相同。

该函数的返回值和错误情况也和 send 相同，不能依靠系统监测并报告错误。最常见的错误是信息包丢失或者是在指定的地址上没有程序负责接收数据，并且机器上的操作系统通常不知道这一情况。

sendto 函数也可以用来报告一个前次调用所产生的错误。

sethostent

〖功能〗该函数打开主机数据库并且开始扫描其内容。随后就可调用 gethostent 函数读取其中的项目内容。

〖原型〗void sethostent (int stayopen)

〖位置〗netdb.h (BSD)

〖说明〗如果参数 stayopen 非 0，则设置标识位，随后调用的 gethostbyname 函数和 gethostbyaddr 函数将不会关闭数据库(像通常情况一样)。应当尽量避免每次调用时不停地重复打开数据库，这样可以提高这些函数被多次调用时的效率。

setnetent

〖功能〗该函数打开并重绕网络数据库。

〖原型〗void setnetent (int stayopen)

〖位置〗netdb.h (BSD)

〖说明〗如果参数 stayopen 非 0，则设置标识位，随后调用 getnetbyname 和 getnetbyaddr 函数就不会关闭数据库(像通常的情况一样)。如果避免每次调用都重复打开数据库，那么就可以提高多次调用时的运行效率。

setprotoent

〖功能〗该函数打开协议数据库，然后开始扫描。

〖原型〗void setprotoent (int stayopen)

〖位置〗netdb.h (BSD)

〖说明〗如果参数 stayopen 非 0，则设置标识位，随后调用的 getprotobyname 和 getprotobynumber 函数就不会关闭数据库(像通常的情况那样)。如果避免每次调用都重复打开数据库，那么就可以提高多次调用时的运行效率。

setservent

〖功能〗该函数打开服务数据库，然后开始扫描其内容。

〖原型〗void setservent (int stayopen)

〖位置〗netdb.h (BSD)

〖说明〗如果参数 stayopen 非 0，则设置标识位，随后调用的 getservbyname 和 getservbyport 函数就不会关闭数据库。如果避免每次调用都重复打开数据库，就可以提高多次调用时的运行效率。

setsockopt

〖功能〗该函数用于在 level 级别上为 socket 设置套接口选项 optname。选项的值在通过缓存区 optval 传递，缓存区大小由 optlen 确定。

〖原型〗int setsockopt (int socket, int level, int optname, void *optval, socklen_t optlen)

〖位置〗sys/socket.h (BSD)

〖说明〗该函数的返回值和错误代码和 getsockopt 函数相同。

shutdown

〖功能〗该函数用于关闭套接口 socket 的连接。

〖原型〗int shutdown (int socket, int how)

〖位置〗sys/socket.h (BSD)

〖说明〗参数 how 说明执行的动作。

0	停止从该套接口接收数据。如果到达更多的数据则拒绝接收。

1　　停止试图从该套接口传送数据，丢弃所有正在等待发送的数据，停止寻找已送出数据的确认，即便数据丢失也不再重新发送。

2　　停止接收和发送。

函数成功时返回 0，失败时返回-1。函数定义的 errno 出错情况如下。

EBADF	socket 是无效的文件描述符。
ENOTSOCK	socket 不是套接口。
ENOTCONN	socket 没有连接。

socket

〖功能〗该函数创建一个套接口，并指定通讯样式为 style。

〖原型〗int socket (int namespace, int style, int protocol)

〖位置〗sys/socket.h (BSD)

〖说明〗参数 namespace 指定名称空间，其数值必须为 PF_FILE 或者 PF_INET。protocol 指定协议，使用 0 值通常都是正确的。

函数 socket 的返回值就是新套接口的文件描述符，如果出现错误则返回-1。该函数还定义了如下 errno 错误代码。

EPROTONOSUPPORT	指定的 namespace 不支持 protocol 或者 style。
EMFILE	进程打开的文件描述符过多。
ENFILE	系统打开的文件描述符过多。
EACCESS	进程没有权限创建指定 style 或者 protocol 的套接口。
ENOBUFS	系统的内部缓存空间不足。

socket 函数返回的文件描述符可以同时支持读操作和写操作。但是套接口象管道一样，不支持文件定位操作。

socketpair

〖功能〗该函数创建套接口对，返回 filedes[0] 和 filedes[1]中的文件描述符。

〖原型〗int socketpair (int namespace, int style, int protocol, int filedes[2])

〖位置〗sys/socket.h (BSD)

〖说明〗套接口对是全双工通信信道，因此读写操作可以在任意一端进行。

参数 namespace、style 和 protocol 的解释方法与 socket 函数相同。style 必须是指定使用的通讯样式。参数 namespace 指定名称空间，必须是 AF_FILE，protocol 指定通讯协议，但是只有 0 值有意义。

如果 style 指定无连接的通讯方式，那么得到的两个套接口就不连接，但是都将对方作为默认的目的地址，从而可以相互发送数据包。

socketpair 函数成功返回 0，失败返回-1。该函数还定义了如下 errno 错误代码。

EMFILE	进程打开的文件描述符过多。
EAFNOSUPPORT	不支持指定的名称空间。
EPROTONOSUPPORT	不支持指定的协议。
EOPNOTSUPP	指定的协议不支持创建套接口对。

1.10 底层终端接口

cfgetispeed

〖功能〗函数返回在*TERMIOS-P 结构中保存的输入行速。

〖原型〗speed_t cfgetispeed (const struct termios *termios-p)

〖位置〗termios.h (POSIX.1)

cfgetospeed

〖功能〗函数返回在*TERMIOS-P 结构中保存的输出行速。

〖原型〗speed_t cfgetospeed (const struct termios *termios-p)

〖位置〗termios.h (POSIX.1)

cfmakeraw

〖功能〗函数提供一种简便的方法来建立*TERMIOS-P，也就是 BSD 中所称的“原始模式”。

〖原型〗int cfmakeraw (struct termios *termios-p)

〖位置〗termios.h (BSD)

〖说明〗函数使用非经典输入，并且关闭大部分的处理工作，为终端提供一个未更改的频道。

实际工作如下。

```
termios-p->c_iflag &= ~(IGNBRK|BRKINT|PARMRK|ISTRIP
                        |INLCR|IGNCR|ICRNL|IXON);
termios-p->c_oflag &= ~OPOST;
termios-p->c_lflag &= ~(ECHO|ECHONL|ICANON|ISIG|IEXTEN);
termios-p->c_cflag &= ~(CSIZE|PARENB);
termios-p->c_cflag |= CS8;
```

cfsetispeed

〖功能〗函数设置输入速度。

〖原型〗int cfsetispeed (struct termios *termios-p, speed_t speed)

〖位置〗termios.h (POSIX.1)

〖说明〗将*TERMIOS-P 中的 speed 值保存为输入速度。正常返回值为 0，返回值-1

表明出现错误。如果 speed 不是速度值，则 cfsetspeed 返回-1。

cfsetospeed

〖功能〗函数设置输出速度。

〖原型〗int cfsetospeed (struct termios *termios-p, speed_t speed)

〖位置〗termios.h (POSIX.1)

〖说明〗将*termios-p 中的 speed 值保存为输出速度。正常返回值为 0，返回值-1 表明出现错误。如果 speed 不是速度值，则 cfsetspeed 返回-1。

cfsetspeed

〖功能〗函数设置输入和输出速度。

〖原型〗int cfsetspeed (struct termios *termios-p, speed_t speed)

〖位置〗termios.h (BSD)

〖说明〗将*termios-p 中的 speed 值保存为输入和输出速度。正常返回值为 0，返回值-1 表明出现错误。如果 speed 不是速度值，则 cfsetspeed 返回-1。该函数是 4.4 BSD 扩展后包含的内容。

isatty

〖功能〗如果 filedes 表示的文件描述符与开放终端设备相关联，则函数返回 1，否则返回 0。

〖原型〗int isatty (int filedes)

〖位置〗unistd.h (POSIX.1)

〖说明〗如果文件描述符与终端相关联，则使用 ttyname 函数就可以得到与之相关联的文件名称。

tcdrain

〖功能〗该函数在所有指向终端 filedes 的输出队列传递完毕之前，始终等待。

〖原型〗int tcdrain (int filedes)

〖位置〗termios.h (POSIX.1)

〖说明〗返回值通常为 0。出错时返回-1。以下是该函数定义的 errno 出错情况。

EBADF	filedes 不是有效的文件描述符。
ENOTTY	filedes 未和终端设备关联。
EINTR	某个信号的传递中断了操作。

tcflow

〖功能〗该函数用于在 filedes 指定的终端文件上执行与 XON/XOFF 流控制相关联的操作。

〖原型〗int tcflow (int filedes, int action)

〖位置〗termios.h (POSIX.1)
〖说明〗参数 action 指定执行的操作，可以使用以下数值。

TCOOFF	延缓输出。
TCOON	重新开始输出。
TCIOFF	传送 STOP 字符。
TCION	传送 START 字符。

通常返回 0，出错时返回-1，以下是定义的 errno 错误代码。

EBADF	filedes 不是有效的文件描述符。
ENOTTY	filedes 未和终端设备关联。
EINVAL	action 参数提供了一个有害的值。

tcflush

〖功能〗该函数用于清除与终端文件 filedes 相关的输入和/或输出队列。
〖原型〗int tcflush (int filedes, int queue)
〖位置〗termios.h (POSIX.1)
〖说明〗参数 queue 指定清除哪些队列，它可以使用下列数值。

TCIFLUSH	清除收到的输入数据，但不再读取。
TCOFLUSH	清除输出数据，但不再发送。
TCIOFLUSH	清除输入和输出队列中的数据。

函数返回值通常为 0，出错时返回值为-1，以下是定义的 errno 错误代码。

EBADF	filedes 不是有效的文件描述符。
ENOTTY	filedes 未与终端设备关联。
EINVAL	queue 参数提供了一个有害的值。

tcgetattr

〖功能〗该函数用于检查文件描述符 filedes 指定的终端设备的属性。属性值在 termios-p 指向的结构中返回。
〖原型〗int tcgetattr (int filedes, struct termios *termios-p)
〖位置〗termios.h (POSIX.1)
〖说明〗成功时 tcgetattr 函数返回 0，返回-1 表示函数出错，以下是定义的 errno 错误代码。

EBADF	filedes 不是有效的文件描述符。

ENOTTY　filedes 没有与终端设备相关联。

tcsendbreak

〖功能〗该函数在与文件描述符 filedes 相关的终端上通过传送 0 位的流产生中断。中断持续时间由参数 duration 控制。如果参数值是 0，持续时间就在 0.25 到 0.5 秒之间。非 0 值的意义则取决于操作系统。

〖原型〗int tcsendbreak (int filedes, int duration)

〖位置〗termios.h (POSIX.1)

〖说明〗如果终端不是异步串行数据端口，则函数不作任何动作。

返回值通常为 0，出错时返回值为-1，该函数定义的 errno 错误代码如下。

EBADF　filedes 不是有效的文件描述符。

ENOTTY　filedes 没有与终端设备相关联。

tcsetattr

〖功能〗该函数使用文件描述符 filedes 设置终端设备的属性。新的属性从 termios-p 指向的结构中获取。

〖原型〗int tcsetattr (int filedes, int when, const struct termios *termios-p)

〖位置〗termios.h (POSIX.1)

〖说明〗参数 when 说明如何处理已经排队等待的输入和输出，它可以是以下值。

TCSCANOW　立即改变属性。

TCSADRAIN　直到所有的输出都写完之后再改变属性。当改变的参数影响输出时常用到该选项。

TCSAFLUSH　和 TCSADRAIN 类似，但同时还丢弃队列中的输入。

TCSASOFT　这是一个标志位，可以加到上述任意选项中。表示可以抑制终端硬件的状态转换。这是一个 BSD 扩展名。只有 BSD 系统和 GNU 系统支持。使用 TCSASOFT 的效果和设置 termios-p 所指结构中 c_cflag 成员的 CIGNORE 位的效果完全一样。

如果从控制终端上的一个后台进程调用该函数，通常将发给进程组中的所有进程一个 SIGTTOU 信号，与进程试图写入终端一样。除非调用进程本身忽略或阻塞 SIGTTOU 信号，此时执行操作并不发出任何信号。

如果执行成功，tcsetattr 函数返回 0。返回值-1 表示错误。以下是定义的 errno 错误代码。

EBADF　filedes 不是有效的文件描述符。

ENOTTY　filedes 未与终端关联。

EINVAL　When 参数值无效或 termios-p 中的数据有错。

尽管 tcgetattr 和 tcsetattr 使用文件指示符指定终端设备，但属性是终端设备本身的而不是文件描述符的。这意味着改变终端属性是持久稳定的；如果另一进程稍后打开终端文件，将会看到改变了的属性，尽管它对曾在改变属性中指定的文件描述符未作任何处理。

同样，如果一个单独的进程对相同的终端设备拥有多重的文件描述符，改变终端属性将影响到所有这些文件描述符。例如，如果在正常的缓存和回显模式下打开一个文件描述符或流，就不能同时在同一终端上使用另一文件描述符按照单字符和非回显模式读入。终端只能在两种模式间来回转换。

ttyname

〖功能〗如果文件描述符 filedes 与终端设备相关，ttyname 函数将返回一个指针，指向一个静态分配的、非空的、包含终端文件的文件名串。

〖原型〗char * ttyname (int filedes)

〖位置〗unistd.h (POSIX.1)

〖说明〗如果文件描述符未和终端相关联或文件名不能确定，则返回空指针。

1.11 数 学 函 数

acos

〖功能〗反余弦函数。

〖原型〗double acos (double x)

〖位置〗math.h (ISO)

〖说明〗该函数计算 x 的反余弦，函数值的单位是弧度。从数学角度看，该值有无穷多个；实际返回的函数值在 0 到 π (包括)之间。

如果 x 超出范围，acos 函数将失败，并将 errno 设置为 EDOM。反余弦函数在数学上仅在域-1 到 1 之间有定义。

acosh

〖功能〗反双曲余弦函数。

〖原型〗double acosh (double x)

〖位置〗math.h (BSD)

〖说明〗该函数返回 x 的反双曲余弦值，即该数值的双曲余弦等于 x。如果 x 的绝对值大于等于 1，则返回 HUGE_VAL。

asinh

〖功能〗反双曲正弦函数。

〖原型〗double asinh (double x)

〖位置〗math.h (BSD)

asinh

〖功能〗反双曲正弦函数。
〖原型〗double asinh (double x)
〖位置〗math.h (BSD)

atan

〖功能〗反正切函数。
〖原型〗double atan (,double x)
〖位置〗math.h (ISO)该函数计算的反正切值，也就是说，返回值的正切函数值等于 x。
〖说明〗返回值的单位是弧度，从数学角度看该值有无穷多个，实际返回值所处的范围从-π/2 到π/2。

atan2

〖功能〗是带有两个参数的反正切函数。
〖原型〗double atan2 (double y, double x)
〖位置〗math.h (ISO)
〖说明〗类似于计算 y/x 的反正切函数值，只不过两个参数的符号是用来判断结果所处的象限，并且 x 允许为 0。返回值的单位是弧度，所处的范围从-π到π。

atanh

〖功能〗反双曲正切函数。
〖原型〗double atanh (double x)
〖位置〗math.h (BSD)
〖说明〗该函数返回 x 的反双曲正切值，即该函数值的双曲正切等于 x。如果 x 的绝对值大于或者等于 1，则返回 HUGE_VAL。

cbrt

〖功能〗立方根函数。
〖原型〗double cbrt (double x)
〖位置〗math.h (BSD)
〖说明〗该函数返回 x 的立方根。函数不会执行失败，因为每个可以表示出来的实数都有一个可以表示的实数立方根。

cosh

〖功能〗返回 x 的双曲余弦值，其数学定义为 exp(x) + exp(-x) /2。如果 x 值过大即溢出，则函数失败，并将 errno 设置为 ERANGE。
〖原型〗double cosh (double x)
〖位置〗math.h (ISO)

cosh

〖功能〗返回 x 的双曲余弦值，其数学定义为 exp(x) + exp(-x) /2。如果 x 值过大即溢出，则函数失败，将 errno 设置为 ERANGE。

〖原型〗double cosh (double x)

〖位置〗math.h (ISO)

exp

〖功能〗该函数返回 e(自然对数的底)的 x 次方。

〖原型〗double exp (double x)

〖位置〗math.h (ISO)

〖说明〗如果由于结果的级数过大无法表示而导致函数失败，则设置 errno 为 ERANGE。

expm1

〖功能〗该函数返回值相当于 exp(x)-1 。

〖原型〗double expm1 (double x)

〖位置〗math.h (BSD)

〖说明〗即便 x 值接近于 0—这种情况下，由于两个数字几乎相等导致 exp(x)-1 的计算可能不够精确—该函数的计算方法仍然可以保证精确。

hypot

〖功能〗该函数返回 sqrt(x*x + y*y)。

〖原型〗double hypot (double x, double y)

〖位置〗math.h (BSD)

〖说明〗返回值就是一个边长为 x 和 y 的直角三角形的斜边长，或者从原点到点(x, y)的距离。

initstate

〖功能〗该函数用来初始化随机数生成器的状态。

〖原型〗void * initstate (unsigned int seed, void *state, size_t size)

〖位置〗stdlib.h (BSD)

〖说明〗参数 state 是一个 size 个字节长的数组，用来保存状态信息。至少 8 个字节，最理想的长度是 8、16、32、64、132 和 256。state 数组越大越好。

返回值为状态信息数组的原有值。可以使用该值作为 setstate 的参数恢复原状态。

log

〖功能〗该函数返回 x 的自然对数值。

〖原型〗double log (double x)

〖位置〗math.h (ISO)
〖说明〗exp (log (x))的返回值与 x 在数学概念上完全相等，在 C 语言中只能近似相等。该函数定义的 errno 错误情况如下。

EDOM	参数 x 为负值。在数学概念上 log 函数的定义是只对正数返回一个实数的结果。
ERANGE	参数 x 为 0，0 的对数没有定义。

log10

〖功能〗该函数返回 x 以 10 为底的对数值。
〖原型〗double log10 (double x)
〖位置〗math.h (ISO)
〖说明〗除了底不同之外，该函数与 log 函数相同。实际上 log10(x)等于 log(x)/log(10)。

log1p

〖功能〗该函数的返回值等于 log(1+x)。
〖原型〗double log1p (double x)
〖位置〗math.h (BSD)
〖说明〗即便 x 的值趋向于 0，该函数的计算方法仍然非常精确。

pow

〖功能〗该函数是一个通用的幂函数，返回 base 的 power 次方。
〖原型〗double pow (double base, double power)
〖位置〗math.h (ISO)
〖说明〗该函数定义了如下 errno 错误代码。

EDOM	参数 base 是一个负数并且 power 不是一个积分值。从数学角度来看，这种情况下的结果将是一个复数。
ERANGE	结果出现向下溢出或者向上溢出的情况。

rand

〖功能〗该函数返回序列中的下一个伪随机数，范围从 0 到 RAND_MAX。
〖原型〗int rand ()
〖位置〗stdlib.h (ISO)

random

〖功能〗该函数返回序列中的下一个伪随机数，范围从 0 到 RAND_MAX。
〖原型〗long int random ()

〖位置〗stdlib.h (BSD)

setstate

〖功能〗该函数用于将随机数的状态信息恢复成 state，参数必须是先前调用 initstate 或 setstate 得到的结果。

〖原型〗void * setstate (void *state)

〖位置〗stdlib.h (BSD)

〖说明〗返回值是状态信息数组以前的值，稍后可以用该值作为 setstate 的参数恢复状态。

sin

〖功能〗返回 x 的正弦值，这里 x 按弧度给出。返回值在-1 到 1 之间。

〖原型〗double sin (double x)

〖位置〗math.h (ISO)

sinh

〖功能〗x 的双曲正弦函数，算术定义为 exp (x) - exp (-x) / 2。当 x 值太大时将出现溢出，则函数失败，并设置 errno 为 ERANGE。

〖原型〗double sinh (double x)

〖位置〗math.h (ISO)

sqrt

〖功能〗返回 x 的非负平方根。

〖原型〗double sqrt (double x)

〖位置〗math.h (ISO)

〖说明〗如果 x 为负数，则函数失败且 errno 设置成 EDOM。平方根可以是复数。

srand

〖功能〗该函数以 seed 作为新的伪随机数序列的种子。

〖原型〗void srand (unsigned int seed)

〖位置〗stdlib.h (ISO)

〖说明〗如果在使用 srand 设定种子之前调用 rand 函数，则使用数值 1 作为默认种子。要产生真正的随机数(而不是伪随机数)，就应当执行 srand(time(0))。

srandom

〖功能〗该函数设定整数 seed 为当前随机数的种子。

〖原型〗void srandom (unsigned int seed)

〖位置〗stdlib.h (BSD)

〖说明〗如果提供的 seed 值为 1，则 random 将得到随机数的默认集合。

要产生真正的随机数(而不是伪随机)，就应当执行 srandom(time(0))。

tan

〖功能〗返回 x 的正切值，x 按弧度给出。
〖原型〗double tan (double x)
〖位置〗math.h (ISO)

tanh

〖功能〗返回 x 的双曲正切值，其数学定义为： sinh (x) / cosh (x)。
〖原型〗double tanh (double x)
〖位置〗math.h (ISO)

1.12　初等数学函数

abs

〖功能〗该函数返回 number 的绝对值。
〖原型〗int abs (int number)
〖位置〗stdlib.h (ISO)
〖说明〗多数计算机都使用带有双补码的整数表示法，无法表示出 INT_MIN(可能出现的最小整数)的绝对值，因此就没有定义 abs(INT_MIN)。

atof

〖功能〗将字符串转换为浮点数。
〖原型〗double atof (const char *string)
〖位置〗stdlib.h (ISO)
〖说明〗该函数与 strtod 函数类似，只不过不用检查向上溢出和向下溢出错误。多数情况下为了与现有代码兼容才使用 atof 函数，使用 strtod 函数的性能更加稳定。

atoi

〖功能〗将字符串转换为整数。
〖原型〗int atoi (const char *string)
〖位置〗stdlib.h (ISO)
〖说明〗该函数与 atol 函数类似，只不过返回的是整数而不是长整数。atoi 函数也同样是过时的函数，应当使用 strtol 函数代替。

atol

〖功能〗将字符串转换为整数。
〖原型〗long int atol (const char *string)

〖位置〗stdlib.h (ISO)

〖说明〗该函数与 strtol 函数类似，使用 10 作为基本参数，但是不需要检查溢出错误。多数情况下为了与现有代码兼容才使用 atol 函数，使用 strtol 函数的性能更加稳定。

cabs

〖功能〗该函数返回复数 z 的绝对值，复数 z 的实部为 z.real，虚部为 z.imag。

〖原型〗double cabs (struct { double real, imag; } z)

〖位置〗math.h (BSD)

〖说明〗返回值等于：sqrt(z.real×z.real+z.imag×z.imag)。

ceil

〖功能〗取整函数。

〖原型〗double ceil (double x)

〖位置〗math.h (ISO)

〖说明〗将 x 向上取整为最接近的整数，按照双精度数值返回，例如，ceil(1.5)等于 2.0。

copysign

〖功能〗返回值的绝对值与 value 的绝对值相同，符号与 sign 一致。该函数为 BSD 函数。

〖原型〗double copysign (double value, double sign)

〖位置〗math.h (BSD)

div

〖功能〗除法函数。

〖原型〗div_t div (int numerator, int denominator)

〖位置〗stdlib.h (ISO)

〖说明〗该函数计算 numerator 除以 denominator 之后所得的商和余数，返回结果以 div_t 类型的结构表示。

如果结果无法表示(例如被 0 除)，则结果不确定。

以下提供一个样例。

```
div_t result:
result = div (20, -6);
```

得到 result.quot 等于-3，result.rem 等于 2。

drem

〖功能〗求余函数。

〖原型〗double drem (double numerator, double denominator)

〖位置〗math.h (BSD)

〖说明〗drem 函数与 fmod 函数类似，只不过其内商 n 是与最接近的整数取整，而不是与大于 0 的整数取整。例如 drem(6.5, 2.3)返回-0.4，也就是 6.5 减去 6.9 得到的结果。

结果的绝对值小于或者等于 denominator 绝对值的一半。fmod(numerator, denominator)与 drem(numerator, denominator)的结果通常相差 denominator、-denominator 或者 0。

如果 denominator 等于 0，则 drem 失败，并设置 errno 为 EDOM。

fabs

〖功能〗该函数返回浮点数 number 的绝对值。

〖原型〗double fabs (double number)

〖位置〗math.h (ISO)

finite

〖功能〗如果 x 有限或者非数字，则该函数返回一个非 0 值，否则返回 0。

〖原型〗int finite (double x)

〖位置〗math.h (BSD)

floor

〖功能〗函数将 x 向下取整为最接近的整数，返回值为 double 类型。

〖原型〗double floor (double x)

〖位置〗math.h (ISO)

〖说明〗floor(1.5)返回 1.0，floor(-1.5)返回-2.0。

fmod

〖功能〗该函数计算 numerator 除以 denominator 之后得到的余数。

〖原型〗double fmod (double numerator, double denominator)

〖位置〗math.h (ISO)

〖说明〗返回值是 numerator - n * denominator，其中 n 等于 numerator 除以 denominator 之后向 0 取整得到的商。因此 fmod(6.5, 2.3)返回 1.9，也就是 6.5 减去 4.6。

结果的符号与 numerator 相同，其数量级小于 denominator 的数量级。

如果 denominator 等于 0，fmod 将出错，同时设置 errno 为 EDOM。

frexp

〖功能〗函数用来将数字 value 分解为标准化的分数和指数。

〖原型〗double frexp (double value, int *exponent)

〖位置〗math.h (ISO)

〖说明〗如果参数 value 非 0，则返回值为 value 除以 2 的倍数所得的商，并且位于 1/2(含)到 1(不含)之间的范围内。相对应的指数保存在*exponet 中，返回值乘以 2 的指数次幂就等于原来的数字 value。

例如，frexp(12.8, &exponet)返回 0.8，并且在 exponent 中保存 4。

如果 value 等于 0，那么返回值为 0，并且在*exponent 中也保存 0。

infnan

〖功能〗提供该函数是为了满足与 BSD 之间的兼容性。

〖原型〗double infnan (int error)

〖位置〗math.h (BSD)

〖说明〗其他数学函数使用 infnan 来确定出现错误时返回何值。参数为错误代码 EDOM 或者 ERANGE，infnan 随后返回相对应的数值。也可以使用-ERANGE 作为参数，相应的返回值为-HUGE_VAL。

在某些机器上使用 BSD 函数库时，infnan 在任何情况下都设置重要符号。但是 GNU 函数库不同，因为这样不符合 ISO C 的规范。

isinf

〖功能〗如果 x 代表负无穷则函数返回-1，如果 x 代表正无穷则函数返回 1，否则函数返回 0。

〖原型〗int isinf (double x)

〖位置〗math.h (BSD)

isnan

〖功能〗如果 x 不是数字则函数返回非 0 值，否则返回 0(使用 x != x 也可以得到相同的结果)。

〖原型〗int isnan (double x)

〖位置〗math.h (BSD)

labs

〖功能〗该函数与 abs 类似，但是其参数和结果都是 long int 类型，而不是 int 类型。

〖原型〗long int labs (long int number)

〖位置〗stdlib.h (ISO)

ldexp

〖功能〗该函数返回浮点数 value 乘以 2 的 exponent 次幂得到的结果(可以用来重新组合被 frexp 分解的浮点数)。

〖原型〗double ldexp (double value, int exponent)

〖位置〗math.h (ISO)

〖说明〗例如，ldexp(0.8,4)返回 12.8。

ldiv

〖功能〗该函数与 div 类似，但是参数为 long int 类型，并且返回的结果为 ldiv 类型的

结构。

〖原型〗ldiv_t ldiv (long int numerator, long int denominator)

〖位置〗stdlib.h (ISO)

logb

〖功能〗该 BSD 函数返回以 2 为底 x 的对数的整数部分，该整数值以 double 类型表示。

〖原型〗double logb (double x)

〖位置〗math.h (BSD)

〖说明〗返回值是 x 中包含的 2 的最高整数次幂。x 的符号忽略不计。例如，logb(3.5)等于 1，logb(4.0)等于 2.0。

2 的返回值次幂除以 x 之后，可以得到一个 1(含)和 2(不含)之间的商。

如果 x 为 0，则返回值为负的无穷大(如果及其支持这类数值)，或者一个极小的数字。如果 x 为无穷，则返回值也为无穷。

logb 的返回值要比 frexp 在*exponent 中保存的数值小 1。

modf

〖功能〗该函数将参数值分解为整数部分和分数部分(在-1 和 1 之间)。

〖原型〗double modf (double value, double *integer-part)

〖位置〗math.h (ISO)

〖说明〗整数部分和分数部分两者之和等于 value。每个部分的符号都与 value 相同，因此整数部分的舍入接近 0。

函数将整数部分保存在*integer-part 中，然后返回分数部分。例如，modf(2.5, &intpart)返回 0.5 并且在 intpart 中保存 2.0。

rint

〖功能〗该函数根据当前的取整模式将 x 取整为一个整数值。

〖原型〗double rint (double x)

〖位置〗math.h (BSD)

〖说明〗默认的取整模式就是取整为最接近的整数，有些机器支持其他模式，但是通常都向最接近的整数取整，除非明确选择其他取整模式。

scalb

〖功能〗该函数是 ldexp 函数在 BSD 中的别名。

〖原型〗double scalb (double value, int exponent)

〖位置〗math.h (BSD)

strtod

〖功能〗将 string 的初始部分转换成浮点数，返回双精度值。

〖原型〗double strtod (const char *string, char **tailptr)

〖位置〗stdlib.h (ISO)

〖说明〗该函数按以下方式分解 string。

- whitespace 字符序列。Isspace 函数确定哪些字符为 whitespace，这些将被舍弃。
- 可选的+或-。
- 非空的、包含小数点(通常是.)的、任意的数字序列。
- 可选的指数部分，由字符 e 和 E、可选的符号和数字序列组成。
- 串中任意可保留的字符。如果 tailptr 不是空指针，指向串的尾部的指针存储在*tailptr 中。

如果串为空，仅包含空白，或不包含可按浮点数表示的字串，就不执行转换。此时，strtod 返回 0 值，且在*tailptr 中的返回值为 string 的值。

strtof

〖功能〗该函数与函数 strtod 类似，但返回值为浮点型而不是双精度型。因为该函数的执行速度在某些结构中比 strtod 快得多，所以如果浮点值的精度就够用时，应当使和该函数。原因很明显，IEEE 754 规定浮点数有 23 位尾数而双精度数有 53 位且精度中每一个额外的位都要求额外的计算。

〖原型〗float strtof (const char *string, char **tailptr)

〖位置〗stdlib.h (GNU)

〖说明〗如果串对于浮点数而言语法有效，但由于溢出导致数值无法表示，则 strdof 将根据值的符号返回正的或负的 HUGE_VALf。

该函数是 GNU 扩展。

strtold

〖功能〗该函数与函数 strtod 类似，但返回长双精度值而不是双精度值。常用于需要较高精度的情况。在定义了长双精度类型的系统中运行该函数需要花费更多的时间，这是因为需要更多的精度位。

〖原型〗long double strtold (const char *string, char **tailptr)

〖位置〗stdlib.h (GNU)

〖说明〗如果串对于浮点数而言语法有效，但由于溢出导致数值无法表示，则 strdof 将根据值的符号返回正的或负的 HUGE_VALl。

该函数是 GNU 扩展。

strtoll

〖功能〗strtoll 是 strtoq 函数的别名，strtoq 函数的一切都适用于 strtoll。

〖原型〗long long int strtoll (const char *string, char **tailptr, int base)

〖位置〗stdlib.h (GNU)

strtoq

〖功能〗函数和 strtol 类似，但它处理 extra long 数且返回值为 long long int 类型。

〖原型〗long long int strtoq (const char *string, char **tailptr, int base)

〖位置〗stdlib.h (BSD)

〖说明〗如果串对于整数而言语法有效，但由于溢出导致数值无法表示，则 strtoq 将依据值的符号返回 LONG_LONG_MAX 或 LONG_LONG_MIN，并将 errno 设置成 ERANGE，表示溢出。

strtoul

〖功能〗函数和 strto 类似，但它处理无符号数且返回值为无符号长整型。数值前没有+或-，但语法和上面为 strol 描述的一样。溢出时返回值为 ULONG_MAX。

〖原型〗unsigned long int strtoul (const char *string, char **tailptr, int base)

〖位置〗stdlib.h (ISO)

〖说明〗如果 base 的值不在合法的范围内，那么和 strol 一样函数将设置 errno 值并返回 0ul。对于 strtoul 而言这也可能发生在其他情况下。当被转换的数为负时，strtoul 也将设置 errno 为 EINVAL 并且返回 0ul。

strtoull

〖功能〗strtoull 是 strtoq 函数的别名，strtoq 的一切适用于 strtoull。与 strtoll 函数的区别在于，该函数值不带符号。

〖原型〗unsigned long long int strtoull (const char *string, char **tailptr, int base)

〖位置〗stdlib.h (GNU)

strtouq

〖功能〗该函数和 strtoul 类似，但它可以处理超长的数字并返回 unsigned long long int 型的值。

〖原型〗unsigned long long int strtouq (const char *string, char **tailptr, int base)

〖位置〗stdlib.h (BSD)

〖说明〗溢出时返回值为 ULONG_LONG_MAX。

1.13　搜索和排序

bsearch

〖功能〗函数检索 key 对象的排序数组 array。数组中包含有 count 个元素，每个元素的大小为 size 字节。

〖原型〗void * bsearch (const void *key, const void *array, size_t count, size_tsize, comparison_fn_t compare)

〖位置〗stdlib.h (ISO)

〖说明〗使用 compare 函数进行比较。调用该函数需要两个指针参数，并且应当根据第 1 个参数小于、等于或者大于第 2 个参数，从而返回一个小于、等于或者大于 0 的整数。根据比较函数的要求，array 的元素必须已经按照升序排列。

返回值为一个指向匹配数组元素的指针，如果没有找到匹配元素则返回一个空指针。如果数组中包含多个匹配元素，那么返回值不确定。

该函数名称的由来是因为它使用二进制检索算法这一事实。

qsort

〖功能〗该函数为数组 array 排序。

〖原型〗void qsort (void *array, size_t count, size_t size, comparison_fn_t compare)

〖位置〗stdlib.h (ISO)

〖说明〗数组中包括 count 个元素，每个元素大小均为 size。

使用 compare 函数在数组元素之间进行比较。调用该函数时需要两个指针参数，并且应当返回一个小于、等于、或者大于 0 的整数，对应说明第一个参数小于、等于、或者大于第 2 个参数。

警告：如果两个对象相等，则排序之后的顺序不确定，也就是说排序不稳定。在仅仅为元素的某个方面进行比较时，可能会出现不同的结果。两个排序关键字相同的元素在其他方面可能存在差异。

如果需要得到稳定的排序结果，可以自己编写一个比较函数，在两个元素之间缺乏其他的区分条件时，比较两者的地址。这样做可能降低排序算法的效率，因此只有在必要的情况下才使用。

以下这个简单的样例用于为一个双精度数组按照数字顺序排序，使用的比较函数就是上文中定义的。

```
{
  double *array;
  int size;
  ...
  qsort (array, size, sizeof (double), compare_doubles);
}
```

qsort 函数名称正是它最初使用的 quich sort 算法的缩写。

1.14 匹　　配

fnmatch

〖功能〗该函数检查字符串 string 是否与 pattern 模式匹配。

〖原型〗int fnmatch (const char *pattern, const char *string, int flags)

〖位置〗fnmatch.h (POSIX.2)

〖说明〗如果匹配则返回 0，否则返回一个非 0 值 FNM_NOMATCH。参数 pattern 和 string 都是字符串。

参数 flags 是由影响匹配细节的标志位组成的，定义的标志列表如下所示。

在 GNU C 函数库中，fnmatch 不可能出现错误，而总是返回匹配是否成功的回答。但是在执行 fnmatch 时也可能会报错，这一情况将通过返回一个不等于 FNM_NOMATCH 的非 0 值来表示。

以下所示为 flags 参数可用的标志。

FNM_FILE_NAME	对/字符特殊处理，用于匹配文件名称。如果设置该标志位，则 pattern 中的通配符不能与 string 中的/匹配。因此匹配/的惟一方法就是在 pattern 中明确说明/。
FNM_PATHNAME	FNM_FILE_NAME 的别名，源自 POSIX.2。建议不要使用该名称，因为文件名称不再使用术语 pathname。
FNM_PERIOD	对出现在 string 起始位置上的.字符特殊处理。如果设置该标志位，则 pattern 中的通配符不能与 string 中位于第 1 个字符位置的.匹配。如果同时设置 FNM_PERIOD 和 FNM_FILE_NAME，则对于字符串初始位置上.的特殊处理也将同样适用于位于/之后的.字符(命令行解释程序同时使用 FNM_PERIOD 和 FNM_FILE_NAME 标志来匹配文件名称)。
FNM_NOESCAPE	不要对样式中的\字符进行特殊处理。通常\都是用来引用随后的字符，关闭其特殊含义(如果存在的话)，从而只能与自身匹配。启动引用时，样式\?就只能匹配字符串?，因为样式中的问号只表示一个普通字符。如果使用 FNM_NOESCAPE，则\就是一个普通字符。
FNM_LEADING_DIR	忽略 string 中以/开始的后续字符，也就是说检查 string 是否以与 pattern 匹配的目录名开始。如果设置该标志位，则 foo*或者 foobar 样式都可以与字符串 foobar/frobozz 相匹配。
FNM_CASEFOLD	在比较 string 和 pattern 时忽略大小写差异。

glob

〖功能〗该函数使用 pattern 模式在当前目录中执行集团查找。

〖原型〗int glob (const char *pattern, int flags, int (*errfunc) (const char *filename, int error-code), glob_t *vector-ptr)

〖位置〗glob.h (POSIX.2)

〖说明〗查找结果保存在一个新分配的向量中，并且将该向量的大小和地址保存在*vector-ptr 中。参数 flags 是标志位的集合。

集团查找返回的结果是一个文件名序列。函数 glob 为每个结果分配一个字符串，然后分配一个 char **类型的向量来保存这些字符串的地址。向量的最后一个元素是一个空指针。该向量被称为 word vector。

为了返回该向量，函数将其地址和长度（即元素个数，结尾的空指针不算在内）保存在*vector-ptr 中。

通常函数在返回结果之前会将文件名称按照字母顺序排序。如果想要尽快得到结果也可以使用 GLOB_NOSORT 标志来关闭这一功能。建议允许 glob 函数进行排序—如果按照字母顺序处理文件，用户会感受到应用程序的进步。

如果函数执行成功，则返回 0。否则返回如下错误代码中的一个。

GLOB_ABORTED	打开目录时出现一个错误，并且使用的 GLOB_ERR 标志或者指定的 errfunc 返回一个非 0 值。
GLOB_NOMATCH	模式无法与任何文件匹配。如果使用 GLOB_NOCHECK 标志，就永远不会出现该错误代码，因为该标志位将通知函数假装至少找到了一个文件。
GLOB_NOSPACE	无法为保存结果分配内存。

出现错误时，函数将当时得到的匹配信息保存在*vector-ptr 中。

regcomp

〖功能〗该函数将规则的表达式编译为一个数据结构，从而可以使用 regexec 来与字符串匹配。

〖原型〗int regcomp (regex_t *compiled, const char *pattern, int cflags)

〖位置〗regex.h (POSIX.2)

〖说明〗编译后的规则表达式保存在*compiled 中，可以完成高效匹配。

用户需要自己分配一个 regex_t 类型的对象，并将其地址传递给 regcomp。

参数 cflags 指定各种选项，控制规则表达式的语法和语义。

如果使用 REG_NOSUB 标志，则 regcomp 将省略编译后规则表达式中记录子表达式如何匹配的重要信息。这种情况下，最好在调用 regexec 时为 matchptr 和 nmatch 参数传递 0 值。

如果不使用 REG_NOSUB 标志，那么编译后规则表达式将可以记录子表达式如何匹配的信息。并且 regcomp 可以通过将数字保存在 compiled->re_nsub 中，从而说明 pattern 中包含多少个子表达式。可以使用该数值来确定保存有关子表达式匹配信息的数组需要分配长度。

如果函数成功编译规则表达式，则返回 0，否则返回一个非 0 的错误代码。可以使用 regerror 来产生一条错误消息字符串，描述得到非 0 值的原因。

regcomp 函数可能返回的非 0 值如下。

REG_BADBR	在规则表达式中存在无效的\{...\}结构。有效的\{...\}结构中必须包含一个单精度数字，或者包含使用逗号分隔的两个按照升序排列的数字。
REG_BADPAT	规则表达式中存在语法错误。
REG_BADRPT	在错误的位置上出现了类似?或者*的循环操作符(之前没有可以操作的子表达式)。
REG_ECOLLATE	规则表达式引用了一个无效的比较元素(元素没有在字符串比较的当前场所中定义)。

REG_ECTYPE	规则表达式引用了无效的字符类名。
REG_EESCAPE	规则表达式以\结束。
REG_ESUBREG	在\digit 结构中存在无效数字。
REG_EBRACK	在规则表达式中存在不匹配的方括号。
REG_EPAREN	扩展的规则表达式中存在不匹配的圆括号，或者基础规则表达式中存在不匹配的\(和\)。
REG_EBRACE	规则表达式中存在不匹配的\{和\}。
REG_ERANGE	范围表达式中的某一个端点无效。
REG_ESPACE	函数运行的内存不足。

regerror

〖功能〗该函数产生错误代码 errcode 对应的错误消息字符串，并将字符串保存在从 buffer 开始的 length 个字节内存中。

〖原型〗size_t regerror (int errcode, regex_t *compiled, char *buffer, size_t length)

〖位置〗regex.h (POSIX.2)

〖说明〗compiled 参数提供 regcomp 或者 regexec 运行出错时使用的编译规则表达式结构。如果 compiled 提供 NULL，则仍然将得到一条有意义的错误消息，但是不够详细。

如果错误消息的长度不等于 length 个字节(包括结束空字符在内)，则 regerror 将截断字符串。regerror 保存的字符串即便截断之后也必须以空字符结束。

regerror 的返回值就是保存整条错误消息所需的最小长度。如果该值小于 length，则错误消息无需截断，可供使用。否则必须使用较大的缓存区再次调用 regerror。

下面是一个使用 regerror 的函数，但总是为错误消息动态分配一个缓存区。

```
char *get_regerror (int errcode, regex_t *compiled)
{
  size_t length = regerror (errcode, compiled, NULL, 0);
  char *buffer = xmalloc (length);
  (void) regerror (errcode, compiled, buffer, length);
  return buffer;
}
```

regexec

〖功能〗该函数用于匹配编译后规则表达式*compiled 和 string。

〖原型〗int regexec (regex_t *compiled, char *string, size_t nmatch, regmatch_t matchptr [], int eflags)

〖位置〗regex.h (POSIX.2)

〖说明〗如果规则表达式匹配成功则返回 0，否则返回一个非 0 值。不同非 0 值表达的含义说明参见下面的表格。可以使用 regerror 产生错误消息字符串，说明出现非 0 值的原因。

参数 eflags 是激活不同选项的标志位。

如果需要了解 string 的哪个部分真正与规则表达式或其子表达式匹配，则使用参数 matchptr 和 nmatch。否则就向 nmatch 传递 0，向 matchptr 传递 NULL。

规则表达式必须在编译时所处场所的相同集合中匹配。

函数 regexec 接受 eflags 参数中的如下标志。

REG_NOTBOL	认为指定字符串的开始位置不是一行的开头，也就是说，不要设想之前的文字。
REG_NOTEOL	认为指定字符串的结束位置不是一行的结尾，也就是说，不要设想之后的文字。

以下是 regexec 可能返回的非 0 值。

REG_NOMATCH	模式与字符串不匹配。这不算真正的错误。
REG_ESPACE	regexec 函数运行的内存不足。

regfree

〖功能〗该函数用于释放*compiled 指向的所有存储内容，其中包括使用手册上没有说明的 regex_t 结构中各种内部字段。

〖原型〗void regfree (regex_t *compiled)

〖位置〗regex.h (POSIX.2)

〖说明〗该函数不会释放*compiled 对象本身。

在使用结构编译其他规则表达式之前，应当使用 regfree 释放 regex_t 结构中的空间。

wordexp

〖功能〗该函数对字符串 words 执行单词扩展，将结果放在新分配的矢量中，并且在*word-vector-ptr 中保存该矢量的大小和地址。参数 flags 是标志位的集合。

〖原型〗int wordexp (const char *words, wordexp_t *word-vector-ptr, int flags)

〖位置〗wordexp.h (POSIX.2)

〖说明〗字符串 words 中不要使用任何|&:<>字符，除非使用引号将它们引起来，换行符也同样处理。如果使用这些字符，将得到 WRDE_BADCHAR 错误代码。如果没有引号就不要使用圆括号或者花括号。如果使用引号字符，则应当成对出现。

单词扩展的结果就是得到一个单词序列。函数 wordexp 为每个结果单词分配一个字符串，然后分配一个 char **类型的矢量，保存这些字符串的地址。矢量的最后一个元素是一个空指针。该矢量称为 work vector。

为了返回该矢量，wordexp 函数将其地址和长度(元素个数，末尾的空指针不计在内)保存在*word-vector-ptr 中。

如果函数成功则返回 0，否则返回如下错误代码中的一个。

WRDE_BADCHAR	输入字符串 words 中包含不带引号的无效字符，例如|。
WRDE_BADVAL	输入字符串引用一个未定义的命令行变量，而用户已经使用 WRDE_UNDEF 标志来禁止这类引用。

WRDE_CMDSUB	输入字符串使用命令替换，而用户已经使用 WRDE_NOCMD 标志来禁止了命令替换。
WRDE_NOSPACE	无法分配用于保存结果的内存。这种情况下，函数可以保存部分结果—根据能够分配的空间量决定。
WRDE_SYNTAX	输入字符串中有语法错误。例如，不匹配的引号字符就是语法错误。

wordfree

〖功能〗该函数释放单词字符串和*word-vector-ptr 指向的矢量所使用的存储空间，但是不会释放*word-vector-ptr 结构本身，而是只释放它指向的其他数据。

〖原型〗void wordfree (wordexp_t *word-vector-ptr)

〖位置〗wordexp.h (POSIX.2)

1.15　日期和时间

adjtime

〖功能〗该函数用于调整时间。

〖原型〗int adjtime (const struct timeval *delta, struct timeval *olddelta)

〖位置〗sys/time.h (BSD)

〖说明〗该函数用于加快或减慢系统时钟，以便对当前时间作出逐步地调整。其确保系统时钟报告的时间总是单调递增的，只是简单地设置当前时间不可能达到这一目的。

delta 参数指定对当前时间要做的有关调整。如果是负的，则将减慢系统时钟，直到已丢弃该部分时间。如果是正的，将加快系统时钟。

如果 olddelta 参数不是空指针，adjtime 函数将返回先前还未完成的时间调整的相关信息。

该函数用于同步局域网中的计算机时钟，只有特许用户才可以使用该函数。返回值为 0 时表示成功，-1 表示失败。该函数定义的 errno 错误代码如下所示。

EPERM	没有设置时间的限权。

alarm

〖功能〗该函数用于设置警报器。

〖原型〗unsigned int alarm (unsigned int seconds)

〖位置〗unistd.h (POSIX.1)

〖说明〗该函数设置在 seconds 秒内终止的实时定时器。如果要取消任何现存的警报器，可以通过调用 seconds 参数为 0 的 alarm 函数来实现。

返回值显示距离下一个警报的发出时间还剩余的秒数。如果下面没有警报器，alarm 返回 0。

使用 setitimer 定义 alarm 函数如下。

```
unsigned int
alarm (unsigned int seconds)
{
  struct itimerval old, new;
  new.it_interval.tv_usec = 0;
  new.it_interval.tv_sec = 0;
  new.it_value.tv_usec = 0;
  new.it_value.tv_sec = (long int) seconds;
  if (setitimer (ITIMER_REAL, &new, &old) < 0)
    return 0;
  else
    return old.it_value.tv_sec;
}
```

如果想让处理器简单地等待指定的秒数，可以使用 sleep 函数。

在多处理器环境中，涉及大量的延迟，程序样例如下。

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
/* This flag controls termination of the main loop. */
volatile sig_atomic_t keep_going = 1;
/* The signal handler just clears the flag and re-enables itself. */
void catch_alarm (int sig)
{
  keep_going = 0;
  signal (sig, catch_alarm);
}
void do_stuff (void)
{
puts ("Doing stuff while waiting for alarm....");
}
int main (void)
{
  /* Establish a handler for SIGALRM signals. */
  signal (SIGALRM, catch_alarm);
  /* Set an alarm to go off in a little while. */
  alarm (2);
  /* Check the flag once in a while to see when to quit. */
  while (keep_going)
    do_stuff ();
  return EXIT_SUCCESS;
}
```

〖参见〗itimerval，setitimer，getitimer，sleep。

asctime

〖功能〗该函数将时间格式化输出为标准格式的字符串。

〖原型〗char * asctime (const struct tm *brokentime)

〖位置〗time.h (ISO)

〖说明〗该函数将 brokentime 指向的不合用时间值转换为一个标准格式的字符串：

“Tue May 21 13:46:22 1991\n”

一星期中每天的缩写是：“Sun”、“Mon”、“Tue”、“Wed”、“Thu”、“Fri”和“Sat”。

月份的缩写是：“Jan”、“Feb”、“Mar”、“Apr”、“May”、“Jun”、“Jul”、“Aug”、“Sep”、“Oct”、“Nov”和“Dec”。

返回值指向一个静态分配的字符串，该字符串可能会由于对 asctime 或 ctime 的并发调用而被覆盖(但是没有其他的库函数可以覆盖该字符串的内容)。

clock

〖功能〗该函数返回消耗的处理器时间，基准时间可以任意确定，但是在一个进程中前后发生变化。如果处理器时间不可用或者无法表示，则 clock 函数返回(CLOCK_T) (-1)。

〖原型〗clock_t clock (void)

〖位置〗time.h (ISO)

ctime

〖功能〗该函数格式化日期和时间。

〖原型〗char * ctime (const time_t *time)

〖位置〗time.h (ISO)

〖说明〗ctime 函数与 asctime 函数类似，只不过时间值由 time_t 的日历时间值指定，而不是由分解的本地时间格式指定。它等价于：

```
asctime (localtime (time))
```

因为 localtime 函数将设置变量 tzname，因此 ctime 函数也会设置变量 tzname。

difftime

〖功能〗该函数用于计算两个时间值之间相差的秒数。

〖原型〗double difftime (time_t time1, time_t time0)

〖位置〗time.h (ISO)

〖说明〗对该函数返回在 time1 和 time0 之间间隔的秒数，返回值类型为 double。如果不激活对闰秒的支持，则闰钞将忽略不计。

在 GNU 系统中可以减去 time_t 数值。但是在其他系统中，time_t 数据类型可能使用其他的编码，因此不能直接使用减法计算。

getitimer

〖功能〗该函数在 old 指向的结构中保存 which 指定的定时器信息。

〖原型〗int getitimer (int which, struct itimerval *old)

〖位置〗sys/time.h (BSD)

〖说明〗返回值和错误代码与 setitimer 相同。

ITIMER_REAL	该常量可以作为 setitimer 和 getitimer 函数的 which 参数使用，指定实时的定时器。
ITIMER_VIRTUAL	该常量可以作为 setitimer 和 getitimer 函数的 which 参数使用，指定虚拟的定时器。
ITIMER_PROF	该常量可以作为 setitimer 和 getitimer 函数的 which 参数使用，指定侧面的定时器。

getpriority

〖功能〗该函数读入由 class 和 id 指定的某一类进程的优先级。
〖原型〗int getpriority (int class, int id)
〖位置〗sys/resource.h (BSD)
〖说明〗如果指定进程的优先级不完全一致，则返回其中的最小值。
成功则函数返回优先级数值，失败返回-1。该函数定义了如下 errno 错误代码。

ESRCH	class 和 id 的组合无法与当前存在的某个进程匹配。
EINVAL	class 的数值无效。

当返回值等于-1 时，既可以说明运行失败，也可以说明某个优先级数值。确定是否失败的惟一方法就是在调用 getpriority 值前设置 errno 等于 0，然后使用 errno != 0 作为判断是否失败的条件。

getrlimit

〖功能〗该函数读入资源 resource 的当前值和最大值，并且将结果保存在*rlp 中。
〖原型〗int getrlimit (int resource, struct rlimit *rlp)
〖位置〗sys/resource.h (BSD)
〖说明〗成功则函数返回 0，失败返回-1。惟一可能出现的 errno 错误代码就是 EFAULT。

getrusage

〖功能〗该函数报告 processes 指定的进程使用总量，并且将信息保存在*rusage 中。
〖原型〗int getrusage (int processes, struct rusage *rusage)
〖位置〗sys/resource.h (BSD)
〖说明〗在多数系统中，processes 只有两个有效值。

RUSAGE_SELF	表示当前进程。
RUSAGE_CHILDREN	表示已经结束的所有子进程(直接和间接)。

在 GNU 系统中，也可以通过指定其进程 ID 来确定查询某个特别的子进程。

成功时 getrusage 函数的返回值为 0，失败则返回-1。函数定义的错误代码如下。

EINVAL　参数 processes 无效。

使用函数 wait4 可以得到某个子进程的使用情况图表，当子进程结束时返回进程使用总量。

gettimeofday

〖功能〗该函数返回 struct timeval 结构中当前的日期和时间，该结构由 tp 指定。
〖原型〗int gettimeofday (struct timeval *tp, struct timezone *tzp)
〖位置〗sys/time.h (BSD)
〖说明〗有关时区的信息在 tzp 指向的结构中返回。如果 tzp 参数是一个空指针，则忽略时区信息。
成功则函数返回 0，失败返回-1。该函数定义了如下 errno 错误代码。

ENOSYS　操作系统不支持时区信息，并且 tzp 不是空指针。GNU 操作系统不支持使用 struct timezone 来表示时区信息，这只是 4.3 BSD 废弃的特性。

gmtime

〖功能〗该函数与 localtime 类似，但是这里的时间使用世界协调时(UTC)表示，也就是格林威治时间(GMT)，与本地时区无关。
〖原型〗struct tm * gmtime (const time_t *time)
〖位置〗time.h (ISO)
〖说明〗日历时间也通常使用世界协调时(UTC)表示。

localtime

〖功能〗该函数将 time 指向的日历时间转换为分解时间表示，结果与用户指定的时区相关。
〖原型〗struct tm * localtime (const time_t *time)
〖位置〗time.h (ISO)
〖说明〗返回一个指针，指向一个静态的分解时间结构，下一次调用 ctime、gmtime 或者 localtime 将覆盖其内容(但是函数库中的其他函数不会覆盖其内容)。
调用 localtime 还将出现另一个后果：就是使用当前时区的有关信息来设置变量 tzname 的值。

mktime

〖功能〗该函数用来将分解时间结构转换为日历时间的表达形式。
〖原型〗time_t mktime (struct tm *brokentime)
〖位置〗time.h (ISO)

〖说明〗同时还得重新规范分解时间结构的内容，根据其他日期和时间内容填写星期几和当年的第几天。

该函数忽略分解时间结构中的 tm_wday 和 tm_yday 成员的特殊内容。使用其他内容来计算日历时间，并且允许这些内容拥有超出正常范围的非正常数值。该函数所做的最后一件事就是调整 brokentime 结构的内容(包括 tm_wday 和 tm_yday)。

如果指定的分解时间不能使用日历时间表示，则函数返回值(time_t)(-1)，并且不改变 brokentime 的内容。

nice

〖功能〗该函数按照参数 increment 的值递增当前进程的优先级。

〖原型〗int nice (int increment)

〖位置〗dunno.h (dunno.h)

〖说明〗返回值与 setpriority 函数相同。

以下是 nice 的定义。

```
int
nice (int increment)
{
  int old = getpriority (PRIO_PROCESS, 0);
  return setpriority (PRIO_PROCESS, 0, old + increment);
}
```

setitimer

〖功能〗该函数根据 new 的值设定由 which 指定的计时器。参数 which 的值由 ITIMER_REAL、ITIMER_VIRTUAL 或者 ITIMER_PROF 组成。

〖原型〗int setitimer (int which, struct itimerval *new, struct itimerval *old)

〖位置〗sys/time.h (BSD)

〖说明〗如果 old 不是空指针，则 setitimer 将返回之前所有没有过期的定时器信息，这些定时器的结构都与其指向的结构类型相同

成功则函数返回 0，失败则返回-1。该函数定义的 errno 错误代码如下。

EINVAL　定时器的间隔过长。

setpriority

〖功能〗将进程的一个类的优先权设置为 priority，用 class 和 id 指定是哪一个进程。

〖原型〗int setpriority (int class, int id, int priority)

〖位置〗sys/resource.h (BSD)

〖说明〗成功时函数返回 0，失败返回-1。该函数定义的 errno 错误代码如下。

ESRCH　class 和 id 的组合与现有的任意进程都不匹配。

EINVAL	class 值无效。
EPERM	试图设置其他用户进程的优先权，但权限不足。
EACCES	试图降低进程的优先级，但权限不足。

参数 class 和 id 可以一起指定一组感兴趣的进程，class 可能的取值有。

PRIO_PROCESS	读取或设置一个进程的优先级。参数 id 是一个进程的 ID。
PRIO_PGRP	读取或设置一个进程组的优先级。参数 id 是一个进程组的 ID。
PRIO_USER	读取或设置一个用户进程的优先级。参数 id 是一个用户的 ID。

如果参数 id 为 0，则代表 class 参数所确定的当前进程、当前进程组或当前用户。

setrlimit

〖功能〗该函数保存*rlp 中的资源 resource 的当前值和最大值。
〖原型〗int setrlimit (int resource, struct rlimit *rlp)
〖位置〗sys/resource.h (BSD)
〖说明〗成功时函数返回 0，失败返回-1。函数定义的 errno 错误代码如下。

EPERM：	试图改变可允许的最大限制值，但权限不足。

settimeofday

〖功能〗该函数根据参数设置当前时间和日期。和 gettimeofday 一样，如果 tzp 是空指针则函数忽略时区信息。
〖原型〗int settimeofday (const struct timeval *tp, const struct timezone *tzp)
〖位置〗sys/time.h (BSD)
〖说明〗有权限的用户才可使用 settimeofday 函数。
成功时函数返回 0，失败时返回-1。该函数定义的 errno 错误代码如下。

EPERM	进程无权设置时间。
ENOSYS	操作系统不支持设置时区信息，且 tzp 不是空指针。

sleep

〖功能〗该函数将使系统等待数秒，直至超过指定时间或直至传递一个信号，无论两者哪个先发生，该函数都将返回。
〖原型〗unsigned int sleep (unsigned int seconds)
〖位置〗unistd.h (POSIX.1)
〖说明〗如果因为超出指定时间返回，则函数返回 0 值。如果因为传递的信号返回，则返回值是在 sleep 过程还剩余的时间。

strftime

〖功能〗该函数与 sprintf 函数类似，但是在格式模板 template 中出现的转换说明符是为 brokentime 中的打印组件提供的。

〖原型〗size_t strftime (char *s, size_t size, const char *template, const struct tm *brokentime)

〖位置〗time.h (POSIX.2)

〖说明〗template 中出现的普通字符复制到输出字符串 s 中，可以包括多字节的字符序列。转换说明符以'%'字符开始，随后的可选标志可以从下面各条内容中任选，这些标志都是 GNU 的扩展，只影响数字的输出。

_ 数字使用空格填充。

- 数字不填充。

0 即便格式指定使用空格填充，数字也要使用 0 填充。

^ 在可能的条件下使用大写字符输出。

默认的动作是使用 0 填充数字，从而保持固定宽度。

time

〖功能〗该函数按照 time_t 类型返回当前时间。

〖原型〗time_t time (time_t *result)

〖位置〗time.h (ISO)

〖说明〗如果 result 不是空指针，时间值也存储在*result 中。如果日历时间不可用，则返回(time_t)(-1)。

times

〖功能〗该函数为 buffer 中的调用进程存储处理器时间信息。

〖原型〗clock_t times (struct tms *buffer)

〖位置〗sys/times.h (POSIX.1)

〖说明〗返回值与 clock( )相同：相对一个任意基准而言的实际使用时间。在特定的进程内基准是不变的，通常代表系统启动后的时间。返回(clock_t)(-1)表示函数执行失败。

注意，clock 函数由 ISO C 标准指定。Times 函数是 POSIX.1 的一部分。在 GNU 系统中，函数 clock 的返回值等于函数 times 的返回值 tms_utime 与 tms_stime 的和。

tzset

〖功能〗该函数用来从 TZ 环境变量值中初始化 tzname 变量。

〖原型〗void tzset (void)

〖位置〗time.h (POSIX.1)

〖说明〗用户程序不必特意调用该函数，因为当使用了其他需要依靠时区的时间转换函数时，将自动调用该函数。

1.16　扩展字符集

mblen

〖功能〗使用非空字符串作为参数时，函数返回从 string 开始组成多字节字符的字节数，最多检查 size 个字节。

〖原型〗int mblen (const char *string, size_t size)

〖位置〗stdlib.h (ISO)

〖说明〗根据 mblen 的返回值可以区分出以下 3 种可能：string 的前 size 个字节以有效的多字节字符开始，开始的字符是无效的字节序列或者某个字符的一部分，string 指向一个空字符串(一个空字符)。

对于有效的多字节字符而言，mblen 返回该字符中的字节数(通常情况下至少为 1，并且不会大于 size)。遇到无效字节序列时，mblen 返回-1，遇到空字符串时返回 0。

如果多字节字符代码使用上档字符，则 mblen 在扫描时将保持或者转换为上档状态。如果调用 mblen 时 string 为一个空指针，则将上档状态初始化为标准的初始值。如果使用的多字节字符代码的确处于上档状态，则返回非 0 值。

mbstowcs

〖功能〗该函数将以空字符结束的多字节字符串转换为一个宽字符码数组。

〖原型〗size_t mbstowcs (wchar_t *wstring, const char *string, size_t size)

〖位置〗stdlib.h (ISO)

〖说明〗从 wstring 开始存储到数组中的宽字符数不大于 size 个，空字符结束符计算在字符个数中，所以 size 小于 string 中宽字符实际个数时将不存储空字符结束符。

mbtowc

〖功能〗该函数将 string 中的第 1 个多字节字符转换为其相应的宽字符码。

〖原型〗int mbtowc (wchar_t *result, const char *string, size_t size)

〖位置〗stdlib.h (ISO)

〖说明〗该函数的结果存储在*result 中。

wcstombs

〖功能〗该函数将以空字符结束的宽字符数组 wstring 转换为一个包含多字节字符的字符串。

〖原型〗size_t wcstombs (char *string, const wchar_t wstring, size_t size)

〖位置〗stdlib.h (ISO)

〖说明〗从 string 开始，该函数所存储的字节数小于等于 size 个，如果还有空间则以空字符结束。

wctomb

〖功能〗该函数将指定的宽字符码 wchar 转换为其相应的多字节字符序列。

〖原型〗int wctomb (char *string, wchar_t wchar)

〖位置〗stdlib.h (ISO)

〖说明〗该函数将转换结果以字节方式存储，最多存储 MB_CUR_MAX 个字符。

1.17 本地和国际化

localeconv

〖功能〗该函数返回一个指针，指向的结构内容中包含有当前场合下数字和货币值格式的有关信息。

〖原型〗struct lconv * localeconv (void)

〖位置〗locale.h (ISO)

〖说明〗结构及其内容不能更改。下一次调用 localeconv 或者 setlocale 函数将覆盖结构的内容，但是函数库中的其他函数不会覆盖其内容。

setlocale

〖功能〗该函数将设置目录 category 的当前场所为 locale。

〖原型〗char * setlocale (int category, const char *locale)

〖位置〗locale.h (ISO)

〖说明〗如果 category 是 LC_ALL，就指定所有的场所。如果 category 使用其他可能的数值，则指定某个特别用途的场所。

也可以通过传递一个空指针作为 locale 参数，从而利用该函数来找出当前场所。在这种情况下，setlocale 返回目录 category 当前选择的场所名。

setlocale 返回的字符串可以被后续的调用覆盖，如果希望保存以前调用 setlocale(标准函数保证绝不调用 setlocale 本身)的返回值，就应当拷贝该字符串。

setlocale 返回的字符串不能修改。它可能就是在先前调用 setlocale 时作为参数传递的那个字符串。

当读取目录 LC_ALL 的当前场所时，返回的数值是所有目录选择的场所集合的混合编码。在这种情况下，数值不仅仅是单个场所的名称。实际上也无法确定返回值的形式。但是如果下一次调用 setlocale 函数时，对 LC_ALL 指定同一个场所名称，则恢复原有的选择场所。

当 locale 参数不是空指针时，setlocale 返回的字符串表示最近更改的场所。

如果 locale 参数指定一个空字符串，则表示读入相应的环境变量，并且用来选择 category 的场所。

如果指定的场所名称无效，则 setlocale 函数将返回一个空指针，当前场所保持不变。

以下的样例说明如何使用 setlocale 临时切换新场所。

```
#include <stddef.h>
#include <locale.h>
#include <stdlib.h>
#include <string.h>
void
with_other_locale (char *new_locale,
                   void (*subroutine) (int),
                   int argument)
{
  char *old_locale, *saved_locale;
  /* Get the name of the current locale.  */
  old_locale = setlocale (LC_ALL, NULL);
  /* Copy the name so it won't be clobbered by setlocale. */
  saved_locale = strdup (old_locale);
  if (old_locale == NULL)
    fatal ("Out of memory");
  /* Now change the locale and do some stuff with it. */
  setlocale (LC_ALL, new_locale);
  (*subroutine) (argument);
  /* Restore the original locale. */
  setlocale (LC_ALL, saved_locale);
  free (saved_locale);
}
```

可移植性说明：有些 ISO C 系统可能定义额外的场所目录。为了提高可移植性，设定以 LC_开头的任何变量都在 locale.h 中定义。

1.18　非本地退出

longjmp

〖功能〗该函数用来将状态恢复为 state，并且从调用 setjmp 设置的返回点位置开始继续向下执行。

〖原型〗void longjmp (jmp_buf state, int value)

〖位置〗setjmp.h (ISO)

〖说明〗setjmp 的返回值由 longjmp 决定，如果传递给 longjmp 的参数 value 不等于 0 则返回该值，否则返回 1。

在使用 setjmp 和 longjmp 时有许多较含糊的限制，但是它们都非常重要。其中多数限制的存在都是因为非本地出口要求 C 编译程序这一部分具备相当的魔力，并且可以通过罕见的方式与语言的其他部分相互联系。

setjmp 函数实际上是一个宏，没有明确的函数定义，因此不能使用#undef 或者得到其地址。此外只有在如下情况中调用 setjmp 才是安全的。

- 作为某个选择或者循环语句的测试表达式(例如 if、switch 或者 while)。
- 作为在某个选择或者循环语句的测试表达式中出现的赋值或者比较操作符的一个操

作元。另一个操作元必须是整数常量表达式。

- 作为一元'!'操作符的操作元出现在某个选择或者循环语句的测试表达式中。
- 自身作为表达式语句。

返回点只有位于调用 setjmp 的函数动态范围之内才有效。如果 longjmp 指向的返回点位于已经返回的函数中，则很可能导致不可预料的灾难性事件发生。

longjmp 应当使用非 0 值作为 value 参数值。因为 longjmp 拒绝传回 0 参数作为 setjmp 的返回值，因此也就潜在地成为避免出现偶然误用的安全网，但这种编程方式实际上并不好。

出现非本地出口时，访问对象通常都保留调用 longjmp 时刻的任何值。这里有一个例外情况，就是由于 setjmp 调用不确定，因此在调用 setjmp 函数的本地范围内的自动变量值都将发生改变，除非将它们全部声明为 volatile。

siglongjmp

〖功能〗该函数与 longjmp 类似，但是参数 state 的类型不同。如果 sigsetjmp 使用一个非 0 的 savesigs 标志设置 state，siglongjmp 也恢复阻塞信号的集合。

〖原型〗void siglongjmp (sigjmp_buf state, int value)

〖位置〗setjmp.h (POSIX.1)

sigsetjmp

〖功能〗该函数与 setjmp 类似。如果 savesigs 非空，则在 state 中保存阻塞信号的集合，稍后使用 state 执行 siglongjmp 时将恢复这一集合。

〖原型〗int sigsetjmp (sigjmp_buf state, int savesigs)

〖位置〗setjmp.h (POSIX.1)

1.19 信 号 处 理

gsignal

〖功能〗该函数与 raise 函数相同。

〖原型〗int gsignal (int signum)

〖位置〗signal.h (SVID)

〖说明〗只是为了与 SVID 的兼容性才提供该函数。

raise 的一个用途就是重新产生已捕获信号的默认动作。例如，假设程序的某个用户输入 SUSP 字符，发送一个交互式的停止信号(SIGTSTP)，并且需要在停止之前清除某些内部数据缓存区。其实现方法如下。

```
include <signal.h>
* When a stop signal arrives, set the action back to the default
  and then resend the signal after doing cleanup actions. */
oid
```

```
stp_handler (int sig)
  signal (SIGTSTP, SIG_DFL);
  /* Do cleanup actions here. */
  ...
  raise (SIGTSTP);
* When the process is continued again, restore the signal handler. */
void
cont_handler (int sig)
{
  signal (SIGCONT, cont_handler);
  signal (SIGTSTP, tstp_handler);
}
/* Enable both handlers during program initialization. */
int
main (void)
{
  signal (SIGCONT, cont_handler);
  signal (SIGTSTP, tstp_handler);
  ...
}
```

可移植性说明：raise 函数由 ISO C 委员会发明。旧系统可能不支持，因此使用 kill 函数的移植性可能更强。

kill

〖功能〗该函数向 pid 指定的进程或者进程组发送信号 signum。

〖原型〗int kill (pid_t pid, int signum)

〖位置〗signal.h (POSIX.1)

〖说明〗除了标准信号之外，signum 取 0 值时还可以检查 pid 的有效性。

pid 用来指定接收信号的进程或者进程组，其取值及含义如下。

pid > 0	标识符为 pid 的进程
pid == 0	与发送方在同一个进程组中的所有进程
pid < -1	标识符为-pid 的进程
pid == -1	如果是特许进程，则将信号发送给除了特殊系统进程之外的所有进程。否则将信号发送给有效用户 ID 相同的所有进程。

进程可以使用类似 kill (getpid(), signum)的调用给自己发送信号 signum。如果某个进程使用 kill 给自己发送信号，并且信号没有阻塞，则 kill 返回之前将向该进程发送至少一个信号(可能使用其他不定的未阻塞信号代替信号 signum)。

如果信号发送成功，则 kill 的返回值为 0。否则将不发送任何信号，并返回-1。如果 pid 指定向多个进程发送信号，则只要一个进程发送成功，kill 就执行成功。但是无法判别出哪个进程收到信号，或者是否所有进程都收到信号。

该函数定义的 errno 错误代码如下。

EINVAL　参数 signum 是无效数字或者不支持的数字。
EPERM　没有向名为 pid 的进程或者进程组发送信号的权限。
ESCRH　参数 pid 引用的进程或者进程组不存在。

调用 kill (getpid (), sig)得到的效果与 raise (sig)相同。

killpg

〖功能〗该函数与 kill 类似，但是只向进程组 pgid 发送信号 signum。
〖原型〗int killpg (int pgid, int signum)
〖位置〗signal.h (BSD)
〖说明〗该函数是为了与 BSD 兼容提供的，使用 kill 函数的可移植性更强。

pause

〖功能〗该函数在执行函数或者中断进程的信号到达之前挂起执行程序。
〖原型〗int pause ()
〖位置〗unistd.h (POSIX.1)
〖说明〗如果信号导致执行某个函数，则 pause 函数将返回。这种情况被看作一次不成功的返回(因为成功的操作就应当永远挂起程序)，此时返回值为-1。即便规定当系统句柄返回时应当继续执行其他原型，对 pause 函数也没有影响，只要处理了信号量就认为该函数失败。

该函数还定义了如下 errno 错误代码。

EINTR　函数被信号量的传递中断。

如果信号量导致程序中断，则 pause 不(明确)返回。

psignal

〖功能〗该函数打印一条消息，在标准的错误输出流 stderr 中描述信号 signum。
〖原型〗void psignal (int signum, const char *message)
〖位置〗signal.h (BSD)
〖说明〗如果调用 psignal 时使用的 message 参数为一个空指针或者空字符串，则函数将打印与 signum 对应的消息，并添加一个换行符。

如果提供的 message 参数非空，则 psignal 将参数值输出前缀在字符串之前。并且添加一个冒号和一个空格字符来分隔 message 与 signum 对应的字符串。

raise

〖功能〗该函数向调用进程发送信号 signum。
〖原型〗int raise (int signum)

〖位置〗signal.h (ISO)

〖说明〗成功则函数返回 0，失败则返回一个非 0 值。失败的惟一原因就是 signum 的值无效。

sigaction

〖功能〗参数 action 用于为信号 signum 建立一个新动作，而参数 old-action 用于返回关于先前与该标记相关的动作信息(换句话说，old-action 与函数 signal 返回值目的相同，可以检查原先生效的动作对信号有何影响，愿意的话，稍后可以存储)。

〖原型〗int sigaction (int signum, const struct sigaction *action, struct sigaction *old-action)

〖位置〗signal.h (POSIX.1)

〖说明〗action 或 old-action 都可以是空指针。如果 old-action 是空指针，就简单地抑制旧操作的返回信息。如果 action 是空指针，与信号 signum 相关联的操作不变。从而允许用户在不改变处理方法的前提下，查询信号如何处理。

成功时函数返回 0，失败为-1。该函数定义的 errno 错误代码如下。

EINVAL　　参数 signum 无效，或试图捕获及忽略 SIGKILL 或 SIGSTOP 信号。

sigaddset

〖功能〗该函数在信号集合 set 中添加信号 signum 。函数所做的就是修改 set；不会阻塞或启动任何信号。

〖原型〗int sigaddset (sigset_t *set, int signum)

〖位置〗signal.h (POSIX.1)

〖说明〗成功时函数返回 0，失败返回-1。以下是该函数定义的 errno 错误代码。

EINVAL　　参数 signum 未指定有效的信号。

sigaltstack

〖功能〗该函数指定在信号处理中使用的备用堆栈。

〖原型〗int sigaltstack (const struct sigaltstack *stack, struct sigaltstack *oldstack)

〖位置〗signal.h (BSD)

〖说明〗当进程接收到信号并且其动作说明已经使用信号堆栈时，系统将安排在执行该信号的句柄时切换到新安装的信号堆栈中。

如果 oldstack 不是一个空指针，则将在其指向的位置上返回新安装的信号堆栈的有关信息。如果 stack 不是一个空指针，则安装这一新堆栈，作为信号句柄使用。

成功则函数返回 0，失败返回-1。如果函数失败，则将 errno 设置为如下错误代码中的一个。

EINVAL　　试图禁用一个实际上正在使用的堆栈。

ENOMEM　　备用堆栈太小。必须大于 MINSIGSTKSZ。

sigblock

〖功能〗该函数等价于 how 参数等于 SIG_SETMASK 的 sigprocmask 函数：设置调用进程的信号掩码为 mask。返回值是上一组阻塞信号。

〖原型〗int sigblock (int mask)

〖位置〗signal.h (BSD)

sigdelset

〖功能〗该函数从信号集合 set 中删除信号 signum。

〖原型〗int sigdelset (sigset_t *set, int signum)

〖位置〗signal.h (POSIX.1)

〖说明〗sigdelset 所做的就是更改 set，它不会阻塞或启动任何信号。返回值和错误代码与 sigaddset 相同。

sigemptyset

〖功能〗该函数初始化 set，拒绝接纳所有已经定义的信号。该函数通常返回 0。

〖原型〗int sigemptyset (sigset_t *set)

〖位置〗signal.h (POSIX.1)

sigfillset

〖功能〗该函数初始化信号集合 set，其中包含所有定义的信号，并且函数通常返回 0。

〖原型〗int sigfillset (sigset_t *set)

〖位置〗signal.h (POSIX.1)

siginterrupt

〖功能〗当信号 signum 中断某种初始状态时，该函数指定调用哪一种方法。如果 failflag 为假，则 signum 重置初始状态。如果 failflag 为真，则 signum 将导致这些初始状态产生错误代码为 EINIR 的错误。

〖原型〗int siginterrupt (int signum, int failflag)

〖位置〗signal.h (BSD)

sigismember

〖功能〗该函数用于测试信号 signum 是否为信号集合 set 的成员。如果信号位于集合中则返回 1，否则返回 0，出错返回-1。

〖原型〗int sigismember (const sigset_t *set, int signum)

〖位置〗signal.h (POSIX.1)

〖说明〗该函数定义的 errno 错误代码如下。

EINVAL　　参数 signum 指定的 signal 无效。

signal

〖功能〗该函数设定信号 signum 执行的动作。

〖原型〗sighandler_t signal (int signum, sighandler_t action)

〖位置〗signal.h (ISO)

〖说明〗第 1 个参数 signum 说明需要控制其行为的信号，它必须是一个信号量。指定信号量的正确方法就是使用一个符号信号名，而不要使用明确的数字，因为在不同的操作系统中，每类信号的数字代码都不同。

第 2 个参数 action 指定信号 signum 使用的动作，可以使用如下值。

SIG_DFL	SIG_DFL 说明特定信号的默认动作。
SIG_IGN	SIG_IGN 说明信号应当忽略。程序通常不会忽略代表重要事件的信号，或者通常用来请求中断的信号。因此不能忽略 SIGKILL 或者 SIGSTOP 信号。可以忽略类似 SIGSEGV 的程序错误信号，但是忽略错误就使得程序无法继续有意义的执行。忽略类似 SIGINT、SIGQUIT 和 SIGTSTP 这样的用户请求是不友好的表现。当用户在程序的某一部分不希望传递信号时，最好阻塞信号，而不是忽略信号。
句柄	提供程序中处理函数的地址，按照传递信号的方法来运行这一句柄。

如果设置某个信号的动作为 SIG_IGN，或者设置为 SIG_DFL 并且默认动作为忽略该信号，那么这种类型的任何挂起信号都将被丢弃(即便被阻塞信号也一样)。丢弃挂起信号就意味着它们无法再次传递，即便再次指定另一个动作并且启动这类信号也不行。

signal 函数返回指定的信号 signum 前次生效的动作。可以保存该值，并且通过再次调用 signal 函数恢复。

如果 signal 函数不能完成请求，则返回 SIG_ERR 错误。该函数定义的 errno 错误代码如下。

EINVAL　　指定了无效的 signum，或者试图忽略信号或为 SIGKILL 或者 SIGSTOP 提供句柄。

以下的简单样例说明在发生某些致命信号时，如何建立句柄来删除临时文件。

```
#include <signal.h>
void
termination_handler (int signum)
{
  struct temp_file *p;
  for (p = temp_file_list; p; p = p->next)
    unlink (p->name);
}
int
main (void)
{
```

```
    ...
    if (signal (SIGINT, termination_handler) == SIG_IGN)
      signal (SIGINT, SIG_IGN);
    if (signal (SIGHUP, termination_handler) == SIG_IGN)
      signal (SIGHUP, SIG_IGN);
    if (signal (SIGTERM, termination_handler) == SIG_IGN)
      signal (SIGTERM, SIG_IGN);
    ...
}
```

sigpause

〖功能〗该函数等价于函数 sigsuspend。将调用进程的信号掩码设置为 mask，并等待信号的到达。返回值是前一阻塞信号集合。

〖原型〗int sigpause (int mask)

〖位置〗signal.h (BSD)

sigpending

〖功能〗该函数在 set 中保存挂起信号的有关信息。

〖原型〗int sigpending (sigset_t *set)

〖位置〗signal.h (POSIX.1)

〖说明〗如果一个挂起信号被传输阻塞，则该信号就是返回集合的一个成员(可以使用 sigismember 检查某个信号是否是该集合的成员)。

成功时函数返回 0，否则返回-1。

测试一个信号是否挂起通常没什么用处。而检查信号何时被启用几乎可以肯定是低级而错误的设计。

下面提供一个样例。

```
#include <signal.h>
#include <stddef.h>
sigset_t base_mask, waiting_mask;
sigemptyset (&base_mask);
sigaddset (&base_mask, SIGINT);
sigaddset (&base_mask, SIGTSTP);
/* Block user interrupts while doing other processing. */
sigprocmask (SIG_SETMASK, &base_mask, NULL);
...
/* After a while, check to see whether any signals are pending. */
sigpending (&waiting_mask);
if (sigismember (&waiting_mask, SIGINT)) {
  /* User has tried to kill the process. */
}
else if (sigismember (&waiting_mask, SIGTSTP)) {
  /* User has tried to stop the process. */
}
```

记住如果进程中的某个信号挂起，那么同时到达的同一类别的其他信号也可能被丢弃。例如，如果一个 SIGINT 信号到达时另一个 SIGINT 信号正被挂起，那么启用该信号时程序只能看到其中一个。

可移植性说明：sigpending 函数是 POSIX.1 中的新内容。旧系统中没有相应的工具。

sigprocmask

〖功能〗该函数用于检查或者修改调用进程的信号掩码。

〖原型〗int sigprocmask (int how, const sigset_t *set, sigset_t *oldset)

〖位置〗signal.h (POSIX.1)

〖说明〗参数 how 决定如何改变信号掩码，必须是如下值。

SIG_BLOCK	阻塞 set 中的信号，将其添加到现在的掩码中。也就是说，新的掩码是旧掩码和 set 的联合。
SIG_UNBLOCK	启用 set 中的信号，将其从现在的掩码中删除。
SIG_SETMASK	使用 set 作为掩码，忽略掩码的原值。

最后一个参数 oldset 用来返回旧进程信号掩码的有关信息。如果仅仅需要更改掩码而不需要了解掩码，则为 oldset 参数传递一个空指针。同样如果需要了解掩码而不希望改变它，则为 set 参数传递一个空指针(这种情况下参数 how 就不重要了)。参数 oldset 经常用来保存原有的信号掩码，以便于今后再次恢复(由于信号掩码从 fork 和 exec 调用中继承，因此程序开始运行时就无法预测其内容)。

如果调用该函数导致其他任何挂起信号被启用，那么这些信号中至少有一个将在函数返回之前传递给进程。挂起信号的传递顺序不确定，但是可以多次调用 sigprocmask 函数，每次启用一个不同的信号，从而强制控制顺序。

sigprocmask 函数成功返回 0，错误返回-1。该函数定义的 errno 错误代码如下。

EINVAL	参数 how 无效。

不能阻塞 SIGKILL 和 SIGSTOP 信号，但是如果信号集合中包含它们，那么函数将忽略这两个信号，并且不会返回错误状态。

sigsetmask

〖功能〗该函数与 how 参数等于 SIG_SETMASK 的 sigprocmask 函数等价。将调用进程的信号掩码设置为 mask。返回值是原阻塞信号的集合。

〖原型〗int sigsetmask (int mask)

〖位置〗signal.h (BSD)

sigstack

〖功能〗该函数在信号处理过程中定义一个备用堆栈。当进程收到一个信号且指出堆

栈已使用时，系统就在信号处理的同时安排切换至新安装的信号堆栈。

〖原型〗int sigstack (const struct sigstack *stack, struct sigstack *oldstack)

〖位置〗signal.h (BSD)

〖说明〗如果 oldstack 不是一个空指针，有关新安装信号堆栈的信息将在它指向的位置中返回。如果 stack 是非空指针，则就象给信号句柄使用新堆栈一样安装备用堆栈。

成功时函数返回 0，失败时为-1。

sigsuspend

〖功能〗该函数使用 set 替换进程的信号掩码，然后挂起进程，直至动作为终止进程或者调用信号处理函数的信号开始传递为止。

〖原型〗int sigsuspend (const sigset_t *set)

〖位置〗signal.h (POSIX.1)

〖说明〗也就是说，在第一个非 set 成员的信号到达之前，程序被有效挂起。

如果进程被调用处理函数的信号传递方唤醒，并且处理函数返回，则 sigsuspend 函数也返回。

在 sigsuspend 函数等待期间，掩码始终保持为 set。函数 sigsuspend 返回时将恢复原来的信号掩码。

该函数的返回值和错误代码与 pause 相同。

使用 sigsuspend 函数可以替换 pause 或者 sleep 循环，并且完全可靠，请看下面的样例。

```
sigset_t mask, oldmask;
...
/* Set up the mask of signals to temporarily block. */
sigemptyset (&mask);
sigaddset (&mask, SIGUSR1);
...
/* Wait for a signal to arrive. */
sigprocmask (SIG_BLOCK, &mask, &oldmask);
while (!usr_interrupt)
  sigsuspend (&oldmask);
sigprocmask (SIG_UNBLOCK, &mask, NULL);
```

代码的最后一部分需要一些技巧。当 sigsuspend 返回时，将进程的信号掩码重置为调用 sigsuspend 之前的原值，此时信号 SIGUSR1 再次阻塞。第 2 次调用 sigprocmask 是为了强制启用该信号。

另外，读者可能认为代码中使用 while 循环没有必要，因为程序很明显只等待一个 SIGUSR1 信号。答案是传递给 sigsuspend 的掩码允许进程在传递其他种类信号时被唤醒，例如任务控制信号。如果进程被某个没有设置 usr_interrupt 的信号唤醒，就将再次挂起自身，直至恰当的种类的信号到达为止。

这一技术需要几行代码做准备，但是对于需要使用的各种等待标准却只需执行一次。实际上真正用于等待的代码只有 4 行。

sigvec

〖功能〗该函数与 sigaction 等效，为信号 signum 安装了动作 action，并返回该信号在 old-action 中前一次生效动作的有关信息。

〖原型〗int sigvec (int signum, const struct sigvec *action,struct sigvec *old-action)

〖位置〗signal.h (BSD)

ssignal

〖功能〗与函数 signal 作用相同，仅用于与 SVID 版本的兼容。

〖原型〗sighandler_t ssignal (int signum, sighandler_t action)

〖位置〗signal.h (SVID)

strsignal

〖功能〗该函数返回一个指针，指向包含描述 signum 信息的静态分配字符串。不可更改该串的内容；并且由于它可被后续的调用重写，所以如果以后需要参考则必须保存其副本。

〖原型〗char * strsignal (int signum)

〖位置〗string.h (GNU)

〖说明〗该函数是 GNU 扩展，在 string.h 中说明。

1.20　进程的启动和终止

_exit

〖功能〗令某个进程以 status 状态终止运行的基本函数。

〖原型〗void _exit (int status)

〖位置〗unistd.h (POSIX.1)

〖说明〗调用该函数不会执行 atexit 或者 on_exit 注册的清除函数。

当某个进程处于任何原因终止运行时——无论是明确终止调用，还是由于某个信号量导致终止，都将完成如下动作。

- 关闭该进程打开的所有文件描述符。注意当进程终止时，流不会自动清空。
- 保存返回状态代码的低 8 位，然后通过 wait 或者 waitid 报告给父进程。
- 终止进程的所有子进程都将会分配一个新的父进程(在包括 GNU 在内的大多数系统中，通常分配 init 进程，进程 ID 为 1)。
- 向父进程发送一个 SIGCHLD 信号。
- 如果进程拥有一个控制终端，是会话的前导，那么将向前台任务中的每个进程发送一个 SIGNUP 信号，并且将控制终端与会话脱离关系。
- 如果进程的终止导致某个进程组合孤立，并且导致该进程组的每个成员都停止运行，那么将向该进程组中的每个进程发送一个 SIGHUP 和 SIGCONT 信号。

abort

〖功能〗该函数异常终止一个程序。

〖原型〗void abort (void)

〖位置〗stdlib.h (ISO)

〖说明〗该函数导致程序异常终止。但是没有实现 atexit 或者 on_exit 的清除功能。该函数实际上是通过发出一个 SIGABRT 信号来终止进程的，程序中可以使用某个句柄处理该信号。

atexit

〖功能〗登记在程序正常终止时被调用的函数。

〖原型〗int atexit (void (*function) (void))

〖位置〗stdlib.h (ISO)

〖说明〗该函数登记在程序正常终止时被调用的函数 function，调用的 function 函数没有参数。

atexit 返回值为 0 时表示成功，如果函数不能登记将返回非 0。

程序样例：

```
#include <stdio.h>
#include <stdlib.h>
void bye (void)
{
  puts ("Goodbye, cruel world....");
}
intmain (void)
{
  atexit (bye);
  exit (EXIT_SUCCESS);
}
```

〖参见〗on_exit。

exit

〖功能〗该函数导致进程以 status 状态终止。

〖原型〗void exit (int status)

〖位置〗stdlib.h (ISO)

〖说明〗该函数没有返回值。

正常的终止将导致如下动作。

1. 按照与注册顺序相反的顺序依次调用 atexit 或者 on_exit 函数注册的函数。这一机制允许应用程序各自指定在程序终止时完成的清除动作。通常用来完成类似在某个文件中保存程序状态信息、或者解开共享数据库中的锁定等动作。

2. 关闭所有打开的流，输出所有缓存的输出数据。此外，删除使用 tmpfile 函数打开

的临时文件。

3. 调用_exit 终止程序。

getenv

〖功能〗该函数返回表示环境变量 name 值的字符串。

〖原型〗char * getenv (const char *name)

〖位置〗stdlib.h (ISO)

〖说明〗该字符串不能更改。在某些不使用 GNU 函数库的非 Unix 系统中，或许可以通过随后调用 getenv 覆盖字符串的值(但是不能使用其他函数库函数)。如果环境变量 name 没有定义，则返回值为一个空指针。

getopt

〖功能〗该函数从 argv 和 argc 参数指定的参数列表中得到下一个可选参数。

〖原型〗int getopt (int argc, char **argv, const char *options)

〖位置〗unistd.h (POSIX.2)

〖说明〗通常这些数值可以直接从 main 函数接收到的参数中得到。

参数 options 是一个字符串，指定该程序中有效的可选字符。字符串中的可选字符后可以跟随一个冒号(:)，说明为必需的参数。

如果 options 参数字符串以连字符(-)开头，则需要特殊处理。此时允许将非可选参数作为与可选字符\0 相关联的情况处理，返回非可选参数。

getopt 函数返回可选字符，提供给下一个命令行使用。如果没有可用的可选参数，则返回-1。可能还存在许多非可选的参数，因此 argc 参数必须与外部变量 optind 进行比较。

如果 option 有参数，则 getopt 返回的参数保存在变量 optarg 中。由于 optarg 字符串是 argv 数组中的一个指针，而不是可能被覆盖的静态区域，因此无需拷贝 optarg 字符串的内容。

如果 getopt 发现 argv 中的某个可选字符没有包括在 options 中，或者发现某个遗漏的可选参数，则返回(?)并设置外部变量 optopt 指向实际的可选字符。如果 options 的第 1 个字符是冒号，则 getopt 返回冒号代替问号，说明遗漏可选参数。此外，如果外部变量 opterr 为非 0 值(默认)，则 getopt 将打印错误信息。

getopt_long

〖功能〗该函数从矢量 argv(其长度为 argc)中分解可选内容。

〖原型〗int getopt_long (int argc, char **argv, const char *shortopts, struct option *longopts, int *indexptr)

〖位置〗getopt.h (GNU)

〖说明〗参数 shortopts 用来描述接受的短可选参数，如 getopt。参数 longopts 则用来描述接受的长可选参数。

当 getopt_long 遇到一个短可选参数时，所做的动作与 getop 相同，返回可选参数的字符代码，并且在 optarg 中保存可选参数(如果存在)。

当 getopt_long 遇到一个长可选参数时，根据该参数定义中的 flag 和 val 域采取相应动作。

如果 flag 是一个空指针，则 getopt_long 返回 val 的内容，说明找到了哪一个可选参数。应当在 val 字段中为意义不同的可选参数安排不同的数值，从而可以在 getopt_long 返回后解析这些数值。如果长可选参数与短可选参数等同，则在 val 中使用短可选参数的字符。

如果 flag 不是一个空指针，则表示该可选参数应当在程序中设置一个标记。标记是用户自己定义的一个 int 类型变量。将标记地址存放在 flag 域中。将该可选参数在标记中保存的数值放在 val 域中。这种情况下，getopt_long 返回 0。

对应任何长可选参数，getopt_long 都可以得到数组 longopts 中可选参数定义的索引，该数值保存在*indexptr 中。可以使用 longpts[*indexptr].name 得到可选参数的名称。从而通过 val 域中的数值或者索引区别长可选参数。此外也可以使用这种方法区别设置标记的长可选参数。

当为长可选项提供参数时，getopt_long 返回前将参数值放在变量 optarg 中。当可选项没有提供参数时，optarg 中的数值为一个空指针。这也是区别是否提供可选项参数的方法。

当 getopt_long 再没有可供处理的选项时，返回-1，并且在变量 optind 中保存 argv 中所剩余的下一个参数的索引。

getsubopt

〖功能〗该函数用于管理子选项。

〖原型〗int getsubopt (char **optionp, const char* const *tokens, char **valuep)

〖位置〗stdlib.h (stdlib.h)

〖说明〗参数 optionp 必须是指向某个变量的指针，其中包含需要处理的字符串地址。当函数返回时，更新引用，指向下一个子选项，如果没有后续的子选项则指向终止\0 字符。

参数 tokens 引用一个字符串数组，其中包含已知的子选项。所有的字符串都必须以\0 结尾，并且必须保存一个空指针来表示结束。当 getsubopt 找到一个可能合法的子选项时，将其与 tokens 数组中提供的所有字符串项比较，然后返回字符串中的索引作为标识符。

如果子选项的一个相关数值以=字符开头，则在 valuep 中返回一个指向该数值的指针。字符串以\0 结尾。如果没有可用的参数，则 valuep 设置为空指针。这样调用方可以检查是否给出必需的值，或者是否列出了不需要的值。

如果字符串中的下一个选项没有出现在 tokens 数组中，则 valuep 中将返回子选项的开始地址，并且函数的返回值为-1。

on_exit

〖功能〗该函数实际上就是功能更加强大的 atexit 函数演变体。

〖原型〗int on_exit (void (*function)(int status, void *arg), void *arg)

〖位置〗stdlib.h (SunOS)

〖说明〗函数接收两个参数，函数 function 和任意指针 arg。程序正常结束时，使用两个参数调用 function：传递给 exit 的 status，以及 arg。

为了与 Sun OS 兼容，GNU C 函数库才包含该函数，其他版本不支持。

putenv

〖功能〗该函数在环境中添加或者删除定义。

〖原型〗int putenv (const char *string)

〖位置〗stdlib.h (SVID)

〖说明〗如果 string 的形式类似 name=value，则在环境中添加定义。否则，字符串就被解释为环境变量的名称，并且从环境中删除这一变量的所有定义。

1.21　进　　程

execl

〖功能〗该函数与 execv 类似，但是使用单独指定的 argv 字符串代替了数组。最后一个参数必须使用空指针传递。

〖原型〗int execl (const char *filename, const char *arg0, ...)

〖位置〗unistd.h (POSIX.1)

execle

〖功能〗该函数与 execl 类似，但是允许为新程序明确指定环境。环境参数通过表示最后一个 argv 参数的空指针传递，应当是一个与 environ 变量格式相同的字符串数组。

〖原型〗int execle (const char *filename, const char *arg0, char *const env[], ...)

〖位置〗unistd.h (POSIX.1)

execlp

〖功能〗该函数与 execl 类似，只不过执行的文件名称搜索与 execvp 函数类似。

〖原型〗int execlp (const char *filename, const char *arg0, ...)

〖位置〗unistd.h (POSIX.1)

execv

〖功能〗该函数作为新进程图像执行 filename 指定的文件。

〖原型〗int execv (const char *filename, char *const argv[])

〖位置〗unistd.h (POSIX.1)

〖说明〗argv 参数是一个以空字符结束的字符串数组，用来为即将执行的程序的 main 函数提供 argv 参数的数值。该数组的最后一个元素必须是一个空指针。按照惯例，该数组的第一个元素应当是无目录名称的程序文件名。

execve

〖功能〗该函数与 execv 类似，但是允许使用 env 参数为新程序明确指定环境。该参数应当是与 environ 变量格式相同的一个字符串数组。

〖原型〗int execve (const char *filename, char *const argv[], char *const env[])

〖位置〗unistd.h (POSIX.1)

execvp

〖功能〗该函数与 execv 类似，但是如果 filename 中不包括斜杠，就搜索在 PATH 环境变量中列出的目录，寻找 filename 的完整文件名称。

〖原型〗int execvp (const char *filename, char *const argv[])

〖位置〗unistd.h (POSIX.1)

〖说明〗该函数在执行系统功能程序时非常有用，因为它可以在用户选择的位置中寻找这些程序。命令行程序使用它来执行用户输入的命令。

fork

〖功能〗该函数用于创建一个新进程。

〖原型〗pid_t fork (void)

〖位置〗unistd.h (POSIX.1)

〖说明〗如果操作成功，则 fork 同时返回父进程和子进程，但是各自的数值不同。在子进程中返回数值 0，而同时在父进程中返回子进程的进程 ID。

如果进程创建失败，则 fork 在父进程中返回数值-1。fork 定义的 errno 错误代码如下。

EAGAIN	没有足够的系统资源创建进程，或者用户正在运行的进程过多。说明已经超出 RLIMIT_NPROC 资源限制，通常可以增加该数值。
ENOMEM	系统不能提供进程需要的空间。

子进程与父进程不同的属性如下。

- 子进程有自己惟一的进程 ID。
- 子进程的父进程 ID 就是其父进程的进程 ID。
- 子进程可以得到其父进程打开文件描述符的拷贝。之后父进程中文件描述符的属性修改将不会影响到子进程的文件描述符，两者相互独立。但是两个进程共享与每个描述符相关联的文件位置。
- 子进程的已用处理器时间设置为 0。
- 子进程不继承由父进程设置的文件锁定。
- 子进程不继承由父进程设置的告警。
- 清空子进程的挂起信号设置。

getpid

〖功能〗该函数返回当前进程的进程 ID。

〖原型〗pid_t getpid (void)

〖位置〗unistd.h (POSIX.1)

getppid

〖功能〗该函数返回当前进程的父进程的进程 ID。

〖原型〗pid_t getppid (void)

〖位置〗unistd.h (POSIX.1)

system

〖功能〗该函数按照 shell 指令执行 command。在 GNU C 库中，常用默认的 shell sh 运行该指令。

〖原型〗int system (const char *command)

〖位置〗stdlib.h (ISO)

〖说明〗详细的说，搜索 PATH 中的目录找出程序来执行。如果不能创建一个 shell 进程返回-1，否则返回 shell 进程的状态。

system 在头文件 stdlib.h 中声明。

vfork

〖功能〗该函数与 fork 类似，但是效率更高。

〖原型〗pid_t vfork (void)

〖位置〗unistd.h (BSD)

〖说明〗要想安全使用该函数，必须遵守几条约束。

尽管 fork 完全复制调用进程的地址空间，并允许父进程和子进程独立执行，但是 vfork 没有执行这样的复制。vfork 创建的子进程在调用 exits 或者某个 exec 函数之前，始终与父进程共享地址空间。此时，父进程挂起执行。

必须小心避免使用 vfork 创建的子进程修改任何全局数据，以及与父进程共享的本地变量。此外，子进程不能从调用 vfork 的函数中返回(或者跳出)。这样会搞乱父进程的控制信息。如果不知所从，就不如使用 fork。

有些操作系统柄没有真正执行 vfork。GNU C 函数库允许在所有系统上使用 vfork，但是如果 vfork 不可用，实际执行的是 fork 函数。如果使用 vfork 时小心谨慎，那么即便系统使用 fork 代替，程序也仍然可以运行。

wait

〖功能〗wait 函数是 waitpid 函数的简化版本，用来等待任意一个子进程终止。

〖原型〗pid_t wait (int *status-ptr)

〖位置〗sys/wait.h (BSD)

〖说明〗调用

```
wait (&status)
```

等效于调用

```
waitpid(-1, &status, 0)
```

wait3

〖功能〗wait3 函数是 wait4 函数的前任，后者更加灵活，目前 wait3 函数已经被废弃。

〖原型〗pid_t wait3 (union wait *status-ptr, int options, struct rusage *usage)

〖位置〗sys/wait.h (BSD)

〖说明〗如果 usage 为空指针，则 wait3 等效于 waitpid(-1, status-ptr, options)。

如果 usage 非空，则 wait3 函数在*rusage 中保存子进程的使用图表(仅在子进程终止时，不包括子进程停止时)。

wait4

〖功能〗如果 usage 是空指针，则 wait4 等效于 waitpid(pid, status-ptr, options)。

〖原型〗pid_t wait4 (pid_t pid, int *status-ptr, int options, struct rusage *usage)

〖位置〗sys/wait.h (BSD)

〖说明〗如果 usage 非空，则 wait4 函数在*rusage 中保存子进程的使用图表(仅在子进程终止时，不包括子进程停止时)。

该函数是 BSD 版本的扩展。

waitpid

〖功能〗该函数用于请求进程 ID 等于 pid 的子进程的状态信息，通常在子进程终止，然后提供状态信息之前调用进程始终挂起。

〖原型〗pid_t waitpid (pid_t pid, int *status-ptr, int options)

〖位置〗sys/wait.h (POSIX.1)

〖说明〗pid 参数的其他值都有特殊解释。数值-1 或者 WAIT_ANY 请求任何子进程的状态信息，数值 0 或者 WAIT_MYPGRP 请求与调用进程处于同一个进程组内的任何子进程信息，而其他所有负值-pgid 都请求进程组 ID 等于 pgid 的子进程信息。

如果立即得到某个子进程的状态信息，则该函数无需等待，立即返回。如果有多个合格子进程提供状态信息，则随机选择其中一个，并立即返回其状态。要得到其他合格子进程的状态，就需要再次调用 waitpid。

返回值通常就是报告状态信息的子进程的进程 ID。如果指定 WNOHANG 选项，并且没有等待子进程，则返回 0。出现错误返回-1。该函数还定义了如下 errno 错误代码。

EINTR	函数被调用进程的信号传递中断执行。
ECHILD	没有等待的子进程，或者指定的 pid 不是调用进程的子进程。
EINVAL	参数 options 的值无效。

1.22 任 务 控 制

ctermid

〖功能〗该函数返回一个字符串，其中包含当前进程控制终端的文件名称。

〖原型〗char * ctermid (char *string)

〖位置〗stdio.h (POSIX.1)

〖说明〗如果 string 不是一个空指针，则应当是一个可以至少保存 L_ctermid 个字符的数组，字符串在该数组中返回。否则就返回一个指针，指向静态区域内的一个字符串，下一次调用该函数时覆盖其内容。

如果由于任何原因导致无法判断出文件名称，则返回一个空字符串。即便返回一个文件名称，也不允许访问该文件。

getpgrp

〖功能〗该函数返回进程 pid 的进程组 ID。

〖原型〗pid_t getpgrp (pid_t pid)

〖位置〗unistd.h (BSD)

〖说明〗为参数 pid 提供数值 0 可以得到调用进程的有关信息。

getpgrp

〖功能〗该函数返回进程 pid 的进程组 ID。

〖原型〗pid_t getpgrp (pid_t pid)

〖位置〗unistd.h (BSD)

〖说明〗为参数 pid 提供数值 0 可以得到调用进程的有关信息。

setpgid

〖功能〗该函数将进程 pid 置于进程组 pgid 中。作为特殊情况，如果 pid 或 pgid 为 0 则说明调用进程的 ID。

〖原型〗int setpgid (pid_t pid, pid_t pgid)

〖位置〗unistd.h (POSIX.1)

〖说明〗当系统不支持任务控制时函数失败。

如果运行成功 setpgid 返回 0 值；否则返回-1，该函数定义的 errno 错误代码如下。

EACCES	由 pid 命名的子进程自从交叉时起，就执行 exec 函数。
EINVAL	pgid 值无效。
ENOSYS	系统不支持任务控制。
EPERM	参数 pid 指定的进程是会话领导，或者与调用进程处在不同的会话中，或者参数 pgid 的值与进程组 ID 不匹配，而该进程组 ID 与调用进程处于同一个会话中。
ESRCH	参数 pid 指定的进程不是调用进程或者调用进程的子进程。

setpgrp

〖功能〗该函数是函数 setpgid 的 BSD Unix 名，二者所执行的动作完全相同。

〖原型〗int setpgrp (pid_t pid, pid_t pgid)

〖位置〗unistd.h (BSD)

setsid

〖功能〗该函数用于创建新的会话。

〖原型〗pid_t setsid (void)

〖位置〗unistd.h (POSIX.1)

〖说明〗调用进程成为会话领导，并且放在一个新的进程组内，该进程组 ID 与进程自身的 ID 相同。最初新进程组中没有其他进程，新会话中也没有其他进程组。

该函数使得调用进程失去控制中断。

setsid 函数成功时返回调用进程的新进程组 ID，返回值-1 表示错误。该函数定义的 errno 错误代码如下。

EPERM	调用进程已经成为进程组领导，或者周围已经存在其他的进程组与进程组 ID 重名。

tcgetpgrp

〖功能〗该函数返回进程组 ID，该进程组是与描述符 filedes 上打开的终端相关联的前台进程组。

〖原型〗pid_t tcgetpgrp (int filedes)

〖位置〗unistd.h (POSIX.1)

〖说明〗如果没有前台进程组，则返回值是一个大于 1 的数字，不与任何已有进程组的 ID 相匹配。如果曾经作为前台任务的所有任务进程都已经中断，并且也没有其他任务被移到前台，就可能发生上述情况。

如果出现错误，则返回-1。该函数定义的 errno 错误代码如下。

EBADF	参数 filedes 不是一个有效的文件描述符。
ENOSYS	系统不支持任务控制。
ENOTTY	与参数 filedes 相关联的终端文件不是调用进程的控制终端。

tcsetpgrp

〖功能〗设置终端的前台进程组 ID。参数 filedes 是定义终端的描述符，pgid 指定进程组。调用进程必须是与 pgid 相同的 session 的成员且有相同的控制终端。

〖原型〗int tcsetpgrp (int filedes, pid_t pgid)

〖位置〗unistd.h (POSIX.1)

〖说明〗对于终端访问该函数可看作输出。如果从它控制的终端后台处理调用，通常向进程组中所有进程发送一个 SIGTTOU 信号。除非调用进程本身正忽略或阻塞 SIGTTOU 信号。此时执行操作并不发出任何信号。

如果成功，则 tcsetpgrp 返回 0；返回值-1 表示错误。以下是该函数定义的 errno 错误代码。

EBADF	filedes 不是有效的文件描述符。
EINVAL	pgid 无效。
ENOSYS	系统不支持工作控制。
ENOTTY	filedes 不是调用进程的控制终端。
EPERM	pgid 不是与调用进程在相同 session 中的进程组。

1.23　用户和群组

cuserid

〖功能〗该函数返回用户 ID。

〖原型〗char * cuserid (char *string)

〖位置〗stdio.h (POSIX.1)

〖说明〗cuserid 函数返回一个指针，指向字符串，说明与进程有效 ID 相关联的用户名。如果 string 不是一个空指针，就应当是一个可以至少保存 L_cuserid 个字符的数组，字符串在该数组中返回。否则将返回一个指针，指向静态区域中的一个字符串。该字符串静态分配，下一次调用该函数或者 getlogin 时将覆盖其内容。

自从该函数在 XPG4.2 中被单独标注之后，就不赞成使用它，在 POSIX.1 中该函数已经被删除。

endgrent

〖功能〗该函数将关闭由 getgrent 或者 getgrent_r 使用的内部流。

〖原型〗void endgrent (void)

〖位置〗grp.h (SVID, BSD)

endnetgrent

〖功能〗该函数将释放所有分配去处理最后一个选定网络组的缓存区，此后所有调用 getnetgrent 返回的字符串指针全部无效。

〖原型〗void endnetgrent (void)

〖位置〗netdb.h (netdb.h)

endpwent

〖功能〗该函数将关闭由 getpwent 或者 getpwent_r 使用的内部流。

〖原型〗void endpwent (void)

〖位置〗pwd.h (SVID, BSD)

fgetgrent

〖功能〗该函数将从 stream 中读入下一项。

〖原型〗struct group * fgetgrent (FILE *stream)

〖位置〗grp.h (SVID)

〖说明〗返回指向下一项的指针。结构为静态分配，因此下一次调用 fgetgrent 将覆盖此次的结果。如果需要保存得到的信息，就必须复制结构的内容。

指定的流必须与标准组数据库格式的文件相对应。

fgetgrent_r

〖功能〗该函数读入 stream 中的下一个用户项，与 fgetgrent 类似，但是结构中返回的结果由 result_buf 指定。

〖原型〗int fgetgrent_r (FILE *stream, struct group *result_buf, char *buffer, size_t buflen, struct group **result)

〖位置〗grp.h (GNU)

〖说明〗buffer 指向的额外缓存中第 1 个 buflen 字节用来保存额外信息，通常就是结构中元素指向的字符串。

指定的流必须与标准组数据库格式的文件相对应。

如果函数返回 0，则 result 指向包含所需数据的结构(通常位于 result_buf 中)。如果出现错误，则返回值为非 0 值，并且 result 中包含空指针。

使用 setgrent、getgrent 和 endgrent 可以在组数据库中扫描所有项。

fgetpwent

〖功能〗该函数读入 stream 中的下一个用户项，然后返回指向它的指针。

〖原型〗struct passwd * fgetpwent (FILE *stream)

〖位置〗pwd.h (SVID)

〖说明〗该结构为静态分配，因此下一次调用 fgetpwent 时将覆盖其内容。如果需要保存信息，则必须复制结构的内容。

流必须与标准口令数据库文件格式的文件相对应，该函数来自于系统 V。

fgetpwent_r

〖功能〗该函数读入 stream 中的下一个用户项，与 fgetpwent 类似。但是结果由 result_buf 指向的结构返回。

〖原型〗int fgetpwent_r (FILE *stream, struct passwd *result_buf, char *buffer, size_t buflen, struct passwd **result)

〖位置〗pwd.h (GNU)

〖说明〗buffer 指向的额外缓存中第 1 个 buflen 字节用来保存额外信息，通常就是结构中元素指向的字符串。

流必须与标准口令数据库格式的文件相对应。

如果函数返回空，则 result 指向包含所需数据的结构(通常位于 result_buf 中)。如果出现错误，则返回值为非空值，并且 result 中包含空指针。

使用 setpwent、getpwent 和 endpwent 可以在组数据库中扫描所有项。

getegid

〖功能〗该函数返回进程的有效组 ID。
〖原型〗gid_t getegid (void)
〖位置〗unistd.h (POSIX.1)

geteuid

〖功能〗该函数返回进程的有效用户 ID。
〖原型〗uid_t geteuid (void)
〖位置〗unistd.h (POSIX.1)

getgid

〖功能〗该函数返回进程的真实组 ID。
〖原型〗gid_t getgid (void)
〖位置〗unistd.h (POSIX.1)

getgrent

〖功能〗该函数从 setgrent 初始化的流中读入下一项。
〖原型〗struct group * getgrent (void)
〖位置〗grp.h (SVID, BSD)
〖说明〗返回指向该项的指针。该结构为静态分配，随后再次调用 getgrent 时将覆盖原值。如果需要保存信息，则必须拷贝结构的内容。

getgrent_r

〖功能〗该函数与 getgrent 类似，返回 setgrent 初始化的流中的下一项内容。
〖原型〗int getgrent_r (struct group *result_buf, char *buffer, size_t buflen, struct group **result)
〖位置〗grp.h (GNU)
〖说明〗但是由于结构放在 result_buf 指向的用户结构中，因此与 getgrent 函数相比，该函数的返回值可以重复读取。额外的数据通常由结果结构中的元素引用，放在从 buffer 开始长为 buflen 的缓存区中。

如果函数返回 0，则 result 指向保存有所需数据的结构(通常位于 result_buf 中)。如果出现错误则返回一个非 0 值，则 result 中包含一个空指针。

getgrgid

〖功能〗该函数返回一个指针，指向一个静态分配的结构，其中包含有关 ID 为 gid 的组信息。
〖原型〗struct group * getgrgid (gid_t gid)
〖位置〗grp.h (POSIX.1)

〖说明〗随后再次调用 getgrgid 将覆盖结构的原值。

返回空指针说明没有 ID 等于 gid 的组。

getgrgid_r

〖功能〗该函数与 getgrgid 类似，返回组 ID 为 gid 的组信息，但是结构不放在静态缓存中。

〖原型〗int getgrgid_r (gid_t gid, struct group *result_buf, char *buffer, size_t buflen, struct group **result)

〖位置〗grp.h (POSIX.1c)

〖说明〗在用户提供的结构中返回信息，该结构由 result_buf 引用。buffer 指向的额外缓存中，使用前 buflen 字节来保存补充信息，通常这一字符串由结构结构中的元素引用。

如果返回值等于 0，则 result 返回的指针指向的记录中包含有所需数据(例如，result 中包含 result_buf 的数值)。如果返回一个非 0 值，则说明数据库中没有组 ID 为 gid 的组，或者缓存区 buffer 的空间太小，不能保存所需的全部信息。后一种情况下，将全局变量 errno 设置为 ERANGE。

getgrnam

〖功能〗该函数返回一个指向静态分配结构指针，其中包含名为 name 的组信息。

〖原型〗struct group * getgrnam (const char *name)

〖位置〗grp.h (SVID, BSD)

〖说明〗随后再次调用 getgrnam 将覆盖结构中的原值。

返回空指针说明不存在名为 name 的组。

getgrnam_r

〖功能〗该函数与 getgrname 类似，返回名为 name 的组信息，但结构不是放在静态缓存中。

〖原型〗int getgrnam_r (const char *name, struct group *result_buf, char *buffer, size_t buflen, struct group **result)

〖位置〗grp.h (POSIX.1c)

〖说明〗使用 result_buf 引用的用户提供结构来存放返回信息。buffer 指向的额外缓存区中，前 buflen 字节用于保存补充信息，通常就是结果结构的元素指向的字符串。

如果返回值等于 0，则 result 返回的指针指向的记录中包含有所需数据(例如，result 中包含 result_buf 的数值)。如果返回一个非 0 值，则说明数据库中没有名为 name 的组，或者缓存区 buffer 的空间太小，不能保存所需的全部信息。后一种情况下，将全局变量 errno 设置为 ERANGE。

getgroups

〖功能〗该函数用来查询进程的辅助组 ID。

〖原型〗int getgroups (int count, gid_t *groups)

〖位置〗unistd.h (POSIX.1)

〖说明〗数组 groups 中可以保存最多 count 个这类组 ID，函数的返回值为实际保存的组 ID 数目。如果 count 小于补充组 ID 的总数目，则 getgroups 返回数值-1，并设置 errno 为 EINVAL。

如果 count 等于 0，则 getgroups 返回补充组 ID 的总数目。在不支持补充组的系统中，通常该数目为 0。

以下是一个如何使用 getgroups 读入所有补充组 ID 的样例。

```
gid_t *
read_all_groups (void)
{
  int ngroups = getgroups (0, NULL);
  gid_t *groups
    = (gid_t *) xmalloc (ngroups * sizeof (gid_t));
  int val = getgroups (ngroups, groups);
  if (val < 0)
    {
      free (groups);
      return NULL;
    }
  return groups;
}
```

getlogin

〖功能〗该函数返回的指针指向一个字符串，其中包含在进程的控制终端上登录的用户名，如果信息无法确定则返回一个空指针。

〖原型〗char * getlogin (void)

〖位置〗unistd.h (POSIX.1)

〖说明〗字符串是静态分配的，下一次调用该函数或者 cuserid 时将覆盖字符串内容。

getnetgrent

〖功能〗该函数返回当前选定网络组中的下一个未处理项。

〖原型〗int getnetgrent (char **hostp, char **userp, char **domainp)

〖位置〗netdb.h (netdb.h)

〖说明〗字符串指针的地址由参数 hostp、userp 和 domainp 传递，成功调用之后将保存相应字符串指针。如果下一项中的字符串为空，则指针为 NULL。只有调用与网络组相关的函数，返回的字符串指针才有效。

如果成功读入下一项，则返回值为 1。数值 0 意味着不存在下一项，或者出现内部错误。

getnetgrent_r

〖功能〗该函数与 getnetgrent 类似，但是 3 个字符串指针 hostp、userp 和 domainp 指

向的字符串放在缓存中从 buffer 开始的 buflen 个字节中。

〖原型〗int getnetgrent_r (char **hostp, char **userp, char **domainp, char *buffer, int buflen)

〖位置〗netdb.h (netdb.h)

〖说明〗即便调用其他与网络组有关的函数，返回值仍然有效。

如果成攻读入下一项并且缓存中有足够的空间存放该字符串，则返回值为 1。如果没有找到其他项、缓存过小、或者出现内部错误则返回 0。

该函数为 GNU 的扩展。在 Sun OS 函数库中的原型不包括该函数。

getpwent

〖功能〗该函数读入 setpwent 初始化的流中的下一项内容。

〖原型〗struct passwd * getpwent (void)

〖位置〗pwd.h (POSIX.1)

〖说明〗返回一个指针，指向下一项内容。

该结构为静态分配，因此下一次调用 getpwent 将覆盖其内容。如果需要保存信息，则必须拷贝结构的内容。

如果没有可用项则返回一个空指针。

getpwent_r

〖功能〗该函数与 getpwent 类似，返回 setpwent 初始化的流中的下一项内容。

〖原型〗int getpwent_r (struct passwd *result_buf, char *buffer, int buflen, struct passwd **result)

〖位置〗pwd.h (GNU)

〖说明〗但是与 getpwent 函数相比，该函数是可重入的，因为结果放在用户提供的结构中，由 result_buf 引用。补充数据通常由结果结构中的元素引用，放在补充缓存中，或者从 buffer 开始 buflen 长的字节中。

如果函数返回 0，则 result 指向的结构中包含有所需的数据(通常位于 result_buf 中)。如果出现错误则返回非 0 值，并且 result 是一个空指针。

getpwnam

〖功能〗该函数返回一个指针指向一个静态分配的结构，其中包含用户名为 name 的用户信息。

〖原型〗struct passwd * getpwnam (const char *name)

〖位置〗pwd.h (POSIX.1)

〖说明〗下一次调用 getpwnam 将覆盖该结构。

空指针说明不存在名为 name 的用户。

getpwnam_r

〖功能〗该函数与 getpwnam 类似，返回名为 name 的用户信息。

〖原型〗int getpwnam_r (const char *name, struct passwd *result_buf, char *buffer, size_t buflen, struct passwd **result)

〖位置〗pwd.h (POSIX.1c)

〖说明〗但是返回的结果不放在静态缓存中。而是将信息放在用户提供的结构中，由 result_buf 引用。buffer 指向的额外缓存的前 buflen 个字节用来保存补充信息，该字符串通常由结果结构中的元素引用。

如果返回值为 0，则 result 中返回指针指向的记录包含有所需数据(例如 result 中包含 result_buf)。如果返回值非空，则说明数据库中没有名为 name 的用户，或者缓存 buffer 无法包含全部所需信息。后一种情况下全局变量 errno 设置为 ERANGE。

getpwuid

〖功能〗该函数返回一个指针指向静态分配结构，其中包含用户 ID 为 uid 的用户信息。

〖原型〗struct passwd * getpwuid (uid_t uid)

〖位置〗pwd.h (POSIX.1)

〖说明〗下一次调用 getpwuid 将覆盖结构内容。

空指针说明数据库中没有用户 ID 为 uid 的用户。

getpwuid_r

〖功能〗该函数与 getpwuid 类似，返回用户 ID 为 uid 的用户信息。

〖原型〗int getpwuid_r (uid_t uid, struct passwd *result_buf, char *buffer, size_t buflen, struct passwd **result)

〖位置〗pwd.h (POSIX.1c)

〖说明〗但是结果没有放在静态缓存中。而是将信息放在用户提供的结构中，由 result_buf 引用。buffer 指向的额外缓存的前 buflen 个字节用来保存补充信息，该字符串通常由结果结构中的元素引用。

如果返回值为 0，则 result 中返回指针指向的记录包含有所需数据(例如 result 中包含 result_buf)。如果返回值非空，则说明数据库中没有用户 ID 为 uid 的用户，或者缓存 buffer 无法包含全部所需信息。后一种情况下全局变量 errno 设置为 ERANGE。

getuid

〖功能〗该函数返回进程的真实用户 ID。

〖原型〗uid_t getuid (void)

〖位置〗unistd.h (POSIX.1)

initgroups

〖功能〗该函数调用 setgroups 将进程的补充组 ID 设置为用户 user 的默认值，组 ID gid 将在参数中提供。

〖原型〗int initgroups (const char *user, gid_t gid)

〖位置〗grp.h (BSD)

innetgr

〖功能〗该函数检测参数 hostp、userp 和 domainp 指定的组合是否是网络组 netgroup 的一部分。

〖原型〗int innetgr (const char *netgroup, const char *host, const char *user, const char *domain)

〖位置〗netdb.h (netdb.h)

〖说明〗该函数的优点如下。

1. 由于使用内部锁定，因此其他网络组函数不能使用全局网络组状态。

2. 与连续调用其他 set/get/endnetgrent 函数相比较，使用该函数的效率更高。

指针 hostp、userp 和 domainp 都可以为 NULL，表示这一位置不认同任何数值。名称也同样可以为空，但是就无法与任何字符串匹配。

如果在网络组中发现与指定组和匹配的项，则返回 1。如果没有找到网络组，或者网络组中不存在匹配项或者出现内部错误，则返回 0。

putpwent

〖功能〗该函数按照标准用户数据库文件的格式，将用户项*p 写入 stream 流中，成功返回 0，失败返回非 0 值。

〖原型〗int putpwent (const struct passwd *p, FILE *stream)

〖位置〗pwd.h (SVID)

〖说明〗该函数是为了与 SVID 兼容提供的，建议尽量避免使用，因为仅有在假定 struct passwd 结构中只有标准成员，并且位于混合了标准 Unix 数据库和其他用户扩展信息的系统上时，该函数才有意义，使用该函数添加项将不可避免地遗漏大量重要信息。

setgid

〖功能〗该函数假定进程具备相应的权限，将进程的真实组 ID 和有效组 ID 都设置为 newgid。

〖原型〗int setgid (gid_t newgid)

〖位置〗unistd.h (POSIX.1)

〖说明〗如果进程没有权限，newgid 必须等于真实组 ID 或已存的组 ID。这种情况下，setgid 仅设置有效组 ID 而不设置真实组 ID。

setgid 的返回值和错误情况都和 setuid 相同。

setgrent

〖功能〗该函数初始化一个从群组数据库读入的流，可以通过调用 getgrent 或者 getgrent_r 使用该流。

〖原型〗void setgrent (void)

〖位置〗grp.h (SVID, BSD)

setgroups

〖功能〗该函数设置进程的辅助组 ID，且只能从有权限的程序中调用。参数 count 指定在 groups 数组中组 ID 的数量。

〖原型〗int setgroups (size_t count, gid_t *groups)

〖位置〗grp.h (BSD)

〖说明〗如果成功函数返回 0，否则返回-1。该函数定义的 errno 错误代码如下。

EPERM　　调用进程权限不足。

setnetgrent

〖功能〗该函数初始化函数库的内部状态，使得随后的 getnetgrent 调用在网络组 netgroup 的所有项目内容之间循环。

〖原型〗int setnetgrent (const char *netgroup)

〖位置〗netdb.h (netdb.h)

〖说明〗如果调用成功(例如，使用这个名字的网络组存在)，则返回值是 1。当返回值为 0 时，说明找不到这个名字的网络组或者出现其他的错误。

记住循环网络组仅有一个单独的状态，这一点很重要。即便程序员使用 getnetgrent_r 函数，结果也不是真正可重入的，因为通常同一时刻只能处理一个网络组。如果程序需要同时处理多个网络组，就必须使用外部锁定进行保护。这一问题在 SunOS 中引入了原始的网络组概念，由于需要与之相匹配，因此不能更改。

有些其他的函数也使用网络组状态。

setpwent

〖功能〗该函数初始化使用 getpwent 和 getpwent_r 读取用户数据库的流。

〖原型〗void setpwent (void)

〖位置〗pwd.h (SVID, BSD)

setregid

〖功能〗该函数将进程的真实组 ID 设置为 rgid，将其有效组 ID 设置为 egid。如果 rgid 等于-1，则说明不更改真实组 ID，同样如果 egid 等于-1，则表示不更改有效组 ID。

〖原型〗int setregid (gid_t rgid, fid_t egid)

〖位置〗unistd.h (BSD)

〖说明〗该函数是为了与 4.3 BSD Unix 兼容才提供的，这一 Unix 版本不支持保存 ID。可以使用该函数交换进程的有效组 ID 和真实组 ID(特许进程不仅限于此用途)。如果支持保存 ID，则应当用这一特性来代替该函数。

该函数的返回值和错误情况定义与 setreuid 相同。

GNU 系统也允许特许进程更改其补充组 ID。使用 setgroups 或者 initgroups 时，程序应当包含头文件 grp.h。

setreuid

〖功能〗该函数将进程真实用户 ID 设置为 ruid，并将有效用户 ID 设置为 euid。

〖原型〗int setreuid (uid_t ruid, uid_t euid)

〖位置〗unistd.h (BSD)

〖说明〗如果 ruid 等于-1，则表示不更改真实用户 ID，同样如果 euid 等于-1，则表示不更改有效用户 ID。

该函数是为了与 4.3 BSD Unix 兼容才提供的，这一 Unix 版本不支持保存 ID。可以使用该函数交换进程的有效组 ID 和真实组 ID(特许进程不仅限于此用途)。如果支持保存 ID，则应当用这一特性来代替该函数。

成功返回 0，失败返回-1。函数定义的 errno 错误代码如下。

EPERM	进程没有相应权限，不能更改指定 ID。

setuid

〖功能〗该函数假定进程具备相应权限，将进程的真实和有效用户 ID 都设置为 newuid。

〖原型〗int setuid (uid_t newuid)

〖位置〗unistd.h (POSIX.1)

〖说明〗如果进程不具备权限，newuid 必须等于真实的用户 ID 或已存的用户 ID(如果系统支持_POSIX_SAVED_IDS)。此时，Setuid 仅设置有效用户 ID 而不设置真实用户 ID。

返回 0 表示完全成功，返回-1 表示错误。该函数定义的 errno 出错情况如下。

EINVAL	newuid 参数无效。
EPERM	进程没有适当的权限，不允许改变指定 ID。

1.24 系 统 信 息

gethostid

〖功能〗该函数返回运行程序的机器 host ID。

〖原型〗long int gethostid (void)

〖位置〗unistd.h (BSD)

〖说明〗按照惯例，机器的 host ID 通常作为其主要的互联网地址，并且转换为一个长整型。但是在有些系统中，该值没有意义，只是作为每个机器惟一的硬件代码。

gethostname

〖功能〗该函数在数组 name 中返回主机名。

〖原型〗int gethostname (char *name, size_t size)

〖位置〗unistd.h (BSD)

〖说明〗参数 size 指定数组的大小，以字节为单位。

成功返回 0，失败则返回-1。在 GNU C 函数库中，如果 size 的值不够大则 gethostname 将失败，必须再次使用一个更大的数组运行。该函数定义的 errno 错误代码如下。

ENAMETOLLONG	参数 size 小于主机名的长度加 1。

在某些系统中，使用一个变量 MAXHOSTNAMELEN 表示可能出现的最大主机名长度。该变量在 sys/param.h 中定义。但是不能指望该变量一定存在，应当正确处理可能出现的失败情况，并且再次尝试。

即便 name 数组与主机名的长度不吻合，gethostname 在 name 中保存主机名的开始部分。在某些情况下，截短的主机名就已经可以满足要求。此时完全可以忽略错误代码。

sethostid

〖功能〗该函数设置主机的 host ID 为 id。

〖原型〗int sethostid (long int id)

〖位置〗unistd.h (BSD)

〖说明〗只有具备权限的程序才允许执行这一操作，通常只在系统启动时发生一次。

成功返回 0，失败返回-1。该函数定义的 errno 错误代码如下。

EPERM	由于进程权限不足，因此无法设置主机名。
ENOSYS	操作系统不支持设置主机 ID。在有些系统中主机 ID 没有意义，只是每个机器惟一的硬件编码而已。

sethostname

〖功能〗该函数设置主机名为长度等于 length 的字符串 name。

〖原型〗int sethostname (const char *name, size_t length)

〖位置〗unistd.h (BSD)

〖说明〗只有有权限的程序才可以。通常只在系统启动时发生一次。

成功返回 0，失败返回-1。该函数定义的 errno 错误代码如下。

EPERM	由于进程权限不足，因此无法设置主机名。

uname

〖功能〗该函数在 info 指向的结构中填入操作系统和主机的信息。

〖原型〗int uname (struct utsname *info)

〖位置〗sys/utsname.h (POSIX.1)

〖说明〗非负值说明数据成功存储，-1 表明错误。惟一可能的错误是 EFAULT，通常不提及，因为这一错误经常可能发生。

1.25 系统配置参数

confstr

〖功能〗该函数读入一个字符串系统参数，将该字符串保存在从 BUF 开始长度为 len 字节的内存空间中。PARAMETER 参数应当是如下所列的_cs_符号中的一个。

〖原型〗size_t confstr (int parameter, char *buf, size_t len)

〖位置〗unistd.h (POSIX.2)

〖说明〗正常返回值是请求字符串值的长度。如果为 BUF 提供一个空指针，则 confstr 不会尝试保存字符串，而仅仅返回其长度。返回值 0 表明出现错误。

如果请求的字符串对于缓存区而言过长(即长度大于 len-1)，则 confstr 只保存允许的部分(并且为结尾空字符保留空间)。如果 confstr 的返回值大于或者等于 len，就说明出现了上述情况。

该函数定义了如下 errno 错误代码。

EINVAL　　PARAMETER 是无效值。

目前只能使用 confstr 读取如下参数。

_CS_PATH　　该参数的数值为搜索可执行文件的默认路径。该路径是用户登录后默认拥有的。

fpathconf

〖功能〗该函数与 pathconf 类似，但是该函数使用打开文件描述符代替文件名称，指定请求信息的文件。

〖原型〗long int fpathconf (int filedes, int parameter)

〖位置〗unistd.h (POSIX.1)

〖说明〗该函数定义了如下 errno 错误代码。

EBADF　　参数 filedes 为无效文件描述符。

EINVAL　　parameter 的数值无效，或者指定文件不支持 parameter 参数。

pathconf

〖功能〗该函数用于查询名为 filename 的文件限制。

〖原型〗long int pathconf (const char *filename, int parameter)

〖位置〗unistd.h (POSIX.1)

〖说明〗参数 parameter 应当为随后列出的_PC_常数之一。

pathconf 的正常返回值就是用户请求的数值。如果出现错误或者没有限制则返回数值 -1。后一种情况下不设置 errno，前一种情况下将设置 errno，表明问题原因。因此该函数

的正确使用方法是，调用之前首先在 errno 中保存 0。

除了常见的文件名错误之外，该函数还定义了如下错误代码。

EINVAL　　parameter 的数值无效，或者指定文件不支持 parameter 参数。

以下是在 pathconf 和 fpathconf 中可以用做参数 parameter 的符号常量，数值全部为整数常量。

_PC_LINK_MAX	查询 LINK_MAX 的数值。
_PC_MAX_CANON	查询 MAX_CANON 的数值。
_PC_MAX_INPUT	查询 MAX_INPUT 的数值。
_PC_NAME_MAX	查询 NAME_MAX 的数值。
_PC_PATH_MAX	查询 PATH_MAX 的数值。
_PC_PIPE_BUF	查询 PIPE_BUF 的数值。
_PC_CHOWN_RESTRICTED	查询_POSIX_CHOWN_RESTRICTED 的数值。
_PC_NO_TRUNC	查询_POSIX_NO_TRUNC 的数值。
_PC_VDISABLE	查询_POSIX_VDISABLE 的数值。

sysconf

〖功能〗该函数用于查询运行时间系统参数。

〖原型〗long int sysconf (int parameter)

〖位置〗unistd.h (POSIX.1)

〖说明〗通常 sysconf 返回所需的值，如果执行没有限制或出现错误则返回值为-1。

以下是 sysconf 函数中 parameter 参数可以使用的符号常量。数值都是整型常量，具体地说，是枚举类型的数值。

_SC_ARG_MAX	查询与 ARG_MAX 相关的参数。
_SC_CHILD_MAX	查询与 CHILD_MAX 相关的参数。
_SC_OPEN_MAX	查询与 OPEN_MAX 相关的参数。
_SC_STREAM_MAX	查询与 STREAM_MAX 相关的参数。
_SC_TZNAME_MAX	查询与 TZNAME_MAX 相关的参数。
_SC_NGROUPS_MAX	查询与 NGROUPS_MAX 相关的参数。
_SC_JOB_CONTROL	查询与_POSIX_JOB_CONTROL 相关的参数。
_SC_SAVED_IDS	查询与_POSIX_SAVED_IDS 相关的参数。
_SC_VERSION	查询与_POSIX_VERSION 相关的参数。
_SC_CLK_TCK	查询与 CLOCKS_PER_SEC 相关的参数。
_SC_2_C_DEV	查询系统是否包含有 POSIX.2 的 C 编译程序命令 C89。
_SC_2_FORT_DEV	查询系统是否包含有 POSIX.2 的 Fortran 编译程序命令 fort77。
_SC_2_FORT_RUN	查询系统是否包含有 POSIX.2 的 localedef 命令。

_SC_2_SW_DEV	查询系统是否包含有 POSIX.2 的 ar、make 和 strip 命令。
_SC_BC_BASE_MAX	查询 bc 工具中 obase 的最大值。
_SC_BC_DIM_MAX	查询 bc 工具中数组的最大长度。
_SC_BC_SCALE_MAX	查询 bc 工具中 scale 的最大值。
_SC_BC_STRING_MAX	查询 bc 工具中字符串常量的最大长度。
_SC_COLL_WEIGHTS_MAX	查询定义场所的比较序列时使用的最大数。
_SC_EXPR_NEST_MAX	查询使用 expr 工具时表达式可以嵌套的最大层数。
_SC_LINE_MAX	查询 POSIX.2 中 text 工具可以处理的最大文本行。
_SC_EQUIV_CLASS_MAX	查询在场所定义时可以为 LC_COLLATE 类别中的 order 关键词分配的最大数，GNU C 函数库目前不支持场所定义。
_SC_VERSION	查询函数库和内核支持的 POSIX.1 版本号。
_SC_2_VERSION	查询系统工具支持的 POSIX.2 版本好。
_SC_PAGESIZE	查询机器中虚拟内存页的大小。可以使用 getpagesize 返回同样的值。

该函数定义的 errno 出错情况如下。

EINVAL　　parameter 值无效。

第 2 章

Turbo C 函数

本章按照函数原型所在的包含文件，分类介绍 Turbo C 2.0 函数的功能、原型、使用说明等内容。

2.1 ALLOC.H

brk

〖功能〗更改数据段空间的分配。

〖原型〗int brk(void *endds)

〖位置〗ALLOC.H

〖说明〗将程序数据段的顶部设置为 endds 所指向的内存位置。调用成功之后，brk 返回 0。如果调用失败则返回-1，同时设置 errno。

〖参见〗coreleft。

calloc

〖功能〗分配内存。

〖原型〗void *calloc(size_t nelem, size_t elsize)

〖位置〗STDLIB.H,ALLOC.H

〖说明〗为 nelem 数据项的每 elsize 个字节分别分配空间，同时在空间中保存 0 值。返回一个指向新分配块的指针，如果没有足够的空间则返回 NULL。

〖参见〗malloc。

coreleft

〖功能〗返回测出的未使用内存量。

〖原型〗小型模块中 unsigend coreleft(void)

大型模块中：unsiged long coreleft(void)

〖位置〗ALLOC.H

〖参见〗mallloc。

farcalloc

〖功能〗从高端开始分配内存。

〖原型〗void far *farcalloc(unsigned long nunits, unsigned long unitsz)

〖位置〗ALLOC.H

〖说明〗为 nunits 数据项的每 unitsz 个字节分别分配空间。返回指向新分配块的指针，如果没有足够的空间创建数据块，则返回 NULL。

〖参见〗farmalloc,farfree,farcoreleft,malloc,calloc。

farcoreleft

〖功能〗返回测出的高端未使用内存量。

〖原型〗unsiged long farcoreleft(void)

〖位置〗ALLOC.H

〖说明〗返回最高分配数据块和内存末端之间所剩的所有空间量(按照字节计算)。

〖参见〗farcalloc,farmalloc,coreleft。

farfree

〖功能〗从高端释放一个数据块。

〖原型〗void farfree(void far *block)

〖位置〗ALLOC.H

〖参见〗farmalloc,farcalloc。

farmalloc

〖功能〗从高端开始分配。

〖原型〗void far *farmalloc(unsigned long nbytes)

〖位置〗ALLOC.H

〖说明〗返回一个指向新分配数据块的指针，如果没有足够的空间创建数据块则返回 NULL。

〖参见〗farcoreleft,farfree,farcalloc,malloc,farrealloc。

farrealloc

〖功能〗调整高端的分配数据块。

〖原型〗void far *farrealloc(void far *oldblock, unsigned long nbytes)

〖位置〗ALLOC.H

〖说明〗返回重新分配数据块的地址，如果分配失败则返回 NULL。返回值可能与源地址不同。

〖参见〗farmalloc,realloc。

free

〖功能〗释放通过 malloc 或者 calloc 分配的数据块。

〖原型〗void free(void *block)

〖位置〗STDLIB.H, ALLOC.H

〖参见〗malloc,calloc,freemem。

malloc

〖功能〗分配内存。

〖原型〗void *malloc(size_t size)

〖位置〗ALLOC.H, STDLIB.H

〖说明〗大小按照字节计算。返回一个指向新分配数据块的指针，如果没有足够的空间创建数据块，则返回 NULL。如果 size == 0，则也返回 NULL。

〖参见〗allocmem,free,calloc,realloc,farmalloc。

realloc

〖功能〗重新分配主存。

〖原型〗void *realloc(void *block, size_t size)

〖位置〗ALLOC.H, STDLIB.H

〖说明〗尝试将原先分配的数据块扩大或者缩小至 size 字节值。返回重新分配之后与原有地址不同的数据块的地址。如果数据块不能重新分配，或者 size == 0，则返回 NULL。

〖参见〗malloc,free。

sbrk

〖功能〗更改数据段的空间分配。

〖原型〗void *sbrk(int incr)

〖位置〗ALLOC.H

〖说明〗为数据块添加 incr 个字节。成功实现之后，sbrk 返回原有中断值。如果失败则返回值-1，同时设置 errno。

〖参见〗brk。

2.2 ASSERT.H

assert

〖功能〗对条件进行测试，可能会退出程序运行。

〖原型〗void assert(int test)

〖位置〗ASSERT.H

〖参见〗abort。

2.3 BIOS.H

bioscom

〖功能〗RS-232 的 I/O 通讯。

〖原型〗int bioscom(int cmd, char abyte, int port)

〖位置〗BIOS.H

〖说明〗cmd 的值及含义如下。

0	设置通讯参数 abyte。
1	将 abyte 发送出去。
2	接收一个字符(位于返回值的下 8 位中)。

3	返回状态。

port 为 0 表示 COM1，为 1 表示 COM2，依此类推。

返回值的高 8 位表示状态位，低 8 位内容根据 cmd 而定。

biosdisk

〖功能〗BIOS 磁盘服务。

〖原型〗int biosdisk(int cmd, int drive, int head, int track, int sector, int nsects, void *buffer)

〖位置〗BIOS.H

〖说明〗成功则返回 0，否则返回值为错误代码。

biosequip

〖功能〗检查设备。

〖原型〗int boisequip(void)

〖位置〗BIOS.H

〖说明〗返回 BIOS 设备的标记。

bioskey

〖功能〗键盘接口。

〖原型〗bioskey(int cmd)

〖位置〗BIOS.H

〖说明〗cmd 的取值及动作如下。

cmd	动作
0	返回缓存中键入的扫描码，并从缓存中删除它。如果缓存为空，则等待下一次键入。
1	返回缓存中键入的扫描码，但是不从缓存中删除它。如果缓存为空，则返回 0。
2	返回 BIOS 切换状态的标志。

biosmemory

〖功能〗返回内存大小。

〖原型〗int biosmemory(void)

〖位置〗BIOS.H

〖说明〗返回内存大小，以 1 KB 为单位。

biosprint

〖功能〗直接使用 BIOS 实现打印机 I/O。

〖原型〗int biosprint(int cmd, int abyte, int port)

〖位置〗BIOS.H

〖说明〗如果 cmd 为 0，说明字节已经打印。如果 cmd 为 1，说明打印机端口已经初始化。如果 cmd 为 2，说明当前打印机状态可读。

对应所有 cmd 数值都返回当前打印机状态。

biostime

〖功能〗BIOS 定时器服务程序。

〖原型〗long biostime(int cmd, long newtime)

〖位置〗BIOS.H

〖说明〗如果 cmd 为 0，则读取 BIOS 定时器，如果 cmd 为 1 则设置 BIOS 定时器。时间从午夜开始计算，以时钟滴答为单位，每秒 18.2 个滴答。

2.4 CONIO.H

cgets

〖功能〗从控制台读取字符串。

〖原型〗char *cgets(char *str)

〖位置〗CONIO.H

〖说明〗str[0]必须包含读入字符串的最大长度，Str[1]则相应地设置为实际读入字符的个数。字符串从 str[2]开始，函数返回&str[2]。

〖参见〗cputs,gets,fgets。

clreol

〖功能〗清除文本窗口中的行尾。

〖原型〗void clreol(void)

〖位置〗CONIO.H

〖参见〗clrscr,delline,window。

clrscr

〖功能〗清除文本模式的窗口。

〖原型〗void clrscr(void)

〖位置〗CONIO.H

〖参见〗clreol,delline,window。

cprintf

〖功能〗在屏幕上的文本窗口中格式化输出。

〖原型〗int cprintf(const char *format, ...)

〖位置〗CONIO.H

〖说明〗返回输出的字节个数。
〖参见〗printf。

cputs

〖功能〗在屏幕上的文本窗口中书写字符串。
〖原型〗int cputs(const char *str)
〖位置〗CONIO.H
〖说明〗返回打印的最后一个字符。
〖参见〗cgets,puts,fputs。

cscanf

〖功能〗从控制台执行格式化输入。
〖原型〗int cscanf(char *format [, argument, ...])
〖位置〗CONIO.H
〖说明〗返回成功处理的输入字段数目。如果函数在文件结尾处读入，则返回值为EOF。
〖参见〗scanf。

delline

〖功能〗删除文本窗口中的行。
〖原型〗void delline(void)
〖位置〗CONIO.H
〖参见〗clreol,clrscr,window。

Getch、getche

〖功能〗getch 从控制台得到字符，但是不回显。getche 也从控制台得到字符，但同时回显在屏幕上。
〖原型〗int getch(void)
　　int getche(void)
〖位置〗CONIO.H
〖说明〗两个函数都返回读取的字符。字符马上就可以使用，无需等到缓存整行之后。类似功能键和方向键这些特殊键都使用两个字符组成的序列表示：一个 0 字符随后跟一个按键的扫描码。
〖参见〗getpass,cgets,cscanf,kbhit,ungetch,putch,getchar,getc。

getpass

〖功能〗读入口令。
〖原型〗char *getpass(const char *prompt)
〖位置〗CONIO.H

〖说明〗返回一个指针，指向这次调用覆盖的一个静态字符串。
〖参见〗getch。

gettext

〖功能〗从文本模式的屏幕上将文本拷贝至内存中。
〖原型〗int gettext(int left, int top, int right, int bottom, void *destin)
〖位置〗CONIO.H
〖说明〗坐标值与屏幕相关，左上角为(1,1)。如果成功则返回一个非 0 值。
〖参见〗puttext,movetext。

gettextinfo

〖功能〗得到文本模式的视频信息。
〖原型〗void gettextinfo(struct text_info *r)
〖位置〗CONIO.H
〖说明〗结果以 inforec 的形式返回。
〖参见〗textattr,textbackground,textcolor,textmode,wherex,wherey,window。

gotoxy

〖功能〗在文本窗口中定位光标。
〖原型〗void gotoxy(int x, int y)
〖位置〗CONIO.H
〖参见〗wherex,wherey,window。

highvideo

〖功能〗选择高密度的文本字符。
〖原型〗void highvideo(void)
〖位置〗CONIO.H
〖说明〗影响随后的文本窗口函数调用，例如 putch 和 cprintf。
〖参见〗lowvideo,normvideo,textcolor,gettextinfo,cprintf,cputs,putch。

insline

〖功能〗在文本窗口的当前光标位置插入空白行。
〖原型〗void insline(void)
〖位置〗CONIO.H
〖说明〗当前光标位置的下一行依次向下推，最后一行消失。
〖参见〗clreol,delline,window。

kbhit

〖功能〗检查最近的键盘输入。

〖原型〗int kbhit(void)
〖位置〗CONIO.H
〖说明〗如果存在键盘输入，则 kbhit 返回一个非 0 整数。
〖参见〗getc。

lowvideo

〖功能〗为文本窗口输出选择低密度的字符。
〖原型〗void lowvideo(void)
〖位置〗CONIO.H
〖说明〗影响随后的文本窗口函数调用，例如 putch 和 cprintf。
〖参见〗highvideo,normvideo。

movetext

〖功能〗将屏幕上一个矩形范围内的文本拷贝至另一个矩形中(文本模式)。
〖原型〗int movetext(int left, int top, int right, int bottom, int destleft, int desttop)
〖位置〗CONIO.H
〖说明〗坐标与屏幕左上角(1,1)相对应，如果操作成功则返回非 0 值。
〖参见〗gettext,puttext。

normvideo

〖功能〗选择正常密度的字符。
〖原型〗void normvideo(void)
〖位置〗CONIO.H
〖说明〗影响随后的文本窗口函数调用，例如 putch 和 cprintf。
〖参见〗highvideo,lowvideo。

putch

〖功能〗在屏幕上的文本窗口中输出字符。
〖原型〗int putch(int ch)
〖位置〗CONIO.H
〖说明〗使用当前的颜色和显示属性，返回显示字符 ch。
〖参见〗cprintf,cputs,getch,putc,putchar。

puttext

〖功能〗将内存中的文本拷贝至屏幕上。
〖原型〗int puttext(int left, int top, int right, int bottom, void *source)
〖位置〗CONIO.H
〖说明〗坐标与屏幕左上角(1,1)相对应，如果成功则返回一个非 0 值。
〖参见〗gettext,movetext,window。

textattr

〖功能〗设置文本窗口函数的文本属性。
〖原型〗void textattr(int newattr)
〖位置〗CONIO.H
〖参见〗textclolor,textbackground。

textbackground

〖功能〗选择新的文本背景颜色。
〖原型〗void textbackground(int newcolor)
〖位置〗CONIO.H
〖参见〗textcolor,textattr。

textcolor

〖功能〗选择文本模式下的新字符颜色。
〖原型〗void textcolor(int newcolor)
〖位置〗CONIO.H
〖参见〗textbackground,textattr,highvideo,lowvideo,normvideo。

textmode

〖功能〗更改屏幕模式(文本模式下)。
〖原型〗void textmode(int newmode)
〖位置〗CONIO.H
〖说明〗不能用来将图片模式更改为文本模式。
〖参见〗initgraph,gettextinfo。

ungetch

〖功能〗将一个字符退回至键盘。
〖原型〗int ungetch(int ch)
〖位置〗CONIO.H
〖说明〗下一次调用 getch 或者其他控制台输入函数时，将返回 ch。如果成功则返回字符 ch，返回 EOF 表示出错。
〖参见〗getch。

wherex

〖功能〗得到当前文本窗口中光标的水平位置。
〖原型〗int wherex(void)
〖位置〗CONIO.H
〖说明〗返回从 1 到 80 范围内的一个整数。

〖参见〗wherey,gettextinfo,gotoxy。

wherey

〖功能〗得到当前文本窗口中光标的垂直位置。
〖原型〗int wherey(void)
〖位置〗CONIO.H
〖说明〗返回一个从 1 到 25 范围内的一个整数。
〖参见〗wherex,gettextinfo,gotoxy。

window

〖功能〗定义激活的文本模式窗口。
〖原型〗void window(int left, int top, int right, int bottom)
〖位置〗CONIO.H
〖说明〗屏幕左上角的坐标为(1,1)
〖参见〗gettextinfo,textmode。

2.5　CTYPE.H

isascii

〖功能〗如果 c 为有效的 ASCII 字符，则返回真。
〖原型〗isascii(c)
〖位置〗CTYPE.H

isalnum

〖功能〗如果 c 为字母或者数字，则返回真。
〖原型〗isalnum(c)
〖位置〗CTYPE.H

isalpha

〖功能〗如果 c 为字母，则返回真。
〖原型〗isalpha(c)
〖位置〗CTYPE.H

iscntrl

〖功能〗如果 c 为删除字符或者普通的控制字符，则返回真。
〖原型〗iscntrl(c)
〖位置〗CTYPE.H

isdigit

〖功能〗如果 c 为数字，则返回真。
〖原型〗isdigit(c)
〖位置〗CTYPE.H

isgraph

〖功能〗除了不包括空格字符之外，其功能与 isprint 一致。
〖原型〗isgraph(c)
〖位置〗CTYPE.H

islower

〖功能〗如果 c 为小写字母，则返回真。
〖原型〗islower(c)
〖位置〗CTYPE.H

isprint

〖功能〗如果 c 为可打印字符，则返回真。
〖原型〗isprint(c)
〖位置〗CTYPE.H

ispuct

〖功能〗如果 c 为标点符号字符，则返回真。
〖原型〗ispunct(c)
〖位置〗CTYPE.H

isspace

〖功能〗如果 c 为空格、制表符、回车符、换行符、垂直制表符或者换页符时，返回真。
〖原型〗isspace(c)
〖位置〗CTYPE.H

isupper

〖功能〗如果 c 为大写字母，则返回真。
〖原型〗isupper(c)
〖位置〗CTYPE.H

isxdigit

〖功能〗如果 c 为十六进制数字，则返回真。

〖原型〗isxdigit(c)
〖位置〗CTYPE.H

toascii

〖功能〗将大于 127 的 c 转换到 0~127 范围，只保留 c 的低 7 位。
〖原型〗toascii(c)
〖位置〗CTYPE.H

_tolower

〖功能〗将[A~Z]范围内的字符 c 转换为[a~z]范围内的字符。
〖原型〗_tolower(c)
〖位置〗CTYPE.H

_toupper

〖功能〗将[a~z]范围内的字符 c 转换为[A~Z]范围内的字符。
〖原型〗_toupper(c)
〖位置〗CTYPE.H

2.6　DIR.H

chdir

〖功能〗更改当前目录。
〖原型〗int chdir(const char *path)
〖位置〗DIR.H
〖说明〗成功实现之后，chdir 返回一个 0 值。否则返回值-1，同时设置 errno。
〖参见〗mkdir,rmdir。

findfirst

〖功能〗搜索磁盘目录。
〖原型〗int findfirst(const char *filename, struct ffblk *ffblk, int attrib)
〖位置〗DIR.H
〖说明〗文件路径中可以包括通配符?(匹配单个字符)和*(匹配多个字符)。如果成功返回 0，没有找到匹配结果或者出错则返回-1，同时设置 errno。

findnext

〖功能〗继续执行 findfirst 搜索。
〖原型〗int findnext(struct ffblk *ffblk)
〖位置〗DIR.H

〖说明〗文件路径中可以包括通配符?(匹配单个字符)和*(匹配多个字符)。如果成功返回 0，没有找到匹配结果或者出错则返回-1，同时设置 errno。

fnmerge

〖功能〗使用各个组成部分组建一个路径。

〖原型〗void fnmerge(char *path, const char *drive, const char *dir, const char *name, const char *ext)

〖位置〗DIR.H

〖参见〗fnsplit。

fnsplit

〖功能〗将一个路径分解成多个组成部分。

〖原型〗int fnsplit(const char *path, char *drive, char *dir, char *name, char *ext)

〖位置〗DIR.H

〖说明〗返回一个整数，由 5 个标志位组成。

〖参见〗fnmerege。

getcurdir

〖功能〗得到指定驱动器的当前目录。

〖原型〗int getcurdir(int drive, char *directory)

〖位置〗DIR.H

〖说明〗drive 为 0 表示默认驱动器，1 表示 A，2 表示 B，依此类推。

成功则返回 0，错误则返回-1。

〖参见〗chdir,getcwd,getdisk,mkdir,rmdir。

getcwd

〖功能〗得到当前工作目录。

〖原型〗char *getcwd(char *buf, int buflen)

〖位置〗DIR.H，DOS.H

〖说明〗返回一个指向 buf 的指针，出现错误时返回 NULL，同时设置 errno。

〖参见〗getcurdir,getdisk,mkdir,chdir,rmdir。

getdisk

〖功能〗得到当前驱动器。

〖原型〗int getdisk(void)

〖位置〗DIR.H

〖说明〗返回当前驱动器号，A 驱动器使用 0 表示。

〖参见〗getcurdir,setdisk,getcwd。

mkdir

〖功能〗创建一个目录。

〖原型〗int mkdir(const char *path)

〖位置〗DIR.H

〖说明〗成功则返回 0，错误则返回-1，同时设置 errno。

〖参见〗chdir,rmdir。

mktemp

〖功能〗创建一个惟一的文件名称。

〖原型〗char *mktemp(char *template)

〖位置〗DIR.H

〖说明〗使用惟一的文件名称替换 template，并且返回 template 的地址。template 必须由一个非空字符串和 6 个 X 后缀组成，例如 MYFILEXXXXXX。

rmdir

〖功能〗删除目录。

〖原型〗int rmdir(const char *path)

〖位置〗DIR.H

〖说明〗成功返回 0，错误返回-1，同时设置 errno。

〖参见〗mkdir,chdir。

searchpath

〖功能〗在 DOS 路径中检索一个文件。

〖原型〗char *searchpath(const char *file)

〖位置〗DIR.H

〖说明〗成功则返回一个指针，指向表示文件完全路径名称的字符串，否则返回 NULL。返回的字符串保存在一个静态区域内，每次重新调用都将覆盖这一区域。

〖参见〗exec..,findfirst,open,system。

setdisk

〖功能〗设置当前磁盘驱动器。

〖原型〗int setdisk(int drive)

〖位置〗DIR.H

〖说明〗drive 数值为 0=A，1=B，2=C，依此类推。函数返回所有可用驱动器的总数。

2.7　DOS.H

absread

〖功能〗读入绝对磁盘扇区。

〖原型〗int absread(int drive, int nsects, int lsect, void *buffer)

〖位置〗DOS.H

〖说明〗drive 值为 0=A，1=B，2=C，依此类推。nsect 表示读/写的扇区号码。lsect 表示开始逻辑扇区(第 1 个为 0)。buffer 表示数据区的地址。成功返回 0，错误返回-1 并设置 errno。

abswrite

〖功能〗写入绝对磁盘扇区。

〖原型〗int abswrite(int drive, int nsects, int lsect, void *buffer)

〖位置〗DOS.H

〖说明〗drive 值为 0=A，1=B，2=C，依此类推。nsect 表示读/写的扇区号码。lsect 表示开始逻辑扇区(第 1 个为 0)。buffer 表示数据区的地址。成功返回 0，错误返回-1 并设置 errno。

allocmem

〖功能〗分配 DOS 内存片断。

〖原型〗int allocmem(unsigned size, unsigned *segp)

〖位置〗DOS.H

〖说明〗size 表示请求的 16 位段落的号码，分配区域的段地址保存在*segp 中(offset=0)。成功返回-1，否则返回最大可用数据块的大小，并设置_doserrno 和 errno。

〖参见〗freemem。

bdos

〖功能〗MS DOS 的系统调用。

〖原型〗int bdos(int dosfun, unsigned dosdx, unsigend dosal)

〖位置〗DOS.H

〖说明〗bdos 的返回值为系统调用设置的 AX 值。

〖参见〗bdosptr,int86,int86x。

bdosptr

〖功能〗MS DOS 的系统调用。

〖原型〗int bdosptr(int dosfun, void *argument, unsigned dosal)

〖位置〗DOS.H

〖说明〗bdosptr 的成功返回值为 AX 值，失败则返回-1，同时设置 errno 和_doserrno。

〖参见〗bdos,int86,int86x。

country

〖功能〗返回与 country 相关的信息。

〖原型〗struct country *country(int xcode, struct country *cp)

〖位置〗DOS.H
〖说明〗返回指针参数 cp，MS DOS 3.0 或以上版本支持。

ctrlbrk

〖功能〗设置控制终端的句柄。
〖原型〗void ctrlbrk(int (*handler)(void))
〖位置〗DOS.H
〖说明〗句柄函数返回 0 退出当前程序，否则程序将继续执行。

delay

〖功能〗中断执行一段时间(以毫秒计算)。
〖原型〗void delay(unsigned milliseconds)
〖位置〗DOS.H
〖参见〗sleep。

disable

〖功能〗禁止中断。
〖原型〗void disable(void)
〖位置〗DOS.H
〖说明〗禁止除了 NMI 之外的所有硬件中断。
〖参见〗enable。

dosexterr

〖功能〗得到 DOS 扩展错误的信息。
〖原型〗int dosexterr(struct DOSERROR *eblkp)
〖位置〗DOS.H
〖说明〗根据最后一次 DOS 调用填写 eblkp 结构。返回 exterror 结构的值。

dostounix

〖功能〗将时间和日期转换成为 UNIX 格式。
〖原型〗long dostounix(struct date *d, struct time *t)
〖位置〗DOS.H
〖说明〗返回 UNIX 个时的日期和时间参数(秒数从 1970 年 1 月 1 日(GMT)开始计算)。
〖参见〗unixtodos。

emit

〖功能〗直接在代码中插入文字。
〖原型〗void _emit_(argument, ...)
〖位置〗DOS.H

enable

〖功能〗允许硬件中断。
〖原型〗void enable(void)
〖位置〗DOS.H
〖参见〗disable。

FP_OFF

〖功能〗得到远端偏移地址。
〖原型〗unsigned FP_OFF(farpointer)
〖位置〗DOS.H
〖参见〗movedata,segread。

FP_SEG

〖功能〗得到远端分段地址。
〖原型〗unsigned FP_SEG(farpointer)
〖位置〗DOS.H
〖参见〗movedata,segread。

freemem

〖功能〗释放原先使用 allocmem 分配的 DOS 内存块。
〖原型〗int freemem(unsigned segx)
〖位置〗DOS.H
〖说明〗成功返回 0，错误返回-1，同时设置 errno。
〖参见〗allocmem,free。

geninterrupt

〖功能〗生成软件中断。
〖原型〗调用之后所有寄存器的状态都根据调用的中断而定。注意，中断可能导致 C 使用的寄存器处于一种不可预见的状态之中。
〖位置〗DOS.H
〖参见〗int86,int86x,intdos,intdosx,bdos,bdosptr,intr,enable,disable。

getcbrk

〖功能〗得到控制中断的设置。
〖原型〗int getcbrk(void)
〖位置〗DOS.H
〖说明〗如果 control_break 检查被关闭则返回 0，如果检查被打开则返回 1。
〖参见〗setcbrk,ctrlbrk。

getcwd

〖功能〗得到当前工作目录。
〖原型〗char *getcwd(char *buf, int buflen)
〖位置〗DIR.H，DOS.H
〖说明〗返回一个指向 buf 的指针，出现错误时返回 NULL，同时设置 errno。
〖参见〗getcurdir,getdisk,mkdir,chdir,rmdir。

getdate

〖功能〗得到 MS DOS 日期。
〖原型〗void getdate(struct date *datep)
〖位置〗DOS.H
〖参见〗setdate,gettime,ctime。

getdfree

〖功能〗得到磁盘的剩余空间。
〖原型〗void getdfree(unsigned char drive, struct dfree *dtable)
〖位置〗DOS.H
〖说明〗如果出错，则 dtable 结构中的 df_sclus 将设置为-1。
〖参见〗getfat。

getdta

〖功能〗得到磁盘转移地址。
〖原型〗char far *getdta(void)
〖位置〗DOS.H
〖说明〗返回一个指针，指向当前的磁盘转移地址。
〖参见〗stuct,fcb,setdta。

getfat

〖功能〗得到指定驱动器的文件分配表信息。
〖原型〗void getfat(unsigned char drive, struct fatinfo *dtable)
〖位置〗DOS.H
〖参见〗getdfree,getfatd。

getfatd

〖功能〗得到文件分配表信息。
〖原型〗void getfatd(struct fatinfo *dtable)
〖位置〗DOS.H
〖参见〗getdfree,getfat。

getftime

〖功能〗得到文件的日期和时间。
〖原型〗int getftime(int handle, struct ftime *fftimep)
〖位置〗IO.H
〖说明〗成功返回 0，错误则返回-1，同时设置 errno。
〖参见〗setftime,open。

getpsp

〖功能〗得到程序分段的前缀。
〖原型〗unsigned getpsp(void)
〖位置〗DOS.H
〖说明〗适用于 MS DOS 3.0 或者更新的版本。
〖参见〗_psp,getenv。

gettime

〖功能〗得到系统时间。
〖原型〗void gettime(struct time *timep)
〖位置〗DOS.H
〖参见〗settime,getdate。

getvect

〖功能〗得到中断扇区。
〖原型〗void interrupt(*getvect(int intr_num)) ()
〖位置〗DOS.H
〖说明〗返回在 intr_num 中断扇区中保存的一个 4 字节数值。
〖参见〗setvect,disable,enable。

getverify

〖功能〗得到校验状态。
〖原型〗int getverify(void)
〖位置〗DOS.H
〖说明〗如果校验标志关闭则返回 0，如果校验标志打开则返回 1。
〖参见〗setverify。

harderr

〖功能〗建立一个硬件错误句柄。
〖原型〗void harderr(int (*(handler)())
〖位置〗DOS.H

〖说明〗当 MS DOS 出现致命错误(int 0x24)时，调用句柄指向的函数。

〖参见〗hardresume,hardretn,longjmp。

hardresume

〖功能〗硬件错误句柄函数。

〖原型〗void hardresume(int axret)

〖位置〗DOS.H

〖说明〗harderr 建立的错误句柄可以将执行控制权返回给通过这个函数发布致命错误的 MS DOS 程序。axret 中的数值返回给 MS DOS(0 表示忽略，1 表示重试，2 表示退出)。

〖参见〗hardretn。

hardretn

〖功能〗硬件错误句柄函数。

〖原型〗void hardretn(int retn)

〖位置〗DOS.H

〖说明〗调用该函数之后，harderr 建立的错误句柄可以直接返回应用程序。retn 中的数值返回到用户程序中，覆盖生成致命错误的 MS DOS 函数返回的正常值。

〖参见〗hardresume。

inp

〖功能〗从硬件端口读取一个字节。

〖宏原型〗int inp(int portid)

〖位置〗DOS.H

〖参见〗inport,inporth。

inport

〖功能〗从硬件端口读取一个词。

〖原型〗int inport(int portid)

〖位置〗DOS.H

〖参见〗inportb,outport。

inportb

〖功能〗从硬件端口读取一个字节。

〖原型〗unsigned char inportb(int portid)

〖位置〗DOS.H

〖参见〗inport,outportb。

int86

〖功能〗常用的 8086 软件中断。

〖原型〗int int86(int intno, union REGS *inregs, union REGS *outregs)

〖位置〗DOS.H

〖说明〗该函数将保存在 inregs 中的数值装载到 CPU 寄存器中，发布中断 intno，然后在 outregs 中保存结果 CPU 寄存器。

〖参见〗int86x,intdos,intr。

int86x

〖功能〗常用的 8086 软件中断接口。

〖原型〗int int86x(int intno, union REGS *inregs, union REGS *outregs, struct SREGS*segregs)

〖位置〗DOS.H

〖说明〗该函数将 inregs 和 segregs 中的数值装载到 CPU 寄存器中，发布中断 intno，然后在 outregs 和 segregs 中保存最终的 CPU 寄存器数值。

〖参见〗int86,intdosx,intr。

intdos

〖功能〗常用的 MS DOS 中断接口。

〖原型〗int intdos(union REGS *inregs, union REGS *outregs)

〖位置〗DOS.H

〖说明〗该函数将 inregs 中保存的数值装载到 CPU 寄存器钟，发布 MS DOS 中断(int 33 或者 0x21)，然后在 outregs 中保存最终的 CPU 寄存器数值。

〖参见〗intdosx,int86,intr,bdos,bdosptr。

intdosx

〖功能〗常用的 MS DOS 中断接口。

〖原型〗int intdosx(union REGS *inregs, union REGS *outregs, struct SREGS *segregs)

〖位置〗DOS.H

〖说明〗intdosx 将 inregs 和 segregs 中的数值装载到 CPU 寄存器中，发布 DOS 中断(int 0x21)，然后在 outregs/segregs 中保存最终的 CPU 寄存器数值。

〖参见〗intdos,int86x,intr,bdos,bdosptr。

intr

〖功能〗转换 8086 软件中断的接口。

〖原型〗void intr(int intno, struct REGPACK *preg)

〖位置〗DOS.H

〖说明〗该函数将 preg 中保存的数值装载到 CPU 寄存器中，发布中断 intno，然后在 preg 中保存结果 CPU 寄存器的数值。

〖参见〗int96,int86x,intdos。

keep

〖功能〗程序退出运行，但是继续驻留内存。

〖原型〗void keep(unsigned char status, unsigned size)

〖位置〗DOS.H

〖说明〗该函数没有返回。它只是退出到 DOS 状态，同时在 status 中保存返回值，但是程序仍然驻留在内存中。程序在中断运行之前被设置成为 size 大小的段落，剩余的程序内存则返回给 DOS。

〖参见〗abort,exit,exec,spawn,system。

MK_FP

〖功能〗创建一个远端指针。

〖原型〗void far *MK_FP(se, off)

〖位置〗DOS.H

〖参见〗movedata,segread。

nosound

〖功能〗关闭 PC 喇叭。

〖原型〗void nosound(void)

〖位置〗DOS.H

〖参见〗sound,delay。

outp

〖功能〗在硬件端口写一个字节。

〖宏原型〗int outp(int portid, int byte_value)

〖位置〗DOS.H

〖参见〗outport,outportb。

outport

〖功能〗在硬件端口输出一个词。

〖原型〗void outport(int portid, int value)

〖位置〗DOS.H

〖参见〗inport,outportb。

outportb

〖功能〗在硬件端口输出一个字节。

〖原型〗void outportb(int portid, unsigned char value)

〖位置〗DOS.H

〖参见〗inportb,outport。

parsfnm

〖功能〗解析文件名称，然后创建文件控制块(FCB)。

〖原型〗char *parsfnm(const char *cmdline, struct fcb *fcb, int option)

〖位置〗DOS.H

〖说明〗成功解析一个文件名称之后，parsfnm 返回一个指针，指向文件名称末尾的下一个字节。如果在解析文件名称时出现任何错误，则返回 0。

peek

〖功能〗返回由 segment:offset 指定内存位置中的词。

〖原型〗int peek(unsigned segment, unsigned offset)

〖位置〗DOS.H

〖参见〗peekb,poke。

peekb

〖功能〗返回由 segment:offset 指定内存位置中的字节。

〖原型〗char peekb(unsigned segment, unsigned offset)

〖位置〗DOS.H

〖参见〗peek,pokeb。

poke

〖功能〗在由 segment:offset 指定内存位置中保存一个整数值。

〖原型〗void poke(unsigned segment, unsigned offset, int value)

〖位置〗DOS.H

〖参见〗peek,pokeb。

pokeb

〖功能〗存值到一个指定存储单元。

〖原型〗void pokeb(unsigned segment, unsigned offset, int value)

〖位置〗DOS.H

〖参见〗peek,poke。

randbrd

〖功能〗读取随机数据块。

〖原型〗int randbrd(struct fcb *fcb, int rcnt)

〖位置〗DOS.H

〖说明〗根据 randbrd 操作的结果，将返回如下数值。

0	读入所有记录。

1　　到达文件结尾，已经读入最后一条记录。
2　　循环读入记录。
3　　到达文件结尾，最后一条记录尚未读入。

〖参见〗randwr。

randbwr

〖功能〗使用文件控制块(FCB)随机写入数据块。
〖原型〗int randbwr(struct fcb *fcb, int rcnt)
〖位置〗DOS.H
〖说明〗根据 randbwr 操作的结果，将返回如下数值。

0　　写入所有记录。
1　　没有足够的空间写入。
2　　循环写入记录。

〖参见〗randbrd。

segread

〖功能〗读入段寄存器。
〖原型〗void segread(struct SREGS *segp)
〖位置〗DOS.H
〖参见〗FP_OFF,intdosx,int86x。

setblock

〖功能〗改变原先分配的数据块大小。
〖原型〗int setblock(unsigned segx, unsigned newsize)
〖位置〗DOS.H
〖说明〗对调用 allocmem 函数分配的数据块使用。
成功时返回-1。如果出现错误，则返回可能出现的最大数据块的大小，并且设置_doserrno。
〖参见〗allocmem,freemem。

setcbrk

〖功能〗设置控制中断。
〖原型〗int setcbrk(int cbrkvalue)
〖位置〗DOS.H
〖说明〗如果 cbrkvalue 等于 1，则每次系统调用都需要检查 Ctrl-Break。如果等于 0，则只在控制台、打印机和通信 I/O 调用时进行检查。返回值通过 cbrkvalue 传递。

setdate

〖功能〗设置 MS DOS 日期。
〖原型〗void setdate(struct date *datep)
〖位置〗DOS.H
〖参见〗settime,getdate。

setdta

〖功能〗设置磁盘转换地址。
〖原型〗void setdta(char far *dta)
〖位置〗DOS.H

settime

〖功能〗设置系统时间。
〖原型〗void settime(struct time *timep)
〖位置〗DOS.H
〖参见〗getdate,gettime。

setvect

〖功能〗设置中断扇区入口。
〖原型〗void setvect(int interruptno, void interrupt(*isr) ())
〖位置〗DOS.H
〖说明〗isr 指向出现中断号码为 interruptno 的中断时调用的函数。如果 isr 为一个 C 函数，就应当使用 interrupt 关键字进行定义。
〖参见〗getvect。

setverify

〖功能〗设置校验状态。
〖原型〗void setverify(int value)
〖位置〗DOS.H
〖说明〗如果 value 等于 1，则每次磁盘的写操作之后都将进行一次读操作以便于确保正确的结果(0 则表示不在随后进行读操作)。
〖参见〗getverify。

sleep

〖功能〗程序执行挂起一段时间。
〖原型〗void sleep(unsigned seconds)
〖位置〗DOS.H
〖参见〗delay。

sound

〖功能〗将 PC 喇叭打开，并且设置到某个频率。
〖原型〗void sound(unsigned frequency)
〖位置〗DOS.H
〖说明〗频率以赫兹为单位(每秒的圈数)。
〖参见〗nosound,delay。

unixtodos

〖功能〗将 UNIX 格式的日期和时间转换成 DOS 格式。
〖原型〗void unixtodos(long time, struct date *d, struct time *t)
〖位置〗DOS.H
〖参见〗dostounix。

unlink

〖功能〗删除一个文件。
〖原型〗int unlink(const char *filename)
〖位置〗DOS.H IO.H,STDIO.H
〖说明〗如果 filename 指定的文件属性为只读，则 unlink 操作失败。首先需要调用 chmod 函数改变文件属性。
成功返回 0，错误返回-1
〖参见〗chmod,remove。

2.8　FLOAT.H

_clear87

〖功能〗清除浮点状态。
〖原型〗unsigned int _clear87 (void)
〖位置〗FLOAT.H
〖说明〗返回值的位表示旧的浮点状态。有关状态的详细解释，参见 FLOAT.H 文件中定义的常量。
〖参见〗_fpreset,_status87。

_control87

〖功能〗改变浮点控制词。
〖原型〗unsigned int _control87(unsigned int new, unsigned int mask)
〖位置〗FLOAT.H
〖说明〗如果 mask 中的某一位为 1，则 new 中的相应位保存控制词中相同位的新数

值。如果 mask 为 0，则说明控制词没有改变。

〖参见〗_clear87,_fpreset,status87,CWDEFAULT。

_fpreset

〖功能〗重新初始化浮点数学包。

〖原型〗void _fpreset(void)

〖位置〗FLOAT.H

〖说明〗该函数必须与 exec、spawn、system 等函数联合使用，因为子进程可能改变父进程的浮点状态。

_status87

〖功能〗得到浮点状态。

〖原型〗unsigned int _status87(void)

〖位置〗FLOAT.H

〖说明〗返回值的位说明了浮点状态。

〖参见〗_clear87,_control87,_fpreset。

2.9 GRAPHICS.H

bar

〖功能〗画出一条栏目。

〖原型〗void far bar(int left, int top, int right, int bottom)

〖位置〗GRAPHICS.H

〖参见〗bar3d,setfillstyle,setlinestyle,rectangle。

bar3d

〖功能〗画出一条 3-D 栏目。

〖原型〗void far bar3d(int left, int top, int right, int bottom, int depth, int topflag)

〖位置〗GRAPHICS.H

〖参见〗bar。

circle

〖功能〗以(x,y)为圆心按照指定的半径画出一个圆。

〖原型〗void far circle(int x, int y, int radius)

〖位置〗GRAPHICS.H

〖参见〗arc。

cleardevice

〖功能〗清除图形画面。

〖原型〗void far cleardevice(void)

〖位置〗GRAPHICS.H
〖参见〗clearviewport。

clearviewport

〖功能〗清除当前可视区域。
〖原型〗clearviewport(void)
〖位置〗GRAPHICS.H
〖参见〗setviewport,cleardevice。

closegraph

〖功能〗关闭图形系统。
〖原型〗void far closegraph(void)
〖位置〗GRAPHICS.H
〖参见〗initgraph。

detectgraph

〖功能〗通过检查硬件确定使用的图形驱动程序和模式。
〖原型〗void far detectgraph(int far *graphdriver int far *graphmode)
〖位置〗GRAPHICS.H
〖参见〗initgraph,graphresult。

drawpoly

〖功能〗画出一个多边形的轮廓。
〖原型〗void far drawpoly(int numpoints, int far polypoints[])
〖位置〗GRAPHICS.H
〖说明〗polypoints 中包括一共 numpoints 对数值。其中每一对都给出了多边形中一个顶点的 x 和 y 值。
〖参见〗fillpoly。

ellipse

〖功能〗画出一条椭圆形的圆弧。
〖原型〗void far ellipse(intx, int y, int stangle, int endangle, int xradius, int yradius)
〖位置〗GRAPHICS.H
〖说明〗中心点在(x,y)，stangle 和 endangle 表示以角度为单位的起始角和终止角。xradius 和 yradius 表示水平轴和垂直轴。
〖参见〗arc,circle,fillellipse。

fillellipse

〖功能〗画出椭圆形圆弧，并填充。

〖原型〗void far fillellipse(int x, int y, int xradius, int yradius)

〖位置〗GRAPHICS.H

〖说明〗使用(x,y)作为中心点，然后使用当前的填充模式填充圆弧。xradius 和 yradius 表示水平轴和垂直轴。

〖参见〗arc,circle,ellipse,pieslice。

fillpoly

〖功能〗画出一个多边形，并填充。

〖原型〗void far fillpoly(int numpoints, int far polypoints[])

〖位置〗GRAPHICS.H

〖说明〗polypoints 中包含有 numpoints 对数值。其中每一对给出了多边形一个顶点的 x 值和 y 值。

〖参见〗drawpoly,fill_patterns,floodfill,graphresult,setfillstyle。

floodfill

〖功能〗填充一个有界区域范围。

〖原型〗void far floodfill(int x, int y, int border)

〖位置〗GRAPHICS.H

〖参见〗drawpoly,fillpoly,setfillstyle,fill_patterns,graphresult。

getarccoords

〖功能〗得到最后一次调用 arc 的坐标值。

〖原型〗void far getarccoords(struct arccoordstype far *arccoords)

〖位置〗GRAPHICS.H

〖参见〗arc。

getaspectratio

〖功能〗得到当前图形模式的纵横比。

〖原型〗void far getaspectratio(int far *xasp, int far *yasp)

〖位置〗GRAPHICS.H

〖说明〗arc 及类似函数使用纵横比令圆圈更加圆，而不会像椭圆。yasp 应当为 10 000。当象素为<VGA>平方并且<10 000 时，xasp 使用 10 000 数值。

〖参见〗arc,setaspectratio。

getbkcolor

〖功能〗返回当前背景颜色。

〖原型〗int far getbkcolor(void)

〖位置〗GRAPHICS.H

〖参见〗setbkcolor,getcolor,getpalette。

getcolor

〖功能〗返回当前的画笔颜色。
〖原型〗int far getcolor(void)
〖位置〗GRAPHICS.H
〖参见〗getbkcolor,setcolor,getmaxcolor,getpalette。

getdefaultpalette

〖功能〗返回调色板定义结构。
〖原型〗struct palettetype *far getdefaultpalette(void)
〖位置〗GRAPHICS.H
〖说明〗返回一个指针，指向调用 initgraph 初始化时，当前驱动程序的默认调色板结构。
〖参见〗getpalette,initgraph。

getdrivername

〖功能〗返回一个指针，指向当前图形驱动程序的名称。
〖原型〗char *far getdrivername(void)
〖位置〗GRAPHICS.H
〖说明〗返回的指针指向标识当前驱动程序的字符串，从而可以探测到硬件适配器。
〖参见〗initgraph。

getfillpattern

〖功能〗将用户定义的填充模式复制到内存中。
〖原型〗void far getfillpattern(char far *pattern)
〖位置〗GRAPHICS.H
〖参见〗getfillsettings,setfillpattern,fill_patterns。

getfillsettings

〖功能〗得到当前填充模式及其颜色的有关信息。
〖原型〗vod far getfillsettings(stuct fillsettingstype far *fillinfo)
〖位置〗GRAPHICS.H
〖参见〗floodfill,fillpoly,setfillstyple,pieslice,setfillpattern,bar3d,getfillpattern,bar。

getgraphmode

〖功能〗返回当前图形模式。
〖原型〗int far getgraphmode(void)
〖位置〗GRAPHICS.H
〖说明〗必须先调用 initgraph 或者 setgraphmode。

〖参见〗getmoderange,restorectrtmode。

getimage

〖功能〗将指定区域的位图保存到内存中。
〖原型〗void far getimage(int left, int top, int right, int bottom, void far *bitmap)
〖位置〗GRAPHICS.H
〖参见〗imagesize,putimage。

getlinesettings

〖功能〗得到当前的直线样式、模式和粗细。
〖原型〗void far getlinesettings(struct linesettingstype far *lineinfo)
〖位置〗GRAPHICS.H
〖参见〗setlinestyle。

getmaxcolor

〖功能〗返回最大颜色值。
〖原型〗int far getmaxcolor(void)
〖位置〗GRAPHICS.H
〖参见〗getbkcolor,getpalette,getcolor,setcolor。

getmaxmode

〖功能〗返回当前驱动程序的最大图形模式号。
〖原型〗int far getmaxmode(void)
〖位置〗GRAPHICS.H
〖说明〗getmaxmode 可以得到当前装载的图形驱动程序的最大模式号。适用于所有驱动程序—包括 Borland 驱动程序及其他驱动程序。
〖参见〗getmoderange。

getmaxx

〖功能〗返回画面坐标的最大 x 值。
〖原型〗int far getmaxx(void)
〖位置〗GRAPHICS.H
〖参见〗getx。

getmaxy

〖功能〗返回画面坐标的最大 y 值。
〖原型〗int far getmaxy(void)
〖位置〗GRAPHICS.H
〖参见〗getx。

getmodename

〖功能〗返回指针，指向图形模式的名称。
〖原型〗char * far getmodename(int mode_number)
〖位置〗GRAPHICS.H
〖说明〗返回的指针指向 mode_number 指定模式的名称(字符串)。
〖参见〗getmaxmode,getmoderange。

getmoderange

〖功能〗得到指定图形驱动程序的模式范围。
〖原型〗void far getmoderange(int graphdriver, int far *lomode, int far *himode)
〖位置〗GRAPHICS.H
〖参见〗initgraph,getmaxmode,setgraphmode,getgraphmode。

getpalette

〖功能〗得到当前调色板的有关信息。
〖原型〗void far getpalette(struct palettetype far *palette)
〖位置〗GRAPHICS.H
〖参见〗setpalette,setallpalette,getbkcolor,getdefaultpalette,getcolor。

getpalettesize

〖功能〗返回调色板表的大小。
〖原型〗int far getpalettesize(void)
〖位置〗GRAPHICS.H
〖说明〗getpalettesize 返回当前图形驱动程序模式允许的调色板条目的数目。
〖参见〗setpalette,setallpalette。

getpixel

〖功能〗得到某个指定象素的颜色。
〖原型〗unsigned far getpixel(int x, int y)
〖位置〗GRAPHICS.H
〖参见〗putpixel,getimage。

gettextsettings

〖功能〗得到当前图形文字字体的有关信息。
〖原型〗void far gettextsettings(struct textsettingstype far *texttypeinfo)
〖位置〗GRAPHICS.H
〖参见〗outtext,outtextxy,textheight,textwidth,settextjustify,settextstyle,registerbgifont。

getviewsettings

〖功能〗得到当前视窗的有关信息。
〖原型〗void far getviewsettings(struct viewporttype far *viewport)
〖位置〗GRAPHICS.H
〖参见〗setviewport,clearviewport。

getx

〖功能〗返回当前位置的 x 坐标。
〖原型〗int far getx(void)
〖位置〗GRAPHICS.H
〖说明〗返回数值与视窗相关。
〖参见〗gety,moveto,getviewsettings。

gety

〖功能〗返回当前位置的 y 坐标。
〖原型〗int far gety(void)
〖位置〗GRAPHICS.H
〖说明〗返回数值与视窗相关。
〖参见〗getx,moveto,getviewsettings。

graphdefaults

〖功能〗将所有图形设置重置为默认值。
〖原型〗void far graphdefaults(void)
〖位置〗GRAPHICS.H
〖参见〗initgraph,setgraphmode。

grapherrormsg

〖功能〗返回一个指针，指向错误消息字符串。
〖原型〗char *far grapherrormsg(int errorcode)
〖位置〗GRAPHICS.H
〖说明〗返回指针指向的字符串与 graphresult 返回值相关。
〖参见〗graphresult。

_graphfreemem

〖功能〗用户申请重新分配图形内存。
〖原型〗void far _graphfreemem(void far *ptr, unsigned size)
〖位置〗GRAPHICS.H
〖说明〗图形函数库中的程序调用该函数来释放内存。可以自己使用_graphgetmem 和

_graphfreemem 函数来控制内存分配。

〖参见〗_graphgetmem。

_graphgetmem

〖功能〗用户申请分配图形内存。

〖原型〗void far *far _graphgetmem(unsigned size)

〖位置〗GRAPHICS.H

〖说明〗图形函数库中的程序调用该函数来分配内存。可以自己使用_graphgetmem 和 _graphfreemem 函数来控制内存分配。

〖参见〗_graphfreemem。

graphresult

〖功能〗返回最后一次失败图形操作的错误代码。

〖原型〗int far graphresult(void)

〖位置〗GRAPHICS.H

〖说明〗返回最后一次报错图形操作的错误代码，并且将错误等级重置为 grOK。

〖参见〗grapherrormsg。

imagesize

〖功能〗返回存储位图所需的字节数。

〖原型〗unsigned far imagesize(int left, int top, int right, int bottom)

〖位置〗GRAPHICS.H

〖说明〗如果选定图像所需的大小大于等于 64 K-1 个字节，则返回 0xFFFF。

〖参见〗getimage,putimage。

initgraph

〖功能〗初始化图形系统。

〖原型〗void far initgraph(int far *graphdriver, int far *graphmode, char far *pathtodriver)

〖位置〗GRAPHICS.H

〖参见〗getgraphmode,closegraph,detectgraph,_graphgetmem,getdrivername,restorecrtmode, setgraphbufsize,registerbgidriver,graphresult,installuserdriver。

installuserdriver

〖功能〗按照设备驱动程序。

〖原型〗int far installuserdriver(char far *name, int huge (*detect) (void))

〖位置〗GRAPHICS.H

〖说明〗name 表示设备驱动程序的文件名称(*.bgi)，detect 指向 initgraph 使用 autodetect 选项调用的函数。

〖参见〗initgraph,registerbgidriver。

installuserfont

〖功能〗装载一个字体文件。

〖原型〗int far installuserfont(char far *name)

〖位置〗GRAPHICS.H

〖说明〗name 表示图形函数还没有识别的字体文件(扩展名为.chr)DOS 文件名称。installuserfont 返回一个字体 ID 号。选择新字体时，使用该 ID 号调用 settextstyle。

〖参见〗settextstyle。

line

〖功能〗在两个指定点之间画出一条直线。

〖原型〗line(int x1, int y1, int x2, int y2)

〖位置〗GRAPHICS.H

〖说明〗使用当前颜色、线条样式和线条粗细从(x1,y1)画一条直线到(x2,y2)。

〖参见〗lineto,linerel,setcolor,getlinesettings。

linerel

〖功能〗从当前位置(CP)画一条已知长度的直线。

〖原型〗linerel(int dx, int dy)

〖位置〗GRAPHICS.H

〖说明〗使用当前颜色、线条样式和线条粗细。

〖参见〗line,lineto,setcolor,getlinesettings。

lineto

〖功能〗从当前位置(CP)到(x,y)之间画一条直线。

〖原型〗void far lineto(int x, int y)

〖位置〗GRAPHICS.H

〖参见〗line,linerel,setcolor,getlinesettings,setwritemode。

moverel

〖功能〗从当前位置(CP)移动一段距离。

〖原型〗moverel(int dx, int dy)

〖位置〗GRAPHICS.H

〖参见〗moveto。

moveto

〖功能〗从当前位置(CP)移动到(x,y)点。

〖原型〗void far moveto(int x, int y)

〖位置〗GRAPHICS.H

〖参见〗moverel。

outtext

〖功能〗在视窗中(图形模式)显示一个字符串。
〖原型〗void far outtext(char far *textstring)
〖位置〗GRAPHICS.H
〖参见〗gettextsettings,textheight,textwidth,outtextxy。

outtextxy

〖功能〗向指定位置(图形模式)发送一个字符串。
〖原型〗void far outtextxy(int x, int y, char far *textstring)
〖位置〗GRAPHICS.H
〖参见〗gettextsettings,textheight,textwidth,outtext。

pieslice

〖功能〗画出扇形区并加以填充。
〖原型〗void far pieslice(int x, int y, int stangle, int endangle, int radius)
〖位置〗GRAPHICS.H
〖参见〗setfillstyle,sector,fill_patterns,fillellipse,graphresult。

putimage

〖功能〗在屏幕上输出一副位图。
〖原型〗void far putimage(int left, int top, void far *bitmap, int op)
〖位置〗GRAPHICS.H
〖说明〗bitmap 指向一副位图，该位图通常由 getimage 函数创建。op 的数值说明如何在(left, top)点将图像与区域中的当前内容结合起来。
〖参见〗getimage。

putpixel

〖功能〗在指定点上画一个像素。
〖原型〗void far putpixel(int x, int y, int pixelcolor)
〖位置〗GRAPHICS.H
〖参见〗getpixel,putimage。

rectangle

〖功能〗画出一个矩形(图形模式)。
〖原型〗void far rectangle(int left, int top, int right, int bottom)
〖位置〗GRAPHICS.H
〖说明〗使用当前线条样式、线条粗细和颜色。

〖参见〗bar,setlinestyle,setcolor。

registerbgidriver

〖功能〗注册链入的图形驱动程序。
〖原型〗int registerbgidriver(void (*driver)(void))
〖位置〗GRAPHICS.H
〖说明〗通知图形系统链接时设备驱动程序指向的参数和驱动程序。
〖参见〗initgraph,registerbgifont,installuserdriver。

registerbgifont

〖功能〗注册链入的字体代码。
〖原型〗int registerbgifont(void (*font)(void))
〖位置〗GRAPHICS.H
〖说明〗通知图形系统连接时指向的字体。
〖参见〗initgraph,registerbgidriver。

restorecrtmode

〖功能〗将屏幕模式恢复到前一次 initgraph 的设置。
〖原型〗void far restorecrtmode(void)
〖位置〗GRAPHICS.H
〖参见〗initgraph,setgraphmode。

sector

〖功能〗画出椭圆形的扇形区并加以填充。
〖原型〗void far sector(int x, int y, int stangle, int endangle, int xradius, int yradius)
〖位置〗GRAPHICS.H
〖说明〗x 和 y 定义出中心点，stangle 和 endangle 则定义起始和终止角。xradius 和 yradius 表示水平和垂直半径。
〖参见〗pieslice,setfillstyle。

setactivepage

〖功能〗为图形输出设置活动页面。
〖原型〗void far setactivepage(int page)
〖位置〗GRAPHICS.H
〖说明〗后续的图形输出就指向该显示页面。该页面可能不是实际显示的可视页面。
〖参见〗setvisualpage。

setallpalette

〖功能〗将所有调色板的颜色更改为指定颜色。

〖原型〗void far setallpalette(struct palettetype far *palette)

〖位置〗GRAPHICS.H

〖参见〗setpalette,getpalettesize,getpalette,setcolor,setbkcolor,graphresult。

setaspectratio

〖功能〗设置图形模式的纵横比。

〖原型〗void far setaspectratio(int xasp, int yasp)

〖位置〗GRAPHICS.H

〖说明〗arc 及类似函数使用纵横比令圆圈更加圆，而不会像椭圆。yasp 应当为 10 000。当象素为<VGA>平方并且<10 000 时，xasp 使用 10 000 数值。

〖参见〗getaspectratio。

setbkcolor

〖功能〗使用调色板设置当前的背景颜色。

〖原型〗void far setbkcolor(int color)

〖位置〗GRAPHICS.H

〖参见〗setpalette,setcolor,getbkcolor。

setcolor

〖功能〗设置当前画笔颜色。

〖原型〗void far setcolor(int color)

〖位置〗GRAPHICS.H

〖参见〗getcolor,setbkcolor,setpalette,graphresult。

setfillpattern

〖功能〗选择一个用户自定义的填充模式。

〖原型〗void far setfillpattern(char far *upattern, int color)

〖位置〗GRAPHICS.H

〖说明〗upattern 指向一个 8 字节的区域，其中定义了一个 8*8 位的模式。

〖参见〗fill_patterns,getfillpattern,getfillsettings,setfillstyle。

setfillstyle

〖功能〗设置填充模式和颜色。

〖原型〗void far setfillstype(int pattern,int color)

〖位置〗DOS.H

〖参见〗get,fillstype。

setgraphbufsize

〖功能〗改变内部图形缓存区的大小。

〖原型〗unsigned far setgraphbufsize(unsigned bufsize)

〖位置〗GRAPHICS.H

〖说明〗必须在调用 initgraph 之前先调用该函数。

〖参见〗initgraph,_graphgetmem。

setgraphmode

〖功能〗系统设置为图形模式，同时清屏。

〖原型〗void far setgraphmode(int mode)

〖位置〗GRAPHICS.H

〖参见〗initgraph,getgraphmode,restorecrtmode,graphresult。

setlinestyle

〖功能〗设置当前的线条样式、宽度或者模式。

〖原型〗void far setlinestyle(int linestyle, unsigned upattern, int thickness)

〖位置〗GRAPHICS.H

〖说明〗可以设置使用 line,lineto,rectangle,drawpoly,arc,circle,ellipse,pieslice 等函数画的所有线条样式。

〖参见〗getlinesettings,graphresult。

setpalette

〖功能〗改变一种调色板颜色。

〖原型〗void far setpalette(int colornum, int color)

〖位置〗GRAPHICS.H

〖参见〗getpalette,graphresult,setallpalette,getpalettesize,setcolor,setrgbcolor,setbkcolor,setrgbpalette。

setrgbcolor

〖功能〗为 VGA 和 IBM-8514 驱动程序设置调色板条目。

〖原型〗void far setrgbcolor(int colornum, int red, int green, int blue)

〖位置〗GRAPHICS.H

〖说明〗colornum 表时需要设置的调色板条目，red、green 和 blue 用来定义颜色。

〖参见〗setpalette,setrgbpalette。

setrgbpalette

〖功能〗为 IBM-8514 图形卡定义颜色。

〖原型〗void far setrgbpalette(int colornum, int red, int green, int blue)

〖位置〗GRAPHICS.H

〖说明〗colornum 表示需要载入的调色板条目，从 0~255 之间的数字，red、green 和 blue 负责定义颜色。这些数值仅仅使用低位字节，而且只有最重要的 6 位载入调色板。

〖参见〗setrgbcolor,setpalette。

settextjustify

〖功能〗设置图形模式下的文本对齐方式。
〖原型〗void far settextjustify(int horiz, int vert)
〖位置〗GRAPHICS.H
〖说明〗该函数影响 outtext 等函数输出的文本，使文本在水平和垂直方向对齐。

settextstyle

〖功能〗设置当前文本属性。
〖原型〗void far settextstyle(int font, int direction, int charsize)
〖位置〗GRAPHICS.H
〖参见〗settextjustify,installuserfont,gettextsettings,graphresult。

setusercharsize

〖功能〗用户为键入字符自定义的字符放大因子。
〖原型〗void far setuserchasize(int multix, int divx, intmulty, int divy)
〖位置〗GRAPHICS.H
〖参见〗gettextsettings,graphresult。

setviewport

〖功能〗设置图形输出的当前视口。
〖原型〗void far setviewport(int left, int top, int right, int bottom, int clip)
〖位置〗GRAPHICS.H
〖参见〗getviewsettings,graphresult。

setvisualpage

〖功能〗设置可视图形页码。
〖原型〗void far setvisualpage(int page)
〖位置〗GRAPHICS.H
〖说明〗有些显示适配器存在多个内存页面，可视页面是真正在屏幕上显示的页面。图形函数将输出写入由 setactivepage 定义的活动页面中。
〖参见〗setactivepage,graphresult。

setwritemode

〖功能〗设置图形画线模式。
〖原型〗void far setwritemode(int mode)
〖位置〗GRAPHICS.H
〖说明〗如果 mode 为 0，则线条覆盖屏幕上当前的内容。如果 mode 为 1，那么线条

象素和已经在屏幕上的象素之间执行一个异域操作(XOR)。

〖参见〗lineto。

textheight

〖功能〗以象素为单位，返回一个字符串的高度。

〖原型〗int far textheight(char far *textstring)

〖位置〗GRAPHICS.H

〖说明〗textheight 与图形函数一起使用，例如 outtext。

〖参见〗gettextsettings,textwidth,outtext。

textwidth

〖功能〗以象素为单位，返回一个字符串的宽度。

〖原型〗int far textwidth(char far *textstring)

〖位置〗GRAPHICS.H

〖说明〗textwidth 与图形函数一起使用，例如 outtext。

〖参见〗gettextsettings,textheight,outtext。

2.10 IO.H

access

〖功能〗确定文件是否可以访问。

〖原型〗int access(cost char *filename, int amode)

〖位置〗IO.H

〖说明〗amode=0 检查文件是否存在。

amode=2 检查是否有写权限。

如果允许执行请求的访问，则返回 0；否则返回数值-1，并且设置 errno。

_chmod

〖功能〗改变访问模式。

〖原型〗int _chmod(const char *filename, int func [, int attrib])

〖位置〗IO.H

〖说明〗如果 func 等于 0，则_chmod 返回文件属性。如果 func 等于 1，则设置属性。成功完成之后，_chmod 返回文件属性；否则返回数值-1，并设置 errno。

〖参见〗chmod。

chmod

〖功能〗更改访问模式。

〖原型〗int chmod(const char *filename, int amode)

〖位置〗IO.H
〖说明〗成功改变文件访问模式之后，chmod 返回 0；否则 chmod 返回数值-1。
〖参见〗_chmod。

chsize

〖功能〗改变文件大小。
〖原型〗int chsize(int handle, long size)
〖位置〗IO.H
〖说明〗成功则 chsize 返回 0，时白则返回-1，同时设置 errno。
〖参见〗creat fopen。

_close

〖功能〗关闭文件句柄。
〖原型〗int _close(int handle)
〖位置〗IO.H
〖说明〗见 close。

close

〖功能〗关闭文件句柄。
〖原型〗int close(int handle)
〖位置〗IO.H
〖说明〗成功完成之后，close 返回 0；否则返回-1，同时设置 errno。

_creat

〖功能〗创建一个新文件，或者覆盖一个已经存在的文件。
〖原型〗int _creat(const char *path, int attrib)
〖位置〗IO.H
〖说明〗creat 按照_fmode 规定的模式打开文件。_creat 则总是按照二进制模式打开文件。成功完成之后，返回新文件的句柄；否则返回-1，同时设置 errno。

creat

〖功能〗创建一个新文件，或者覆盖一个已经存在的文件。
〖原型〗int creat(const char *path, int amode)
〖位置〗IO.H
〖说明〗creat 按照_fmode 规定的模式打开文件。_creat 则总是按照二进制模式打开文件。成功完成之后，返回新文件的句柄；否则返回-1，同时设置 errno。

creatnew

〖功能〗创建一个新文件。

〖原型〗int creatnew(const char *path, int attrib)

〖位置〗IO.H

〖说明〗creatnew 与_creat 的功能基本一致，但是当文件已经存在时，creatnew 将返回一个错误。该函数适用于 MS DOS 3.0 或者更新的版本。

creattemp

〖功能〗在文件名指定的目录中创建一个文件。

〖原型〗int creattemp(char *path, int attrib)

〖位置〗IO.H

〖说明〗该函数与_creat 类似，但是文件名称为路径名称，必须以\结束。文件名称中应当包括完整的文件名。该函数适用于 MS DOS 3.0 或者更新的版本。

〖参见〗creat _creat。

dup

〖功能〗复制一个文件句柄。

〖原型〗int dup(int handle)

〖位置〗IO.H

〖说明〗成功完成之后，dup 返回一个新文件的句柄；否则 dup 返回-1，同时设置 errno。

〖参见〗dup2。

dup2

〖功能〗复制一个文件句柄 oldhandle，新文件句柄为 newhandle。

〖原型〗int dup2(int oldhandle, int newhandle)

〖位置〗IO.H

〖说明〗成功返回 0，错误返回-1。

〖参见〗dup。

eof

〖功能〗检查文件结尾标志。

〖原型〗int eof(int handle)

〖位置〗IO.H

〖说明〗该函数的返回值及其含义如下。

返回值	含义
1	文件结束。
0	文件没有结束。
-1	出错，设置 errno。

〖参见〗ferror,perror,EOF。

filelength

〖功能〗得到文件大小的字节数。
〖原型〗long filelength(int handle)
〖位置〗IO.H
〖说明〗出现错误时返回-1，同时设置 errno。
〖参见〗open,lseek。

getftime

〖功能〗得到文件的日期和时间。
〖原型〗int getftime(int handle, struct ftime *fftimep)
〖位置〗IO.H
〖说明〗成功返回 0，错误则返回-1，同时设置 errno。
〖参见〗setftime,open。

ioctl

〖功能〗控制 I/O 设备。
〖原型〗int ioctl(int handle, int func[, void *argdx, int argcx])
〖位置〗IO.H
〖说明〗func 为 0 或者 1 时，返回值表示设备信息(IOCTL 调用的 DX)。
func 数值从 2 到 5 时，返回值为实际传递的字节数。
func 数值为 6 或者 7 时，返回值为设备状态。
在任何情况下如果出现错误，则返回数值-1，同时设置 errno。

isatty

〖功能〗检查设备类型。
〖原型〗int isatty(int handle)
〖位置〗IO.H
〖说明〗如果为字符设备，则 isatty 返回一个非 0 整数。

lock

〖功能〗设置文件共享锁定，控制并发的文件访问。
〖原型〗int lock(int handle, long offset, long length)
〖位置〗IO.H
〖说明〗防止另一个程序读写访问从 offset 地址开始 length 个字节长的区域。
调用成功返回 0，调用失败返回-1。
〖参见〗open,unlock。

lseek

〖功能〗移动读/写文件指针。
〖原型〗long lseek(int handle, long offset, int fromwhere)

〖位置〗IO.H

〖说明〗返回从文件起始位置开始新的文件位置，以字节为单位。错误返回 01L，同时设置 errno。

〖参见〗fopen,fseek,ftell,getc,setbuf,ungetc。

_open

〖功能〗打开文件进行读写。

〖原型〗int _open(const char *filename, int oflags)

〖位置〗IO.H

〖说明〗成功完成之后，_open 返回一个文件句柄，否则返回-1。

〖参见〗open。

open

〖功能〗打开一个文件进行读写。

〖原型〗int open(const char *pathname, int access[, unsigned mode])

〖位置〗IO.H

〖说明〗成功完成之后，open 返回一个文件句柄，否则返回-1，同时设置 errno。

〖参见〗close,creat,_creat,dup,fopen,lseek,_open,read,sopen,write。

_read

〖功能〗从文件中读取。

〖原型〗int _read(int handle, void *buf, unsigned len)

〖位置〗IO.H

〖说明〗返回读入的字节数，遇到文件结尾返回 0，出现错误返回-1，同时设置 errno。

〖参见〗read,_open,_write。

read

〖功能〗从文件中读取。

〖原型〗int read(int handle, void *buf, unsigned len)

〖位置〗IO.H

〖说明〗成功完成之后，返回一个整数说明缓存区中放置的字节数，如果文件在文本模式下打开，read 不会计算返回读取字节数中的 carriage 或者 Ctrl+Z 字符。出现错误时返回-1，同时设置 errno。

〖参见〗_read,open,write。

setftime

〖功能〗获得文件的日期和时间。

〖原型〗int setftime(int handle, struct ftime *ftimep)

〖位置〗IO.H

〖说明〗成功返回 0，否则返回-1。
〖参见〗getftime。

setmode

〖功能〗设置一个打开文件的模式。
〖原型〗int setmode(int handle, int amode)
〖位置〗IO.H
〖说明〗成功返回 0，否则返回-1。
〖参见〗open,creat。

sopen

〖宏原型〗sopen(path, access, shflag, mode)
〖位置〗IO.H
〖说明〗在共享模式下打开文件。可以与就版本的 Turbo C 和其他编译程序兼容。
〖参见〗open。

tell

〖功能〗得到文件指针的当前位置。
〖原型〗long tell(int handle)
〖位置〗IO.H
〖说明〗返回当前文件指针的位置，出错则返回-1
〖参见〗fseek。

unlink

〖功能〗删除一个文件。
〖原型〗int unlink(const char *filename)
〖位置〗DOS.H IO.H STDIO.H
〖说明〗如果 filename 指定的文件属性为只读，则 unlink 操作失败。首先需要调用 chmod 函数改变文件属性。
成功返回 0，错误返回-1。
〖参见〗chmod,remove。

unlock

〖功能〗释放文件共享的锁定，控制并发访问。
〖原型〗int unlock(int handle, long offset, long length)
〖位置〗IO.H
〖说明〗成功返回 0，错误返回-1。
〖参见〗lock。

_write

〖功能〗写入一个文件。
〖原型〗int _write(int handle, void *buf, unsigned len)
〖位置〗IO.H
〖说明〗返回写入字节数目，出现错误则返回-1。
〖参见〗write。

write

〖功能〗写入一个文件。
〖原型〗int write(int handle, void *buf, unsigned nbyte)
〖位置〗IO.H
〖说明〗返回写入的字节数目，出现错误则返回-1。
〖参见〗creat,open,read,_write,lseek。

2.11 MATH.H

abs

〖功能〗得到一个整数的绝对值，abs(x)中的 x 表示一个 int 值。
〖宏原型〗int abs(int x)
〖位置〗MATH.H,STDLIB.H

acos

〖功能〗反余弦函数。
〖原型〗double acos(double x)
〖位置〗MATH.H,STDLIB.H
〖说明〗返回值范围从 0 到 π 。
〖参见〗trig,hyperb。

asin

〖功能〗反正弦函数。
〖原型〗double asin(double x)
〖位置〗MATH.H
〖说明〗返回值范围从- π /2 到 π /2
〖参见〗trig,hyperb。

atan

〖功能〗反正切函数。

〖原型〗double atan(double x)
〖位置〗MATH.H
〖说明〗返回值范围从-π/2 到π/2。
〖参见〗trig,hyperb。

atan2

〖功能〗y/x 的反正切函数。
〖原型〗double atan2(double y, double x)
〖位置〗MATH.H
〖说明〗返回值范围从-π到π。
〖参见〗trig,hyperb。

atof

〖功能〗将一个字符串转换成浮点数。
〖原型〗double atof(const char *s)
〖位置〗MATH.H,STDLIB.H
〖说明〗返回 s 经过转换之后的值，如果 s 不能转换则返回 0。
〖参见〗atoi,atol,ecvt,fcvt,sscanf,strtod。

cabs

〖功能〗得到一个复数的绝对值。
〖原型〗double cabs(struct complex znum)
〖位置〗MATH.H
〖说明〗以双精度返回 znum 的绝对值。
〖参见〗abs,fabs,labs。

ceil

〖功能〗向上取整函数。
〖原型〗double ceil(double x)
〖位置〗MATH.H
〖说明〗返回一个大于等于 x 的最小整数。
〖参见〗floor。

cos

〖功能〗余弦函数。
〖原型〗double cos(double x)
〖位置〗MATH.H
〖说明〗x 表示弧度，返回值范围从-1 到 1。
〖参见〗trig,hyperb。

cosh

〖功能〗双曲余弦函数。
〖原型〗double cosh(double x)
〖位置〗MATH.H
〖参见〗hyperb,trig。

exp

〖功能〗计算 e 的 x 次方。
〖原型〗double exp(double x)
〖位置〗MATH.H
〖参见〗frexp,ldexp,log,pow。

fabs

〖功能〗得到一个浮点数的绝对值。
〖原型〗double fabs(doubls x)
〖位置〗MATH.H
〖参见〗abs,cabs,labs。

floor

〖功能〗向下取整函数。
〖原型〗double floor(double x)
〖位置〗MATH.H
〖说明〗返回小于或者等于 x 的最大整数。
〖参见〗ceil。

fmod

〖功能〗计算 x/y 的余数，也就是 x 对 y 求模。
〖原型〗double fmod(double x, double y)
〖位置〗MATH.H
〖参见〗modf。

frexp

〖功能〗将一个双精度数分解成为尾数和指数。
〖原型〗double frexp(double value, int *exponent)
〖位置〗MATH.H
〖说明〗计算出 x(x<1)和 n，并且 value = x * (2 ** n)。返回尾数 x，并且令*exponent 指向 n。
〖参见〗exp。

hypot

〖功能〗计算直角三角形的斜边。
〖原型〗double hypot(double x, double y)
〖位置〗MATH.H

labs

〖功能〗长整数的绝对值。
〖原型〗long labs(long x)
〖位置〗MATH.H,STDLIB.H
〖参见〗abs,cabs。

ldexp

〖功能〗计算 value 乘以 2 为底的 exp 次幂得到的数值。
〖原型〗double ldexp(double value, int exp)
〖位置〗MATH.H
〖说明〗返回值 x = value * pow(2 , exp)。
〖参见〗exp,frexp。

log

〖功能〗对数函数 ln(x)。
〖原型〗double log(double x)
〖位置〗MATH.H
〖参见〗exp。

log10

〖功能〗对数函数 log 10(x)。
〖原型〗double log10(double x)
〖位置〗MATH.H
〖参见〗exp。

matherr

〖功能〗用户可修改的数学错误处理程序。
〖原型〗int matherr(struct exception *e)
〖位置〗MATH.H
〖说明〗可以运用自己的 matherr 函数控制由 math 函数库探测的错误句柄。

matherr 应当返回一个非 0 值来说明可以解决问题，否则返回 0。也可以更改返回值 e，通过初始调用方的接口返回。

modf

〖功能〗将数字分离成为整数部分和分数部分。
〖原型〗double modf(double x, double *ipart)
〖位置〗MATH.H
〖说明〗将参数值分解成两部分：整数部分和分数部分。将整数部分保存在*ipart 中，返回分数部分。

poly

〖功能〗根据参数生成一个多项式。
〖原型〗double poly(double x, int degree, double coeffs[])
〖位置〗MATH.H
〖说明〗在 x 中返回一个多项式的数值，次数为 n，系数为 coeffs[0]，...coeffs[n]。

pow

〖功能〗幂函数，x 的 y 次幂。
〖原型〗double pow(double x, double y)
〖位置〗MATH.H

pow10

〖功能〗幂函数，10 的 p 次幂。
〖原型〗double pow10(int p)
〖位置〗MATH.H

sin

〖功能〗正弦函数。
〖原型〗double sin(double x)
〖位置〗MATH.H
〖说明〗x 表示弧度，返回值范围为-1 到 1。
〖参见〗trig。

sinh

〖功能〗双曲正弦函数。
〖原型〗double sinh(double x)
〖位置〗MATH.H
〖参见〗hyperb,trig。

sqrt

〖功能〗计算平方根。

〖原型〗double sqrt(double x)
〖位置〗MATH.H
〖说明〗返回 x 的平方根。

tan

〖功能〗正切函数。
〖原型〗double tan(double x)
〖位置〗MATH.H
〖说明〗x 表示弧度，返回 x 的正切值。
〖参见〗trig,hyperb。

tanh

〖功能〗双曲正切函数。
〖原型〗double tanh(double x)
〖位置〗MATH.H
〖说明〗返回 x 的双曲正切值。
〖参见〗hyperb,trig。

2.12　MEM.H

memccpy

〖功能〗从 src 位置拷贝 n 个字节到 dest 位置。
〖原型〗*memccpy(void *dest, const void *src, int c, size_t n)
〖位置〗MEM.H,STRING.H
〖说明〗该函数拷贝一个与 c 匹配的字节之后将停止操作，返回一个指针，指向 dest 位置上 c 之后的字节；否则将返回 NULL。

memchr

〖功能〗在数组 s 的前 n 个字节中查找字符 c。
〖原型〗void *memchr(const void *s, int c, size_t n)
〖位置〗MEM.H,STRING.H
〖说明〗返回一个指针，指向 s 中出现的第一个 c。如果在数组 s 中没有出现 c，则返回 NULL。

memcmp

〖功能〗比较两个长度均为 n 的字符串 s1 和 s2。
〖原型〗int memcmp(const void *s1, const void *s2, size_t n)
〖位置〗MEM.H,STRING.H

〖说明〗如果 s1 小于 s2，返回一个小于 0 的数值。如果 s1 等于 s2 则返回一个 0 值，如果 s1 大于 s2 则返回一个大于 0 的数值。

memcpy

〖功能〗从 src 位置拷贝 n 个字节到 dest 位置。
〖原型〗*memcpy(void *dest, const void *src, size_t n)
〖位置〗MEM.H,STRING.H
〖说明〗返回 dest。

memicmp

〖功能〗比较 s1 和 s2 的前 n 个字节，忽略大小写差异。
〖原型〗int memicmp(const void *s1, const void *s2, size_t n)
〖位置〗MEM.H,STRING.H
〖说明〗如果 s1 小于 s2，返回一个小于 0 的数值。如果 s1 等于 s2 则返回一个 0 值，如果 s1 大于 s2 则返回一个大于 0 的数值。

memmove

〖功能〗从 src 拷贝 n 个字节到 dest。
〖原型〗*memmove(void *dest, const void *src, size_t n)
〖位置〗MEM.H,STRING.H
〖说明〗返回 dest。

memset

〖功能〗将 s 中的 n 个字节设置为 c。
〖原型〗void *memset(void *s, int c, size_t n)
〖位置〗MEM.H,STRING.H
〖说明〗返回 s。

movedata

〖功能〗拷贝字节。
〖原型〗void movedata(unsigned srcset, unsigned srcoff, unsigned destseg, unsigned destoff, size_t n)
〖位置〗MEM.H,STRING.H
〖说明〗从 sreseg:srcoff 拷贝 n 个字节到 destseg:destoff。
〖参见〗FP_OFF,memcpy,segread。

movmem

〖功能〗从 src 移动一定长度的字节到 dest。
〖原型〗void movmem(void *src, void *dest, unsigned length)

〖位置〗MEM.H

setmem

〖功能〗将某个内存范围设置为 value。
〖原型〗void setmem(void *dest, int len, char value)
〖位置〗MEM.H,STRING.H
〖参见〗memset,strset。

2.13　PROCESS.H

abort

〖功能〗异常终止某个进程。
〖原型〗void abort(void)
〖位置〗STDLIB.H,PROCESS.H
〖参见〗atexit,assert,exit。

exec

〖功能〗载入并运行另外一个程序。
〖原型〗int execl(char *path, char *arg0, ..., NULL)
int execle(char *path, char *arg0, ..., NULL, char **envp)
int execlp(char *path, char *arg0, ...)
int execlpe(char *path, char *arg0, ..., NULL, char **envp)
int execv(char *path, char *argv[])
int execve(char *path, char *argv[], char **envp)
int execvp(char *path, char *argv[])
int execvpe(char *path, char *argv[], char **envp)
〖位置〗PROCESS.H
〖说明〗如果成功则 exec 函数不返回任何值。如果出现错误则 exec 函数返回-1，并且设置 errno。
〖参见〗spawn,searchpath,system。

exit

〖功能〗终止程序。
〖原型〗void exit(int status)
〖位置〗STDLIB.H,PROCESS.H
〖说明〗终止程序之前，清空输出缓存，关闭文件，并且退出调用的函数。
〖参见〗abort,atext,_exit。

spawn

〖功能〗spawn 函数允许程序运行其他文件，并且在文件运行结束之后返回控制权。

成功执行之后，返回子进程的退出状态(0 表示正常终止)。如果某次子调用使用非 0 参数退出，则退出状态设置为一个非 0 值。

如果进程没有成功执行，则 spawn 函数将返回-1。

在使用如下这些 spawn 函数之前，首先需要了解将会出现多少个不同的参数。

〖原型〗int spawnl(int mode, char *path, char *arg0, ..., NULL)
int spawnle(int mode, char *path, char *arg0, ..., NULL, char **env)
int spawnlp(int mode, char *path, char *arg0, ..., NULL)
int spawnlpe(int mode, char *path, char *arg0, ..., NULL, char **env)
int spawnv(int mode, char *path, char *argv[])
int spawnve(int mode, char *path, char *argv[], char **env)
int spawnvp(int mode, char *path, char *argv[])
int spawnvpe(int mode, char *path, char *argv[], char **env)

〖位置〗PROCESS.H

〖说明〗每个 spawn 函数的结尾字母表明其使用哪一种变形。

P	在 DOS 路径中查找子进程。
l	传递固定长度的参数列表。
v	传递可变长度的参数列表。
e	传递一个 env 指针，允许改变子进程的环境。

〖参见〗abort,atexit,exit,exec,system。

system

〖功能〗执行一个 MS DOS 命令。

〖原型〗int system(const char *command)

〖位置〗PROCESS.H,STDLIB.H

〖说明〗command 可以执行一条内部 DOS 命令，例如 DIR、一个.COM 或者.EXE 程序文件，或者一个.BAT 批处理文件。成功返回 0，失败返回-1。

〖参见〗searchpath。

2.14 SETJMP.H

longjmp

〖功能〗执行非本地的跳转。

〖原型〗void longjmp(jmp_buf jmpb, int retval)

〖位置〗SETJMP.H

〖说明〗将控制权传递给调用 setjmp(初始化 jmpb)的语句。该点将继续向下执行，假设 setjmp 返回在 retval 中定义的数值。longjmp 不能返回数值 0，如果在 retval 中返回 0，则 longjmp 将返回 1。

〖参见〗ctrlbrk,signal。

setjmp

〖功能〗设置非本地跳转。

〖原型〗int setjmp(jmp_buf jmpb)

〖位置〗SETJMP.H

〖说明〗保存类似寄存器数值的相关信息，以便于 longjmp 函数可以将控制权返回给调用 setjmp 之后的语句。首次调用时返回 0。

〖参见〗longjmp。

2.15　SIGNAL.H

raise

〖功能〗向执行程序发送一个信号。

〖原型〗int raise(int sig)

〖位置〗SIGNAL.H

〖说明〗程序可以使用 raise 给自己发送信号，执行由该信号类型设置的句并(或者默认句柄)。sig 的信号类型在 signal.h 中定义。

〖参见〗signal。

signal

〖功能〗设置某个信号的句柄。

〖原型〗typedef void(*sigfun) (int subcode)
　　int signal(int sig, sigfun fname)

〖位置〗SIGNAL.H

〖说明〗当产生 sig 类型的信号时，调用 fname 指向的函数。信号类型在 signal.h 中定义。当出现例外条件，或者调用 raise 时才会产生信号。

〖参见〗raise。

2.16　STDIO.H

clearerr

〖功能〗重设错误标识。

〖原型〗void clearerr(FILE *fp)

〖位置〗STDIO.H
〖说明〗重设流 fp 中的错误标识和文件结尾标识。
〖参见〗eof,ferror,feof,perror。

fclose

〖功能〗关闭流。
〖原型〗int fclose(FILE *fp)
〖位置〗STDIO.H
〖说明〗成功返回 0，如果出现任何错误则返回 EOF。
〖参见〗fflush,fcloseall,fopen,flushall,close。

fcloseall

〖功能〗关闭所有打开的流。
〖原型〗int fcloseall(void)
〖位置〗STDIO.H
〖说明〗返回关闭流的总数目，如果出现任何错误则返回 EOF。
〖参见〗fclose,flushall。

fdopen

〖功能〗将某个流和某个文件句柄相关联。
〖原型〗FILE *fdopen(int handle, char *type)
〖位置〗STDIO.H
〖说明〗返回一个指针，指向最近打开的流，如果出现错误则返回 NULL。
〖参见〗fopen,open。

feof(f)

〖功能〗如果流 f 达到文件结尾则返回非 0 值。
〖宏原型〗int feof(FILE *stream)
〖位置〗STDIO.H
〖参见〗clearerr,eof。

ferror(f)

〖功能〗如果流 f 出现错误则返回非 0 值。
〖宏原型〗int ferror(FILE *stream)
〖位置〗STDIO.H
〖参见〗clearerr,fopen,eof。

fflush

〖功能〗清空某个流。
〖原型〗int fflush(FILE *fp)

〖位置〗STDIO.H
〖说明〗如果出现任何错误则返回 EOF。
〖参见〗flushall,fclose。

fgetc

〖功能〗从流中得到字符。
〖原型〗int fgetc(FILE *fp)
〖位置〗STDIO.H
〖说明〗该函数是宏 getc 的函数版本。
〖参见〗fgetchar,fputc。

fgetchar

〖功能〗从 stdin 中得到一个字符。
〖原型〗int fgetchar(void)
〖位置〗STDIO.H
〖说明〗函数是宏 getchar 的函数版本。

fgetpos

〖功能〗得到文件指针的当前位置。
〖原型〗int fgetpos(FILE *fp, fpos_t *pos)
〖位置〗STDIO.H
〖说明〗*pos 中保存的位置可以传递给 fsetpos，从而设置文件指针。成功返回 0，否则返回一个非 0 值。
〖参见〗fseek,fsetpos,ftell,tell。

fgets

〖功能〗从流中得到一个字符串。
〖原型〗char *fgets(char *s, int n, FILE *fp)
〖位置〗STDIO.H
〖说明〗成功则返回目的字符串 s，错误或者遇到文件结尾则返回 NULL。
〖参见〗fputs,gets。

fileno(f)

〖功能〗返回与流 f 相关联的文件句柄。
〖宏原型〗int fileno(FILE *stream)
〖位置〗STDIO.H

flushall

〖功能〗清空所有打开的流。

〖原型〗flushall(void)

〖位置〗STDIO.H

〖说明〗清空输入流的缓存区和输出流文件的输出缓存。返回一个整数，表示打开的输入输出流的数目。

〖参见〗fflush,fclose,fcloseall。

fopen

〖功能〗打开一个流。

〖原型〗FILE *fopen(const char *filename, const char *mode)

〖位置〗STDIO.H

〖说明〗如果成功则返回一个指针，指向最近打开的流，否则返回 NULL。

〖参见〗fclose,creat,open,dup,ferror,_fmode,rewind,setbuf,setmode。

fprintf

〖功能〗向流发送格式化输出。

〖原型〗int fprintf(FILE *fp, const char *format, ...)

〖位置〗STDIO.H

〖说明〗使用的格式说明符与 printf 相同，但是 fprintf 向指定流 fp 发送输出，fprintf 返回输出字节数。出现错误则返回 EOF。

〖参见〗putc,fscanf。

fputc

〖功能〗向某个流输出一个字符。

〖原型〗int fputc(int c, FILE *fp)

〖位置〗STDIO.H

〖说明〗该函数是宏 putc 的函数版本。

fputchar

〖功能〗向 stdout 输出一个字符。

〖原型〗int fputchar(int c)

〖位置〗STDIO.H

〖说明〗该函数是宏 putchar 的函数版本。

fputs

〖功能〗向某个流输出一个字符串。

〖原型〗int fputs(const char *s, FILE *fp)

〖位置〗STDIO.H

〖说明〗成功完成之后，返回最近写入的字符，否则返回 EOF。

〖参见〗fgets,puts。

fread

〖功能〗从某个流读入数据。

〖原型〗size_t fread(void *ptr, size_t size, size_t n, FILE *fp)

〖位置〗STDIO.H

〖说明〗每次读入 n 个条目的字节大小。返回实际读入的条目数(而不是字节数)。

〖参见〗fopen,fwrite。

freopen

〖功能〗将一个新文件和某个打开的流相关联。

〖原型〗FILE *freopen(const char *filename, const char *mode, FILE *stream)

〖位置〗STDIO.H

〖说明〗首先关闭流。然后打开使用路径命名的文件，并且将其与流相关联。成功则返回流，失败返回 NULL。

〖参见〗fopen。

fscanf

〖功能〗从某个流执行格式化输入。

〖原型〗int fscanf(FILE *fp, const char *format, ...)

〖位置〗STDIO.H

〖说明〗返回成功扫描、转换并且保存的输入字段数目，返回值中不包括没有保存的扫描字段。

〖参见〗getc,fprintf,scanf。

fseek

〖功能〗定位某个流的文件指针。

〖原型〗int fseek(FILE *fp, long offset, int whence)

〖位置〗STDIO.H

〖说明〗offset 表示与 whence 指定位置相对应的新位置。成功返回 0，失败则返回非 0 值。

〖参见〗ftell,fopen,lseek,rewind。

fsetpos

〖功能〗定位某个流的文件指针。

〖原型〗int fsetpos(FILE *fp, const fpos_t *pos)

〖位置〗STDIO.H

〖说明〗pos 指向的新位置是通过上一次调用 fgetpos 得到的数值。成功返回 0，失败则返回一个非 0 值。

〖参见〗fseek,fgetpos,ftell。

ftell

〖功能〗返回当前文件指针。
〖原型〗long ftell(FILE *fp)
〖位置〗STDIO.H
〖说明〗成功则返回当前的文件指针位置，出现错误则返回-1L。
〖参见〗tell,fseek,fgetpos,fsetpos。

fwrite

〖功能〗写入某个流。
〖原型〗size_t fwrite(const void *ptr, size_t size, size_t n, FILE *fp)
〖位置〗STDIO.H
〖说明〗每次写入 n 个条目大小的字节，返回实际写入的条目数(而不是字节数)。
〖参见〗fopen,fread。

getc

〖功能〗从某个流中得到一个字符。
〖宏原型〗int getc(FILE *fp)
〖位置〗STDIO.H
〖说明〗返回读入的字符。如果遇到文件结尾或者出现错误，则返回 EOF。
〖参见〗fgets,fscanf,fopen,getch,getw,fread,putc,fgetc,vfscanf,getchar。

getchar

〖功能〗从 stdin 得到字符。
〖宏原型〗int getchar(void)
〖说明〗成功则 getchar 将读入的字符转换为不加符号的 int 并返回。如果遇到文件结尾或者出现错误，则返回 EOF。
〖参见〗getc,fgetchar,ungetc,scanf,putchar,getch,vscanf,getc。

gets

〖功能〗从 stdin 得到一个字符串。
〖原型〗*gets(char *string)
〖位置〗STDIO.H
〖说明〗从 stdin 收集输入内容，直到发现换行符(\n)为止，\n 不放在 string 中。返回一个指针，指向参数 string。
〖参见〗ferror,fopen,fread,getc,puts,scanf。

getw

〖功能〗从流中得到整数。

〖原型〗int getw(FILE *fp)

〖位置〗STDIO.H

〖说明〗返回输入流中的下一个整数，如果遇到文件结尾或者出现错误则返回 EOF。使用 feof 或者 ferror 校验是否遇到文件结尾或者出现错误。

〖参见〗putw。

perror

〖功能〗系统错误消息。

〖原型〗void perror(const char *s)

〖位置〗STDIO.H

〖说明〗向 stderr 打印一条错误消息。首先打印参数字符串，然后打印一个冒号，然后是与当前 errno 数值相对应的错误消息，最后打印换行符。

〖参见〗strerror。

printf

〖功能〗向 stdout 发送格式化输出。

〖原型〗int printf(const char *format, ...)

〖位置〗STDIO.H

〖说明〗按照 format 的要求格式化数目不定的参数，然后将输出结果发送到 stdout。该函数返回输出字节数，如果出现错误则返回 EOF。

〖参见〗ecvt,puts,fprintf,scanf,putc,vprintf。

putc

〖功能〗向某个流输出一个字符。

〖原型〗int putc(int ch, FILE *fp)

〖位置〗STDIO.H

〖说明〗成功则 putc 返回字符 ch，错误则返回 EOF。

〖参见〗fputs,fputc,fwrite,fputchar,putchar,putw,frintf,vfprintf,getc,putch。

putchar

〖功能〗向 stdout 输出字符。

〖原型〗int putchar(int ch)

〖位置〗STDIO.H

〖说明〗成功则返回字符 ch，错误则返回 EOF。

〖参见〗puts,putw,fputchar,putc,printf,vprintf,getchar,putch。

puts

〖功能〗向 stdout 输出一个字符串(并且添加一个换行符)。

〖原型〗int puts(const char *s)

〖位置〗STDIO.H
〖说明〗成功完成之后，puts 返回写入的最后一个字符；否则返回 EOF。
〖参见〗putchar,gets,fputs,cputs。

putw

〖功能〗向某个流输出一个整数。
〖原型〗int putw(int w, FILE *fp)
〖位置〗STDIO.H
〖说明〗返回整数 w。
〖参见〗printf,getw。

remove(filename)

〖功能〗删除一个文件，该宏翻译了对 unlink 的调用。
〖宏原型〗int remove(filename)
〖位置〗STDIO.H
〖参见〗unlink。

rename

〖功能〗重命名一个文件。
〖原型〗int rename(const char *oldname, const char *newname)
〖位置〗STDIO.H
〖说明〗成功重命名文件之后，rename 返回 0。出现错误则返回-1，并且设置 errno。

rewind

〖功能〗重新将文件指针定位在流的开始位置。
〖原型〗void rewind(FILE *fp)
〖位置〗STDIO.H
〖参见〗fseek,fopen。

scanf

〖功能〗执行从 stdin 的格式化输入。
〖原型〗int scanf(const char *format, ...)
〖位置〗STDIO.H
〖说明〗返回成功处理的输入字段数目。要求按照 format 的要求处理输入，并且将结果放在参数指向的内存为之中。
〖参见〗atof,getc,vfscanf,cscanf,printf,vscanf,fscanf,sscanf,vsscanf。

setbuf

〖功能〗为某个流分配缓存。

〖原型〗void setbuf(FILE *fp, char *buf)

〖位置〗STDIO.H

〖说明〗如果 buf 为 NULL，则无法缓存 I/O，否则就可以缓存。缓存区的大小为 BUFSIZ 字节长。

〖参见〗fopen,setvbuf。

setvbuf

〖功能〗为某个流分配缓存。

〖原型〗setvbuf(FILE *fp, char *buf, int type, size_t size)

〖位置〗STDIO.H

〖说明〗size 表示以字节为单位的缓存区大小。使用 setvbuf 可以为流分配一个较大的缓存区。成功返回 0，否则返回一个非 0 值。

〖参见〗fopen,setbuf。

sprintf

〖功能〗向某个字符串发送格式化输出。

〖原型〗int sprintf(char *buffer, const char *format, ...)

〖位置〗STDIO.H

〖说明〗返回输出的字节数。出现错误时，返回 EOF。

〖参见〗printf,fprintf。

sscanf

〖功能〗从某个字符串执行格式化输入。

〖原型〗int sscanf(const char *buffer, const char *format, ...)

〖位置〗STDIO.H

〖说明〗返回成功扫描、转换并保存的输入字段数目。如果 sscanf 试图读入的位置超过缓存区范围，则返回值为 EOF。

〖参见〗scanf,fscanf。

_strerror

〖功能〗构建一条自定义的错误消息字符串。

〖原型〗*_strerror(const char *s)

〖位置〗STDIO.H

〖说明〗错误消息由 s、一个冒号、一个空格、最近生成的系统错误消息及一个换行符构成。s 应当小于等于 94 个字符。返回一个指针，指向错误消息字符串。

〖参见〗strerror。

strerror

〖功能〗返回指向错误消息字符串的指针。

〖原型〗char *strerror(int errnum)

〖位置〗STDIO.H STRING.H

〖说明〗返回一个指针，指向与 errnum 相关联的错误消息。

〖参见〗_strerror,perror。

tmpfile

〖功能〗以二进制模式打开一个 scratch 文件。

〖原型〗FILE *tmpfile(void)

〖位置〗STDIO.H

〖说明〗返回一个指针，指向临时文件创建的流。如果无法创建文件，则 tmpfile 返回 NULL。

〖参见〗fopen,tmpnam。

tmpnam

〖功能〗创建一个惟一的文件名称。

〖原型〗char *tmpnam(char *sptr)

〖位置〗STDIO.H

〖说明〗如果 sprt 为 NULL，则 tmpnam 返回一个指针，指向一个内部的静态对象。否则 tmpnam 返回 sptr。

〖参见〗tmpfile。

ungetc

〖功能〗将一个字符退回输入流。

〖原型〗int ungetc(int c, FILE *fp)

〖位置〗STDIO.H

〖说明〗下一次对流 fp 调用 getc(或者其他的流输入函数)时将返回 c。如果执行成功，则 ungetc 返回退回的字符。如果操作失败则返回 EOF。

〖参见〗getc。

unlink

〖功能〗删除一个文件。

〖原型〗int unlink(const char *filename)

〖位置〗DOS.H,IO.H,STDIO.H

〖说明〗如果 filename 指定的文件属性为只读，则 unlink 操作失败。首先需要调用 chmod 函数改变文件属性。

成功返回 0，错误返回-1。

〖参见〗chmod,remove。

vfprintf

〖功能〗使用参数列表向某个流发送格式化输出。
〖原型〗int vfprintf(FILE *fp, const char *format, va_list arglist)
〖位置〗STDIO.H
〖说明〗返回输出的字节数，如果出现错误则返回 EOF。
〖参见〗printf。

vfscanf

〖功能〗使用参数列表从某个流执行格式化输入。
〖原型〗int vfscanf(FILE *fp, const char *format, va_list arglist)
〖位置〗STDIO.H
〖说明〗返回成功扫描、转换并保存的输入字段数。
〖参见〗scanf。

vprintf

〖功能〗使用参数列表向 stdout 发送格式化输出。
〖原型〗int vprintf(const char *format, va_list arglist)
〖位置〗STDIO.H
〖说明〗返回输出的字节数，如果出现错误则返回 EOF。
〖参见〗printf。

vscanf

〖功能〗使用参数列表从 stdin 执行格式化输入。
〖原型〗int vscanf(const char *format, va_list arglist)
〖位置〗STDIO.H
〖说明〗返回成功扫描、转换并保存的输入字段数，遇到文件结尾则返回 EOF。
〖参见〗scanf。

vsprintf

〖功能〗使用参数列表向一个字符串发送格式化输出。
〖原型〗vsprintf(char *buffer, const char *format, va_list arglist)
〖位置〗STDIO.H
〖说明〗返回输出的字节数。如果出现错误则返回 EOF。
〖参见〗printf。

vsscanf

〖功能〗使用参数列表从一个字符串执行格式化输入。
〖原型〗int vsscanf(const char *buffer, const char *format, va_list arglist)

〖位置〗STDIO.H
〖说明〗返回成功扫描、转换并保存的输入字段数，如果遇到文件结尾则返回 EOF。
〖参见〗scanf,sscanf,vfscanf。

2.17 STDLIB.H

abort

〖功能〗异常终止某个进程。
〖原型〗void abort(void)
〖位置〗STDLIB.H,PROCESS.H
〖参见〗atexit,assert,exit。

abs

〖功能〗得到一个整数的绝对值，abs(x)中的 x 表示一个 int 值。
〖宏原型〗int abs(int x)
〖位置〗MATH.H,STDLIB.H

atexit

〖功能〗寄存器终止函数。
〖原型〗int atexit(atexit_t func)
〖位置〗STDLIB.H
〖说明〗成功返回 0，失败返回非 0 值。
〖参见〗exit,abort。

atof

〖功能〗将一个字符串转换成浮点数。
〖原型〗double atof(const char *s)
〖位置〗MATH.H STDLIB.H
〖说明〗返回 s 经过转换之后的值，如果 s 不能转换则返回 0。
〖参见〗atoi,atol,ecvt,fcvt,sscanf,strtod。

atoi

〖功能〗将一个字符串转换为一个短整数。
〖宏原型〗int atoi(const char *s)
〖位置〗STDLIB.H
〖说明〗返回输入字符串转换后的值。如果字符串无法转换，则返回值为 0。
〖参见〗atof,atol,ecvt,fcvt,sscanf,strtod。

atol

〖功能〗将一个字符串转换为一个长整数。

〖宏原型〗int atol(const char *s)

〖位置〗STDLIB.H

〖说明〗返回输入字符串转换后的值。如果字符串无法转换，则返回值为 0。

〖参见〗atof,atoi,ecvt,fcvt,sscanf,strtod。

bsearch

〖功能〗二分法查找。

〖原型〗*bsearch(const void key, const void base, size_t nelem, size_t width, int (fcmp)(const void, const void*))

〖位置〗STDLIB.H

〖说明〗返回表中的一个符合查找关键字的条目的值。如果没有找到匹配条目，则返回 0。在 bsearch 中，如果*elem1 小于*elem2 则*fcmp 的返回值小于 0，如果*elem1 等于*elem2 则返回值等于 0，如果*elem1 大于*elem2 则返回值大于 0。表必须按照升序排列。

〖参见〗lsearch qsort。

calloc

〖功能〗分配主存。

〖原型〗void *calloc(size_t nelem, size_t elsize)

〖位置〗STDLIB.H ALLOC.H

〖说明〗为 nelem 数据项的每 elsize 个字节分别分配空间，同时在空间中保存 0 值。返回一个指向新分配块的指针，如果没有足够的空间则返回 NULL。

〖参见〗malloc。

div

〖功能〗两个整数相除。

〖原型〗div_t div(int number, int denom)

〖位置〗STDLIB.H

〖说明〗每个函数都返回一个结构，其中的元素由 quot(商)和 rem(余数)组成。div 的元素为整型。

ecvt

〖功能〗将一个浮点数转换为字符串。

〖原型〗char *ecvt(double value, int ndig, int *dec, int *sign)

〖位置〗STDLIB.H

exit

〖功能〗终止程序。
〖原型〗void exit(int status)
〖位置〗STDLIB.H PROCESS.H
〖说明〗终止程序之前，清空输出缓存，关闭文件，并且退出调用的函数。
〖参见〗abort,atext,_exit。

fcvt

〖功能〗将一个浮点数转换为一个字符串。
〖原型〗char *fcvt(double value, int ndig, int *dec, int *sign)
〖位置〗STDLIB.H

free

〖功能〗释放通过 malloc 或者 calloc 分配的数据块。
〖原型〗void free(void *block)
〖位置〗STDLIB.H ALLOC.H
〖参见〗malloc,calloc,freemem。

gcvt

〖功能〗将一个浮点数转换成一个字符串。
〖原型〗char *gcvt(double value, int ndig, char *buf)
〖位置〗STDLIB.H
〖说明〗返回 buf。
〖参见〗ecvt,fcvt。

getenv

〖功能〗从环境变量中得到字符串。
〖原型〗char *getenv(const char *name)
〖位置〗STDLIB.H
〖说明〗成功则返回一个指针，指向与 name 相关的数值；如果 name 在环境中没有定义，则返回 NULL。
〖参见〗putenv,envirir。

itoa

〖功能〗将一个整数转换成一个字符串。
〖原型〗char *itoa(int value, char *string, int radix)
〖位置〗STDLIB.H
〖说明〗返回一个指针，指向目的字符串。

〖参见〗ltoa。

labs

〖功能〗求长整数的绝对值。
〖原型〗long labs(long x)
〖位置〗MATH.H STDLIB.H
〖参见〗abs,cabs。

ldiv

〖功能〗两个长整数相除。
〖原型〗ldiv_t ldiv(long lnumer, long ldenom)
〖位置〗STDLIB.H
〖说明〗每个函数都返回一个结构，其中的元素由 quot(商)和 rem(余数)组成。ldiv 的元素为长整型。

lfind

〖功能〗执行线性查找。
〖原型〗void *lfind(const void *key, const void *base, size_t *pnelem, size_t width, int (*fcmp)(const void *, const void *))
〖位置〗STDLIB.H
〖说明〗该函数使用一个用户自定义的程序(fcmp)，在顺序记录的书组中查找 key 值。
数组中一共有 pnelem 个记录，每个的宽度为 width 字节，并且从 base 指向的内存位置开始。
返回表中与查找关键字匹配的第一个条目的地址。如果没有找到匹配条目，则 lfind 返回 0。如果*elem1 等于*elem2 则*fcmp 程序必须返回 0，否则返回非 0 值。
〖参见〗bsearch,qsort。

_lrotl

〖功能〗将一个长整数向左侧循环。
〖原型〗_lrotl(unsigned long val, int count)
〖位置〗STDLIB.H
〖说明〗函数返回 val 循环 count 位后的数值。
〖参见〗_rotl。

_lrotr

〖功能〗将一个长整数向右侧循环。
〖原型〗_lrotr(unsigned long val, int count)
〖位置〗STDLIB.H
〖说明〗函数返回 val 循环 count 位后的数值。

〖参见〗_rotl。

lsearch

〖功能〗执行线性查找。

〖原型〗void *lsearch(const void *key, void *base, size_t *pnelem, size_t width, int (*fcmp)(const void *, const void *))

〖位置〗STDLIB.H

〖说明〗该函数使用一个用户自定义的程序(fcmp)，在顺序记录的书组中查找 key 值。数组中一共有 pnelem 个记录，每个的宽度为 width 字节，并且从 base 指向的内存位置开始。

返回表中与查找关键字匹配的第 1 个条目的地址。如果没有找到匹配条目，则 lsearch 将其添加到列表结尾。如果*elem1 等于*elem2 则*fcmp 程序必须返回 0，否则返回非 0 值。

〖参见〗bsearch,qsort。

ltoa

〖功能〗将一个长整数转换成一个字符串。

〖原型〗char *ltoa(long value, char *string, int radix)

〖位置〗STDLIB.H

〖说明〗使用十进制表示，则使用 radix=10。使用十六进制表示，则使用 radix=16。返回一个指针，指向参数 string。

〖参见〗itoa,ultoa。

malloc

〖功能〗分配主存。

〖原型〗void *malloc(size_t size)

〖位置〗ALLOC.H,STDLIB.H

〖说明〗大小按照字节计算。返回一个指向新分配数据块的指针，如果没有足够的空间创建数据块，则返回 NULL。如果 size == 0 也返回 NULL。

〖参见〗allocmem,free,calloc,realloc,farmalloc。

max

〖功能〗产生联机代码，得到两个整数中的最大值。

〖宏原型〗max(a,b)

〖位置〗STDLIB.H

min

〖功能〗产生联机代码，得到两个整数中的最小值。

〖宏原型〗min(a,b)

〖位置〗STDLIB.H

putenv

〖功能〗在当前环境变量中添加字符串。

〖原型〗int putenv(const char *name)

〖位置〗STDLIB.H

〖说明〗成功则 putenv 返回 0，失败返回-1。

〖参见〗getenv。

qsort

〖功能〗使用 Quicksort 程序进行排序。

〖原型〗void qsort(void *base, size_t nelem, size_t width, int (*fcmp)(const void *, const void *))

〖位置〗STDLIB.H

〖说明〗为 base 中 width 大小的 nelem 个条目排序。排名由用户自定义程序 fcmp 决定，如果 elem1 小于 elem2 则返回值小于 0，如果 elem1 等于 elem2 则返回值等于 0，如果 elem1 大于 elem2 则返回值大于 0。

〖参见〗bsearch,lsearch。

rand

〖功能〗随机数生成器。

〖原型〗int rand(void)

〖位置〗STDLIB.H

〖说明〗返回一个从 0 到 RAND_MAX 范围内的随机数。RAND_MAX 在 stdlib.h 中定义为 32 767。

〖参见〗srand,randomize,random。

random

〖功能〗返回一个从 0 到(num-1)之间的整数。

〖宏原型〗int random(int num)

〖位置〗STDLIB.H

〖参见〗rand,srand,randomize。

randomize

〖功能〗使用一个随机数值初始化随机数生成器。因为使用了 time 函数，因此使用该程序时应当包括 time.h。

〖宏原型〗void randomize(void)

〖位置〗STDLIB.H

〖参见〗rand,srand,random。

realloc

〖功能〗重新分配主存。

〖原型〗void *realloc(void *block, size_t size)

〖位置〗ALLOC.H,STDLIB.H

〖说明〗尝试将原先分配的数据块扩大或者缩小至 size 字节值。返回重新分配之后与原有地址不同的数据块的地址。如果数据块不能重新分配，或者 size == 0，则返回 NULL。

〖参见〗malloc,free。

_rotl

〖功能〗将一个数值向左侧循环。

〖原型〗_rotl(unsigned value, int count)

〖位置〗STDLIB.H

〖说明〗函数返回 value 循环 count 位后的数值。

〖参见〗_lrotl。

_rotr

〖功能〗将一个数值向右侧循环。

〖原型〗_rotr(unsigned value, int count)

〖位置〗STDLIB.H

〖说明〗函数返回 value 循环 count 位后的数值。

〖参见〗_lrotl。

srand

〖功能〗初始化随机数产生器。

〖原型〗void srand(unsigned seed)

〖位置〗STDLIB.H

〖说明〗srand 不返回数值。

〖参见〗rand,random,randomize。

strtod

〖功能〗将字符串转换成为双精度的浮点数。

〖原型〗double strtod(const char *s, char **endptr)

〖位置〗STDIO.H

〖说明〗返回一个双精度数 s，这里的 s 是一个字符序列。字符串必须符合如下格式。

```
[ws] [sn] [ddd] [.] [ddd] [fmt[sn]ddd]
```

〖参见〗atof。

strtol

〖功能〗使用给定的基数将字符串转换成为一个长整数。

〖原型〗long strtol(const char *s, char **endptr, int radix)

〖位置〗STDIO.H

〖说明〗返回 s 转换后的数值，如果出现错误则返回 0，这里的 s 是一个字符序列。字符串必须符合如下格式。

```
[ws] [sn] [0] [x] [ddd]
```

〖参见〗atol,strtoul。

strtoul

〖功能〗按照给定的基数将字符串转换成为无符号的长整数。

〖原型〗unsigned long strtoul(const char *s, char **endptr, int radix)

〖位置〗STDIO.H

〖说明〗返回字符转换后的数值，出现错误则返回 0。

〖参见〗atol,strtol。

swab

〖功能〗交换字节。

〖原型〗void swab(char *from, char *to, int nbytes)

〖位置〗STDIO.H

〖说明〗拷贝 nbyte 个字节，然后将奇数字节和偶数字节相互交换：

```
to[0] = from[1]
to[1] = from[0]
...
```

system

〖功能〗执行一个 MS DOS 命令。

〖原型〗int system(const char *command)

〖位置〗PROCESS.H,STDLIB.H

〖说明〗command 可以执行一条内部 DOS 命令，例如 DIR、一个.COM 或者.EXE 程序文件，或者一个.BAT 批处理文件。成功返回 0，失败返回-1。

〖参见〗searchpath。

ultoa

〖功能〗将一个无符号的长整数转换为字符串。

〖原型〗char *ultoa(unsigned long value, char *string, int radix)

〖位置〗STDIO.H

〖说明〗返回一个指针指向 string，该函数没有错误返回值。

〖参见〗itoa。

2.18 STRING.H

memccpy

〖功能〗从 src 位置拷贝 n 个字节到 dest 位置。

〖原型〗*memccpy(void *dest, const void *src, int c, size_t n)

〖位置〗MEM.H,STRING.H

〖说明〗该函数拷贝一个与 c 匹配的字节之后将停止操作，返回一个指针，指向 dest 位置上 c 之后的字节；否则将返回 NULL。

memchr

〖功能〗在数组 s 的前 n 个字节中查找字符 c。

〖原型〗void *memchr(const void *s, int c, size_t n)

〖位置〗MEM.H,STRING.H

〖说明〗返回一个指针，指向 s 中出现的第 1 个 c；如果在数组 s 中没有出现 c，则返回 NULL

memcmp

〖功能〗比较两个长度均为 n 的字符串 s1 和 s2。

〖原型〗int memcmp(const void *s1, const void *s2, size_t n)

〖位置〗MEM.H,STRING.H

〖说明〗如果 s1 小于 s2，返回一个小于 0 的数值，如果 s1 等于 s2 则返回一个 0 值；如果 s1 大于 s2 则返回一个大于 0 的数值。

memcpy

〖功能〗从 src 位置拷贝 n 个字节到 dest 位置。

〖原型〗*memcpy(void *dest, const void *src, size_t n)

〖位置〗MEM.H,STRING.H

〖说明〗返回 dest。

memicmp

〖功能〗比较 s1 和 s2 的前 n 个字节，忽略大小写差异。

〖原型〗int memicmp(const void *s1, const void *s2, size_t n)

〖位置〗MEM.H,STRING.H

〖说明〗如果 s1 小于 s2，返回一个小于 0 的数值，如果 s1 等于 s2 则返回一个 0 值；如果 s1 大于 s2 则返回一个大于 0 的数值。

memmove

〖功能〗从 src 拷贝 n 个字节到 dest。

〖原型〗*memmove(void *dest, const void *src, size_t n)

〖位置〗MEM.H,STRING.H

〖说明〗返回 dest。

memset

〖功能〗将 s 中的 n 个字节设置为 c。

〖原型〗void *memset(void *s, int c, size_t n)

〖位置〗MEM.H,STRING.H

〖说明〗返回 s。

movedata

〖功能〗拷贝字节。

〖原型〗void movedata(unsigned srcset, unsigned srcoff, unsigned destseg, unsigned destoff, size_t n)

〖位置〗MEM.H,STRING.H

〖说明〗从 sreseg:srcoff 拷贝 n 个字节到 destseg:destoff。

〖参见〗FP_OFF,memcpy,segread。

movmem

〖功能〗从 src 移动一定长度的字节到 dest。

〖原型〗void movmem(void *src, void *dest, unsigned length)

〖位置〗MEM.H

strchr

〖功能〗在 str 中查找 c。

〖原型〗char *strchr(const char *str, int c)

〖位置〗STRING.H

〖说明〗返回一个指针，指向 str 中出现的一个字符 c；如果 str 中没有出现 c，则 strchr 返回 NULL。

strcmp

〖功能〗比较 s2 和 s1。

〖原型〗int strcmp(const char *s1, const char *s2)

〖位置〗STRING.H

〖说明〗如果 s1 小于 s2 则返回一个小于 0 的数值，如果 s1 等于 s2 则返回值等于 0；如果 s1 大于 s2 则返回一个大于 0 的数值。这里执行的是带符号的比较。

strcmpi

〖宏原型〗int strcmpi(const char*s1,const char *s2)
〖位置〗STRING.H
〖参见〗stricmp。
〖说明〗程序作为宏执行，可以与其他编译程序匹配。

strcpy

〖功能〗将字符串从 src 拷贝到 dest。
〖原型〗char *strcpy(char *dest, const char *src)
〖位置〗STRING.H
〖说明〗返回 dest。

strcspn

〖功能〗查找 s1 中包含的全部是 s2 中不存在字符的初始段长度。
〖原型〗size_t strcspn(const char *s1, const char *s2)
〖位置〗STRING.H

strdup

〖功能〗得到 s 的复制拷贝。
〖原型〗char *strdup(const char *s)
〖位置〗STRING.H
〖说明〗返回一个指针，指向复制的 s，如果无法为复制分配足够的空间则返回 NULL。如果不再需要由 strdup 分配的空间，则由用户负责释放这部分空间。
〖参见〗free。

_strerror

〖功能〗构建一条自定义的错误消息字符串。
〖原型〗*_strerror(const char *s)
〖位置〗STDIO.H
〖说明〗错误消息由 s、一个冒号、一个空格、最近生成的系统错误消息以及一个换行符构成。s 应当小于等于 94 个字符。返回一个指针，指向错误消息字符串。
〖参见〗strerror。

strerror

〖功能〗返回指向错误消息字符串的指针。
〖原型〗char *strerror(int errnum)
〖位置〗STDIO.H,STRING.H
〖说明〗返回一个指针，指向与 errnum 相关联的错误消息。

〖参见〗_strerror perror。

stricmp

〖功能〗比较 s2 和 s1，忽略大小写差异。
〖原型〗int stricmp(const char *s1, const char *s2)
〖位置〗STRING.H
〖说明〗如果 s1 小于 s2 则返回值小于 0，如果 s1 等于 s2 则返回值等于 0，如果 s1 大于 s2 则返回值大于 0。这里执行的是带符号的比较。
string.h 中定义的宏 strcmpi 与 stricmp 功能相同。

setmem

〖功能〗将某个内存范围设置为 value。
〖原型〗void setmem(void *dest, int len, char value)
〖位置〗MEM.H,STRING.H
〖参见〗memset,strset。

stpcpy

〖功能〗将一个字符串拷贝到另一个字符串中。
〖原型〗char *stpcpy(char *dest, const char *src)
〖位置〗STRING.H
〖说明〗除了返回 dest + strlen(src)之外，功能与 strcpy 相同。

strcat

〖功能〗将 src 添加到 dest 之后。
〖原型〗char *strcat(char *dest, const char *src)
〖位置〗STRING.H
〖说明〗返回 dest。

strncmp

〖功能〗比较 s2 和 s1 中最多 maxlen 个字符。
〖原型〗int strncmp(const char *s1, const char *s2, size_t maxlen)
〖位置〗STRING.H
〖说明〗如果 s1 小于 s2 则返回值小于 0，如果 s1 等于 s2 则返回值等于 0；如果 s1 大于 s2 则返回值大于 0。这里执行的是带符号的比较。

strncmpi

〖宏原型〗strncmpi(const char *s1,const char *s2 int n)
〖位置〗STRING.H
〖参见〗strnicmp。

〖说明〗程序作为宏执行，可以与其他编译程序匹配。

strncpy

〖功能〗从 src 拷贝最多 maxlen 个字符到 dest。

〖原型〗char *strncpy(char *dest, const char *src, size_t maxlen)

〖位置〗STRING.H

〖说明〗如果拷贝 maxlen 个字符，则不添加空字符，dest 区域中的内容不能是以空结束的字符串。该函数返回 dest。

strnicmp

〖功能〗比较 s1 和 s2 中最多 n 个字符，忽略大小写差异。

〖原型〗int strnicmp(const char *s1, const char *s2, size_t maxlen)

〖位置〗STRING.H

〖说明〗如果 s1 小于 s2 则返回值小于 0，如果 s1 等于 s2 则返回值等于 0；如果 s1 大于 s2 则返回值大于 0。这里执行的是带符号的比较。

string.h 中定义的宏 strncmpi 与 strnicmp 功能相同。

strnset

〖功能〗将 s 中前 n 个字符设置为 c。

〖原型〗char *strnset(int *s, int c, size_t n)

〖位置〗STRING.H

〖说明〗当设置了 n 个字符或者发现 NULL 字符时停止操作。返回指向 s 的指针。

strpbrk

〖功能〗扫描 s1 中出现的第一个 s2 中的字符。

〖原型〗char *strpbrk(const char *s1, const char *s2)

〖位置〗STRING.H

〖说明〗返回一个指针，指向第 1 次出现 s2 中字符的位置；如果 s1 中没有出现任何 s2 的字符，则返回 NULL。

strrchr

〖功能〗在 s 中查找最后一次出现的 c。

〖原型〗char *strrchr(const char *s, int c)

〖位置〗STRING.H

〖说明〗返回一个指针，指向最后一次出现的字符 c；如果 s 中没有出现过 c，则返回 NULL。

strlen

〖功能〗得到 s 的长度。

〖原型〗size_t strlen(const char *s)
〖位置〗STRING.H
〖说明〗返回 s 中的字符个数，不计算终止的空字符。

strlwr

〖功能〗将 s 转换为小写字母。
〖原型〗char *strlwr(char *s)
〖位置〗STRING.H
〖说明〗返回指向 s 的指针。

strncat

〖功能〗在 dest 后添加 src 中最多 maxlen 个字符。
〖原型〗char *strncat(char *dest, const char *src, size_t maxlen)
〖位置〗STRING.H
〖说明〗返回 dest。

strrev

〖功能〗反转 s 中的所有字符(除了结束的空字符之外)。
〖原型〗char *strrev(char *s)
〖位置〗STRING.H
〖说明〗返回一个指针，指向反转后的字符串。

strset

〖功能〗设置 s 中的所有字符为 c。
〖原型〗char *strset(char *s, int c)
〖位置〗STRING.H
〖说明〗当遇到第 1 个空字符时退出，返回一个指针指向 s。

strspn

〖功能〗查找 s1 中包含的全部是 s2 中存在字符的初始段长度。
〖原型〗size_t strspn(const char *s1, const char *s2)
〖位置〗STRING.H

strstr

〖功能〗查找 s1 中的第 1 次出现子字符串 s2 的位置。
〖原型〗char *strstr(const char *s2, const char *s2)
〖位置〗STRING.H
〖说明〗返回一个指针，指向 s1 中包含 s2 的元素(指向 s1 中的 s2)；如果 s1 中找不到 s2，则返回 NULL。

strtok

〖功能〗在 s1 查找第 1 个不包含在 s2 中的标记。

〖原型〗char *strtok(char *s1, const char *s2)

〖位置〗STRING.H

〖说明〗s2 定义字符分离器。strtok 将字符串 s2 解释成为一系列使用 s2 中的字符分割范围的标记。

如果没有在 s1 中发现任何标记，则返回 NULL。如果发现标记，则在 s1 中的标记之后插入一个空字符，然后 strtok 返回一个指向标记的指针。

下一次使用 NULL 调用 strtok 时，将使用原来的 s1 字符串，并且从上一次发现标记的位置之后开始。

strupr

〖功能〗将 s 中的所有字符转换为大写字母。

〖原型〗char *strupr(char *s)

〖位置〗STRING.H

〖说明〗返回一个指向 s 的指针。

2.19 SYS\STAT.H

fstat

〖功能〗得到打开文件的有关信息。

〖原型〗int fstat(int handle, struct stat *statbuf)

〖位置〗SYS\STAT.H

〖说明〗如果成功则返回 0，出现错误则返回-1，并且设置 errno。

〖参见〗stat,chmod,access。

stat

〖功能〗得到打开文件的有关信息。

〖原型〗int stat(char *path, struct stat *statbuf)

〖位置〗SYS\STAT.H

〖说明〗如果成功则返回 0，出现错误则返回-1，并且设置 errno。

〖参见〗access,fstat,chmod。

2.20　SYS\TIMEB.H

ftime

〖功能〗在 timeb 结构中保存当前时间。

〖原型〗void ftime(struct timeb *buf)

〖位置〗SYS\TIMEB.H

〖说明〗timeb 结构中包含从 1970 年 1 月 1 日开始计算的秒数，以及一个单独的毫秒字段。同时返回本地时区和一个日夜标志。

〖参见〗time,asctime,ftime,stucture,tzset。

2.21　TIME.H

asctime

〖功能〗将日期和时间转换为 ASCII 码。

〖原型〗char *asctime(const struct tm *tblock)

〖位置〗TIME.H

〖说明〗返回一个指针，指向包含日期和时间的字符串。该字符串是静态值，每次调用时都覆盖原值。

〖参见〗ctime,difftime,gmtime,localtime,time。

clock

〖功能〗返回从程序启动时开始的时钟滴答次数。

〖原型〗clock_t clock()

〖位置〗TIME.H

〖说明〗返回从程序开始执行到现在使用的处理器时间，按照时钟滴答为单位衡量。返回值除以 CLK_TCK 可以转换为秒。

〖参见〗time。

ctime

〖功能〗将日期和时间转换为一个字符串。

〖原型〗char *ctime(const time_t *tiem)

〖位置〗TIME.H

〖说明〗返回一个指针，指向包含日期和时间的字符串。该字符串为一个静态值，每次调用都将覆盖原值。

〖参见〗asctime,difftime,gmtime,localtime,time,settime,tzset。

difftime

〖功能〗计算 2 个时间之间的差异。
〖原型〗double difftime(time_t time2, time_t time1)
〖位置〗TIME.H
〖说明〗返回从 time1 到 time2 之间相距的时间，以秒为单位。
〖参见〗ctime,asctime,gmtime,localtime,time。

gmtime

〖功能〗将日期和时间转换为格林威治时间。
〖原型〗struct tm *gmtime(const time_t *timer)
〖位置〗TIME.H
〖说明〗返回一个指针指向故障时间的结构。该结构是一个静态变量，每次调用 gmtime 都将覆盖原值。
〖参见〗ctime。

localtime

〖功能〗将日期和时间转换为一个结构。
〖原型〗struct tm *localtime(const time_t *timer)
〖位置〗TIME.H
〖说明〗返回故障时间的结构，该结构是一个静态值，每次调用将覆盖原值。
〖参见〗ctime。

stime

〖功能〗设置时间。
〖原型〗int stime(time_t *tp)
〖位置〗TIME.H
〖说明〗返回数值 0。

time

〖功能〗得到时间值。
〖原型〗time_t time(time_t *timer)
〖位置〗TIME.H
〖说明〗得到从 1970 年 1 月 1 日 00:00:00 GMT 时起到当前时间为止的秒数，并且将该数值保存在 timer 指向的位置中。同时返回该数值。
〖参见〗struct,time,settime。

tzset

〖功能〗与 UNIX 时间相匹配。

〖原型〗void tzset(void)
〖位置〗TIME.H
〖说明〗tzset 试图找到 TZ=...形式的环境字符串。该字符串负责定义时区。
〖参见〗ctime  。

第 3 章

Microsoft C 函数

本章分类介绍每个 Microsoft C 6.0 函数的功能、原型、原型所在的包含文件、使用说明等内容，对一些重要函数还给出了样例。

3.1 以字为单位的内存处理函数

_fmemccpy

〖功能〗基于字符的内存处理函数。

〖原型〗void_far*_fmemccpy(void_far *dest，void_far *src，int c，unsigned count)

〖位置〗string.h， memory.h

〖说明〗memccpy 和_fmemccpy 函数将 src 指向的 0 个或多个字节复制到 dest 所指向的存储空间，直到将字符 c 复制完或复制了 count 个字节为止。若函数复制了字符 c，则返回指向 dest 存储空间中该字符的下一个字节的指针(或远程指针)。如果没有复制字符 c 则返回 NULL。

〖参见〗_fmemchr,_fmemcmp,_fmemcpy,_fmemset。

_fmemchr

〖功能〗在 buf 指向单元的前 count 个字节中查找字符 c 首次出现的位置。找到字符 c 或查找了 count 个字节后函数结束。

〖原型〗void _far *_fmemchr(void_far *buf，int c，size_t count)

〖位置〗string.h ，memory.h

〖说明〗_fmemchr 函数是 memchr 函数的独立模式(大规模)形式，可以通过任何程序中的任何指针调用。

如果查找成功，函数 memchr 和_fmemchr 返回字符 c 在 buf 中首次出现位置的指针。否则返回 NULL。

〖参见〗_fmemccpy,_fmemcmp,_fmemcpy,_fmenset。

_fmemcmp

〖功能〗比较 buf1 和 buf2 的前 count 个字节，并根据比较结果返回一个整数。

〖原型〗int _fmemcmp(void _far *buf1，void _far *buf2，size_t count)

〖位置〗string.h，memory.h。

〖说明〗该函数执行的规则如下。

返回值	含义
<0	buf1 指向单元的内容小于 buf2 指向单元的内容。
=0	buf1 指向单元的内容同 buf2 指向单元相同。
>0	buf1 指向单元的内容大于 buf2 指向单元的内容。

_fmemcmp 和_fmemicmp 函数是 memcmp 和 memicmp 函数的独立模式形式(大规模)。_f…形式的函数可以用程序中的任何指针调用。

memicmp 和_fmemicmp 函数是 memcmp 和_fmemcmp 函数的不灵敏版本。memcmp 的函数版本同其固有版本在语义上有所区别。函数版本支持压缩或大规模程序中的大指针，但固有版本不支持。

〖参见〗_fmemccpy,_fmemchr,_fmemcpy,_fmemicmp,_fmemset。

_fmemicmp

〖功能〗比较 buf1 和 buf2 的前 count 个字节，并根据比较结果返回一个整数。

〖原型〗int _fmemicmp(void _far *buf1，void _far *buf2，size_t count)

〖位置〗string.h ，memory.h。

〖说明〗函数执行的规则如下。

返回值	含义
<0	buf1 指向单元的内容小于 buf2 指向单元的内容。
=0	buf1 指向单元的内容同 buf2 指向单元相同。
>0	buf1 指向单元的内容大于 buf2 指向单元的内容。

_fmemcmp 和_fmemicmp 函数是 memcmp 和 memicmp 函数的独立模式形式(大规模)。_f…形式的函数可以用程序中的任何指针调用。

memicmp 和_fmemicmp 函数是 memcmp 和_fmemcmp 函数的不灵敏版本。memcmp 的函数版本同其固有版本在语义上有所区别。函数版本支持压缩或大规模程序中的大指针，但固有版本不支持。

〖参见〗_fmemccpy,_fmemchr,_fmemcpy,_fmemcmp,_fmemset。

_fmemmove

〖功能〗从 src 指向空间向 dest 指向空间复制 count 个字符。如果 src 指向空间和 dest 指向空间发生交迭，memmove 和_fmemmove 函数确保在覆盖之前先将 src 的交迭部分的原始内容复制。

〖原型〗void *_far *_fmemmove(void _far *dest，void _far *src，size_t count)

〖位置〗string.h

〖说明〗_fmemmove 函数是 memmove 函数的独立模式(大规模)，可以用程序中的任何指针调用。

memmove 和_fmemmove 函数返回指针 dest 的值。

〖参见〗_fmemccpy,_fmemcpy。

_fmemset

〖功能〗将 dest 指向空间的前 count 个字节设置成字符 c。

〖原型〗void _far *_fmemset(void _far *dest，int c，size_t count)

〖位置〗string.h，memory.h

〖说明〗_fmemset 函数是 memset 函数的独立模式(大规模)，可以通过程序中的任何指

针调用。

memeset 的函数版本和固有版本之间有语义差别。函数版本支持压缩、大型和大规模程序中的大指针，固有版本不支持。

memset 和_fmemset 函数返回指针 dest 的值。

〖参见〗_fmemccpy,_fmemchr,_fmemcmp,_fmemcpy。

memccpy

〖功能〗基于字符的内存处理函数。

〖原型〗void *memccpy(void *dest，void *src，int c，unsigned count)

〖位置〗string.h， memory.h

〖说明〗memccpy 和_fmemccpy 函数将 src 指向的 0 个或多个字节复制到 dest 所指向的存储空间，直到将字符 c 复制完或复制了 count 个字节为止。若函数复制了字符 c，则返回指向 dest 存储空间中该字符的下一个字节的指针(或远程指针)。如果没有复制字符 c 则返回 NULL。

〖参见〗memchr,memcmp,memcpy,memset。

memchr

〖功能〗在 buf 指向单元的前 count 个字节中查找字符 c 首次出现的位置。找到字符 c 或查找了 count 个字节后函数结束。

〖原型〗void *memchr(void *buf，int c， size_t count)

〖位置〗string.h ，memory.h

〖说明〗_fmemchr 函数是 memchr 函数的独立模式(大规模)形式，可以通过程序中的任何指针调用。

如果查找成功，函数 memchr 和_fmemchr 返回字符 c 在 buf 中首次出现位置的指针。否则返回 NULL。

〖参见〗memccpy,memcmp,memcpy,menset。

memcmp

〖功能〗比较 buf1 和 buf2 的前 count 个字节，并根据比较结果返回一个整数。

〖原型〗int memcmp(void *buf1，void *buf2，size_t count)

〖位置〗string.h ，memory.h。

〖说明〗函数执行的规则如下。

返回值	含义
<0	buf1 指向单元的内容小于 buf2 指向单元的内容。
=0	buf1 指向单元的内容同 buf2 指向单元相同。
>0	buf1 指向单元的内容大于 buf2 指向单元的内容。

_fmemcmp 和_fmemicmp 函数是 memcmp 和 memicmp 函数的独立模式形式(大规模)。

_f…形式的函数可以用程序中的任何指针调用。

memicmp 和_fmemicmp 函数是 memcmp 和_fmemcmp 函数的不灵敏版本。memcmp 的函数版本同其固有版本在语义上有所区别。函数版本支持压缩或大规模程序中的大指针，但固有版本不支持。

〖参见〗memccpy,memchr,memcpy,memicmp,memset。

memicmp

〖功能〗比较 buf1 和 buf2 的前 count 个字节，并根据比较结果返回一个整数。

〖原型〗int memicmp(void *buf1，void *buf2，size_t count)

〖位置〗string.h，memory.h。

〖说明〗函数执行的规则如下。

返回值	含义
<0	buf1 指向单元的内容小于 buf2 指向单元的内容。
=0	buf1 指向单元的内容同 buf2 指向单元相同。
>0	buf1 指向单元的内容大于 buf2 指向单元的内容。

_fmemcmp 和_fmemicmp 函数是 memcmp 和 memicmp 函数的独立模式形式(大规模)。_f…形式的函数可以用程序中的任何指针调用。

memicmp 和_fmemicmp 函数是 memcmp 和_fmemcmp 函数的不灵敏版本。memcmp 的函数版本同其固有版本在语义上有所区别。函数版本支持压缩或大规模程序中的大指针，但固有版本不支持。

〖参见〗memccpy,memchr,memcpy,memcmp,memset。

memmove

〖功能〗从 src 指向空间向 dest 指向空间复制 count 个字符。如果 src 指向空间和 dest 指向空间发生交迭，memmove 和_fmemmove 函数确保在覆盖之前先将 src 的交迭部分的原始内容复制。

〖原型〗void *memmove(void *dest，void *src，size_t count)

〖位置〗string.h

〖说明〗_fmemmove 函数是 memmove 函数的独立模式(大规模)，可以用程序中的任何指针调用。

memmove 和_fmemmove 函数返回指针 dest 的值。

〖参见〗memccpy,memcpy。

memset

〖功能〗将 dest 指向空间的前 count 个字节设置成字符 c。

〖原型〗void *memset(void *dest，int c，size_t count)

〖位置〗string.h，memory.h

〖说明〗_fmemset 函数是 memset 函数的独立模式(大规模)，可以通过程序中的任何指针调用。

memeset 的函数版本和固有版本之间存在语义差别。函数版本支持压缩、大型和大规模程序中的大指针，固有版本不支持。

memset 和_fmemset 函数返回指针 dest 的值。

〖参见〗memccpy，memchr，memcmp，memcpy。

movedata

〖功能〗movedata 函数从指定的源地址 srcseg：srcoff 向目的地址 destseg：destoff 中复制 count 个字节。

〖原型〗void movedata(unsigned srcseg，unsigned srcoff，unsigned destseg，
unsigned destoff，unsigned count)

〖位置〗memory.h，string.h

〖说明〗movedata 函数用于在小型数据程序中转移远程数据，较新的独立模式的_fmemcpy 和_fmemmove 函数可以代替 movedata 函数。在大规模程序中，也可使用 memcpy 或 memmove 函数。

表示段地址的 srcseg 和 destseg 参数可以使用 segread 函数或 FP_SEG 宏获得。

movedata 函数不能够处理空间交迭(部分目的地址空间和部分源地址空间相同)的情况，而 memmove 函数则能够正确处理空间交迭情况。

该函数无返回值。

〖参见〗FP_OFF,FP_SEG,memcpy,memmove,segread。

_outmem

〖功能〗_outmem 函数显示 text 指向的字符串，length 参数指定字符串的长度。

〖原型〗void _outmem(unsigned char _far *text，short length)

〖位置〗graph.h

〖说明〗与_outtext 函数不同的是，_outmem 函数逐字输出所有字符，包括 0x11，0x13 和 0x00 等图形字符。该函数不提供输出格式，从当前文本位置开始使用当前的文本颜色。若要用指定字体输出文本，则必须使用_outgtext 函数。

该函数无返回值。

〖参见〗_outtext,_settextcolor,_settextposition,_settextwindow。

swab

〖功能〗swab 函数将 src 的 n 个字节的相邻字节两两交换后复制到 dest 中，整数 n 应该是偶数以保证交换。

〖原型〗void swab(char *src，char *dest，int n)

〖位置〗stdlib.h

〖说明〗swab 函数通常用于完成向使用不同字节顺序的机器传送二进制数据的预备工作。

该函数无返回值。

3.2　单个字符处理函数

isalnum

〖功能〗字母数字字符测试函数。
〖原型〗int isalnum(int c)
〖位置〗ctype.h

isalpha

〖功能〗字母测试函数。
〖原型〗int isalpha(int c)
〖位置〗ctype.h

isascii

〖功能〗ASCII 字符测试函数。
〖原型〗int isascii(int c)
〖位置〗ctype.h

iscntrl

〖功能〗控制字符测试函数。
〖原型〗int iscntrl(int c)
〖位置〗ctype.h

isdigit

〖功能〗十进制数字测试函数。
〖原型〗int isdigit(int c)
〖位置〗ctype.h

isgraph

〖功能〗非空格的可打印字符测试函数。
〖原型〗int isgraph(int c)
〖位置〗ctype.h

islower

〖功能〗小写字母测试函数。
〖原型〗int islower(int c)
〖位置〗ctype.h

isprint

〖功能〗可打印字符测试函数。

〖原型〗int isprint(int c)
〖位置〗ctype.h

ispunct

〖功能〗标点符号测试函数。
〖原型〗int ispunct(int c)
〖位置〗ctype.h

isspace

〖功能〗空白字符测试函数。
〖原型〗int isspace(int c)
〖位置〗ctype.h

isupper

〖功能〗大写字母测试函数。
〖原型〗int isupper(int c)
〖位置〗ctype.h

isxdigit

〖功能〗十六进制数字测试函数。
〖原型〗int isxdigit(int c)
〖位置〗ctype.h
〖说明〗这是一组测试给定的整数值的程序，如果整数满足测试条件，则返回一个非 0 值；否则返回 0。这里假定在 ASCII 字符集环境中。这些程序即可以作为函数也可以作为宏进行执行。

isascii 函数对所有的整数值都产生有意义的结果，其余的函数只对 ASCII 字符集中的相应整数值产生定义的结果(isascii 函数值为 true)，对非 ASCII 值产生 EOF(该常量在 stdio.h 中定义)。

当测试字符满足相应条件时返回一个非 0 值，否则返回 0。
〖参见〗tocascii,tolower,toupper。

toascii

〖功能〗该函数返回 c 的 ASCII 值。
〖原型〗int toascii(int c)
〖位置〗ctype.h
〖说明〗toascii 程序将 c 字符中除了低 7 位之外的所有位都设为 0，经过转换后的字符就可以代表 ASCII 字符集中的字符。如果 c 原本就表示 ASCII 字符，则不改变 c 的值。
〖参见〗is…。

tolower

〖功能〗当 c 表示大写字母时，该函数返回 c 的小写形式。
〖原型〗int tolower(int c)
〖位置〗ctype.h
〖说明〗当 c 表示大写字母时，将 c 转换成小写形式，否则 c 的值不变。
〖参见〗is…。

_tolower

〖功能〗当 c 表示大写字母时，该函数返回 c 的小写形式，否则 c 的值不变。
〖原型〗int _tolower(int c)
〖位置〗ctype.h
〖说明〗仅当已知 c 是大写形式时，才能使用_tolower 程序。若 c 不是大写字母，_tolower 没有意义。
〖参见〗is…。

toupper

〖功能〗当 c 表示小写字母时，该函数返回 c 的大写形式。
〖原型〗int toupper(int c)
〖位置〗ctype.h
〖说明〗当 c 表示小写字母时，该函数将 c 转换成大写形式，否则 c 的值保持不变。
〖参见〗is…。

_toupper

〖功能〗当 c 是小写字母时，函数返回 c 的大写形式。
〖原型〗int _toupper(int c)
〖位置〗ctype.h
〖说明〗当 c 表示小写字母时，将 c 转换成大写形式，否则 c 的值保持不变。仅当已知 c 是小写形式时，才能使用_toupper 程序。若 c 不是小写形式，则_toupper 没有意义。
〖参见〗is…。

3.3　数字与字符串转换函数

abs

〖功能〗该函数返回参数 x 的绝对值。
〖原型〗int abs(int n)
double fabs(double x)
long labs(long n)
long double fabsl(long double x)

〖位置〗stdlib.h，math.h

〖说明〗abs、fabs、fabsl 和 labs 函数返回参数 x 的绝对值。下面是这些函数及其返回值的类型。

函数	参数/返回值
abs	整型值
fabs	浮点类型
fabsl	双精度浮点类型
labs	长整型

这些函数没有出错返回。

〖参见〗cabs。

atof

〖功能〗函数将字符串转换成双精度浮点数值，字符串应是能够表示指定类型数值的一系列字符。

〖原型〗double atof(char *string)

〖位置〗stdlib.h 或 math.h

〖说明〗atof 函数能够处理长度不超过 100 的字符串。遇到不能识别为数值的字符时，函数终止。这个非数字字符可能就是表示字符串结束的 null 字符(\0)。

atof 函数的参数 string 指向的字符串格式如下：

[空白字符][符号][｛符号｝][数字][.数字][｛d|D|e|E｝][符号][数字]

空白字符包括空格或 tab 字符，忽略不计；符号可以是+或-号；数字可以是一个或多个十进制数。若在小数点前没有数字，小数点后就必须有数字。十进制数字后可以跟指数，指数由一个表示字符(d，D，e，E)和一个带符号十进制整数构成。

函数返回字符串对应的 double 类型数值。若输入字符串不能够转换成相应类型，则返回 0.0。发生溢出时未定义返回值。

〖参见〗ecvt,fcvt,gcvt。

atoi

〖功能〗函数将字符串转换成整型数值，字符串应是能够表示指定类型数值的一系列字符。

〖原型〗int atoi(char *string)

〖位置〗stdlib.h

〖说明〗遇到不能识别为数值的字符时，函数终止。这个非数字字符可能就是表示字符串结束的 null 字符(\0)。

atoi 函数不能识别小数点或指数。参数 string 指向的字符串格式如下：

[空白字符][符号][符号][数字]

空白字符包括空格或 tab 字符，忽略不计；符号可以是＋或－号；数字可以是一个或

多个十进制数。

函数返回字符串所对应的 int 类型数值。若输入字符串不能够转换成相应类型，则返回 0。发生溢出时未定义返回值。

〖参见〗ecvt,fcvt,gcvt。

atol

〖功能〗函数将字符串转换成长整型数值，字符串应是能够表示指定类型数值的一系列字符。

〖原型〗long atol(char *string)

〖位置〗stdlib.h

〖说明〗不能识别为数值的字符时，函数终止。这个非数字字符可能就是表示字符串结束的 null 字符(\0)。

atol 函数不能识别小数点或指数，参数 string 指向的字符串格式如下：

[空白字符][符号][符号]

空白字符包括空格或 tab 字符，忽略不计；符号可以是＋或－号；数字可以是一个或多个十进制数。

函数返回字符串所对应的 long 类型数值。若输入字符串不能够转换成相应类型，则返回 0L。发生溢出时未定义返回值。

〖参见〗ecvt,fcvt,gcvt。

_atold

〖功能〗函数将字符串转换成长双精度浮点数值，字符串应是能够表示指定类型数值的一系列字符。

〖原型〗long double _atold(char *string)

〖位置〗math.h

〖说明〗函数能够处理长度不超过 100 的字符串。遇到不能识别为数值的字符时，函数终止。这个非数字字符可能就是表示字符串结束的 null(\0)字符。

_atold 函数的参数 string 指向的字符串格式如下：

[空白字符][符号][｛符号｝][数字][.数字][｛d|D|e|E｝][符号][数字]

空白字符包括空格或 tab 字符，忽略不计；符号可以是＋或－号；数字可以是一个或多个十进制数。若在小数点前没有数字，小数点后就必须有数字。十进制数字后可以跟指数，指数由一个表示字符(d，D，e，E)和一个带符号十进制整数构成。

函数返回字符串所对应的 long double 类型数值。若输入字符串不能够转换成相应类型，则返回 0.0。发生溢出时未定义返回值。

〖参见〗ecvt,fcvt,gcvt。

div、ldiv

〖功能〗div 和 ldiv 函数实现将参数 numer 除以参数 denom，计算商和余数。

〖原型〗div_t div(int numer，int denom)

ldiv_t ldiv(long int numer，long int denom)

〖位置〗stdlib.h

〖说明〗div 函数的参数是整型，ldiv 函数的参数是长整型。

商的符号与算术商相同。其绝对值是小于算术商绝对值的最大整数。如果除数为 0，程序提供错误信息并终止运行。

div 函数返回 div_t 结构体类型，该类型由商和余数组成。ldiv 函数返回 ldiv_t 结构体类型。这两种结构体均在 stdlib.h 中定义。

ecvt

〖功能〗ecvt 函数将浮点数转换成字符串，参数 value 为欲转换的浮点数。

〖原型〗char *ecvt(double value，int count，int *dec，int *sign)

〖位置〗stdlib.h

〖说明〗ecvt 和 fcvt 函数保存 value 的 count 位并在最后添加一个空字符(\0)。若 value 中的数位超过 count 个，低位数值被舍入。若 value 的数位不足 count 个，则用 0 补位。

字符串中仅保存数位。函数调用之后，可以从 dec 和 sign 指向的单元获得 value 中的小数点和符号。参数 dec 指向一个整数，它表示小数点位于从字符串开始的第几位上。0 或负数表示小数点在第 1 位的左边。参数 sign 指向一个表示转换数值符号的整数：若该整数值为 0 表示数值为正，否则为负。

ecvt 函数使用单一的静态分配缓存区进行转换。因此，每次调用都会破坏上一次的调用结果。

函数返回一个指向转换的字符串的指针，无错误返回。

〖参见〗atof,atoi,atol。

fcvt

〖功能〗函数将浮点数转换成字符串，参数 value 为欲转换的浮点数。

〖原型〗char *fcvt(double value，int count，int *dec，int *sign)

〖位置〗stdlib.h

〖说明〗fcvt 函数保存 value 的 count 位并在最后添加一个空字符(\0)。若 value 中的数位超过 count 个，低位数值被舍入。若 value 的数位不足 count 个，则用 0 补位。

字符串中仅保存数位。函数调用之后，可以从 dec 和 sign 指向的单元获得 value 中的小数点和符号。参数 dec 指向一个整数，它表示小数点位于从字符串开始的第几位上。0 或负数表示小数点在第 1 位的左边。参数 sign 指向一个表示转换数值符号的整数：若该整数值为 0 表示数值为正，否则为负。

fcvt 函数使用单一的静态分配缓存区进行转换，每次调用都会破坏上一次的调用结果。

函数返回一个指向转换的字符串的指针，无错误返回。

〖参见〗atof,atoi,atol。

gcvt

〖功能〗函数将浮点数转换成字符串，参数 value 为欲转换的浮点数。

〖原型〗char *gcvt(double value，int digits，char *buffer)

〖位置〗stdlib.h

〖说明〗gcvt 函数将一个浮点数值转换成一个字符串，并存储在参数 buffer 指向的空间。缓存区应足够容纳转换的值加上一个终止空字符(\0)，该字符是自动添加的。没有提供溢出处理。

gcvt 函数产生 digits 位有意义的数位。当数值大于等于 0.1 时以小数形式输出，当数值小于 0.1 时以指数形式输出。转换时消除尾部的 0。

函数返回一个指向转换的字符串的指针，无错误返回。

〖参见〗atof,atoi,atol。

itoa

〖功能〗itoa 函数将给定的 value 转换成以空字符结束的字符串，并将结果存储在参数 string 指向的空间。

〖原型〗char *itoa(int value，char *string，int radix)

〖位置〗stdlib.h

〖说明〗itoa 函数的 string 空间最多能够存储 17 个字节。

参数 radix 指定了 value 的基，该值必须在 2~36 之间。若 radix 等于 10 而 value 的值为负，则字符串的第 1 个字符就会存储一个减号(-)。

该函数返回指向 string 的指针，无错误返回。

labs

〖功能〗函数返回参数 x 的绝对值。

〖原型〗long labs(long n)

〖位置〗stdlib.h，math.h

〖说明〗abs、fabs、fabsl 和 labs 函数返回参数 x 的绝对值。下面是这些函数及其返回值的类型。

函数	参数/返回值
abs	整型值。
fabs	浮点类型。
fabsl	双精度浮点类型。
labs	长整型。

这些函数没有出错返回。

〖参见〗cabs。

_lrotl、_lrotr、_rotl、_rotr

〖功能〗循环移位函数按照 shift 指定的位数循环移位。

〖原型〗unsigned long _lrotl(unsigned long value，int shift)

unsigned long _lrotr(unsigned long value，int shift)
unsigned _rotl(unsigned value，int shift)
unsigned _rotr(unsigned value，int shift)

〖位置〗stdlib.h

〖说明〗下面是移位函数、移位方向和 value 的参数类型表。

函数	循环移位	参数类型
_lrotl	左移	long
_lrotr	右移	long
_rotl	左移	unsigned
_rotr	右移	unsigned

移位时从 value 的一端移出的位，将被装配到另一端。

以上函数返回移位后的 value 的值，无错误返回。

ltoa

〖功能〗ltoa 函数将给定的 value 转换成以空字符结束的字符串，并将结果存储在参数 string 指向的空间中。

〖原型〗char *ltoa(long value，char *string，int radix)

〖位置〗stdlib.h

〖说明〗ltoa 函数的 string 空间最多能够存储 33 个字节。

参数 radix 指定了 value 的基，该值必须在 2~36 之间。若 radix 等于 10 而 value 的值为负，则字符串的第 1 个字符就会存储一个减号(-)。

该函数返回指向 string 的指针，无错误返回。

max

〖功能〗max 宏比较两个值，并返回较大的值。

〖原型〗type max(type a，type b)

〖位置〗stdlib.h

〖说明〗数据类型可以是任何数值类型，可以是 signed 或 unsigned。参数的类型应同返回值的类型一致。

该宏返回两个参数的较大者。

min

〖功能〗min 宏比较两个值，并返回较小的值。

〖原型〗type min(type a，type b)

〖位置〗stdlib.h

〖说明〗数据类型可以是任何数值类型，可以是 signed 或 unsigned。参数的类型应同返回值的类型一致。

该宏返回两个参数的较小者。

rand

〖功能〗rand 函数返回一个 0~RAND_MAX 范围内的伪随机数。

〖原型〗int rand(void)

〖位置〗stdlib.h

〖说明〗在调用 srand 函数之前调用 rand 函数能够生成一个伪随机数序列，它和用 1 作 seed 参数的值调用 srand 函数获得的结果相同。

rand 函数返回一个伪随机数无错误返回。

srand

〖功能〗srand 函数设置伪随机整数序列生成器的初始点。

〖原型〗void srand(unsigned seed)

〖位置〗stdlib.h

〖说明〗用 1 作 seed 参数的值，可重新初始化生成器。用其他任何数作 seed 的值会为生成器设置一个随机起点。

srand 函数无错误返回。

strtod、strtol、strtoul、_strtold

〖功能〗strtod 函数将字符串转换成双精度数值，strtol 函数将字符串转换成长整型数值，strtoul 函数将字符串转换成无符号长整型数值。_strtold 函数将字符串转换长双精度浮点数值。

〖原型〗double strtod(char *nptr，char **endptr)

long double _strtold (char *nptr, char **endptr)

long strtol(char *nptr, char **endptr, int base)

unsigned long strtoul(char *nptr, char **endptr, int base)

〖位置〗stdlib.h

〖说明〗输入字符串应是能够识别为指定类型数值的字符序列。若 strtod 函数出现在压缩、大规模程序中，参数 nptr 指向字符串的长度最大是 100 个字符。

这些函数遇到第一个不能识别为数字的字符时就停止继续读取字符串。这个非数字字符可能是字符串结尾的空字符(\0)。对 strtol 函数或 strtoul 函数来说，这个终止字符也可以是第一个大于等于 base 的数字字符。若参数 endptr 不是空字符，它应指向读取字符的结束位置。

strtod 和_strtold 函数的 nptr 参数指向的字符串格式如下：

[空白字符][符号][数位][.数位][｛d | D| e | E｝[符号]数位]

遇到第一个不符合这种格式的字符就停止读入。

strtol 函数的 nptr 参数指向的字符串格式如下：

[空白字符][符号][0][｛x | X｝][数位]

strtoul 函数的 nptr 参数指向的字符串格式如下：

[空白字符][0][｛x | X｝][数位]

如果参数 base 在 2~36 之间，就将其作为转换数值的基。

如果 base 等于 0，就用 nptr 指向的原始字符串来确定数值的基。若字符串的第 1 个字符是 0，第 2 个字符不是 x 或 X，就将字符串转换成八进制数值。若第 1 个字符不是 0 则转换成十进制数值。若第一个字符是 0 且第 2 个字符是 x 或 X，将字符串转换成十六进制整数。

若第 1 个字符是 1~9，将字符串转换成十进制整数。字母 a 到 z(或 A 到 Z)分别表示数值 10~35。注意只有那些数值小于参数 base 的字符是合法的。

strtoul 函数允许‘+’或‘-’号前缀；开头的减号表示对返回值求反。

返回值：strtod 函数返回字符串表示的浮点数值，若发生表示法上溢时，返回+/-HUGE_VAL。当没有执行转换或发生下溢时，返回 0。

strtol 函数返回字符串表示的长整型数值，若发生表示法溢出时，返回 LONG_MAX 或 LONG_MIN。当没有执行转换时返回 0。

进行转换后，stroul 函数返回转换的值。若没有执行转换，函数返回 0。发生溢出时，返回 ULONG_MAX。

_strtold 函数返回字符串表示的浮点数值，若发生表示法上溢时，返回 LHUGE_VAL。当没有执行转换或发生下溢时返回 0。

当发生溢出时，上述 4 个函数都将 errno 设为 ERANGE。

〖参见〗atof,atol。

ultoa

〖功能〗ultoa 函数将给定的 value 转换成以空字符结束的字符串，并将结果存储在参数 string 指向的空间中。

〖原型〗char *ultoa(unsigned long value，char *string，int radix)

〖位置〗stdlib.h

〖说明〗ultoa 函数的 string 空间最多能够存储 17 个字节。

参数 radix 指定了 value 的基，该值必须在 2~36 之间。

该函数返回指向 string 的指针，无错误返回。

3.4 目录结构和信息处理函数

chdir

〖功能〗chdir 函数将当前工作目录改变为由参数 dirname 指定的目录。参数 dirname 必须是已存在的目录名称。

〖原型〗int chdir(char *dirname)

〖位置〗direct.h，errno.h

〖说明〗这个函数可以在任何驱动器中修改当前目录；但不能修改其自身的默认驱动器。例如，A:是默认驱动器，\BIN 是当前工作目录，下面的调用修改 c 驱动器的当前工作

目录：

```
chdir( “c:\\temp” );
```

注意在字符串中必须用两个斜杠才能表示一个斜杠。因为在 C 字符串中斜杠是转义字符，需要进行特殊处理。

上述函数调用后不会有明显的结果。但是当调用_chdrive 函数将默认驱动器改变为 C：后，当前工作目录就成为 C:\TEMP。

在 OS/2 保护模式下，当前工作目录只是针对某个局部进程而言，并非针对整个系统。

当该进程结束后，当前工作目录恢复到初始状态。在 DOS 环境下，程序设置的新目录就会成为新的当前工作目录。

如果成功地修改了当前工作目录，该函数返回 0。返回-1 表示出错，这时将 errno 设为 ENOENT，表示无法找到指定的路径名。

〖参见〗mkdir,rmdir,system。

_chdrive

〖功能〗_chdrive 函数将当前工作驱动器改变为由参数 drive 指定的驱动器。drive 参数用整数来指定新的驱动器(1=A，2=B 等)。

〖原型〗int _chdrive(int drive)

〖位置〗direct.h

〖说明〗这个函数仅修改工作驱动器，chdir 函数修改工作目录。

在 OS/2 保护模式下，当前工作驱动器只是针对某个局部进程而言，并非对整个系统。

当该进程结束后，当前工作驱动器恢复到初始状态。但在 DOS 环境下，程序设置的新驱动器就会成为新的当前驱动器。

如果成功地修改了当前驱动器，该函数返回 0；若出现错误则返回-1。

〖参见〗chdir,_dos_setdrive,_fullpath,_getcwd,_getdrive,_mkdir,rmdir,system。

getcwd

〖功能〗getcwd 函数获取当前工作目录的完整路径名，并存储在 buffer 指向的存储空间中。

〖原型〗char *getcwd(char *buffer，int maxlen)

〖位置〗direct.h，errno.h

〖说明〗整型参数 maxlen 指定了路径名的最大长度。路径名的长度(包括结束空字符)超过 maxlen 就会出错。stdlib.h 中定义了常量_MAX_PATH，它表示可能的最大路径长度。

参数 buffer 可以为 NULL；通过调用 malloc 函数自动分配至少 maxlen 大小的缓存区(如果需要还可以更大)，用来存储路径名。随后调用 free 函数释放这个缓存区，并通过函数返回值(指向分配的缓存区的指针)返回。

该函数返回路径。若返回 NULL 表示出错，并将 errno 设为 ENOMEN 或 ERANGE。

〖参见〗chdir,_getdrive,mkdir,rmdir。

_getdcwd

〖功能〗_getcwd 函数获取指定驱动器的当前工作目录的完整路径名，并存储在 buffer 指向的存储空间中。

〖原型〗char *_getdcwd(int drive，char *buffer，int maxlen)

〖位置〗direct.h，errno.h

〖说明〗_getdcwd 函数的 drive 参数指定了驱动器(0=默认，1=A，2=B 等)。

整型参数 maxlen 指定了路径名的最大长度。路径名的长度(包括结束空字符)超过 maxlen 就会出错。stdlib.h 中定义了常量_MAX_PATH，它表示可能的最大路径长度。

参数 buffer 可以为 NULL;通过调用 malloc 函数自动分配至少 maxlen 大小的缓存区(如果需要还可以更大)，用来存储路径名。随后调用 free 函数释放这个缓存区，并通过函数返回值(指向分配的缓存区的指针。)返回。

该函数返回路径。若返回 NULL 表示出错，并将 errno 设为 ENOMEN 或 ERANGE。

〖参见〗chdir,_getdrive,mkdir,rmdir。

_getdrive

〖功能〗_getdrive 函数返回当前驱动器。

〖原型〗int _getdrive(void)

〖位置〗direct.h

〖说明〗_getdrive 函数返回当前驱动器(1=A，2=B 等)，无错误返回。

〖参见〗_chdrive,_dos_getdrive,_dos_setdrive,_getcwd,_getdcwd。

getenv

〖功能〗getenv 函数在环境变量列表中查找同 varname 相关的条目。

〖原型〗char *getenv(char *varname)

〖位置〗stdlib.h

〖说明〗环境变量定义了程序执行的环境(例如，LIB 环境变量定义了程序链接库的默认查找路径)。因为 getenv 函数是区分大小写的，因此参数 varname 应同环境变量的大小写一致。

环境变量表中的项目不能直接修改。如果必须要修改某个项目，应调用 putenv 函数实现。为了不影响环境变量表但能够修改返回值，应使用 strdup 或 strcpy 函数保存串的副本。

getenv 和 putenv 函数使用全局变量 environ 访问环境变量表。由于 putenv 函数可以修改 environ 的值，这样主函数的参数 envp 就无效了。因此，使用 environ 变量访问环境表更安全。

getenv 函数返回指向包括当前 varname 串值的环境变量表项目的指针。如果给定的变量没有定义则返回 NULL。

〖参见〗putenv。

mkdir

〖功能〗mkdir 函数按照参数 dirname 指定的名称创建一个新目录。每次调用只能创建

一个目录，因此只有 dirname 的最后一部分成为新目录的名称。

〖原型〗int mkdir(char *dirname)

〖位置〗direct.h，errno.h。

〖说明〗mkdir 函数不对路径名分隔符进行任何翻译。DOS 和 OS/2 都承认路径名中的\或/为合法分隔符。

若成功创建了新目录，mkdir 函数返回 0。函数返回值为-1 表示出错，并将 errno 设为 EACCES 或 ENOENT。

〖参见〗chdir,rmdir。

putenv

〖功能〗putenv 函数增加新的环境变量或修改现存的环境变量。

〖原型〗int putenv(char *envstring)

〖位置〗stdlib.h

〖说明〗环境变量定义了程序执行的环境(例如，查找同程序链接的库默认路径)。

参数 envstring 必须是指向如下格式字符串的指针：

```
varname=string。
```

这里的 varname 是要添加或修改的环境变量名，string 是变量的值。如果 varname 已经存在，则用 string 的值对其更新；否则向环境中增加新串。可以指定空串将变量设为空值。

这个函数只影响当前运行程序的环境；不会修改命令级别的环境。当正在运行的程序终止后，环境会恢复到父进程级别(多时情况下，是操作系统级别)的状态。但是由 spawn 或 exec 创建的进程及其子进程会拥有 putenv 添加的任何新项目。

不要释放指向环境变量的指针，因为环境变量会指向释放的空间。如果向 putenv 函数的局部变量传递指针，退出定义变量的函数后，也会发生类似的问题。

putenv 函数只能对运行库的可访问数据结构进行操作，而不能对 DOS 或 OS/2 为进程创建的环境段进行操作。

注意环境变量表中的项目不能直接修改。如果必须要修改某个项目，应调用 putenv 函数实现。为了不影响环境变量表又能够修改返回值，应使用 strdup 或 strcpy 函数保存串的副本。

getenv 和 putenv 函数使用全局变量 environ 访问环境变量表。由于 putenv 函数可以修改 environ 的值，主函数的参数 envp 就无效。因此，使用 environ 变量访问环境信息更安全。

putenv 函数成功后返回 0，若出错则返回-1。

〖参见〗getenv。

rmdir

〖功能〗rmdir 函数删除参数 dirname 指定的目录。

〖原型〗int rmdir(char *dirname)

〖位置〗direct.h，errno.h

〖说明〗要求该目录必须为空，并且不能是当前目录或当前目录的根目录。

rmdir 函数成功删除指定目录后返回 0。出错时返回-1，并将 errno 设为 EACCES 或 ENOENT。

〖参见〗chdir,mkdir。

_searchenv

〖功能〗_searchenv 函数按照指定的域查找目标文件。

〖原型〗void _searchenv(char *filename，char *varname，char *pathname)

〖位置〗stdlib.h

〖说明〗参数 varname 可以是任何指定一组目录路径的环境变量，如 PATH，LIB，INCLUDE 或其他用户自定义变量。最常见的是 PATH，表示在 PATH 中指定的所有路径下查找文件 filename。_searchenv 函数是区分大小写的，因此参数 varname 的大小写形式应同环境变量一致。

程序首先在当前目录下查找文件。如果没有找到，再从环境变量指定的路径下查找。

如果在任何一个目录下找到了目标文件，将最近创建的路径复制到参数 pathname 指向的缓存区中。必须要保证缓存区足够容纳构造路径名。如果没有找到 filename 指定的文件，pathname 指向的缓存区中就仅存储空串。

在 OS/2 下，varname 指定的路径列表中可以包含双引号，表示双引号中间的字符不被 searchenv 识别。例如，searchenv 忽略一对引号中间的分号，将其识别为文件名中的字符(如 OS/2 版本的文件名 1.2)，而不是路径分隔符。下面的例子设置可被_searchenv 识别的包含 3 个目录的情况：

```
PATH+C：\BIN；"D：＼SEMI；COLON\DIN"；\BINP
```

该函数无返回值。

〖参见〗getenv,putenv。

3.5 文件处理函数

access

〖功能〗access 函数对文件进行操作，确认指定文件是否存在，且是否能够以 mode 方式进行访问。

〖原型〗int access(char *pathname，int mode)

〖位置〗io.h，errno.h

〖说明〗下面是 access 函数调用时，参数 mode 可能的取值及其含义。

00	只检查文件是否存在。
02	检查文件是否允许进行写操作。
04	检查文件是否允许进行读操作。
06	检查文件是否允许进行读写操作。

access 函数对目录进行操作，只能够确定指定目录是否存在；在 MS DOS 和 OS/2 下，所有的目录都可以进行读写访问。

如果文件具备指定的模式，则 access 函数返回 0，若指定文件不存在或不能够以指定的模式进行访问，函数返回-1，同时将 errno 设为 EACCES 或 ENOENT。

〖参见〗chmod,fstat,open,stat。

chmod

〖功能〗chmod 函数修改参数 filename 指定的文件的权限设置。权限设置控制文件的读写访问。

〖原型〗int chmod(char *filename，int pmode)

〖位置〗io.h，sys\types.h，sys\stat.h，errno.h

〖说明〗常量表达式 pmode 可以是在 SYS\STAT.H 中定义的常量 S_IWRITE 和 S_IREAD 中的任意一个或两个。pmode 取其他值时均被忽略。当同时给出两个常量时，之间用按位或操作符(|)连接。pmode 参数的含义由 S_IWRITR 或 S_IREAD 表明。

如果没有给出写权限，则文件是只读方式。在 MS DOS 和 OS/2 下，所有文件均是可读的；不可能设置只写权限。因此 S_IWRITE 和 S_IREAD|S_IWRITE 模式是等价的。

若成功地修改了权限设置，chmod 函数返回 0，发生错误时，函数返回-1，并将 errno 设为 ENOENT，表示没有找到指定文件。

〖参见〗access,creat,fstat,open,stat。

chsize

〖功能〗chsize 函数将参数 handle 关联的文件扩展或截断为参数 size 指定的长度。

〖原型〗int chsize(int handle，long size)

〖位置〗io.h，errno.h

〖说明〗文件必须以写模式打开。如果是对文件进行扩展则要添加空字符(\0)。若是截断文件，则截断部分的所有数据均丢失。

在 DOS 下，关闭文件时目录才更新。因此，在一个程序运行期间，查询剩余磁盘空间数量可能会得到不准确的结果。

若成功地改变了文件尺寸，则 chsize 函数返回 0，出错时返回-1，并经 errno 设为 EACCES、EBADF 或 ENOSPC。

〖参见〗close,creat,open。

filelength

〖功能〗filelength 函数以字节为单位，返回同参数 handle 关联的文件的长度。

〖原型〗long filelength(int handle)

〖位置〗io.h，errno.h

〖说明〗当出现错误时，函数返回-1，文件句柄非法则将 errno 设为 EBADF。

〖参见〗chsize,fileno,fstat。

fstat

〖功能〗fstat 函数获取打开文件的信息，并将其存储到 buffer 指向的结构体中。

〖原型〗int fstat(int handle，struct stat *buffer)

〖位置〗sys\types.h，sys\stat.h，errno.h

〖说明〗fstat 函数用文件句柄的方式指定文件。

SYS\STAT.H 中定义了 stat 类型的结构体，其成员如下。

成员	值
st_atime	最后一次修改文件的时间(与 st_mtime 和 st_ctime 相同)。
st_ctime	最后一次修改文件的时间(与 st_atime 和 st_mtime 相同)。
st_dev	文件所在的驱动器号或设备句柄(与 st_rdev 相同)。
st_mode	文件模式信息位掩码。如果 handle 指向一个设备，则设置为 S_IFCHR，若 handle 指向普通文件，则设置为 S_IFREG。根据文件权限模式(S_IFCHR 和其他 SYS\STAT.H 中定义的常量)设置用户读写位。
st_mtime	最后一次修改文件的时间(与 st_atime 和 st_ctime 相同)。
st_nlink	始终为 1
st_rdev	文件所在的驱动器号或设备句柄(与 st_dev 相同)。
st_szie	以字节为单位的文件大小。

在 OS/2 下，st_dev 中的信息无意义。实际上，通常是设为 0，OS/2 不提供从打开文件句柄恢复主驱动器。

注意，若 handle 指向设备，则 stat 结构体中的 size 和 time 成员均无意义。

获得文件状态信息后，fstat 函数返回 0。出错时返回-1，同时将 errno 设为 EBADF，表示文件句柄非法。

〖参见〗access,chmod,filelength。

_fullpath

〖功能〗_fullpath 程序将 pathname 中保存的局部路径转换成完整路径，并存储到 buffer 中。

〖原型〗char *_fullpath(char *buffer，char *pathname，size_t maxlen)

〖位置〗stdlib.h

〖说明〗和_makepath 不同，_fullpath 可以在路径中使用.\和..\。

若完整路径的长度超过 maxlen 的值，就返回 NULL。否则返回 buffer 表示的地址。

若 buffer 为 NULL，_fullpath 将分配一块_MAX_PATH(在 STDLIB.H 中定义)大小的缓存区，同时忽略参数 maxlen。

若 pathname 参数指定的是磁盘驱动器，则将该驱动器的当前目录同路径组合_fullpath 函数不检查驱动器或目录的合法性；它仅建立一个完整的合法路径名。

_fullpath 函数返回指向存储在缓存区中的完整路径的指针。若出现错误，则返回 NULL。

〖参见〗getcwd,_getdcwd,_makepath,_splitpath。

isatty

〖功能〗isatty 函数测试参数 handle 是否和一个字符设备(终端、控制台、打印机或串口)关联。

〖原型〗int isatty(int handle)

〖位置〗io.h

〖说明〗若设备是字符设备，isatty 函数返回非 0 值。

locking

〖功能〗locking 函数对由参数 handle 指定的文件的 nbyte 个字节进行锁定和解锁。

〖原型〗int locking(int handle，int mode，long nbytes)

〖位置〗io.h，sys\locking.h，errno.h

〖说明〗文件中的锁定字节不能被其他进程访问。

锁定和解锁均从当前文件指针的位置开始，对其后 nbyte 个字节，或直到文件结束位置进行操作。对文件结尾之后的字节也可以进行锁定。

mode 参数指定了要执行的锁定操作，其取值为下列常量之一。

LK_LOCK	LK_RLCK
LK_NBLCK	LK_UNLCK
LK_NRBLCK	

可以对一个文件的多个区域进行锁定，但不允许对重叠区域操作。

文件的某个区域解锁后，必须同先前锁定的区域保持一致。locking 函数不能合并相邻区域；如果两个锁定区域相邻，则必须分别单独解锁。

只能对文件的区域暂时锁定，在关闭文件或退出程序前应进行解锁。

locking 函数只能在 OS/2 或 DOS3.0 或更高版本环境下使用；在较早的 DOS 版本下不起作用。注意，在 DOS 3.0 或 3.1 下，父进程锁定的文件，当其任意子进程退出时可能会自动解锁。

使用 locking 函数必须要加载文件共享。OS/2 和某些网络软件环境下，会自动加载文件共享。在 DOS 3.0 或更高版本下，需要使用 DOS 提供的 SHARE 程序启动文件共享。可以查阅网络软件和 DOS 的 SHARE 程序文档，了解更多细节。

locking 函数执行成功就返回 0，失败返回-1，同时将 errno 设为 EACCES、EBADF、EDEADLOCK 或 EINVAL。

〖参见〗creat,open。

_makepath

〖功能〗_makepath 程序创建一个由驱动器符、目录路径、文件名和文件扩展名组成的独立的文件路径。

〖原型〗void _makepath(char *path，char *drive ，char *dir，char *fname，char *end)

〖位置〗stdlib.h

〖说明〗参数 path 应指向一个足够容纳整个路径名的缓存区。常量_MAX_PATH(在 STDLIB.H 中定义)说明系统能够处理的最大的 path 容量。其他参数分别指向存储路径名元素的其他缓存区，如下所示。

缓存区	描述
drive	参数 drive 包括一个表示驱动器的字母(A，B 等等)和一个可选的结尾冒号。若这里没有冒号，_makepath 程序会在组成路径名时自动添加。如果 drive 参数是空字符或空串，构成 path 指向的串时就不会出现驱动器符和冒号。
dir	这个参数中包含不带驱动器符或实际文件名的目录路径。结尾的斜杠可以省略。在这个参数中即可以使用斜杠(\)也可以使用反斜杠(/)。若结尾没有斜杠或反斜杠，_makepath 程序会在组成路径名时自动添加。若 dir 是空字符或空串，构成 path 指向的串时就不会出现斜杠。
fname	参数 fname 包含不带扩展名的基本文件名。如果 fname 是 NULL 或指向空串，构成 path 串时就没有文件名。
ext	参数 ext 包含实际文件的扩展名，前面可以有句点也可以没有。如果 ext 中不包含句点，_makepath 程序在组成完整路径时会自动插入句点。若 ext 是空字符或空串，构成 path 串时就不会插入句点。

上面 4 个字段都没有大小限制。但是，它们组成的路径长度不能超过_MAX_PATH 常量。_MAX_PATH 的界限值远远超过当前任何版本的 DOS 或 OS/2 所能处理的路径长度。

该函数无返回值。

〖参见〗_fullpath,_splitpath。

mktemp

〖功能〗mktemp 函数修改给定的参数 template，并创建一个惟一的文件名。参数 template 的格式为：baseXXXXXX。

〖原型〗char *mktemp(char *template)

〖位置〗io.h

〖说明〗这里 base 是由用户提供那部分新文件名，X 是 mktemp 函数提供的文件名部分的占位符。mktemp 函数保存 base 的值并用跟在五位数值后的数字字母字符替换尾部的 6 个 X。用参数 template 首次调用 mktemp 函数时，这个数字字母字符是 0。五位数值是一个惟一的用来识别调用进程的数。

同一个进程再用同样的参数 template 调用 mktemp 函数时，mktemp 函数就会检查先前返回的文件名是否被用来创建文件了。如果没有相应名称的文件，mktemp 就返回这个文件名。如果前面返回的文件名都存在，mktemp 就会用下一个小写字母替换这个数字字母字符，创建一个新文件名。

例如，若首次返回的名字是 t012345，且这个名字已经被使用了，则下一次返回的文

件名就是 ta12345。创建新文件名时，mktemp 函数按顺序使用 0，小写字母 a~z。

注意，第 1 次调用 mktemp 时就修改了原始的 template 的值。若继续用同一个 template 参数(即最初的参数)调用 mktemp 函数就会出错。

mktemp 函数只是生成惟一的文件名，并不创建或打开文件。

mktemp 函数返回指向修改后的 template 的指针。如果 template 参数不合法或通过给定的 template 无法再建立惟一的文件名，则返回 NULL。

〖参见〗fopen,getpid,open,tempnam,temfile,tmpnam。

remove

〖功能〗remove 函数和 unlink 函数删除参数 path 指定的文件。

〖原型〗int remove(char *path)

〖位置〗io.h，stdio.h，errno.h

〖说明〗UNIX(R)和 XENIX(R)操作系统支持 unlink 函数。在 DOS 和 OS/2 环境下，这两个函数是等价的。

若删除文件成功，函数返回 0；否则返回-1，并将 errno 设为 ENOENT。

〖参见〗close。

rename

〖功能〗rename 函数用参数 newname 重新命名参数 oldname 指定的文件或目录。

〖原型〗int rename(char *oldname，char *newname)

〖位置〗io.h，stdio.h，errno.h

〖说明〗旧文件名必须是存在的文件或目录。新名字则不能是已经存在的文件或目录名。

在 newname 中给出不同的路径名，rename 函数就可将文件从一个目录移动到另一个目录。但是，不能将文件从一个设备移动到另一个设备(例如，从 A 驱动器移动到 B 驱动器。)。只能重新命名目录，不能移动目录。

当 rename 操作成功时，返回 0。出现错误时函数返回非 0 值，同时将 errno 设为 EACCES、ENOENT 或 EXDEV。

〖参见〗creat,fopen,open。

setmode

〖功能〗setmode 函数按照参数 mode 给定的方式设置参数 handle 指定文件的转移模式。mode 的值必须是常量 O_TEXT 或 O_BINARY。

〖原型〗int setmode(int handle，int mode)

〖位置〗fcntl.h，io.herrno.h

〖说明〗尽管 setmode 函数通常用来修改 stdin、stdout、stderr、stdaux 和 stdprn 的默认传输模式，但是它可以对任何文件使用。若将 setmode 函数应用于流文件句柄，应该在对流进行任何输入输出之前调用 setmode 函数。

setmode 函数成功时返回以前的传输模式。出现错误时，函数返回-1 并将 errno 设为

EBADF 或 EINVAL。

〖参见〗creat,fopen,open。

_splitpath

〖功能〗_splitpath 程序将一个全路径名分成四部分。

〖原型〗void _splitpath(char *path，char *drive，char *dir，char *fname，char *ext)

〖位置〗stdlib.h

〖说明〗参数 path 指向存储全路径名的缓存区。每个缓存区的最大尺寸分别由常量 _MAX_DRIVE，_MAX_DIR，_MAX_FNAME 和 MAX_EXT(在 STDLIB.H 中定义)指定。其余参数分别指向存储路径名成员的缓存区。

缓存区	描述
Drive	如果 path 缓存区中指定了驱动器，则存储后跟冒号的该驱动器符。
Dir	存储 path 中指定的子目录路径，结尾有斜杠。path 中即可以出现斜杠也可以出现反斜杠。
fname	存储不带扩展名的基本文件名。
Ext	若在 path 中指定了扩展名的话，存储前面带句点形式的文件扩展名。

若在 path 中没有相应的路径名成员，则该参数指向的区域就存储空串。

该函数没有返回值。

〖参见〗_fullpath,_makepath。

stat

〖功能〗stat 函数获取打开文件的信息，并将其存储到 buffer 指向的结构体中。

〖原型〗int stat(char *pathname，struct stat *buffer)

〖位置〗sys\types.h，sys\stat.h，errno.h

〖说明〗stat 函数用路径指定文件或目录。

SYS\STAT.H 中定义了 stat 类型的结构体，其成员如下。

成员	值
st_atime	最后一次修改文件的时间(与 st_mtime 和 st_ctime 相同)。
st_ctime	最后一次修改文件的时间(与 st_atime 和 st_mtime 相同)。
st_dev	文件所在的驱动器号或设备句柄(与 st_rdev 相同)。
st_mode	文件模式信息位掩码。如果 handle 指向一个设备，则设置为 S_IFCHR，若 handle 指向普通文件，则设置为 S_IFREG。根据文件权限模式(S_IFCHR 和其他 SYS\STAT.H 中定义的常量)设置用户读写位。
st_mtime	最后一次修改文件的时间(与 st_atime 和 st_ctime 相同)。
st_nlink	始终为 1。
st_rdev	文件所在的驱动器号或设备句柄(与 st_dev 相同)。

st_szie　　以字节为单位的文件大小。

在 OS/2 下，st_dev 中的信息无意义。实际上，通常是设为 0。OS/2 不提供从打开文件句柄恢复主驱动器。

注意，若 handle 指向设备，则 stat 结构体中的 size 和 time 成员均无意义。

获得文件状态信息后，stat 函数返回 0。出错时返回-1，同时将将 errno 设为 ENOENT，表示找不到文件名或路径名。

〖参见〗access,chmod,filelength。

umask

〖功能〗umask 函数按照参数 pmode 指定的模式为当前进程设置文件权限掩码。

〖原型〗int umask(int pmode)

〖位置〗io.h，sys\types.h，sys\stat.h

〖说明〗文件权限掩码用于修改由 creat、open 或 sopen 函数创建的新文件的权限设置。如果掩码的某一位为 1，则文件权限值的相应位就被设为 0(禁止)；如果掩码的某一位为 0，则权限值的相应位保持不变。设置好新的文件权限，要关闭文件后才生效。

参数 pmode 是包含常量 S_IWRITE 和 S_IREAD(在 SYS\STAT.H 中定义的)一个常量表达式。若两个常量同时出现，则用按位或(|)运算符连接。

例如，若设置掩码中的写位，则新文件属性为只读。

注意，在 DOS 和 OS/2 环境下，所有的文件都是可读的，不可能设置只写权限。因此用 umask 函数设置读位不会修改文件的模式。

umask 函数返回参数的 pmode 原始值。无错误返回。

〖参见〗chmod,creat,mkdir,open。

unlink

〖功能〗函数删除参数 path 指定的文件。

〖原型〗int unlink(char *path)

〖位置〗io.h，stdio.h，errno.h

〖说明〗remove 函数和 unlink 函数删除参数 path 指定的文件。UNIX(R)和 XENIX(R)操作系统支持 unlink 函数。在 DOS 和 OS/2 环境下，这两个函数是等价的。

若删除文件成功，函数返回 0；否则返回-1，并将 errno 设为 ENOENT。

〖参见〗close。

3.6　图形字体处理函数

_getfontinfo

〖功能〗_getfontinfo 函数获取当前字体特征并将其存储到_fontinfo 类型(在 GRAPH.H 中定义)的结构体中。

〖原型〗shrot _getfontinfo(struct _fontinfo _far *fontbuffer)

〖位置〗graph.h

〖说明〗若字体信息未注册或加载，则函数返回一个负数；否则返回一个非负数。

〖参见〗_getgtextextent,_outgtext,_registerfonts,_setfont,_setgtextvector,_unregisterfonts。

_getgtextextent

〖功能〗_getgtextextent 函数返回用当前_outgtext 输出字符串所使用的字体的像素宽度。

〖原型〗short _getgtextextent(unsigned char _far *text)

〖位置〗graph.h

〖说明〗这个函数对确定使用比例间隔字体的文本大小尤其有用。

该函数返回像素宽度。若字体为注册则返回-1。

〖参见〗_getfontinfo,_outgtext,registerfonts,_setfont,_unregisterfonts。

_outgtext

〖功能〗_outgtext 函数向屏幕输出参数 text 指向的以空字符结尾的字符串。文本以当前颜色，按当前图形位置的字体输出。

〖原型〗void _outgtext(unsigned char _far *text)

〖位置〗graph.h

〖说明〗同标准控制台的 I/O 程序，与 printf 不同，该函数中没有提供格式信息。

_outgtext 输出文本后将更新当前图形位置。

该函数仅能在图形视频模式(MRES4COLOR)下执行。由于这是一个图形函数，因此文本的颜色应由 setcolor 函数，而不是_settextcolor 函数设置。

该函数无返回值。

〖参见〗_moveto,_setcolor,_setfont。

_registerfonts

〖功能〗_registerfonts 函数初始化图形字体系统。

〖原型〗short _registerfonts(unsigned char _far *pathname)

〖位置〗graph.h

〖说明〗在使用任何同字体有关的库函数(_getgtextextent，_outgtext，_setfont，_unregisterfonts。)之前，必须用_registerfonts 函数注册字体文件。

_registerfonts 函数读取指定文件并向内存加载字体头信息。每个字体头信息大约占用 140 个字节。

参数 pathname 是指定文件的路径和合法的.FON 文件。pathname 参数中可以包含标准的 DOS 通配符。

字体函数仅影响字体输出函数_outgtext 输出的文件；其他 C 输出函数都不会受到字体用法的影响。

该函数返回一个正整数，表示成功注册字体数量。若返回负数则表示函数注册失败。可能返回的负数值及其含义如下。

数值	含义
-1	指定的文件或目录不存在。
-2	存在一个或多个.FON 文件不是合法的二进制.FON 文件。
-3	一个或多个.FON 文件损坏。

〖参见〗_getfontinfo,_getgtextextent,_outgtext,_setfont,_unregisterfonts。

_setfont

〖功能〗_setfont 函数从注册的字体集中查找一种具有参数 option 串指定特征的字体。

〖原型〗short _setfont(unsigned char _far *options)

〖位置〗graph.h

〖说明〗若找到这种字体，将其设为当前字体，其后所有调用_outgtext 函数输出文本都使用这种字体。任何时候都只能有一种当前字体。

参数 option 串是一组说明字体特征的字符，_setfont 函数在注册字体列表中查找符合指定特征的字体。

下面是 option 参数指定的字体特征的可能值，option 串中的特征信息是不区分大小写和顺序的。

特征	描述
t'字体名'	铅字字体。
hx	字符高度，x 是像素数。
wy	字符宽度，y 是像素数。
f	仅寻找固定间隔字体(不能和“p”特征字符一起使用)。
p	仅寻找比例间隔字体(不能和“f”特征字符一起使用)。
v	仅寻找矢量字体(不能和“r”特征字符一起使用)。
r	仅寻找光栅映射(位映射)字体(不能和“v”特征字符一起使用)。
b	选择最匹配字体。
nx	选择字体号 x，x 是小于等于函数_registerfonts 的返回值的数。可以使用这个选项“遍历”整个字体集。

除了 nx 选项之外，其他的选项可以混合使用。如果同时使用了互斥的选项(如 f/p 或 r/v)，则_setfont 函数会忽略这些选项。对与 nx 一起使用的不兼容参数不进行错误检测。

这些选项在参数 option 中可以用空格分隔。_setfont 函数忽略其他字符。

参数 option 中的 t 字符(铅字字体)用 t 后跟括在单引号中的字体名指定字体。字体名可以是如下的合法字体。

字体名	描述
Courier	带衬线的固定宽度位映射字体。
Helv	无衬线成比例位映射字体。

Tms Rmn	带衬线成比例位映射字体。
Script	连续线条的斜体格式成比例向量映射字体。
Modern	无衬线成比例向量映射字体。
Roman	带衬线成比例向量映射字体。

参数 option 中的“b”选项可以自动选择同指定的其他特征匹配的“最佳匹配”字体。若指定了“b”特征且至少有一种字体已经注册，则_setfont 函数总能够设置一种字体，并返回 0 表示设置成功。

选择字体时，_setfont 函数遵循如下优先级(从高到低)。

1. 像素高度。
2. 铅字字体。
3. 像素宽度。
4. 固定或成比例字体。

也可以指定字体的像素高度和宽度。若高度或宽度是一个不存在的值，且指定了 b 选项，则_setfont 函数会选择最靠近的字体。其中较小的字体比较大字体的优先级高。若_setfont 以最佳匹配的方式指定 Helv 12，而只能使用 Helv 10 和 Helv 14，那么_setfont 将会选择 Helv 10。

如果选择了不存在的像素高度和宽度，该函数将会自动对像素映射字体应用适当的字体尺寸。若指定了“r”选项或指定铅字字体但没有指定“b”选项时，则不能应用这种自适用方法。

如果指定了 nx 选项，_setfont 将会忽略其他选项字符并且只提供字体号同 x 相符的字体。

注意，字体函数仅仅相信字体输出函数_outgtext 的输出；其他 C 运行时的输出函数都不受字体使用的影响。

_setfont 函数返回 0 表示成功，返回-1 表示错误。如果指定的字体出错且没有指定“b”选项，或字体尚未注册时会发生错误。

〖参见〗_getfontinfo,_getgtextextent,_outgtext,_registerfonts,_unregisterfonts。

_setgtextvector

〖功能〗_setgtextvector 函数设置由向量参数 x，y 指定的当前位置的输出文本的字体。

〖原型〗strcut xycoord _setgtextvector(short x，short y)

〖位置〗graph.h

〖说明〗再调用_outgtext 函数将要用到当前位置。

参数 x 和 y 的值确定了向量，该向量确定了屏幕文本字体的旋转方向。下面是文本旋转选项。

<x，y>	文本位置
<0,0>	不变。
<1,0>	水平文本(默认)。

<0,1>　　逆时针旋转 90 度。

<-1,0>　　旋转 180 度。

<0,-1>　　逆时针旋转 270 度。

如果输入了其他值，则只使用其正负号。例如，将<-3,0>看做<-1,0>。

_setgtextvector 函数用结构体类型 xycoord 返回初始位置向量。若向函数中传递的参数为<0,0>则返回 xycoord 结构体的当前向量值。

〖参见〗_getfontinfo,_getgtextextent,_outgtext,_registerfonts,_setfont,_unregisterfonts。

_unregisterfonts

〖功能〗_unregisterfonts 函数释放以前由_registerfonts 函数分配的空间。

〖原型〗void _unregisterfonts(void)

〖位置〗graph.h

〖说明〗该函数删除所有字体的头信息，并从内存中卸载当前选中的字体数据。

调用_unregisterfonts 函数之后，再想使用_setfont 或_outgtext 函数都是非法的。

该函数无返回值。

〖参见〗_getfonts,_getgtextextent,_outgtext,_registerfonts,_setfont。

3.7　图形原语函数

_arc

〖功能〗_arc 函数用于画弧线。

〖原型〗short _arc(short x1，short y1，short x2，short y2，short x3，short y3，short x4，short y4)

〖位置〗graph.h

〖说明〗_arc 程序使用窗口坐标系。圆弧的圆心就是由点<<x1>,<y1>>和<<x2>,<y2>>限定的矩形的中心。圆弧以其圆心和<<x3>,<y3>>连线的延长线同矩形的交点为起点，沿逆时针方向到其圆心和<<x4>,<y4>>连线的延长线相交为止。

这个程序以当前颜色画弧。由于圆弧没有产生封闭区域，因此无需填充。

如果成功画弧，函数返回非 0 值，否则返回 0。

〖参见〗_ellipse,_getarcinfo,_lineto,_pie,_rectangle,_setcolor。

_arc_wxy

〖功能〗_arc_wxy 函数用于画弧线。

〖原型〗short _arc_wxy(struct _wxycoord _far *pwxy1，struct _wxycoord _far *pwxy2，struct _wxycoord _far *pwxy3，struct _wxycoord _far *pwxy4)

〖位置〗graph.h

〖说明〗_arc_wxy 程序(宏)除了使用真实坐标系外，在其他方面同_arc 程序一样。

这个程序都以当前颜色画弧。由于圆弧没有产生封闭区域，因此无需填充。

如果成功画弧，函数返回非 0 值，否则返回 0。

〖参见〗_ellipse,_getarcinfo,_lineto,_pie,_rectangle,_setcolor。

_clearscreen

〖功能〗_clearscreen 函数清除目标区域的所有显示内容，用当前背景色填充。

〖原型〗void _clearscreen(short area)

〖位置〗graph.h

〖说明〗参数 area 可以是下面的常量之一(在 GRAPH.H 中定义)：

_GCLEARSCREEN，_GVIEWPORT，_GWINDOW。

该函数无返回值。

〖参见〗_getbkcolor,_setbkcolor。

_displaycursor

〖功能〗该函数返回初始 toggle 值。

〖原型〗short _displaycursor(short toggle)

〖位置〗graph.h

〖说明〗一旦进入任何图形程序，屏幕光标都会被关闭。但程序退出图形程序时，用 _displaycursor 函数确定是否重新打开光标。toggle 的取值为 _GCURSORON 或 _GCURSOROEF。如果参数 toggle 的值是_GCURSORON，则退出图形程序后打开光标，若 toggle 的值是_GCURSOROEF，则光标仍然保持关闭。

该函数返回初始 toggle 值。无错误返回。

〖参见〗_gettextcursor,_settextcursor。

_ellipse、_ellipse_w、_ellipse_wxy

〖功能〗_ellipse 函数用来画图形椭圆或圆。图形边界为当前颜色。

〖原型〗

short _ellipse(short control，short x1，short y1，short x2，short y2)

short _ellipse_w(short control，double wx1，double wy1，double wx2，double wy2)

short _ellipse_wxy(short control，struct _wxycoord _far *pwxy1，struct _wxycoord _far *pwxy2)

〖位置〗graph.h

〖说明〗_ellipse 函数中，椭圆的圆心是由视窗坐标系的点<<x1>,<y1>>和<<x2>,<y2>>限定的矩形的中心。

_ellipse_w 函数(作为宏执行)中，椭圆的圆心是由窗口坐标系的点<<wx1>,<wy1>>和<<wx2>,<wy2>>限定的矩形的中心。

_ellipse_wxy 函数(作为宏执行)中，椭圆的圆心是由窗口坐标对<<pwxy1>和<pwxy2>>限定的矩形的中心。

如果边界矩形参数定义在一个点或一条水平或垂直的直线上，则无法画出图形。

参数 control 可以是下列常量之一：_GBORDER，_GFILLINTERIOR。

参数 control 若为_GFILLINTERIOR，则等价于该函数之后以椭圆圆心为起点，使用当前颜色(由_setcolor 设置)为边界色，调用 floodfill 函数。

若成功画出椭圆，_ellipse 函数返回非 0 值，否则返回 0。

〖参见〗_arc,_floodfill,_lineto,_pie,_rectangle,_setcolor,_setfillmask。

_floodfill、_floodfill_w

〖功能〗_floodfill 函数使用当前颜色和填充 mask 来填充显示的某个区域。

〖原型〗short _floodfill(short x，short y，short boundary)

short _floodfill_w(double wx，double wy，short boundary)

〖位置〗graph.h

〖说明〗_floodfill 函数从视窗坐标点<<x>,<y>>位置开始填充。

_floodfill_w 函数从窗口坐标点<<wx>,<wy>>位置开始填充。

如果指定的点在图形之内，则填充图形内部区域；如果指定点在图形之外，则填充背景色。指定点必须在图像内部或外部，不能位于图形边界上。填充从各个方向进行，遇到颜色 boundary 停止。

_floodfill_w 程序是宏。

如果能够成功填充，_floodfill 函数返回一个非 0 值，若不能完成填充返回 0，当填充开始点位于 boundary 颜色上，或超出限制时会发生填充失败。

〖参见〗_ellipse,_getcolor,_getfillmask,_pie,_setcliprgn,_setcolor,_setfillmask。

_getactivepage

〖功能〗_getactivepage 函数返回当前活动页面的数量。

〖原型〗short _getactivepage(void)

〖位置〗graph.h

〖说明〗所有的硬件组合都至少支持一个页面(页号为 0)。在 OS/2 中，近 0 页面是合法的。

该函数返回当前活动页面的数量。

〖参见〗_getvisualpage,_setactivepage,_setvisualpage。

_getarcinfo

〖功能〗_getarcinfo 函数确定最近画的圆弧或饼图的结束点的视窗坐标。

〖原型〗short _getarcinfo(struct xycoord _far *srart，struct xycoord _far *end，struct xycoord _far *fillpoint)

〖位置〗graph.h

〖说明〗若成功，则_getarcinfo 函数更新结构体参数 start 和 end 使其存储最近调用_arc 或_pie 函数画的圆弧或饼图的结束点。

另外，参数 fillpoint 确定了饼图的填充起点。这个信息对同边界的不同的颜色填充饼图非常有用。可以在调用_getarcinfo 函数之后，调用_setcolor 函数改变颜色。然后用 fillpoint

中的坐标和当前颜色作为_floodfill 过程的参数。

如果该函数成功，返回一个非 0 值。如果由于最近一次屏幕被清除或选择了新的图形模式或窗口，而不存在_arc 函数和_pie 函数的成功调用，该函数返回 0。

〖参见〗_arc,_floodfill,_getvideoconfig,_grstatus。

_getbkcolor

〖功能〗_getbkcolor 函数返回当前的背景色，默认背景色为 0。

〖原型〗long _getbkcolor(void)

〖位置〗graph.h

〖说明〗在彩色文本模式(如_TEXTC80)下，_getgkcolor 返回一个颜色索引。在彩色图形模式(如_ERESCOLOR)下，_getbkcolor 返回颜色值(同_remappalette 函数中使用的一样)。参见：图形模式颜色。

多数函数中，只要参数是长整型，就表示颜色值；参数是短整型，表示颜色索引属性。这中间有两个例外，是_setbkcolor 函数和_getbkcolor 函数。

该函数返回当前背景色值。无错误返回。

〖参见〗_getcolor,_gettextcolor,_remappalette,_setbkcolor。

_getcolor

〖功能〗_getcolor 函数返回当前图形颜色索引，默认值是当前调色板中的最高合法值。

〖原型〗short _getcolor(void)

〖位置〗graph.h

〖参见〗_getbkcolor,_gettextcolor,_setcolor。

_getcurrentposition

〖功能〗_getcurrentposition 函数返回当前图形输出位置的坐标。

〖原型〗struct xycoord _getcurrentposition(void)

〖位置〗graph.h

〖说明〗_getcurrentposition 函数返回坐标的形式是 xycoord 类型结构体(在 GRAPH.H 中定义)。

可以用函数_lineto、_moveto 和_outgtext 改变当前位置。

用函数_setvideomode、_setvideomoderows 或_setviewport 设置的默认位置是视窗的中心。

函数仅影响图形输出。不会影响当前文本位置开始的文本输出。(详细信息参看函数_settextposition)。

该函数返回当前图形输出位置的坐标，没有错误返回。

〖参见〗_grstatus,_lineto,_moveto,_outgtext。

_getcurrentposition_w

〖功能〗_getcurrentposition 函数返回当前图形输出位置的坐标。

〖原型〗struct wxycoord _getcurrentposition_w(void)

〖位置〗graph.h

〖说明〗_getcurrentposition_w 函数返回坐标的形式是_wxycoord 类型结构体(在GRAPH.H 中定义)。

可以用函数_lineto、_moveto 和_outgtext 改变当前位置。

用函数_setvideomode、_setvideomoderows 或_setviewport 设置的默认位置是视窗的中心。

函数仅影响图形输出。不会影响当前文本位置开始的文本输出(详细信息参见函数_settextposition)。

该函数返回当前图形输出位置的坐标，没有错误返回。

〖参见〗_grstatus,_lineto,_moveto,_outgtext。

_getfillmask

〖功能〗_getfillmask 函数返回当前填充掩码。

〖原型〗unsigned char _far *_getfillmask(unsigned char _far *mask)

〖位置〗graph.h

〖说明〗这个掩码是一个 8 位乘 8 位的数组，每个位代表一个像素。当某位为 1 时，其相应像素设为当前色，为 0 的像素点颜色不变。

在整个填充区域中重复进行屏蔽。如果没有设置填充掩码，或参数 maks 为 NULL，就会使用当前色进行固定 unpatterned 填充。

如果没有设置掩码，则函数返回 NULL。

〖参见〗_ellipse,_floodfill,_pie,_polygon,_rectangle,_setfillmask。

_getimage

〖功能〗_getimage 函数将指定边界的矩形区域的屏幕图像存储到 image 指向的缓存区中。

〖原型〗void _getimage(short x1，short y1，short x2，short y2，char _huge *image)

〖位置〗graph.h

〖说明〗_getimage 函数用视窗坐标点<<x1>,<y1>>和 <<x2>,<y2>>定义边界矩形。

缓存区应足够容纳图像。可以通过调用适当的_imagesize 函数或使用描述_imagesize 函数的公式自定义缓存区尺寸。

该函数无返回值。可通过调用_grstatus 查看是否成功。

〖参见〗_imagesize,_putimage。

_getimage_w

〖功能〗_getimage 函数将指定边界的矩形区域的屏幕图像存储到 image 指向的缓存区中。

〖原型〗void _getimage_w(double wx1，double wy1，double wx2，double wy2，char _huge *image)

〖位置〗graph.h

〖说明〗_getimage_w 程序(宏)用窗口坐标点<<wx1>,<wy1>>和 <<wx2>,<wy2>>定义边界矩形。

缓存区应足够容纳图像。可以通过调用适当的_imagesize 函数或使用描述_imagesize 函数的公式自己定义缓存区尺寸。

该函数无返回值，可通过调用_grstatus 查看是否成功。

〖参见〗_imagesize,_putimage。

_getimage_wxy

〖功能〗_getimage 函数将指定边界的矩形区域的屏幕图像存储到 image 指向的缓存区中。

〖原型〗void _getimage_wxy(struct_wxycoord _far *pwxy1，struct_wxycoord _far *pwxy2，char _huge *image)

〖位置〗graph.h

〖说明〗_getimage_wxy 程序(宏)用窗口坐标对<<pwxy1>,<pwxy2>>定义边界矩形。

缓存区应足够容纳图像。可以通过调用适当的_imagesize 函数或使用描述_imagesize 函数的公式自己定义缓存区尺寸。

该函数无返回值，可通过调用_grstatus 查看是否成功。

〖参见〗_imagesize,_putimage。

_getlinestyle

〖功能〗_getlinestyle 函数返回当前线型掩码。掩码是一个 16 位数组；每位代表所画直线上的一个像素。

〖原型〗unsigned short _getlinestyle(void)

〖位置〗graph.h

〖说明〗有些图形程序(如：_lineto，_polygon 和_rectangle)向屏幕输出直线，直线的类型由当前线型掩码控制。

如果某位为 1，将相应像素点设为直线的颜色(当前色)。若某位为 0，表示相应位的像素不改变。掩码值在直线区间内重复出现，掩码的默认值是 0xFFFF(实线)。

如果没有设置掩码，_getlinestyle 函数返回默认掩码。

〖参见〗_lineto,_pie,_rectangle,_setlinestyle,_setwritemode。

_getphyscoord

〖功能〗_getphyscoord 函数将视窗坐标<<x>,<y>>转换成物理坐标，以 xycoord 类型结构体(在 GRAPH.H 中定义)返回物理坐标。

〖原型〗struct xycoord _getphyscoord(short x，short y)

〖位置〗graph.h

〖说明〗详细内容可参看图形坐标系统。

〖参见〗_getviewcoord,_grstatus,_setvieworg,_setviewport。

_getpixel

〖功能〗_getpixel 类函数返回指定位置的像素值(颜色索引)。

〖原型〗short _getpixel(short x，short y)

〖位置〗graph.h

〖说明〗_getpixel 函数的参数是视窗坐标(<<x>,<y>>)。

_getpixel 和_getpixel_w 函数的像素值范围和颜色转换都分别取决于当前的视频模式和调色板。

如果函数成功，则返回颜色索引。否则(如，给定点越界，或程序处于文本模式)，函数返回-1。

〖参见〗_remapallpalette,_remappalette,_selectpalette,_setpixel,_setvideomode。

_getpixel_w

〖功能〗_getpixel 类函数返回指定位置的像素值(颜色索引)。

〖原型〗 short _getpixel_w(double wx，double wy)

〖位置〗graph.h

〖说明〗_getpixel_w 函数使用窗口坐标(<<wx>,<wy>>)。

_getpixel 和_getpixel_w 函数的像素值范围和颜色转换都分别取决于当前的视频模式和调色板。

_getpixel_w 程序是作为宏而执行的。

如果函数成功，则返回颜色索引。否则(如，给定点越界，或程序处于文本模式)，函数返回-1。

〖参见〗_remapallpalette,_remappalette,_selectpalette,_setpixel,_setvideomode。

_gettextcolor

〖功能〗_gettextcolor 函数返回当前文本颜色的颜色索引。

〖原型〗short _gettextcolor(void)

〖位置〗graph.h

〖说明〗文本颜色只能由_settextcolor 函数设置，并且只影响_outtext 和_outmem 函数的文本输出。可以使用_setcolor 函数和_outtext 函数设置文本输出的颜色。

在图形模式中，默认值是当前调色板的最高的合法颜色。在文本模式中，默认值是 7。

〖参见〗_remappalette，_selectpalette，_setcolor，_settextcolor。

_gettextposition

〖功能〗_gettextposition 函数以 rccoord 类型结构体(在 GRAPH.H 中定义)返回当前文本位置。

〖原型〗struct rccoord _gettextposition(void)

〖位置〗graph.h

〖说明〗将文本窗口的左上角定义为文本坐标位置<1,1>。

_outtext 函数和_outmem 函数输出的文本从当前文本位置开始。字体文本不受当前文本位置的影响。字体文本从当前图形输出位置开始，这是独立的位置。

〖参见〗_getcurrentposition,_moveto,_outmem,_outtext,_settextposition,_settextwindow,_wrapon。

_gettextwindow

〖功能〗_gettextwindow 函数查找当前文本窗口的边界。

〖原型〗void _gettextwindow(short _far *r1，short _far *c1，short _far *r2，short _far *c2)

〖位置〗graph.h

〖说明〗文本窗口是屏幕上_outtext 和_outmem 函数输出文本的区域，如果没有用_settextwindow 函数重新定义该窗口，则为全屏幕。

_settextwindow 函数定义的屏幕不会影响_outgtext 函数的输出。_outgtext 函数显示的文本被限制在当前视窗。

〖参见〗_gettextposition,_outmem,_outtext,_scrolltextwindow,_settextposition,_settextwindow。

_getvideoconfig

〖功能〗_getvideoconfig 函数以 videoconfig 类型结构体(在 GRAPH.H 中定义)形式返回当前图形环境的设置。

〖原型〗struct _videoconfig _far *getvideoconfig(struct videoconfig _far *config)

〖位置〗graph.h

〖说明〗函数的返回值反映了当前活动监视器和当前指定的视频模式。

videoconfig 结构体包括以下成员。

成员名	内容
numxpixels	x 轴上的像素数量。
numypixels	y 轴上的像素数量。
numtextcols	可用文本列数。
numtextrows	可用文本行数。
numcolors	实际可用颜色数量。
bitsperpixel	每个像素的位数。
numvideopages	可用视频页数。
mode	当前视频模式。
adapter	活动显示适配器。
monitor	活动显示监视器。
memory	以 kb 为单位的适配器视频内存大小。

videoconfig 结构体的 adapter 成员的值可取下面的常量。注意对任何类型的适配器(_CGA，_EGA 或_VGA)，其相关的 Olivetti(R)适配器(_OCGA，_OEGA 或_OVGA)代表了其图形容量的扩集。

常量	含义
_MDPA	单色显示打印适配器。
_CGA	彩色图形适配器。
_OCGA	Olivetti(AT&T)彩色图形适配器。
_EGA	增强图形适配器。
_OEGA	Olivetti(AT&T)加强图形适配器。
_VGA	视频图形阵列。
_OVGA	Olivetti(AT&T)视频图形阵列。
_MCGA	多色图形阵列。
_HGC	大力神(Hercules)图形卡。

videoconfig 结构体的 monitor 成员的值可取下面的常量。

常量	含义
_MONO	单色监视器。
_COLOR	彩色(或仿彩色加强)监视器。
_ENHCOLOR	增强彩色监视器。
_ANALOGMONO	模拟监视器(仅单色)。
_ANALOGCOLOR	模拟监视器(仅彩色)。
_ANALOG	模拟监视器(可彩色也可单色模式)。

包含单色在内的每种文本模式下，_getvideoconfig 函数返回可获得颜色的值 32。32 表示_settextcolor 函数能够接受的取值范围(0~31)，包括 16 种标准色(0~15)和 16 种闪烁色(16~31)，给标准色值加 16 即是相应的闪烁色。

单色文本模式只有少数惟一的显示特性，因此有些颜色值就冗余了。但是，由于可以通过同样的方式获得闪烁色，单色文本模式也同其他文本模式有同样的颜色值范围(0~31)。

_getvideoconfig 函数无错误返回。

〖参见〗_setvideomode,_setvideomoderows。

_getviewcoord

〖功能〗_getviewcoord 函数将一种坐标系下的点坐标<<x>,<y>>转换成视窗坐标，并以 xycoord 类型结构体(在 GRAPH.H 中定义)返回坐标。

〖原型〗struct xycoord _getviewcoord(short x，short y)

〖位置〗graph.h

〖说明〗不同的_getviewcoord 函数以下面的方式分别进行转换。

函数	转换
_getviewcoord	将物理坐标<<x>,<y>>转换成视窗坐标。
_getviewcoord_w	将窗口坐标<<wx,<wy>>转换成视窗坐标。

_getviewcoord_wxy	将窗口坐标结构体<pwxy1>转换成视窗坐标。

注意，在 Microsoft C 5.1 中，_getviewcoord 函数是由_getlogcoord 函数调用的。
该函数无错误返回。
〖参见〗_getphyscoord,_getwindowcoord。

_getviewcoord_w

〖功能〗_getviewcoord 函数将一种坐标系下的点坐标<<x>,<y>>转换成视窗坐标，并以 xycoord 类型结构体(在 GRAPH.H 中定义)返回坐标。
〖原型〗struct xycoord _getviewcoord_w(double wx，double wy)
〖位置〗graph.h
〖说明〗_getviewcoord_w 函数和_getviewcoord_wxy 程序作为宏执行。
不同的_getviewcoord 函数以下面的方式分别进行转换。

函数	转换
_getviewcoord	将物理坐标<<x>,<y>>转换成视窗坐标。
_getviewcoord_w	将窗口坐标<<wx,<wy>>转换成视窗坐标。
_getviewcoord_wxy	将窗口坐标结构体<pwxy1>转换成视窗坐标。

注意，在 Microsoft C 5.1 中，_getviewcoord 函数是由_getlogcoord 函数调用的。
该函数无错误返回。
〖参见〗_getphyscoord,_getwindowcoord。

_getviewcoord_wxy

〖功能〗_getviewcoord 函数将一种坐标系下的点坐标<<x>,<y>>转换成视窗坐标，并以 xycoord 类型结构体(在 GRAPH.H 中定义)返回坐标。
〖原型〗struct xycoord _getviewcoord_wxy(struct _wxycoord _far *pwxy1)
〖位置〗graph.h
〖说明〗_getviewcoord_w 函数和_getviewcoord_wxy 程序作为宏执行。
不同的_getviewcoord 函数以下面的方式分别进行转换。

函数	转换
_getviewcoord	将物理坐标<<x>,<y>>转换成视窗坐标
_getviewcoord_w	将窗口坐标<<wx,<wy>>转换成视窗坐标
_getviewcoord_wxy	将窗口坐标结构体<pwxy1>转换成视窗坐标。

注意，在 Microsoft C 5.1 中，_getviewcoord 函数是由_getlogcoord 函数调用的。
该函数无错误返回。
〖参见〗_getphyscoord,_getwindowcoord。

_getvisualpage

〖功能〗_getvisualpage 函数返回当前可视页面的数量。

〖原型〗short _getvisualpage(void)

〖位置〗graph.h

〖说明〗所有的硬件组件至少支持一页(0 页面)，在 OS/2 中，只有 0 页面是可用的。

〖参见〗_getactivepage,_setactivepage,_setvisualpage。

_getwindowcoord

〖功能〗_getwindowcoord 函数将视窗坐标<<x>,<y>>抓获成窗口坐标，并以_wxycoord 类型结构体(在 GRAPH.H 中定义)返回坐标值。

〖原型〗struct _wxycoord _getwindowcoord(short x，short y)

〖位置〗graph.h

〖说明〗该函数无出错返回。

〖参见〗_getphyscoord,_getviewcoord,_setwindow。

_getwritemode

〖功能〗_getwritemode 函数返回由_lineto、_polygon 和_rectangle 函数画线时的当前逻辑写模式。

〖原型〗short _getwritemode(void)

〖位置〗graph.h

〖说明〗逻辑写模式的默认值是_GPSET，在这种模式下，使用当前图形颜色画线。其他可能的值为：_GAND，_GOR，_GPRESET 和_GXOR。

_getwritemode 函数返回当前逻辑写模式，若当前不是图形模式则返回-1。

〖参见〗_grstatus,_lineto,_putimage,_rectangle,_setcolor,_setlinestyle,_setwritemode。

_grstatus

〖功能〗_grstatus 函数返回最近使用的图形函数的状态。

〖原型〗short _grstatus(void)

〖位置〗graph.h

〖说明〗可以在调用图形程序之后立即使用_grstatus 函数，确认是否出现了错误或警告。若返回值小于 0 则说明出现错误，返回值大于 0 说明出现警告。

下面是在 GRAPH.H 头文件中定义的_grstatus 函数使用的常量。

值	常量	含义
0	_GROK	成功。
-1	_GRERROR	图形错误。
-2	_GRMODENOTSUPPORTED	不支持要求的视频模式。
-3	_GRNOTINPROPERMODE	程序只能在特定视频模式下运行。

-4	_GRINVALIDPARAMETER	存在非法参数。
-5	_GRFONTFILENOTFOUND	找不到匹配的字体文件。
-6	_GRINVALIDFONTFILE	存在非法字体文件。
-7	_GRCORRUPTEDFONTFILE	存在不一致字体文件。
-8	_GRINSUFFICIENTMEMORY	内存不足，无法为_floodfill 操作分配缓存区。
-9	_GRINVALIDIMAGEBUFFER	图像缓冲取决不一致。
1	_GRNOOUTPUT	无操作。
2	_FRCLIPPED	输出受到视口限制。
3	_GRPARAMETERALTERED	将输入的参数转换到某范围内，或将交换两个参数的位置以满足正确的参数顺序。

调用图形函数后，使用一条 if 语句比较_grstatus 函数的返回值和_GROK，例如：

```
if(_grstatus<_GROK )
/*处理图形错*/;
```

下面列出的函数不能给出出错信息，这些函数将_grstatus 函数的返回值均设为_GROK。

_displaycursor	_gettextposition	_outmem	_getactivepage
_gettextwindow	_outtext	_getgtextvector	_getvideoconfig
_unregisterfonts	_gettextcolor	_getvisualpage	_wrapon

下面是影响 grstatus 函数值的图形函数，同时也给出了图形函数设置的出错或警告信息。除了这些错误代码之外，下面的所有函数都能够产生_GRERROR 错误代码。

函数	_grstatus 可能出现的错误代码	_grstatus 可能出现的警告代码
_arc functions	_GRNOTINPROPERMODE, _GRINVALIDPARAMETER,	_GRNOOUTPUT, _GRCLIPPED
_clearscreen	_GRNOTINPROPERMODE, _GRINVALIDPARAMETER	
_ellipse functions	_GRNOTINPROPERMODE, _GRINVALIDPARAMETER, _GRINSUFFICIENTMEMORY	_GRNOOUTPUT, _GRCLIPPED
_getarcinfo	_GRERROR, _GRNOTINPROPERMODE	

_imagesize、_imagesize_w、_imagesize_wxy

〖功能〗_imagesize 函数返回存储边界矩形定义的图像所需的字节数，函数调用时给出了指定该矩形的坐标。

〖原型〗

long _imagesize(short x1，short y1，short x2，short y2)

long _imagesize_w(double wx1，double wy1，double wx2，double wy2)

long _imagesize_wxy(struct _wxycoord _far *pwxy1，struct _wxycoord _far *pwxy2)

〖位置〗graph.h

〖说明〗_imagesize 函数用视图坐标点<<x1>,<y1>>和<<x2>,<y2>>定义边界矩形。可用下面的公式计算矩形的大小。

```
xwid=abs(x1－x2)＋1；
ywid=abs(y1－y2)＋1；
size=4＋((long)((xwid*bits-per-pixel＋7)/8)*(long)ywid)；
```

bit-per-pixel 的值可通过调用_getvidoconfig 函数以 videocofing 结构体的 bitsperpixel 乘以返回。

_imagesize_w 函数用窗口坐标点<<wx1>,<wy1>>和<<wx2>,<wy2>>定义边界矩形。

_imagesize_wxy 函数用窗口坐标对<<pwxy1>>和<<pwxy2>>>定义边界矩形。

_imagesize_w 和_imagesize_wxy 函数都可作为宏执行。

这些函数返回以字节为单位的图像的存储大小。无错误返回。

〖参见〗_getimage,_putimage。

_lineto、_lineto_w

〖功能〗_lineto 函数画一条从当前位置到目标点之间的直线。

〖原型〗short _lineto(short x，short y)

short _lineto_w(double wx，double wy)

〖位置〗graph.h

〖说明〗_lineto 函数的目标点以视窗坐标点参数<<x>,<y>>的形式给出。_lineto_w 函数的目标点以窗口坐标点参数<<wx>,<wy>>的形式给出。

_lineto_w 函数可作为宏执行。

函数使用当前颜色、逻辑写模式和线型画线。如果没有出错，_lineto 函数将当前位置设置为视窗点<<x>,<y>>；_lineto_w 函数将当前位置设置为窗口点<<wx>,<wy>>；

如果调用_lineto 函数和_floodfill 函数填充一个封闭图形一起使用时，必须用实线线型画图。

如果成功地画好直线，则_lineto 和_lineto_w 函数返回非 0 值；否则返回 0。

〖参见〗_getcurrentposition,_moveto,_setlinestyle。

_moveto、_moveto_w

〖功能〗_moveto 函数将当前位置移动到指定点。

〖原型〗struct xycoord _moveto(short x，short y)

struct _wxycoord _moveto_w(double wx，double wy)

〖位置〗graph.h

〖说明〗_moveto 函数使用视窗坐标点<<x>,<y>>为当前位置，_moveto_w 函数使用视窗坐标点<<wx>,<wy>>为当前位置。注意，这里不画任何图形。

这些函数返回初始位置的坐标，_moveto 函数的返回值是 xycoord 类型结构体(在 GRAPH.H 中定义)，_moveto_w 函数的返回值是_wxycoord 类型结构体(在 GRAPH.H 中定义)。

〖参见〗_lineto。

_outmem

〖功能〗_outmem 函数显示 text 指向的字符串。length 指定字符串的长度。

〖原型〗void _outmem(unsigned char _far *text，short length)

〖位置〗graph.h

〖说明〗同_outtext 函数不同，_outmem 函数逐字输出所有字符，包括 0x11，0x13 和 0x00 等的图形字符。该函数不提供输出格式。从当前文本位置开始使用当前的文本颜色。若要用指定字体输出文本，必须使用_outgtext 函数。

该函数无返回值。

〖参见〗_outtext,_settextcolor,_settextposition,_settextwindow。

_outtext

〖功能〗_outtext 函数输出指针 text 指向的以空字符结尾的字符串。

〖位置〗graph.h

〖原型〗void _outtext(unsigned char _far *text)

〖说明〗同 printf 这样的标准控制台 I/O 程序不同，这个函数中没有格式说明符。该函数可以在任何屏幕模式下工作。

文本输出从当前文本位置开始。

若欲以指定字体输出文本，必须使用_outgtext 函数。

该函数无返回值。

〖参见〗_outmem,_settextcolor,_settextposition,_settextwindow。

_pie、_pie_wxy

〖功能〗_pie 函数用椭圆弧画一个饼图的边界，这个椭圆的中心和两个端点用直线连接。

〖原型〗

short _pie(short control，short x1，short y2，short x2，short y2，short x3，short y3，short x，short y4)

short _pie_wxy(short control，struct _wxycoord _far *pwxy1，struct _wxycoord _far *pwxy2，struct _wxycoord _far *pwxy3，struct _wxycoord _far *pwxy4)

〖说明〗_pie 函数使用视窗坐标系。弧线的中心就是由视窗坐标点<<x1>,<y1>>和<<x2>,<y2>>确定的矩形区域的中心。弧线从<<x3>,<y3>>点与其交点开始，到<<x4>,<y4>>点与其交点为止。

_pie_wxy 程序(作为宏执行)使用窗口坐标系。弧线的中心就是由窗口坐标对

<pwxy1>,<pwxy2>>确定的矩形区域的中心。弧线从<pwxy3>点与其交点开始，到<pwxy4>点与其交点为止。

函数用当前颜色，按照逆时针方向画饼图边界。参数 control 是下面两个常量之一：_GBORDER 或_GFILLINTERIOR。

_GFILLINTERIOR 控制选项等价于调用函数之后，再以椭圆中心作为起点和当前颜色(由_setcolor 设置的)作为边界颜色调用_floodfill 函数。

该函数若成功画出饼图，则返回非 0 值；否则返回 0。

〖参见〗_arc,_ellipse,_floodfill,_getarcinfo,_getcolor,_lineto,_rectangle,_setcolor,_setfillmask。

_polygon、_polygon_w、_polygon_wxy

〖功能〗_polygon 函数用当前颜色和线型画多边形。

〖原型〗

short _polygon(short control，struct xycoord _far *points，short numpoints)

short _polygon_w(short control，double _far *points，short numpoints)

short _polygon_wxy(short control，struct _wxycoord _far *points，short numpoints)

〖位置〗graph.h

〖说明〗_polygon 函数使用视窗坐标系(用 xycoord 类型结构体表示)。_polygon_w 函数和_polygon_wxy 函数使用真实窗口坐标系(用_wxycoord 类型结构体表示)。

_polygon_w 函数的参数 points 是指定多边形顶点的一组数值。_polygon_wxy 函数的参数 points 是 xycoord 类型结构体数组，每个数组元素对应多边形的一个顶点。

参数 numpoints 表示参数 points 中的元素个数(即顶点个数)。

参数 control 应是下面两个常量之一：_GBORDER 或_GFILLINTERIOR。

_setwritemode、_setlinestyle 和_setfillmask 函数都会影响这 3 个函数的输出。

如果成功画出多边形，则函数返回非 0 值，否则返回 0。

〖参见〗_arc,_ellipse,_floodfill,_lineto,_pie,_rectangle,_setcolor,_setfillmask,_setlinestyle。

_putimage、_putimage_w

〖功能〗_putimage 函数将存储在参数 image 指向缓存区中的图像输出到屏幕。

〖原型〗

void _putimage(short x，short y，char _huge *image，short action)

void _putimage_w(double wx，double wy，char _huge *image，short action)

〖位置〗graph.h

〖说明〗_putimage 函数将图像的左上角输出到视窗坐标点<<x>,<y>>的位置上，_putimage_w 程序将图像的左上角输出到窗口坐标点<<wx>,<wy>>的位置上，_putimage_w 程序作为宏执行。

参数 action 定义了缓存区存储的图像同屏幕上现存图像之间的作用关系，其值可取如下常量(这些常量在 GRAPH.H 中定义)之一：_GAND，_GOR，_GPRESET，_GPSET，_GXOR。

该函数无返回值。

〖参见〗_getimage,_imagesize。

_rectangle、_rectangle_w、_rectangle_wxy

〖功能〗_rectangle 函数使用当前线型画矩形。

〖原型〗

short _rectangle(short control，short x1，short y1，short x2，short y2)

short _rectangle_w(short control，double wx1，double wy1，double wx2，double wy2)

short _rectangle_wxy(short control，strucut _wxycorrd _far *pwxy1，struct _wxycoord _far *pwxy2)

〖位置〗graph.h

〖说明〗_rectangle 函数使用视窗坐标系，视窗坐标点<<x1>,<y1>>和<<x2>,<y2>>是矩形的两个对角顶点。

_rectangle_w 程序使用窗口坐标系，窗口坐标点<<wx1>,<wy1>>和<<wx2>,<wy2>>是矩形的两个对角顶点。

_rectangle_wxy 程序使用窗口坐标系，窗口坐标点<<pwxy1>>和<<pwxy2>>是矩形的两个对角顶点，_rectangle_wxy 程序中的坐标是以_wxycoord 类型结构体(在 GRAPH.H 中定义)的形式给出的。

_rectangle_w 和_rectangle_wxy 程序作为宏执行。

参数 control 可以是下列两个常量之一：_GBORDER，_GFILLINTERIOR。

若当前填充掩码是 NULL，则不使用掩码填充，反之则使用当前颜色填充矩形。如果想用_floodfill 函数填充矩形，则矩形的边界必须是实线。

如果成功画出矩形，函数返回一个非 0 值，否则返回 0。

〖参见〗_arc,_ellipse,_floodfill,_getcolor,_lineto,_pie,_setcolor,_setfillmask。

_remapallpalette、_remappalette

〖功能〗_remappalette 函数给一个颜色索引赋新的颜色值。_remapallpalette 函数同时将所有的颜色索引更新为数组 color 中的颜色值。

〖原型〗short _remapallpalette(long _far *colors)

long _remappalette(short index，long color)

〖位置〗graph.h

〖说明〗这两个函数都会立即影响当前的显示。

这两个函数可以在所有图像模式下工作，但只能支持 EGA，MCGA 或 VGA 硬件。如果在其他硬件配置，包括 Olivetti 或 Hercules(大力神)下调用函数就会产生错误。

_remapallpalette 函数只用到参数 color 数组的前 n 个元素，n 为当前视频模式支持的颜色数量。可以从 videoconfig 结构体的 numcolors 成员中获得 n 的值。

colors 的取值为_BLACK, _BLUE, _BRIGHTWHITE, _BROWN, _CYAN, _GRAY, _GREEN, _LIGHTBLUE, _LIGHTCYAN, _LIGHTGREEN, _LIGHTMAGENTA, _LIGHTRED, _YELLOW, _MAGENTA, _RED 或_WHITE 之一。

关于颜色索引的默认值和用长整型值指定颜色的有关细节，可参阅图像模式颜色。

_remapallpalette 和_remappalette 函数不会影响外观图像调色板，这些调色板用

_pg_getpalette，_pg_setpalette 和_pg_resetpalette 函数进行处理。

如果函数运行成功，remapallpalette 返回 0(short 类型)，remappalette 函数返回原先的颜色索引值。当函数返回-1 时，表示硬件配置出错(不是 VGA，MCGA 或 EGA)。当颜色索引值超出颜色范围时，_remappalette 函数也返回-1。

注意，_remapallpalette 函数返回短整型值，而_remappalette 函数返回的是长整型值。

〖参见〗_selectpalette,_setbkcolor,_setvideomode。

_scrolltextwindow

〖功能〗_scrolltextwindow 函数在文本窗口(事先由_settextwindow 函数定义)中滚动文本屏幕。

〖原型〗void _scrolltextwindow(short lines)

〖位置〗graph.h

〖说明〗参数 lines 规定了滚动的行数。正数表示向上滚动(通常方向)，负数表示向下滚动。如果指定的滚动行数超过当前文本窗口的高度，等价于调用_clearscreen(_GWINDOW)函数。lines 的值等于 0 不产生任何操作。

lines 的取值范围为_GSCROLLDOWN 或_GSCROLLUP。

该函数无返回值。

〖参见〗_gettextposition,_outmem,_outtext,_settextposition,_settextwindow。

_selectpalette

〖功能〗该函数返回原先的调色板值，无出错返回。

〖原型〗short _selectpalette(short number)

〖位置〗graph.h

〖说明〗_selectpalette 函数仅能在视频模式_MRES4COLOR 和_MRESNOCOLOR 下工作。调色板由一个可选择的背景色(颜色 0)和 3 组颜色组成。在_MRES4COLOR 模式下，参数 number 可选择下面所示的事先定义好的 4 种调色板之一。

_MRES4COLOR 调色板颜色如下。

调色板号	颜色 1	颜色 2	颜色 3
0	Green	Red	Brown
1	Cyan	Magenta	Light gray
2	Light green	Light red	Yellow
3	Light cyan	Light magenta	White

_MRESNOCOLOR 视频模式通常和黑白显示器一起使用，调色板由不同灰度值组成。当使用彩色显示器时，也可以产生颜色。可用的调色板数量取决于使用 CGA 还是 EGA。在 CGA 配置下，只能使用下面的两种调色板。

_MRESNOCOLOR 模式 CGA 调色板颜色如下。

调色板号	颜色 1	颜色 2	颜色 3
0	Blue	Red	Light gray
1	Light blue	Light red	White

在 EGA 配置下，可用使用下面的 3 种调色板。

_MRESNOCOLOR 模式 EGA 调色板颜色如下。

调色板号	颜色 1	颜色 2	颜色 3
0	Green	Red	Brown
1	Light green	Light red	Yellow
2	Light cyan	Light red	Yellow

注意，在 EGA 的_MRESNOCOLOR 视频模式下，调色板 3 和调色板 1 是一样的。

〖参见〗_getvideoconfig,_setbkcolor,_setvideomode。

_setactivepage

〖原型〗short _setactivepage(short page)

〖位置〗graph.h

〖说明〗针对内存足够支持多屏幕页的硬件和模式配置，_setactivepage 指定了图形输出写入的内存区域。参数 page 选择当前活动页面。默认页面号为 0。

屏幕的活动可通过交换图形显示页实现。使用_setvisualpage 函数显示完成的图形页面时，可同时在另一个活动页面中执行图形处理语句。

如果使用文本函数_gettextcursor，_settextcursor,_outtext,_settextposition, _gettextposition，_settextcolor，_gettextcolor，_settextwindow 和_wrapon 代替 C 的标准 I/O 函数时，还可以用_setactivepage 和_setvisualpage 函数控制文本输出。

CGA 硬件配置只能在文本模式下，有 16 KB 可用 RAM 支持多视频页面。EGA 和 VGA 配置可以在图形模式下配备 256 KB 的 RAM 支持多视频页面。

函数在成功时返回原先的活动页面号。若函数失败，则返回-1。

〖参见〗_getactivepage,_getvideoconfig,_getvisualpage,_setvisualpage。

_setbkcolor

〖功能〗_setbkcolor 函数将当前背景色设为颜色参数 color。

〖原型〗long _setbkcolor(long color)

〖位置〗graph.h

〖说明〗在彩色文本模式(如_TEXTC80)下，_setbkcolor 接收的是颜色索引值。默认颜色值参见_settextcolor 函数的有关说明。

例如，_setbkcolor(2L)将背景色设为 2 号颜色索引。显示的真实颜色取决于 2 号颜色索引在调色板中的相应值。在彩色文本模式下的默认值为绿色。

在图形模式(如，_ERESCOLOR)下，_setbkcolor 函数接收的是颜色值。最基本的背景

色值可用 GRAPH.H 中定义的常量表示(如：GREEN)。这些常量可用来定义和处理多数常用色。因此，实际的颜色范围通常都要大得多。

有关细节参阅图形模式颜色。

通常，当函数的参数为长整型时表示颜色值，为短整型时表示颜色索引。只有_setbkcolor 和_getbkcolor 这两个函数例外。

用参数 0 调用_remappalette 函数就会产生同调用_setbkcolor 同样的效果。与_remappalette 函数不同的是，_setbkcolor 不要求 EGA 或 VGA 环境。

在文本模式下，_setbkcolor 函数不会影响已经显示在屏幕上的内容(只影响以后的输出)。在图形模式下，它立即改变所有背景像素的颜色。

在文本模式下，_setbkcolor 返回旧背景色的索引。在图形模式下，_setbkcolor 函数返回 0 号索引的旧颜色值。该函数无错误返回。

〖参见〗_getbkcolor,_remappalette,_selectpalette, _setcolor,_settextcolor。

_setcliprgn

〖功能〗_setcliprgn 函数将后面的图形输出和字体文本输出限制在屏幕的某个限定区域。物理点<x1,y1>和<x2,y2>是定义限定区域的矩形对角点。

〖原型〗void _setcliprgn(short x1，short y1，short x2，short y2)

〖位置〗graph.h

〖说明〗这个函数不改变视窗坐标系，仅仅屏蔽显示屏幕。

注意，_setcliprgn 函数仅影响图形和字体文本输出。要想屏蔽文本输出屏幕，应使用_settextwindow 函数。

该函数无返回值。

〖参见〗_settextwindow,_setvieworg,_setviewport,_setwindow。

_setcolor

〖功能〗_setcolor 函数将当前颜色设置为颜色索引参数 color。

〖原型〗short _setcolor(short color)

〖位置〗graph.h

〖说明〗参数 color 是被屏蔽过的，总在处于合法的范围内，_arc，_ellipse，_floodfill，_lineto，_outgtext，_pie，_rectangle 和_setpixel 函数都使用当前颜色。

_setcolor 函数接收的是整型值参数，表示颜色索引。

默认颜色索引是当前调色板中的最高颜色索引值。

注意，_setcolor 函数不影响前面图形函数的输出。

该函数返回以前的颜色索引，如果函数失败(如这在使用文本模式)则返回-1。

〖参见〗_arc,_ellipse,_floodfill,_getcolor,_lineto,_outgtext,_pie,_rectangle,_selectpalette,_setbkcolor,_setpixel,_settextcolor。

_setfillmask

〖功能〗_setfillmask 函数设置当前填充掩码，掩码决定了填充方式。

〖原型〗void _setfillmask(unsigned char _far *mask)

〖位置〗graph.h

〖说明〗掩码是一个 8 位乘 8 位的阵列，每位代表一个像素。某位为 1 表示将对应像素设为当前颜色，某位为 0 表示不改变对应像素的颜色。在整个填充区域内重复使用掩码。

如果没有设置填充掩码(参数 mask 为 NULL——默认值)，仅用当前颜色进行填充。

该函数无返回值。

〖参见〗_ellipse,_floodfill,_getfillmask,_pie,_rectangle。

_setlinestyle

〖功能〗_setlinestyle 函数为画线选择掩码。

〖原型〗void _setlinestyle(unsigned short mask)

〖位置〗graph.h

〖说明〗参数 mask 是一个 16 位的数组，每一位代表所画直线中的一个像素。

如果某位为 1，则将相应像素设为直线颜色(当前色)。如果某位为 0，则不改变相应像素的颜色。在直线区间中重复使用该模板。

默认掩码为 0xFFFF(实线)。

该函数无返回值。

〖参见〗_getlinestyle,_lineto,_rectangle。

_setpixel、_setpixel_w

〖功能〗_setpixel 和_setpixel_w 函数将指定位置的像素设为当前色。

〖原型〗short _setpixel(short x，short y)

short _setpixel_w(double wx，double wy)

〖位置〗graph.h

〖说明〗_setpixel 函数设置视窗坐标点<x,y>的像素为当前色。

_setpixel_w 函数设置窗口坐标点<wx,wy>的像素为当前色，该程序作为宏执行。

函数返回目标像素的原来值。如果函数失败(如，目标点超出限制区域)，则返回-1。

〖参见〗_getpixel,_setcolor。

_settextcolor

〖功能〗_settextcolor 函数将当前文本颜色设为参数 index 指定的颜色，默认文本颜色是最大颜色索引值。

〖原型〗short _settextcolor(short index)

〖位置〗graph.h

〖说明〗_settextcolr 函数只为_outtext 和_outmem 函数的输出设置颜色，不会影响 printf 函数或_outgtext 字体程序的输出文本的颜色。可以使用_setcolor 函数改变字体输出的颜色。

在彩色文本模式下，可以使用 0~31 范围内的任意索引值。0~15 的索引值代表正常颜色。16~31 内的索引值是同 0~15 相同，但具有闪烁文本的颜色。普通颜色范围定义如下。

索引	颜色
0	Black
1	Blue
2	Green
3	Cyan
4	Red
5	Magenta
6	Brown
7	White
8	Dark gray
9	Light blue
10	Light green
11	Light cyan
12	Light red
13	Light magenta
14	Yellow
15	Bright white

在包括单色模式的每种文本模式下，_getvideoconfig 函数都返回可用颜色数量 32。这个数值 32 表示_settextcolor 函数能够接收的值范围(0~31)。其中包括 16 种正常色和 16 种闪烁色。

正常色索引值加 16 就得到了相应的闪烁色。由于单色文本模式的惟一显示属性较少，有些颜色值就出现冗余。但是，因为闪烁色的选择也遵循上述方法，因此单色文本模式的颜色范围同其他的文本模式一样都是 0~31。

该函数返回原先文本颜色的索引，无错误返回。

〖参见〗_gettextcolor,_outtext,_setbkcolor,_setcolor。

_settextcursor

〖功能〗_settextcursor 函数按照 attr 参数的值设置光标属性(比如，形状)。

〖原型〗short _settextcursor(short attr)

〖位置〗graph.h

〖说明〗参数 attr 的高字节确定在字符单元内光标开始的位置，低字节确定光标的终止位置。

_settextcursor 函数使用同 BIOS 程序同样的格式设置光标。

光标	形状
0x0707	字符下划线。
0x0007	完整光标。
0x0607	双下划线。
0x2000	无光标。

注意，该函数仅能在文本视频模式下工作。

该函数返回原先的光标属性，如果出现错误(如在图形模式下调用此函数)，则返回-1。

〖参见〗_displaycursor,_gettextcursor。

_settextposition

〖功能〗_settextposition 函数将当前文本位置重新部署到点<<row>,<colum>>所指位置。

〖原型〗struct rccoord _settextposition(short row，short colum)

〖位置〗graph.h

〖说明〗_outtext 和_outmem 函数(标准控制台 I/O 程序，如 printf)从这个点开始输出文本。

文本窗口的左上角的文本位置坐标为<1,1>。

该函数用 rccoord 类型结构体(在 GRAPH.H 中定义)返回原先的文本位置。

〖参见〗_gettextposition,_outtext,_settextwindow。

_settextrows

〖功能〗_settextrows 函数指定了文本模式的屏幕行数。

〖原型〗short _settextrows(short rows)

〖位置〗graph.h

〖说明〗如果参数 rows 是常量_MAXTEXTROWS，函数将会选择最大可使用行数。在文本模式下，VGA 适配器最大为 50 行，EGA 为 43 行，其他为 25 行。在图形模式下，支持 30 或 60 行，_MAXTEXTROWS 表示 60 行。

该函数返回设置的行数，如果出现错误，则返回 0。

〖参见〗_getvideoconfig,_setvideomode,_setvideomoderows。

_settextwindow

〖功能〗_settextwindow 函数限定了屏幕显示的文本输出窗口的行和列坐标。参数<<r1>, <c1>>指出了文本窗口的左上角。参数<<r2>, <c2>>指定了文本窗口的右下角。

〖原型〗void _settextwindow(short r1，short c1，short r2，short c2)

〖位置〗graph.h

〖说明〗文本窗口的左上角是第 1 行第 1 列。

文本在文本窗口中从上向下显示。当窗口满时，最上面的一行自动滚动出显示窗口。

注意该函数不会影响表述图形文本的输出(如标签、轴标记等)。它也不会影响字体显示程序_outgtext 的输出。表述图形文本和字体的显示区域由_setviewport 函数控制。

该函数无返回值。

〖参见〗_gettextposition,_gettextwindow,_outtext, _settextposition。

_setvideomode

〖功能〗_setvideomode 函数根据硬件/显示配置选择适当的屏幕模式，参数 mode 的值

为下列在 GRAPH.H 中定义的常量之一。

〖原型〗short _setvideomode(short mode)

〖位置〗graph.h

〖说明〗注意，在 OS/2 下，_setvideomode 函数只能选择文本视频模式。

屏幕模式的表示常量如下。

模式	类型	大小	颜色	适配器
_DEFAULTMODE	硬件默认模式			
_MAXRESMODE	最高分辨率图形模式			
_MAXCOLORMODE	最多颜色图形模式			
_TEXTBW40	M/T	40x25	16	CGA
_TEXTC40	C/T	40x25	16	CGA
_TEXTBW80	M/T	80x25	16	CGA
_TEXTC80	C/T	80x25	16	CGA
_MRES4COLOR	C/G	320x200	4	CGA
_MRESNOCOLOR	M/G	320x200	4	CGA
_HRESBW	M/G	640x200	2	CGA
_TEXTMONO	M/T	80x25	1	MA
_HERCMONO	大力神图形模式	720x348	1	HGC
_MRES16COLOR	C/G	320x200	16	EGA
_HRES16COLOR	C/G	640x200	16	EGA
_ERESNOCOLOR	M/G	640x350	1	EGA
_ERESCOLOR	C/G	640x350	16	EGA
_VRES2COLOR	C/G	640x480	2	VGA
_VRES16COLOR	C/G	640x480	16	VGA
_MRES256COLOR	C/G	320x200	256	VGA
_ORESCOLOR	C/G	640x400	1 of 16	Olivetti 图形

_setvideomoderows

〖功能〗_setvideomoderows 函数为特定的硬件/ 显示组合选择屏幕模式。

〖原型〗short _setvideomoderows(short mode，short rows)

〖位置〗graph.h

〖说明〗_setvideomode 函数的说明中给出了关于屏幕模式的常量，_setvideomoderows 函数同时还指定了文本模式中的文本行数。

mode 参数的取值参见-setvideomode 函数的说明。

如果参数 row 为_MAXTEXTROWS，函数就会选择可用的最大行数。文本模式下，VGA 最大是 50 行，EGA 是 43 行，其他适配器是 25 行。图形模式支持 30 行或 60 行，

_MAXTEXTROWS 表示 60 行。

该函数返回设置的行数。如果出现错误(如不支持选择的模式)，则返回 0。

〖参见〗_getvideoconfig,_settextrows,_setvideomode。

_setvieworg

〖功能〗_setvieworg 函数将视窗坐标的原点(0，0)移动到物理点<<x>,<y>>。

〖原型〗struct xycoord _setvieworg(short x，short y)

〖位置〗graph.h

〖说明〗所有其他视窗坐标点都要进行相应的变换。

注意，这个函数代替了 Microsoft C 5.1 中的_setlogorg 函数。

该函数用 xycoord 类型结构体(在 GRAPH.H 中定义)返回原先原点的物理坐标。

〖参见〗_getphyscoord,_getviewcoord,_getwindowcoord,_setcliprgn,_setviewport。

_setviewport

〖功能〗_setviewport 函数重新定义图形视口。

〖原型〗void _setviewport(short x1，short y1，short x2，short y2)

〖位置〗graph.h

〖说明〗_setviewport 函数同_setcliprgn 函数一样，定义了一块限制区域，并将视窗坐标系的原点设在该区域的左上角。

物理坐标点(<x1>,<y1>)和(<x2>,<y2>)为限定区域的对角点。调用_setwindow 函数进行的任何窗口转换都是在视口中而非整个屏幕中进行的。

该函数无返回值。

〖参见〗_setcliprgn,_setvieworg,_setwindow。

_setvisualpage

〖功能〗在具有 EGA 或 VGA 适配器并且有足够内存支持多屏幕页面的硬件配置模式下，_setvisualpage 函数选择当前可视页面。

〖原型〗short _setvisualpage(short page)

〖位置〗graph.h

〖说明〗参数 page 指定了当前可视页面，默认页面为 0 页面。

该函数返回前一个可视页面号。如果函数失败，则返回一个负数。

〖参见〗_getactivepage,_getvisualpage,_setactivepage,_setvideomode。

_setwindow

〖功能〗_setwindow 函数用指定的坐标定义了窗口边界。参数(<wx1>, <wy1>)指定了窗口的左上角顶点，(<wx2>, <wy2>)指定了窗口的右下角顶点。

〖原型〗short _setwindow(short finvert，double wx1，double wy1，double wx2，double wy2)

〖位置〗graph.h

〖说明〗参数 finvert 指定了坐标的方向。如果 finvert 为 TRUE，则 y 轴方向为从屏幕下部指向上部(笛卡尔坐标系)。如果 finvert 为 FALSE，则 y 轴方向位从屏幕上部指向下部(屏幕坐标)。

任何使用_setwindow 函数进行的窗口转换都只在视口中应用，而不是在整个屏幕应用。

若 wx1 等于 wx2 或 wy1 等于 wy2，函数失败。

注意，这个函数不会影响表述图形文本(如标签、轴标记)的输出。它也不会影响字体显示程序_outgtext 的输出。

如果该函数成功，返回一个非 0 值；否则(如不是处于图形模式)返回 0。

〖参见〗_setviewport。

_setwritemode

〖功能〗_setwritemode 函数设置当前逻辑写模式，当使用_lineto 和_rectangle 函数画线时要用到这个模式。

〖原型〗short _setwritemode(short action)

〖位置〗graph.h

〖说明〗参数 action 定义了写模式，其可能的取值为：_GAND, _GOR, _GPRESET, _GPSET 和 _GXOR。

该函数无返回值。

〖参见〗_getwritemode,_lineto,_putimage,_rectangle,_setcolor,_setlinestyle。

_wrapon

〖功能〗_wrapon 函数控制_outtext 函数的输出，当文本输出到达文本窗口边界时，是换到新的一行，还是简单地截断。

〖原型〗short _wrapon(short option)

〖位置〗graph.h

〖说明〗参数 option 可以是下列常量之一。

常量	含义
_GWRAPOFF	遇到窗口边界时截断文本行。
_GWRAPON	遇到窗口边界时换行。

注意，该函数不会影响表述图形程序或字体程序的输出。

该函数返回原先的 option 参数的值，无出错返回。

〖参见〗_outtext,_settextwindow。

3.8　图形和图表函数

_pg_analyzechart、_pg_analyzechartms

〖功能〗_pg_analyzechart 函数在不实际显示表述图形图像的情况下，分析一个或多个

数据序列。

〖原型〗

short _pg_analyzechart(chartenv _far *env，char _far *_far *categories，float _far *values，short n)

short _pg_analyzechartms(chartenv _far*env，char _far*_far *categories，float _far *values， short nseries，short n，short arraydim，char _far *_far *serieslabels)

〖位置〗pgchart.h

〖说明〗根据调用_pg_defaultchart 函数指定的类型，_pg_analyzechart 函数用一组条、列和行图表的默认值填充图表环境。

_pg_analyzechart 计算出的变量反映了参数 categories 和 values 给出的数据。它的所有参数同函数_pg_chart 函数的一样。

根据调用_pg_defaultchart 函数指定的类型，_pg_analyzechartms 函数用多组条、列和行图表的默认值填充图表环境。

_pg_analyzechartms 计算出的变量反映了参数 categories 和 values 给出的数据。它的所有参数同函数_pg_chartms 函数的一样。

在调用这两个函数的任意一个之前，应先将图表环境中的 Boolean 标志，如 AUTOSCALE 和 LEGEND 设为 TRUE，以便能够计算出所有默认值。

_pg_analyzechart 函数和_pg_analyzechartms 函数如果执行无误就返回 0，返回非 0 值表示出现错误。

〖参见〗_pg_chart,_pg_chartms,_pg_defaultchart,_pg_initchart。

_pg_analyzepie

〖功能〗_pg_analyzepie 函数在不需实际显示图像图像的情况下，分析一组数据序列。

〖原型〗short _pg_analyzepie(chartenv _far *env，char _far *_far *categories，float _far *values，short _far *explode，short n)

〖位置〗pgchart.h

〖说明〗_pg_analyzepie 函数用包含在参数 value 数组中的数据为饼形图表填充图表环境。该函数的所有参数同_pg_chartpie 函数的参数一样。

_pg_analyzepie 函数执行无误时返回 0，若函数返回非 0 值，表示函数失败。

〖参见〗_pg_chartpie,_pg_defaultchart,_pg_initchart。

_pg_analyzescatter、_pg_analyzescatterms

〖功能〗_pg_analyzescatter 组的函数无需实际显示图形图像，只是分析一组或多组数据序列。

〖原型〗

short _pg_analyzescatter(chartenv _far*env，float _far *xvalues，float _far *yvalues，short n)

short _pg_analyzescatterms(chartenv _far *env，float _far *xvalues，float _far *yvalues，short nseries，short n，short rowdim，char _far *_far *serieslabels)

〖位置〗pgchart.h

〖说明〗_pg_analyzescatter 函数为一组散布图表填充图表环境，_pg_analyzescatter 函数计算出的变量反映了参数 xvalues 和 yvalues 给出的数据，该函数的所有参数同 _pg_chartscatter 函数的参数一样。

_pg_analyzescatterms 函数为多组散布图表填充图表环境，_pg_analyzescatterms 函数计算出的变量反映了参数 xvalues 和 yvalues 给出的数据，该函数的所有参数同 _pg_chartscattemsr 函数的参数一样。

在调用 pg_analyzescatterms 函数之前，应先将图表环境中的 Boolean 标志，如 AUTOSCALE 和 LEGEND 设为 TRUE，以便该函数能够计算出所有默认值。

如果_pg_analyzescatter 和_pg_analyzescatterms 函数执行无误就返回 0，函数返回非 0 值表示出现错误。

〖参见〗_pg_chartscatter,_pg_chartscatterms,_pg_defaultchart,_pg_initchart。

_pg_chart

〖功能〗_pg_chart 函数根据图表环境变量 env 指定的类型，显示一组条、列和行图表。

〖原型〗short _pg_chart(chartenv _far *env，char _far *_far *categories，float _far *values，short n)

〖位置〗pgchart.h

〖说明〗若该函数正确执行则返回 0；否则返回一个非 0 值。

〖参见〗_pg_analyzechart,_pg_analyzechartms,_pg_chartms,_pg_defaultchart,_pg_initchart。

_pg_chartms

〖功能〗_pg_chartms 函数根据图表环境指定的类型，显示多组条、列和行图表。

〖原型〗short _pg_chartms(chartenv _far *env，char _far *_far *categories，float _far *values，short nseries，short n，short arraydim，char _far *_far *serieslabels)

〖位置〗pgchart.h

〖说明〗所有组都必须包含参数 n 指定的同样数量的数据点。

values 数组是一个二维数组，包含了每组在图表中图示的所有数据点。数组的每列代表一组图表符。参数 rowdim 是一个声明 values 数组行数的整数值。

例如，下面的程序段将标识符 values 定义为 20 行 10 列的二维浮点数组。

```
#define ARRAYDIM 20
float values [ARRAYDIM][10];
short rowdim = ARRAYDIM;
```

注意，数组 values 的列数不能超过 10，即单一图表中最大数据数量。

还要注意，rowdim 必须大于等于参数 n，数组的列数必须大于等于参数 nseries。将 n 和 nseries 的值设为小于数组 values 的全部维数就只允许对 values 数组中的部分数据进行绘图。

字符数组 serieslabels 保存了用在图表图例中识别每组序列的标签。

如果函数执行无误，返回 0；否则返回一个非 0 值。

〖参见〗_pg_analyzechart,_pg_analyzechartms,_pg_chart,_pg_defaultchart,_pg_initchart。

_pg_chartpie

〖功能〗_pg_chartpie 函数将数组 values 中包含的数据显示为饼图图表。

〖原型〗short _pg_chartpie(chartenv _far *env，char _far *_far *categories，float _far *values，short _far *explode，short n)

〖位置〗pgchart.h

〖说明〗饼图是由一组数据构成，饼图不像其他类型的图表那样有多组数据的版本。

数组 explode 必要的维数应大于等于参数 n，其所有元素的值都是 0 或 1。如果某元素为 1，则表示相应的饼图段应稍微离开其他饼图显示。

例如，如果 explode 数组初始值为：

```
short explode[5]={0,1,0,0,0};
```

显示时，categories 数组的第 2 个元素对应的饼图段就会和其他四个段分离。

如果函数执行无误，返回 0；否则返回一个非 0 值。

〖参见〗_pg_analyzepie,_pg_defaultchart,_pg_initchart。

_pg_chartscatter、_pg_chartscatterms

〖功能〗_pg_chartscatter 函数将一组数据显示成散布图表，_pg_chartsctterms 函数将多组数据显示成散布图表。

〖原型〗

short _pg_chartscatter(chartenv _far *env，float_far *xvalues，float_far *yvalues，short n)

short _pg_chartscatterms(chartenv _far *env，float _far *xvalues，float _far *yvalues，short nseries，short n，short rowdim，char _far *_far *serieslabels)

〖位置〗pgchart.h

〖说明〗参数 xvalues 和 yvalues 分别是包含 x 轴和 y 轴数据的二维数组。每个数组的列为一个序列数据;因此 xvalues 的第 1 列和 yvalues 的第 1 列就包含了第 1 组的图表数据，第 2 列包含了第 2 组的图表数据，依次类推。

参数 n、rowdim、nseries 和 seireslabels 的作用与_pg_chartms 函数相同。参阅_pg_chartms 函数的有关说明。

如果函数执行无误，返回 0；否则返回一个非 0 值。

〖参见〗_pg_analyzescatter,_pg_analyzescatterms,_pg_defaultchart,_pg_initchart。

_pg_defaultchart

〖功能〗_pg_defaultchart 函数根据参数 charttype 设置的图表类型，初始化图表环境下所有必须的变量。

〖原型〗short _pg_defaultchart(chartenv _far *env，short charttype，short chartstyle)

〖位置〗pgchart.h

〖说明〗environment 结构体中所有 title 成员均为空。调用_pg_defaultchart 函数之后，将恰当标题设置到各成员中。

参数 charttype 可以是下面 5 个常量之一。

_PG_BARCHART	_PG_PIECHART
_PG_COLUMNCHART	_PG_SCATTERCHART
_PG_LINECHART	

参数 chartstyle 参数可设为两个常量之一。根据图表类型的不同，允许选取的两个常量如下。

图表类型	可用图表样式
饼图	_PG_NOPERCENT, _PG_PERCENT
条状图	_PG_PLAINBARS, _PG_STACKEDBARS
列图	_PG_PLAINBARS, _PG_STACKEDBARS
行图	_PG_POINTANDLINE, _PG_POINTONLY
散布图	_PG_POINTANDLINE, _PG_POINTONLY

若该函数执行无误，则返回 0；否则返回一个非 0 值。

〖参见〗_pg_getchardef,_pg_getpalette,_pg_getstyleset,_pg_hlabelchart,_pg_initchart,_pg_resetpalette,_pg_resetstyleset,_pg_setchardef,_pg_setpalette,_pg_setstyleset,_pg_vlabelchart。

_pg_getchardef

〖功能〗_pg_getchardef 函数获得 ASCII 值为 charnum 的字符的当前 8 乘 8 像素的位图。并将位图存储在 chardef 数组中。

〖原型〗short _pg_getchardef(short charnum，unsigned char _far *chardef)

〖位置〗pgchart.h

〖说明〗这个函数不考虑当前注册的字体，重新获取指定字符的默认字体位图。该函数只能应用于表述图形库的 8 乘 8 光栅字体，不能对 windows 字体使用。

如果该函数执行无误返回 0，否则返回一个非 0 值。

〖参见〗_pg_defaultchart,_pg_initchart,_pg_setchardef。

_pg_getpalette

〖功能〗_pg_getpalette 函数获取所有调色板的调色板颜色、线型、填充方式和图表字符。指针参数 palette 指向一个调色板结构体数组，该数组中包含期望的调色板值。

〖原型〗short _pg_getpalette(paletteentry _far *palette)

〖位置〗pgchart.h

〖说明〗表述图形程序使用的调色板和低端图形程序使用的调色板是相互独立的。

该函数执行无误时，返回 0。如果没有调用_pg_setpalette 函数初始化当前调色板，该函数返回_BADSCREENMODE。

〖参见〗_pg_defaultchart,_pg_initchart,_pg_resetpalette,_pg_setpalette。

_pg_getstyleset

〖功能〗_pg_getstyleset 函数获取当前设置的样式。

〖原型〗void _pg_getstyleset(unsigned short _far *styleset)

〖位置〗pgchart.h

〖说明〗该函数无返回值。

〖参见〗_pg_defaultchart,_pg_initchart,_pg_resetstyleset,_pg_setstyleset。

_pg_hlabelchart

〖功能〗_pg_hlabelchart 函数向屏幕水平输出文本。

〖原型〗short _pg_hlabelchart(chartenv _far *env，short x，short y，short color，char _far *label)

〖位置〗pgchart.h

〖说明〗参数 x 和 y 是相对于图表窗口左上角的文本输出起始位置的像素坐标。

如果该函数执行无误，返回 0；否则返回一个非 0 值。

〖参见〗_pg_defaultchart,_pg_initchart,_pg_vlabelchart。

_pg_initchart

〖功能〗_pg_initchart 函数初始化表述图形包。初始化颜色、样式库，复位线型设置，构造默认调色板模式，从磁盘读入字体定义。

〖原型〗short _pg_initchart(void)

〖位置〗pgchart.h

〖说明〗所有使用表述图形的程序都要用到_pg_initchart 函数。调用表述图形库中的其他函数之前，必须首先调用该函数。

_pg_initchart 函数认为已经建立了合法的图形模式，因此只能在正确调用 C 库函数_setvideomode 之后调用它。

如果该函数执行无误返回 0，否则返回一个非 0 值。

〖参见〗_pg_defaultchart,_pg_getchardef,_pg_getpalette,_pg_getstyleset,_pg_hlabelchart,_pg_resetpalette,_pg_resetstyleset,_pg_setchardef,_pg_setpalette,_pg_setstyleset,_setvideomode,_pg_vlabelchart。

_pg_resetpalette

〖功能〗_pg_resetpalette 函数为当前屏幕模式的默认调色板，设置颜色、线型、填充方式和图表字符。

〖原型〗short _pg_resetpalette(void)

〖位置〗pgchart.h

〖说明〗表述图形程序使用的调色板和低端图形程序使用的调色板是相互独立的。

如果该函数执行无误返回 0。如果屏幕模式不合法，则返回_BADSCREENMODE。

〖参见〗_pg_defaultchart,_pg_getpalette,_pg_initchart,_pg_setpalette。

_pg_resetstyleset

〖功能〗_pg_resetstyleset 函数将当前屏幕模式的式样重新设置为默认值。
〖位置〗pgchart.h
〖原型〗void _pg_resetstyleset(void)
〖说明〗该函数无返回值。
〖参见〗_pg_defaultchart,_pg_getstyleset,_pg_initchart,_pg_setstyleset。

_pg_setchardef

〖功能〗_pg_setchardef 函数为 ASCII 码值为 charnum 的字符设置 8 乘 8 像素位图。位图存储在 chardef 数组中。
〖原型〗short _pg_setchardef(short charnum，unsigned char _far *chardef)
〖位置〗pgchart.h
〖说明〗这个函数为指定字符按默认字体设置位图，而不考虑当前注册的字体。该函数只应用于表述图形库的 8 乘 8 位的关系字体，而不能使用 Windows 字体。
若函数执行无误，则返回 0；否则返回一个非 0 值。
〖参见〗_pg_defaultchart,_pg_getchardef,_pg_initchart。

_pg_setpalette

〖功能〗_pg_resetpalette 函数为所有调色板设置颜色、线型、填充方式和图表字符。
〖原型〗short _pg_setpalette(paletteentry _far *palette)
〖位置〗pgchart.h
〖说明〗指针 palette 指向一个调色板结构体数组，该数组中包含了期望的调色板值。
表述图形程序使用的调色板和低端图形程序使用的调色板是相互独立的。
如果该函数执行无误返回 0。如果新调色板不合法，则返回_BADSCREENMODE。
〖参见〗_pg_defaultchart,_pg_getpalette,_pg_initchart,_pg_resetpalette。

_pg_setstyleset

〖功能〗_pg_setstyleset 函数使用当前的样式设置。
〖原型〗void _pg_setstyleset(unsigned short _far *styleset)
〖位置〗pgchart.h
〖说明〗该函数无返回值。
〖参见〗_pg_defaultchart,_pg_getstyleset,_pg_initchart,_pg_resetstyleset。

_pg_vlabelchart

〖功能〗_pg_vlabelchart 函数向屏幕垂直输出文本，参数 x 和 y 是相对于图表窗口左上角的文本输出起始位置的像素坐标。
〖原型〗short _pg_vlabelchart(chartenv _far *env，short x，short y，short color，char _far *label)

〖位置〗pgchart.h

〖说明〗如果该函数执行无误，则返回 0；否则返回一个非 0 值。

3.9 流 I/O 处理函数

clearerr

〖功能〗clearerr 函数复位文件指针 stream 中的错误指示器和文件结束指示器。

〖原型〗void clearerr(FILE *stream)

〖位置〗stdio.h

〖说明〗错误指示器不能自动清除。一旦指定流中的错误指示器置位，则对该流的操作始终返回错误值，直到调用 clearerr，fseek，fsetpos 或 rewind 函数为止。

该函数无返回值。

〖参见〗eof,feof,ferror,perror。

fclose

〖功能〗fclose 函数关闭参数 stream 指定的流。

〖原型〗int fclose(FILE *stream)

〖位置〗stdio.h

〖说明〗使用这个函数，要先关闭所有与流相关的缓存区。系统分配的缓存区会在关闭流的同时释放。setbuf 和 sevbuf 函数分配的缓存区不会自动释放。

若 fclose 函数成功地关闭了指定流，返回 0。当出现错误时，函数返回 EOF。

〖参见〗close,fdopen,fflush,fopen,freopen,_fsopen。

fcloseall

〖功能〗fcloseall 函数关闭除了 stdin、stdout 和 stderr(在 DOS 下还有 stdaux 和 stdprn)之外的所有打开的流。同时关闭并删除 temfile 创建的所有临时文件。

〖原型〗int fcloseall(void)

〖位置〗stdio.h

〖说明〗使用这两个函数，要先关闭所有与流相关的缓存区。系统分配的缓存区会在关闭流的同时释放。setbuf 和 sevbuf 函数分配的缓存区不会自动释放。

如果 fcloseall 函数成功，则返回关闭流的数量。当出现错误时，函数返回 EOF。

〖参见〗close,fdopen,fflush,fopen,freopen,_fsopen。

fdopen

〖功能〗fdopen 函数将输入/输出流同 handle 表示的文件联系起来，这样就允许打开文件，进行低端 I/O 缓冲和格式化操作。

〖原型〗FILE *fdopen(int handle，char *mode)

〖位置〗stdio.h

〖说明〗一旦使用 fdopen 函数，将缓存区分配给一个打开的句柄，该文件就等价于用

fopen()函数打开的流。在随后的所有 I/O 操作中，将使用新的流，而不是句柄进行。用 fclose 而不是 close 关闭流。这样会释放缓存区并同时关闭流和句柄。

mode 字符串指定了访问文件的类型。下面是在 fopen 和 fdopen 函数中用到的 mode 字符串。同时还给出了 open 和 sopen 函数中使用的相应的参数 oflag。关于 mode 字符串参数的完整描述参见 fopen 函数的说明。

类型字符串	open/sopen 函数中的等效值
r	O_RDONLY
w	O_WRONLY (usually O_WRONLY | O_CREAT | O_TRUNC)
a	O_WRONLY | O_APPEND (usually O_WRONLY | O_CREAT| O_APPEND)
r+	O_RDWR
w+	O_RDWR (usually O_RDWR | O_CREAT | O_TRUNC)
a+	O_RDWR | O_APPEND (usually O_RDWR | O_APPEND | O_CREAT)

除了上面给出的这些类型字符串之外，t 或 b 也可用在 mode 串中表示为新行指定转换模式。这些字符同 open 和 sopen 函数中的相应字符对应，如下所示。

模式	open/sopen 函数中的等效值
t	O_TEXT
b	O_BINARY

t 并不是 fopen 函数的 ANSI 标准中的一部分，它是 Microsoft 扩展的，在可移植的 ANSI C 中不能用。

如果在 mode 串中没有出现 t 或 b，则表示转换模式由默认模式变量_fmode 确定。

fdopen 函数返回指向打开流的指针，如果返回 NULL，则表示执行出现错误。

〖参见〗dup,fclose,fopen,freopen,open。

feof

〖功能〗feof 程序(作为宏执行)确定参数 stream 指定的流是否结束。

〖原型〗int feof(FILE *stream)

〖位置〗stdio.h

〖说明〗一旦文件结束，读文件操作就返回文件结束标志，直到关闭流或调用 rewind、fsetpos、fseek 或 clearerr 函数清除该标志。

当读操作首次企图越过文件结束边界时，feof 函数返回一个非 0 值。如果当前位置不是文件结束，则返回 0。该函数没有执行出错返回。

〖参见〗clearerr,eof,ferror,perror。

ferror

〖功能〗ferror 程序(宏)测试同参数 stream 关联的文件的读写错误。

〖原型〗int ferror(FILE *stream)

〖位置〗stdio.h

〖说明〗如果发生错误，则将 stream 的错误指示器置位，直到关闭流 stream 或调用 rewound 或 clearerr 函数清除它为止。

如果 stream 流无读写错误发生，则 ferror 函数返回 0；否则返回一个非 0 值。

〖参见〗clearerr,feof,fopen,perror。

fflush

〖功能〗该函数刷新缓存区内容。

〖原型〗int fflush(FILE *stream)

〖位置〗stdio.h

〖说明〗如果同 stream 关联的文件已经为输出打开，fflush 函数刷新同 stream 关联的缓存区；将缓存区的内容写入文件中。如果同 stream 关联的文件为输入打开，fflush 清除缓存区。fflush 函数取消任何先前调用 ungetc 函数的作用。

当缓存区满、关闭流或程序正常终止时，会自动刷新缓存区。

调用流之后，流始终保持打开状态。fflush 函数对未分配缓存区的流不产生影响。

若成功刷新缓存区后，fflush 函数返回 0。当指定的流没有缓存区或以只读方式打开时，函数也返回 0。出现错误时，函数返回 EOF。

〖参见〗fclose,flushall,setbuf。

fgetc、fgetchar

〖功能〗fgetc 函数从 stream 关联的文件的当前位置读入一个字符。将字符转换成整数并返回。函数将关联文件指针增 1，指向下一个字符。fgetchar 函数同 fgetc(stdin)等价。

〖原型〗int fgetc(FILE *stream)

int fgetchar(void)

〖位置〗stdio.h

〖说明〗fgetc 和 fgetchar 程序和 getc 和 getchar 类似，但 fgetc 和 fgetchar 是函数而不是宏。

fgetc 和 fgetchar 函数返回读入的字符。如果遇到文件结束或出现错误则返回 EOF。但是，EOF 也是一个合法的整数值，因此，应使用 feof 或 ferror 来区分错误或文件结束条件。

〖参见〗fputc,getc。

fgetpos

〖功能〗 fgetpos 函数获取 stream 关联文件的位置指示器的当前值，并将其存储到 pos 指向的对象中。

〖原型〗int fgetpos(FILE *stream，fpos_t *pos)

〖位置〗stdio.h

〖说明〗fsetpos 函数将 stream 的文件位置指示器设置为 pos 指向的位置，pos 的值是通过以前对 stream 调用 fgetpos 函数获得的。fsetpos 函数清除文件结束指示器，并取消对

stream 调用 ungetc 函数的任何作用。调用 fsetpos 之后，就可以对 stream 进行输入或输出操作了。

pos 的值以一种内部格式存储，只能被 fgetpos 和 fsetpos 函数使用。

如果函数执行成功，fgetpos 和 fsetpos 返回 0。执行出错时，返回一个非 0 值，并将 errno 设为 EINVAL 或 EBADF。

fgets

〖功能〗fgets 函数从 stream 中读取一个串，并存储到 string 中。

〖原型〗char *fgets(char *string，int n，FILE *stream)

〖位置〗stdio.h

〖说明〗读入串的范围是从流的当前位置开始(包括第 1 个换行符\n)到流结束或读入 n-1 个字符为止。

读入的字符存储到 string 中，自动在结尾添加空字符(\0)。如果读到换行符，也包含在该串中。

当 n 等于 1 时，srting 应为空串。fgets 函数和 gets 函数类似；不过，gets 函数用 NULL 代替换行符。

如果 fgets 函数成功，则返回 string 的值。当函数返回 NULL 时，表示出现错误或文件结束。可调用 feof 或 ferror 函数确定是否发生了错误。

〖参见〗fputs,gets,puts。

fileno

〖功能〗fileno 程序(宏)返回同 stream 关联的当前文件句柄。

〖原型〗int fileno(FILE *stream)

〖位置〗stdio.h

〖说明〗该程序无错误返回，未定义 stream 没有指定打开的文件时的返回值。

〖参见〗fdopen,filelength,fopen,freopen。

flushall

〖功能〗fulshall 函数将同输出流关联的所有缓存区内容写入相应的文件中。清除所有同输入流关联的缓存区中的当前内容。随后的读操作就可以从输入文件向缓存区读入新数据了。

〖原型〗int flushall(void)

〖位置〗stdio.h

〖说明〗当缓存区满、关闭流或程序正常终止时，都会自动刷新缓存区。

调用 fulshall 函数之后，不会关闭打开的流。

flushall 函数返回打开流的数量(包括输入和输出)。该函数无出错返回。

〖参见〗fflush。

fopen

〖功能〗fopen 函数打开 filename 指定的文件。mode 字符串表示访问文件的方式。

〖原型〗FILE *fopen(char *filename，char *mode)

〖位置〗stdio.h

〖说明〗合法的文件访问方式有 r(读)、w(写)和 a(追加)，任何一种方式都可以后跟一个+号表示同时允许读写操作。还可以在后面加字符 t(文本)或 b(二进制)指定新行的转换方式。

fopen 函数返回指向打开文件的指针，若出现错误则返回 NULL。

〖参见〗fclose,fdopen,_fsopen,ferror,fileno,freopen,open,setmode。

fprintf

〖功能〗fprintf 函数按格式向输出流中输出一组字符或数值。根据 format 中的格式说明符转换并输出每个 argument 参数。

〖原型〗int fprintf(FILE *stream，char *format [，argument]...)

〖位置〗stdio.h

〖说明〗参数 format 同 printf 中的参数 format 格式相同。有关细节参阅 printf 函数的说明。

fprintf 函数返回输出的字符数，当发生输出错误时，返回一个负数。

〖参见〗cprintf,fscanf,printf,sprintf。

fputc

〖功能〗fputc 函数向输出流的当前位置写字符 c。

〖原型〗int fputc(int c，FILE *stream)

〖位置〗stdio.h

〖说明〗fputc 和 fputchar 函数与 putc 和 putchar 类似，但 fputc 和 fputchar 是函数而不能是宏。

fputc 和 fputchar 函数返回所写的字符，如果出现错误则返回 EOF。

〖参见〗fgetc,putc。

fputchar

〖功能〗fputchar 函数同 fputc(c，stdout)等价。

〖原型〗int fputchar(int c)

〖位置〗stdio.h

〖说明〗fputc 和 fputchar 函数与 putc 和 putchar 类似，但 fputc 和 fputchar 是函数而不能是宏。

fputc 和 fputchar 函数返回所写的字符，如果出现错误则返回 EOF。

〖参见〗fgetc,putc。

fputs

〖功能〗fputs 函数将字符串 string 复制到输出流 stream 的当前位置。不复制字符串结尾的空字符(\0)。

〖原型〗int fputs(char *string，FILE *stream)；

〖位置〗stdio.h

〖说明〗fputs 函数执行成功后，返回一个非负值。若出现错误返回 EOF。

〖参见〗fgets,gets,puts。

fread

〖功能〗fread 函数从输入流 stream 中读入 count 个大小为 size 字节的数据项，并将其存储到 buffer 中。

〖原型〗size_t fread(void *buffer, size_t size，size_t count，FILE *stream)

〖位置〗stdio.h

〖说明〗同 stream 关联的文件指针(如果存在这样的指针的话)增加实际读入的字节数。

如果以文本方式打开指定流，用一个行码字符代替回车换行码对。这个替换不会影响文件指针或返回值。

如果出现错误，文件指针的位置和读入的部分数据的值都是不确定的。

fread 函数返回实际完整读入的数据项的数量，当出现错误或未读到 count 个数据项文件就结束时，这个值可能比 count 少。

可以用 feofheuo ferror 函数来区别读入错误和文件结束条件。如果 size 或 count 等于 0，fread 就返回 0，并且不修改缓存区的内容。

〖参见〗fwrite,read。

freopen

〖功能〗freopen 函数关闭当前同 stream 关联的文件，并将 stream 同指定文件 filename 关联。

〖原型〗FILE *freopen(char *filename，char *mode，FILE *stream)

〖位置〗stdio.h

〖说明〗freopen 函数通常对用于将用户指定文件重新定向到已经打开的 stdin、stdout 和 stderr 中。

同 stream 关联的新文件以 mode 方式打开，mode 是指定文件的访问类型的字符串。

合法的文件访问方式有 r(读)、w(写)和 a(追加)。任何一种方式都可以后跟一个+号表示同时允许读写操作。还可以在后面加字符 t(文本)或 b(二进制)指定新行的转换方式。

freopen 函数返回执行新打开文件的指针。如果出现错误，则关闭原先的文件，函数返回 NULL 指针。

〖参见〗fclose,fdopen,fileno,fopen,open,setmode。

fscanf

〖功能〗fscanf 函数从 stream 的当前位置将数据读入到参数 argument 给出的位置。每个 argument 参数必须是同 format 中格式说明符一致的指针类型。

〖原型〗int fscanf(FILE *stream，char *format [，argument]...)

〖位置〗stdio.h

〖说明〗参数 format 控制输入流的判读，它的格式同 scanf 函数中的参数 format 一致。参阅 scanf 函数中关于 format 的说明。

fscanf 函数返回成功转换和赋值的数据数量，返回值中不包括读入但未成功赋值的数据。

当读入到文件结束时，返回 EOF；返回值为 0 表示没有进行赋值。

〖参见〗cscanf,fprintf,scanf,sscanf。

fseek

〖功能〗fseek 函数将 stream 关联的文件指针移动到距离 origin 位置 offset 个字节的新位置。

〖原型〗int fseek(FILE *stream，long offset，int origin)

〖位置〗stdio.h

〖说明〗对该流的下一步操作就应从这个新位置开始。若要更新流，则下面就可以对其进行读或写操作了。

参数 origin 必须是鲜明常量(在 STDIO.H 中定义)之一：SEEK_CUR，SEEK_END，SEEK_SET。

fseek 函数可用来在文件的任何位置重新配置指针。指针可以超出文件的结尾，但是若将指针定位到文件开头之前，就会导致错误。

fseek 函数清除文件结束指示器，并忽略以前对 stream 调用 ungetc 函数产生的作用。

当以追加方式打开一个文件后，当前文件位置是由最后一次 I/O 操作确定，而不是由下一次写入的位置确定。如果以追加方式打开的文件尚未进行 I/O 操作，则文件位置是文件的开头。

在以文本方式打开的流中限制使用 fseek 函数，因为回车换行符抓获会导致 fseek 函数出现难以预料的结果。只有如下所示的文本模式流中才能保证 fseek 操作正常运行。

1. 对任何 origin 的值偏移量 offset 均为 0。

2. 从文件头开始，偏移量 offset 为调用 ftell 函数的返回值。

如果 fseek 函数成功执行，则返回 0；否则返回一个非 0 值。未定义在无法查找的设备中运行时函数的返回值。

〖参见〗ftell,lseek,rewind。

fsetpos

〖功能〗fsetpos 函数使用存储在 pos 中的信息将文件指针复位到调用 fgetpos 函数时的位置。

〖原型〗int fsetpos(FILE *stream，fpos_t *pos)

〖位置〗stdio.h

〖说明〗见 fgetpos。

_fsopen

〖功能〗_fsopen 函数以流的方式打开 filename 指定的文件，并根据参数 mode 和 shflag

中定义的方式将文件读或写共享。

〖原型〗FILE *_fsopen(char *filename，char *mode，int shflag)

〖位置〗stdio.h

〖说明〗字符串 mode 指定了文件的访问类型。

合法的文件访问方式有 r(读)、w(写)和 a(追加)。任何一种方式都可以后跟一个+号表示同时允许读写操作。还可以在后面加字符 t(文本)或 b(二进制)指定新行的转换方式。

参数 shflag 是由下面常量(在 SHARE.H 中定义)组成的常量表达式。

SH_COMPAT	SH_DENYRW
SH_DENYNO	SH_DENYWR
SH_DENYRD	

如果操作系统中没有安装 SHARE.COM(或 SHARE.EXE)，DOS 会忽略共享模式。

_fsopen 函数只能用在 OS/2 和 DOS 3.0 以上版本。在较低版本的 DOS 中，忽略参数 shflag。

fsopen 函数发挥指向流的指针，如果出现错误则返回 NULL 指针。

〖参见〗fclose,fdopen,ferror,fileno,fopen,freopen,open,setmode。

ftell

〖功能〗ftell 函数获取同 stream 关联的文件指针(若存在)的当前位置。该位置以相对于文件开头的偏移量表示。

〖原型〗long ftell(FILE *stream)

〖位置〗stdio.h

〖说明〗注意，当文件以追加数据的方式开始时，当前文件位置由上一次 I/O 操作确定，而不是由下一次的写入位置确定。

例如，以追加方式打开一个文件，最后一次操作为读，文件位置为下一次读操作的开始点，而不是下一次写操作应开始的位置。(文件以追加方式打开后，在进行写操作之前，文件指针移动到文件结尾。)如果以追加方式打开文件后尚未进行 I/O 操作，文件位置在文件开头。

ftell 函数返回当前文件位置。在文本模式下，由于文本模式需要进行回车换行符转换，因此 ftell 函数的返回值不反映物理字节偏移量。ftell 函数和 fseek 函数联合使用就可正确低返回文件的位置。

出现错误时，该函数返回-1L，同时将 errno 设为下面常量(在 ERRNO.H 中定义)之一：EBADF 或 EINVIAL。

当出现设备无法进行查找或 stream 没有指向打开的文件时，未定义返回值。

〖参见〗fgetpos,fseek,lseek,tell。

fwrite

〖功能〗fwrite 函数从 buffer 向输出流写 count 个大小为 length 的数据项。

〖原型〗size_t fwrite(void *buffer，size_t size，size_t count，FILE *stream)

〖位置〗stdio.h

〖说明〗同 stream 关联的文件指针增加实际写入的字节数量。

如果以文件模式代替 stream，用回车换行符对代替回车符。这个替换不会影响返回值。

fwrite 函数返回写完整的数据项的数量，出现错误时，这个数值就可能比 count 少。如果出现错误，也就无法确定文件位置指示器了。

〖参见〗fread,write。

getc

〖功能〗getc 程序从 stream 当前位置读一个字符，并将文件指针自增指向下一个字符。

〖原型〗int getc(FILE *stream)

〖位置〗stdio.h

〖说明〗getc 和 getchar 程序与 fgetc 和 fgetchar 函数类似，但 getc 和 getchar 是宏不是函数。

getc 和 getchar 程序返回读入的字符。当返回 EOF 时，表示文件结束或出错。可使用 ferror 或 feof 函数确定究竟是发生了错误还是文件结束。

〖参见〗fgetc,getch,putc,ungetc。

getchar

〖功能〗getchar 程序同 getc(stdin)作用相同。

〖原型〗int getchar(void)

〖位置〗stdio.h

〖说明〗getc 和 getchar 程序与 fgetc 和 fgetchar 函数类似，但 getc 和 getchar 是宏不是函数。

getc 和 getchar 程序返回读入的字符。当返回 EOF 时，表示文件结束或出错。可使用 ferror 或 feof 函数确定究竟是发生了错误还是文件结束。

〖参见〗fgetc,getch,putc,ungetc。

gets

〖功能〗gets 函数从标准输入流 stdin 中读入一行并存储到 buffer 中，该行是包括第一个换行符(\n)在内的所有字符。

〖原型〗char *gets(char *buffer)

〖位置〗stdio.h

〖说明〗gets 函数在返回这一行之前，用空字符(\0)替换换行符。而 fgets 函数保留换行符。

如果 gets 函数执行成功，则返回其参数的值。当出现错误或文件结束时，返回 NULL。可使用 ferror 或 feof 判断到底出现了哪种情况。

〖参见〗fgets,fputs,puts。

getw

〖功能〗getw 函数从同 stream 关联的文件中读取整型值的下一个二进制整型值。然后增加关联的文件指针，令其指向下一个未读字符。该函数假定流中无特殊对齐项。

〖原型〗int getw(FILE *stream)

〖位置〗stdio.h

〖说明〗该函数返回读到的整数，返回 EOF 表示出现错误或文件结束。但是，EOF 也是合法整数值，因此应使用 feof 或 ferror 识别文件结束或出错状态。

注意，getw 函数起初为较早的库版本提供兼容性。由于 int 类型的大小和字节顺序随系统的不同而变化，因此使用 getw 函数可能会出现移植性问题。

〖参见〗putw。

printf

〖功能〗该函数向 stdout 标准输出流格式化输出字符或数值。

〖原型〗int printf(char *format[, argument]...)

〖位置〗stdio.h

〖说明〗format 字符串包括转义字符、普通字符和格式说明符。将普通字符和转义字符按照出现的顺序复制到输出流 stdout 中。

从左到右逐个扫描 format 中的字符。如果遇到格式说明符，将其对应的参数转换成指定格式并输出。如果参数的个数比格式说明符的个数多，则忽略多余的参数。

printf 函数返回输出的数据项的个数。当出现输出错误时，返回一个负数。

〖参见〗fprintf,scanf,sprintf,vfprintf,vprintf,vsprintf。

putc、putchar

〖功能〗putc 向输出流的当前位置写字符 c，putchar 同 putc(c，stdout)的效果相同。这两个函数可作为函数使用也可作为宏使用。

〖原型〗int putc(int c，FILE *stream)

int putchar(int c)

〖位置〗stdio.h

〖说明〗putc 和 putchar 程序返回写入的字符，当出现错误时，返回 EOF。putc 可以输出任何整数，但只能输出低 8 位。

〖参见〗fputc,fputchar,getc,getchar。

puts

〖功能〗该函数向标准输出流 stdout 中写字符串 string，并将字符串结束空字符(\0)替换为换行符(\n)。

〖原型〗int puts(char *string)

〖位置〗stdio.h

〖说明〗该函数成功执行，则返回一个非 0 值。否则返回 EOF。

〖参见〗fputs,gets。

putw

〖功能〗该函数向输出流的当前位置写一个二进制整型值。

〖原型〗int putw(int binint，FILE *stream)

〖位置〗stdio.h

〖说明〗该函数假定流中没有任何对齐项，也不会影响流中的对齐项。

该函数起初是为较早库版本的兼容性而提供的。注意，由于系统的不同，整型值的字节顺序会有所差别，因此使用该函数可能会出现移植性问题。

该函数返回写入的值。当出现错误时，返回 EOF。由于 EOF 也是一个合法的整数，因此应该使用 ferror 函数来识别错误情况。

〖参见〗getw。

rewind

〖功能〗该函数将 stream 关联的文件指针重新定位到文件开头。

〖原型〗void rewind(FILE *stream)

〖位置〗stdio.h

〖说明〗调用 rewind 函数和下面的 fseek 函数语句等价：

void fseek(stream，0L，SEEK_SET)；

但两者有些区别：rewind 函数同时清除文件结束和错误指示器，但 fseek 函数只清除文件结束指示器。fseek 函数的返回值表示指针是否成功移动了。但 rewind 函数不返回任何值。

rewind 函数还能清除键盘缓存区。可以对与键盘默认关联的 stdin 流使用 rewind 函数实现。

该函数无返回值。

〖参见〗fseek,ftell。

rmtmp

〖功能〗该函数删除当前目录中由 temfile 创建的所有临时文件。

〖原型〗int rmtmp(void)

〖位置〗stdio.h

〖说明〗应该在创建临时文件的目录中使用该函数。

该函数返回关闭和删除的临时文件数。

〖参见〗flushall,tmpfile,tmpnam。

scanf

〖功能〗该函数从标准输入流中读取数据，用 format 字符串中的格式字符解释输入流，并将这些数值存储到参数指定的位置。

〖原型〗int scanf(char *format [，argument]...)

〖位置〗stdio.h

〖说明〗format 字符串包括格式说明符，还可以有普通字符。每个格式说明符对应输入流中一项，同时对应一个参数。

从左到右读取格式字符串。当遇到格式说明符时，按照该格式解释输入字段，并将结果存储到相应参数指定的位置。如果参数比说明符多，则求出多余参数的值，且忽略。未定义参数比格式说明符少的情况。

该函数返回转换并存储的字段数量，这个值可能小于格式说明符的数量。返回值不计算读入但未存储的数据项。如果第 1 次读字符遇到文件结束或字符串结束，则返回 EOF。

〖参见〗fscanf,printf,sscanf,vfprintf,vprintf,vsprintf。

setbuf

〖功能〗该函数允许用户控制 stream 的缓存区。

〖原型〗void setbuf(FILE *stream，char *buffer)

〖位置〗stdio.h

〖说明〗使用参数 stream 之前，它必须指向一个打开的文件。如果参数 buffer 为 NULL，则流 stream 无缓存区。否则 buffer 必须指向长度为 BUFSIZ 的字符数组。BUFSIZ 为定义在 STDIO.H 中的缓存区尺寸。为指定流进行 I/O 缓冲时，用自定义缓存区代替默认的系统分配缓存区。

系统默认情况下，stderr 和 stdaux 流无缓冲，但可以用 setbuf 为其分配缓存区。

该函数已经被 setvbuf 函数取代，编写代码时应首选 setvbuf 函数。setbuf 函数是为解决旧代码的可移植性而保留的。

该函数无返回值。

〖参见〗fclose,fflush,fopen,setvbuf。

setvbuf

〖功能〗setvbuf 函数允许程序控制流 stream 的缓冲方式和缓存区大小。

〖原型〗int setvbuf(FILE *stream，char *buffer，int mode，size_t size)

〖位置〗stdio.h

〖说明〗参数 stream 必须指向一个打开的文件，该文件从打开后尚未进行过读写操作。用参数 buffer 指向的长为 size 字节的数组作为缓存区，除非 buffer 为 NULL 或使用自动分配缓存区。

mode 必须是_IOFBF，_IOLBF 或_IONBF。如果 mode 为_IOFBF 或_IOLBF，则用 size 作为缓存区大小。如果 mode 为_IONBF，则不分配缓存区，并忽略 size 和 buffer。

size 的合法值为 0 到 32 768。

setvbuf 函数成功执行返回 0，如果指定了非法类型或缓存区大小则返回一个非 0 值。

〖参见〗fclose,fflush,fopen,setbuf。

sprintf

〖功能〗该函数向缓存区 buffer 中按照格式存储一组字符和数值。

〖原型〗int sprintf(char *buffer，char *format [，argument]...)

〖位置〗stdio.h

〖说明〗每个 argument 都根据 format 中的相应格式说明符转换并输出。format 包括普通字符和同 printf 函数的参数 format 相同的形式。在写入完所有字符后追加一个空字符，但它不计入返回值。

sprintf 函数返回存储在 buffer 中的字符数，不包括结尾的空字符。

〖参见〗fprintf,printf,sscanf。

sscanf

〖功能〗该函数从 buffer 向每个参数 argument 指定的位置读入数据。

〖原型〗int sscanf(char *buffer，char *format [，argument] ...)

〖位置〗stdio.h

〖说明〗每个 argument 必须是同 format 中指定类型一致的指针类型。format 控制输入流的解释，同 scanf 函数的 format 参数有相同的格式。(细节参看 scanf 函数的说明)

sscanf 函数返回成功转换和存储的数据数量。返回值不包括读入但未存储的数据项。

若读入字符串结束时，函数返回 EOF。返回值为 0 表示没有对任何数据项进行赋值。

〖参见〗fscanf,scanf,sprintf。

tempnam、tmpnam

〖原型〗char *tempnam(char *dir，char *prefix)

char *tmpnam(char *string)

〖位置〗stdio.h

〖说明〗tmpnam 函数生成一个临时文件名，可以使用它打开临时文件而不覆盖现有的文件。将这个名字存储到 string 中。如果 string 是 NULL，tmpnam 的结果为义内部静态指针。这样其后的任何函数调用都可将此指针破坏。如果 string 不是 NULL，则认为其指向一个至少 L_tmpnam 字节的数组(L_tmpnam 的值在 STDIO.H 中定义)。函数可为至多 TMP_MAX 次调用生成惟一的文件名。

tmpnam 生成的字符串由 STDIO.H 中的 P_tmpdir 定义的路径前缀后跟一组 0 到 9 字符序列组成。这个字符串的数值范围是 1 到 65 535。改变 STDIO.H 中 P_tmpnam 和 L_tmpnam 的值不会改变 tmpnam 的操作。

tempnam 函数允许程序为其他目录中的文件创建临时文件名，该文件名和现存文件名均不相同。prtfix 是文件名前缀。tempnam 函数使用宏为文件名分配存储空间，程序只负责当空间无用时回收空间。

tempnam 函数在如下目录中查找给定的文件名，按优先级顺序排列。

使用目录	条件
TMP 指定的目录	设置了 TMP 环境变量，TMP 指定的目录存在。
tempnam 的目录参数	未设置 TMP 环境变量或 TMP 指定的目录不存在。
STDIO.H 中的 P_tmpdir	参数 dir 为 NULL，或 dir 为不存在的目录名。
当前工作目录	P_tmpdir 不存在。

如果函数对上述目录的查找均失败，则 tempnam 返回 NULL。

tmpnam 和 tempnam 函数都返回指向生成的文件名的指针，除非无法创建文件名或文件名不惟一。当无法创建文件名或文件名已经存在时，tmpnam 和 tempnam 都返回 NULL。

〖参见〗tmpfile。

tmpfile

〖功能〗该函数创建临时文件并返回指向该流的指针。

〖原型〗FILE *tmpfile(void)

〖位置〗stdio.h

〖说明〗如果无法打开文件，则该函数返回 NULL。

在不改变当前工作目录的情况下，当文件关闭、程序正常终止或调用 rmtmp 时，会自动删除该临时文件。临时文件以 w+b 的模式(二进制读写)打开。

如果函数执行成功，返回文件指针。否则返回 NULL。

〖参见〗rmtmp,tempnam,tmpnam。

ungetc

〖功能〗该函数将从流 stream 中读取的字符 c 重新放回 stream 中，并清除文件结束指示器。

〖原型〗int ungetc(int c, FILE *stream)

〖位置〗stdio.h

〖说明〗stream 必须以读方式打开，随后对其进行的读操作从字符 c 开始，忽略使用 ungetc 函数将 EOF 放回流中的操作。如果从流 stream 中没有读取字符或字符 c 无法放会，则 ungetc 函数返回出错值。

如果在从流 stream 中读取字符之前，调用了 fflush、fseekfsetpos 或 rewind 函数，则会删除用 ungetc 函数放入流 stream 中的字符。在放回字符之前，文件位置指示器的值不变。

对文本流成功调用 ungetc 函数，直到将所有字符读完或删除为止，文件位置指示器才确定。对二进制流每调用一次 ungetc 函数，文件位置指示器后移一步。未定义调用前位置指示器的值为 0 时调用后的值。

若两次调用 ungetc 函数之间没有进行读操作，则产生无法预料的结果。调用 fscanf 函数后调用 ungetc 函数可能会失败，除非之前再进行一次读操作(如 getc)。这是因为 fscanf 函数本身调用了 ungetc 函数。

ungetc 函数返回字符参数 c。当返回值为 EOF 时表示无法将指定字符放回流中。

〖参见〗getc,getchar,putc,putchar。

vfprintf、vprintf、vsprintf

〖原型〗int vfprintf(FILE *stream，char *format，va_list argptr)
　　int vprintf(char *format，va_list argptr)
　　int vsprintf(char *buffer，char *format，va_list argptr)

〖位置〗stdio.h

〖说明〗vfprintf，vprintf 和 vsprintf 函数分别按格式将数据输出到流 stream、标准输出流和缓存区 buffer 中。这些函数同其对应的 fprintf、printf 和 sprintf 类似，不同之处是它们接收指向参数列表的指针而不是参数列表。

参数 format 的格式和 printf 函数的参数 format 格式一样。

参数 argprt 的类型是 va_list，在 VARARGS.H 和 STDARG.H 中定义。argprt 指向按格式转换和输出的参数列表。

vprintf 和 vsprintf 函数返回输出字符数，其中不包括结尾空字符。vfprintf 函数执行成功返回输出的字符数；若产生输出错误，则返回一个负数。

〖参见〗fprintf,printf,sprintf,va_arg,va_end,va_start。

3.10 低端 I/O 函数

close

〖功能〗该函数关闭同 handle 关联的文件。

〖原型〗int close(int handle)

〖位置〗io.h，errno.h

〖说明〗close 函数成功关闭文件，则返回 0。若出现错误则返回-1，同时将 error 设为 EBADF，表示文件句柄参数非法。

〖参见〗chsize,creat,dup,dup2,open,unlink。

creat

〖功能〗该函数要么创建新文件要么打开并截取现存的文件。

〖原型〗int creat(char *filename，int pmode)

〖位置〗io.h，sys\types.h，sys\stat.h，errno.h

〖说明〗如果 filename 指定的文件不存在，则建立指定许可权设置的文件并以写方式打开。如果文件已经存在，并允许对其写入，creat 函数将文件长度截断为 0 字节，破坏文件中以前的内容，并以写方式打开文件。

Pmode 用来进行权限设置，只适用于新建立的文件。新文件第 1 次关闭时，会接收到指定的权限设置。整型表达式 pmode 包含下面两个常量之一或全部：S_IWRITR 和 S_IREAD(在 SYS\STAT.H 中定义)。如果同时包含两个常量时，之间用按位或操作符(|)连接。pmode 参数的含义由 S_IWRITE 和 S_IREAD 确定。

如果没有给出写权限，则文件为只读。在 DOS 或 OS/2 环境下不可能设置只写权限。因此，模式 S_IWRITE 和 S_IREAD|S_WRITE 是等价的。在 MS DOS 3.0 或更高版本环境下，适用 creat 打开的文件总是以兼容模式打开(参见 sopen 函数)。

creat 函数在设置权限前，令 pmode 为当前文件权限掩码。

注意，creat 程序起初为较旧版本库的兼容性提供。当其参数 oflag 的值为 O_CREAT 和 O_TRUNC 时，同调用 creat 函数等价，对较新的代码来说，选用 creat 程序更好。

当 creat 成功执行时，返回创建文件的句柄。否则，返回-1，并将 errno 设为 EACCES,EMFILE 或 ENOENT。

〖参见〗chmod,chsize,close,dup,dup2,open,sopen,umask。

dup、dup2

〖原型〗int dup(int handle)
　　int dup2(int handle1，int handle2)

〖位置〗io.h，errno.h

〖说明〗dup 和 dup2 函数建立同当前打开文件关联的第 2 个文件句柄。可以使用两个文件句柄的任意一个对文件进行操作，建立新的文件句柄不会影响文件的访问权限。

dup 函数返回指定文件的下一个可用文件句柄，dup2 函数令 handle2 指向 handle1 指向的同一个文件。在调用时若 handle2 已经同另一个打开的文件关联，则关闭该文件。

dup 函数返回新的文件句柄。dup2 函数返回 0 表示成功。如果出现错误，两个函数都返回-1，并将 errno 设为 EBADF 或 EMFILE。

〖参见〗close,creat,open。

eof

〖功能〗该函数判断同 handle 关联的文件是否结束。

〖原型〗int eof(int handle)

〖位置〗io.h，errno.h

〖说明〗如果当前位置是文件结束位置，则 eof 函数返回 1；否则返回 0。若函数返回-1 表示出现错误；这种情况下，将 errno 设为 EBADF，表示非法文件句柄。

〖参见〗clearerr,feof,ferror,perror。

lseek

〖功能〗该函数将同 handle 关联的文件指针移动到距离 origin 位置 offset 个字节的新位置。

〖原型〗long lseek(int handle，long offset，int origin)

〖位置〗io.h，stdio.h，errno.h

〖说明〗随后对文件进行的操作就从此新位置开始执行。参数 origin 必须是常量 SEEK_SET，SEEK_CUR 或 SEEK_END(在 STDIO.H 中定义)之一。

lseek 函数可用来将文件指针重新定位到文件的任何位置。指针可以指向超出文件结尾的位置，但将指针定位到文件开头之前的位置会引起错误。

lseek 函数返回新位置距离文件开头以字节为单位的偏移量。发生错误时，返回值为-1L，同时将 errno 设为 EBADF 或 EINVAL。

未定义设备无法进行查找时(如终端和打印机)的返回值。

〖参见〗fseek,tell。

open

〖功能〗该函数打开 filename 指定的文件，并按照参数 oflag 的定义为后续读写操作做准备。

〖原型〗int open(char *filename，int oflag[，int pmode])

〖位置〗fcntl.h，io.h，sys\types.h，sys\stat.h

〖说明〗参数 oflag 的值是由下列常量组成的整型表达式常量(在 FCNTL.H 中定义)。

O_APPEN	O_EXCL	O_TEXT
O_BINARY	O_RDONLY	O_TRUNC
O_CREAT	O_RDWR	O_WRONLY

如果使用多个常量，它们之间用按位或运算符(|)连接。

使用 O_TRUNC 标志时要小心，因为它会破坏已存在文件的全部内容。

必须使用 O_RDONLY，O_RDWR 或 O_WRONLY 给出访问模式，访问模式没有默认值。

只有将模式指定为 O_CREAT 时，才需要用参数 pmode。如果文件已经存在，则忽略 pmode；否则用 pmode 指定文件权限，并在首次关闭新文件时进行设置。参数 pmode 的值是包含常量 S_IWRITE 和 S_IREAD(在 SYS\STAT.H 中定义)中的一个或全部的整型表达式。若两者同时出现，它们之间用按位或运算符(|)连接。

如果没有设置写权限，文件为只读模式。在 DOS 和 OS/2 环境下，所有文件都是可读的；因此不可能设置只写权限。这样，模式 S_IWRITE 和 S_IREAD|S_IWRTIE 就是等价的。

open 函数在设置文件权限之前，对 pmode 应用当前文件权限掩码。

open 函数中用到的 filename 受到 DOS 命令 APPEND 的影响。

DOS 3.0 和较高版本中安装了 SHARE，当 oflag 设为 O_CREAT | O_RDONLY 或 O_CREAT | O_WRONLY，pmode 设为 S_IREAD 并打开新文件时，会出现问题。操作系统在 open 进行系统调用时，会过早地关闭文件。

解决这个问题的方法是，打开文件时，将 pmode 设为 S_IWRITE；关闭文件后，调用 chmod 函数将模式改回 S_IREAD。另一种方法是，将 pmode 设为 S_IREAD，oflag 设为 O_CREAT | O_RDWR，然后打开文件。

open 函数返回打开文件的句柄。出现错误时，返回-1，并将 errno 设为 EACCES EEXIST, EINVAL，EMFILE 或 ENOENT。

〖参见〗access,chmod,close,creat,dup,dup2,fopen,sopen,umask。

read

〖功能〗该函数从与 handle 关联的文件中读 count 个字节到 buffer 中。

〖原型〗int read(int handle，void *buffer，unsigned count)

〖位置〗io.h，errno.h

〖说明〗读操作从给定文件的文件指针的当前位置开始，读完成后，文件指针指向下一个未读的字符。

read 函数返回实际读入的字节数。如果文件中剩余的字节比 count 少，或文件不是以文本模式打开，则这个值可能比 count 小。返回值为 0 表示读到文件结束符。返回值为-1 表示出现错误，这时将 errno 设为 EBADF。

如果从文件中读的字节数超过 32 K(整型的最大值)，返回值应为一个无符号整数。但是，因为 65 535(0xFFFF)和-1 的表示形式一样，无法和错误返回区别，所以每次从文件中读的字节数不能超过 65 534。

如果以文本模式打开文件，则返回值可能不是实际读入的字节数。在文本模式下，每对回车换行对都被单独的换行符代替。在返回值中只计入了单个的换行符。这种替换不会影响文件指针。

注意，在 DOS 和 OS/2 环境下以文本模式打开文件时，将字符 CTRL+Z 看成文件结束指示符。一遇到 CTRL+Z 就终止读入，下一次的读入将返回 0 字节。可以用 lseek 函数清除文件结束指示符。

〖参见〗creat,fread,open,write。

sopen

〖功能〗sopen 函数打开 filename 指定的文件，并按照 oflag 和 shflag 定义的方式为后续的共享读写做好准备。

〖原型〗int sopen(char *filename，int oflag，int shflag [，int pmode])

〖位置〗fcntl.h，io.h，share.h，sys\types.h，sys\stat.h，errno.h

〖说明〗整型表达式 oflag 由下面一个或多个常量组成(在 FCNTL.H 中定义)。(如果包含多个常量，则各常量之间用按位或操作符| 连接。)

O_APPEND	O_EXCL	O_TEXT
O_BINARY	O_RDONLY	O_TRUNC
O_CREAT	O_RDWR	O_WRONLY

参数 shflag 的值为下面常量(在 SHARE.H 中定义)组成的表达式。

SH_COMPAT	SH_DENYRW
SH_DENYNO	SH_DENYWR
SH_DENYRD	

如果没有安装 SHARE.EXE(或 SHARE.COM)，MS DOS 忽略共享模式。

sopen 函数只能在 OS/2 和 MS DOS 3.0 以上版本下使用，早期的 MS DOS 版本忽略参数 shflag。

只有将模式指定为 O_CREAT 时，才需要用参数 pmode。如果文件已经存在，则忽略 pmode；否则用 pmode 指定文件权限，并在首次关闭新文件时进行设置。参数 pmode 的值是包含常量 S_IWRITE 和 S_IREAD(在 SYS\STAT.H 中定义)中的一个或全部的整型表达式。若两者同时出现，它们之间用按位或运算符(|)连接。

如果没有设置写权限，文件为只读模式。在 DOS 和 OS/2 环境下，所有文件都是可读的；因此不可能设置只写权限。这样，模式 S_IWRITE 和 S_IREAD|S_IWRTIE 等价。

注意，安装了 SHARE 的 MS DOS 3.x 版本环境中，下列条件下用 sopen 打开新文件时会出现问题。

1. oflag 设为 O_CREAT | O_RDONLY 或 O_CREAT | O_WRONLY，pmode 设为

S_IREAD，shflag 设为 SH_COMPAT。

2. oflag 为包含 O_FLAG 的任何组合，pmode 为 S_IREAD，shflag 为除了 SH_COMPAT 之外的任何值。

在任何一种情况下，sopen 函数进行系统调用时，操作系统会过早关闭文件，或系统中出现共享违规(INT 24H)。为了避免这些问题，打开文件时，将 pmode 设为 S_IWRITE。关闭文件之后，再调用 chmod 将模式改回 S_IREAD。另一个解决方法是，将 pmode 设为 S_IREAD，oflag 设为 O_CREAT | O_RDWR，shflag 设为 SH_COMPAT。

sopen 函数在设置权限前，对 pmode 应用当前文件权限掩码。

sopen 函数返回打开文件的句柄。出现错误时，返回-1，同时将 errno 设为 EACCES，EEXIST，EMFILE 或 ENOENT。

〖参见〗close,creat,fopen,open,umask。

tell

〖功能〗该函数获取同参数 handle 相关的文件指针的当前位置，该位置以距离文件开头的字节数表示。

〖原型〗long tell(int handle)

〖位置〗io.h，errno.h

〖说明〗该函数执行出现错误时，返回-1L，并将 errno 设为 EBADF 表示非法文件句柄。该函数未定义设备无法查找情况下的返回值。

〖参见〗ftell,lseek。

umask

〖功能〗该函数按照参数 pmode 指定的模式为当前进程设置文件权限掩码。

〖原型〗int umask(int pmode)

〖位置〗io.h，sys\types.h，sys\stat.h。

〖说明〗文件权限掩码用于修改由 creat、open 或 sopen 函数创建的新文件的权限设置。如果掩码的某一位为 1，则文件权限值的相应位就被设为 0(禁止)；如果掩码的某一位为 0，则权限值的相应位保持不变。设置好新的文件权限，要关闭文件后才生效。

参数 pmode 是包含常量 S_IWRITE 和 S_IREAD(在 SYS\STAT.H 中定义)的一个常量表达式。若两个常量同时出现，则用按位或(|)运算符连接。

例如，若设置掩码中的写位，则新文件属性为只读。

注意，在 DOS 和 OS/2 环境下，所有的文件都是可读的，不可能设置只写权限。因此用 umask 函数设置读位不会修改文件的模式。

umask 函数返回参数的 pmode 原始值，无错误返回。

〖参见〗chmod,creat,mkdir,open。

write

〖功能〗write 函数从 buffer 中向同 handle 关联的文件写入 count 个字节。

〖原型〗int write(int handle，void *buffer，unsigned count)

〖位置〗io.h，errno.h

〖说明〗写操作从指定文件的文件指针的当前位置开始。如果文件以追加方式打开，则操作从文件的当前结尾开始。写操作执行完后，文件指针增加了实际写入的字节数。

write 函数返回实际写入的字节数，返回值可能是小于 count 的正数(例如，当写入 count 个字节之前，磁盘空间越界)。

返回值为-1 表示出现错误，这时，将 errno 设为 EBADF 或 ENOSPC。

如果向文件中写的字节数超过 32 K(整型的最大值)，返回值应为一个无符号整数。但是，因为 65 535(0xFFFF)和-1 的表示形式一样，无法和错误返回区别，所以每次向文件中写的字节数不能超过 65 534。

如果以文本模式打开文件，输出时将每个换行符用一对回车换行对代替。这种替换不会影响返回值。

当以文本写模式打开文件时，write 函数将字符 CTRL+Z 看成文件逻辑结束符。当向设备写入时，write 函数将缓存区中的 CTRL+Z 看做输出终止符。

〖参见〗fwrite, open, read。

3.11　控制台和端口 I/O 函数

cgets

〖功能〗该函数直接从控制台读取一个字符串，将该字符串及其长度存储到 buffer 指向的位置。

〖原型〗char *cgets(char *buffer)

〖位置〗conio.h

〖说明〗参数 buffer 的值必须是指向字符数组的指针。数组的第 1 个元素 buffer[0]中存储允许读取字符串的最大长度(以字符为单位)。数组必须足够容纳字符串、结束空字符和两个附加字节。

cgets 函数遇到回车换行组合或读入了指定数量的字符后才停止读入。字符串从 buffer[2]开始存储。如果遇到回车换行组合，则用空字符(\0)代替之并存储。然后 cgets 函数在第 2 个数组元素 buffer[1]中存储字符串的实际长度。

调用 cgets 函数时，所有 DOS 的编辑键均有效。因此，可以用 F3 重复最后一项输入。

cgets 函数返回指向字符串开始的指针，即 buffer 的地址。该函数无错误返回值。

〖参见〗getch,getche。

cprintf

〖功能〗该函数直接向控制台格式化输出一系列字符或数值。

〖原型〗int cprintf(char *format [，argument]...)

〖位置〗conio.h

〖说明〗根据参数 format 中的格式说明符，逐个将参数 argument 转换并输出。

〖参见〗cscanf,fprintf,printf,sprintf,vprintf。

cputs

〖功能〗该函数将 string 指向的字符串直接写入控制台。注意，不会自动添加回车换行符(CR-LF)。

〖原型〗int cputs(char *string)

〖位置〗conio.h

〖说明〗若函数执行成功，返回 0，否则返回一个非 0 值。

〖参见〗putch。

cscanf

〖功能〗该函数直接从控制台读取数据，并写入到 argument 指定的存储单元。该函数使用 getche 函数读字符。

〖原型〗int cscanf(char *format [, argument] ...)

〖位置〗conio.h

〖说明〗每个可选的参数 argument 必须是指向变量的指针，并且要和 format 中指定的格式说明符匹配。参数 format 对输入字段进行解释，其格式与 scanf 函数的 format 参数相同。

cscanf 函数通常都回显输入的字符，但调用 ungetch 函数之后立即调用此函数时不能回显。

cscanf 函数返回成功转换并赋值的字段数量，返回值不包括读取但没有赋值的数据项。

若读到文件结束，则返回 EOF。当将键盘输入重定向为操作系统命令行时，可能会发生这种情况。返回值为 0 表示没有进行赋值。

〖参见〗cprintf,fscanf,scanf,sscanf。

getch

〖功能〗该函数从控制台读取一个字符，并且不在屏幕上回显。

〖原型〗int getch(void)

〖位置〗conio.h

〖说明〗当读取功能键或光标移动键时，必须调用两次 getch 函数。第 1 次函数返回 0 或 0xE0，第 2 次调用返回实际的键码。

该函数返回读取的字符，并且无出错返回值。

〖参见〗cgets,getchar,ungetch。

getche

〖功能〗该函数从控制台读取一个字符并回显读到的字符。

〖原型〗int getche(void)

〖位置〗conio.h

〖说明〗该函数返回读取的字符并且无出错返回值。

〖参见〗cgets,getchar,ungetch。

inp、inpw

〖功能〗inp 和 inpw 函数分别从指定的输入端口读取一个字节和一个字。

〖原型〗int inp(unsigned port)

unsigned inpw(unsigned port)

〖位置〗conio.h

〖说明〗参数 port 的值可以是 0 到 65 535 之间的任何无符号整型值。

必须使用.DEF 文件声明使用运行时库对端口执行输入输出操作的 IOSEG 段。另外，这些函数的内部(/Oi)版本只有在.DEF 文件的标记为 IOPL 关键字的段中才能工作。

由于不能在常规代码段中执行 IOPL，运行时库声明一个名为_IOSEG 的单独的代码段。为了在任意保护模式运行库(?LIBCP, LLIBCDLL, LLIBCMT, 或基于 CDLLOBJS 的 DLL)中使用所有的这些函数，必须在.DEF 文件中包含下面的语句行。

```
SEGMENTS _IOSEG CLASS 'IOSEG_CODE' IOPL
```

inp 和 inpw 函数返回从 port 中读取的字节或字，并且无出错返回值。

kbhit

〖功能〗该函数检查控制台的最近击键情况。

〖原型〗int kbhit(void)

〖位置〗conio.h

〖说明〗如果函数返回非 0 值，则表示缓存区中有按键等待。程序可调用 gech 或 geche 获取按键。

如果有按键，kbhit 函数返回非 0 值，否则返回 0。

outp、outpw

〖功能〗outp 和 outpw 函数分别向指定的输出端口写一个字节和一个字。

〖原型〗int outp(unsigned port, int databyte)

unsigned outpw(unsigned port, unsigned dataword)

〖位置〗conio.h

〖说明〗参数 port 的值可以是 0 到 65 535 之间的任何无符号整型值，参数 databyte 可以是任何 0 到 255 范围内的整数，参数 dataword 可以是任何 0 到 65 535 之间的无符号整型值。

必须使用.DEF 文件声明使用运行时库对端口执行输入输出操作的 IOSEG 段。另外，这些函数的内部(/Oi)版本只有在.DEF 文件的标记为 IOPL 关键字的段中才能工作。

由于不能在常规代码段中执行 IOPL，运行时库声明一个名为_IOSEG 的单独的代码段。为了在任意保护模式运行库(?LIBCP, LLIBCDLL, LLIBCMT, 或基于 CDLLOBJS 的 DLL)中使用所有的这些函数，必须在.DEF 文件中包含下面的语句行。

```
SEGMENTS _IOSEG CLASS 'IOSEG_CODE' IOPL
```

outp 和 outpw 函数返回输出的字节或字，并且无出错返回值。

putch

〖功能〗该函数直接向控制台写入字符 c(无需缓存区)。
〖原型〗int putch(int c)
〖位置〗conio.h
〖说明〗如果函数执行成功，则返回字符 c，否则返回 EOF。
〖参见〗cprintf, getch, getche。

ungetch

〖功能〗ungetch 函数将字符 c 重新写回控制台，使 c 成为 getch 或 getche 函数读取的下一个字符。
〖原型〗int ungetch(int c)
〖位置〗conio.h
〖说明〗若在下次读操作前多次调用 ungetch 函数，则会失败，参数 c 的值不能是 EOF。ungetch 函数成功执行后，返回字符 c。若出现错误则返回 EOF。
〖参见〗cscanf, getch, getche。

3.12 定 位 函 数

localeconv

〖功能〗该函数获取程序当前点数值格式指定场所设置的细节信息，该信息存储在 lconv 类型的结构体(在 LOCALE.H 中定义)中。
〖原型〗struct lconv *localeconv(void)
〖位置〗locale.h
〖说明〗localeconv 函数返回指向该 lconv 类型的指针。用类别值 LC_ALL, LC_MONETARY 或 LC_NUMERIC 调用 setlocale 函数会将该结构体中的内容覆盖。
〖参见〗setlocale,strcoll,strftime,strxfrm。

setlocale

〖功能〗setlocale 函数控制程序中与区域有关的状态，如日期格式。
〖原型〗char *setlocale(int category，char *locale)
〖位置〗locale.h
〖说明〗category 参数指定受影响的函数，参数 locale 指定获取属性信息的区域。参数 category 的值必须是以下定义在 LOCALE.H 中的常量之一。

LC_ALL	LC_MONETARY
LC_COLLATE	LC_NUMERIC

LC_CTYPE　　LC_TIME

参数 locale 是一个指向位置名称字符串的指针。如果 locale 指向空串，则 locale 在执行过程中由本地环境决定。值“C”表示 C 转换遵循的 ANSI 最小环境，这也是 Microsoft C 6.0 版本惟一支持的区域。

如果参数 locale 是 NULL，则 setlocale 将返回指向与程序区域目录关联的字符串的指针，并且不修改程序当前区域设置。

如果 locale 和 category 参数均合法，则 setlocale 函数返回指向与新区域目录关联的字符串的指针。若 locale 或 category 的值非法，则 setlocale 函数返回 NULL，且不修改当前程序区域设置。

可以在后续函数调用中使用 setlocale 函数返回的指针恢复程序的区域信息。下一次调用 setlocale 函数会覆盖该字符串。

〖参见〗localeconv,strcoll,strftime,strxfrm。

strcoll

〖功能〗该函数比较两个字符串。

〖原型〗int strcoll(char *string1，char *string2)

〖位置〗string.h

〖说明〗和 strcmp 函数不同，strcoll 函数使用区域信息提供待比较的序列。目前，只能支持“C”区域，strcoll 的执行过程和 strcmp 类似。

strcoll 函数比较 string1 和 string2，并根据比较结果返回下面的值。

值	含义
< 0	<string1> 小于<string2>
= 0	<string1> 等于 <string2>
> 0	<string1> 大于 <string2>

〖参见〗localeconv,setlocale,strcmp,strncmp,strxfrm。

strftime

〖功能〗strftime 函数根据提供的格式参数 format 将 timeptr 指向的 tm 类型的时间值格式化，将结果存储到缓存区 string 中。

〖原型〗size_t strftime(char *string，size_t maxsize，char *format，struct tm *timeptr)

〖位置〗time.h

〖说明〗参数 maxsize 为缓存区中可存储的最多字符数。

格式由一个或多个代码组成；同 printf 一样，格式代码以％开头。不是以％开头的字符原样复制到 string 中。当前区域的属性 LH_TIME 会影响 strftimes 的输出格式。

下面是 strftime 的格式代码。

格式	描述
%A	缩写星期名。
%A	完整星期名。
%B	缩写月份名称。
%B	完整月份名称。
%C	于当前位置一致的日期和时间表示法。
%D	用十进制表示每月中的日期 (01~31)。
%H	用 24 小时格式表示时间 (00~23)。
%I	用 12 小时格式表示时间 (01~12)。
%J	用十进制数表示一年中的每一天(001~366)。
%M	用十进制数表示 12 个月(01~12)。
%M	用十进制数表示分 (00~59)。
%P	用当前位置的 AM/PM 表示 12 小时时钟。
%S	用十进制数表示每秒 (00~61)。
%U	用十进制数表示一年中的星期；星期日为一周的开头(00~53)。
%W	用十进制数表示一周的每天(0~6; 星期日为 0)。
%W	用十进制数表示一年中的星期；星期一为每周的第 1 天(00~53)。
%X	当前位置的日期表示法。
%X	当前位置的时间表示法。
%Y	两位十进制数表示年份 (00~99)。
%Y	年份的完整表示法(4 位十进制数)。
%Z	时区名或缩写；若时区未知则不加字符。
%%	百分号。

当结果字符串(包括结束空字符)不超过 maxsize 时，strftime 函数返回 string 中存储的字符数。否则 strftime 返回 0，string 中的内容不确定。

〖参见〗localeconv,setlocale,strxfrm。

strxfrm

〖功能〗strxfrm 函数将 string2 指向的字符串转换成 string1 指向的数组形式。

〖原型〗size_t strxfrm(char *string1，char *string2，size_t count)

〖位置〗string.h

〖说明〗对两个转换后的字符串调用 strcmp 函数，相当于对两个原始字符串调用 strcoll 函数。

最多可以转换并存储到结果字符串中 count 个字符(包括结尾空字符)。

下面是计算存储源字符串转换结果所需数组尺寸的表达式：

```
1 + strxfrm( NULL, string, 0 )
```

目前，C 库仅支持“C”区域。这样 strxfrm 函数等价于：

```
strncpy( string1，string2，count )；
return( strlen( string2 ) )；
```

strxfrm 函数返回转换字符串的长度，不包括结尾空字符。如果返回值大于等于 count，则 string1 的内容无法预测。

〖参见〗localeconv,setlocale,strcoll。

3.13　数 学 函 数

acos、asin、atan、atan2

〖原型〗double acos(double x)
　　double asin(double x)
　　double atan(double x)
　　double atan2(double y, double x)

〖位置〗math.h，errno.h

〖说明〗acos 函数、asin 函数、atan 函数和 atan2 函数分别返回参数 x 的反余弦值、反正弦值和反正切值，其中参数 x 为弧度数。atan2 函数用两个参数的符号确定返回值的象限。下面为参数和返回值的范围。

函数	参数范围	返回值范围
acos	-1 到 1	0 到 π
asin	-1 到 1	-π/2 到 π/2
atan	无限	-π/2 到 π/2
atan2	无限	-π 到 π

long double 类型函数使用 80 位 long double 类型的参数和返回值，其他方面同普通函数相同。

acos 和 asin 函数的参数 x 必须在-1 到 1 之间。如果 x 大于 1 或小于-1，则函数将 errno 设为 EDOM，向 stderr 输出 DOMAIN 错误信息并返回 0。

可以通过 matherr(或_matherrl)程序修改出错处理。

〖参见〗cos,matherr,sin,tan。

acosl

〖原型〗long double acosl(long double x)

〖位置〗math.h，errno.h

〖说明〗参见 acos 的说明。

〖参见〗cos,cosl,matherr,_matherrl,sin,sinl,tan,tanl。

asinl

〖原型〗long double asinl(long double x)
〖位置〗math.h，errno.h
〖说明〗参见 acos 的说明。
〖参见〗cos,cosl,matherr,_matherrl,sin,sinl,tan,tanl。

atanl

〖原型〗long double atanl(long double x)
〖位置〗math.h，errno.h
〖说明〗参见 acos 的说明。
〖参见〗cos,cosl,matherr,_matherrl,sin,sinl,tan,tanl。

atan2l

〖原型〗long double atan2l(long double y, long double x)
〖位置〗math.h，errno.h
〖说明〗参见 acos 的说明。
〖参见〗cos,cosl,matherr,_matherrl,sin,sinl,tan,tanl。

cabs、cabsl

〖原型〗double cabs(struct complex z)
long double cabsl(struct _complexl z)
〖位置〗math.h
〖说明〗cabs 函数计算复数的绝对值，该复数以 complex 类型结构体表示。结构体变量 z 由一个实部 x 和一个虚部 y 组成。调用 cabs 函数相当于计算下面的表达式：

```
sqrt( z.x * z.x + z.y * z.y )
```

cabsl 函数使用 80 位 long double 类型的参数和返回值，其余与 cabs 函数相同。
当发生溢出时，函数调用 matherr(或_matherrl)程序，返回 HUGE_VAL(cabsl 函数返回 _LHUGE_VAL)，并将 errno 设为 ERANGE。
〖参见〗abs,fabs,labs。

ceil、ceill

〖原型〗double ceil(double x)
long double ceill(long double x)
〖位置〗math.h
〖说明〗ceil 函数返回大于等于参数 x 的最小整数的浮点表示法。
long double 类型的函数使用 80 位 long double 类型的参数和返回值，其余与 ceil 函数相同。

这两个函数无出错返回值。

〖参见〗fmod。

_clear87

〖功能〗该函数返回浮点状态字，并将状态字清 0。

〖原型〗unsigned _clear87(void)

〖位置〗float.h

〖说明〗_status87 函数相反，仅返回状态字，不进行任何修改。浮点状态字由 8087/80287 状态字和 8087/80287 测试到的除了如浮点栈溢出等处理器状态之外的其他条件组成。

返回值的每一位代表浮点状态。关于每一位的详细定义，参看 FLOAT.H。

很多 math 库函数都会不可预知地修改 8087/80287 的状态字。由于_clear87 和_status87 函数在已知浮点状态字间执行的浮点操作很少，因此它们的返回值相对更可靠。

〖参见〗_control87。

_control87

〖功能〗_control87 函数获取并设置浮点控制字，控制字允许程序修改精度、舍入和浮点 math 包中的无穷模式。

〖原型〗unsigned _control87(unsigned new，unsigned mask)

〖位置〗float.h

〖说明〗可以使用_control87 函数对浮点异常事件进行屏蔽或解除屏蔽。

如果参数 mask 等于 0，则_control87 函数获取浮点控制字。如果 mask 为非 0 值，则按照下述方式设置新的浮点控制字：mask 中为 1 的位用参数 new 的相应位替换控制字。设置过程等价于下面的方式：

```
fpcntrl = ( ( fpcntrl & ~mask ) | ( new & mask ) )
```

其中 fpcntrl 为浮点控制字。

下面是掩码常量 mask 和新控制字 new 的取值。

掩码常量	含义	控制值	十六进制值	含义
MCW_EM	中断异常		0x003F	
		EM_INVALID	0x0001	非法操作
		EM_DENORMAL	0x0002	非正规
		EM_ZERODIVIDE	0x0004	被 0 除
		EM_OVERFLOW	0x0008	上溢
		EM_UNDERFLOW	0x0010	下溢
		EM_INEXACT	0x0020	精度
MCW_IC	无穷控制		0x1000	
		IC_AFFINE	0x1000	仿射
		IC_PROJECTIVE	0x0000	投影

MCW_RC	舍入控制		0x0C00	
		RC_CHOP	0x0C00	截断
		RC_UP	0x0800	进入
		RC_DOWN	0x0400	舍掉
		RC_NEAR	0x0000	就近舍入
MCW_PC	精度控制		0x0300	
		PC_24	0x0000	24 位
		PC_53	0x0200	53 位
		PC_64	0x0300	64 位

函数返回值的每一位代表了浮点控制状态，每位的详细定义参阅 FLOAT.H。

〖参见〗_clear87,_status87。

cos、sin、tan

〖原型〗double cos(double x)
double sin(double x)
double tan(double x)

〖位置〗math.h

〖说明〗cos、sin、tan 函数分别返回参数 x 的余弦、正弦和正切值。

对应的 long double 类型函数使用 80 位的参数和返回值，其余同以上这些函数。

如果参数 x 过大，则三角函数的结果可能会丢失有效位。这时，函数会产生 PLOSS 错误。如果参数 x 太大，以至于全部有效位均丢失，则函数行 stderr 输出 TLOSS 信息并返回 0。这两种情况下，都将 errno 设为 ERANGE。

〖参见〗acos,asin,atan,atan2,matherr,sinh,tanh。

cosh、sinh、tanh

〖原型〗double cosh(double x)
double sinh(double x)
double tanh(double x)

〖位置〗math.h，errno.h

〖说明〗cosh、sinh、tanh 是双曲函数，分别返回参数 x 的双曲余弦、双曲正弦和双曲正切值。

对应的 long double 类型的函数使用 80 位的 long double 类型参数和返回值，其他方面与以上这些函数相同。

调用 cosh 时，若结果太大，则返回 HUGE_VAL(long double 函数返回_LHUGE_VAL)并将 errno 设为 ERANGE。

调用 sinh 时，若结果太大，则函数返回+ HUGE_VAL 或-HUGE_VAL(long double 函数返回+_LHUGE_VAL 或-_LHUGE_VAL)，并将 errno 设为 ERANGE。

tanh 函数无错误返回值。

〖参见〗acos,asin,atan,atan2,matherr,sin,tan。

coshl、sinhl、tanhl

〖原型〗long double coshl(long double x)
long double sinhl(long double x)
long double tanhl(long double x)
〖位置〗math.h，errno.h
〖说明〗参见 cosh 的说明。
〖参见〗acosl,asinl,atanl,atan2l,_matherrl,sinl,tanl。

cosl、sinl、tanl

〖原型〗long double cosl(long double x)
long double sinl(long double x)
long double tanl(long double x)
〖位置〗math.h
〖说明〗参见 cos 的说明。
〖参见〗acosl,asinl,atanl,atan2l,_matherrl,sinhl,tanhl。

dieeetomsbin、dmsbintoieee、fieeetomsbin、fmsbintoieee

〖原型〗int dieeetomsbin(double *src8, double *dst8)
int dmsbintoieee(double *src8, double *dst8)
int fieeetomsbin(float *src4, float *dst4)
int fmsbintoieee(float *src4, float *dst4)
〖位置〗math.h
〖说明〗dieeetomsbin 程序将 IEEE 标准格式的双精度数值转换成 Microsoft 二进制格式。dmsbintoieee 程序将 Microsoft 二进制双精度数值转换为 IEEE 标准格式。

fieeetomsbin 程序将 IEEE 标准格式的单精度数值转换成 Microsoft 二进制格式。fmsbintoieee 程序将 Microsoft 二进制单精度数值转换为 IEEE 标准格式。

这些程序使得 C 程序(按照 IEEE 标准格式存储浮点数值)能够使用由 Microsoft BASIC 版本以 Microsoft 二进制格式存储浮点数创建的数据文件。反之亦然。

参数 src4 或 src8 指向待转换的浮点数，将结果存储到参数 dst4 或 dst8 指向的位置。

这些程序不处理 IEEE NAN(非数值)和无穷值，并在转换过程中将非正规的 IEEE 数值转换为 0。

如果转换成功，则函数返回 0；当转换导致溢出时，返回 1。

div、ldiv

〖功能〗div 和 ldiv 函数计算参数 numer 除以参数 denom 的商和余数。div 函数的参数是整型，ldiv 函数的参数是长整型。

〖原型〗div_t div(int numer，int denom)

ldiv_t ldiv(long int numer，long int denom)

〖位置〗stdlib.h

〖说明〗商的符号与算术商相同，其绝对值是小于算术商的绝对值的最大整数。如果除数为 0，则程序提供错误信息并终止。

div 函数返回 div_t 结构体类型，该类型由商和余数组成。ldiv 函数返回 ldiv_t 结构体类型。这两种结构体均在 stdlib.h 中定义。

exp

〖功能〗exp 函数返回浮点参数 x 的指数函数值。

〖原型〗double exp(double x)

〖位置〗math.h，errno.h

〖说明〗当函数结果发生上溢时，返回 HUGE_VAL(expl 返回_LHUGE_VAL)，并将 errno 设为 ERANGE；当结果下溢时，返回 0，但不设置 errno。

〖参见〗log。

expl

〖功能〗该函数使用 80 位的 long double 类型参数和返回值，其余与 exp 函数相同。

〖原型〗long double expl(long double x)

〖位置〗math.h，errno.h

〖说明〗当函数结果发生上溢时，返回 HUGE_VAL(expl 返回_LHUGE_VAL)，并将 errno 设为 ERANGE；当结果下溢时，返回 0，但不设置 errno。

〖参见〗log。

fabs

〖原型〗double fabs(double x)

〖位置〗stdlib.h，math.h

〖说明〗参见 abs 说明。

〖参见〗cabs。

fabsl

〖原型〗long double fabsl(long double x)

〖位置〗stdlib.h，math.h

〖说明〗参见 abs 说明。

〖参见〗cabs。

floor、floorl

〖原型〗double floor(double x)

long double floorl(long double x)

〖位置〗math.h

〖说明〗floor 函数返回小于等于参数 x 的最大整数的浮点表示法。

long double 类型的函数使用 80 位 long double 类型的参数和返回值，其他与 floor 函数相同。

这两个函数无出错返回值。

〖参见〗fmod。

fmod

〖功能〗该函数计算参数 x/y 的余数 f 的浮点值，这样 x=i*y+f，i 是一个整数，f 是和 x 同符号的数，f 的绝对值应小于 y 的绝对值。

〖原型〗double fmod(double x, double y)

〖位置〗math.h

〖说明〗该函数返回余数的浮点值。如果除数 y 为 0，则函数返回 0。

〖参见〗ceil,fabs。

fmodl

〖功能〗该函数使用 80 位的 long double 类型参数和返回值，其余与 fmod 函数相同。

〖原型〗long double fmodl(long double x, long double y)

〖位置〗math.h

〖说明〗该函数返回余数的浮点值。如果除数 y 为 0，则函数返回 0。

〖参见〗ceil,fabs。

_fpreset

〖功能〗该函数重新初始化浮点算术包，通常和 signal、system 或 exec、spawn 函数一起使用。

〖原型〗void _fpreset(void)

〖位置〗float.h

〖说明〗如果程序用 signal 函数追踪浮点错误信号(SIGFPE)，可以通过调用_fpreset 函数和 longjmp 函数安全地从浮点错误中恢复。

在 MS DOS 3.0 以前的版本中，如果使用 8087/80287 协处理器，exec、spawn 或 system 子进程可能会影响父进程的浮点状态。因此，当使用协处理器时，则推荐采用下列预防措施。

- 当对浮点表达式求值时，不要调用 exec、spawn 和 system 函数。
- 执行上述进程后，如果子进程有可能进行了任何浮点操作，则应调用_fpreset 函数。

该函数无返回值。

〖参见〗execl..., execv..., signal, spawnl..., spawnv...。

frexp

〖功能〗该函数将浮点数 x 分割为尾数 m 和指数 n。m 的绝对值大于等于 0.5，小于 1.0，x 等于 $m*2^n$。

〖原型〗double frexp(double x, int *expptr)
〖位置〗math.h
〖说明〗整型指数 n 存储在参数 expptr 指向的位置。
该函数返回尾数值。如果 x 等于 0，则尾数和指数均为 0，函数返回 0。
该函数无出错返回值。
〖参见〗ldexp,modf。

frexpl

〖功能〗该函数使用 80 位 long double 类型参数和返回值，其余与 frexp 函数相同。
〖原型〗long double frexpl(long double x, int *expptr)
〖位置〗math.h
〖说明〗该函数返回尾数值。如果 x 等于 0，则尾数和指数均为 0，函数返回 0。
该函数无出错返回值。
〖参见〗ldexp,modf。

hypot、hypotl

〖原型〗double hypot(double x, double y)
long double hypotl(long double x, long double y)
〖位置〗math.h，errno.h
〖说明〗hypot 函数计算直角三角形斜边的长度，两条直角边的长度分别由参数 x 和 y 给定。调用 hypot 函数等价于下面的语句：

```
sqrt( x * x + y * y );
```

hypotl 函数使用 80 位 long double 类型参数和返回值，其余与 hypot 函数相同。
这两个函数返回斜边的长度。如果计算结果上溢，则 hypot 返回 HUGE_VAL，hypotl 返回_LHUGE_VAL，同时均将 errno 设为 ERANGE。
〖参见〗cabs。

j0、j1、jn

〖原型〗double j0(double x)
double j1(double x)
double jn(int n, double x)
〖位置〗math.h
〖说明〗j0、j1、jn 分别返回第一类贝塞尔函数次序为 0，1，n 的值。
对应的 long double 类型函数使用 80 位 long double 类型参数和返回值，其他与相应的普通函数相同。
这些函数返回 x 的贝塞尔函数值。对 y0、y1 和 yn，若 x 为负数，则将 errno 设为 EDOM，并向 stderr 输出 DOMAIN 错误信息，同时返回-HUGE_VAL。
可使用 matherr(或_matherrl)程序修改错误处理过程。

〖参见〗_j0l,_j1l,_jnl,matherr,_y0l,_y1l,_ynl。

_j0l、_j1l、_jnl

〖原型〗long double _j0l(long double x)
　　long double _j1l(long double x)
　　long double _jnl(int n, long double x)
〖位置〗math.h
〖说明〗参见 j0、ji、jn 说明。
〖参见〗j0,j1,jn,matherr,_matherrl,y0,y1,yn。

ldexp、ldexpl

〖原型〗double ldexp(double x, int exp)
　　long double ldexpl(long double x, int exp)
〖位置〗math.h，errno.h
〖说明〗ldexp 函数计算并返回 x*2exp 的值。ldexpl 函数使用 80 位的 long double 类型参数和返回值，其余与 ldexp 函数相同。

如果计算结果发生上溢，则 ldexp 函数根据 x 的符号返回+HUGE_VAL 或-HUGE_VAL，ldexpl 函数返回+LHUGE_VAL 或-LHUGE_VAL。同时将 errno 设为 ERANGE。
〖参见〗frexp,modf。

log、logl

〖原型〗double log(double x)
　　long double logl(long double x)
〖位置〗math.h，errno.h
〖说明〗log 函数计算 x 的自然对数值。logl 函数使用 80 位 long double 类型参数和返回值，其余与 log 函数相同。

这两个函数返回 x 的自然对数值。若 x 为负数，则向 stderr 输出 DOMAIN 错误信息，返回-HUGE_VAL(long double 类型函数输出-LHUGE_VAL)，并将 errno 设为 EDOM。如果 x 为 0，则函数向 stderr 输出 SING 错误信息，返回-HUGE_VAL，并将 errno 设为 ERANGE。

可以调用 matherr(或_matherrl)程序修改错误处理的方式。
〖参见〗exp,matherr,pow。

log10、log10l

〖原型〗double log10(double x)
　　long double log10l(long double x)
〖位置〗math.h，errno.h
〖说明〗log10 函数计算 x 的以 10 为底的对数值。log10l 函数使用 80 位 long double 类型参数和返回值，其余与 log10 函数相同。

这两个函数返回 x 以 10 为底的对数值。若 x 为负数，则向 stderr 输出 DOMAIN 错

误信息，返回-HUGE_VAL(long double 类型函数返回-LHUGE_VAL)，并将 errno 设为 EDOM。如果 x 为 0，则函数向 stderr 输出 SING 错误信息，返回-HUGE_VAL，并将 errno 设为 ERANGE。

可以调用 matherr(或_matherrl)函数修改错误处理的方式。

〖参见〗exp,matherr,pow。

matherr、_matherrl

〖原型〗int matherr(struct exception *except)
int _matherrl(struct _exceptionl *except)

〖位置〗math.h

〖说明〗matherr 函数处理由 math 库函数产生的错误。当这些函数检测到错误时，会调用适当的 matherr 函数处理。_matherrl 函数使用 80 位 long double 类型的参数和返回值。

可以自己定义其他的 matherr 函数，以便执行特殊的错误处理。

exception 结构体中包含下列成员。

成员	描述
int type	exception 类型。
char * name	指向出现错误的函数名。
double arg1, arg2	函数的第 1 和第 2 个实参(如果存在的话)。
double retval	函数的返回值。

type 成员指明 math 错误的类型，是下列常量(在 MATH.H 中定义)之一。

DOMAIN	SING
OVERFLOW	TLOSS
PLOSS	UNDERFLOW

结构体成员 name 是指向包含出错函数名的字符串的指针，其成员 arg1 和 arg2 指明导致错误的数值。(如果只有一个参数，则存储在 arg1 中。)

所给错误的默认返回值在 retval 中。如果修改了返回值，则必须指明修改的返回值是否确实出现了错误。

如果 matherr 返回 0，则表示显示了错误信息并将 errno 设为适当的错误值。如果 matherr 返回一个非 0 值，则表示未显示错误信息，errno 的值也未修改。matherr 函数返回 0 表示有错误，返回非 0 值表示成功。

〖参见〗acos,asin,atan,atan2,bessel,cabs,cos,cosh,exp,hypot,log,pow,sin,sinh,sqrt,tan。

modf、modfl

〖原型〗double modf(double x, double *intptr)
long double modfl(long double x, long double *intptr)

〖位置〗math.h

〖说明〗modf 函数将浮点表示的数值 x 转换成小数和整数两部分，每部分的符号与 x 相同。该函数返回带符号的 x 的小数部分，将整数部分以浮点数值存储到 intptr 指向的位置。

modfl 函数使用 80 位 long double 类型参数和返回值，其余与 modf 函数相同。

该函数无错误返回值。

〖参见〗frexp,ldexp。

pow、powl

〖原型〗double pow(double x, double y)

long double powl(long double x, long double y)

〖位置〗math.h

〖说明〗pow 函数计算 x 的 y 次幂的值。powl 函数使用 80 位 long double 类型参数和返回值，其余与 pow 函数相同。

pow 和 powl 函数返回 x 的 y 次幂的值，其计算结果根据 x 和 y 的不同有所差别：

- 如果 x 不等于 0.0 但 y 等于 0.0，函数返回 1.0。
- 如果 x 等于 0.0，y 为负数，函数将 errno 设为 EDOM 并返回 0.0。
- 如果 x 和 y 均等于 0.0，或 x 为负数，y 不是整数，则函数向 stderr 输出 DOMAIN 错误信息并将 errno 设为 EDOM，同时返回 0.0。
- 如果结果发生上溢，函数将 errno 设为 ERANGE 并返回+HUGE_VAL 或-HUGE_VAL(pow)，+LHUGE_VAL 或-LHUGE_VAL(powl)。如果出现下溢，则返回 0.0，不设置 errno。发生上溢和下溢时都不输出错误信息。

当浮点值超过 2 的 64 次幂时如 1.0E100，pow 函数不能完整识别。

〖参见〗exp,log,sqrt。

sqrt、sqrtl

〖原型〗double sqrt(double x)

long double sqrtl(long double x)

〖位置〗math.h，errno.h

〖说明〗sqrt 函数计算参数 x 的平方根。sqrtl 函数使用 80 位 long double 类型的参数和返回值，其余与 sqrt 函数相同。

sqrt 函数返回计算出的平方根。如果 x 是负数，该函数向 stderr 输出 DOMAIN 错误信息，并将 errno 设为 EDOM，同时返回 0。

可以使用 matherr(或_matherrl)函数修改错误处理方式。

〖参见〗exp,log,matherr,pow。

_status87

〖功能〗该函数返回浮点状态字，并且不对其进行任何修改。

〖原型〗unsigned _status87(void)

〖位置〗float.h

〖说明〗浮点状态字由 8087/80287 状态字和 8087/80287 测试到的除了如浮点栈溢出等处理器状态之外的其他条件组成。

返回值的每一位代表浮点状态，关于每一位的详细定义，参看 FLOAT.H。

很多 math 库函数都会不可预知地修改 8087/80287 的状态字。由于_clear87 和_status87 函数在已知浮点状态字间执行的浮点操作很少，因此其返回值相对更可靠。

〖参见〗_control87。

y0、y1、yn

〖原型〗double y0(double x)
double y1(double x)
double yn(int n, double x)

〖位置〗math.h

〖说明〗y0、y1、yn 分别返回第二类贝塞尔函数次序为 0，1，n 的值，其参数 x 必须为正数。

对应的 long double 类型函数使用 80 位 long double 类型参数和返回值，其他与相应的普通函数相同。

这些函数返回 x 的贝塞尔函数值。对 y0、y1 和 yn，若 x 为负数，则将 errno 设为 EDOM，并向 stderr 输出 DOMAIN 错误信息，同时返回-HUGE_VAL。

可使用 matherr(或_matherrl)函数修改错误处理过程。

〖参见〗_j0l,_j1l,_jnl,matherr,_y0l,_y1l,_ynl。

_y0l、_y1l、_ynl

〖原型〗long double _y0l(long double x)
long double _y1l(long double x)
long double _ynl(int n, long double x)

〖位置〗math.h

〖说明〗参见 y0、y1、yn 说明。

〖参见〗j0,j1,jn,matherr,_matherrl,y0,y1,yn。

3.14 内存分配、释放及重新分配函数

alloca

〖功能〗该函数从程序栈中分配 size 个字节。当调用该函数退出时，将自动释放所分配的空间。

〖原型〗void *alloca(size_t size)

〖位置〗malloc.h

〖说明〗不推荐使用 alloca 函数，只有同早期版本兼容的源程序才支持这个函数，使

用该函数的程序应关闭优化编译(即编译时选用/Od 选项)。

当进行打开优化编译时(默认方式或使用/O 选项)，在没有局部变量但使用了 alloca 函数的函数中，栈指针可能无法正确恢复。为了确保栈指针能够正确恢复，在任何使用 alloca 的函数中至少要定义一个局部变量。

Alloca 函数返回的指针值不应该作为参数传递给 free 函数，也不应将 alloca 用在函数参数的表达式中。

Alloca 函数返回指向分配空间的空类型指针，这样，可使得指针适合于任何类型的对象。若要使空类型指针指向某种类型的对象，则应对返回值进行类型转换。如果无法分配存储空间，则该函数返回值为 NULL。

〖参见〗calloc,malloc,realloc。

calloc、_bcalloc、_fcalloc、ncalloc

〖功能〗这一组函数分配 num 个大小为 size 字节的存储空间，每个存储单元均初始化为 0。

〖原型〗void *calloc(size_t num，　size_t size)
　　void _based(void) *_bcalloc(_segment seg，size_t num，size_t size)
　　void _far *_fcalloc(size_t num，size_t size)
　　void _near *_ncalloc(size_t num，size_t size)

〖位置〗malloc.h，stdlib.h

〖说明〗在大型数据模式(压缩、大型程序)calloc 函数映射为_fcalloc 函数。在小型数据模式中(中、小型程序)，calloc 函数映射为_ncalloc 函数。

各种不同的 calloc 函数分配存储块的数据段位置为如下。

函数	堆段
calloc	根据程序的数据模块。
_bcalloc	由段选择器 seg 指定的基堆。
_fcalloc	远程堆 (默认数据段之外)。
_ncalloc	近程堆(默认数据段内)。

calloc 函数返回指向所分配空间的指针，该类指针可适用于任何数据类型。若想令其指向某种数据类型，应对返回值进行类型转换。

当无足够空间分配或参数 num、size 为 0 时，_fcalloc 和_ncalloc 函数返回 NULL，_bcalloc 函数返回_NULLOFF。

〖参见〗free,halloc,hfree,malloc,realloc。

_expand、_bexpand、_fexpand、_nexpand

〖功能〗这组_expand 函数修改以前分配的存储空间的大小，增大或缩小空间后不会移动其在堆中所处的位置。

〖原型〗

void *_expand(void *memblock，size_t size)

void _based(void) *_bexpand(_segment seg，void _based(void) *memblock，size_t size)

void _far *_fexpand(void _far *memblock，size_t size)

void _near *_nexpand(void _near *memblock，size_t size)

〖位置〗malloc.h

〖说明〗参数 memblock 指向存储块的起始位置，参数 size 给出该存储块新的字节数，修改存储块的大小后，不改变其中的内容。

参数 memblock 也可以指向已经释放的存储块，只要释放该块后没有调用过 calloc、_expend、malloc 或 realloc 函数。若 memblock 指向已经释放的存储块，当调用_expand 函数之后，该块仍保持释放状态。

参数 segment 是基堆的段地址。

在大型数据模式(压缩、大型程序)中，_expand 函数映射为_fexpand 函数。在小型数据模式中(中、小型程序)，_expand 映射为_nexpand。

不同的_expand 函数修改数据段中不同位置的存储块的大小，如下所示。

函数	数据段
expand	根据程序的数据规模。
_bexpand	由 seg 指定的基堆，若 seg 为 0 则为所有基堆。
_fexpand	远程堆(默认数据段之外)。
_nexpand	近程堆(默认数据段内)。

_expand 函数返回指向重新分配空间的空类型指针。与 realloc 函数不同，_expand 不能通过移动存储块的位置来改变大小。这就表明，若有足够空间扩展存储块的话，参数 memblock 等于函数的返回值。

当不移动存储块且无足够空间扩展时，函数返回 NULL；_bexpand 函数返回 _NULLOFF。这种情况下，memblock 存储块将在当前位置尽可能地扩展。

返回值指向的空间适用于存储任何数据类型，可用 msize 函数检查新的数据对象的大小。若想获得指向某种数据类型的指针，应对返回值进行类型转换。

〖参见〗calloc,free,malloc,_msize,realloc。

free、_bfree、_ffree、_nfree

〖功能〗这组 free 函数用来释放存储块。

〖原型〗void free(void *memblock)

void _bfree(_segment seg，void _based(void) *memblock)

void _ffree(void _far *memblock)

void _nfree(void _near *memblock)

〖位置〗malloc.h，stdlib.h

〖说明〗参数 memblock 指向先前调用 calloc、malloc 或 realloc 函数分配的存储块，释放的字节数就分配(或重新分配)该存储块时指定的字节数。调用 free 函数后，该存储块

可参与再分配。

参数 seg 指定包含使用_bfree 函数释放的存储块的基堆。

释放非法指针可能会影响随后的空间分配操作，并会导致错误。未通过调用动态分配函数得到的指针是非法的。

下面是使用 free、_ffree 和_nfree 函数的限制。

分配存储块的函数	应使用的释放函数
calloc, malloc, realloc	free
_fcalloc, _fmalloc, _frealloc	_ffree
_ncalloc, _nmalloc, _nrealloc	_nfree
_bcalloc, _bmalloc, _brealloc	_bfree

这些函数将忽略 NULL 指针参数。

在大型数据模式(压缩、大型程序)中，free 函数映射为_ffree 函数。在小型数据模式(中、小型程序)中，free 函数映射为_nfree。

下面是各种 free 函数释放存储块的段位置。

函数	数据段
free	根据程序数据规模。
_bfree	seg 指定的基堆。
_ffree	远程堆(默认数据段之外)。
_nfree	近程堆(默认数据段内)。

这组函数无返回值。

〖参见〗calloc,malloc,realloc。

_bfreeseg

〖功能〗该函数释放基堆。

〖原型〗int _bfreeseg(_segment seg)

〖位置〗malloc.h

〖说明〗参数 seg 是由以前调用_bheapseg 函数返回的基堆，指明要释放的基堆。

释放的字节数为存储块分配时的字节数。调用该函数之后，释放的基堆可以重新进行分配。

_bfreeseg 函数成功执行后返回 0；若出现错误，则返回-1。

〖参见〗_bcalloc,_bexpand,_bfree,_bheapseg,_bmalloc,_brealloc。

_heapadd、_bheapadd

〖功能〗_heapadd 函数和_bheapadd 函数为堆增加一块未使用的存储空间。

〖原型〗int _heapadd(void _far *memblock, size_t size)

int _bheapadd(_segment seg，void _based(void) *memblock，size_t size)

〖位置〗malloc.h

〖说明〗_bheapadd 函数将存储空间增加到参数 seg 指定的基堆上。_heapadd 函数将查看段值，若为 DGROUP，将存储空间增加到近堆上；否则，将存储空间增加到远堆上。

若函数执行成功则返回 0，出现错误则返回-1。

〖参见〗_bcalloc,_bexpand,_bfree,_bheapmin,_bmalloc,_bmsize,_brealloc。

_heapchk、_bheapchk、_fheapchk、_nheapchk

〖功能〗_heapchk 系列函数通过检查堆的最小相容性帮助查找与堆有关的问题。

〖原型〗int _heapchk(void)

int _bheapchk(_segment seg)

int _fheapchk(void)

int _nheapchk(void)

〖位置〗malloc.h

〖说明〗如下所示，每个函数检查特定的堆。

函数	检查的堆
_heapchk	根据程序数据规模决定。
_bheapchk	seg 指定的基堆。
_fheapchk	远程堆(默认数据段之外)。
_nheapchk	近程堆(默认数据段内)。

在大型数据模式(压缩、大型程序)中，_heapchk 函数映射为_fheapchk 函数。在小型数据模式(中、小型程序)中，_heapchk 函数映射为_nheapchk 函数。

这 4 个函数均返回下面 4 个整数常量(在 malloc.h 中定义)之一：_HEAPOK，_HEAPEMPTY，_HEAPBADBEGIN，_HEAPBADNODE。

〖参见〗_heapset,_heapwalk。

_heapmin、_bheapmin、_fheapmin、nheapmin

〖功能〗这组_heapmin 函数通过向操作系统释放无用的存储空间将堆最小化。

〖原型〗int _heapmin(void)

int _bheapmin(_segment seg)

int _fheapmin(void)

int _nheapmin(void)

〖位置〗malloc.h

〖说明〗不同的_heapmin 函数将释放下面所示堆中的无用空间。

函数	最小化的堆
_heapmin	根据程序的数据规模决定。

_bheapmin	seg 指定的基堆，_NULLSEG 指定所有基堆。
_fheapmin	远程堆(默认数据段之外)。
_nheapmin	近程堆(默认数据段内)。

在大型数据模式(压缩、大型程序)中，_heapmin 函数映射为_fheapmin 函数。在小型数据模式(中、小型程序)中，_heapmin 函数映射为_nheapmin 函数。

不能用_bheapmin 函数释放基堆段(即，切断同基堆列表的连接并释放给操作系统)，可以用_bfreeseg 函数实现这项功能。

_heapmin 函数调用成功时返回 0，出错时则返回-1。

〖参见〗_bfreeseg,free,malloc。

_bheapseg

〖功能〗该函数分配一个至少 size 个字节的基堆段(由于队列和维护信息需要占用空间，因此该空间可能比 size 字节多)。

〖原型〗_segment _bheapseg(size_t size)

〖位置〗malloc.h

〖说明〗堆代码会根据需要扩展堆空间。如果最初的存储块已用完(例如，调用了_bmalloc 或_brealloc 函数)，则运行时代码会根据需要扩充堆空间。

_bheapseg 函数的返回值是基堆段的标识符，应将这个值保存起来，以便随后调用其他基堆函数时使用。

可以反复调用_bheapseg 函数。每次调用该函数时，C 库都会分配一个新的基堆段。

_bheapseg 函数返回最近分配的段选择器，将其保存以便随后调用基堆函数时使用。返回值为-1 表示出错。

注意即使所要求空间很小，也一定要检查_bheapseg 函数的返回值(尤其是在实模式下运行时)。

〖参见〗_bcalloc,_bexpand,_bfree,_bfreeseg,_bheapmin,_bmalloc,_brealloc。

_heapset、_bheapset、_fheapset、_nheapset

〖功能〗这组_heapset 函数通过显示自由空间位置或无意覆盖掉的节点来帮助检测与堆有关的问题。

〖原型〗int _heapset(unsigned fill)
　　int _bheapset(_segment seg, unsigned fill)
　　int _fheapset(unsigned fill)
　　int _nheapset(unsigned fill)

〖位置〗malloc.h

〖说明〗_heapset 函数首先以与_heapchk 函数同样的方式查看堆的最小相容性，然后为_heapset 函数堆中自由项的每个字节填入一个数值。这个已知的数值将会显示堆的哪些位置包含自由节点，哪些自由空间被无意写入了数据。

不同的_heapset 函数检查和填充不同的堆。

函数	最小化的堆
_heapset	根据程序的数据规模决定。
_bheapset	seg 指定的基堆，_NULLSEG 指定所有基堆。
_fheapset	远程堆(默认数据段之外)。
_nheapset	近程堆(默认数据段内)。

在大型数据模式(压缩、大型程序)中，_heapset 函数映射为_fheapset 函数。在小型数据模式(中、小型程序)中，_heapset 函数映射为_nheapset 函数。

这 4 个函数均返回下面 4 个整数常量(在 malloc.h 中定义)之一：_HEAPOK，_HEAPEMPTY，_HEAPBADBEGIN，_HEAPBADNODE。

〖参见〗_heapchk,_heapwalk。

_heapwalk、_bheapwalk、_fheapwalk、_nheapwalk

〖功能〗这组 heapwalk 程序帮助检查程序中与堆有关的问题。

〖原型〗int _heapwalk(_HEAPINFO *entryinfo)
int _bheapwalk(_segment seg, _HEAPINFO *entryinfo)
int _fheapwalk(_HEAPINFO *entryinfo)
int _nheapwalk(_HEAPINFO *entryinfo)

〖位置〗malloc.h

〖说明〗_heapwalk 程序沿着堆遍历，每调用一次函数遍历一项。这些函数返回指向_heapinfo 类型结构体(在 malloc.h 中定义)的指针，这个结构体包含着下一个堆项的有关信息。

返回_HEAPOK 的_heapwalk 函数调用，将该项的大小存储到_size 成员中，并将_useflag 自动设为_FREEENTRY 或_USEDENTRY(这两个常量都定义在 malloc.h 中)。向_heapwalk 传递一个指向_pentry 成员为 NULL 的_heapinfo 类型结构体的指针，可获得堆中第 1 项的信息。

不同的_heapwalk 函数按下面所示遍历指定的堆。

函数	遍历的堆
_heapwalk	根据程序的数据规模决定。
_bheapwalk	seg 指定的基堆，_NULLSEG 指定所有基堆。
_fheapwalk	远程堆(默认数据段之外)。
_nheapwalk	近程堆(默认数据段内)。

在大型数据模式(压缩、大型程序)中，_heapwalk 函数映射为_fheapwalk 函数。在小数据模式(中、小型程序)中，_heapwalk 函数映射为_nheapwalk 函数。

这 4 个函数均返回下面 4 个整数常量(在 malloc.h 中定义)之一：_HEAPOK，_HEAPEMPTY，_HEAPBADPTR，_HEAPBADBEGIN，_HEAPBADNODE 或_HEAPEND。

〖参见〗_heapchk,_heapset。

malloc、_bmalloc、_fmalloc、_nmalloc

〖功能〗这组 malloc 函数分配至少 size 个字节的内存块。

〖原型〗void *malloc(size_t size)
　　void _based(void) *_bmalloc(_segment seg, size_t size)
　　void _far *_fmalloc(size_t size)
　　void _near *_nmalloc(size_t size)

〖位置〗malloc.h，stdlib.h(仅对 ANSI 兼容宏)

〖说明〗由于数组和维护信息需要空间，内存块可能会超过 size 个字节。如果参数 size 为 0，malloc 函数在堆中分配一个 0 长度项，并返回指向该内存项的合法指针。

返回指针指向的存储空间适合于存储任何类型的对象，对返回值进行类型转换可获得指向某种特定类型的指针。

在大型数据模式(压缩、大型程序)中，malloc 函数映射为_fmalloc 函数。在小数据模式(中、小型程序)中，malloc 函数映射为_nmalloc 函数。

_fmalloc 函数在默认数据段之外分配至少 size 字节的空间，_fmalloc 函数返回空类型远程指针，如果需要超过 64 KB 大小的存储块，则应使用 halloc 函数。

_bmalloc 函数在由段选择器 seg 指定的基堆中分配至少 size 字节的存储块。

不同的 malloc 函数在不同的堆中分配存储块。

函数	分配的堆
malloc	根据程序的数据规模决定。
_bmalloc	seg 指定的基堆。
_fmalloc	远程堆(默认数据段之外)。
_nmalloc	近程堆(默认数据段内)。

如果程序可同时在实模式和保护模式下运行，应该同时用 APILMR.OBJ，API.LIB 和 OS2.LIB 进行约束。当程序使用_nmalloc 函数时，在任何情况下都应该这样做。

下面是调用了 malloc 家族函数的函数，C 启动代码使用 malloc 函数为 environ/envp[]和 argv[]字符串和数组分配空间。

调用_nmalloc 的 C 运行时库函数有：_nrealloc()　_ncalloc()　_nstrdup()。

调用_fmalloc 函数的 C 运行时库函数有：_frealloc()　_fcalloc()　_fstrdup()。

调用 malloc 函数的 C 库函数如下。

calloc()	fseek()	scanf()
execl()	fsetpos()	_searchenv()
execle()	_fullpath()	setvbuf()
execlp()	fwrite()	spawnl()
execlpe()	getc()	spawnle()

execv()	getchar()	spawnlp()
execve()	getcwd()	spawnlpe()
execvp()	_getdcwd()	spawnv()
execvpe()	gets()	spawnve()
fgetc()	getw()	spawnvp()
fgetchar()	_popen()	spawnvpe()
fgets()	printf()	strdup()
fprintf()	putc()	system()
fputc()	putchar()	tempnam()
fputchar()	putenv()	ungetc()
fputs()	puts()	vfprintf()
fread()	putw()	vprintf()
fscanf()	realloc()	

下面的函数仅在多流运行时库中(LLIBCMT，LLIBCDLL 和 CDLLOBJS)函数调用 malloc 函数。

asctime()	localtime()	_strerror()
_beginthread()	mktime()	tmpfile()
ctime()	strerror()	tmpnam()
gmtime()		

在 Microsoft C 5.1 版本中，当默认数据段之外没有足够空间可分配时，_fmalloc 函数会重新尝试在默认数据段中分配空间。在这种情况下，6.0 以上版本返回 NULL。

5.1 版本中，仅当使用通配符扩展名时，启动代码才会使用 malloc 函数。

_freect，_memavl 和_memmax 函数在 5.1 版本中调用 malloc 函数，但在 6.0 版本中不再调用。

malloc 函数返回指向分配空间的空类型指针。_nmalloc 函数返回(void _near *)，_fmalloc 返回(void _far *)。_bmalloc 函数返回(void _based(void) *)。

当无足够空间可分配时，malloc，_fmalloc 和_nmalloc 函数返回 NULL，_bmalloc 函数返回_NULLOFF。

注意，即使所需分配空间很小时，也应检查 malloc 的返回值，查看是否成功分配了空间。

〖参见〗_bfreeseg,_bheapseg,calloc,free,realloc。

_msize、_bmsize、_fmsize、_nmsize

〖功能〗这组_msize 函数以字节为单位返回由相应的内存分配函数 calloc，malloc 或 realloc 分配的空间的大小。

〖原型〗size_t _msize(void *memblock)

size_t _bmsize(_segment seg，void _based(void) *memblock)
size_t _fmsize(void _far *memblock)
size_t _nmsize(void _near *memblock)

〖位置〗malloc.h

〖说明〗在大型数据模式(压缩、大型程序)中，_msize 函数映射为_fmsize 函数。在小数据模式(中、小型程序)中，_msize 函数映射为_nmsize 函数。

各种_msize 函数显示的内存块的位置如下。

函数	数据段
Msize	根据程序的数据规模决定。
_bmsize	seg 指定的基堆。
_fmsize	远程堆(默认数据段之外)。
_nmalloc	默认数据段(近程堆内)。

这 4 个函数都返回无符号整型，表示存储空间的字节数。

〖参见〗calloc,_expand,_fmalloc,malloc,_nmalloc,realloc。

_realloc、_brealloc、_frealloc、_nrealloc

〖功能〗这组 realloc 函数改变先前所分配内存块的大小。

〖原型〗

void *realloc(void *memblock, size_t size)
void _based(void) *_brealloc(_segment seg，void _based(void) *memblock，size_t size)
void _far *_frealloc(void _far *memblock，size_t size)
void _near *_nrealloc(void _near *memblock，size_t size)

〖位置〗malloc.h，stdlib.h

〖说明〗参数 memblock 指向已分配内存块的起始位置。如果 memblock 为 NULL，则 realloc 函数的作用与 malloc 函数相同，分配一块 size 字节的空间。如果 memblock 为非 NULL，则应是以前调用 calloc、malloc 或 realloc 函数返回的值。

参数 size 给定了内存块的新字节数。尽管新的存储块可能在不同的位置，但从头开始到新旧尺寸中较短长度为止的存储块中的内容不改变。

参数 memblock 也可以指向已经释放的存储块，只要释放该存储块之后尚未调用过 calloc、_expand、malloc 或 realloc 等函数分配空间即可。如果 realloc 函数调用成功，则重新分配的块被标记为已占有。

在大型数据模式(压缩、大型程序)中，realloc 函数映射为_frealloc 函数。在小数据模式(中、小型程序)中，realloc 函数映射为_nrealloc 函数。

各 realloc 函数按下面所示重新分配不同堆中的内存块。

函数	堆
realloc	根据程序的数据规模决定。

_brealloc	seg 指定的基堆。
_frealloc	远程堆(默认数据段之外)。
_nrealloc	近程堆(默认数据段内)。

realloc 函数返回指向重新分配的存储块的指针(可能会发生移动)。

如果 size 为 0 且缓冲参数为非 NULL，或没有足够的空间对内存块按给定的大小进行扩展时，函数返回 NULL。第 2 种情况下，不改变原先的存储块。

返回指针指向的存储空间可适合存储任何类型的对象，对返回值进行类型转换可获得指向某种特定类型的指针。

〖参见〗calloc,free,malloc。

_freect

〖功能〗该函数可获得有关在近程堆中可供动态分配的内存空间信息。

〖原型〗unsigned _freect(size_t size)

〖位置〗malloc.h

〖说明〗该函数通过返回程序大约可以调用_nmalloc 函数(在小型数据模式中为 malloc 函数)，在近程堆(默认数据段)中分配 size 字节空间的次数来表达这个信息。

_freect 函数返回无符号整型数表示可调用次数。

〖参见〗calloc,_expand,malloc,_memavl,_msize,realloc。

halloc

〖功能〗该函数从操作系统中分配包含 num 个元素的大型数组，每个元素为 size 个字节。

〖原型〗void _huge *halloc(long num, size_t size)

〖位置〗malloc.h

〖说明〗分配后将每个元素初始化为 0。如果数组的大小超过 128 KB(131 072 个字节)，数组元素的大小必须为 2 的幂。

halloc 函数返回指向分配空间的空类型指针，这一空间适合存储任何类型的对象。对函数返回值进行类型转换可令其指向特定的数据类型。如果无法满足需求，则函数返回 NULL。

〖参见〗calloc,_ffree,_fmalloc,free,malloc,_nfree,_nmalloc。

hfree

〖功能〗该函数释放一块存储块，将释放的存储块交还给操作系统。

〖原型〗void hfree(void _huge *memblock)

〖位置〗malloc.h

〖说明〗参数 memblock 指向先前通过调用 halloc 函数分配的存储块，释放的字节数为分配时指定的字节数。

注意，若参数 memblock 非法(不是由 halloc 函数分配的存储块)，则可能会影响以后的

存储分配，并导致错误。

该函数无返回值。

〖参见〗halloc。

_memavl

〖功能〗该函数以字节为单位返回近程堆(默认数据段)中可供动态分配的存储空间的近似大小。

〖原型〗size_t _memavl(void)

〖位置〗malloc.h

〖说明〗在中、小存储式中，_memavl 函数可以和 calloc、malloc 或 realloc 函数一起使用，在任何存储式中可以和_ncalloc，_nmalloc，nrealloc 函数一起使用。

_memavl 函数返回的字节数可能不是连续空间的字节数。因此，用_memavl 函数的返回值调用 malloc 函数分配存储块，可能会失败。可以用_memmax 函数寻找可分配的最大连续空间的大小。

_memavl 函数以无符号整型返回字节数。

〖参见〗calloc,_freect,malloc,_memmax,realloc。

_memmax

〖功能〗该函数返回近程堆(例如，默认数据段)中可供动态分配的最大连续空间字节数。

〖原型〗size_t _memmax(void)

〖位置〗malloc.h

〖说明〗只要_memmax 函数返回非 0 值，则调用_nmalloc(_memmax)一定会成功。

如果函数执行成功，则返回存储块的尺寸；否则函数返回 0，表示近程堆中无可分配空间。

〖参见〗malloc,_msize。

stackavail

〖功能〗该函数返回可供 alloca 函数动态分配的栈空间的近似字节数。

〖原型〗size_t stackavail(void)

〖位置〗malloc.h

〖说明〗stackavail 函数返回无符号整型表示的字节数。

3.15　进程处理函数

abort

〖功能〗该函数向 stderr 输出信息：“abnormal program termination”，然后调用 raise(SIGABRT)。

〖原型〗void abort(void)

〖位置〗process.h，stdlib.h

〖说明〗对 SIGABRT 信号的响应，取决于先前调用 signal 函数时所进行的定义。默认的 SIGABRT 响应为终止程序并以代码 3 退出，然后将控制权交还给父进程或操作系统。

abort 函数不刷新流缓存区，也不执行 atexit 或 onexit 进程。

在多流库中，abort 函数不调用 raise(SIGABRT)，仅终止程序并以代码 3 退出。

abort 函数不将控制权返回给调用者。它结束程序，并且在默认情况下向父进程或操作系统返回退出代码 3。

〖参见〗execl...,execv...,exit,_exit,raise,signal,spawnl...,spawnv…。

assert

〖功能〗assert 程序输出诊断信息，当 expression 为 false(0)时调用 abort 函数。

〖原型〗void assert(int expression)

〖位置〗assert.h，stdio.h

〖说明〗诊断信息的形式如下：

Assertion failed： expression，file filename，line linenumber

其中 filename 为源文件的文件名，line number 为源文件中出现错误行的行号。如果 expression 为 true(非 0)，程序不执行任何操作。

assert 程序(宏)通常用于识别程序的逻辑错误，因此，表达式的值应仅当程序按预期要求执行时才为真。

检查完程序后，可以使用专门的“no debug”标识符 NDEBUG 从程序清除 assert 调用。

如果在命令行用/D 选项或用#define 命令定义 NDEBUG(可以是任何值)，则 C 预处理会从程序中清除所有的 assert 调用。但是对 4 级警告错误，会显示这样的信息：“Statement has no effect”和“Unreferenced formal parameters”。

〖参见〗abort,raise,signal。

atexit、onexit

〖功能〗当程序正常终止时，调用 atexit 和 onexit 函数传递函数 func 的地址。

〖原型〗int atexit(void (*func)(void))

onexit_t onexit(onexit_t func)

〖位置〗stdlib.h

〖说明〗连续调用这些函数可创建一个执行函数的“后进先出”的注册表。最多可注册 32 个函数；如果函数的数量超过 32 个，则返回 NULL。所传递的函数不带参数。

atexit 函数同 ANSI 标准一致，如果希望对 ANSI 可移植，则应使用 atexit 函数代替 onexit 函数。

在 OS/2 环境下，atexit 函数调用 OS/2 函数 DosExitList。在多流动态链接库中，所有传递到 atexit 或 onexit 函数中的程序都应具备_loadds 属性。

atexit 函数若执行成功则返回 0，出现错误时，返回一个非 0 值。(例如，当已经定义了 32 个退出函数时。)

onexit 函数成功时，返回指向函数的指针，如果没有空间存储函数指针，则返回 NULL。

〖参见〗abort,exit。

_beginthread

〖功能〗该函数在 stackaddress 位置上创建开始执行远端程序的线程。

〖原型〗int _beginthread(void (_far *threadfunc)(void _far *)，void _far *stackaddress，unsigned stacksize，void _far *arglist)

〖位置〗process.h，stddef.h

〖说明〗当线程从远端程序返回时，会自动终止；也可以调用_endthread 函数终止线程。仅当使用多线程库 LLIBCMT.LIB， LLIBCDLL.LIB 和 CDLLOBJS.LIB 时，才能使用_beginthread 和_endthread 函数。可以使用/MT 选项访问多线程库。

参数 stackaddress 给出了线程栈的地址。如果 stackaddress 为 NULL，运行时库会根据需要分配和回收线程栈。由于_beginthread 函数了解所有线程 ID 的当前状态，因此它能够根据线程的需要释放旧栈和分配新栈。

如果参数 stackaddress 的值非 NULL，则应指定一个字地址，参数 stacksize 指定了栈的长度，它必须是非 0 的偶数，所分配的栈空间至少应有 stacksize 长。通常，这块存储空间是全局数组或由 malloc 或_fmalloc 函数分配的空间。

编写一个从子线程进行 C 运行时调用的多线程程序时，应保证分配足够大的栈空间。例如，C 函数 printf 要求大于 500 字节的栈空间。为安全起见，应至少为线程栈分配 2 048 字节的空间。(如果子线程不进行运行时调用，则栈空间通常不会有问题。)

通常的规则是，调用任何 API(应用程序界面)程序时(如 OS/2 系统调用)，应具有 2 K 字节的自由栈空间。

arglist 是传递到最新创建的线程中表示远端指针大小的形参。通常，它是传递到新线程中的数据项(如字符串)的地址。如果不需要这个参数时，arglist 也可为 NULL，但是应该为_beginthread 函数提供传递到子线程中的数值。

如果任何一个线程调用了 abort、exit、_exit 或 DosExit，则所有线程都终止。关于多线程编程好的经验是令第 1 个线程为主线程，直到其他线程都终止后主线程才退出程序。

不能直接调用 OS/2 函数 DosCreatThreat 创建线程。_beginthread 函数执行安全地调用其他 C 运行时库函数所需的初始化工作。

当函数成功时，返回新线程的线程标识号。若返回-1 则表示出现错误，同时将 errno 设为 EINVAL 或 EAGAIN。

〖参见〗_endthread, _threadid。

_cexit、_c_exit

〖功能〗_cexit 和_c_exit 函数清除操作，并且不终止进程返回调用者。

〖原型〗void _cexit(void)
　　void _c_exit(void)

〖位置〗process.h

〖说明〗_cexit 函数首先按照“后进先出”的顺序调用 atexit 和 onexit 注册的函数，然

后在返回之前，刷新所有缓存区，关闭所有打开的文件。

_c_exit 函数仅返回调用进程，而不处理 atexit 或 onexit 函数，也不刷新流缓存区。

下面是 exit、_exit、_cexit 和_c_exit 函数的执行过程。

函数	执行过程
exit	执行完整的 C 库终止过程，终止进程，按照提供的状态码退出。
_exit	执行“快速”C 库终止过程，终止进程，按照提供的状态码退出。
_cexit	执行完整的 C 库终止过程，返回调用者但不终止进程。
_c_exit	执行“快速”C 库终止过程，返回调用者但不终止进程。

这两个函数均无返回值。

〖参见〗abort,atexit,execl...,execv...,exit,_exit,onexit,spawnl...,spawnv...,system。

cwait、wait

〖功能〗挂起调用的进程。

〖原型〗int wait(int *termstat)

int cwait (int *termstat, int procid, int action)

〖位置〗process.h，errno.h

〖说明〗wait 函数将调用进程挂起，直到调用者的当前子进程结束为止。如果调用者的所有子进程都在调用 wait 函数之前结束，则 wait 函数立即返回。cwait 函数直到指定子进程结束才终止等待。

当参数 termstat 不为 NULL 时，应指向包含子进程的终止状态字和返回码的缓存区。状态字表示子进程是否通过调用 OS/2 的 DosExit 函数正常终止。若不需要子进程终止状态字，则可令 termstat 为 NULL。

如果子进程正常终止，终止状态字的高字节和低字节如下所示。

字节	内容
低字节	0
高字节	由子进程传递给 DosExit 的结果码的低字节。若子进程调用 exit 或_exit、从 main 函数返回或已经到了 main 的结尾时，调用 DosExit。结果码的低字节可以是_exit 或 exit 的参数的低字节、 main 函数返回值的低字节或者当子进程已经到达 main 程序结尾时为随机值。

注意，OS/2 的 DosExit 函数运行程序返回 16 位结果码。但是 wait 和 cwait 函数只返回结果码的低字节。

如果子进程不调用 DosExit 而终止，则终止状态字的低字节和高字节如下所示。

字节	内容
低字节	来自 DosWait 的终止码：

	代码	含义
	1	硬件错误终止。
	2	中断操作。
	3	未截获到 SIGTERM 信号。
高字节	0	

cwait 函数的参数 procid 指定了等待终止的子进程，该参数的值是调用 spawn 函数开始子进程的返回值。如果指定的进程在调用 cwait 之前已经结束，则 cwait 函数立即返回。

cwait 函数的参数 action 指定了重新开始执行的父进程，其值为常量 WAIT_CHILD 或 WAIT_GRANDCHILD(在 PROCESS.H 中定义)。

当子进程正常结束时，wait 和 cwait 函数返回子进程 ID。当子进程非正常结束时，wait 和 cwait 函数返回-1 并将 errno 设为 EINTR。

另外，当 wait 立即返回-1 并将 errno 设为 ECHILD 时，表示调用进程不存在子进程。这种情况下，cwait 函数返回-1 并将 errno 设为 EINVAL 或 ECHILD。

〖参见〗exit,spawnl...,spawnv...。

_endthread

〖功能〗该函数终止由_beginthread 函数创建的线程。

〖原型〗void _endthread(void)

〖位置〗process.h

〖说明〗仅当使用多线程库 LLIBCMT.LIB，LLIBBCDLL.LIB 和 CDLLOBJS.LIB 时，才可以使用_beginthread 和_endthread 函数。

由于线程执行完后可以自动终止，因此这个函数并不是必须的。它可用来有条件地终止线程。

不能直接使用 OS/2 函数 DosExit 终止由 C 运行时库函数_beginthread 创建的线程；否则，结果将无法预测。

该函数无返回值。

〖参见〗_beginthread,_threadid。

execl、execle、execlp、execlpe

〖功能〗exec 系列函数加载并执行新的子进程。

〖原型〗int execl(char *cmdname，char *arg0，...char *argn，NULL)
int execle(char *cmdname，char *arg0，...char *argn，NULL，char **envp)
int execlp(char *cmdname，char *arg0，... char *argn，NULL)
int execlpe(char *cmdname，char *arg0，...char *argn，NULL，char **envp)

〖位置〗process.h，errno.h

〖说明〗当在 DOS 下成功调用子进程时，会提前将子进程加载到调用进程所占据的内存空间中。另外，必须为加载和执行子进程提供足够的空间。

exec 家族中的所有函数使用相同的 exec 函数头，跟在 exec 函数名后的字母按下列规

则表示执行变化情况。

字母	变化
p	使用 PATH 环境变量寻找执行文件。
l	单独将命令行参数传递到 exec 函数。
v	以指针数组的形式将命令行参数传递到 exec 函数。
e	将指向环境参数的指针数组显式地传递到子进程。

参数 cmdname 指定了作为子进程执行的文件名，可以是全路径(从根目录开始)，部分路径(从当前工作目录开始)或单独的文件名。

如果 cmdname 没有扩展名或不是以句点结尾(.)，exec 函数按给定的名称查找文件。如果查找失败，则会首先查找具有扩展名.COM 的文件，再查找具有扩展名.EXE 的文件。如果 cmdname 有扩展名，在查找过程中仅使用指定的扩展名。如果 cmdname 以句点结尾，则 exec 函数仅查找无扩展名的文件。

execlp、execlpe、execvp 和 execvpe 程序在 PATH 环境变量指定的目录中查找 cmdname(过程同上)。如果 cmdname 包含盘符或斜杠(例如，相对路径名)，exec 仅对指定文件进行查找，而不再查找路径。

调用 exec 时，用一个或多个指向字符串的指针为参数传递到新的进程中，这些字符串组成了子进程的参数列表。组成新进程参数列表的字符串总长度不能超过 128 个字节(在实模式下)，每个字符串的结束空字符不计入，但计入空格字符(自动插入以区分各个参数)。

可以单独传递参数指针(execl、execle、execlp 和 execlpe)或以指针数组的形式传递参数(execv、execve、execvp 和 execvpe)，但至少要向子进程传递一个参数 arg0(也可为 argv[0])。通常，这个参数是参数 cmdname 的副本 (若不一致时，不会产生错误)。

在 DOS 3.0 以前的版本中，子进程中不能使用 arg0 传递的值。但是在 OS/2 和 DOS 3.0 以后的版本中，可以用 arg0 获得 cmdname 的值。

execl、execle、execlp 和 execlpe 函数通常用在事先知道参数个数的情况下。arg0 通常是指向 cmdname 的指针，arg1 到 argn 指向组成新参数列表的字符串，且必须在 argn 后跟一个 NULL 指针表示参数列表的结束。

当新进程的参数是变长参数时，execv、execve、execvp 和 execvpe 函数非常有用。用数组 argv 传递指向参数的指针。argv[0]通常是指向 cmdname 的指针。argv[1]到 argv[n]指向组成新参数列表的字符串，参数 argv[n+1]必须为 NULL 以标志参数列表的结束。

调用 exec 时已经打开的文件在新进程中仍保持打开。调用 execl、execlp、execv 和 execvp 时，子进程继承了父进程的环境，调用 execle、execlpe、execve 和 execvpe 时，可以通过 envp 参数传递环境设置而修改子进程的环境。参数 envp 是字符指针数组，每个元素(除了最后一个之外)都指向一个定义环境变量的字符串，字符串的形式如下：

NAME=value

其中 NAME 为环境变量名，value 是设置变量的字符串值(注意 value 没有用双引号括起来)。envp 的最后一个元素应为 NULL，当 envp 本身为 NULL 时，子进程继承父进程的环境设置。

用某个 exec 函数执行的程序总是被加载到内存中，就好像程序.EXE 文件头中的“最大分配”空间被设为默认值 0FFFFH 一样。如果使用 EXEMOD 改变程序的最大分配空间，则当用某个 exec 函数调用程序时，其结果可能和直接从操作系统命令行调用或用某个 spawn 函数调用的结果不同。

exec 函数调用不保存打开文件的转换模式。如果子进程必须要使用从父进程继承的文件，必须调用 setmode 程序设置这些文件的转换模式。

调用 exec 函数之前，必须手工刷新(调用 fflush 或 flushall 函数)或关闭所有流。

调用 exec 程序创建的子进程不保存信号设置，在子进程中将信号设置还原为默认值。

由于 DOS 2.1 和 2.0 版本的差别，exec 家族函数(或用 P_OVERLAY 等价调用 spawn 函数)产生的子进程可能会导致致命系统错误。若要使用这些函数请更新操作系统到 DOS 3.0 以上版本。

bound 程序在实模式下不能使用 exec 家族的函数。

exec 函数不能正常返回调用进程。如果 exec 函数返回，则表示出现了错误，返回值为 -1。同时将 errno 设为 E2BIG，EACCES，EMFILE，ENOENT，ENOEXEC 或 ENOMEM。

〖参见〗abort,atexit,execv...,exit,_exit,onexit,spawnl...,spawnv...,system。

execv

〖原型〗int execv(char *cmdname，char **argv)；

〖位置〗process.h，errno.h

〖说明〗参阅 execl 函数的说明。

〖参见〗abort,atexit,execl...,exit,_exit,onexit,spawnl...,spawnv...,system。

execve

〖原型〗int execve(char *cmdname，char **argv，char **envp)；

〖位置〗process.h，errno.h

〖说明〗参阅 execl 函数的说明。

〖参见〗abort,atexit,execl...,exit,_exit,onexit,spawnl...,spawnv...,system。

execvp

〖原型〗int execvp(char *cmdname，char **argv)；

〖位置〗process.h，errno.h

〖说明〗参阅 execl 函数的说明。

〖参见〗abort,atexit,execl...,exit,_exit,onexit,spawnl...,spawnv...,system。

execvpe

〖原型〗int execvpe(char *cmdname，char **argv，char **envp)；

〖位置〗process.h，errno.h

〖说明〗参阅 execl 函数的说明。

〖参见〗abort,atexit,execl...,exit,_exit,onexit,spawnl...,spawnv...,system。

exit、_exit

〖功能〗终止调用进程。

〖原型〗void exit(int status)
　　void _exit(int status)

〖位置〗process.h，stdlib.h

〖说明〗exit 函数首先以“后进先出”的顺序调用 atexit 和 onexit 注册的函数，然后在终止进程之前刷新所有文件缓存区。

status 的值通常设为 0，表示正常退出，为其他值时表示错误。

尽管 exit 和_exit 函数不返回值，但等待的父进程可以在调用进程退出后，获得 status 的低字节，使用操作系统批处理命令 ERRORLEVEL 可以获得 status 的值。

下面是 exit、_exit、_cexit 和_c_exit 函数的执行过程。

函数	执行过程
exit	执行完整的 C 库终止过程，终止进程，按照提供的状态码退出。
_exit	执行“快速”C 库终止过程，终止进程，按照提供的状态码退出。
_cexit	执行完整的 C 库终止过程，返回调用者但不终止进程。
_c_exit	执行“快速”C 库终止过程，返回调用者但不终止进程。

〖参见〗abort,atexit,execl...,execv...,_cexit,_c_exit,onexit,spawnl...,spawnv...,system。

getpid

〖功能〗该函数返回进程 ID，即惟一识别调用进程的整数。

〖原型〗int getpid(void)

〖位置〗process.h

〖说明〗该函数无出错返回值。

〖参见〗mktemp。

longjmp

〖功能〗该函数恢复先前使用 setjmp 函数存储在 env 中的栈环境和执行现场。

〖原型〗void longjmp(jmp_buf env，int value)

〖位置〗setjmp.h

〖说明〗longjmp 和 setjmp 函数通常用于向错误处理程序传递控制权或恢复以前调用的进程中的代码，且不使用正常的调用和返回机制。

调用 setjmp 函数时，将当前栈环境存储到 env 中。随后调用 longjmp 函数恢复存储的环境并将控制权交还给 setjmp 函数调用之后的点。执行过程认为 value 的值是 setjmp 调用的返回值。

调用 longjmp 函数时，接收控制权的程序可以获得包含它们拥有变量在内的全部变量(除了注册表变量之外)的值。注册表变量的值不可预测。

调用了 setjmp 的函数返回之前，必须调用 longjmp 函数。如果调用 setjmp 的函数返回之后才调用 longjmp，则程序会出现不可预知的结果。

作为 longjmp 函数参数 value 的 setjmp 函数的返回值，必须为非 0。如果传递的参数 value 为 0，则在实际返回时用 1 代替。

使用 longjmp 函数时需要注意下面的条件。

1. 不要认为注册表变量的值会保持不变。调用 setjmp 函数的程序中，执行完 longjmp 后，注册表变量的值可能无法恢复到以前的值。

2. 不要使用 longjmp 函数将控制权从一个覆盖段内传递到另一个覆盖段内，覆盖段管理器在调用 longjmp 后会保持内存中的覆盖段。

3. 同样，不要使用 longjmp 函数将控制权从中断处理程序中传递出来，除非中断是由于浮点异常引起的。在这种情况下，如果程序调用_fpreset 函数对浮点 math 包进行重新初始化，可能会借助 longjmp 函数从中断处理程序中返回。

该函数无返回值。

〖参见〗setjmp。

_pclose

〖功能〗_pclose 函数等待先前_popen 函数调用开始的子进程终止，然后关闭子进程中与标准输入输出(同_popen 调用中定义的一样)相关联的流。

〖原型〗int _pclose(FILE *stream)

〖位置〗stdio.h

〖说明〗_pclose 函数返回子进程的退出状态。如果子进程正常终止，则返回状态字的低字节和高字节分别如下。

字节	内容
高字节	0
低字节	子进程传递到 DosExit 中的状态码的低字节。如果子进程调用了 exit 或_exit，从 main 中返回或到达了 main 函数的结尾，则调用 DosExit 函数。结果代码的低字节可以是 _exit 或 exit 参数的低字节、返回值的低字节或当子进程到达 main 结尾时为随机值。

注意，OS/2 的 DosExit 函数运行程序返回 16 位结果代码，但是 wait 和 cwait 函数只返回结果码的低字节。

如果子进程没有调用 DosExit 而终止，则终止状态字的低字节和高字节如下。

字节	内容	
高字节	来自 DosWait 的终止代码：	
	代码	含义
	1	硬件错误终止。
	2	陷阱操作。
	3	未截获到 SIGTERM 信号。

低字节　　0

该函数成功返回时子命令的退出状态，否则返回-1。

〖参见〗cwait,_popen。

perror

〖功能〗perror 函数向 stderr 输出错误信息。

〖原型〗void perror(char *string)

〖位置〗stdio.h

〖说明〗该函数首先输出参数 string 的值，和后跟一个冒号，产生错误的最后一次库函数调用的系统错误信息，以及一个换行符。如果参数 string 为 NULL 或指向空串的指针，则 perror 函数仅输出系统错误信息。

实际错误号存储在变量 errno 中(在 ERRNO.H 中定义)，通过按错误号排列的信息数组变量 sys_errlist 可以访问系统错误信息。perror 函数用 errno 作为 sys_errlist 的下标输出适当的错误信息，变量 sys_nerr 的值定义了 sys_errlist 数组元素的最大数量。

为了产生准确的结果，应在库程序返回错误后立即调用 perror 函数。否则，后续的函数调用可能覆盖 errno 的值。

在 DOS 和 OS/2 下，列在 ERRNO.H 中的 errno 的有些值未使用，对这些值 perrno 函数输出空串。

该函数无返回值。

〖参见〗clearerr,errno,ferror,strerror,sys_errlist,sys_nerr。

_pipe

〖功能〗该函数用来创建可用来向其他程序传递信息的人工类文件 I/O 通道——管道。

〖原型〗int _pipe(int *phandles，unsigned psize，int textmode)

〖位置〗fcntl.h，errno.h，io.h

〖说明〗管道与文件类似，也有文件指针或文件描述符，或者两种均有，可以使用标准输入输出库函数进行读写。

与文件不同的是，管道不代表特定的文件或设备。管道代表与程序所占有内存独立的临时内存空间，完全由操作系统控制。

使用管道可以在程序间传递信息。例如，执行下面的命令时，OS/2 中的指令处理器将创建管道：

```
PROGRAM1 | PROGRAM2
```

PROGRAM1 的标准输出句柄连接到管道的写句柄，PROGRAM2 的标准输入句柄连接到管道的读句柄。这样将无需创建向其他程序传递信息的临时文件。

_pipe 函数同 open 函数类似，同时以读写方式打开管道，并返回两个而不是一个文件句柄。

这个函数通常用在和子进程通信的预处理中打开管道。例如，父进程打开管道并将数

据传递到写句柄，子进程可以通过读句柄接收数据。子进程必须能够以某种方式获得父进程打开的句柄，通常是作为参数传递或将其存储到共享内存区域中。如果父进程和子进程都要读写数据，最好打开两组句柄，而不要对同一组句柄进行读写调整。

_pipe 函数打开一个管道并通过参数 phandles 返回两个管道句柄。元素 phandles[0]中是读句柄，phandles[1]中是写句柄。管道文件句柄的用法同其他文件句柄类似(低端读写函数可以对管道进行读写操作)。

参数 psize 指定了管道所需的缓存区大小。无论缓存区大小怎样，管道都能够正常工作。如果写入管道的速度超过读出数据的速度，将阻塞写进程，直到有数据读出为止。缓存区的尺寸越大，发生阻塞的概率越小。

参数 textmode 指定管道的转换模式。常量 O_TEXT 表示文本转换，O_BINARY 表示二进制转换。如果 textmode 为 0，表示_pipe 函数使用默认模式变量_fmode 给定的默认转换模式。

在多线程程序中，不执行死锁。返回的句柄都是最近打开的，并且直到_pipe 调用完成之后，才会被线程引用。

在 OS/2 下，关闭管道的所有句柄后管道会被撤销。(如果关闭了管道的全部读句柄，那么向管道中写数据会出错。)所有对管道进行的读写操作都会等到具备完成 I/O 要求的足够数据或缓存区空间时才能进行。可以用低端关闭函数来关闭管道句柄。

_pipe 函数成功执行时返回 0，出错时返回-1 并将 errno 设为 EMFILE 或 ENFILE。

〖参见〗cwait,_pclose,_popen。

_popen

〖功能〗该函数使用指定命令字符串异步执行指令处理器的子进程副本。

〖原型〗FILE *_popen(char *command，char *mode)

〖位置〗stdio.h

〖说明〗该函数同时为子进程进行标准输入或输出打开一个管道，也就是说，_popen 函数和联合使用 system 和_pipe 函数的效果类似。父进程使用 system 函数将输入或输出重定向到文件时，执行效率会稍微低一些。

参数 command 是指定的命令或程序和子进程 CMD.EXE 要处理的参数字符串，参数 mode 以如下方式指定访问类型。

类型	描述
r	调用进程可以借助返回流读取子进程命令的标准输出。
w	调用进程可以借助返回流向子进程命令的标准输入写入。
b	以二进制模式打开。
t	以文本模式打开。

mode 字符串中应包含“r”或“w”之一。如果出现冲突字符(如“rw”或“wr”)，则只使用第 1 个字符。

对管道数据处理完毕后(通常是 CMD.EXE 终止时)，应使用_pclose 函数关闭管道。

关于 OS/2 中的管道，参阅_pipe 的说明。

_popen 函数返回和创建管道一端关联的流，管道的另一端与子进程命令标准输入或输出关联。如果出现错误，则该函数返回 NULL。

〖参见〗_pclose,_pipe。

raise

〖功能〗该函数向正在执行的程序传递 sig 信号。

〖原型〗int raise(int sig)

〖位置〗signal.h

〖说明〗如果已调用 singal 函数为 sig 安装类信号处理程序，则 raise 函数将调用该程序。如果没有安装处理程序，则执行下述默认操作。

信号值 sig 应是下列常量之一。

信号	含义	默认操作
SIGABRT	非正常终止。	以退出码 3 终止调用程序。
SIGBREAK	CTRL+BREAK 中断。	以退出码 3 终止调用程序。
SIGFPE	浮点错误。	终止调用程序。
SIGILL	非法指令。DOS 和 OS/2。	终止调用程序。
	不产生这个信号，但 ANSI 兼容支持这个信号。	
SIGINT	CTRL+C 中断。	发出 INT 23H 中断。
SIGSEGV	非法存储访问。DOS 和 OS/2 不产生这个信号，	终止调用程序。
	但 ANSI 兼容支持这个信号。	
SIGTERM	向程序传递终止要求。	忽略这个信号。
SIGUSR1	用户自定义信号。	忽略这个信号。
SIGUSR2	同上。	
SIGUSR3	同上。	

该函数成功执行，则返回 0；否则返回一个非 0 值。

〖参见〗abort,signal。

setjmp

〖功能〗该函数存储当前栈环境，以便以后可以使用 longjmp 函数恢复该环境。

〖原型〗int setjmp(jmp_buf env)

〖位置〗setjmp.h

〖说明〗联合使用 setjmp 和 longjmp 函数可以提供一种执行非局部转移的方式，通常用于向错误处理程序传递控制权或恢复以前调用的进程中的代码。

调用 setjmp 函数将当前栈环境存储到 env 中，其后调用 longjmp 函数将恢复保存的环境并将控制权交还到调用 setjmp 函数之后的点。调用 setjmp 函数时，接收控制权的程序可以获得包含它们拥有变量在内的全部变量(除了注册表变量之外)的值，但注册表变量的值

不可预测。

setjmp 函数保存栈环境后，返回 0。如果将 setjmp 函数的返回值作为 longjmp 调用的结果，则 setjmp 函数返回 longjmp 函数的参数值。如果 longjmp 的参数为 0，则 setjmp 返回 1。

该函数无出错返回值。

〖参见〗longjmp。

signal

〖功能〗signal 函数允许进程选择一种处理来自操作系统中断信号的方式。

〖原型〗void (*signal(int sig，void(*func) (int sig [，int subcode])))(int sig)

〖位置〗signal.h

〖说明〗参数 sig 必须是下列常量(定义在 SIGNAL.H 中)之一。

SIGABRT	SIGILL	SIGTERM	SIGUSR2
SIGBREAK	SIGINT	SIGUSR1	SIGUSR3
SIGFPE	SIGSEGV		

SIGUSR1、SIGUSR2、SIGUSR3 是可以借助 DosFlagProcess 传递的用户自定义信号。

注意，在 DOS 下不产生 SIGILL、 SIGSEGV 和 SIGTERM 信号，在 OS/2 下不产生 SIGSEGV 信号。它们是为了 ANSI 兼容而存在的。这样，就可以调用 signal 函数为这些信号设置处理器，也可以调用 raise 函数人为产生这些信号。

注意，调用 exec 或 spawn 函数创建的子进程中不保存这些信号设置，在子进程中，信号设置复原为默认值。

接收到中断信号后的响应取决于参数 func 的值，func 的值必须为函数地址或定义在 SIGNAL.H 中的下列常量。

常量值	含义
SIG_ACK	承认接收到信号(仅在 OS/2 下)。 仅当安装了用户自定义信号处理器时，这个常量才是合法的。进程接收到给定的信号后，直到操作系统接收到来自进程的 SIG_ACK 应答，才继续传送该类信号。操作系统不将这类信号保存到队列中。因此，如果在进程返回 SIG_ACK 值之前，积聚了多个该类信号，操作系统接收到 SIG_ACK 值后，仅将最近的信号传递到进程中。这个选项对安装了处理器的信号不起作用。常量 SIG_ACK 不支持 SIGFPE 信号。
SIG_DFL	使用系统默认响应。 除了 SIGUSR1、SIGUSR2 和 SIGUSR3 之外，所有信号的系统默认响应为中止调用程序。调用进程以退出代码 3 终止，将控制权交还给 DOS 或 OS/2。如果调用程序使用了 I/O 流，则不刷新运行时库创建的缓存区，但刷新操作系统创建的缓存区。 SIGUSR1、SIGUSR2 和 SIGUSR3 的默认响应是忽略信号。
SIG_ERR	忽略中断信号(仅 OS/2)。

除了传递该信号的进程接收到错误之外，这个信号常量与 SIG_IGN 等价。进程可以使用 raise 函数向自身传递信号，其他进程也可以通过 DosFlagProcess 函数(信号是 SIGUSR1、SIGUSR2 和 SIGUSR3 时)或 DosKillProcess(信号是 SIGTERM 时)传递信号。

SIG_IGN　忽略中断信号。由于没有定义进程的浮点状态，所以对信号 SIGFPE 不能指定这个值。

函数地址　将指定函数作为给定信号的处理器。

对除了 SIGFPE 和 SIGUSRx 之外的所有信号，将 SIGINT 作为函数参数 sig 的值，并执行函数。

对 SIGFPE 信号来说，向函数传递两个参数值：SIGFPE 和标识异常类型的浮点错误码。

对 SIGUSRx 信号来说，向函数传递两个参数值：信号数和 DosFlagProcess 函数提供的参数。

对 SIGFPE，向 func 指向的函数传递两个参数值：SIGFPE 和整型错误子代码 FPE_xxx；执行该函数。接收到信号后，不复位参数 func 的值，可联合使用 setjmp 和 longjmp 函数从浮点异常中恢复。若函数返回，调用进程以进程的浮点状态恢复继续执行。

如果函数返回，调用进程立即从接收到中断信号之后的点开始继续执行。无论是何种类型的信号和操作模式，都遵循这个原则。

在 DOS 3.x 或更低版本中，执行指定函数之前，将 func 的值设为 SIG_DFL。若没有指定其他信号，则按照上述 SIG_DFL 方式处理下一个中断信号。这样用户就可以根据需要复位被调用函数中的信号。

在 OS/2 下，不将信号处理器复位为系统默认响应，相反，直到进程向操作系统发出 SIG_ACK 值后，进程才会接收给定类型的信号。用户可以通过先向操作系统传递 SIG_DFL，再传递 SIG_ACK，恢复来自处理器的系统默认响应。

由于发生中断时，通常是异常调用信号处理程序，所以，当操作在未知状态下尚未完成时，信号处理函数也可能获得控制权。信号处理程序中可以使用的函数有一定的约束，下面是有关规则。

1. 不能调用低端或 STDIO.H 的 I/O 程序(如 printf，fread 等)。

2. 不能调用 heap 程序或任何使用 heap 程序的程序(如 malloc，freect，strdup，putenv 等)。

3. 不要使用任何产生系统调用的 C 函数(如 getcwd，time 等)。

4. 除非浮点异常导致的中断(例如，sig 为 SIGFPE)，否则不要使用 longjmp 函数。浮点异常时，程序应首先调用_fpreset 重新初始化浮点包。

5. 不要使用任何 overlay 程序。

signal 函数返回以前与给定信号关联的 func 的值。例如，如果 func 的前一个值是 SIG_IGN，则函数的返回值为 SIG_IGN。但有一个例外值是 SIG_ACK，这时，函数返回当前安装的处理器地址。

返回值为-1 表示出现错误，同时将 errno 设为 EINVAL。可能导致错误的原因有：非法的 sig 值，非法的 func 值(即尚未定义的小于 SIG_ACK 的值)，或 func 的值为 SIG_ACK，

但当前未安装处理器。

〖参见〗abort,execl...,execv...,exit,_exit,_fpreset,spawnl...,spawnv…。

spawnl、spawnle、spawnlp、spawnlpe

〖功能〗spawn 系列函数创建并执行新的子进程。

〖原型〗

int spawnl(int mode，char *cmdname，char *arg0，char *arg1，...char *argn，NULL)

int spawnle(int mode，char *cmdname，char *arg0，char *arg1，...char *argn，NULL，char **envp)

int spawnlp(int mode，char *cmdname，char *arg0，char *arg1，...char *argn，NULL)

int spawnlpe(int mode，char *cmdname，char *arg0，char *arg1，...char *argn，NULL，char **envp)

〖位置〗process.h，stdio.h，errno.h

〖说明〗要加载并执行子进程必须有足够的内存空间。参数 mode 确定了父进程在 spawn 之前和期间进行的操作，下面是定义在 PROCESS.H 中的 mode 的取值。

P_DETACH	P_OVERLAY
P_NOWAIT	P_WAIT
P_NOWAITO	

参数 cmdname 指定了作为子进程执行的文件。cmdname 可以是全路径(从根目录)、部分路径(从当前工作目录)或单独的文件名。若 cmdname 没有文件扩展名，或不是以句点结尾，则 spawn 首先查找扩展名为.COM 的匹配文件，然后是.EXE 文件，最后是.BAT 文件(在 OS/2 保护模式下是.CMD 文件)。spawn 批处理文件的这种功能是 6.0 版本的新特征。

如果 cmdname 有扩展名，则仅使用与其匹配的文件。如果 cmdname 以句点结尾，则 spawn 会使用不带扩展名的匹配文件。spawnlp、spawnlpe、spawnvp 和 spawnvpe 程序在环境变量 PATH 指定的目录中查找 cmdname 文件(过程同上)。

如果 cmdname 中包含驱动器符或斜杠(即相对路径)，则 spawn 只查找路径指定文件。

调用 spawn 时，通过一个或多个指向字符串的指针向子进程传递参数。这些字符串组成了子进程的参数列表。在实模式下，组成子进程参数列表的字符串总长度不能超过 128 字节。每个字符串的结束空字符不计入，但要计算空格(自动插入以分隔字符串)。

可以以独立参数的形式传递指针参数(spawnl、spawnle、spawnlp 和 spawnlpe 中)，也可以以指针数组的形式传递(spawnv、spawnve、spawnvp 和 spawnvpe 中)。至少要向子进程传递一个参数 arg0 或 argv[0]。根据约定，这个参数是用户在命令行输入的程序名。(若输入不同的值也不会导致错误。)在实模式中，argv[0]的值由 DOS 提供，是完全有效的执行程序的路径名。在包含模式中，这个参数通常是在命令行输入的程序名。

spawnl、spawnle、spawnlp 和 spawnlpe 函数通常用在已知参数数量的情况下。arg0 通常是指向 cmdname 的指针，参数 arg1 到 argn 是指向组成新参数列表字符串的指针。在 argn 的后面必须有一个 NULL 指针，表示参数列表结束。

spawnv、spawnve、spawnvp 和 spawnvpe 函数通常用在子进程的参数数量可变的情况下，以数组 argv 的形式传递指向参数的指针。argv[0]是指针，在实模式下指向路径名，在保护模式下指向程序名，argv[1]到 argv[n]是指向组成新参数列表的字符串的指针。argv[n+1]必须为 NULL，表示参数列表结束。

调用 spawn 时打开的文件在新进程中仍保持打开。调用 spawnl、spawnlp、spawnv 和 spawnvp 时，子进程继承了父进程的环境。调用 spawnle、spawnlpe、spawnve 和 spawnvpe 时，可以通过 envp 参数传递环境设置而修改子进程的环境。参数 envp 是字符指针数组，每个元素(除了最后一个之外)都指向一个定义环境变量的字符串。字符串的形式如下：

```
NAME=value
```

其中 NAME 为环境变量名，value 是设置变量的字符串值(注意，value 没有用双引号括起来)。envp 的最后一个元素应为 NULL。当 envp 为 NULL 时，子进程继承父进程的环境设置。

在实模式下，spawn 函数通过环境中的 C_FILE_INFO 项向子进程传递包括转换模式在内的打开文件的有关信息(保护模式下为_C_FILE_INFO)。

通常 C 启动代码处理此项，然后从环境中将其删除。但是如果 spawn 产生了一个非 C 进程(如 CMD.EXE)，则仍在环境中保存这项。由于在实模式下，传递的环境信息是二进制形式，因此输出定义此项的字符串以图形字符显示。这对正常的操作不会产生任何其他影响。在保护模式下，以文本形式传递环境信息，因此不会包含图形字符。

调用 spawn 函数前，必须人工刷新或关闭所有流(使用 fflush 或 flushall 函数)。

从 Microsoft C 6.0 版本开始，可以使用变量_fileinfo 控制是否向子进程传递进程的打开文件信息。有关细节参阅_fileinfo。

在 OS/2 和 DOS 兼容模式下，用 FAPI 双模式约束的程序中不能使用 spawn 函数的 P_OVERLAY 模式。

与 DOS 模式的.EXE 文件链接的程序和保护模式的程序工作方式相同，规则只约束实模式的程序。

为了确保正确进行重叠初始化和终止，请不要使用 longjmp 和 setjmp 函数进入或离开 overlay 程序。

同步 spawn(mode 的值为 P_WAIT)函数的返回值是子进程的退出状态。

异步 spawn(mode 的值为 P_NOWAIT 或 P_NOWAITO)函数的返回值是进程 ID。必须调用 wait 或 cwait 函数并指定进程 ID，才能获得 P_NOWAIT 模式下产生进程的退出代码。无法获得用 P_NOWAITO 模式产生的进程退出代码。

进程正常终止时退出状态为 0。如果子进程用非 0 参数调用了 exit，则将退出状态设为非 0 值。若子进程没有设置正的退出代码，则退出代码为正数，表示由于中止或中断而非正常退出。返回值为-1 表示出现错误(子进程没有开始)，这种情况下，将 errno 设为 E2BIG，EINVAL，ENOENT，ENOEXEC 或 ENOMEN。

〖参见〗abort,atexit,execl...,execv...,exit,_exit,onexit,spawnv...,system。

spawnv、spawnve、spawnvp、spawnvpe

〖原型〗int spawnv(int mode，char *cmdname，char **argv)
　　int spawnve(int mode，char *cmdname，char **argv，char **envp)
　　int spawnvp(int mode，char *cmdname，char **argv)
　　int spawnvpe(int mode，char *cmdname，char **argv，char **envp)
〖位置〗process.h，stdio.h，errno.h
〖说明〗参阅 spawn 的说明。
〖参见〗abort,atexit,execl...,execv...,exit,_exit,onexit,spawnl...,system。

system

〖功能〗system 函数向指令解释器传递参数 command，指令解释器将该字符串翻译成操作系统命令。
〖原型〗int system(char *command)
〖位置〗process.h，stdlib.h，errno.h
〖说明〗在 OS/2 下，同步执行指令。system 函数借助环境变量 COMSPEC 和 PATH 确定命令解释器文件(DOS 为 COMMAND.COM，OS/2 为 CMD.EXE)的位置。若参数 command 是指向 NULL 串的指针，则函数仅简单地检查指令解释器是否存在。

当 command 为 NULL，且找到指令解释器文件时，函数返回一个非 0 值。若未找到指令解释器文件，函数返回 0 并将 errno 设为 ENOENT。若 command 不为 NULL，当 system 函数成功启动命令解释器时，函数返回 0。在 OS/2 下，system 函数从指令解释器返回退出状态。

函数返回值为-1 表示出现错误，同时将 errno 设为 E2BIG，ENOENT，ENOEXEC 或 ENOMEM。
〖参见〗exit,_exit,execl...,execv...,spawnl...,spawnv...。

3.16　字符串处理函数

strcat、_fstrcat

〖原型〗char *strcat(char *string1, char *string2)
　　char _far *_fstrcat(char _far *string1，char _far *string2)
〖位置〗string.h
〖说明〗strcat 函数对以空字符结尾的字符串进行操作，函数的字符串参数包含一个空字符(\0)表示结尾。对字符串进行复制或连接操作时，不做越界检查。

strcat 函数将字符串 string2 连接到字符串 string1 后，结果字符串以空字符结尾，并返回指向连接后字符串 string1 的指针。

函数的_f...形式是独立模式(大型模式)形式，使用远程字符串指针形式的参数和返回值。

〖参见〗strncat,strncmp,strncpy,strnicmp,strrchr,strspn。

strchr、_fstrchr

〖原型〗char *strchr(char *string，int c)
　　char _far *_fstrchr(char _far *string，int c)

〖位置〗string.h

〖说明〗strchr 函数对以空字符结尾的字符串进行操作，函数的字符串参数包含一个空字符(\0)表示结尾。对字符串进行复制或连接操作时，不做越界检查。

strchr 函数返回指向字符 c 在 string 中首次出现位置的指针，字符 c 可以是空字符(\0)，查找范围包括 string 的结尾空字符。如果没有找到，则函数返回 NULL。

函数的_f...形式是独立模式(大型模式)形式，使用远程字符串指针形式的参数和返回值。

〖参见〗strcspn,strncat,strncmp,strncpy,strnicmp,strpbrk,strrchr,strspn,strstr。

strcmp、_fstrcmp

〖原型〗int strcmp(char *string1，char *string2)
　　int _fstrcmp(char _far *string1，char _far *string2)

〖位置〗string.h

〖说明〗strcmp 函数对以空字符结尾的字符串进行操作，函数的字符串参数包含一个空字符(\0)表示结尾。

strcmpi 和 stricmp 函数是 strcmp 函数忽略大小写的版本。strcmpi 函数是 stricmp 函数的旧同义版本，二者互相兼容。

strcmp、strcmpi 和 stricmp 函数比较 string1 和 string2，根据比较结果返回如下值。

返回值	含义
<0	string1 小于 string2。
=0	string1 和 string2 相同。
>0	string1 大于 string2。

函数的_f...形式是独立模式(大型模式)形式，使用远程字符串指针形式的参数和返回值。可以在程序的任何位置调用这些独立模式函数。注意，strcmpi 函数没有独立模式版本，若需要使用独立模式形式时，应该使用_fstricmp 函数。

〖参见〗memcmp,memicmp,strcmpi,strncat,strncmp,strncpy,strnicmp,strrchr,strspn。

strcpy、_fstrcpy

〖原型〗char *strcpy(char *string1，char *string2)
　　char _far *_fstrcpy(char _far *string1，char _far *string2)

〖位置〗string.h

〖说明〗strcpy 函数对以空字符结尾的字符串进行操作，函数的字符串参数包含一个

空字符(\0)表示结尾。复制字符串时不做越界检查。

strcpy 函数将字符串 string2(包括结尾空字符在内)复制到 string1 指定的位置，并返回 string1。

函数的_f...形式是独立模式(大型模式)形式，使用远程字符串指针形式的参数和返回值。可以在程序的任何位置调用这些独立模式函数。

〖参见〗strcat,strcmp,strncat,strncmp,strncpy,strnicmp,strrchr,strspn。

strcspn、_fstrcspn

〖原型〗size_t strcspn(char *string1，char *string2)
　　size_t _fstrcspn(char _far *string1，char _far *string2)

〖位置〗string.h

〖说明〗strcspn 函数对以空字符结尾的字符串进行操作，函数的字符串参数包含一个空字符(\0)表示结尾。复制或连接字符串时不做越界检查。

strcspn 函数返回 string2 字符串中的字符在 string1 中首次出现的下标，这个值等于 srting1 的完全由非 string2 中字符组成的第 1 个子串的长度。查找范围不包括结尾空字符。如果 string1 以 string2 中的字符开头，则 strcspn 返回 0。

函数的_f...形式是独立模式(大型模式)形式，使用远程字符串指针形式的参数和返回值。可以在程序的任何位置调用这些独立模式函数。

〖参见〗strncat,strncmp,strncpy,strnicmp,strrchr,strspn。

strdup、_fstrdup、nstrdup

〖原型〗char *strdup(char *string)
　　char _far *_fstrdup(char _far *string)
　　char _near *_nstrdup(char _far *string)

〖位置〗string.h

〖说明〗这组 strdup 函数对以空字符结尾的字符串进行操作，函数的字符串参数包含一个空字符(\0)表示结尾。

strdup 函数为 string 的副本分配空间(调用 malloc 函数)，并返回指向包含副本字符串的存储空间的指针。如果无法分配空间，则返回 NULL。

_fstrdup 和_nstrdup 函数提供字符串副本使用的堆的完全控制权，普通 strdup 函数返回指向 string 副本的指针。字符串空间是从当前使用的内存模式所指定的堆中分配的。在大型数据模式中(压缩或大型程序)，strdup 从远程堆分配空间；在小型数据模式中(中、小型程序)，strdup 从近程堆分配空间。

_fstrdup 函数返回指向从远程堆分配的字符串副本空间的指针，_nstrdup 函数从近程堆为副本分配空间。

函数的_f...形式是独立模式(大型模式)形式，使用远程字符串指针形式的参数和返回值。可以在程序的任何位置调用这些独立模式函数。

〖参见〗strcat,strcmp,strncat,strncmp,strncpy,strnicmp,strrchr,strspn。

stricmp、_fstricmp、_strcmpi

〖原型〗int stricmp(char *string1，char *string2)
int _fstricmp(char _far *string1，char _far *string2)
int strcmpi(char *string1，char *string2)

〖位置〗string.h

〖说明〗strcmpi 和 stricmp 函数是 strcmp 函数的忽略大小写版本，strcmpi 函数是 stricmp 函数的旧同义版本，二者互相兼容。参阅_fstrcmp 函数的说明。

〖参见〗memcmp,memicmp,strcat,strcpy,strncat,strncmp,strncpy,strnicmp,strrchr,strset,strspn。

strlen、_fstrlen

〖原型〗size_t strlen(char *string)
size_t _fstrlen(char _far *string)

〖位置〗string.h

〖说明〗strlen 函数返回参数 string 以字节为单位的长度，不包括结尾空字符(\0)。

函数的_fstrlen 形式是独立模式(大型模式)形式，使用远程字符串指针形式的参数和返回值。可以在程序的任何位置调用这些独立模式函数。

strlen 函数无出错返回值。

strlwr、strupr、_fstrlwr、fstrupr

〖原型〗char *strlwr(char *string)
char *strupr(char *string)
char _far *_fstrlwr(char _far *string)
char _far *_fstrupr(char _far *string)

〖位置〗string.h

〖说明〗strlwr 函数将给定字符串中的所有大写字母转换成小写。strupr 函数将小写字母转换成大写，其他字符保持不变。

函数的_f...形式是独立模式(大型模式)形式，使用远程字符串指针形式的参数和返回值。可以在程序的任何位置调用这些独立模式函数。

这些函数返回指向转换后字符串的指针，无出错返回值。

strncat、_fstrncat

〖原型〗char *strncat(char *string1，char *string2，size_t count)
char _far *_fstrncat(char _far *string1，char _far *string2，size_t count)

〖位置〗string.h

〖说明〗strncat 函数对以空字符结尾的字符串进行操作，函数的字符串参数包含一个空字符(\0)表示结尾。对字符串进行复制或连接操作时，不做越界检查。

strncat 函数最多将 string2 的前 count 个字符连接到 string1 上，结果字符串以空字符结尾，并返回指向连接后字符串 string1 的指针。如果 count 大于 string2 的长度，则用 string2

的长度代替 count 做连接操作。

函数的_f...形式是独立模式(大型模式)形式，使用远程字符串指针形式的参数和返回值。

〖参见〗strcat,strcmp,strcpy,strncmp,strncpy,strnicmp,strrchr,strset,strspn。

strncmp、_fstrncmp

〖原型〗int strncmp(char *string1，char *string2，size_t count)
　　int _fstrncmp(char _far *string1，char _far *string2，size_t count)

〖位置〗string.h

〖说明〗strncmp、strnicmp 函数对以空字符结尾的字符串进行操作，函数的字符串参数包含一个空字符(\0)表示结尾。

strnicmp 函数是 strncmp 的忽略大小写版本。

strncmp 和 strnicmp 函数至多对字符串的前 count 个字符进行比较。strncmp 和 strnicmp 函数最多比较 string1 和 string2 的前 count 个字符，然后根据子串的比较结果返回如下值之一。

返回值	含义
<0	substring1 小于 substring2。
=0	substring1 和 substring2 相同。
>0	substring1 大于 substring2。

函数的_f...形式是独立模式(大型模式)形式，使用远程字符串指针形式的参数和返回值。可以在程序的任何位置调用这些独立模式函数。

〖参见〗strcat,strcpy,strncat,strncpy,strrchr,strset,strspn。

strncpy、_fstrncpy

〖原型〗char *strncpy(char *string1，char *string2，size_t count)
　　char _far *_fstrncpy(char _far *string1，char _far *string2，size_t count)

〖位置〗string.h

〖说明〗strncpy 函数对以空字符结尾的字符串进行操作，函数的字符串参数包含一个空字符(\0)表示结尾。复制字符串时不做越界检查。

strncpy 函数将 string2 的 count 个字符复制到 string1 中，并返回 string1。如果 count 小于 string2 的长度，则不会自动在复制后的字符串结尾添加空字符(\0)。如果 count 大于 string2 的长度，则将结果字符串 string1 用空字符(\0)填充到 count 个字符位置。当 string2 和 string1 所占空间重叠时，未定义 strncpy 函数的相应操作。

函数的_f...形式是独立模式(大型模式)形式，使用远程字符串指针形式的参数和返回值。可以在程序的任何位置调用这些独立模式函数。

〖参见〗strcat,strcmp,strncat,strncmp,strnicmp,strrchr,strspn。

strnicmp、_fstrnicmp

〖原型〗int strnicmp(char *string1，char *string2，size_t count)
int _fstrnicmp(char _far *string1，char _far *string2，size_t count)

〖位置〗string.h

〖说明〗strnicmp 函数是 strncmp 函数的忽略大小写版本，参阅 strncmp 函数的说明。

〖参见〗strcat,strcpy,strncat,strncpy,strrchr,strset,strspn。

strnset、_fstrnset

〖原型〗char *strnset(char *string, int c，size_t count)
char _far *_fstrnset(char _far *string，int c，size_t count)

〖位置〗string.h

〖说明〗strnset 函数将字符串 string 的最多前 count 个字符设为 c，然后返回指向转换后的字符串的指针。如果 count 大于 string 的长度，则在操作时，用 string 的长度代替 count。

函数的_f...形式是独立模式(大型模式)形式，使用远程字符串指针形式的参数和返回值。可以在程序的任何位置调用这些独立模式函数。

这两个函数无出错返回值。

〖参见〗strcat,strcmp,strcpy,strset。

strpbrk、_fstrpbrk

〖原型〗char *strpbrk(char *string1，char *string2)
char _far *_fstrpbrk(char _far *string1，char _far *string2)

〖位置〗string.h

〖说明〗strpbrk 函数在 string1 中寻找 string2 中的字符首次出现的位置，结尾空字符(\0)不在查找范围内。

函数的_fstrpbrk 形式是独立模式(大型模式)形式，使用远程字符串指针形式的参数和返回值。可以在程序的任何位置调用这些独立模式函数。

strpbrk 和_fstrpbrk 函数返回指向 string2 中的字符在 string1 中首次出现位置的指针。若返回 NULL 指针，则表示 string1 和 string2 中没有相同的字符。

〖参见〗strchr,strrchr。

strrchr、_fstrrchr

〖原型〗char *strrchr(char *string，int c)
char _far *_fstrrchr(char _far *string，int c)

〖位置〗string.h

〖说明〗strrchr 函数对以空字符结尾的字符串进行操作，函数的字符串参数包含一个空字符(\0)表示结尾。对字符串进行复制或连接操作时，不做越界检查。

strrchr 函数在 string 中寻找字符 c 最后一次出现位置，查找范围包括 string 的结尾空字符。strrchr 函数返回指向 c 在 string 中最后一次出现位置的指针。如果没有找到，则函数

返回 NULL。

函数的_f...形式是独立模式(大型模式)形式，使用远程字符串指针形式的参数和返回值。

〖参见〗strchr,strcspn,strncat,strncmp,strncpy,strnicmp,strpbrk,strspn。

strrev、_fstrrev

〖原型〗char *strrev(char *string)
　　char _far *_fstrrev(char _far *string)

〖位置〗string.h

〖说明〗strrev 函数翻转 string 中所有字符的顺序，结尾空字符位置不变。

函数的_fstrrev 形式是独立模式(大型模式)形式，使用远程字符串指针形式的参数和返回值。可以在程序的任何位置调用这些独立模式函数。

strrev 和_fstrrev 函数返回指向转换后的字符串的指针，无出错返回值。

〖参见〗strcpy,strset。

strset、_fstrset

〖原型〗char *strset(char *string，int c)
　　char _far *_fstrset(char _far *string，int c)

〖位置〗string.h

〖说明〗strset 函数将 string 中除了结尾空字符之外的所有字符都设为字符 c。

函数的_f...形式是独立模式(大型模式)形式，使用远程字符串指针形式的参数和返回值。可以在程序的任何位置调用这些独立模式函数。

这两个函数返回指向转换后字符串的指针，无出错返回值。

〖参见〗memset,strcat,strcmp,strcpy,strnset。

strspn、_fstrspn

〖原型〗size_t strspn(char *string1，char *string2)
　　size_t _fstrspn(char _far *string1，char _far *string2)

〖位置〗string.h

〖说明〗strspn 函数对以空字符结尾的字符串进行操作，函数的字符串参数包含一个空字符(\0)表示结尾。复制或连接字符串时不做越界检查。

strspn 函数返回 string1 中第 1 个不属于 string2 字符串的字符的下标，这个值等于 srting1 中完全由 string2 中字符组成的第 1 个子串的长度。匹配的范围不包括 string2 的结尾空字符。如果 string1 以不包含在 string2 中的字符开头，则 strspn 返回 0。

函数的_f...形式是独立模式(大型模式)形式，使用远程字符串指针形式的参数和返回值。可以在程序的任何位置调用这些独立模式函数。

〖参见〗strcspn,strncat,strncmp,strncpy,strnicmp,strrchr。

strstr、_fstrstr

〖原型〗char *strstr(char *string1，char *string2)
　　char _far *_fstrstr(char _far *string1，char _far *string2)

〖位置〗string.h

〖说明〗strstr 函数返回指向字符串 string2 在 string1 中首次出现位置的指针。如果 string2 没有在 string1 中出现，则函数返回 NULL。

函数的_f...形式是独立模式(大型模式)形式，使用远程字符串指针形式的参数和返回值。可以在程序的任何位置调用这些独立模式函数。

〖参见〗strcspn,strncat,strncmp,strncpy,strnicmp,strpbrk,strrchr,strspn。

strtok、_fstrtok

〖原型〗char *strtok(char *string1，char *string2)
　　char _far *_fstrtok(char _far *string1，char _far *string2)

〖位置〗string.h

〖说明〗strtok 函数将 srting1 作为一系列记号(0 个或多个)读入，将 string2 作为 string1 中记号的分隔字符读入。string1 中的记号可以由 string2 中的一个或多个分隔符分隔。

通过一系列的 strtok 调用，可以将记号从 string1 中分离出来。对 string1 首次调用 strtok 时，strtok 跳过开头分隔符，在 string1 中查找第 1 个记号，返回指向第 1 个记号的指针。

令参数 string1 的值为 NULL，调用 strtok，可以从 string1 中读取下一个记号。string1 为 NULL 时，strtok 会查找前面的记号字符串中的下一个记号。每次调用函数时，分隔符可以变化，因此 string2 可以为任何值。

注意，由于每次调用 strtok 函数都会在 string1 的记号后插入空字符(\0)，所以调用 strtok 函数会改变 string1 的值。

函数的_fstrtok 形式是独立模式(大型模式)形式，使用远程字符串指针形式的参数和返回值。可以在程序的任何位置调用这些独立模式函数。

第 1 次调用 strtok 函数，返回指向 string1 中第 1 个记号的指针。以后对同一个记号字符串调用 strtok 时，返回指向下一个记号的指针。如果没有记号，则返回 NULL。所有的记号都以空字符(\0)结尾。

〖参见〗strcspn,strspn。

strerror、_strerror

〖原型〗char *strerror(int errnum)
　　char *_strerror(char *string)

〖位置〗string.h

〖说明〗strerror 函数将参数 errnum 映射为一个错误信息字符串，并返回指向该字符串的指针。函数本身不能输出信息，因此，需要调用输出函数(如 printf)实现。

如果向参数 string 传递的是 NULL，_strerror 函数返回指向字符串的指针，该字符串包含最后一次出现错误的库函数调用的系统错误信息。错误信息字符串以换行字符(\n)结束。

若参数 string 不为 NULL，_strerror 函数返回指向字符串的指针，该字符串按顺序包括：自定义字符串、冒号、空格、最后一次出现错误的库函数调用的系统错误信息和一个换行符。自定义字符串不能超过 94 个字节。

与 perror 函数不同，单独使用_strerror 函数不能输出任何信息。必须使用下面的 fprintf 语句，才能将_strerror 返回的信息输出到 stderr：

```
if( ( access( "datafile", 2 ) ) == -1 )
   fprintf( _strerror( NULL ) );
```

_strerror 函数的实际错误号存储在 STDLIB.H 中定义的变量 errno 中。借助变量 sys_errlist 可以访问系统错误信息，该变量是一个按错误号排列的信息字符串数组。_strerror 函数以变量 errno 为 sys_errlist 的下标访问适当的错误信息。变量 sys_nerr 为数组 sys_errlist 中最大元素个数。

为了保证_strerror 函数结果的准确性，应在库程序返回错误后立即调用_strerror；否则，errno 的值可能被随后的函数调用覆盖。

strerror 函数返回指向错误信息字符串的指针，再次调用 strerror 函数会覆盖该字符串。

〖参见〗clearerr,errno,ferror,perror,sys_errlist,sys_nerr。

3.17　BIOS 中断服务

_bios_disk

〖功能〗_bios_disk 使用 INT 0x13 提供若干磁盘访问功能。

〖原型〗unsigned _bios_disk(unsigned service，struct diskinfo_t *diskinfo)

〖位置〗bios.h

〖说明〗参数 service 用来选择所需的功能，结构体 diskinfo(diskinfo_t 类型结构体)提供了必要的参数。

注意，由于该函数允许的低端操作可以直接处理磁盘，因此使用起来很危险。

service 的取值如下。

_DISK_FORMAT	_DISK_RESET	_DISK_VERIFY
_DISK_READ	_DISK_STATUS	_DISK_WRITE

_bios_disk 函数返回 BIOS 中断存储在寄存器 AX 中的值。

_bios_equiplist

〖功能〗_bios_equiplist 使用 INT 0x11 确定当前所安装的硬件和外围设备。

〖原型〗unsigned _bios_equiplist(void)

〖位置〗bios.h

〖说明〗该函数返回值各个位及其含义如下。

位	含义
0	为真表示安装了磁盘驱动器。
1	协处理器(除 PC 外)。
2~3	以 16 KB 为单位的系统 RAM 数量(16~64 KB)。
4~5	初始视频模式。
6~7	安装软盘驱动器的数量(00=1，01=2 等)。
8	当且仅当安装了 DMA 芯片时，为假(0)。
9~11	RS232 串口的数量。
12	当且仅当安装了游戏适配器时为真(1)。
13	当且仅当安装了内置调制解调器时为真(1)。
14~15	安装打印机的数量。

_bios_keybrd

〖功能〗_bios_keybrd 使用 INT 0x16 提供访问键盘服务。
〖原型〗unsigned _bios_keybrd(unsigned service)
〖位置〗bios.h
〖说明〗参数 service 为下列常量之一。

_KEYBRD_READ	_NKEYBRD_READ
_KEYBRD_READY	_NKEYBRD_READY
_KEYBRD_SHIFTSTATUS	_NKEYBRD_SHIFTSTATUS

当参数为...READ 和...SHIFTSTATUS 类型时，_bios_keybrd 函数返回调用 BIOS 后 AX 寄存器中的内容。

当参数为...READY 类型时，若没有击键，则_bios_keybrd 返回 0；若有击键则返回等待读入的键值(例如，同_KEYBRD_READ 相同的值)。

当参数为...READ 和 ...READY 类型时，若按下 CTRL+BREAK 或下一个待读入按键为 CTRL+BREAK，则返回-1。

_bios_memsize

〖功能〗_bios_memsize 使用 INT 0x12 确定可用主内存的总数。
〖原型〗unsigned _bios_memsize(void)
〖位置〗bios.h
〖说明〗该函数返回以 1 KB 为单位建立的总内存数量，最大值为 640，代表 640 KB 主内存。

_bios_printer

〖功能〗_bios_printer 使用 INT 0x17 为并行打印机执行打印机输出服务。

〖原型〗unsigned _bios_printer(unsigned service，unsigned printer，unsigned data)
〖位置〗bios.h
〖说明〗参数 printer 指定了使用的打印机，为 0 表示 LPT1，为 1 表示 LPT2，等等。有些打印机不完全支持所有信号。因此，在“缺纸”条件下，有可能无法返回程序。参数 service 应为下列常量之一。

_PRINTER_INIT　　_PRINTER_STATUS　　_PRINTER_WRITE

_bios_printer 程序返回 BIOS 中断后寄存器 AX 中的内容。

_bios_serialcom

〖功能〗_bios_serialcom 使用 INT 0x14 提供串行通信服务。
〖原型〗unsigned _bios_serialcom(unsigned service，unsigned serial_port，unsigned data)
〖位置〗bios.h
〖说明〗参数 serial_port 为 0 表示 COM1，为 1 表示 COM2，等等。参数 service 应为下列常量之一。

_COM_INIT　　_COM_RECEIVE　　_COM_SEND　　_COM_STATUS

由于与服务计算机中断相关的系统开销，_bios_serialcom 在传输速率超过 1 200 波特(_COM_1200)时可能无法建立可靠通信。应直接对串口控制器编程以建立高速数据通信。
注意，这个函数只能在 IBM(R)及其兼容个人计算机上使用。
当 service 的值为_COM_RECEIVE 或_COM_STATUS 时，忽略参数 data。当 service 的值为_COM_INIT 时，data 的值为下面常量组成的表达式(直接用按位或操作符连接)。

_COM_CHR7	_COM_CHR8	
_COM_STOP1	_COM_STOP2	
_COM_NOPARITY	_COM_EVENPARITY	_COM_ODDPARITY
_COM_110	_COM_600	_COM_4800
_COM_150	_COM_1200	_COM_9600
_COM_300	_COM_2400	

data 的默认值为 1 表示停止位，无奇偶校验，110 波特。
该函数返回一个高字节为状态位的 16 位整数。低字节根据 service 的值不同而变化。高字节位的含义如下。

位号	置为 1 的含义
15	超时。
14	传送转换寄存器为空。

13	传送保持寄存器为空。
12	检测到 break。
11	帧同步错误。
10	奇偶校验错。
9	越界错误。
8	数据就绪。

当 serivce 为_COM_SEND 时，如果数据无法发送则将第 15 位置 1。

当 service 为_COM_RECEIVE 且调用成功时，用低字节返回读入的字节。若发生错误时，将第 9、10、11 或 15 位置 1。

当 service 为_COM_INIT 或_COM_STATUS 时，低字节定义如下。

位号	置为 1 的含义
7	检测到接收信号。
6	响铃指示器。
5	数据集就绪。
4	清除发送。
3	修改检测到的接收信号。
2	边缘追踪响铃指示器。
1	修改数据集就绪状态。
0	修改清除发送状态。

_bios_timeofday

〖功能〗_bios_timeofday 使用 INT 0x1A 获取或设置时钟计数器。

〖原型〗unsigned _bios_timeofday(unsigned service，long *timeval)

〖位置〗bios.h

〖说明〗参数 service 的值为常量_TIME_GETCLOCK 或_TIME_SETCLOCK。

参数为_TIME_GETCLOCK，若从上次读入开始，已经过了午夜，则返回一个非 0 值，若尚未经过午夜，则返回 0。未定义参数为_TIME_SETCLOCK 时的函数返回值。

3.18 DOS 中断例程

bdos

〖功能〗bdos 函数将 dosdx 和 dosal 指定的值分别赋给 DX 和 AL 寄存器，然后调用 DOS 系统调用 dosfunc。

〖原型〗int bdos(int dosfunc, unsigned dosdx, unsigned dosal)

〖位置〗dos.h

〖说明〗bdos 函数执行 INT 21H 指令调用系统调用。系统调用完成后，bdos 返回 AX

寄存器中的内容。

当不需参数或只需 DX(DH，DL)或 AL 中的参数来调用 DOS 系统调用时，通常使用 bdos 函数。

不要使用 bdos 函数调用修改 DS 寄存器的中断。这时，应使用 intdosx 或 int86x 函数。intdosx 和 int86x 函数从参数 segregs 中加载 DS 和 ES 寄存器，调用结束后将这些寄存器的值存储到 segregs 中。

不要使用 bdos 函数调用设置进位标志表示错误的系统调用。由于 C 程序不能访问进位标志，所以无法确定返回值的状态。这时，可以使用 intdos 函数。

bdos 函数返回系统调用后 AX 寄存器中的内容。

〖参见〗intdos,intdosx。

_chain_intr

〖功能〗_chain_intr 函数将控制权从一个中断处理程序转移到另一个。同时将第 1 个程序的栈和寄存器交给第 2 个程序，第 2 个程序可以像被直接调用一样返回。

〖原型〗void _chain_intr(void (_interrupt _far *target) ( ))

〖位置〗dos.h

〖说明〗_chain_intr 通常用于以用户自定义中断处理程序开始，然后链接到系统中断处理程序，并以此结束的程序中。

链接可使用下列两种技术之一，用于从新中断程序向老中断程序传递控制权。

1. 如果程序已经完成，并希望以第 2 个中断程序中止中断调用时，以_interrupt 程序为参数调用_chain_intr。

```
   void _interrupt _cdecl new_int( unsigned _es，unsigned _ds，unsigned _di，
unsigned _si,... )；
   {
     ++_di；                          // 初始化处理过程
       _chain_intr( old_int )；       // 向 old_int 中传递新的 DI

       --_di；                        // 不执行该语句
   }
```

2. 如果第 2 个中断程序完成后，还需要进行其他处理，调用_interrupt 程序(必要时，要先将其转换为 interrupt 函数)。

```
   void _interrupt _cdecl new_int( unsigned _es，unsigned _ds，unsigned _di，
unsigned _si,... )；
   {
      ++_di；                    // 初始化处理过程
      (*old_int)()；             // 将新 DI 传递到 old_int 中
      _asm mov _di，di           // 将 old_int 中的真实 DI 放入_di，用于返回
   }
```

注意，老的_interrupt 函数设置的真实寄存器不会自动设置新程序的伪寄存器。

若不希望替换默认中断处理程序，但需要看到输入时，可使用_chain_intr 函数。例如，

检查所有键盘输入查找“热键”序列的 TSR()(内存驻留)程序。

只有对声明为_interrupt 类型的 C 函数才能使用_chain_intr 函数。声明为_interrupt 类型可确保过程的进入和退出符合中断处理程序的要求。

〖参见〗_dos_getvect,_dos_keep,_dos_setvect,_interrupt。

_disable

〖功能〗该函数通过执行 8086CLI 机器指令禁止中断。

〖原型〗void _disable(void)

〖位置〗dos.h

〖说明〗修改中断向量之前应使用_disable。

该函数无返回值。

_dos_allocmem

〖功能〗_dos_allocmem 函数分配 size 段长的内存空间(一段为 16 字节)。

〖原型〗unsigned _dos_allocmem(unsigned size，unsigned *seg)

〖位置〗dos.h，errno.h

〖说明〗所分配的存储空间总是整段的。用参数 seg 指向的字返回所分配空间的首段描述符。如果无法按要求分配，则用这个字返回最大可分配的段数。

_dos_allocmem 函数成功执行则返回 0；否则返回 DOS 错误代码，并将 errno 设为 ENOMEM 表示无足够空间或非法实存块(内存区域)头。

〖参见〗alloca,calloc,_dos_freemem,_dos_setblock,halloc,malloc。

_dos_close

〖功能〗该函数使用系统调用 0x3E 关闭参数 handle 表示的文件。

〖原型〗unsigned _dos_close(int handle)

〖位置〗dos.h，errno.h

〖说明〗表示文件的 handle 参数值是创建或最后一次打开文件时的返回值。

函数成功执行，则返回 0；否则返回 DOS 错误代码，并将 errno 设为 EBADF，表示非法文件句柄。

不要使用同控制台、低端或流 I/O 关联的 DOS 接口程序。

〖参见〗creat,_dos_creat,_dos_creatnew,_dos_open,_dos_read,_dos_write,dup,fclose,open。

_dos_creat、_dos_creatnew

〖功能〗_dos_creat 和_dos_creatnew 函数创建并打开名为 filename 的新文件。

〖原型〗unsigned _dos_creat(char *filename，unsigned attrib，int *handle)

unsigned _dos_creatnew(char *filename，unsigned attrib，int *handle)

〖位置〗dos.h，errno.h

〖说明〗新文件的访问属性由参数 attrib 指定。新文件的文件句柄将被复制到 handle 指向的整型空间中，并以读写方式打开。如果安装了文件共享，则文件以兼容模式打开。

_dos_creat 函数使用系统调用 INT 0x3C，_dos_creatnew 函数使用系统调用 INT 0x5B。如果文件已经存在，_dos_creat 会清除其内容，并保持其属性不变。但_dos_creatnew 程序无法创建已经存在的新文件。

若成功执行，则两个函数都返回 0；否则返回 DOS 错误代码，并将 errno 设为 EACCES，EEXIST，EMFILE 或 ENOENT。

dosexterr

〖功能〗dosexterr 函数获取 DOS 系统调用 0x59 返回的扩展错误信息，并将其存储到 errorinfo 指向的结构体中。

〖原型〗int dosexterr(struct DOSERROR *errorinfo)

〖位置〗dos.h

〖说明〗在提供了扩展错误处理的 DOS 3.0 或更高版本中进行系统调用时，这个函数非常有用。

结构体类型 DOSERROR 在 DOS.H 中定义。

若指针参数为 NULL，则该函数返回 AX 寄存器的值，并且不向结构体中填充数据。

dosexterr 函数返回 AX 寄存器中的值(与 exterror 结构体成员中的值相同)。

〖参见〗perror。

_dos_findfirst、_dos_findnext

〖原型〗unsigned _dos_findfirst(char *filename，unsigned attrib，struct find_t *fileinfo)
　　unsigned _dos_findnext(struct find_t *fileinfo)

〖位置〗dos.h，errno.h

〖说明〗_dos_findfirst 函数使用系统调用 INT 0x4E 返回第 1 个文件名和属性与参数 filename 和 attributes 匹配的文件实例的有关信息。这些信息通过定义在 DOS.H 中的 find_t 类型的结构体返回。

参数 filename 可以使用通配符(*和?)。参数 attrib 可以取下列常量。

_A_ARCH	_A_RDONLY	_A_SYSTEM
_A_HIDDEN	_A_SUBDIR	_A_VOLID
_A_NORMAL		

若取多个值，则各值之间可以用按位或运算符(|)连接。

_dos_findnext 函数使用系统调用 INT 0x4F 寻找下一个与前面调用_dos_findfirst 指定的 filename 和 attrib 匹配的文件。参数 fileinfo 指向前面调用_dos_findfirst 初始化的结构体。入前所述，如果找到匹配的文件，则修改结构体的内容。

若参数 attrib 为_A_HIDDEN，_A_RDONLY，_A_SUBDIR 或_A_SYSTEM，则函数还返回与 filename 匹配的正常属性的文件。正常文件为不具有只读、隐藏、系统或目录属性的文件。

在调用_dos_findfirst 和后续的_dos_findnext 调用之间，不要改变缓存区的内容。同时

也不要在两次_dos_findnext 调用之间改变缓存区的内容。

wr_time 和 wr_date 元素的格式为 DOS 格式，任何其他 C 运行时函数都不能使用。

当函数成功执行时，返回 0；否则返回 DOS 错误代码，并将 errno 设为 ENOENT，表示没有与 filename 匹配的文件。

_dos_freemem

〖功能〗_dos_freemem 函数使用系统调用 INT 0x49 释放由_dos_allocmem 函数分配的内存空间。

〖原型〗unsigned _dos_freemem(unsigned seg)

〖位置〗dos.h，errno.h

〖说明〗参数 seg 的值应为前面调用_dos_allocmem 的返回值，应用程序无法再使用释放的空间。

如果函数执行成功，则返回 0；否则返回 DOS 错误代码，并将 errno 设为 ENOMEM，表示段值非法(不是由前面调用_dos_allocmem 返回的段)或非法内存块头。

〖参见〗_dos_allocmem,_dos_setblock,_ffree,free,hfree,_nfree。

_dos_getdate

〖功能〗_dos_getdate 函数使用系统调用 0x2A 获得当前系统日期。

〖原型〗void _dos_getdate(struct dosdate_t *date)

〖位置〗dos.h

〖说明〗通过定义在 DOS.H 中的 dosdate_t 类型结构体返回日期。

该函数无返回值。

〖参见〗_dos_gettime,_dos_setdate,_dos_settime,gmtime,localtime,mktime,_strdate,_strtime,time。

_dos_getdiskfree

〖功能〗_dos_getdiskfree 函数使用系统调用 0x36 获取由 drive 指定的磁盘驱动器信息。

〖原型〗unsigned _dos_getdiskfree(unsigned drive，struct diskfree_t *diskspace)

〖位置〗dos.h，errno.h

〖说明〗默认驱动器为 0，A 驱动器为 1，B 驱动器为 2，依此类推。该函数通过参数 diskspace 指向的定义在 DOS.H 中的 diskfree_t 类型结构体返回这些信息。

如果函数执行成功，则返回 0；否则返回一个非 0 值，并将 errno 设为 EINVAL，表示指定的是非法驱动器。

〖参见〗_dos_getdrive,_dos_setdrive。

_dos_getdrive

〖功能〗_dos_getdrive 函数使用系统调用 0x19 获得当前磁盘驱动器。

〖原型〗void _dos_getdrive(unsigned *drive)

〖位置〗dos.h

〖说明〗通过参数 drive 指向的字返回当前驱动器：1=A 驱动器，2=B 驱动器，依此类推。

该函数无返回值。

〖参见〗_dos_getdiskfree,_dos_setdrive,_getdrive。

_dos_getfileattr

〖功能〗_dos_getfileattr 函数使用系统调用 0x43 获取 pathname 指向的文件或目录的当前属性。

〖原型〗unsigned _dos_getfileattr(char *pathname, unsigned *attrib)

〖位置〗dos.h，errno.h

〖说明〗将该属性复制到 attrib 的低字节，各种属性用下面的常量表示。

_A_ARCH	_A_NORMAL	_A_SYSTEM
_A_HIDDEN	_A_RDONLY	

如果函数执行成功则返回 0；否则返回 DOS 错误代码，并将 errno 设为 ENOENT，表示无法找到目标文件或目录。

〖参见〗access,chmod,_dos_setfileattr,umask。

_dos_getftime

〖功能〗dos_getftime 函数使用系统调用 0x57 获取对 handle 标识的文件最后一次进行写操作的日期和时间。

〖原型〗unsigned _dos_getftime(int handle, unsigned *date，unsigned *time)

〖位置〗dos.h，errno.h

〖说明〗在调用_dos_getftime 之前，必须先调用_dos_open 或_dos_creat 打开该文件。通过 date 和 time 指向的字单元返回日期和时间。

如果函数成功则返回 0；否则返回 DOS 错误代码，并将 errno 设为 EBADF，标识传递了非法文件句柄。

〖参见〗_dos_setftime。

_dos_gettime

〖功能〗_dos_gettime 函数使用系统调用 0x2C 获得当前系统时间。

〖原型〗void _dos_gettime(struct dostime_t *time)

〖位置〗dos.h

〖说明〗该函数通过定义在 DOS.H 中的 dostime_t 类型结构体返回该时间。

该函数无返回值。

〖参见〗_dos_getdate,_dos_setdate,_dos_settime。

_dos_getvect、_dos_setvect

〖功能〗_dos_getvect 函数使用系统调用 0x35 获得 intnum 指定的中断向量的当前值。

〖原型〗void (_interrupt _far *_dos_getvect(unsigned intnum)) ( )

void _dos_setvect(unsigned intnum，void (_interrupt _far *handler) ( ))

〖位置〗dos.h

〖说明〗_dos_setvect 函数通过系统调用 0x25 将当前中断向量 intnum 设为 handler 指向的函数。以后，只要产生 intnum 中断，就会调用 handler 程序。

如果 handler 是 C 函数，则必须先将其声明为_interrupt 属性；否则，必须要确保该函数能够满足中断处理程序的要求。

如果 handler 是汇编函数，则必须是用 IRET 而不是 RET 返回的远端程序。

_dos_setvect 函数通常和_dos_getvect 函数一起使用。替换中断向量之前，首先应使用_dos_getvect 保存当前中断向量，然后再用_dos_setvect 设置自己的中断程序。如果必要，以后还可使用_dos_setvect 恢复保存起来的中断向量。用户自定义的程序也可能需要原来的中断向量，以便调用或用_chain_intr 链接。

_interrupt 属性表示函数是中断处理程序。编译器会为中断处理程序产生适当的进入和退出方式，包括保存所有寄存器和执行 IRET 指令返回。

使用寄存器和中断函数时应注意如下一些问题。

调用中断函数时，将 DS 寄存器初始化为 C 数据段，这样就可以在中断函数中访问全局变量了。

另外，除 SS 之外的所有寄存器都保存在栈中。如果在函数形参列表中为每个寄存器声明了正式参数，则可以在函数中访问这些寄存器。下面是一个声明实例。

```
void _interrupt _cdecl _far int_handler ( unsigned _es,
            unsigned _ds, unsigned _di, unsigned _si,
            unsigned _bp, unsigned _sp, unsigned _bx,
            unsigned _dx, unsigned _cx, unsigned _ax,
            unsigned _ip, unsigned _cs, unsigned _flags )
{
……
}
```

参数的顺序必须与其入栈的顺序相反。只能够从参数列表尾部开始忽略参数，而不能从参数列表头开始忽略。例如，如果中断处理程序只需要使用 DI 和 SI，则 ES 和 ES 也必须提供，但不需要提供 BX 或 DX。

如果是通过 C 直接调用而不是通过 INT 指令调用中断处理程序，则可以越过附加的参数。这种情况下，必须声明所有的寄存器参数，然后将函数的参数定义在参数列表的尾部。

编译器总是按照固定的顺序保存和恢复寄存器。这样，无论在参数列表中使用的是什么名称，第 1 个参数总是表示 ES，第 2 个表示 DS，等等。如果中断处理程序使用内嵌汇编程序，应该区别参数名，以便它们和真实寄存器名不重复。

如果在中断函数执行期间修改了寄存器参数，则当函数返回时，相应的寄存器保留修

改后的值。例如：

```
void _interrupt _cdecl _far int_handler( unsigned _es,
                                         unsigned _ds,
                                         unsigned _di,
                                         unsigned _si )
{
   _di = -1;
}
```

当 handler 函数返回时，该代码令 DI 寄存器的值为-1。在中断函数中修改代表 IP 和 CS 寄存器的参数是不明智的。如果需要修改细节标志(如特定 DOS 和 BIOS 中断程序的进位标志)，可使用按位或操作符(|)，这样将不修改标志寄存器中的其他位。

使用 INT 指令调用中断函数时，清除中断标志。如果中断函数中还需要处理中断，则使用_enable 函数设置中断标志，以打开中断。

使用中断函数应预防的问题。

由于 DOS 不能够重载(不能够在 DOS 中断内部调用 DOS 中断)，在中断函数内部调用任何调用了 DOS INT 21H 的库函数通常都是不安全的。

对很多 BIOS 函数也应注意同样的问题：依赖包括 I/O 函数和_dos 家族函数在内的 INT 21H 调用的函数，依赖包括图形函数和_bios 家族函数在内的机器 BIOS 的函数。使用如字符串处理函数等，不依赖 INT 21H 或 BIOS 的函数通常是安全的，在中断函数中使用库函数之前，应了解这些库函数是如何实现的。

_dos_getvect 函数返回指向中断 intnum 当前处理程序的远程指针，_dos_setvect 函数无返回值。

〖参见〗_chain_intr,_dos_keep,_interrupt。

_dos_keep

〖功能〗该函数使用系统调用 INT 0x31 在内存中安装驻留程序(TSR)。

〖原型〗void _dos_keep(unsigned retcode, unsigned memsize)

〖位置〗dos.h

〖说明〗该程序首先退出调用进程，将其留在内存中，然后向调用进程的父进程返回参数 retcode 的低字节，为当前驻留的进程分配 memsize 段(1 段为 16 字节)内存，多余内存返回系统。

_dos_keep 函数和 exit 函数调用同样的内部程序。因此，它还完成下述 3 种操作。

1. 如果定义的话，调用 atexit 和 onexit 函数。

2. 刷新所有文件缓存区。

3. 恢复被 C 启动代码(通常是中断 0，即被 0 除)取代的中断向量。如果使用了仿真算术库并且无协处理器，则恢复 0x34 到 0x3D。如果有协处理器，则恢复中断 2。

_dos_keep 函数不能自动关闭文件，除非想让 TSR 安装代码打开的文件在 TSR 中仍保持打开，否则应执行关闭文件操作。

除非对 C 启动代码和协处理器非常熟悉，否则不要在 TSR 中使用仿真算术库。如果

TSR 必须进行浮点运算，可使用 math 包代替。

不要在程序员 WorkBench 环境下运行使用了_dos_keep 函数的程序，这样做会导致后续的内存问题，_dos_keep 函数会终止在程序员 WorkBench 环境中运行的程序。

该函数无返回值。

〖参见〗_chain_intr,_dos_getvect,_dos_setvect,_interrupt。

_dos_open

〖功能〗该函数程序使用系统调用 0x3d 打开 filename 指向的已存在文件。

〖原型〗unsigned _dos_open(char *filename，unsigned mode，int *handle)

〖位置〗dos.h，errno.h，fcntl.h，share.h

〖说明〗将打开文件的句柄复制到 handle 指向的整型单元中。参数 mode 指定了文件的访问、共享和继承模式，它的值可为下面 3 组常量组成的表达式(用或操作连接)。一次最多只能指定一种访问模式和一种共享模式。

访问模式	共享模式	继承模式
O_RDONLY	SH_COMPAT	O_NOINHERIT
O_WRONLY	SH_DENYRW	
O_RDWR	SH_DENYWR	
	SH_DENYRD	
	SH_DENYNONE	

不要对和控制台、低端或流 I/O 程序有关的 DOS 接口程序使用这个函数。

如果函数执行成功，则返回 0；否则返回 DOS 错误代码，并将 errno 设为 EACCES、EINVAL、 EMFILE 或 ENOENT。

〖参见〗_dos_close,_dos_read,_dos_write。

_dos_read

〖功能〗该函数使用系统调用 0x3F 从 handle 指定的文件中读取 count 个字节数据，并将其复制到 buffer 指向的缓存区。

〖原型〗unsigned _dos_read(int handle，void _far *buffer，unsigned count，unsigned *numread)

〖位置〗dos.h，errno.h

〖说明〗numread 指向的整型变量中存储了实际读入的字节数，这个值有可能小于 count。如果实际读入的字节数为 0，则表明程序从文件结尾开始读入。

不要对和控制台、低端或流 I/O 程序有关的 DOS 接口程序使用这个函数。

如果函数执行成功，则返回 0；否则返回 DOS 错误代码，并将 errno 设为 EACCES 或 EBADF。

〖参见〗_dos_close,_dos_open,_dos_write。

_dos_setblock

〖功能〗该函数使用系统调用 0x4A 将_dos_allocmem 以前分配的 seg 存储空间的大小改为 size 段。

〖原型〗unsigned _dos_setblock(unsigned size，unsigned seg，unsigned *maxsize)

〖位置〗dos.h，errno.h

〖说明〗如果无法满足要求，将最多可用段的大小复制的 maxsize 指向的缓存区中。

如果函数执行成功，则返回 0；若调用失败，则函数返回 DOS 错误代码并将 errno 设为 ENOMEM，表示传递到函数中的段值非法(不是由_dos_allocmem 返回的段值)或包含了非法的内存块头。

〖参见〗_dos_allocmem,_dos_freemem,realloc。

_dos_setdate

〖功能〗该函数使用系统调用 0x2B 设置当前系统日期。

〖原型〗unsigned _dos_setdate(struct dosdate_t *date)

〖位置〗dos.h，errno.h

〖说明〗所设置日期存储在参数 date 指向的 dosdate_t 类型(在 DOS.H 中定义)的结构体中。

函数执行成功，则返回 0；否则返回一个非 0 值，并将 errno 设为 EINVAL，表示指定了非法日期。

〖参见〗_dos_getdate,_dos_gettime,_dos_settime,gmtime,localtime,mktime,_strdate,_strtime,time。

_dos_setdrive

〖功能〗该函数使用系统调用 0x0E 将参数 drive 指定的驱动器设为当前默认驱动器：1=驱动器 A，2=驱动器 B，依此类推。

〖原型〗void _dos_setdrive(unsigned drive，unsigned *numdrives)

〖位置〗dos.h

〖说明〗参数 numdrives 表示系统中驱动器总数。例如，该值为 4，仅表示系统中有 4 个驱动器，不表示这 4 个驱动器为：A，B，C，D。

该函数无返回值。如果传递的驱动器数量非法，则函数失败，但无任何显示。可以使用_dos_getdrive 函数来确认是否设置了预期的驱动器。

〖参见〗_dos_getdiskfree,_dos_getdrive。

_dos_setfileattr

〖功能〗该函数使用系统调用 0x43 设置 pathname 指定的文件或目录属性。

〖原型〗unsigned _dos_setfileattr(char *pathname，unsigned attrib)

〖位置〗dos.h，errno.h

〖说明〗实际属性存储在 attrib 的低字节，属性由下列常量表示。

_A_ARCH　　_A_RDONLY　_A_SYSTEM
_A_HIDDEN　　_A_SUBDIR　_A_VOLID
_A_NORMAL

函数执行成功，则返回 0；否则返回 DOS 错误代码，并将 errno 设为 EACCES 或 ENOENT。

〖参见〗_dos_getfileattr。

_dos_setftime

〖功能〗该函数使用系统调用 0x57 设置对 handle 标识的文件最后一次进行写操作的时间和日期。

〖原型〗unsigned _dos_setftime(int handle，unsigned date，unsigned time)

〖位置〗dos.h，errno.h

〖说明〗时间和日期值使用 DOS 的日期和时间格式。

函数执行成功，则返回 0；否则返回 DOS 错误代码，并将 errno 设为 EBADF，表示文件句柄非法。

〖参见〗_dos_getftime。

_dos_settime

〖功能〗该函数使用系统调用 0x2D 将参数 time 指向的 dostime_t 类型(在 DOS.H 中定义)结构体中的时间值设为当前时间。

〖原型〗unsigned _dos_settime(struct dostime_t *time)

〖位置〗dos.h，errno.h

〖说明〗函数执行成功，则返回 0；否则返回一个非 0 值，并将 errno 设为 EINVAL，表示指定的时间非法。

〖参见〗_dos_getdate,_dos_gettime,_dos_setdate,gmtime,localtime,mktime,_strdate,_strtime。

_dos_write

〖功能〗该函数使用系统调用 0x40 从 buffer 指向的缓存区向 handle 指向的文件中写 count 个字节的数据。

〖原型〗unsigned _dos_write(int handle，void _far *buffer，unsigned count，unsigned *numwrt)

〖位置〗dos.h，errno.h

〖说明〗参数 numwrt 指向的整型单元中存储实际写入的字节数，这个值可能会小于 count。

不要对和控制台、低端或流 I/O 程序连接的 DOS 接口程序使用该函数。

函数成功执行，则返回 0；否则返回 DOS 错误代码，并将 errno 设为 EACCES 或 EBADF。

〖参见〗_dos_close,,_dos_open,_dos_read。

_enable

〖功能〗通过执行 8086STI 机器指令打开中断。
〖原型〗void _enable(void)；
〖位置〗dos.h
〖说明〗该函数无返回值。

FP_OFF

〖功能〗用来设置或获取 address 位置的远程指针的偏移量。
〖原型〗unsigned FP_OFF(void _far *address)；
〖位置〗dos.h

FP_SEG

〖功能〗用来设置或获取 address 位置的远程指针的段地址。
〖原型〗unsigned FP_SEG(void _far *address)；
〖位置〗dos.h

_harderr

〖功能〗处理调用 DOS 中断 0x24 的紧急错误。
〖原型〗void _harderr(void (_far *handler)( ))；
〖位置〗dos.h
〖说明〗_harderr 函数为中断 0x24 建立新的紧急错误处理程序。

_harderr 函数不能直接将_handler 指向的程序设为处理程序，而是建立一个调用_handler 关联函数的处理程序。处理程序可用下面的参数调用指定函数：

```
handler( unsigned deverror，unsigned errcode，unsigned far *devhdr )；
```

参数 deverror 是设备错误代码，包含 DOS 通过 INT 0x24 处理程序传递的 AX 寄存器的值。参数 errcode 是 DOS 传递给处理程序的 DI 寄存器的值，errcode 的低字节可取下列值之一。

代码	含义
0	企图向写保护磁盘写数据。
1	未知单元。
2	驱动器未准备好。
3	未知命令。
4	数据 CRC 校验错。
5	设备结构体长度非法。
6	寻道错误。
7	未知媒介类型。

8	未找到扇区。
9	打印机缺纸。
A	写错误。
B	读错误。
C	一般错误。

参数 devhdr 为指向包含出错设备信息的设备头的远程指针，用户自定义的处理程序不能修改设备头控制块中的信息。

如果磁盘设备发生错误，则 deverror 的最高位(第 15 位)为 0，参数 deverror 含义如下。

位号	含义	
15	若为假(0)表示磁盘错误。	
14	未使用。	
13	若为假(0)表示不允许“Ignore”响应。	
12	若为假(0)表示不允许“Retry”响应。	
11	若为假(0)表示不允许“Fail”响应(注意，DOS 中将“fail”改为了“abort”)。	
9~10	代码	位置
	00	DOS
	01	文件分配表(FAT)
	10	目录
	11	数据区
8	若为假(0)表示读错误；为真(1)表示写错误。	

参数 deverror 的低字节表示发生错误的驱动器(0=驱动器 A，1=驱动器 B，等等)。

如果是除了磁盘设备以外的其他设备出现了错误，则 deverror 的最高位为 1。设备头块中偏移地址为 4 的属性字表明发生错误的设备类型。如果属性字的第 15 位为 0，则表示是 FAT 表内存映像错误。如果第 15 位为 1，则表示字符设备发生错误。属性字的 0~3 位代表如下设备类型。

位号	含义
0	当前标准输入。
1	当前标准输出。
2	当前无效设备。
3	当前时钟设备。

用户自定义的处理函数只能通过 0x0C 或 0x59 发出系统调用 0x01。这样，很多标准 C 运行时函数(如流 I/O 和低端 I/O)都不能用做硬件错误处理程序。使用 0x59 可用于获取关于所发生错误的更详细信息。

可以通过下面 3 种方式从处理程序返回：

- 通过 return 语句返回。
- 通过_hardresume 函数返回。
- 通过_hardretn 函数返回。

如果处理程序通过_hardresume 函数或 return 语句返回，则返回到 DOS。

只应在用户定义的硬件错误处理函数中调用_hardresume 函数，其 result 参数为下列常量之一。

_HARDERR_ABORT	_HARDERR_IGNORE
_HARDERR_FAIL	_HARDERR_RETRY

_hardretn 函数允许用户自定义硬件错误处理程序直接返回到应用程序，而不是返回到 DOS。应用程序恢复到失败 I/O 请求之后，只应在用户自定义的硬件错误处理函数中调用 _hardretn 函数。

_hardretn 函数的参数 error 应为 DOS 错误代码，与 errno 中的 XENIX 风格的错误代码不同。

这些函数无返回值。

〖参见〗_chain_intr,_dos_getvect,_dos_setvect。

_hardresume

〖功能〗处理调用 DOS 中断 0x24 的紧急错误。

〖原型〗void _hardresume(int result)

〖说明〗_hardresume 和_hardretn 函数控制程序如何从_harderr 建立的错误处理程序中返回。_hardresume 函数从用户建立的错误程序返回到 DOS，_hardreturn 函数从直接从用户建立的错误处理程序返回到应用程序。

result 参数的取值为：_HARDERR_ABORT，_HARDERR_FAIL，_HARDERR_IGNORE，_HARDERR_RETRY 之一。

_hardretn

〖功能〗处理调用 DOS 中断 0x24 的紧急错误。

〖原型〗void _hardretn(int error)

〖说明〗_hardresume 和_hardretn 函数控制程序如何从_harderr 建立的错误处理程序中返回。_hardresume 函数从用户建立的错误程序返回到 DOS，_hardreturn 函数从直接从用户建立的错误处理程序返回到应用程序。

int86

〖功能〗该函数执行中断号 intnum 指定的 8086 处理器家族的中断。

〖原型〗int int86(int intnum，union REGS *inregs，union REGS *outregs)

〖位置〗dos.h

〖说明〗执行中断之前，int86 首先将 inregs 中的内容复制到相应的寄存器中。中断返回后，函数将当前寄存器的值复制到 outregs 中。函数同时还将系统进位标志状态复制到 outregs 的 cflag 字段中。

参数 inregs 和 outregs 都是 REGS 类型的共用体，这种共用体类型在 DOS.H 中定义。

不要使用 int86 函数调用修改 DS 寄存器的中断程序，这时可以使用 int86x 代替。int86x 函数从参数 segregs 中加载 DS 和 ES 寄存器，函数调用结束后将 DS 和 ES 寄存器的值存储到 segregs 中。

函数返回中断返回后的 AX 寄存器的值。如果 outregs 中的 cflag 字段不为 0，则说明出现了错误，这时，将变量_doserrno 设为相应的错误代码。

〖参见〗bdos,intdos,intdosx,int86x。

int86x

〖功能〗该函数执行有中断号 intnum 指定的 8086 处理器家族中断程序。

〖原型〗int int86x(int intnum，union REGS *inregs，union REGS *outregs，struct SREGS *segregs)

〖位置〗dos.h

〖说明〗和 int86 不同，int86x 接收 segregs 中的段寄存器值。这样程序就可以使用大型数据段或远程指针指定系统调用时应使用的段或指针。

执行指定的中断程序之前，int86x 函数首先将 inregs 和 segregs 的内容复制到相应的寄存器中。函数只使用 segregs 中的 DS 和 ES 寄存器值。

中断程序返回后，int86x 将当前寄存器的值复制到 outregs 中，并且将当前 ES 和 DS 值复制到 segregs，恢复 DS 寄存器。函数同时还将系统进位标志的状态复制到 outregs 的 cflag 字段中。

参数 inregs 和 outregs 为 REGS 类型的共用体，参数 segregs 为 SREGS 类型的结构体，这些数据类型在 DOS.H 中定义。

可以使用 segread 函数或 FP_SEG 宏获得参数 segregs 中的段值。

函数返回中断返回后的 AX 寄存器的值。如果 outregs 中的 cflag 字段不为 0，则说明出现了错误，这时，将变量_doserrno 设为相应的错误代码。

〖参见〗bdos,FP_SEG,intdos,intdosx,int86,segread。

intdos

〖功能〗该函数调用定义在 inregs 中的寄存器变量指定的 DOS 系统调用。

〖原型〗int intdos(union REGS *inregs，union REGS *outregs)

〖位置〗dos.h

〖说明〗该函数通过 outregs 返回系统调用的结果。参数 inregs 和 outregs 为 REGS 类型(定义在 DOS.H 中)的共用体。

intdos 函数执行 INT 21H 指令，实现系统调用。执行该指令之前，intdos 函数先将 inregs 中的内容复制到相应的寄存器中。INT 指令返回后，intdos 函数将当前寄存器值复制到 outregs 中，同时还将系统进位标志的状态复制到 outregs 的 cflag 字段中。若 cflag 字段为

非 0 值，则表示系统调用设置了进位标志，同时表示出现了错误。

intdos 函数用于调用以除了 DX 或 AL 之外的寄存器为输入输出的 DOS 系统调用，也用于调用通过设置进位标志表示出错的系统调用。在其他条件下，可以使用 bdos 函数。

不要用 intdos 函数调用修改了 DS 寄存器的中断程序，这时，可以用 intdosx 或 int86x 函数代替。

intdos 函数返回系统调用完成后的 AX 寄存器值。如果 outregs 的 cflag 字段非 0，表示出现了错误，并将_doserrno 设为相应的错误代码。

〖参见〗bdos,intdosx。

intdosx

〖功能〗该函数调用定义在 inregs 中的寄存器变量指定的 DOS 系统调用。

〖原型〗int intdosx(union REGS *inregs，union REGS *outregs，struct SREGS *segregs)

〖位置〗dos.h

〖说明〗该函数通过 outregs 返回系统调用的结果，参数 inregs 和 outregs 为 REGS 类型(定义在 DOS.H 中)的共用体。

和 intdos 函数不同，intdosx 函数接收参数 segregs 中的段寄存器值。这样程序就可以使用大型数据段或远程指针指定系统调用时应使用的段或指针。

参数 segregs 为 SREGS 类型的结构体，这些数据类型在 DOS.H 中定义。

intdosx 函数执行 INT 21H 指令，实现系统调用。执行该指令之前，intdosx 函数先将 inregs 和 segregs 中的内容复制到相应的寄存器中。函数仅使用 segregs 中的 DS 和 ES 寄存器值。

INT 指令返回后，intdosx 函数将当前寄存器值复制到 outregs 中并恢复 DS，同时还将系统进位标志的状态复制到 outregs 的 cflag 字段中。若 cflag 字段为非 0 值，则表示系统调用设置了进位标志，同时表示出现了错误。

intdosx 函数用于调用以 ES 寄存器值为参数或将 DS 寄存器的值同数据段不同的系统调用。

可以通过调用 segread 函数或 FP_SEG 宏获取 segregs 中的段值。

intdosx 函数返回系统调用完成后的 AX 寄存器值。如果 outregs 的 cflag 字段非 0，表示出现了错误，并将_doserrno 设为相应的错误代码。

〖参见〗bdos,FP_SEG,intdos,segread。

segread

〖功能〗该函数将当前段寄存器的内容填入到参数 segregs 指向的结构体(SREGS 类型)中。

〖原型〗void segread(struct SREGS *segregs)

〖位置〗dos.h

〖说明〗这个函数通常和 intdosx 和 int86x 配合使用，用来恢复上一次使用时的段寄存器的值。

该函数无返回值。

〖参见〗FP_SEG,intdosx,int86x。

3.19 系统时间处理函数

asctime

〖功能〗该函数将存储在结构体中的时间转换成字符串。

〖原型〗char *asctime(struct tm *timeptr)

〖位置〗time.h

〖说明〗参数 temptr 的值通常通过调用 gmtime 或 localtime 函数获得，这两个函数都返回指向 tm 类型结构体的指针。

asctime 的结果字符串包含 26 个字符，形式如下所示：

```
Wed Jan 02 02:03:55 1980\n\0
```

这里使用的是 24 小时时钟，所有字段的域宽都是固定的，字符串的最后两个字符为换行符(\n)和空字符(\0)。asctime 函数使用一个静态分配的缓存区保存返回字符串，每次调用时都会破坏上一次调用的结果。

asctime 函数返回指向结果字符串的指针，无出错返回值。

〖参见〗ctime,ftime,gmtime,localtime,time,tzset。

clock

〖功能〗该函数获取调用进程使用的处理器时间。

〖原型〗clock_t clock(void)

〖位置〗time.h

〖说明〗将 clock 函数的返回值除以常量 CLOCKS_PER_SEC 可获得时间的近似秒数。

也就是说，colck 函数返回的是处理器流逝的滴答数量。一个滴答约等于 1/CLOCKS_PER_SEC 秒。

在 DOS 和 OS/2 下，clock 函数返回从进程开始所占用的时间，这个值可能和进程实际使用处理器的时间不同。

在 Microsoft C 的较早版本中，常量 CLOCKS_PER_SEC 为 CLK_TCK。

clock 函数返回以秒为单位的时间和常量 CLOCKS_PER_SEC 的乘积。如果不能获得处理器时间，函数返回-1，并转换为 clock_t 类型。

〖参见〗difftime,time。

ctime

〖功能〗该函数将存储在 timer 指向的 time_t 类型中的时间值转换为字符串。

〖原型〗char *ctime(time_t *timer)

〖位置〗time.h

〖说明〗通常可以调用 time 函数获得参数 timer 的值，time 函数返回从 1970 年 1 月 1

日格林威治时间 00:00:00 开始到当前时间的秒数。

ctime 函数的结果字符串包括 26 个字符，形式如下。

```
Wed Jan 02 02:03:55 1980\n\0
```

这里使用 24 小时时钟，所有字段有固定的域宽，字符串的最后两个字符为换行符(\n)和空字符(\0)。

ctime 函数修改 gmtime 函数和 localtime 函数使用的一个静态分配的缓存区，每次调用时都会破坏前一次调用的结果。ctime 函数同时还和 asctime 函数共享静态缓存区。调用 ctime 函数会破坏以前调用 asctime、gmtime 或 localtime 的结果。

ctime 函数返回指向结果字符串的指针。如果 timer 代表的日期早于 1980 年，ctime 返回 NULL。

〖参见〗asctime,ftime,gmtime,localtime,time。

difftime

〖功能〗该函数计算两个时间参数 timer0 和 timer1 之间的差。

〖原型〗double difftime(time_t timer1，time_t timer0)

〖位置〗time.h

〖说明〗difftime 函数以秒为单位返回从 timer0 到 timer1 之间的时间，返回值为双精度类型。

〖参见〗time。

ftime

〖功能〗函数获得当前时间并将其存储到 timeptr 指向的结构体中。

〖原型〗void ftime(struct timeb *timeptr)

〖位置〗sys\types.h，sys\timeb.h

〖说明〗timeb 类型结构体定义在 SYS\TIMEB.H 中，共有 4 个字段：dstflag，millitm，time 和 timezone，其值如下。

字段	值
dstflag	若本地时区现在使用的是夏令时，则为非 0 值(关于夏令时参阅 tzset 函数)。
millitm	秒的以毫秒为单位的小数部分。由于毫秒的最小增量为百分之一秒，所以最后一位始终为 0。
time	从格林威治时间 1970 年 1 月 1 日 00:00:00 开始的秒数。
timezone	从东向西，格林威治时间和本地时间的时差(以分为单位)，timezone 的值来自全局变量 timezone(参阅 tzset)。

ftime 函数给 timeptr 指向的结构体的各个字段赋值，该函数无返回值。

〖参见〗asctime,ctime,gmtime,localtime,time,tzset。

gmtime

〖功能〗该函数将参数 timer 指向的值转换为结构体。

〖原型〗struct tm *gmtime(time_t *timer)

〖位置〗time.h

〖说明〗参数 timer 代表了从格林威治时间 1970 年 1 月 1 日 00:00:00 开始经历的时间秒数，这个值通常是调用 time 函数获得的。

gmtime 函数分解 timer 的值并将其存储到 tm 类型(在 TIME.H 中定义)的结构体中。结果结构体反映的是格林威治时间，而不是本地时间。

gmtime、mktime 和 localtime 函数使用同一个静态分配的结构体保存结果，每次调用这些函数时会破坏上一次调用的结果。

DOS 和 OS/2 不能提供 1980 年以前的时间，如果 timer 代表的日期早于 1980 年，则 gmtime 返回 NULL。

gmtime 函数返回指向结果结构体的指针。

〖参见〗asctime,ctime,ftime,localtime,time。

localtime

〖功能〗该函数将存储在 time_t 类型对象中的时间转换为结构体类型。

〖原型〗struct tm *localtime(time_t *timer)

〖位置〗time.h

〖说明〗timer 代表了从格林威治时间 1970 年 1 月 1 日 00:00:00 开始经历的秒数，这个值通常可从 time 函数获取。

localtime 函数分解 timer 的值，并根据本地时区和夏令时校正为适当的本地时间，然后将校正后的时间存储到 tm 类型的结构体中。

注意，gmtime、mktime 和 localtime 函数使用同一个静态分配的缓存区，每次调用这些函数时都会破坏上一次调用的结果。

若用户首次设置环境变量 TZ，则 localtime 函数将时间校正为本地时区时间，设置好 TZ 后，其他 3 个环境变量(timezone、daylight 和 tzname)会自动设置。关于这些变量的描述参阅 tzset 函数。

ANSI 标准没有为 localtime 定义变量 TZ，它是 Microsoft 扩展的。

localtime 函数返回指向结果结构体的指针。DOS 和 OS/2 不提供早于 1980 年以前的时间值，如果 timer 表示的日期早于 1980 年 1 月 1 日，则函数返回 NULL。

〖参见〗asctime,ctime,ftime,gmtime,time,tzset。

mktime

〖功能〗该函数首先将 timeptr 指向的时间结构体转换成完整的“规格化”结构体，然后在转换成 time_t 日历时间值。

〖原型〗time_t mktime(struct tm *timeptr)

〖位置〗time.h

〖说明〗timeptr 指向的结构体可以是不完整的。

转换后的时间值同 time 函数返回值的编码一致。忽略 timeptr 结构体初始值的 tm_wday 和 tm_yday 字段，将其他字段不局限于它们的规格范围内。

如果转换成功，mktime 恰当地设置 tm_wday 和 tm_yday 字段的值。令其他字段代表指定的日历时间，但需将它们的值限制在规格范围内。直到确定了 tm_mon 和 tm_year 的值后才设置 tm_mday 的最终值。

DOS 和 OS/2 不提供早于 1980 年的日期。如果 timeptr 指向的日期早于 1980 年 1 月 1 日，则 mktime 返回-1。

注意，gmtime 和 localtime 函数使用同一个静态分配的缓存区。如果 mktime 函数使用这个缓存区，就会破坏上一次调用这些函数的结果。

mktime 函数返回 time_t 类型的日历时间编码。若无法表达日历时间，则函数将-1 转换成 time_t 类型返回。

〖参见〗asctime,gmtime,localtime,time。

_strdate

〖功能〗该函数将日期赋值到 datestr 指定的缓存区中。

〖原型〗char *_strdate(char *datestr)

〖位置〗time.h

〖说明〗日期的格式为 mm/dd/yy，mm 为两位数的月份，dd 为两位数的日期，yy 为两位数的年份。

缓存区至少要有 9 字节。

strdate 函数返回指向结果字符串的指针 datestr。

〖参见〗asctime,ctime,gmtime,localtime,mktime,time,tzset。

strftime

〖功能〗strftime 函数根据提供的格式参数 format 将 timeptr 指向的 tm 类型的时间值格式化，将结果存储到缓存区 string 中。

〖原型〗size_t strftime(char *string，size_t maxsize，char *format，struct tm *timeptr)

〖位置〗time.h

〖说明〗参数 maxsize 为缓存区中可存储的最多字符数。

格式由一个或多代码组成；同 printf 一样，格式代码以%开头。不以%开头的字符原样复制到 string 中。当前位置的属性 LH_TIME 会影响 strftimes 的输出格式。

下面是 strftime 的格式代码。

格式	描述
%a	缩写星期名。
%A	完整星期名。
%b	缩写月份名称。
%B	完整月份名称。

%c	与当前位置一致的日期和时间表示法。
%d	用十进制表示每月中的日期(01~31)。
%H	用 24 小时格式表示时间(00~23)。
%I	用 12 小时格式表示时间(01~12)。
%j	用十进制数表示一年中的每一天 (001~366)。
%m	用十进制数表示 12 个月(01~12)。
%M	用十进制数表示分 (00~59)。
%p	用当前位置的 AM/PM 表示 12 小时时钟。
%S	用十进制数表示每秒 (00~61)。
%U	用十进制数表示一年中的星期；星期日为一周的开头(00~53)。
%w	用十进制数表示一周的每天(0~6; 星期日为 0)。
%W	用十进制数表示一年中的星期；星期一为每周的第 1 天(00~53)。
%x	当前位置的日期表示法。
%X	当前位置的时间表示法。
%y	两位十进制数表示年份 (00~99)。
%Y	年份的完整表示法(4 位十进制数)。
%z	时区名或缩写，若时区未知则不加字符。
%%	百分号。

当结果字符串(包括结束空字符)不超过 maxsize 时，strftime 函数返回 string 中存储的字符数；否则 strftime 返回 0，string 中的内容不确定。

〖参见〗localeconv,setlocale,strxfrm。

_strtime

〖功能〗该函数将当前时间复制到 timestr 指定的缓存区中。

〖原型〗char *_strtime(char *timestr)

〖位置〗time.h

〖说明〗时间的格式为 hh/mm/ss，hh 为两位数的 24 小时制的小时数，mm 为两位数的分钟数，ss 为两位数的秒数。

缓存区至少要有 9 字节。

_strtime 函数返回指向结果字符串的指针 timestr。

〖参见〗asctime,ctime,gmtime,localtime,mktime,time,tzset。

time

〖功能〗该函数根据系统时钟，返回从格林威治时间 1970 年 1 月 1 日 00:00:00 开始经历的秒数。

〖原型〗time_t time(time_t *timer)

〖位置〗time.h

〖说明〗该函数根据 timezone 系统变量校正系统时间(关于这个变量参阅 tzset 函数)。

返回值存储在 timer 指定的位置。参数 timer 可以为 NULL，表示不存储返回值。

该函数无错误返回值。

〖参见〗asctime,ftime,gmtime,localtime,tzset,utime。

tzset

〖功能〗该函数使用当前环境变量 TZ 的设置为 3 个全局变量：daylight，timezone 和 tzname 赋值。

〖原型〗void tzset(void)

〖位置〗time.h

〖说明〗ftime 和 localtime 函数将 time 函数返回的格林威治时间校正为本地时间时，需要用到这些变量。

环境变量 TZ 的值应是一个 3 字母时区名，如 PST，后跟表示格林威治和本地时差小时数的带符号数(可选项)。这个数字后面还可跟 3 字母的夏令时(DST)时区，如 PDT。例如，“PST8PDT”为太平洋时区的正确 TZ 值。如果不是夏令时，则无需设置夏令时时区。

若不设置 TZ，则默认值为 PSD8PDT，表示太平洋时区。

根据环境变量 TZ 调用 tzset 函数时，将下列值赋给 daylignt、timezome 和 tzname 变量。

变量	值
daylight	在 TZ 中指定了夏令时，为非 0 值；否则为 0。
timezone	本地时间和格林威治时间之间的秒数时差。
tzname[0]	TZ 中的 3 字母时区名。
tzname[1]	TZ 中的 3 字母夏令时时区名。若 TZ 中无此项设置，则为空串。

daylight 的默认值为 1，timezone 的默认值为 28 800，tzname[0]的默认值为 PST，tzname[1]的默认值为 PDT。对应的 TZ 为 PST8PDT。

如果 TZ 设置中省略了夏令时，则 daylight 的值为 0，ftime、gmtime 和 localtime 函数返回的 DST(夏令时)标志也为 0。

该函数无返回值。

〖参见〗asctime,daylight,ftime,gmtime,localtime,time,timezone,tzname。

utime

〖功能〗该函数设置 filename 指定文件的修改时间。

〖原型〗int utime(char *filename，struct utimbuf *times)

〖位置〗sys\types.h，sys\utime.h，errno.h

〖说明〗当前进程必须能够写访问该文件，否则无法改变时间。

尽管 utimbuf 类型结构体中包含了访问时间字段，但只能在 DOS 或 OS/2 下设置修改时间。如果 times 为 NULL，则将修改时间设为当前时间；否则 times 必须指向一个 utimbuf 类型结构体(在 SYS\UTIME.H 中定义)。修改时间来自这个结构体的 modtime 字段。

若改变了文件修改时间，utime 函数返回 0。若函数返回-1 表示出现错误，同时将 errno

设为 EACCES，EINVAL，EMFILE 或 ENOENT。

〖参见〗asctime,ctime,fstat,ftime,gmtime,localtime,stat,time。

3.20 可变长参数列表

va_arg、va_end、va_start

〖功能〗va_start、va_arg 和 va_end 宏为参数数量变化的函数提供了一种简便的访问参数的途径。

〖原型〗type va_arg(va_list arg_ptr，type)
void va_end(va_list arg_ptr)
void va_start(va_list arg_ptr，prev_param)

〖位置〗stdarg.h，stdio.h

〖说明〗这些宏有两个版本：定义在 ATDARG.H 中的宏和 ANSI C 标准一致，定义在 VARARGS.H 中的宏和 UNIX 系统的 V 定义兼容。

这两个版本的宏都认为函数有一组固定数量的必选参数，后跟一组变数量可选的参数。用通常的方式声明必选参数，可用参数名访问参数。用 STDARG.H 或 VARARGS.H 宏访问可选参数，它们设置一个指向参数列表中的第 1 个可选参数的指针，通过指针检索参数列表，当参数处理过程完成后将指针复位。

前面提到的 ANSI C 标准宏(定义在 STDARG.H)的用法有如下 4 种。

1. 函数的所以必选参数按照普通方式声明。

2. va_start 宏将 arg_ptr 指向传递给函数的参数列表的第 1 个可选参数，参数 arg_ptr 的值必须是 va_list 类型，参数 prev_param 为参数列表中第 1 个可选参数之前的必选参数名。若 prev_param 为寄存器存储类型，则未定义宏的操作。必须在首次使用 va_arg 宏之前使用 va_start 宏。

3. va_arg 宏实现如下操作。

- 从 arg_ptr 指定的位置开始检索 type 的值。
- 对 arg_prt 自增，令其指向参数列表中的下一个参数，type 类型的大小确定了增加的量。

在函数中可以多次调用 va_arg 宏检索参数。

4. 检索完所有参数后，va_end 将指针复位为 NULL。

va_arg 宏返回当前的参数，va_start 和 va_end 宏无返回值。

〖参见〗vfprintf,vprintf,vsprintf。

索　　引

Unix C 函数

Turbo C 函数

Microsoft C 函数