# date01

## func:open(2)

## func:read(2)

## func:write(2)

## func:lseek(2)

## func:close(2)

## func:dup(2)

## func:dup2(2)

## func:creat(2)

## func:sync(2)

## func:fsync(2)

## func:fdatasync(2)

## func:fcntl(2)

## func:ioctl(2)

# date02

## func:fopen(3)

#include <stdio.h>

FILE \*fopen(const char \*path,const char \*mode);

简介：

返回直：

失败返回NULL，errno被设置。

## func:fflush(3)

#include <stdio.h>

int fflush(FILE \*stream);

简介：

如果stream是一个输出流，该函数会强制的把其缓存区的内容写入stream

中，如果stream为NULL，则刷新所有输出流。如果成功则返回0,失败则反

回EOF这个宏，errno会被重设。

## func:gets(3)

#include <stdio.h>

char \*gets(char \*s);

简介：

从标准输入中读取一行字符串包含空格，回添到\*s，返回值这个\*s的地址

,失败返回NULL。

## func:puts(3)

#include <stdio.h>

int puts(const char \*s);

简介：

将字符串\*s写如到标准输出，并换行。成功返回一个非负的数字，失败返

回EOF。

## func:fgets(3)

#include <stdio.h>

char \*fgets(char \*s, int size, FILE \*stream);

简介：

最多从\*stream中读取size-1个字符，存储到\*s的缓存区中。当遇到EOF或

者是换行时结束。读完后会在\*s中存一个'\0'。成功返回指向\*s的指针，

失败或者对完了返回NULL。

## func:fputs(3)

## func:getline(3)

## func:fgetc(3)

## func:fputc(3)

## func:getchar(3)

## func:putchar(3)

## func:putc(3)

## func:getc(3)

## func:fread(3)

## func:fwrite(3)

## func:fscanf(3)

## func:fprintf(3)

## func:ferror(3)

#include <stdio.h>

int ferror(FILE \*stream);

简介：

该函数返回的是\*stream的文件的错误标志。如果有该标志，则返回一个非

0的数。

## func:feof(3)

#include <stdio.h>

int feof(FILE \*stream)

简介：

该函数返回的是\*stream的文件结束标志。如果是文件结束位置，返回的是

一个非0的数。

## func:perror(3)

#include <stdio.h>

void perror(const char \*s);

简介：

该函数会打印当前代码位置之前的最后一个由于系统调用或库调用所产生

的错误的错误信息。\*s是一个提示字符串，错误信息加空格后打印在该字

符串后。

## func:fseek(3)

#include <stdio.h>

int fseek(FILE \*stream, long offset, int whence);

简介：

该函数用于设置文件流\*stream的当前的positon。whence代表相对的位置

offset代表偏移多少。成功返回0,使用ftell（）可以确定当前文件流的位

置，错误返回-1,errno重设。

## func:fclose(3)

#include <stdio.h>

int fclose(FILE \*stream);

简介：

关闭一个已经打开的文件流，成功则返回0,不成功返回EOF，errno重设。

## func:tmpfile(3)

#include <stdio.h>

FILE \*tmpfile(void);

简介：

该函数将以w+b模式打开一个文件，并返回该文件流。当要关闭该文件流时

这个文件将会被自动删除。

## func:fdopen(3)

#include <stdio.h>

FILE \*fdopen(int fd, const char \*mode);

简介：

通过一个文件描述符fd，打开一个文件流并返回。\*mode代表的是打开方式

。失败返回NULL，errno重置。

## func:fileno(3)

#include <stdio.h>

int fileno(FILE \*stream);

简介：

该函数会根据文件流\*stream返回该函数的文件描述符。

# date03

## func:stat(2) fstat(2) lstat(2)

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

int stat(const char \*pathname, struct stat \*buf);

int fstaat(int fd, struct stat \*buf);

int lstat(const char \*pathname, struct stat \*buf);

简介：

三个函数的作用都是读取pathname或者是一个已经打开的文件的fd所代表

的文件的一些描述信息，并把这些信息存储在一个结构体 \*buf中。与st-

at不同的是，fstat要读取的文件必须是打开的，通过整个文件的文件描

述符去读取。lstat与stat的区别是：如果分析的文件是一个符号链接文

件，lstat分析的是这个符号链接文件本身的信息，而stat读取的是符号

链接所指向的那个文件。

参数：

\*buf所代表的结构体struct stat 成员如下：

struct stat {

dev\_t st\_dev; /\* ID of device containing file \*/

ino\_t st\_ino; /\* inode number \*/

mode\_t st\_mode; /\* protection \*/

nlink\_t st\_nlink; /\* number of hard links \*/

uid\_t st\_uid; /\* user ID of owner \*/

gid\_t st\_gid; /\* group ID of owner \*/

dev\_t st\_rdev; /\* device ID (if special file) \*/

off\_t st\_size; /\* total size, in bytes \*/

blksize\_t st\_blksize; /\* blocksize for filesystem I/O \*/

blkcnt\_t st\_blocks; /\* number of 512B blocks

allocated \*/

/\* Since Linux 2.6, the kernel supports nanosecond

precision for the following timestamp fields.

For the details before Linux 2.6, see NOTES. \*/

struct timespec st\_atim; /\* time of last access \*/

struct timespec st\_mtim; /\* time of last modification \*/

struct timespec st\_ctim; /\* time of last status change \*/

#define st\_atime st\_atim.tv\_sec

/\* Backward compatibility \*/

#define st\_mtime st\_mtim.tv\_sec

#define st\_ctime st\_ctim.tv\_sec

};

返回直：

成功返回0,失败返回-1。errno被设置。

## func:getpwuid(3) getpwnam(3)

#include <sys/types.h>

#include <pwe.h>

struct passwd \*getpwnam(const char \*name);

struct passwd \*getpwuid(uid\_t uid);

简介：

两个函数都作用相同，通过用户的名字\*name或者用户的uid，去/etc/pa-

sswd文件中寻找与之相对应的信息，并且会开劈一个静态结构体存储这些

信息，结果返回一个结构体指针，指向这些存储信息。

参数：

结构体 struct passwd的成员：

struct passwd {

char \*pw\_name; /\* username \*/

char \*pw\_passwd; /\* user password \*/

uid\_t pw\_uid; /\* user ID \*/

gid\_t pw\_gid; /\* group ID \*/

char \*pw\_gecos; /\* user information \*/

char \*pw\_dir; /\* home directory \*/

char \*pw\_shell; /\* shell program \*/

};

返回直：

成功返回一个指向struct passwd静态区的指针，失败或者没有找到返回

NULL，失败errno会诚信设置。如果要判断到底是出错还是没有找到调用

前要将errno赋值为0,调用后检查。

## func:getgrgid(3) getgrnam(3)

#include <sys/types.h>

#include <grp.h>

struct group \*getgrname(const char \*name);

struct group \*getgrgid(gid\_t gid);

简介：

功能与getpwuid(3) getpwnam(3)类似。不同的是组函数的作用是从/etc-

/group中查询用户组的信息。struct group结构体成员如下：

struct group {

char \*gr\_name; /\* group name \*/

char \*gr\_passwd; /\* group password \*/

gid\_t gr\_gid; /\* group ID \*/

char \*\*gr\_mem; /\* NULL-terminated array of pointers

to names of group members \*/

};

## func:getspnam(3)

#include <shadow.h>

struct spwd \*getspanm(const char \*name);

简介：

该函数被调用后会在/etc/shadow中寻找与用户名\*name相匹配的用户密

码信息，如果找到，返回一个指向静态的结构体的 structspwd的指针。

失败或者找不到的话返回NULL，出错时errno会被设置，所以要判断到底

是没找到还是出错需要提前将errno设置为0。

## func:crypt(3)

#define \_XOPEN\_SOURCE

#include <unistd.h>

char \*crypt(const char \*key, const char \*salt);

link with -lcrypt.

简介：

这个函数的功能是根据用户输入的密码 \*key和指定的盐值\*salt，haxi生

成一个加密后的字符串。结果是返回一个指针，指向生成后的加密字符串

，如果函数失败返回直为NULL。\*salt盐值包含两部分可以理解为/etc/s-

hadow中加密字符串的前两个被$区分出来的字符串。

## func:getpass(3)

#include <unistd.h>

char \*getpas(const char \*prompt);

简介：

这是一个被淘汰的函数，不要用，使用ECHO参数的termios(3)可以代替他

。这个函数的作用是从标准输入读取一行字符串，但不回显。如果调用成

功，返回一个指向静态存储输入字符串的指针，失败则返回0。参数\*pro-

mpt是一段字符串，打印在标准输出上作为下一刻要读取标准输入的提示

字符。

## func:time(2)

#include <time.h>

time\_t time(time\_t \*tloc);

简介：

该函数的作用是返回一个代表从1970-01-01 00:00:00到当前时刻的秒数

。tloc可以是NULL，如果不是，这个描述会回添到\*tloc中。函数调用失

败返回-1,errno被设置。

## func:localtime(3)

#include <time.h>

struct tm \*localtime(const time\_t \*timep);

简介：

该函数被调用时会根据一个从1970-01-01 00:00:00到某一时刻的秒数，

计算该时刻的本地时间的信息，并且存储到一个静态的结构体struct tm

中，返回一个指针指向这个结构体。失败会返回NULL。strut tm结构体的

成员如下：

struct tm {

int tm\_sec; /\* Seconds (0-60) \*/

int tm\_min; /\* Minutes (0-59) \*/

int tm\_hour; /\* Hours (0-23) \*/

int tm\_mday; /\* Day of the month (1-31) \*/

int tm\_mon; /\* Month (0-11) \*/

int tm\_year; /\* Year - 1900 \*/

int tm\_wday; /\* Day of the week (0-6, Sunday = 0) \*/

int tm\_yday; /\* Day in the year (0-365, 1 Jan = 0) \*/

int tm\_isdst; /\* Daylight saving time \*/

};

## func:mktime(3)

#include <time>

time\_t mktime(struct tm \*tm);

简介：

该函数的作用是根据，一个\*tm代表时间信息的结构体，返回这个结构体

代表的时刻到1970-01-01 00:00:00的秒数。失败返回-1。

## func:strftime(3)

#include <time.h>

size\_t strftime(char \*s, size\_t max, const char \*format,

const struct tm \*tm);

简介：

该函数的作用是将从存储时间信息的结构体\*tm中的信息，按照格式化字

符串\*format规定的格式，回添到具有max大小的缓存去\*s中。如果max足

够大的容纳回添结果（包含\0）返回直是回添字节个数。如果不足够容纳

返回直是0。

## func:opendir(3)

#include <sys/types.h>

#include <dirent.h>

DIR \*opendir(const char \*name);

简介：

该函数的作用是根据\*name打开一个目录，返回一个DIR类的指针。失败返

回NULL，errno被重置。

## func:readdir(3)

#include <dirent.h>

struct diren \*readdir(DIR \*dirp);

简介：

根据DIR \*dirp去读取一个路径，跟读文件类似。读取结果存储在静态区的

struct dirent结构体中，返回一个指针指向这个结构体,失败返回NULL。

struct dirent的成员如下：

struct dirent {

ino\_t d\_ino; /\* inode number \*/

off\_t d\_off; /\* not an offset; see NOTES \*/

unsigned short d\_reclen; /\* length of this record \*/

unsigned char d\_type; /\* type of file; not supported

by all filesystem types \*/

char d\_name[256]; /\* filename \*/

};

## func:closedir(3)

#include <sys/types.h>

#include <dirent.h>

int closedir(DIR \*dirp);

简介：

根据\*dirp关闭一个路径流，成功返回0,失败返回-1,并且设置errno。

## func:glob(3)

#include <glob.h>

int glob(const char \*pattern, int flags,

int (\*errfunc)(const char \*epath, int errno),

glob\_t \*pglob);

void globfree(glob\_t \*pglob);

简介：

glob函数会根据字符串\*pattern的内容在所有路径中查找和他相匹配的项

的文件路径（包括名字）并存储到一个结构体\*pglob中。如果想模糊查找

\*pattern的最后一个字符用'\*',此外要匹配的字符中不能有波浪线。搜索

的路径中也不包含隐藏文件，如果要搜索要以'.'开头。

参数：

flags是一些特殊功能的选项，选怎特定的宏，功能不同，详细的查看man

手册。glob\_t的结构体中，有三个成员，第一个是符合匹配项的数目，第

二个存储的是匹配项字符串的一个指针，存放这些字符串的空间，应该是

glob函数开辟的所以最后，要通过globfree释放。

返回直：

如果成功返回0,如果不成功可能会返回三个宏，代表不同的作用。参数er-

rfunc是一个自定义的函数指针，当glob出错时会自动调用这个函数，一般

不用设置为NULL即可，如果用记得该错误处理函数定义时成功状态要返回0

。

# date04

## func:getopt(3)

#inlcude <unistd.h>

int getopt(int argc, char \* const argv[], const char \*optstring);

extern char \*optarg;

extern int optind, opterr,optot;

简介：

该函数每次被调用时会根据选项字符串optstring,检查命令行参数的第op-

tind(ps:初始值是1,即从第二个命令行参数开始)个参数开始解析其是否是

optstring中规定的选项字符，如果是则分析该选项的参数情况。四个全局

变量，optarg存储该次解析的选项字符的参数的值，optind指定了本次解

析第几个命令行参数（初次调用是1,以后会自加），opterr如果是1,那么

本次解析如果不符合optstring的要求则会打印出错误信息，如果解析到的

不是optstring中的选项字符，该字符ascii码会存储在optot中。

形参：

optstring——"-ab:c::" "-"代表可以解析不带选项的参数，"a"参数后不加

冒代表只有选项a没有参数，"b:"加一个冒号表示必须加一个参数，参数和

选项可以加空格也可以不加空格，"c::"加两个冒号表示可以加参数也可以

不加参数,但是参数必须紧跟选项不能加空格。

返回值：

如果本次解析到了optstring中的选项字符，则返回该字符的asscii码。

## func:getoptlong(3)

#include <getopt.h>

int getopt\_long(int argc, char \* const argv[],

const char \*optstring,

const struct optin \*longopts, int \*longindex);

简介：该函数的功能是getopt的增强版，getopt能解析的选项只能是单个

字符,getoptlong 可以解析以字符串为整体的选项(--option),前三个参

数的用法不变，如果只想解析长选项，optstring = "",longopts

## func:strftime

# date05

## func:exit(3)

#include <stdlib.h>

void exit(int status);

简介：

该函数可以以正常的方式从此位置结束当前的进程，并且调用atexit所指

向的处理函数，而且能够自动的处理IO缓存区(和return一样)，最后返回

status & 0377(即返 回的是status的低八位)。

形参：

status——c标准中有两个宏（EXIT\_SUCCESS \ EXIT\_FAILURE）可以作为该

形参的直，分别代表成功与失败。

## func:\_exit(2) \_Exit(2)

#include <unistd.h>

void \_exit(int status);

#include <stdlib.h>

void \_Exit(int status);

简介：

两函数的功能一样，都是立即结束当前调用的进程，关闭该进程已经打开

的文件描述符，该进程的子进程会被进程1接管，status作为一个返回状态

给该进程的双亲进程。该函数的功能与exit()的不同是，调用该函数后程

直接就被内核接管了，而exit()还会调用atexit，清理IO缓存。

## func:pthread\_exit(3)

#include <pthread.h>

void pthread\_exit(void \*retval);

简介：

该函数会结束会结束当前的线程，如果这个线程来源与joinable形式，\*-

retval中的直将会返回，这个函数同样对同一个进程中被pthread\_join生

成的线程有效。

## func:abort(3)

#include <stdlib.h>

void abort(void);

简介：

异常的结束当前的进程，兵器不会有返回直。

## func:atexit(3)

#include <stdlib.h>

int atexit(void (\*function)(void));

简介：

当一个进程正常的结束，或者通过exit(3)\retrun结束时，该函数会被自

动调用，并且会调用指针function所指向的函数。

返回值：

成功返回0,失败返回非0。

## func:getenv(3)

#include <stdlib.h>

char \*getenv(const char \*name);

简介：

该函数在环境变量的列表中寻找名字为\*name的环境变量，如果找到了返回

指向该直字符串的指针，如果找不到匹配的，返回NULL。没有失败的解释

。

## func:putenv(3)

#include <stdlib.h>

int putenv(char \*string);

简介：

该函数的公功能是添加或者改变一个环境变量的直（如果该环境变量的

name已经存在）,环境变量的name和value存储在\*string中。这个字符串

的格式必须为“name=value”。

返回值：

成功返回0,失败返回非零的数。errno会被设置。

## func:setenv(3)

#include <stdlib.h>

int setenv(const char \*name, const char \*value,int overwrite);

简介：

该函数的作用是添加一个环境变量的名字name，并且复制为value。如果

这个环境变量已经存在，并且overwrite不为0,value会替换原来的名字。

如果overwrite为0,则不会改变原来环境变量的直，并且会返回一个成功

状态。

返回值：

成功返回0,失败返回-1。errno会被设置。

## func:setjmp(3) longjmp(3)

#include <setjmp.h>

int setjmp(jmp\_buf env);

#include <setjmp.h>

void longjmp(jmp\_buf env, int val);

简介：

两个函数是相互配合使用的，两个函数可以实现在两个函数或者两个.c文

件之间的goto操作。当首次调用setjmp时，setjmp函数将此位置的信息存

储在一个自己定义的全局变量env中，此时setjmp的返回直是0。当有调用

longjmp时，longjmp函数会自动的分析全局变量env中的信息，然后程序自

动跳转到，setjmp初次调用的位置运行。并且longjmp会将val传回第二次

调用的setjmp函数中，这个val将作为第二次setjmp的返回直。

## func:fork(2)

#include <unistd.h>

pid\_t fork(void);

简介：

该函数的作用是复制当前进程（完全生成令一套的虚拟内存），新的进程

作为当前进程的子进程，父子进程具有相同的页表、进程表项、标准io缓

存，但是pid、ppid、资源使用量不相同。注意的是虽然具有相同的页表，

但是“写时复制”。

返回直：

当前进程中的返回直是子进程的pid，子进程中的返回直是0。失败时，zi

进程不会产生，parent进程中返回直为-1,errno会被重设置。

## func:getpid(2) getppid(2)

#include <sys/types.h>

#include <unistd.h>

pid\_t getpid(void);

pid\_t getppid(void);

简介：

两个函数的功能分别是获得当前进程的pid和获得当前进程的parent进程，

这两个函数被调用后不会失败。

## func:wait(2) waitpid(2)

#include <sys/types.h>

#inlcude <sys/wait.h>

pid\_t wait(int \*status);

pid\_t waitpid(pid\_t pid, int \*status, int options);

简介：

wait——当前进程中的各个子进程当有发生结束的状态改变时，wait函数会

返回，否则会阻塞等带，知道有子进程状态发生改变。

参数：

wait——子函数发生改变的状态会以一个整数的形式存在\*status中，查询这

个整数应该使用man手册中的宏函数去查看，详情产看man手册。

返回直：

函数成功返回，发生改变的进程的pid号，失败返回-1,errno重置。

## func:execl(3)

#include <unistd.h>

extern char \*\*environ;

int execl(const char \*path, const char \*arg, ...

/\*(char \*) NULL\*/);

int execvp(const char \*file, char \*const argv[]);

简介：

这两个函数的作用是调佣另一个进程的代码，将当前的进程完全替换并执

行。

参数：

execl ——\*path是将要调佣的进程的路径(包涵代码), \*arg中存储的是被调

用的进程执行时可使用的选项，这个选项可以是多个。

execvp——\*file中是将要调用的进程的可执行文件的名字，至于文件的路径

将会自动的在环境变量中寻找，argv是一个二级的指针，指向的

是存储在一个数组中的各个字符串，这些字符串是将要被调用进

程的的选项，不用指定选项个数，最后一个指向的是NULL。注意

这个字符串数组中，第一个应该是要被调用的可执行文件的名字

。

返回直：

这个函数不会有返回直，因为成功了，原来的进程就被替换了。如果这个

函数没有成功将会返回-1,并且会设置errno。

注意：

可执行程序重写的是text, data, bss, and stack这些，该进程还是那个

进程pid号没有变化，也就是进程表项没有改变。

# date06

## func:vfork(2)

#include <sys/types.h>

#include <unistd.h>

pid\_t vfork(void);

简介：

父子进程数据段共享,因为页表不复制共同用同一个页表

\*用\_exit \_Exit

\*其中一个建议要使用exec

阻塞

过时的函数

## ^func:waitpid(3)

#include <sys/tyepes.h

#include <sys/wait.h>

pid\_t waitpid(pid\_t pid,int \*status, int options);

简介：

该函数会等待pid\_t指定的子进程的状态（被option指定的状态）发生改变

时才会返回，不然会一直等待。\*status用于存放状态改变的信息。函数反

回是状态改变的那个进程的pid号。

参数：

pid的直，< -1,等待的是进程组号为|pid|的子进程; -1,等待所有子进程;

0等待和调用进程同一组的所有子进程. > 0,等待pid号的子进程。

optons有三个宏选项，WNOHANG，立即返回，就是在这一时刻看看有没有进

成发生改变；WUNTRACED，如果改变的方式是进程被信号stop也返回；WCO-

NTINUED，如果改变的方式是进程被信号resumed也返回。

status，也可以是NULL，如过不是在产看到底是什么信息时，使用宏函数

，具体查看手册。

## func:getpgid(2)

#include <unistd.h>

ptd\_t getpgid(pid\_t pid);

简介：

获取进程pid所属的进程组的id，如果pid的直为0,就是获取当前进程的组

的id，成功返回对应的进程组的id，失败返回-1,设置errno。

## func:system(3)

#include <stdlib.h>

int system(const char \*command);

简介：

该函数的作用是床见一个子进程，在该子进程中执行本地的shell程序去之

执行command所执行的命令，相当于“execl("/bin/sh", "sh", "-c", com-

mand, NULL);”.如果哦command为NULL，返回值代表该sh是否存在。如果

所有都是成功的返回直是command执行的程序的返回状态。这个状态的信息

要用waitpid（）中的宏函数去查看。

SIGCHLD--->block

SIGINI SIGQUIT --->

## fun:acct(2)

#include <unistd.h>

int acct(const char \*filename);

简介：

这个函数的功能是开启或关闭程序会计功能（一种对程序的记录功能），

\*filename必须是一个真是存在的文件。当任意进程结束时，会在\*filena-

me中追加一些记录。如果filename不是NULL代表打开进程会计功能，如果

取NULL代表关闭会计功能。成功返回0,失败返回-1,errno被设置。

## func:getlogin(3)

#include <unistd.h>

char \*getlogin(void);

简介：

该函数返回一个指向一个保存当前进程的用户的名字的静态区的字符串。

失败返回NULL，设置errno。

## func:setsid(2)

#include <unistd.h>

pid\_t setsid(void);

简介：

如果当前进程不是一个进程组的leader，调用该函数时，将会创建一个新

的会话，当前进程是这个新的会话的唯一一个进程，也是会话leader和进

程组leader。新创建的会话没有控制终端。如果成功，新会话的id将会返

回，失败返回-1,设置errno。

## func:umask(2)

#include <sys/types.h>

#include <sys/stat.h>

mode\_t umask(mode\_t mask);

简介;

该函数的作用是改变当前进程的umask直（当该进程创建文件时会用到）为

mask&0777(取低九位）。该函数一定会成功，返回直是之前设立的umask直

。

## func:chdir()

#include <unistd.h>

int chdir(const char \*path);

简介：

该函数的作用是改变当前进程的工作路径到path，成功返回0,失败返回-1,

设置erno。

## func:daemon(3)

#include <unistd.h>

int daemon(int nochdir,int noclose);

简介：

将当前进程转换为一个守护进程，当前进程必须是由另一个进程创建的子

进程，原父进程应该使用\_exit结束。nochdir为0代表改变当前进程的工作

路径改为/，不为0代表不改变。noclose为0代表将当前进程的标准输入、

标准输出、标准错误重定向进/dev/null，不为0代表不变。

## func:strerror(3)

#include <string.h>

char \*strerror(int errnum);

简介：

创建一个静态区，用来存储errnum所代表的错误信息的字符串，返回直指

向这个字符串。

## func:openlog()

#include <sysylog.h>

void openlong(const char &ident,int option, int facility);

简介：

这个函数的作用是建立一个和系统日志文件的链接关系，参数ident代表着

在日志文件中每条日之前添加的提示信息，如果设置成NULL，则会自动添

加成拥有者的名字。optinon和faciltty代表着一些操作选项，前者代表如

何操作，后者代表信息类型，详细说明参考man手册。

## func:syslog()

#include <syslong.h>

void syslog(int priority, const char \*format, ...);

简介：

这个函数的作用是向系统日志文件中添加日志信息，priority这个选项是

facility（消息的类别）类的宏和level（消息的重要性）类的宏的位或运

算，format是信息的格式化字符串，使用方法和printf类似。特别的是"%m

"和("%s",strerror(errno)效果一样)。格式化字符不用写\n,写了就输入

一个\n不是换行。

## func:closelog(3)

#include <syslog.h>

void closelog(void);

简介：

关闭指向系统日志的文件描述符号，该函数可以不调用。

## func:flock(2)

#include <sys/file.h>

int flock(int fd,int operation);

简介：

为fd指向的文件进行锁操作，operation有对应的宏代表着不同的操作，L-

OCK\_SH代表加一个共享锁;LOCK\_EX代表加一个独立的阻塞的锁（如果想要

加一个非阻塞的锁与宏LOCK\_NB进行位或运算）;LOCK\_UN代表解锁。

返回直：

成功返回0,失败返回-1,设置errno。

## func:lockf(3)

#include <unistd.h>

int lockf(int fd ,int cmd, off\_t len);

简介：

这个函数用于对一个已经打开的可以写的文件fd中指定的内容段（按位）

进行锁操作。具体的操作通过cmd的宏确定。要操作的内容通过参数len指

定，如果len是正数操作的内容是当前positon~pso+len-1,如果len是负数

操作的内容是pos-len~pso-1,如果len是0,代表从当前位置到永远的文件末

尾的内容（即使文件大小有改变）。

参数：

选项cmd可选的宏有：

F\_LOCK ——设置一个独立的锁，去锁定指定的为难间，如果该文件已经被加

锁了，则会阻塞等待这个锁释放。当关闭了文件后该文件的锁也

会被自动释放，子进程是不会继承锁的。

F\_TLOCK——不阻塞的锁，如果试图要枷锁的文件已经锁了，马上返回一个错

误。

F\_ULOCK——解锁

F\_TEST ——测试锁，如果指定文件中的指定内容没有被锁或者是被当前文件

锁住了，返回0.如过是被另一个进程锁住了返回-1。

返回直：

成功返回直，失败返回-1,设置errno。

## func:ftruncate(2)

#include <unistd.h>

#include <sys/types.h>

int ftruncate(int fd, off\_t length)

简介：

该函数的作用是将一个打开的能写的普通文件fd，截断到指定长度length

；如果文件的实际大小要不length小，则该文件将一'\0'补全；该函数不

改变文件当前的position。

返回直：

成功返回0,失败返回-1,errno设置。

# date07

## func:signal(2)

#include <signal.h>

typedef void (\*sighandler\_t)(int);

sighandler\_t signal(int signum, sighandler\_t handler);

简介：

该函数的行为和unix的版本有关，所以尽量不要使用他，而是使用sigact-

ion代替他。

## func:pause(2)

#include <unistd.h>

int pause(void);

简介：

该函数会等待有信号被传输过来或者是进程结束，不然会一直sleep等待。

如果返回，返回直是-1,errno会被设置成EINTR。

## func:kill(2)

#include <sys/types.h>

#include <signal.h>

int kill(pid\_t pid, int sig);

简介：

向进程pid发送信号sig，成功返回0,失败返回-1,errno被设置。pid :>0

向pid进程发送送信号， =0向该进程同组的所有进程发送， =-1向除了一

号进程外所有被允许的进程发送，<-1向进程组id为-pid的进程组发送。

## func:raise(2)

#include <signal.h>

int raise(int sig);

简介：

该函数向当前的进程或者线程发送一个信号sig，如果该信号有行为函数

raise需要等行为函数返回后才放回。成功返回0,失败返回一个非0的数。

## func:alarm(2)

#include <unistd.h>

unsigned int alarm(unsigned int seconds);

简介：

在当前进程中seconds后向本进程发送一个SIGALRM信号，该函数最后一次

的调用会使之前的调用变的无效。返回直是上一次的定时还剩多少时间到

时，如果是第一次调用返回直是0.

# date08

## func:sigemptyset(3)

#include <signal.h>

int sigemptyset(sigset\_t \*set);

简介：

将信号集\*set的所有信号清空，即该信号集中不包含任何信号，成功返回

0,失败返回-1,设置errno。

## func:sigfillset(3)

#include <signal.h>

int sigfillset(sigset\_t \*set);

简介：

将所有的信号填入信号集\*set中，成功返回0,失败返回-1,设置errno。

## func:sigaddset(3)

#include <signal.h>

int sigaddset(sigset\_t \*set, int signum);

简介：

将信号signum，添加到信号集\*set中，成功返回0,失败返回-1，设置errno

## func:sigdelset(3)

#include <signal.h>

int sigdelset(sigset\_t \*set, int signum);

简介：

将信号signum从信号集\*set中剖出，成功返回0,失败返回-1,设置errno。

## func:sigismember(3)

#include <signal.h>

int sigismember(const sigset\_t \*set, int signum);

简介：

判断信号signum是否在信号集\*set中，是则返回1,不是返回0,失败返回

-1,设置errno。

## func:sigprocmask(2)

#include <signal.h>

int sigprocmask(int how,const sigset\_t \*set,sigeset\_t \*olds);

简介：

每一个进程都有一个信号屏蔽字集，该集中有哪些信号，当这些信号向该

进程发送时就会被屏蔽。该函数的作用是，将\*set中的信号向屏蔽字集中

添加、抛出、重设置,同时将就的屏蔽字集信息存储到\*olds中。至于是何

种操作，决定于how可选的宏有SIG\_BLOCK\SIG\_UNBLOCK\SIG\_SETMASK。分

别是添加、抛出、重设置。函数成功返回0,失败返回-1,设置errno。

## func:sigsuspend(3)

#include <signal.h>

int sigsuspend(const sigset\_t \*mask);

简介：

该函数的作用是暂时将当前进程的信号屏蔽集使用\*mask替换，然后等待

直到有其他未被屏蔽的信号执行万行为函数后，该函数才返回，并且回复

原来的信号屁股比集。该函数对SIGKILL SIGSTOP信号效。返回值总是-1,

设置errno，正常返回EINTR.

## func:sigaction(2)

#include <signal.h>

int sigaction(int signum, const struct sigaction \*act,

struct sigaction \*oldact);

struct sigaction{

void (\*sa\_handler)(int);

void (\*sa\_sigaction)(int,siginfo\_t \*, void \*);

sigset\_t sa\_mask;

int sa\_flags;

void (\*sa\_restorer)(void);

}

简介：

该函数的作用是注册信号signum的新的行为为 \*act，旧的行为存储在

\*odlact中。该函数对信号SIGKILL SIGSTOP无效。

参数：

struct sigaction——

sa\_handler和sa\_sigaction二者只能选其一个，是具体的信号的行为函数

,也可以使用宏SIG\_DFL、SIG\_IGN代表使用用默认行为或被忽视。参数sa-

\_flags如果使用宏SA\_SIGINFO代表使用sa\_sigaction指向的行为函数，否

则默认使用sa\_handler指向的行为函数。最后一个成员sa\_restorer不使

用 。参数sa\_mask是一个信号集，当调用行为函数时，当前线程sa\_mask

中的信号将会被屏蔽，当行为函数返回时，再恢复。目的是为了达到，在

执行函数的运行期间要屏蔽sa\_mask中的信号。函数成功返回0,失败返回

-1,设置errno。

SIG\_NOCLDWAIT

## func:setitimer(2)

#include <sys/time.h>

int setitimer(int which, construct itimerval \*new\_value,

struct itimerval \*old\_value);

简介：

系统为每一个进程分配了三个定时器。该函数就是为当前进程设置定时器

，当计时结束时，计时其会发送一个信号给当前的进程，然后定时器会根

据指定的计时周期进行下一次计时。

参数：

which指定的是计时方式共三种，方式不同计时结束时的发送的信号不同。

ITIMER\_REAL——以系统的真是时间计时；

ITIMER\_VIRTUAL——从该进程开始事开始计时；

ITIMER\_PROF——记录的是该进程在用户态和内核态总的时间。

new\_value是将要设置的新的直，old\_value保存之前设置的直。

设置时间的结构体成员如下：

struct itimerval {

struct timeval it\_interval;

/\* Interval for periodic timer \*/

struct timeval it\_value;

/\* Time until next expiration \*/

};

struct timeval {

time\_t tv\_sec; /\* seconds \*/

suseconds\_t tv\_usec; /\* microseconds \*/

};

## func:sleep(3)

#include <unistd.h>

unsigned int sleep(unsigned int seconds);

简介：

函数的作用是使当前的线程进入睡眠状态seconds秒，知道时间到了或者

被一个不能忽视的信号唤醒比如SIGALRM。该函数返回的直是距离预设直

还剩多少秒，当然如果是计时时间到了当然返回的是0.

不要同时出现alarm()hesetitimer()

## func:sigqueue(3)

#include <signal.h>

int sigqueue(pid\_t pid, int sig, const union sigval value);

简介：

这个函数的作用是将一个信号sig排队发送到进程pid，好像是专门操实时

信号的，因为只有实时信号才排队。权限设置请参考kill（2）,sig为0时

可以检测pid进程是否存在。value可以做为一个跟随信号传递到那个具有

两个参数的信号行为函数所要传递的那个参数。函数成功返回0,失败返回

-1,设置errno。

## fucn:raise(3)

#include <signal>

int raise(int sig);

简介：

该函数的作用是向当前的进程或者是线程发送信号sig，如果这是一个单线

程程序等价于"kill(getpid(),sig)",如果当前是一个多线程等价于"pthr-

ead\_kill(pthread\_self(), sig)"。函数在信号的行为函数返回后才返回

。函数成功返回0,失败返回一个非0的数。

# date09

## func:pipe(2) pipe2()

#include pipe()

int pipe(int pipefd[2]);

简介：

创建一个管道文件，并且生成代表这个管道文件两端的两个文件描述符，

存储在数组pipefd中，pipefd[0]端只能读， pipefd[1]只能写。由于fork

时文件描述符是也要复制的，所以可以利用该文件进行父子进程间的通信

。函数成功返回0,失败返回-1,设置errno。

## func:mkfifo(3)

## func:msgget(3)

## func:ftok(3)

## func:msgsnd(3)

## func:msgrcv(3)

## func:msgctl()

## func:strstr(3)

## func:semget()

## func:semop()

## func:semctl()

## func:shmget()

## func:shmop()

## func:shmctl()

## func:mmap()

# date10

man 7 prhreads

## func:pthread\_create(3)

## func:othread\_self()

## func:pthread\_equal()

## func:pthread\_exit(3)

## func:pthread\_join(3)

## func:pthread\_cleanup\_push()

## func:pthread\_cleanup\_pop()

## func:pthread\_cancle()

## type:pthread\_mutex\_t

## func:pthread\_mutex\_init()

## func:pthread\_mutex\_lock()

## func:pthread\_mutex\_unlock()

## func:pthread\_mutex\_trylock()

## func:pthread\_mutex\_destroy()

## type:pthread\_cond\_t

## func:pthread\_cond\_init()

## func:pthread\_cond\_wait()

## func:pthread\_cond\_signal()

## func:pthread\_cond\_broadcast()

## func:pthread\_cond\_destroy()

# date11

## func:unlocked\_stdio(3)

## func:pthread\_cancel(3)

## func:pthread\_setcancelstate(3)

## func:pthread\_testcancel(3)

## func:pthread\_exit(3)

## key :man 7 pthreads

## func:pthread\_detach(3)

## func:pthread\_attr\_init(3);

## func:pthread\_attr\_setdetachstate(3)

## func:pthread\_attr\_destroy(3)

#include <pthread.h>

int pthread\_attr\_destroy(pthread\_attr\_t \*attr);

compile and link with -pthread

简介：

## func:pthread\_sigmask(3)

#include <signal.h>

int pthread\_sigmask(int how, const sigset\_t \*set,

sigset\_t \*oldset);

compile and link with -pthread

简介：

该函数的作用与sigprocmask的作用是一样的，不同的是该函数作用的是线

程，也就是说每个线程也有一个信号屏蔽字集，将新的信号集set中的信号

添加、移除、设置(至于是哪种操作决定于how)成信号屏蔽集中，旧的信号

屏蔽集存储到oldset中，set为NULL，则当前的信号屏蔽集不变，只是存储

到oldset中。成功返回0,失败返回-1.

## func:sigwait(3)

#include <signal.h>

int sigwait(const sigset\_t \*set, int \*sig);

简介：

这个函数的作用是暂停当前的线程直到有一个信号集set中的信号被传入，

函数会接受这个信号存储到sig中，并且将该信号从set中移除，然后解除

暂停。成功返回0,失败返回errno。`

## func:pthread\_kill(3)

#include <signal.h>

int pthread\_kill(pthread\_t thread, int sig);

compile and link with -pthread

简介：

函数的作用是在同一个进程内向指定的线程thread发送信号sig，函数成功

返回0,失败返回errno。

## func:fcntl(2)

#include <nistd.h>

#include <fcntl.h>

int fcntl(int fd, int cmd, ... /\* arg \*/);

简介：

当我们用open打开一个文件时，会指定一些打开模式，比如非阻塞，可读

写等，那如果已经打开了文件怎么修改这些属性呢？就是用fcntl这个函数

去写改。第一个参数指定的是文件描述符号，cmd是操作选项，不同的选项

功能不同，而且选项可能带参数也可能不带参数。有一个常用的选项:

F\_SETFL 后面可以根的参数有O\_RDONLY, O\_WRONLY, O\_RDWR. O\_CREAT,

O\_EXCL, O\_NOCTTY, O\_TRUNC. O\_APPEND, O\_ASYNC, O\_DIRECT,

O\_NOATIME, and O\_NONBLOCK .使用该选项时，函数成功返回0,失败返回

-1,设置errno。

# date12

## ^func:select(2)

#include <sys/select.h>

int select(int nfds, fd\_set \*readfds, fd\_set \*writefds,

fd\_set \*exceptfds, struct timeval \*timeout);

简介：

这个函数的作用是，监视readfds，writefds，exceptfds中文件描述符，

产看其是否发生了可读，可写，异常状态的发生，如果不发生，则函数会

在timeout时间内阻塞，如果时间过了，函数回以超时的状态返回。nfds的

直应该是那三个文件描述符集中的最大直的文件描述符数加1的直。这个直

是函数内部使用的，为的是指定文件描述符号的最大的个数，目前实际使

用，发现把这个数字再写的大一些也可以。当函数返回时三个文件描述符

号集会重新填入那些文件发生改变的文件描述符号。函数的返回直是发生

改变的文件的文件描述符号的个数。

## ^func:poll(2)

#include <poll.h>

int poll(struct pollfd \*fds, nfds\_t nfds, int timeout);

简介：

这个函数也是在timeout(us)时间内等待存储在fds内的nfds个文件描述符

所描述的文件达到fds.events时返回，并且将返回状态存储在fds.revents

中。也就是说fds指向的是一个数组，数组成员是struct pollfd类型，其

成员有文件描述符号fd，等待事件events，回添的响应事件状态revents，

这些事件的状态使用红定义的位图描述的。timeout内如果有等待事件发生

函数返回这些事件的数量，如果没有事件发生又超出timeout返回0,出错返

回-1,设置errno。

## func:epoll(2)

epoll\_create(2);

#include <sys/epoll.h>

int epoll\_create(int size);

简介：

调用该函数时会创建一个epoll实例，size是期望的实例文件描述符，在新

版的内核中这个参数不用指定了，只需要大于0即可。函数成功返回实例的

文件描述符号，失败返回-1,设置errno。

epoll\_ctl(2)

#include <sys/epoll.h>

int epoll\_tcl(int epfd, int op, int fd,strut epoll\_event \*event);

typedef union epoll\_data{

void \*ptr;

int fd;

uint32\_t u32;

uint64\_t u64;

} epoll\_data\_t;

struct epoll\_event{

uint32\_t events;

epoll\_data\_t data;

}

简介：

这个函数是这样使用的，将所有需要监测的文件的fd和监测事件event都存

放到实例epfd中，当然可以通过op，对epfd中的文件进行添加、删除、修

改event事件。而event的结构体中不仅有一个事件events成员，还有一个

data成员，data成员是一个联合体，一般只是用来记录fd，也就是说该事

件对应的哪个文件。函数成功返回0,失败返回-1,设置errno。

epoll\_wait(2)

#include <sys/epoll.h>

int epoll\_wait(int epfd, struct epoll\_event \*events,

int maxevents, int timeout);

简介：

这个函数的作用是在timeout时间内等待已经加入epfd中的文件发生max-

events个events事件然后返回发生events事件的个数。而且具体发生的哪

种事件和该事件对应的data成员也会回添到events中。函数超时返回0,失

败返回-1,设置errno。

theo:socket

man 7 socket

theo:ip

man 7 ip

ip的地址类型

# date13

theo:udp

man 7 udp

## func:socket(2)

#include <sys/types.h>

#include <sys/socket.h>

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

简介：

socket的作用是创建一个套接字文件，可以使用这个文件来完成不同主机

或者同一主机下的不同进程之间的通信。完成不同主机之间的进程的通信

可以使用各种通信协议。对这些协议进行分类就形成了不同的通信协议族

，domain表述使用哪一个通信协议族，protocol指定了使用这个通信协议

族中的那个协议，而type指定的是你使用的通信协议的传输层使用哪种形

式传输数据。这些协议族和协议很多，选择起来很复杂，不过万幸的是目

前只需要掌握一种，即TCP/IP.

参数：

常用的参数组合：

（1）domain——AF\_INET,type——SOCK\_STREAM/SOCK\_DGRAM,protocol——0

这种组合的意思是使用IPV4协议族，这个协议族巧了只有这一个协议

所以protocol写0，表示使用默认的就可以了。而传输层的传输方式

有两种，分别是流式传输和数据报传输。

返回直：

该函数成功返回的是一个新建的套接字文件的文件描述符，失败返回-1,

设置errno.

## func:bind(2)

#include <sys/types.h>

#include <sys/socket.h>

int bind(int sockfd, const struct sockaddr \*addr,\

socklen\_t addrlen);

简介：

该函数的作用是将套接字文件和addr中指定的本地的ip地址绑定。函数成

功返回0,失败返回-1。

参数：

表示ip地址的类型是一个类似以下结构的结构体，addrlen代表的这个结

构体的实际的长度，因为这个结构体实际可能更长，比如直接用struct-

sockaddr\_in.

struct sockaddr {

sa\_family\_t sa\_family;

char sa\_data[14];

}

## func:recvfrom(2)

#include <sys/types.h>

#include <sys/socket.h>

ssize\_t recvfrom(int sockfd, void \*buf,size\_t len, int flags,

struct sockaddr \*src\_addr, socklen\_t \*addrlen);

简介：

这个函数的作用是将套接字文件sockfd接收到的消息缓存在长度len的buf

中，如果len小了，消息将会被截断。src\_addr用来存储消息的源ip地址

，addrlen指定src\_addr的大小，addrlen应该初始化如果实际的ip地址内

容比src\_addr提供的addrlen长度大，地址信息会截断，但是实际的ip信

息长度将保存在addrlen中。flags是一些位或的操作选项，比如MSG\_DONT

WAIT,就是使该函数进行不阻塞的操作。函数成功返回抓到的数据的的字

节数，失败返回-1,设置errno。

## func:sendto(2)

#include <sys/types.h>

#include <sys.socket.h>

ssize\_t sendto(int sockfd,const void \*buf,size\_t len,int flags

const struct sockaddr \*dest\_addr, socklen\_t addrlen);

简介：

将buf中的len长度的信息发送到套接字文件sockfd，flags是一些特殊的

操作选项。这个套接字文件如果是一个流式的传输协议，后面的两个参数

src\_addr addrlen没有用,写NULL。如果是数据报式的后面两个的参数带

表的是要传输向的目标IP。

## func:connect(2)

#include <sys/types.h>

#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr \*addr\

socklen\_t addrlen);

简介：

这个函数的作用是套接子文件sockfd和ip地址为addr，ip地址长度为addr-

len创建一个连接关系。如果sockfd是一个数据报的类型，那么只有addr中

指定的地址能够受到数据，如果sockfd是一个数据流的类型，那么将尝试

把该套接子文件和一个bind 地址addr的套接字文件建立连接。函数成功返

回0,失败返回-1,设置errno。

## func:recv(2)

#include <sys/types.h>

#include <sys/socket.h>

ssize\_t recv(int sockfd, void \*buf, size\_t len, int flags);

简介：

这个函数的作用是读取套接字文件sockfd，试图读取len个字节，存到buf

中。flags是一些操作选项。比如该函数默认是阻塞的，可以通过该选项

设置成非阻塞的，没有特殊要求就写0.函数成功返回读取的字节的个数，

失败返回-1,设置errno。

## func:send(2)

#include <sys/types.h>

#include <sys/socket.h>

ssize\_t send(int sockfd, const void \*buf, size\_t len ,int flags);

简介：

这个函数的作用是写入套接字文件sockfd，试图写入len个字节，从buf

中。flags是一些操作选项。比如该函数默认是阻塞的，可以通过该选项

设置成非阻塞的，没有特殊要求就写0.函数成功返回写入的字节的个数，

失败返回-1,设置errno。

## func:htons(3)

#include <arpa/inet.h>

uint32\_t htonl(uint32\_t hostlong);

uint16\_t htonl(uint16\_t hostlong);

uint32\_t ntohl(uint32\_t netlong);

uint16\_t ntohl(uint16\_t netlong);

简介：

这几个函数的作用是，将本地字节序转换成网络字节序，或者将网络字节

序号转换成网路字节序，返回直是转换结果。之所以要转换是因为，不同

主机之间同一个数据类型所占用的字节个数不一样，数序也可能不一样。

所以上传前先有本地格式转化为网络字节序，下载时在由网路字节序号转

换为另一个机器的本地字节序号，这样才不会出错。

## func:inet\_pton(3)

#include <arpa/inet.h>

int inet\_pton(int af, const char \*src, void \*dst);

简介：

这个函数的作用是将AF\_INET或AF\_INET6族的点分十进制ip地址src转换成

网络结构体格式的ip地址，并存储到dst中。af用于指定ip的地址族，只能

选择AF\_INET或AF\_INET6中的一个。函数成功返回1,如果src写的不合适返

回0,如果af指定的不合适返回-1.

## func:inet\_ntop(3)

#include <arpa/inet.h>

const char \*inet\_ntop(int af, const void \*src,

char \*dst, socklen\_t size);

简介：

这个函数的作用是和inet\_pton类似，不同的是这个是将一个点分十进制的

地址src转换成一个af指定的网络结构体格式的ip地址，存储到dst中，si-

ze记录转换后的实际的长度，函数成功返回一个指向dst的指针，失败返回

null，设置errno。

## func:inet\_aton(3) inet\_ntoa()

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

int inet\_aton(const char \*cp,struct in\_addr \*inp);

char \*inet\_ntoa(struct in\_addr in);

简介：

inet\_aton(3)这个函数的作用是将一个IPV4的点分十进制的ip地址cp转换

成网络字节序格式，并存储到inp中。cp不合适返回0,合适返回一个非0的i

数。

inet\_ntoa(3)和inet\_aton的转换作用相反。将网络字节序的ip地址in转换

成点分十进制地址的字符串，并返回该字符串的指针。字符串存在静态区

中。

## func:setsockopt(2)

#include <sys/types.h>

#include <sys/socket.h>

int setsockopt(int sockfd,int level,int optname\

const void \*optval, socklen\_t potlen);

简介：

这个函数的功能是设置套接字文件属性的选项，因为使用socket创建套接

字文件时是没有参数提供设置套接字文件的属性的。所以这个函数的重要

显而易见了。函数的第一个参数指定的是要设置的套接字文件的文件描述

符。后面的四个参数如何设置需要man 7 ip中查找，ip指的是ipv4,因为

一开始还没有ipv6,ip就是指的ipv4.这一节将的是ipv4的网络层协议是如

和运行的。同样的选项的名字optname很多，所以为他们划分了等级level

，optval 指向的结构是用来设置选项参数的，optname不同其设置的参数

也不同，那么代表参数的结构体也就不同，所以使用了一void\*类型的指

针，当然这就必须要有一个量 potlen指定参数的结构体的大小是多少。

参数：

常用的参数组合：

对于IPV4协议来书level的选择是IPPROTO\_IP.

如果man 7 socket 查到如果level选择SOL\_SOCKET还有其他功能,如

过使用SOL\_SOCKET除非特殊说明，那么该选项的参数是一个整数。

（1）(sockfd,IPPROTO\_IP,IP\_ADD\_MEMBERSHIP,\

(void \*)ip\_mreqn,sizeof(struct ip\_mreqn))

ip\_mreqn的结构如下：

struct ip\_mreqn {

struct in\_addr imr\_multiaddr; /\* IP multicast group

address \*/

struct in\_addr imr\_address; /\* IP address of local

interface \*/

int imr\_ifindex; /\* interface index \*/

};

in\_addr的结构如下：

struct in\_addr {

uint32\_t s\_addr; /\* address in network byte order \*/

};

额外的如果要表示一个ip地址应该用如下结构表示：

struct sockaddr\_in {

sa\_family\_t sin\_family; /\* address family: AF\_INET \*/

in\_port\_t sin\_port; /\*port in network byte order \*/

struct in\_addr sin\_addr; /\* internet address \*/

};

这个组合的含义是将本地的ip加入到一个组播中imr\_multiaddr,imr-

address是本地的ip地址，imr——ifindex是本地的网卡的id号，本地

ip可用“0.0.0.0”表示，而网卡的id号可以通过if\_nametoindex()通

过网卡的名字去得到。

（2）(sockfd,SOL\_SOCKET,SO\_BROADCAT,\

int \*anyone,sizeof(int))

这组合的含义是将套接子文件sockfd设置成广播的形式，即使用该套

接子发送信息时，和主机同个路由器下的其他主机都能收到信息。

（3）(sockfd,SOL\_SOCKET,SO\_REUSEADDR,\

int \*anyone,sizeof(int))

（4）(sockfd,SOL\_SOCKET,SO\_REUSEPORT,\

int \*anyone,sizeof(int))

## func:if\_nametoindex(3)

#include <net/if.h>

unsigned int if\_nametoindex(const char \*ifname);

简介：

函数的作用是将网卡端口的名字ifname转换成代表该网卡端口的id，成功

返回该id，失败返回0,并设置errno。

## func:listen(2)

#include <sys/types.h>

#include <sys/socket.h>

int listen(int sockfd, int backlog);

简介：

该函数使套接字文件sockfd处于linsten状态。就是将一个套接字文件切换

到被动连接状态，等待其他套接字文件的connect，只有被connect后才会

完成三次握手，而backlog是指定三次握手的过程中内核维护的两个队列的

总的长度。函数成功返回0,失败返回-1,设置errno。

## func:accept(2)

#include <sys/types.h>

#include <sys/socket.h>

int accept(int sockfd,struct sockaddr \*addr,socklen\_t \*addrlen);

简介:

这个函数的作用是创建一个新的已经完成三次握手的专门用于通信的套接

字，并且返回会这个新的套接字的文件描述符，原来的套接字文件仍回到l

isten状态。这个新的套接字的另一端与谁通信，取决于sockfd，sockfd已

经帮绑定了本地的ip，经过listen已经和其他主机的套接字文件完成了co-

nnect。如果当前的内核队列中没有完成这三个动作的套接字，而sockfd选

向中也没有标记非阻塞，那么该函数会阻塞等待。如果标记了非阻塞，则

会返回一个特定的错误的宏定义。后面的两个参数用于充填，另一端连接

的套接字的ip，记得要初始化。

# date14

theo: man 7 unix

## func:unlink(2)

#include <unistd.h>

int unlink(const char \*pathname);

简介：

函数的作用是删除pathname指定的我恶见系统中的文件。

如果这是一个文件唯一的硬连接，并且没有被打开，则会删除该文件，并

且该文件之前使用的空间被释放。

如果在是唯一的硬连接，但是被进程打开了，那么这个文件会在进程结束

关闭其文件描述符号后删除

如果这是一个符号连接，删除的是这个符号

如果这是一个套接字、有名管道、设备文件，删除的只是这个文件的名字

但是打开这个文件的进程还是能够使用他

成功返回0,失败返回-1,设置errno。

## func:rmdir(2)

#include <unistd.h>

int rmdir(const char \*pathname);

简介：

删除一必须是空的路径文件，成功返回0,失败返回-1,设置errno

## func:remove(3)

#include <stdio.h>

int remove(const char \*pathname);

简介：

如果pathname是一个文件，这个函数就相当于使用unlink（2）。如果pat-

hname是一个路径，这个函数相当于使用remove。

## func:socketpair(2)

#include <sys/types.h>

#include <sys/socket.h>

int socketpair(int domain,int type,int protocol,int sv[2]);

简介：

这个函数能生成两个套接字文件，新生成的套接字文件的文件描述符号存

储在sv中。domain、type、protocol分别指定了该套接子文件的协议族、

传输曾协议、网际层协议。一般的将这三个参数配置为AF\_UNIX、SOCK\_DG-

RAM,0代表的是本地，数据报（本地的数据包传输是可靠的和使用数据流传

输效果一样），默认设置。这种配置下的用法和管道文件类似，不同的是

两个文件描述符号谁都可以作读端或写端，没有规定死。