# Linux编程思路

1. 与系统打交道的库函数、系统调用一定要做返回值检查
2. 常用的库要理解原理和倒背如流
3. 资源的竞争与释放，调用顺序也会导致竞争
4. Linux为提高效率，会延时写脏页，修改文件时，磁盘存储同步,调用fsync()函数或sync命令
5. 写函数，开关配对，申请内存与释放内存配对,谁new谁delete
6. 用函数时考虑阻塞和非阻塞
7. 释放cpu
8. 资源在多个线程使用要避免竞争，锁一释放，就会存在竞争。
9. 绘制并重审所修改bug所在函数的程序流程图，写代码前写流程图，仔细推敲分支并发情况。
10. 可重入

技巧：#define sigfillset(ptr) (\*(ptr) = ˜(sigset\_t)0, 0)——宏返回逗号后的值

原子操作思想:atomicoperation:If the operation is performed atomically, either all the s teps are performed, or none are performed.

# 信号——<signal.h>、<setjmp.h>、<stdlib.h>

原理：when a signal handler is invoked, the signal being caught is added to the current signal mask of the process. The original mask is restored when the signal handler returns.

捕获信号：void ( \*signal(int signo, void (\*func)(int)))(int);

Signo：the name of the signal；func：SIG\_IGN、SIG\_DFL、the address of a function

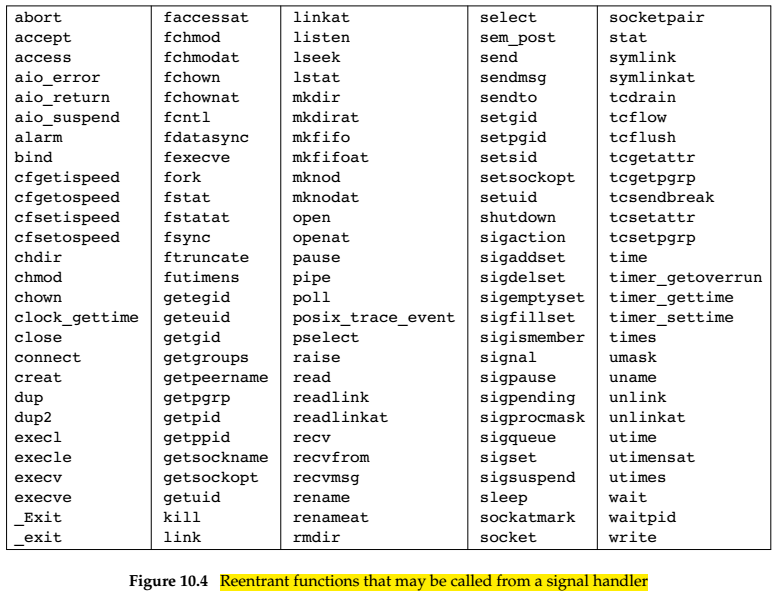
Returns: previous disposition of signal (see following) if OK, SIG\_ERR on error

中断: sig\_int()

{

signal(SIGINT, sig\_int); /\* reestablish handler for next time \*/

. . /\* process the signal ... \*/}



**signal mask**：defines the set of signals currently blocked from delivery to that process.

发送信号：int kill(pid\_t pid, int signo);

int raise(int signo); = kill(getpid(), signo);

signo:0 as the null signal, unblocked signal is delivered to the process before kill returns

Both return: 0 if OK, −1 on error

定时器：unsigned int alarm(unsigned int seconds);

Seconds: 0, the previous alarm clock is canceled; SIGALRM: terminate the process

Returns: 0 or number of seconds until previously set alarm

等待信号：int pause(void);——suspends the calling process until a signal is caught

Returns: −1 with errno set to EINTR

创建信号集：int sigemptyset(sigset\_t \*set);——zeros all bits，unblock all signal

int sigfillset(sigset\_t \*set);——turns on all the bits in the integer，block all signal

All return: 0 if OK, −1 on error

加/减信号：int sigaddset(sigset\_t \*set, int signo);

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

All return: 0 if OK, −1 on error

查询信号：int sigismember(const sigset\_t \*set, int signo);

Returns: 1 if true, 0 if false, −1 on error

设置mask：int sigprocmask(int how, const sigset\_t \*restrict set,sigset\_t \*restrict oset);——only for single-threaded processes

how: SIG\_BLOCK——after the signal occur,the handle function cannot be called;, SIG\_UNBLOCK, SIG\_SETMASK；set: The new signal mask for the process, NULL, how is ignored:no change;oset:the old set of signal

Returns: 0 if OK, −1 on error

查询挂起的信号：int sigpending(sigset\_t \*set);

set: The set of signals is returned through the set argument

Returns: 0 if OK, −1 on error

查/改信号属性：int sigaction(int signo, const struct sigaction \*restrict act, struct sigaction \*restrict oact);

Returns: 0 if OK, −1 on error

struct sigaction {

void (\*sa\_handler)(int); /\* addr of signal handler, \*/

/\* or SIG\_IGN, or SIG\_DFL \*/

sigset\_t sa\_mask; /\* additional signals to block \*/

int sa\_flags; /\* signal options, Figure 10.16 \*/

/\* alternate handler \*/

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

};

跳转：int sigsetjmp(sigjmp\_buf env, int savemask);

Savemask: nonzero, then sigsetjmp also saves the current signal mask of the process in env

Returns: 0 if called directly, nonzero if returning from a call to siglongjmp

void siglongjmp(sigjmp\_buf env, int val);

原子变量类型：sig\_atomic\_t——can be written without being interrupted

临界代码：

sigset\_t newmask, oldmask;

sigemptyset(&newmask);

sigaddset(&newmask, SIGINT);

/\* block SIGINT and save current signal mask \*/

if (sigprocmask(SIG\_BLOCK, &newmask, &oldmask) < 0)

err\_sys("SIG\_BLOCK error");

/\* critical region of code \*/

/\* restore signal mask, which unblocks SIGINT \*/

if (sigsuspend(&waitmask) != -1)

err\_sys("sigsuspend error");

/\*

\* Reset signal mask which unblocks SIGINT.

\*/

if (sigprocmask(SIG\_SETMASK, &oldmask, NULL) < 0)

err\_sys("SIG\_SETMASK error");

设置并等待信号：int sigsuspend(const sigset\_t \*sigmask);——restore the signal mask and put the

process to sleep in a single atomic operation，the process is suspended until a signal is caught or until a signal occurs that terminates the process. If a signal is caught and if the signal handler returns, then sigsuspend returns, and the signal mask of the process is set to its value before the call to sigsuspend（mask被设置为调用sissupend之前的值）

sigmask：The signal mask of the process will be set

Returns: −1 with errno set to EINTR——indicating an interrupted system call

唤醒程序：while (quitflag == 0) //wake up the main routine only when the quit signal is caught

sigsuspend(&zeromask);

quitflag = 0;

sig\_int(int signo) /\* one signal handler for SIGINT and SIGQUIT \*/

{

if (signo == SIGQUIT)

quitflag = 1; /\* set flag for main loop \*/

}

终止程序：void abort(void);——This function never returns，and send SIGABRT

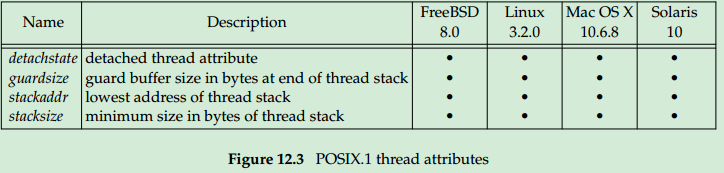
kill(getpid(), SIGABRT);

# 线程控制——<pthread.h>

创/删属性：int pthread\_attr\_init(pthread\_attr\_t \*attr);

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

return: 0 if OK , error number on failure



读/写析构：int pthread\_attr\_getdetachstate(const pthread\_attr\_t \*restrict attr,

int \*detachstate);

int pthread\_attr\_setdetachstate(pthread\_attr\_t \*attr, int detachstate);

detachstate ：PTHREAD\_CREATE\_DETACHED or THREAD\_CREATE\_JOINABLE

Both return: 0 if OK, error number on failure

读/写堆栈：int pthread\_attr\_getstack(const pthread\_attr\_t \*restrict attr,void \*\*restrict stackaddr,size\_t \*restrict stacksize);

int pthread\_attr\_setstack(pthread\_attr\_t \*attr,void \*stackaddr, size\_t tacksize);

Both return: 0 if OK, error number on failure

读/写堆栈大小：int pthread\_attr\_getstacksize(const pthread\_attr\_t \*restrict attr,size\_t \*restrict stacksize);

int pthread\_attr\_setstacksize(pthread\_attr\_t \*attr, size\_t stacksize);

Both return: 0 if OK, error number on failure

读/写保护区：int pthread\_attr\_getguardsize(const pthread\_attr\_t \*restrict attr,size\_t \*restrict guardsize);

int pthread\_attr\_setguardsize(pthread\_attr\_t \*attr, size\_t guardsize);

Both return: 0 if OK, error number on failure

创/删mutex属性：int pthread\_mutexattr\_init(pthread\_mutexattr\_t \*attr);

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

Both return: 0 if OK, error number on failure

读/写share属性:int pthread\_mutexattr\_getpshared(const pthread\_mutexattr\_t \*restrict attr,int \*restrict pshared);

int pthread\_mutexattr\_setpshared(pthread\_mutexattr\_t \*attr,int pshared);

pshared: PTHREAD\_PROCESS\_PRIVATE

Both return: 0 if OK, error number on failure

读/写robust属性：int pthread\_mutexattr\_getrobust(const pthread\_mutexattr\_t \*restrict attr,

int \*restrict robust);

int pthread\_mutexattr\_setrobust(pthread\_mutexattr\_t \*attr,int robust);

robust：PTHREAD\_MUTEX\_STALLED or PTHREAD\_MUTEX\_ROBUST

Both return: 0 if OK, error number on failure

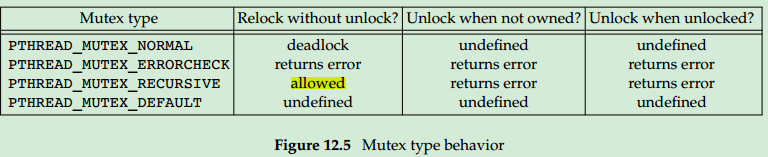
写死锁属性：int pthread\_mutex\_consistent(pthread\_mutex\_t \* mutex);

Returns: 0 if OK, error number on failure

读/写type属性：int pthread\_mutexattr\_gettype(const pthread\_mutexattr\_t \*restrict attr, int \*restrict type);

int pthread\_mutexattr\_settype(pthread\_mutexattr\_t \*attr, int type);

Both return: 0 if OK, error number on failure



创/删读写锁属性：int pthread\_rwlockattr\_init(pthread\_rwlockattr\_t \*attr);

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

Both return: 0 if OK, error number on failure

读/写share属性：int pthread\_rwlockattr\_getpshared(const pthread\_rwlockattr\_t \*restrict attr,int \*restrict pshared);

int pthread\_rwlockattr\_setpshared(pthread\_rwlockattr\_t \*attr,int pshared);

Both return: 0 if OK, error number on failure

创/删Condition属性：int pthread\_condattr\_init(pthread\_condattr\_t \*attr);

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

Both return: 0 if OK, error number on failure

读/写share属性：int pthread\_condattr\_getpshared(const pthread\_condattr\_t \*restrict attr,

int \*restrict pshared);

int pthread\_condattr\_setpshared(pthread\_condattr\_t \*attr,int pshared);

Both return: 0 if OK, error number on failure

读/写超时属性：int pthread\_condattr\_getclock(const pthread\_condattr\_t \*restrict attr,

clockid\_t \*restrict clock\_id);

int pthread\_condattr\_setclock(pthread\_condattr\_t \*attr,clockid\_t clock\_id);

Both return: 0 if OK, error number on failure

创/删障碍属性：int pthread\_barrierattr\_init(pthread\_barrierattr\_t \*attr);

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

Both return: 0 if OK, error number on failure

读/写share属性：int pthread\_barrierattr\_getpshared(const pthread\_barrierattr\_t \*restrict attr,

int \*restrict pshared);

int pthread\_barrierattr\_setpshared(pthread\_barrierattr\_t \*attr,int pshared);

Both return: 0 if OK, error number on failure

获得文件锁：int ftrylockfile(FILE \*fp);

Returns: 0 if OK, nonzero if lock can’t be acquired

void flockfile(FILE \*fp);

释放文件锁：void funlockfile(FILE \*fp);

字符锁：int getchar\_unlocked(void);

int getc\_unlocked(FILE \*fp);

Both return: the next character if OK, EOF on end of file or error

int putchar\_unlocked(int c);

int putc\_unlocked(int c, FILE \*fp);

Both return: c if OK, EOF on error

创建指定数据的key：int pthread\_key\_create(pthread\_key\_t \*keyp, void (\*destructor)(void \*));

Destructor：When the thread exits, if the data address has been set

to a non-null value, the destructor function is called

Returns: 0 if OK, error number on failure

删除key：int pthread\_key\_delete(pthread\_key\_t key);

Returns: 0 if OK, error number on failure

避免线程竞争：pthread\_once\_t initflag = PTHREAD\_ONCE\_INIT;

int pthread\_once(pthread\_once\_t \*initflag, void (\*initfn)(void));

Returns: 0 if OK, error number on failure

读/写线程数据：void \*pthread\_getspecific(pthread\_key\_t key);

Returns: thread-specific data value or NULL if no value has been associated with the key

int pthread\_setspecific(pthread\_key\_t key, const void \*value);

Returns: 0 if OK, error number on failure

使能取消点：int pthread\_setcancelstate(int state, int \*oldstate);

cancelability state：PTHREAD\_CANCEL\_ENABLE、PTHREAD\_CANCEL\_DISABLE

Returns: 0 if OK, error number on failure

设置取消点：void pthread\_testcancel(void);

设置取消同/异步：int pthread\_setcanceltype(int type, int \*oldtype);

Type: PTHREAD\_CANCEL\_DEFERRED or PTHREAD\_CANCEL\_ASYNCHRONOUS

Returns: 0 if OK, error number on failure

绑定信号：int pthread\_sigmask(int how, const sigset\_t \*restrict set,sigset\_t \*restrict oset);

how：SIG\_BLOCK、SIG\_SETMASK、SIG\_UNBLOCK

Returns: 0 if OK, error number on failure

获取信号：int sigwait(const sigset\_t \*restrict set, int \*restrict signop);

Returns: 0 if OK, error number on failure

发送信号：int pthread\_kill(pthread\_t thread, int signo);

Returns: 0 if OK, error number on failure

fork清除锁的状态：int pthread\_atfork(void (\*prepare)(void), void (\*parent)(void),void (\*child)(void));

prepare：acquire all locks；parent：unlock all the locks；child：unlock all the locks

Returns: 0 if OK, error number on failure

# 线程——<pthread.h>、<time.h>

线程号相等：int pthread\_equal(pthread\_t tid1, pthread\_t tid2);

Returns: nonzero if equal, 0 otherwise

获取线程id：pthread\_t pthread\_self(void);

Returns: the thread ID of the calling thread

创建线程：int pthread\_create(pthread\_t \*restrict tidp,const pthread\_attr\_t \*restrict attr,void \*(\*start\_rtn)(void \*), void \*restrict arg);

Returns: 0 if OK, error number on failure

退出线程：void pthread\_exit(void \*rval\_ptr);

rval\_ptr：return value

结束线程：int pthread\_cancel(pthread\_t tid);

Returns: 0 if OK, error number on failure

等待线程结束：int pthread\_join(pthread\_t thread, void \*\*rval\_ptr);

rval\_ptr ：contain the return code

Returns: 0 if OK, error number on failure

结束线程时的回掉函数：void pthread\_cleanup\_push(void (\*rtn)(void \*), void \*arg);

When call pthread\_exit or pthread\_cancel or o pthread\_cleanup\_pop with a nonzero execute argument will do the function

Rtn: cleanup function, arg:the arg of the cleanup function

删除回掉函数：void pthread\_cleanup\_pop(int execute);

execute : zero, the cleanup function is not called; otherwise, call the cleanup function

卸载线程：int pthread\_detach(pthread\_t tid);

Returns: 0 if OK, error number on failure

创建互斥锁：int pthread\_mutex\_init(pthread\_mutex\_t \*restrict mutex,const pthread\_mutexattr\_t \*restrict attr);

Pthread\_mutex f\_lock = PTHREAD\_MUTEX\_INITIALIZER;静态创建，不需要销毁

销毁互斥锁：int pthread\_mutex\_destroy(pthread\_mutex\_t \*mutex);

Both return: 0 if OK, error number on failure

获取互斥锁：int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex); ——block

int pthread\_mutex\_trylock(pthread\_mutex\_t \*mutex);——nonblack

int pthread\_mutex\_timedlock(pthread\_mutex\_t \*restrict mutex, const struct timespec \*restrict tsptr); ——timeout

All return: 0 if OK, error number on failure

释放互斥锁：int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);

return: 0 if OK, error number on failure

创建读写锁：int pthread\_rwlock\_init(pthread\_rwlock\_t \*restrict rwlock, const pthread\_rwlockattr\_t \*restrict attr);

return: 0 if OK, error number on failure

销毁读写锁：int pthread\_rwlock\_destroy(pthread\_rwlock\_t \*rwlock);

return: 0 if OK, error number on failure

获取读/写锁: int pthread\_rwlock\_rdlock(pthread\_rwlock\_t \*rwlock);——block

int pthread\_rwlock\_wrlock(pthread\_rwlock\_t \*rwlock); ——block

int pthread\_rwlock\_tryrdlock(pthread\_rwlock\_t \*rwlock); ——nonblack

int pthread\_rwlock\_trywrlock(pthread\_rwlock\_t \*rwlock); ——nonblack

int pthread\_rwlock\_timedrdlock(pthread\_rwlock\_t \*restrict rwlock,

const struct timespec \*restrict tsptr);——timeout

int pthread\_rwlock\_timedwrlock(pthread\_rwlock\_t \*restrict rwlock,

const struct timespec \*restrict tsptr); ——timeout

return: 0 if OK, error number on failure

释放读/写锁：int pthread\_rwlock\_unlock(pthread\_rwlock\_t \*rwlock)

return: 0 if OK, error number on failure

创建条件变量：int pthread\_cond\_init(pthread\_cond\_t \*restrict cond, const pthread\_condattr\_t \*restrict attr);

销毁条件变量：int pthread\_cond\_destroy(pthread\_cond\_t \*cond);

Both return: 0 if OK, error number on failure

获取条件变量：int pthread\_cond\_wait(pthread\_cond\_t \*restrict cond,pthread\_mutex\_t \*restrict mutex);

int pthread\_cond\_timedwait(pthread\_cond\_t \*restrict cond,pthread\_mutex\_t \*restrict mutex,const struct timespec \*restrict tsptr);

Both return: 0 if OK, error number on failure

发送条件变量：int pthread\_cond\_signal(pthread\_cond\_t \*cond);

int pthread\_cond\_broadcast(pthread\_cond\_t \*cond);

Both return: 0 if OK, error number on failure

创建自旋锁：int pthread\_spin\_init(pthread\_spinlock\_t \*lock, int pshared);

销毁自旋锁：int pthread\_spin\_destroy(pthread\_spinlock\_t \*lock);

Both return: 0 if OK, error number on failure

获取自旋锁：int pthread\_spin\_lock(pthread\_spinlock\_t \*lock);

int pthread\_spin\_trylock(pthread\_spinlock\_t \*lock);

All return: 0 if OK, error number on failure

释放自旋锁：int pthread\_spin\_unlock(pthread\_spinlock\_t \*lock);

return: 0 if OK, error number on failure

创建障碍：int pthread\_barrier\_init(pthread\_barrier\_t \*restrict barrier,const pthread\_barrierattr\_t \*restrict attr,unsigned int count);

return: 0 if OK, error number on failure

销毁障碍：int pthread\_barrier\_destroy(pthread\_barrier\_t \*barrier);

获得障碍：int pthread\_barrier\_wait(pthread\_barrier\_t \*barrier);

Returns: 0 or PTHREAD\_BARRIER\_SERIAL\_THREAD if OK, error number on failure

# 文件I/O——#include <fcntl.h>、#include <unistd.h>

开：int open(const char \*pathname, intoflag, ... /\* mode\_t mode \*/ );

Returns: file descriptor if OK, –1 on error

创：intcreat(const char \*pathname, mode\_t mode);

Returns: file descriptor opened for write-only if OK, –1 on error

关：int close(intfiledes);

Returns: 0 if OK, –1 on error

偏移：off\_t lseek(intfiledes, off\_t offset, int whence);

Returns: new file offset if OK, –1 on error;whence: SEEK\_SET, SEEK\_CUR, SEEK\_END

读：ssize\_t read(intfiledes, void \*buf, size\_tnbytes) ;

Returns: number of bytes read, 0 if end of file, –1 on error

写：ssize\_t write(intfiledes, const void \*buf, size\_tnbytes);

Returns: number of bytes written if OK, –1 on error

atom写：ssize\_t pread(int filedes, void \*buf, size\_t nbytes , off\_t offset);

Returns: number of bytes read, 0 if end of file, –1 on error

atom读：ssize\_t pread(int filedes, void \*buf, size\_t nbytes , off\_t offset);

Returns: number of bytes written if OK, –1 on error

复制描述符：int dup(int filedes); int dup2(int filedes, int filedes2);

Both return: new file descriptor if OK, –1 on error

同步：int fsync(int filedes);int fdatasync(int filedes); void sync(void)//写入磁盘

Returns: 0 if OK, –1 on error

c库缓冲-----fflush---------〉内核缓冲--------fsync-----〉磁盘

void sync(void);

读/写属性:int fcntl(int filedes, intcmd, ... /\* intarg \*/ ) ;

Returns: depends on cmd if OK (see following), –1 on error

# 文件和目录——#include <sys/stat.h>

读属性：int stat(const char \*restrict pathname, struct stat \*restrict buf);

intfstat(intfiledes, struct stat \*buf);

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

All three return: 0 if OK, –1 on error

读类型：stat.mode& S\_IFMT

S\_ISREG(struct stat \*restrict buf) ——regular file

S\_ISDIR(struct stat \*restrict buf) ——directory file

S\_ISCHR(struct stat \*restrict buf) ——character special file

S\_ISBLK(struct stat \*restrict buf) ——block special file

S\_ISFIFO(struct stat \*restrict buf) ——pipe or FIFO

S\_ISLNK(struct stat \*restrict buf) ——symbolic link

S\_ISSOCK(struct stat \*restrict buf) ——socket

S\_TYPEISMQ(struct stat \*restrict buf) ——message queue

S\_TYPEISSEM(struct stat \*restrict buf) ——semaphore

S\_TYPEISSHM(struct stat \*restrict buf)—— shared memory object

Allreturn: 1 if ture, 1 if false

读访问权限：int access(const char \*pathname, int mode);

Returns: 0 if OK, –1 on error；mode: R\_OK, W\_OK, X\_OK, F\_OK

设置访问权限：mode\_t umask(mode\_t cmask);

Returns: previous file mode creation mask

设置访问权限：int chmod(const char \*pathname, mode\_t mode);

Int fchmod(intfiledes, mode\_t mode);

Both return: 0 if OK, –1 on erro

设置分组：int chown(const char \*pathname, uid\_t owner, gid\_t group);

Int fchown(intfiledes, uid\_t owner, gid\_t group);

Int lchown(const char \*pathname, uid\_towner,gid\_t group);

All three return: 0 if OK, –1 on error

设置文件大小：int truncate(const char \*pathname, off\_t length);

Int ftruncate(intfiledes, off\_t length);

Both return: 0 if OK, –1 on error

访问权限标志位：Figure 4.6. The nine file access permission bits, from <sys/stat.h>

st\_mode mask Meaning

S\_IRUSR user-read

S\_IWUSR user-write

S\_IXUSR user-execute

S\_IRGRP group-read

S\_IWGRP group-write

S\_IXGRP group-execute

S\_IROTH other-read

S\_IWOTH other-write

S\_IXOTH other-execute

新建链接符：int link(const char \*existingpath, const char \*newpath);

Int linkat(intefd, const char \*existingpath, intnfd, const char \*newpath,int flag);

flag: AT\_SYMLINK\_NOFOLLOW

Both return: 0 if OK, −1 on error

删除连接符：int unlink(const char \*pathname);

Int unlinkat(intfd, const char \*pathname, int flag);

flag: AT\_REMOVEDIR

Returns: 0 if OK, –1 on error

删除文件/夹：int remove(const char \*pathname);

Returns: 0 if OK, –1 on error

重命名文件/夹:int rename(const char \*oldname, const char \*newname);

Returns: 0 if OK, –1 on error

创建符合链接：intsymlink(const char \*actualpath, const char \*sympath);

Returns: 0 if OK, –1 on error

读符号链接：ssize\_treadlink(const char\* restrict pathname,char \*restrict buf,size\_tbufsize);

Returns: number of bytes read if OK, –1 on error

改变文件时间：int futimens(intfd, conststructtimespec times[2]);

Int utimensat(intfd, const char \*path, conststructtimespec times[2],int flag);

Int utimes(const char \*pathname, conststructtimeval times[2]);

Both return: 0 if OK, −1 on error

返回目录信息：DIR \*opendir(const char \*pathname);pathname:directory name

DIR \*fdopendir(intfd);

StructDIR{

ino\_td\_ino; /\* i-node number \*/

chard\_name[]; /\* null-terminated filename \*/

}

Both return: pointer if OK, NULL on erro

读目录的文件/目录：struct dirent \*readdir(DIR \*dp);

Returns: pointer if OK, NULL at end of directory or error

关闭目录：void rewinddir(DIR \*dp);

Int closedir(DIR \*dp);

Returns: 0 if OK, −1 on error

Long telldir(DIR \*dp);

Returns: current location in directory associated with dp

Void seekdir(DIR \*dp, long loc);

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Description | Example | ls(1) option |
| st\_atim st\_mtim st\_ctim | last-access time of file data  last-modification time of file data  last-change time of i-node status | read  write  chmod, chown | -u default -c |

改变当前工作目录：int chdir(const char \*pathname);

Int fchdir(intfd);

Both return: 0 if OK, −1 on error

获取当前工作目录：char \*getcwd(char \*buf, size\_t size);

Returns: buf if OK, NULL on error

获取设备号：stat.st\_dev:containing that filename and its corresponding i-node

stat.st\_rdev:contains the device number for the actual device

major(stat.st\_rdev) minor(stat.st\_rdev)

# 标准I/O——#include <wchar.h><stdio,h>

设置字节模式：intfwide(FILE \*fp, int mode);

Returns: positive if stream is wide oriented,negative if stream is byte oriented,or 0 if stream has no orientation

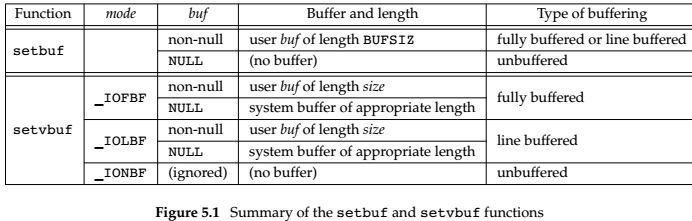
自定义buf：voidsetbuf(FILE \*restrict fp, char \*restrict buf );

intsetvbuf(FILE \*restrict fp, char \*restrict buf, intmode,size\_t size);

use only before we do any other operation on the stream

mode:\_IOFBF,\_IOLBF,\_IONBF,null size:BUFSIZ

Returns: 0 if OK, nonzero on error



将buf写入内核：int fflush(FILE \*fp); ——只是将buffer写到内核内存，要保证写到磁盘，调用fsync（）；any unflushed standard I/O buffers in memory are discarded

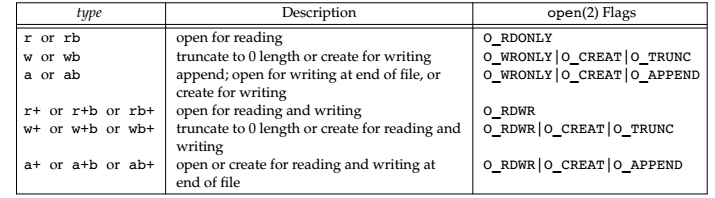
fflush(NULL); /\* all output streams to be flushed \*/

Returns: 0 if OK, EOF on error

创：FILE \*fopen(const char \*restrict pathname, const char \*restrict type);

FILE \*freopen(const char \*restrict pathname, const char \*restrict type,FILE \*restrict fp);

FILE \*fdopen(intfd, const char \*type);



Standard error is always unbuffered； All other streams are line buffered if they refer to a terminal device; otherwise,they are fully buffered.

All three return: file pointer if OK, NULL on error

关：int fclose(FILE \*fp);

Returns: 0 if OK, EOF on error

exit function directly orby returning from the main function, all standard I/O streams with unwritten buffereddata are flushed and all open standard I/O streams are closed

读byte：intgetc(FILE \*fp);

intfgetc(FILE \*fp);

intgetchar(void); = getc(stdin)

All three return: next character if OK, EOF(-1) on end of file or error

写buffer byte：intungetc(int c, FILE \*fp);

they are not written back to the underlying fileor device

Returns: c if OK, EOF on error

写文件byte：intputc(int c, FILE \*fp);

intfputc(int c, FILE \*fp);

intputchar(int c);

All three return: c if OK, EOF on error

区分EOF:intferror(FILE \*fp);

intfeof(FILE \*fp);

Both return: nonzero (true) if condition is true, 0 (false) otherwise

清除EOF标志：voidclearerr(FILE \*fp);

读line：char \*fgets(char \*restrict buf, int n, FILE \*restrict fp);buf:buf[MAXLINE] n:MAXLINE

char \*gets(char \*buf );--gets function readsfrom standard input

the bufferno more than n − 1 characters， is terminated with a null byte

Both return: buf if OK, NULL on end of file or error

写文件line：intfputs(const char \*restrict str, FILE \*restrict fp);

int puts(const char \*str);

Both return: non-negative value if OK, EOF on error

读/写结构体：size\_tfread(void \*restrict ptr, size\_t size, size\_tnobj,FILE \*restrict fp);

size\_tfwrite(const void \*restrict ptr, size\_t size, size\_tnobj,FILE \*restrict fp);

size:the size of object ;nobj:number of object

Both return: number of objects read or written

读写位置：longftell(FILE \*fp);

Returns: current file position indicator if OK, −1L on error

intfseek(FILE \*fp, long offset, int whence);

Returns: 0 if OK, −1 on error；whence :SEEK\_SET, SEEK\_CUR

void rewind(FILE \*fp);

be set to the beginning ofthe file with the rewind function

off\_t ftello(FILE \*fp);

Returns: current file position indicator if OK, (off\_t)−1 on error

Int fseeko(FILE \*fp, off\_t offset, int whence);

Returns: 0 if OK, −1 on error

Int fgetpos(FILE \*restrict fp, fpos\_t \*restrict pos);

Int fsetpos(FILE \*fp, constfpos\_t \*pos);

格式化输出：

Int printf(const char \*restrict format, ...);

Int fprintf(FILE \*restrict fp, const char \*restrict format, ...);

Int dprintf(intfd, const char \*restrict format, ...);

All three return: number of characters output if OK, negative value if output error

Int sprintf(char \*restrict buf, const char \*restrict format, ...);

Returns: number of characters stored in array if OK, negative value if encoding error

sprintf function automatically appends anull byte at the end of the array, but this null byte is not included in the return value.

Int snprintf(char \*restrict buf, size\_tn,const char \*restrict format, ...);

Returns: number of characters that would have been stored in array, if buffer was large enough, negative value if encoding error

格式转换：%[flags][fldwidth][precision][lenmodifier]convtype——the sequence %n$

represents the nth argument

int vprintf(const char \*restrict format, va\_listarg);

int vfprintf(FILE \*restrict fp, const char \*restrict format,va\_listarg);

int vdprintf(intfd, const char \*restrict format, va\_listarg);

All three return: number of characters output if OK, negative value if output error

Int vsprintf(char \*restrict buf, const char \*restrict format,va\_listarg);

Returns: number of characters stored in array if OK, negative value if encoding error

Int vsnprintf(char \*restrict buf, size\_tn,const char \*restrict format, va\_listarg);

Returns: number of characters that would have been stored in arrayif buffer was large enough, negative value if encoding error

格式化输入：

Int scanf(const char \*restrict format, ...);

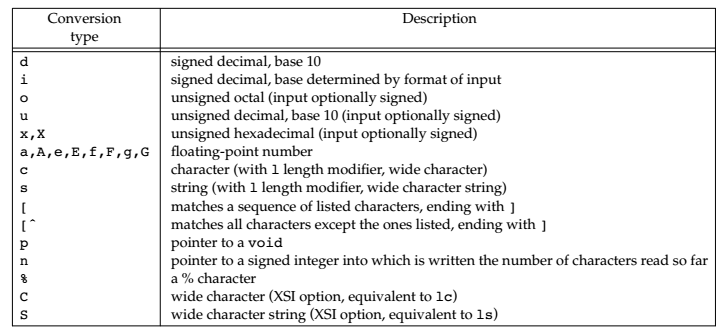
Int fscanf(FILE \*restrict fp, const char \*restrict format, ...);

Int sscanf(const char \*restrict buf, const char \*restrict format, ...);

All three return: number of input items assigned,EOF if input error or end of file before any conversion

格式转换：%[\*][fldwidth][m][lenmodifier]convtype——the sequence %n$

represents the nth argument



Int vscanf(const char \*restrict format, va\_listarg);

Int vfscanf(FILE \*restrict fp, const char \*restrict format,va\_listarg);

Int vsscanf(const char \*restrict buf, const char \*restrict format,va\_listarg);

All three return: number of input items assigned,EOF if input error or end of file before any conversion

转换为fd：int fileno(FILE \*fp);

Returns: the file descriptor associated with the stream，We need this function if we want to call the dup or fcntl functions,

建临时文件：char \*tmpname(char \*ptr);

Ptr: null or char name[L\_tmpnam];

Returns: pointer to unique pathname

FILE \*tmpfile(void);

Returns: file pointer if OK, NULL on error

创建memory stream：FILE \*fmemopen(void \*restrict buf, size\_tsize,const char \*restrict type);

Returns: stream pointer if OK, NULL on error

FILE \*open\_memstream(char \*\*bufp, size\_t \*sizep);byte oriented

FILE \*open\_wmemstream(wchar\_t \*\*bufp, size\_t \*sizep);wide oriented

Both return: stream pointer if OK, NULL on error

# 字符串——<stdlib.h>、<string.h>、<sys/types.h>、<regex.h>

str转换为数字：double atof(const char \*nptr);int atoi(const char \*nptr);long atol(const char \*nptr);——有符号数

unsigned long strtoul (const char\* str, NULL, 0);——无符号数，

Return the number跳过前面的空格字符，直到遇上数字或正负符号才开始做转换，而再遇到非数字或字符串结束时('\0')才结束转换

数字转换为str：char \*gcvt(double number，size\_tndigits，char \*buf);inttoascii(int c)

Ndigits:显示的位数 ; int c:为有符号数

大小写：string\* tolower(string str); string\* toupper(string str);

比较内存：int memcmp ( const void \*s1,const void \* s2,int n);

Return : 0:s1=s2 >0:s1>s2 ; <0:s1<s2

清0：void bzero(void \*s,int n)；所指的内存区域前n个字节，全部设为零值

查找字符：char \* index( const char \*s, int c);

char \* strchr (const char \*s,int c);

char \*strstr(const char \*source,const char \*destination);

Return :该字符c/destination首次出现的地址

char \* rindex( const char \*s,int c);

Return : 该字符c最后一次出现的地址

拷贝n个字符：void \* memcpy (void \* dest ,const void \*src, size\_t n);指针src和dest所指的内存区域不可重叠

char \*strcpy(char \*dest,const char \*src);

void \* memmove(void \*dest,const void \*src,size\_t n);指针src和dest所指的内存区域可以重叠。

All return：指向dest的指针

内存置为c：void \* memset (void \*s ,int c, size\_t n);

比较大小：intstrcasecmp (const char \*s1, const char \*s2);忽略大小写

intstrcmp(const char \*s1,const char \*s2);不忽略大小写

Return : 0:s1=s2 >0:s1>s2 ; <0:s1<s2

连接字符串：char \*strcat (char \*dest,const char \*src);

Return:dest的字符串起始地址

复制：char \* strdup( const char \*s);

Return:指向复制后的新字符串地址 or null;最后可以利用free()来释放

计算长度：size\_tstrlen (const char \*s);

分割：char \* strtok(char \*s,const char \*delim);

Return:the address of the string or NULL;用‘\0’替代delim，第一次用:char s=strtok(“wmh11wmh”,”wmh”);以后用：:char s = strtok (NULL,”wmh”)

建立正则表达式：int regcomp (regex\_t \*compiled, const char \*pattern, int cflags)

Cflags: REG\_EXTENDED 、REG\_ICASE 匹配字母时忽略大小写、

REG\_NOSUB 不用存储匹配后的结果、REG\_NEWLINE 识别换行符，这样'$'就可以从行尾开始匹配，'^'就可以从行的开头开始匹配。

Return: 0 success;-1 failed

匹配正则表达式：int regexec (regex\_t \*compiled, char \*string, size\_t nmatch, regmatch\_t matchptr [], int eflags)

typedef struct

{

regoff\_t rm\_so; //存放匹配文本串在目标串中的开始位置

regoff\_t rm\_eo; //存放结束位置

} regmatch\_t;

释放正则表达式：void regfree (regex\_t \*compiled)

输出错误：size\_t regerror (int errcode, regex\_t \*compiled, char \*buffer, size\_t length)

# 系统文件和信息——<pwd.h>、<shadow.h>、<grp.h>、<sys/utsname.h>、<sys/time.h>

关键字获取密码：struct passwd \*getpwuid(uid\_t uid);

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

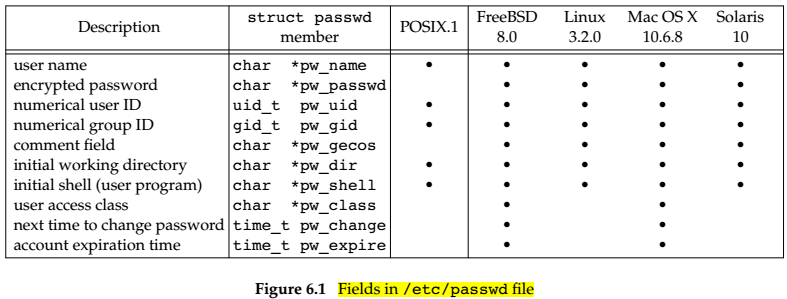
Both return: pointer if OK, NULL on error

轮询获取密码：void setpwent(void);

struct passwd \*getpwent(void);

Returns: pointer if OK, NULL on error or end of file

void endpwent(void);



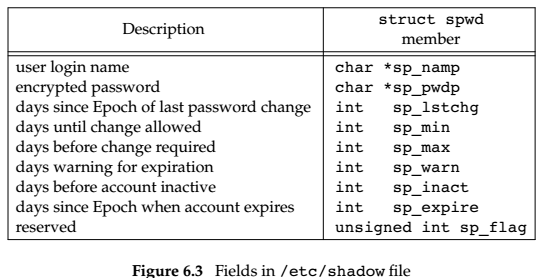
关键字获取加密密码：struct spwd \*getspnam(const char \*name);

轮询获取加密密码：void setspent(void);

struct spwd \*getspent(void);

Both return: pointer if OK, NULL on error

void endspent(void);



关键字获取组：struct group \*getgrgid(gid\_t gid);

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

Both return: pointer if OK, NULL on error

轮询获取组：struct group \*getgrent(void);

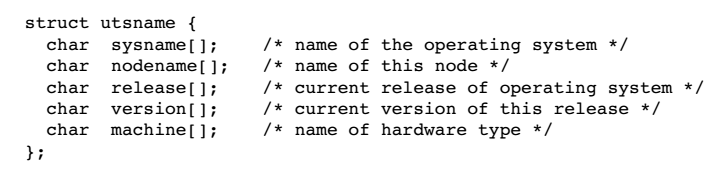
Returns: pointer if OK, NULL on error or end of file

void setgrent(void);

void endgrent(void);

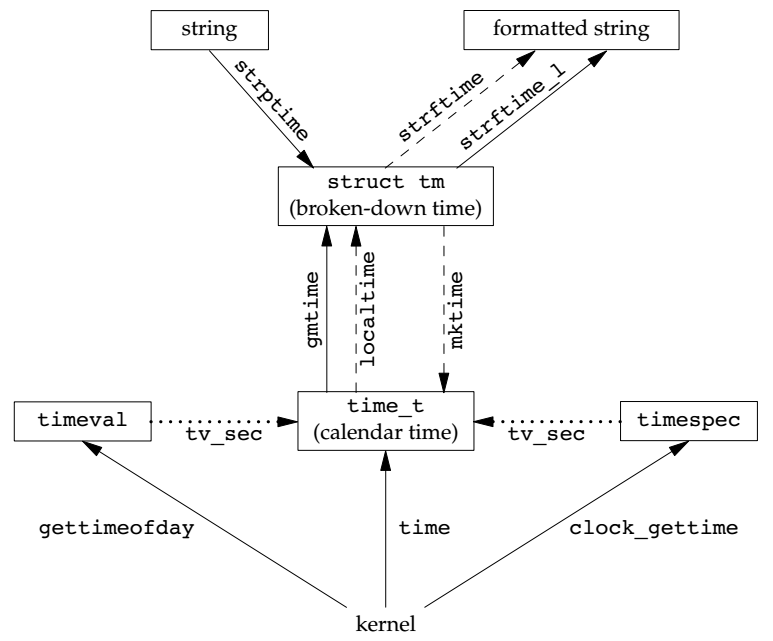
查询操作系统：int uname(struct utsname \*name);

Returns: non-negative value if OK, −1 on error



获取TCP/IP主机名：int gethostname(char \*name, int namelen);

Returns: 0 if OK, −1 on error



获取时间（秒）：time\_t time(time\_t \*calptr);

Returns: value of time if OK, −1 on error

获取不同类型时间：int clock\_gettime(clockid\_t clock\_id, struct timespec \*tsp);

Returns: 0 if OK, −1 on error

时间转换：int clock\_getres(clockid\_t clock\_id, struct timespec \*tsp);

Returns: 0 if OK, −1 on error

设置时间：int clock\_settime(clockid\_t clock\_id, const struct timespec \*tsp);

Returns: 0 if OK, −1 on error

获取时间（秒+ms）：int gettimeofday(struct timeval \*restrict tp, void \*restrict tzp);

tzp is NULL

Returns: 0 always

秒转为当地时间: struct tm \*gmtime(const time\_t \*calptr);

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

Both return: pointer to broken-down time, NULL on error

当地时间转为秒：time\_t mktime(struct tm \*tmptr);

Returns: calendar time if OK, −1 on error

当地时间转为字符串：size\_t strftime(char \*restrict buf, size\_t maxsize, const char \*restrict format, const struct tm \*restrict tmptr);

size\_t strftime\_l(char \*restrict buf, size\_t maxsize,const char \*restrict format,

const struct tm \*restrict tmptr, locale\_t locale);

Both return: number of characters stored in array if room, 0 otherwise

# 进程环境——<stdlib.h>, <unistd.h>

exit(main(argc, argv));——if the main function returns, the exit function is called

退出：void exit(int status);

void \_Exit(int status);—— prevent any standard I/O buffers

void \_exit(int status);——prevent any standard I/O buffers

注册退出函数：int atexit(void (\*func)(void));

Returns: 0 if OK, nonzero on error

内存分配heap：void \*malloc(size\_t size); ——The initial value of the memory is indeterminate

void \*calloc(size\_t nobj, size\_t size); T——he space is initialized to all 0 bits

void \*realloc(void \*ptr, size\_t newsize); ——if ptr is a null pointer, realloc behaves like malloc and allocates a region of the specified newsize.

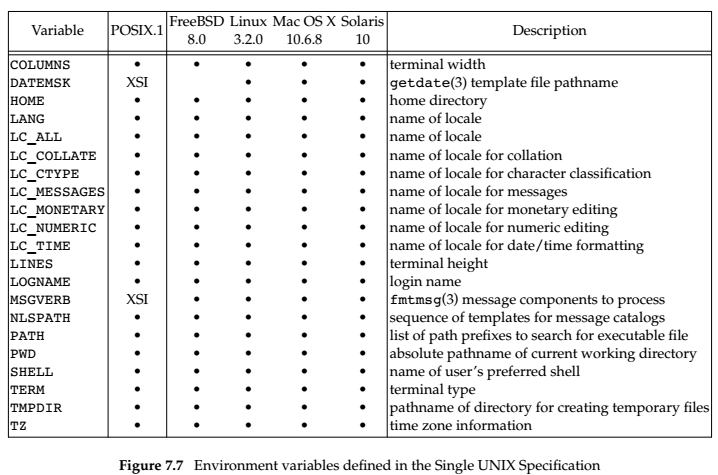
All three return: non-null pointer if OK, NULL on error

void free(void \*ptr);

内存分配stack：void alloca(size\_t size)——the memory is allocated from the stack frame of the current function，we don’t have to free the space

获取环境变量：char \*getenv(const char \*name);

Returns: pointer to value associated with name, NULL if not found



设置环境变量：int putenv(char \*str);——putenv is free to place the string passed to it directly into the environment. it would be an error to pass putenv a string allocated on the stack

Returns: 0 if OK, nonzero on error

int setenv(const char \*name, const char \*value, int rewrite); ——setenv must allocate memory to，create the name=value string from its arguments,

change the current environment and the environment of any subsequent child processes, but we couldn’t affect the environment of the parent process.

Both return: 0 if OK, −1 on error

清除环境变量：int unsetenv(const char \*name);

Both return: 0 if OK, −1 on error

跳转：int setjmp(jmp\_buf env);

Returns: 0 if called directly, nonzero if returning from a call to longjmp

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

env: be equal to the setjmp’s env ,val: the val of the setjmp return . Variables that are declared as global , volatile variables or static are left alone when longjmp is executed. when we call longjmp, we abort the other signal handler. The longjmp function is often called from a signal handler to return to the main loop of a program, instead of returning from the handler, which do not save and restore the signal mask.

读写限制：int getrlimit(int resource, struct rlimit \*rlptr);

int setrlimit(int resource, const struct rlimit \*rlptr);

Both return: 0 if OK, −1 on error

string-creation operator (#)

# 进程控制——<sys/wait.h>、<sys/resource.h>、<sys/time.h>

获取ID: pid\_t getpid(void);

Returns: process ID of calling process

pid\_t getppid(void);

Returns: parent process ID of calling process

uid\_t getuid(void);

Returns: real user ID of calling process

uid\_t geteuid(void);

Returns: effective user ID of calling process

gid\_t getgid(void);

Returns: real group ID of calling process

gid\_t getegid(void);

Returns: effective group ID of calling process

创建子进程：pid\_t fork(void);

Returns: 0 in child, process ID of child in parent, −1 on erro

pid\_t fork(void);

Returns: 0 in child, process ID of child in parent, −1 on erro，the child runs in the address space of the parent，vfork guarantees that the child runs first, until the child calls exec or exit。

等待子进程退出：pid\_t wait(int \*statloc);——wait for the child that terminates first

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

statloc：store the the termination status of the terminated process；pid:the pid of a specific process.

Options: either is 0 or is constructed from the bitwise OR of the constants

Both return: process ID if OK, 0 (see later), or −1 on error

int waitid(idtype\_t idtype, id\_t id, siginfo\_t \*infop, int options);

Returns: 0 if OK, −1 on error

pid\_t wait3(int \*statloc, int options, struct rusage \*rusage);

pid\_t wait4(pid\_t pid, int \*statloc, int options, struct rusage \*rusage);

Both return: process ID if OK, 0, or −1 on error

等待父进程退出：while (getppid() != 1) sleep(1);

执行新程序：int execl(const char \*pathname, const char \*arg0, ... /\* (char \*)0 \*/ );

int execv(const char \*pathname, char \*const argv[]);

int execle(const char \*pathname, const char \*arg0, .../\* (char \*)0, char \*const envp[] \*/ );

int execve(const char \*pathname, char \*const argv[], char \*const envp[]);

If either execlp or execvp finds an executable file using one of the path prefixes,

int execlp(const char \*filename, const char \*arg0, ... /\* (char \*)0 \*/ );

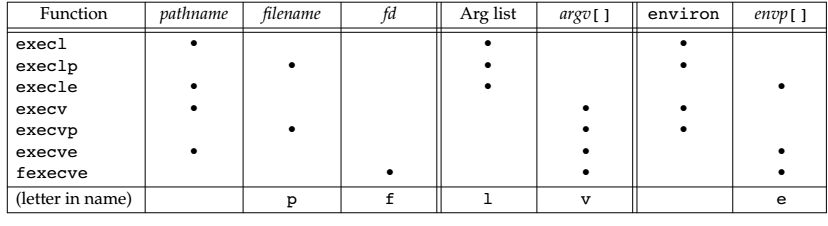
int execvp(const char \*filename, char \*const argv[]);

int fexecve(int fd, char \*const argv[], char \*const envp[]);

All seven return: −1 on error, no return on success

l stands for list null pointer is specified by the constant 0, we must cast it to a pointer and v stands for vector , e stands for environment parameter. p means that the function takes a

filename argument and uses the PATH environment variable to find the executable file, 



设置程序的ID: int setuid(uid\_t uid);——set real user ID and effective user ID

int setgid(gid\_t gid);——set real group ID and effective group ID

Both return: 0 if OK, −1 on error

int setreuid(uid\_t ruid, uid\_t euid);——set real user ID

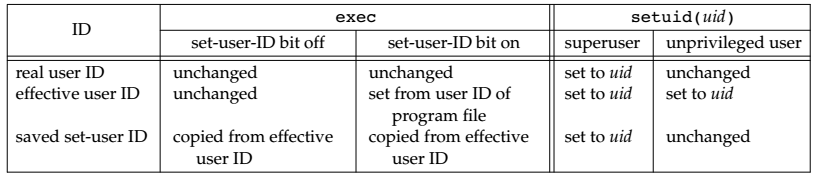
int setregid(gid\_t rgid, gid\_t egid); ;——set real group ID

Both return: 0 if OK, −1 on error

int seteuid(uid\_t uid); ——set effective user ID

int setegid(gid\_t gid); ——set effective group ID

Both return: 0 if OK, −1 on error



文件解析器：#!—and the interpreter

执行命令：int system(const char \*cmdstring);

Returns: (see below)

execl("/bin/sh", "sh", "-c", cmdstring, (char \*)0);

获取登录用户名：char \*getlogin(void);

Returns: pointer to string giving login name if OK, NULL on error

获取登录密码： char \* getpwnam(const char \* login)

Returns: pointer to string giving login name if OK, NULL on error

设置nice值：int nice(int incr);

Incr：is added to the nice value of the calling process

Returns: new nice value − NZERO if OK, −1 on error

int setpriority(int which, id\_t who, int value);

Returns: 0 if OK, −1 on error，value：is added to NZERO and this becomes the new nice value

读取nice值：int getpriority(int which, id\_t who);

Returns: nice value between −NZERO and NZERO−1 if OK, −1 on error

获取时间值：clock\_t times(struct tms \*buf );

Returns: elapsed wall clock time in clock ticks if OK, −1 on error

每秒的tick数：clock\_t sysconf(\_SC\_CLK\_TCK))

# 进程关系——<unistd.h>

获取组ID:pid\_t getpgrp(void);

Returns: process group ID of calling process

pid\_t getpgid(pid\_t pid);

Returns: process group ID if OK, −1 on error

设置组ID：int setpgid(pid\_t pid, pid\_t pgid);

Pgid: process group id ; pid: process id

Returns: 0 if OK, −1 on error

建立session：pid\_t setsid(void); ——A session is a collection of one or more process groups

Returns: process group ID if OK, −1 on error

获取session ID: pid\_t getsid(pid\_t pid);

Returns: session leader’s process group ID if OK, −1 on error

# 守护进程——<syslog.h>

void openlog(const char \*ident, int option, int facility);

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

void closelog(void);

int setlogmask(int maskpri);

Returns: previous log priority mask value

# 套接字——#include <sys/socket.h>,include <netdb.h>

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

Domain：AF\_INET,AF\_INET6，AF\_UNIX,AF\_UNSPEC

type: SOCK\_STREAM, SOCK\_DGRAM, SOCK\_SEQPACKET

Returns: file (socket) descriptor if OK, −1 on error

禁止读/写：int shutdown(intsockfd, int how);

how: SHUT\_RD, SHUT\_WR, SHUT\_RDWR

Returns: 0 if OK, −1 on error

大小端转换：uint32\_thtonl(uint32\_t hostint32);

Returns: 32-bit integer in network byte order

uint16\_thtons(uint16\_t hostint16);

Returns: 16-bit integer in network byte order

uint32\_tntohl(uint32\_t netint32);

Returns: 32-bit integer in host byte order

uint16\_tntohs(uint16\_t netint16);

Returns: 16-bit integer in host byte order

地址转换：const char \*inet\_ntop(int domain, const void \*restrict addr,char \*restrict str, socklen\_t size);

Size：INET\_ADDRSTRLEN,INET6\_ADDRSTRLEN

Returns: pointer to address string on success, NULL on error

intinet\_pton(int domain, const char \*restrict str,void \*restrict addr);

Returns: 1 on success, 0 if the format is invalid, or −1 on error

查看属性：

主机：structhostent \*gethostent(void);

Returns: pointer if OK, NULL on error

voidsethostent(intstayopen);

voidendhostent(void);

网络：structnetent \*getnetbyaddr(uint32\_t net, int type);

structnetent \*getnetbyname(const char \*name);

structnetent \*getnetent(void);

All return: pointer if OK, NULL on error

voidsetnetent(intstayopen);

voidendnetent(void

协议：structprotoent \*getprotobyname(const char \*name);

structprotoent \*getprotobynumber(int proto);

structprotoent \*getprotoent(void);

All return: pointer if OK, NULL on error

voidsetprotoent(intstayopen);

voidendprotoent(void);

服务：structservent \*getservbyname(const char \*name, const char \*proto);

structservent \*getservbyport(int port, const char \*proto);

structservent \*getservent(void);

All return: pointer if OK, NULL on error

voidsetservent(intstayopen);

voidendservent(void);

地址：intgetaddrinfo(const char \*restrict host,const char \*restrict service,conststructaddrinfo \*restrict hint,structaddrinfo \*\*restrict res);

Returns: 0 if OK, nonzero error code on error

voidfreeaddrinfo(structaddrinfo \*ai);

const char \*gai\_strerror(int error);

convert the error code returnedinto an error message.

Returns: a pointer to a string describing the error；

地址转为主机和服务：intgetnameinfo(conststructsockaddr \*restrict addr, socklen\_talen,char \*restrict host, socklen\_thostlen,char \*restrict service, socklen\_tservlen, int flags);

Returns: 0 if OK, nonzero on error

设置socket：int bind(intsockfd, conststructsockaddr \*addr, socklen\_tlen);

Returns: 0 if OK, −1 on error

查找socket：intgetsockname(intsockfd, structsockaddr \*restrict addr,socklen\_t \*restrict alenp);

Alenp:an integer containing the size ofthe sockaddr buffer.

Returns: 0 if OK, −1 on error

intgetpeername(intsockfd, structsockaddr \*restrict addr,socklen\_t \*restrict alenp);

Returns: 0 if OK, −1 on error

连接socket：int connect(int sockfd, const struct sockaddr \*addr, socklen\_tlen);

Addr：the address of the server with which we wishto communicate

Returns: 0 if OK, −1 on error

server允许连接：int listen(int sockfd, int backlog);

Backlog:the number of outstanding connect requests.

Returns: 0 if OK, −1 on error

server建立连接：int accept(int sockfd, struct sockaddr \*restrict addr,socklen\_t \*restrict len);

Addr：a bufferlarge enough to hold the addresslen：the size of the address.

Returns: new file (socket) descriptor if OK, −1 on error

发数据：ssize\_t send(intsockfd, const void \*buf, size\_tnbytes, int flags);

Returns: number of bytes sent if OK, −1 on error

ssize\_tsendto(intsockfd, const void \*buf, size\_tnbytes, intflags,conststructsockaddr \*destaddr, socklen\_tdestlen);

destaddr：a destination address to be used with connectionless sockets

Returns: number of bytes sent if OK, −1 on error

ssize\_tsendmsg(intsockfd, conststructmsghdr \*msg, int flags);

Returns: number of bytes sent if OK, −1 on error

收数据：ssize\_trecv(intsockfd, void \*buf, size\_tnbytes, int flags);

ssize\_trecvfrom(intsockfd, void \*restrict buf, size\_tlen, intflags,structsockaddr \*restrict addr,socklen\_t \*restrict addrlen);

ssize\_trecvmsg(intsockfd, structmsghdr \*msg, int flags);

Returns: length of message in bytes,0 if no messages are available and peer has done an orderly shutdown,or −1 on erro

设置socket：intsetsockopt(intsockfd, int level, int option, const void \*val,socklen\_tlen)

level :the number of the protocol that controls the option ;Len：the size ofthe object to which valpoints;val: a data structure or an integer,

Returns: 0 if OK, −1 on error

查看socket:intgetsockopt(intsockfd, int level, int option, void \*restrict val,socklen\_t \*restrict lenp);

Returns: 0 if OK, −1 on error

Server： socket——>bind——>listen——>accept

Client: socket——>bind——>connect

# 压缩并删除原文件夹（递归）

zip -rvmq ./test.zip test 中 –r 递归

tar -cvf test.tar ./test --remove-files 打包 最大

tar -zcf test.tar.gz ./test --remove-files 小

tar -jcf test.tar.bz2 ./test --remove-files 大

tar –tf test.tar ——参考文件列表

# 正则表达式——<regex.h>

**一、行定位符（^和$）**

　　行定位符就是用来描述字串的边界。“^”表示行的开始；“$”表示行的结尾。如：

　　^tm : 该表达式表示要匹配字串tm的开始位置是行头，如tm equal Tomorrow Moon就可以匹配

　　tm$ : 该表达式表示要匹配字串tm的位置是行尾，Tomorrow Moon equal tm匹配。

　　如果要匹配的字串可以出现在字符串的任意部分，那么可以直接   写成 ：tm

二、单词定界符（\b、\B）

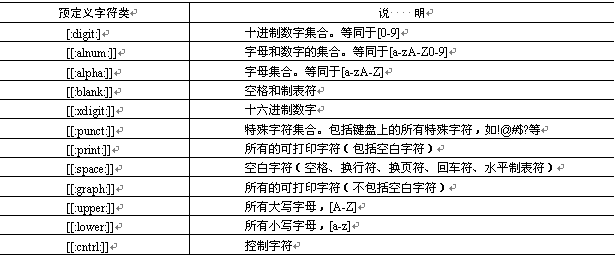
　　单词分界符\b，表示要查找的字串为一个完整的单词。如：\btm\b

　　还有一个大写的\B，意思和\b相反。它匹配的字串不能是一个完整的单词，而是其他单词或字串的一部分。如：\Btm\B

**三、字符类（[ ]）**

　　正则表达式是区分大小写的，如果要忽略大小写可使用方括号表达式“[]”。只要匹配的字符出现在方括号内，即可表示匹配成功。但要注意：一个方括号只能匹配一个字符。例如，要匹配的字串tm不区分大小写，那么该表达式应该写作如下格式：[Tt][Mm]

　　POSIX风格的预定义字符类如表所示：



**四、选择字符（|）**

 　　还有一种方法可以实现上面的匹配模式，就是使用选择字符（|）。该字符可以理解为“或”，如上例也可以写成 (T|t)(M|m)，该表达式的意思是以字母T或t开头，后面接一个字母M或m。

　　使用“[]”和使用“|”的区别在于“[]”只能匹配单个字符，而“|”可以匹配任意长度的字串。如果不怕麻烦，上例还可以写为 ：TM|tm|Tm|tM

**五、连字符（-）**

　　变量的命名规则是只能以字母和下划线开头。但这样一来，如果要使用正则表达式来匹配变量名的第一个字母，要写为 ：[a,b,c,d…A,B,C,D…]

　　这无疑是非常麻烦的，正则表达式提供了连字符“-”来解决这个问题。连字符可以表示字符的范围。如上例可以写成 ：[a-zA-Z]

**六、排除字符（[^]）**

　　上面的例子是匹配符合命名规则的变量。现在反过来，匹配不符合命名规则的变量，正则表达式提供了“^”字符。这个元字符在前面出现过，表示行的开始。而这里将会放到方括号中，表示排除的意思。

　　例如：[^a-zA-Z]，该表达式匹配的就是不以字母和下划线开头的变量名。

**七、限定符（? \* + {n,m}）**

　　对于重复出现字母或字串，可以使用限定符来实现匹配。限定符主要有6种，如表所示：



**八、点号字符（.）**

　　点字符（.）可以匹配出换行符外的任意一个字符。

　　注意：是除了换行符外的、任意的一个字符。如匹配以s开头、t结尾、中间包含一个字母的单词。

　　格式如下： ^s.t$，匹配的单词包括：sat、set、sit等。

　　再举一个实例，匹配一个单词，它的第一个字母为r，第3个字母为s，最后一个字母为t。能匹配该单词的正则表达式为：^r.s.\*t$

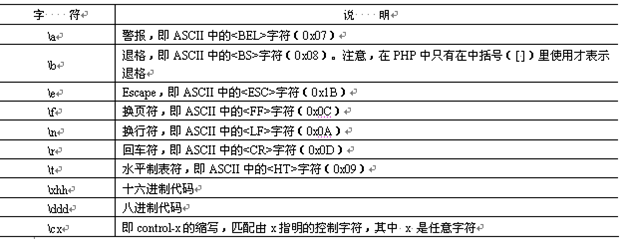
**九、转义字符（\）**

　　正则表达式中的转移字符（\）和PHP中的大同小异，都是将特殊字符（如“.”、“?”、“\”等）变为普通的字符。举一个IP地址的实例，用正则表达式匹配诸如127.0.0.1这样格式的IP地址。如果直接使用点字符，格式为：[0-9]{1,3}(.[0-9]{1,3}){3}

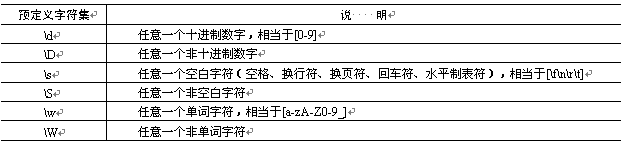
　　这显然不对，因为“.”可以匹配一个任意字符。这时，不仅是127.0.0.1这样的IP，连127101011这样的字串也会被匹配出来。所以在使用“.”时，需要使用转义字符（\）。修改后上面的正则表达式格式为： [0-9]{1,3}(\.[0-9]{1,3}){3}

**十、反斜线（\）**

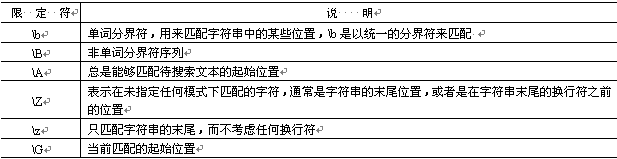
　　除了可以做转义字符外，反斜线还有其他一些功能。反斜线可以将一些不可打印的字符显示出来，如表所示：



　　还可以指定预定义字符集，如表所示：



　　反斜线还有一种功能，就是定义断言，其中已经了解过了\b、\B，其他如表所示：



**十一、括号字符（()）**

　　小括号字符的第一个作用就是可以改变限定符的作用范围，如“|”、“\*”、“^”等。来看下面的一个表达式。

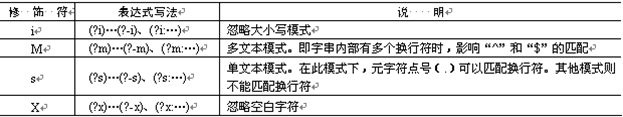
　　(thir|four)th，这个表达式的意思是匹配单词thirth或fourth，如果不使用小括号，那么就变成了匹配单词thir和fourth了。

　　小括号的第二个作用是分组，也就是子表达式。如(\.[0-9]{1,3}){3}，就是对分组(\.[0-9]{1,3})进行重复操作。后面要学到的反向引用和分组有着直接的关系。

**十二、反向引用**

**十三、模式修饰符**

　　模式修饰符的作用是设定模式。也就是规定正则表达式应该如何解释和应用。

　　不同的语言都有自己的模式设置，PHP中的主要模式如表所示：　　

# End