## UNIX基础知识

## UNIX标准及实现

### 限制（笔记1）

1. **两种类型限制是必需的：**

编译时限制（例如：短整型的最大值是什么？）

运行时限制（例如：文件名有多少个字符？）

PS：编译时限制可在头文件中定义，程序在编译时可包含这些头文件；

运行时限制则要求进程调用一个函数获得限制值；

1. **基于不同OS的不同文件系统：**

某些限制在给定实现中可能固定的（静态的在一个头文件中定义）；而在另一个实现中则可能是变动的（需要有一个运行时函数调用）；

例如：**文件名的最大字符数；**

为了解决该类问题，提供了一下3中限制：

1. 编译时限制（头文件）
2. 与文件或目录无关的运行时限制（**sysconf函数**）
3. 与文件或目录有关的运行时限制（**pathconf 和 fpathconf 函数**）

3种获取运行时限制值的函数运行：

unistd.h中的函数

/\*

\* @breif sysconf

\* @param name 标识系统限制

\* @return

\* 若 name不是一个合适的常量：返回-1，并将errno设置为EINVAL；

\* 有些name : 会返回一个变量值（>=0）或提示该值不确定；不确定的值通过-1体现，而不改变errno的值；

\*/

long sysconf ( int name ) ;

/\*

\* @breif pathconf

\* @param pathname路径名

\* @param name标识系统限制

\* @return

\* 若 name不是一个合适的常量：返回-1，并将errno设置为EINVAL；

\* 有些name : 会返回一个变量值（>=0）或提示该值不确定；不确定的值通过-1体现，而不改变errno的值；

\*/

long pathconf( const char \*pathname, int name );

/\*

\* @breif fpathconf

\* @param fd 文件描述符

\* @param name标识系统限制

\* @return

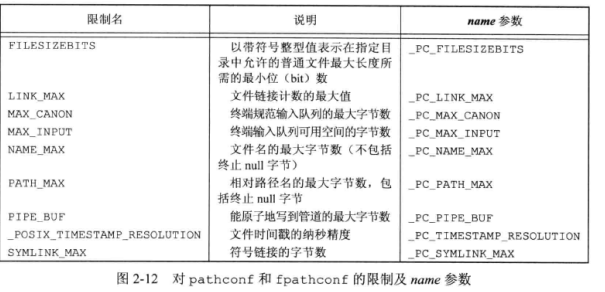
\* 若 name不是一个合适的常量：返回-1，并将errno设置为EINVAL；

\* 有些name : 会返回一个变量值（>=0）或提示该值不确定；不确定的值通过-1体现，而不改变errno的值；

\*/

long fpathconf( int fd , int name );





### ISO C限制（笔记2）

1. **ISO C定义的所有编译时限制都列在头文件<limits.h>中，见下表：**

****

第3列：用于16位整型系统（ISO C标准可接受的最小值，1的补码）。

第4列：用于32位整型Linux系统的值（2的补码）；

### POSIX 限制（笔记3）

1. **POSIX.1与接口有关的部分限制，分成下列7类：**

数值限制：LONG\_BIT , SSIZE\_MAX , WORD\_BIT

最小值：图2-8中的25个常量；

最大值：\_POSIX\_CLOCKRES\_MIN ;

运行时可以增加的值：CHARCLASS\_NAME\_MAX , COLL\_WEIGHTS\_MAX , LINE\_MAX , NGROUPS\_MAX 和 RE\_DUP\_MAX ;

运行时不变值（可能不确定）：图2-9的17个常量

其他不变值：NL\_ARGMAX , NL\_MSGMAX , NL\_SETMAX , NL\_TEXTMAX ;

路径名可变值：FILESIZEBITS , LINK\_MAX , MAX\_CANON , MAX\_INPUT , NAME\_MAX , PATH\_MAX , PIPE\_BUF , SYMLINK\_MAX ;



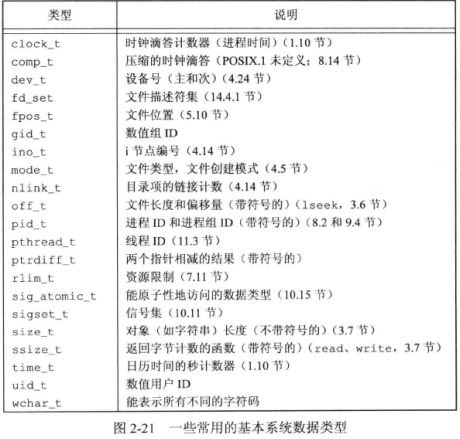


### XSI 限制（笔记4）

**…**

### 基本系统数据类型（笔记5）

1. **位于头文件<sys/types.h>中**
2. **使用基本系统数据类型，就不再考虑因系统不同而变化的程序实现细节；**

****

## 文件I/O

本章中，描述的I/O都成为不带缓冲I/O；

UNIX系统中，大多数文件I/O只需用到5个函数：open / read / write / lseek / close ;

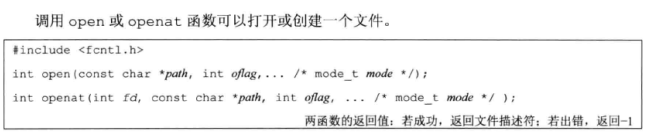
### 文件描述符（笔记1）

1. 对于内核而言，所有打开的文件都通过文件描述符引用；
2. 文件描述符是一个非负整数；
3. 当进程打开或创建一个新文件时，内核向进程返回一个文件描述符（用来标识该文件）；
4. **按照惯例：UNIX系统shell程序**
   1. 文件描述符0 (STDIN\_FILENO) 与“进程的标准输入”关联；
   2. 文件描述符1 (STDOUT\_FILENO) 与“进程的标准输出”关联；
   3. 文件描述符2 (STDERR\_FILENO) 与“进程的标准错误”关联；
5. **文件描述符的变化范围：**

0 ~ OPEN\_MAX-1 （ OPEN\_MAX ： 每个进程最大打开文件数）

### 打开文件

open 和 openat函数



Fd参数将open和openat函数区分开(三种情况)：

* + 若path为绝对路径，fd参数被忽略，openat函数相当于open函数
  + 若path 为相对路径，fd参数指定相对路径名在文件系统中的开始地址；fd参数是通过打开相对路径名所在的目录来获取；
  + 若path为相对路径，fd参数具有特殊值AT\_FDCWD。在这种情况下，路径名在当前工作目录中获取；

Path ：路径名；

Oflag ：系统定义的“文件状态标志”；

Mode ：文件的访问权限位；

Return ：返回的文件描述符一定是最小的未使用的描述符数值；

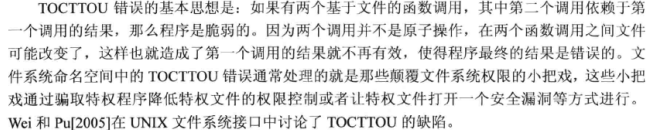
oflag 常量在头文件<fcntl.h>中定义

|  |  |
| --- | --- |
| 以下五个常量必须且只能指定一个 | |
| O\_RDONLY | 只读打开 |
| O\_WRONLY | 只写打开 |
| O\_RDWR | 读写打开 |
| O\_EXEC | 只执行打开 |
| O\_SEARCH | 只搜索打开 |

|  |  |
| --- | --- |
| 以下常量可组合使用 | |
| O\_APPEND | 追加到文件尾端 |
| O\_CLOEXEC | 把FD\_CLOEXEC常量设置为文件描述符标志（文件描述符标志暂时只有“FD\_CLOEXEC”） |
| O\_CREAT | 若此文件不存在则创建，使用该“文件状态标志位”用mode指定新文件的访问权限位（详情见：4.5） |
| O\_DIRECTORY | 若path引用的不是目录，则出错； |
| O\_EXCL | 如果同时指定了O\_CREAT，而文件已经存在，则出错；用该“文件状态标志位”可测试一个文件是否存在，若不存在，则创建此文件； |
| O\_NOCTTY | 若path引用的时终端设备，则不将该设备分配作为此进程的控制终端； |
| O\_NOFOLLOW | 若path引用的是一个符号链接，则出错； |
| O\_NONBLOCK | 若path引用的是一个FIFO、一个块特殊文件、一个字符特殊文件，则此选项为文件的本次打开操作和后续的I/O操作设置非阻塞模式； |
| O\_SYNC | 使每次write等待物理I/O操作完成，包括由该write操作引起的文件属性更新所需的I/O； |
| O\_TRUNC | 如果此文件存在，而且为只写或读写打开，则将其长度截断为0 |
| O\_TTY\_INIT | 如果打开一个还未打开的终端设备，设置非标准termios参数值。（详见第18章） |
| O\_DSYNC | 使每次write要等待物理I/O操作完成，但如果该写操作不影响读取刚写入的数据，则不需要等待文件属性被更新； |
| O\_RSYNC | 使每一个以文件描述符作为参数进行的read操作等待，直至所有对文件同一部分挂起的写操作都完成； |

openat函数解决两个问题：

* + 1. 让线程可以使用相对路径名打开目录中的文件，而不再只能打开当前工作目录。（第11章会讲解）
    2. 避免time-of-check-to-time-of-use 错误；



### 函数create

### 函数close

### 函数lseek

### 函数read

### 函数write

### I/O效率

### 文件共享

### 原子操作

### 函数dup及dup2

### 函数sync / fsync / fdatasync

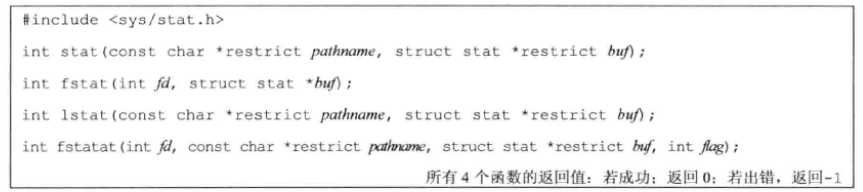
### 函数fcntl

### 函数ioctl

### /dev/fd

## 文件和目录

### 相关函数原型（笔记1）



stat / fstat / lstat / fstatat 详解

stat函数：

返回与此命名文件有关的信息结构；

fstat函数：

获得已在描述符fd上打开文件的有关信息；

lstat函数：

类似与stat , 但当命名的文件是一个符号链接时，lstat返回该符号链接的有关信息，而不是由该符号链接引用的文件的信息；

fstatat函数：

一个相对于当前打开目录（由fd参数指向 ， 相对路径）的路径名返回文件统计信息；

flag参数：控制着是否跟随着一个符号链接；

若AT\_SYMLINK\_NOFOLLOW标志被设置时，fstatat返回符号链接本身的信息；否则，返回的是符号链接所指向的实际文件的信息（默认情况）；

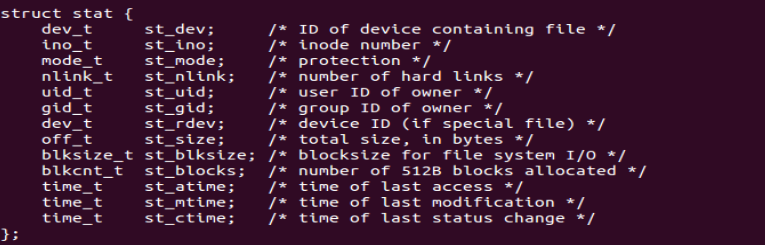
当fd为AT\_FDCWD 时：

若pathname参数是一个相对路径时，fstatat会计算相对于当前目录的pathname参数。

若pathname参数是一个绝对路径时，fstatat的fd参数会被忽略。

以上两种情况，根据flag的取值，fstatat的作用就跟stat或lstat一样；

Buf参数：是一个指针，指向一个我们必须提供的结构。stat/fstat/lstat/fstatat来填充由buf指向的结构；



使用示例：ls –l 是 stat函数使用最频繁的；

### 文件模式字（st\_mode）的应用（笔记2）

#### 文件类型

##### 文件类型分类

* **普通文件**

最常用的文件类型，这种文件包含了某种形式的数据；对于该文件内容的解释由处理该文件的应用程序进行；

二进制可执行文件需要遵循一种标准化的格式，这种格式使内核能够确定程序文本和数据的加载位置；普通文件的数据是文本还是二进制数据，对内核而言并无区别；

* **目录文件**

目录文件包含了其他文件的名字以及指向与这些文件有关的信息指针；

对于目录文件，具有读权限的任一进程都可以读该目录的内容，但只有内核可以直接写该目录；进程必须使用本章介绍的函数才能更改目录。

* **块特殊文件**

这种类型的文件提供对设备（如磁盘）带缓冲的访问，每次访问以固定长度为单位进行；

* **字符特殊文件**

这种类型的文件提供对设备不带缓冲的访问，每次访问长度可变。系统中的所有设备要么是“字符特殊文件”，要么是“块特殊文件”；

* **FIFO**

这种类型的文件用于进程间通信，有时也称为“命名管道”；

* **套接字（socket）**

这种类型的文件用于进程间的网络通信。套接字也可用于在一台宿主机上进程之间的非网络通信；

* **符号链接**

这种类型的文件指向另一个文件。

##### 文件类型获取

文件类型信息包含在stat结构中的st\_mode成员中；

两种获取文件类型的方法：

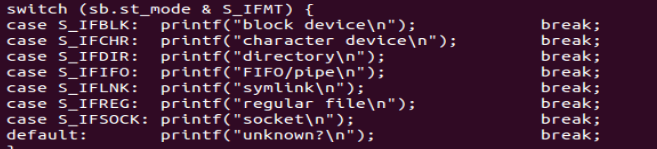
* 方法一：使用指定的宏

宏的参数为st\_mode ;

****

* 方法二：使用屏蔽字S\_IFMT

st\_mode 与 S\_IFMT进行&（逻辑与）

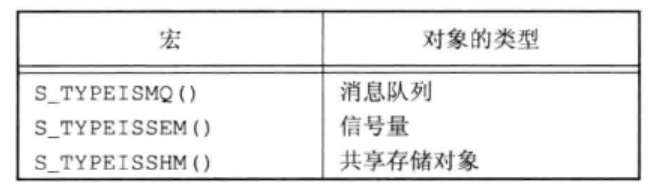
****

##### IPC对象类型获取

POSIX.1标准允许将进程间通信（IPC）对象说明为文件；

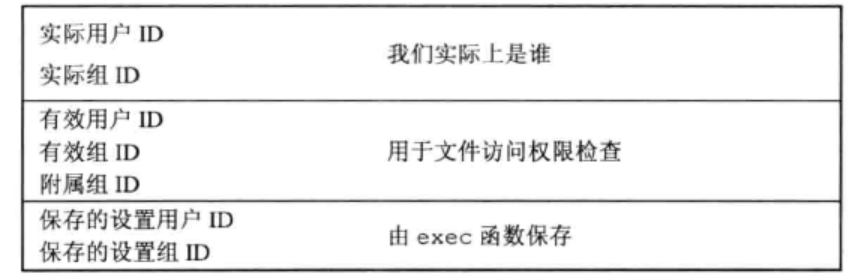
确定IPC对象的类型（如：消息队列 / 信号量）

宏的参数为：指向stat结构的指针

****

#### 设置用户ID和设置组ID

进程相关联的ID



实际用户ID / 实际组ID

取自口令文件中的登录项；root用户有方法改变它（详见8）

有效用户ID / 有效组ID / 附属组ID

决定文件访问权限；

保存设置用户ID / 保存设置组ID

执行一个程序时，包含了有效用户ID / 有效组ID的副本；（详见8 – setuid()）

以下三点注意：

* 一般 有效用户ID 等于 实际用户ID ，有效组ID 等于 实际组ID；
* 可在文件模式字（st\_mode）中设置一个特殊标志；（ 当执行此文件时，可以将进程的 “有效用户ID” , 设置成 “文件所有者用户ID( st\_uid -- 设置用户ID)” ， 将进程的”有效组ID” , 设置成 “文件组所有者ID( st\_gid -- 设置组ID)” ）;设置用户ID操作方式：chmod u+s
* “设置用户ID”及“设置组ID”都包含在文件的st\_mode值中，这两位可分别使用常量S\_ISUID 和 S\_ISGID测试；

#### 文件访问权限

内核以进程的“有效用户ID”和“有效组ID”为访问权限进行访问文件的；

所有文件类型都有文件访问权限；

文件访问权限如下：（ u(user) / g(group) / o (other) ）



3类访问权限以各种方式由不同的函数使用：

1. 打开任意类型文件时，对该名字包含的每一个目录，包括它可能隐含的当前工作目录都具有执行权限。（这就是为何 “执行权限位”被称为“搜索位”的原因）；

如：打开/usr/include/stdio.h ; 需要对 / ， /usr ， /usr/include三个目录都具有执行权限；然后在根据“文件本身的适当权限”，选取以何种方式打开（读写 ， 只读 ， 只写等）；

注意：对目录来说：读权限和执行权限意义是不同的。读权限，允许我们读目录，获得在该目录中所有文件名列表；执行权限，当访问一个文件时，执行权限可使我们通过该文件所在的目录（另一个例子：PATH变量，指定一个不具有执行权限的目录，那shell绝不会在该目录下找到可执行文件）；

1. 对于一个文件的读权限，决定了我们是否能够打开现有文件进行读操作；这与open函数的O\_RDONLY和O\_RDWR标志有关；
2. 对于一个文件的写权限，决定了我们是否能够打开现有文件进行写操作；这与open函数的O\_WRONLY和O\_RDWR标志有关；
3. 为了在open函数中，对一个文件指定O\_TRUNC标志必须对该文件具有写权限；
4. 为了在一个目录中创建一个新文件，必须对该目录具有写权限和执行权限；（执行权限：见第一条规则）
5. 为了删除一个现有文件，必须对包含该文件的目录具有写权限和执行权限。对该文件本身则不需要读、写权限；
6. 如果用7个exec函数（见8.10节）中的任何一个执行某个文件，都必须对该文件具有执行权限，该文件还必须是一个普通文件；

进程每次打开、创建、删除一个文件时，内核就进行文件访问权限测试；

测试范围：

1. 文件所有者（st\_uid和st\_gid -- 文件性质）；
2. 进程的有效ID（有效用户ID和有效组ID -- 进程性质 ）；
3. 进程的附属组ID（若支持的话 – 进程性质）；

内核进行文件访问权限测试具体如下：

1. 若进程的有效用户ID是0（超级用户），则允许访问。文件系统对超级用户（root）不设访问权限；
2. 若进程的有效用户ID等于文件的所有者ID（也就是进程拥有此文件），那么如果所有者适当的访问权限位被设置，则允许访问；否则，拒绝访问；（适当访问权限位：若进程以读方式打开该文件，用户读位，应为1；若进程以写方式打开该文件，用户写位，应为1；若进程以执行方式打开该文件，用户执行位，应为1；）
3. 若进程有效组ID或进程的附属组ID之一等于文件的组ID，那么如果组适当的访问权限位被设置，则允许访问；否则，拒绝访问；
4. 若其他用户适当的访问权限位被设置，则允许访问；否则拒绝访问；

以上权限测试的执行顺序：

* + - 若进程拥有此文件（第2步），则按用户访问权限批准或拒绝该进程对文件的访问 ----- 不查看组访问权限；
    - 若进程并不拥有该文件，但进程属于某个适当的组，则按组访问权限批准或拒绝该进程对文件的访问 ----- 不查看其他用户的访问权限。

#### 新文件和新目录的所有权

新文件和新目录的所有权规则相同；

新文件所有权规则：

* 新文件的用户ID 设置为 创建文件进程的有效用户ID；
* 新文件的组ID ：

可以是创建文件进程的有效组ID;

可以是它所在目录的组ID;

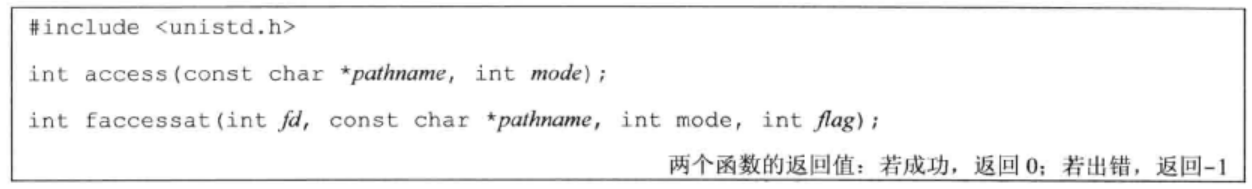
Linux 3.2.0规则：新文件的组ID取决于它所在目录的 设置组ID是否被设置；（mkdir函数自动传递一个 “目录的设置组ID位” ）

若已被设置：新文件组ID设置为 目录的组ID；

若未被设置：新文件的组ID 设置为 创建文件进程的有效组ID；

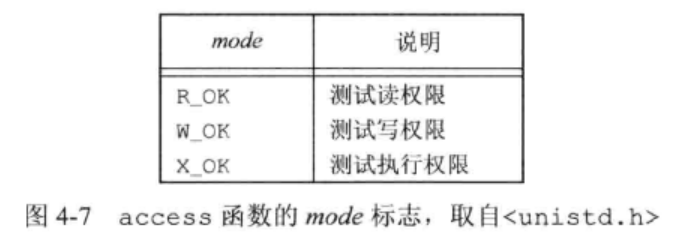
#### 函数access 和 faccessat

access 和faccessat 函数作用：按实际用户ID和实际组ID进行访问权限测试的；



Pathname（测试文件）已存在，mode为F\_OK；

Pathname（测试文件）不存在，mode为以下常量的按位或；（？？？？）



flag 参数：用于改变faccessat的行为，若flag设置AT\_EACCESS，访问检查用调用进程的有效用户ID和有效组ID，而不是实际用户ID和实际组ID；

faccessat 与access 函数在以下两种情况下是相同的：

* pathname参数为绝对路径；
* fd参数为AT\_FDCWD，pathname参数为相对路径；（否则，faccessat 计算相对于打开目录的 pathname ,并由fd参数执向其内存 ）