## 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

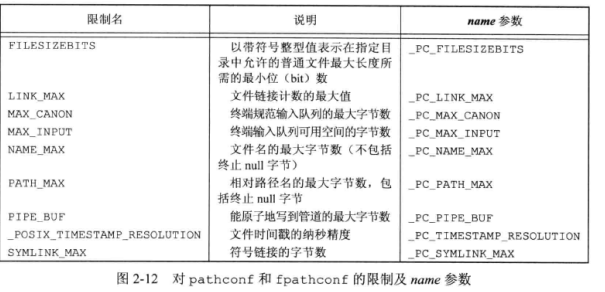
\* 若 name不是一个合适的常量：返回-1，并将errno设置为EINVAL；

\* 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 ;



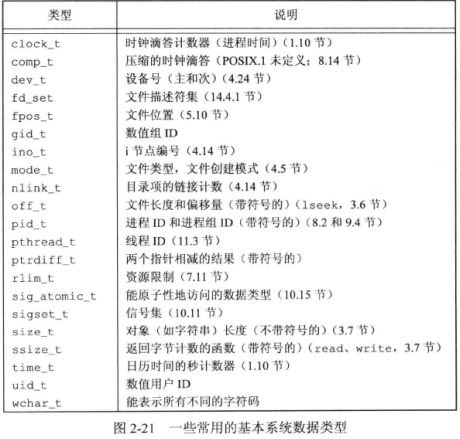


### XSI 限制（笔记4）

**…**

### 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 ;

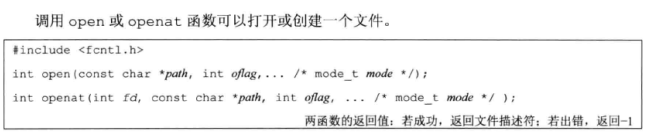
### 文件描述符（笔记1）

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）

1. **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；

