# C创建目录\_mkdir()函数

## 一、创建目录

两种方式，第二种可以设置权限

（1）direct.h头文件

int mkdir(const char \* path, mode\_t mode);

用法：int mkdir(const char \* dirname);

头文件库：direct.h

返回值：创建一个目录，成功返回0，失败返回-1

（2）Linux下mkdir函数

头文件库：

#include <sys/stat.h>

#include <sys/types.h>

原型： int mkdir(const char \*pathname, mode\_t mode);

函数说明： mkdir()函数以mode方式创建一个以参数pathname命名的目录，mode定义新创建目录的权限。

返回值： 若目录创建成功，则返回0；否则返回-1，并将错误记录到全局变量errno中。

mode方式：

S\_IRWXU 00700权限，代表该文件所有者拥有读，写和执行操作的权限

S\_IRUSR(S\_IREAD) 00400权限，代表该文件所有者拥有可读的权限

S\_IWUSR(S\_IWRITE) 00200权限，代表该文件所有者拥有可写的权限

S\_IXUSR(S\_IEXEC) 00100权限，代表该文件所有者拥有执行的权限

S\_IRWXG 00070权限，代表该文件用户组拥有读，写和执行操作的权限

S\_IRGRP 00040权限，代表该文件用户组拥有可读的权限

S\_IWGRP 00020权限，代表该文件用户组拥有可写的权限

S\_IXGRP 00010权限，代表该文件用户组拥有执行的权限

S\_IRWXO 00007权限，代表其他用户拥有读，写和执行操作的权限

S\_IROTH 00004权限，代表其他用户拥有可读的权限

S\_IWOTH 00002权限，代表其他用户拥有可写的权限

S\_IXOTH 00001权限，代表其他用户拥有执行的权限

代码

mkdir("./prognew", S\_IRWXU | S\_IRWXG | S\_IROTH | S\_IXOTH);

I.用int access(const char \*pathname, int mode); 判断有没有此文件或目录 --它区别不出这是文件还是目录

II.用int stat(const char \*file\_name, struct stat \*buf); 判断该文件或目录是否存在 ；得到st\_mode，然后判断是不是目录文件。

stat()系统调用看是否成功，不成功就不存在，成功判断返回的st\_mode是否是一个文件夹。

## 二、Liunx c关于目录是否存在，新建目录等操作

（1）创建目录

#include <sys/stat.h>

#include <sys/types.h>

int mkdir(const char \*pathname, mode\_t mode);

运用条件：只能在已存在的目录下建立一级子目录

返回值： 返回0表示成功，返回-1表述出错。

mode 表示新目录的权限，可以取以下值：

其中，mode就用0777，0755这种形式。

（2）判断一个目录是否存在

可以使用opendir来判断，这是比较简单的办法。

#include <sys/types.h>

#include <dirent.h>

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

The opendir() function opens a directory stream corresponding to the directory name, and returns a pointer to the directory stream. The stream is positioned at the first entry in the directory.

代码

#include <sys/types.h>

#include <sys/stat.h>

#include <dirent.h>

#include <cstddef>

int main()

{

if(NULL==opendir("/home/Test/data/html"))

mkdir("/home/Test/data/html",0775);

return 0;

}

## 三、更改权限

相关函数：fchmod, stat, open, chown

头文件：#include <sys/types.h> #include <sys/stat.h>

定义函数：int chmod(const char \* path, mode\_t mode);

函数说明：chmod()会依参数mode 权限来更改参数path 指定文件的权限。

返回值：权限改变成功返回0, 失败返回-1, 错误原因存于errno.

参数 mode 有下列数种组合：

1、S\_ISUID 04000 文件的 (set user-id on execution)位

2、S\_ISGID 02000 文件的 (set group-id on execution)位

3、S\_ISVTX 01000 文件的sticky 位

4、S\_IRUSR (S\_IREAD) 00400 文件所有者具可读取权限

5、S\_IWUSR (S\_IWRITE)00200 文件所有者具可写入权限

6、S\_IXUSR (S\_IEXEC) 00100 文件所有者具可执行权限

7、S\_IRGRP 00040 用户组具可读取权限

8、S\_IWGRP 00020 用户组具可写入权限

9、S\_IXGRP 00010 用户组具可执行权限

10、S\_IROTH 00004 其他用户具可读取权限

11、S\_IWOTH 00002 其他用户具可写入权限

12、S\_IXOTH 00001 其他用户具可执行权限

注：只有该文件的所有者或有效用户识别码为0，才可以修改该文件权限。

基于系统安全，如果欲将数据写入一执行文件，而该执行文件具有S\_ISUID 或S\_ISGID 权限，则这两个位会被清除。如果一目录具有S\_ISUID 位权限，表示在此目录下只有该文件的所有者或root 可以删除该文件。

错误代码：

1、EPERM 进程的有效用户识别码与欲修改权限的文件拥有者不同, 而且也不具root 权限.

2、EACCESS 参数path 所指定的文件无法存取.

3、EROFS 欲写入权限的文件存在于只读文件系统内.

4、EFAULT 参数path 指针超出可存取内存空间.

5、EINVAL 参数mode 不正确

6、ENAMETOOLONG 参数path 太长

7、ENOENT 指定的文件不存在

8、ENOTDIR 参数path 路径并非一目录

9、ENOMEM 核心内存不足

10、ELOOP 参数path 有过多符号连接问题.

11、EIO I/O 存取错误

范例

/\* 将/etc/passwd 文件权限设成S\_IRUSR|S\_IWUSR|S\_IRGRP|S\_IROTH \*/

#include <sys/types.h>

#include <sys/stat.h>

main()

{

chmod("/etc/passwd", S\_IRUSR|S\_IWUSR|S\_IRGRP|S\_IROTH);

}