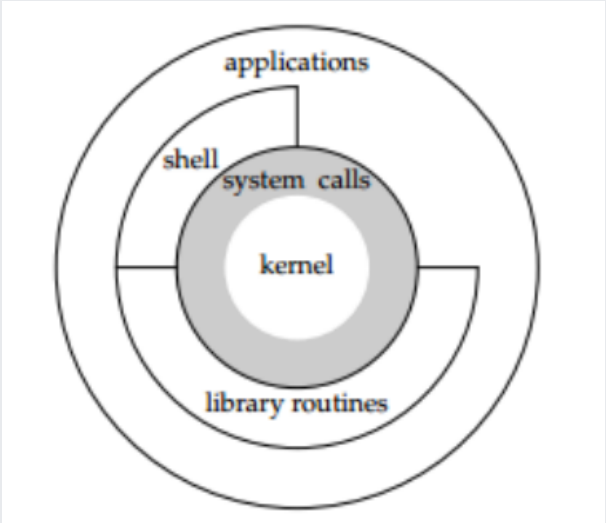
Linux_4_文件

从现在开始，我们就正式进入了 Linux 系统编程的学习了。在学习之前，我们先来回顾下 Linux 系统的结构：



前面，我们学习了和文件相关的一些 shell 命令；接下来这一章，我们就来学习和文件相关的库函数和系统调用。

1 目录相关操作

首先，我们来看和目录相关的操作。

获取当前工作目录

我们可以调用库函数 `getcwd` 获取当前工作目录的绝对路径：

```
#include <unistd.h>

char *getcwd(char *buf, size_t size);
```

参数

buf：指向存放当放当前目录的数组

size：数组的大小

返回值

成功：返回包含当前工作目录的字符串。如果buf不为NULL，即返回buf。

失败：返回NULL，并设置errno。

如果传入的 buf 为 `NULL`，且 size 为 0，则 `getcwd` 会调用 `malloc` 申请合适大小的内存空间，填入当前工作目录的绝对路径，然后返回 `malloc` 申请的空间的地址。注意：`getcwd` 不负责 `free` 申请的空间，`free` 是调用者的职责。

示例：

```
// func.h, 将func.h移动到/usr/include目录下
// 里面包含了要用到的头文件，以及一些常用的宏函数，方便学习使用。
#ifndef __WD_FUNC_H
#define __WD_FUNC_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define ARGS_CHECK(argc, n) {               \
    if (argc != n) {                        \
        fprintf(stderr, "ERROR: expected %d arguments.\n", n);      \
        exit(1);                            \
    }                                       \
}

#define ERROR_CHECK(retval, val, msg)   {   \
    if (retval == val) {                    \
        perror(msg);                        \
        exit(1);                            \
    }                                       \
}

#define SIZE(a) (sizeof(a)/sizeof(a[0]))

#endif
```

```c
// getcwd.c
#include <func.h>

int main(void) {
    // char buf[10];
    // 由系统调用malloc，动态申请内存空间
    char* cwd = getcwd(NULL, 0);
    ERROR_CHECK(cwd, NULL, "getcwd");

    puts(cwd);
    // 由调用者负责释放申请的内存空间
    free(cwd);
    return 0;
}
```

改变当前工作目录

`chdir` 函数可以改变当前工作目录。

```c
#include <unistd.h>

int chdir(const char *path);
参数
    path：改变后的路径。
返回值
    成功：返回0。
    失败：返回-1，并设置errno。
```

示例：

```c
// chdir.c
#include <func.h>

int main(int argc, char* argv[]) {
    // ./chdir dir
    ARGS_CHECK(argc, 2);

    char buf[256];
    getcwd(buf, SIZE(buf)); // 获取当前工作目录
    puts(buf);              // 打印当前工作目录

    int ret = chdir(argv[1]);   // 改变当前工作目录
    ERROR_CHECK(ret, -1, "chdir");
    getcwd(buf, SIZE(buf));
```

```
    puts(buf);

    return 0;
}
```

注意：当前工作目录是进程的属性，也就是说每一个进程都有自己的当前工作目录。且父进程创建(fork)子进程的时候，子进程会继承父进程的当前工作目录。(PS：进程相关知识后面会详细讲解~)

创建目录

`mkdir` 函数可以用来创建目录。

```
#include <sys/stat.h>
#include <sys/types.h>

int mkdir(const char *pathname, mode_t mode);
参数
    pathname：要创建目录的路径
    mode：目录的权限位，会受文件创建掩码umask的影响，实际的权限为(mode & ~umask & 0777)
返回值
    成功：返回0
    失败：返回-1，并设置errno
```

```
// mkdir.c
#include <func.h>

int main(int argc, char* argv[]) {
    // ./mkdir dir mode
    ARGS_CHECK(argc, 3);

    mode_t mode;
    sscanf(argv[2], "%o", &mode);
    // 创建的目录会受文件创建掩码的影响
    int ret = mkdir(argv[1], mode);
    ERROR_CHECK(ret, -1, "mkdir");

    return 0;
}
```

改变目录权限

`chmod` 不仅仅可以改变文件的权限，自然也可以改变目录的权限。

```
#include <sys/stat.h>

int chmod(const char *pathname, mode_t mode);
参数
    pathname：要改变权限的文件的路径
    mode：修改后的权限位
返回值
    成功：返回0
    失败：返回-1，并设置errno
```

示例：

```
// chmod.c
#include <func.h>

int main(int argc, char* argv) {
    // ./chmod file mode
    ARGS_CHECK(argc, 3);
    mode_t mode;
    // 将字符串转换为整数
    sscanf(argv[2], "%o", &mode);

    int ret = chmod(argv[1], mode);
    ERROR_CHECK(ret, -1, "chmod");

    return 0;
}
```

删除空目录

`rmdir` 可以删除空目录。

```
#include <unistd.h>

int rmdir(const char *pathname);
参数
    pathname：要删除的目录
返回值
    成功：返回0
    失败：返回-1，并设置errno
```

```c
// rmdir.c
#include <func.h>

int main(int argc, char* argv[]) {
    // ./rmdir dir
    ARGS_CHECK(argc, 2);

    int ret = rmdir(argv[1]);
    ERROR_CHECK(ret, -1, "rmdir");

    return 0;
}
```

1.1 目录流

使用目录流，可以查看目录中的内容。在讲目录流之前，我们一起来回顾一下流模型："流"类似工厂中的流水线，顺序访问流中的数据时，程序员是不需要 care 位置的。

前面我们学习了文件流，文件流中的基本单位是字符或字节。现在我们来学习目录流，目录流中的基本单位是目录项。如下图所示：

目录流 VS 文件流

文件流	目录流
fopen	opendir
fclose	closedir
fread	readdir
fwrite	×
ftell	telldir
fseek	seekdir
rewind	rewinddir

`opendir` 可以打开一个目录，得到一个指向目录流的指针 DIR*。

```
#include <sys/types.h>
#include <dirent.h>

DIR *opendir(const char *name);
参数
    name: 目录路径
返回值
    成功: 返回指向目录流的指针。
    失败: 返回NULL，并设置errno。
```

`closedir` 关闭目录流。

```
#include <sys/types.h>
#include <dirent.h>

int closedir(DIR *dirp);
参数
    dirp: 指向目录流的指针
返回值
    成功: 返回0
    失败: 返回-1，并设置errno
```

`readdir` 读目录流，得到指向下一个目录项的指针。

```
#include <dirent.h>

struct dirent *readdir(DIR *dirp);
参数
    dirp: 指向目录流的指针
返回值
    成功: 返回指向结构体dirent的指针；如果读到流的末尾，返回NULL，不改变errno的值。
    失败: 返回NULL，并设置errno

结构体dirent的定义如下:
    struct dirent {
        ino_t   d_ino;              /* inode编号 */
        off_t   d_off;
        unsigned short d_reclen;    /* 结构体的长度(d_name在有些实现上是一个可变长数组) */
        unsigned char  d_type;      /* 文件的类型 */
        char           d_name[256]; /* 文件名 */
    };
    d_type的可选值如下:
```

```
        DT_BLK      This is a block device.
        DT_CHR      This is a character device.
        DT_DIR      This is a directory.
        DT_FIFO     This is a named pipe (FIFO).
        DT_LNK      This is a symbolic link.
        DT_REG      This is a regular file.
        DT_SOCK     This is a UNIX domain socket.
        DT_UNKNOWN  The file type could not be determined.
```

`seekdir` 可以移动目录流中的位置。

```
$ man seekdir
    seekdir - set the position of the next readdir() call in the
directory stream.
```

```
#include <dirent.h>

void seekdir(DIR *dirp, long loc);
参数
    dirp: 目录流
    loc:  位置，是前面调用telldir函数的返回值。
返回值
    void
```

`telldir`：返回目录流中现在的位置。

```
#include <dirent.h>

long telldir(DIR *dirp);
参数
    dirp: 目录流
返回值
    成功：返回目录流中现在的位置。
    失败：返回-1，并设置errno。
```

`rewinddir`：重置目录流，即移动到目录流的起始位置。

```
#include <sys/types.h>
#include <dirent.h>

void rewinddir(DIR *dirp)
参数
    dirp: 目录流
返回值
    void
```

示例：

```c
#include <func.h>

int main(int argc, char *argv[])
{
    DIR* pdir = opendir(".");
    ERROR_CHECK(pdir, NULL, "opendir");

    long loc = telldir(pdir);
    struct dirent* pdirent;
    errno = 0;
    while(1) {
        // 事先记录目录流的位置
        long tmploc = telldir(pdir);
        pdirent = readdir(pdir);
        if (pdirent == NULL) {
            break;
        }
        printf("%s ", pdirent->d_name);
        if (strcmp(pdirent->d_name, "Makefile") == 0) {
            loc = tmploc;
        }
    }
    printf("\n");
    // 判断是否发生了错误
    if (errno != 0) {
        perror("readdir");
        exit(1);
    }

    printf("------------------------\n");

    seekdir(pdir, loc);
    pdirent = readdir(pdir);
    puts(pdirent->d_name);

    printf("------------------------\n");

    rewinddir(pdir);
    pdirent = readdir(pdir);
    puts(pdirent->d_name);
```

```
    // 关闭目录流
    closedir(pdir);
    return 0;
}
```

注意：和 ls -l 命令不一样，目录项是没有通过任何方式进行排序的。

课堂小练习

写一个小程序，实现青春版 tree 命令。效果如下：

```
$ ./tree .
.
    dir1
        text1
        text2
    dir2
         file1
        file2
    dir3
        a.txt
        b.txt

3 directories, 6 files
```

2 文件相关操作

创建硬链接

`link`：创建硬链接 (给文件创建一个新的名字)。

```
#include <unistd.h>

int link(const char *oldpath, const char *newpath);
参数
    oldpath: 原文件路径
    newpath: 新文件路径
返回值
    成功: 返回0
    失败: 返回-1, 并设置errno
```

示例:

```
// link.c
#include <func.h>

int main(int argc, char *argv[])
{
    ARGS_CHECK(argc, 3);

    int ret = link(argv[1], argv[2]);
    ERROR_CHECK(ret, -1, "link");

    return 0;
}
```

删除硬链接

`unlink`：删除硬链接，如果文件的硬链接数为0，则删除文件。

```
#include <unistd.h>

int unlink(const char *pathname);
参数
    pathname: 文件路径
返回值
    成功: 返回0
    失败: 返回-1, 并设置errno。
```

示例:

```c
// unlink.c
#include <func.h>

int main(int argc, char *argv[])
{
    ARGS_CHECK(argc, 2);

    int ret = unlink(argv[1]);
    ERROR_CHECK(ret, -1, "unlink");

    return 0;
}
```

创建符号连接

`symlink`：创建符号链接。

```c
#include <unistd.h>

int symlink(const char *target, const char *linkpath);
参数
    target：符号链接指向的文件(即符号链接中要包含的字符串)。
    linkpath：符号链接的路径。
返回值
    成功：返回0。
    失败：返回-1，并设置errno。
```

示例：

```c
// symlink.c
#include <func.h>

int main(int argc, char *argv[])
{
    ARGS_CHECK(argc, 3);

    int ret = symlink(argv[1], argv[2]);
    ERROR_CHECK(ret, -1, "symlink");

    return 0;
}
```

课堂小练习

1. 写一个小程序，实现递归删除目录。

2．写一个小程序，实现递归复制目录。