Для каждого фала есть inode (уникальное числовое значение) – методанные (владелец, группа, разрешение, время последнего открытия)

ls –i

Семейство stat – извлечение методанных

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

Каждая функция возвращает информацию о файле.

int stat (const char *path, struct stat *buf); path – путь к файлу

int fstat (int fd, struct stat *buf); fd – дескриптор файла

Информация о файле сохраняется в структуре **stat**

Структура stat определяется в <bits/stat.h>, которая включается через <sys/stat.h>

struct stat {

dev_t st_dev; /* идентификатор устройства, на котором хранится файл */

ino_t st_ino; /* номер inode */

mode_t st_mode; /* Битовая маска для информации о режиме файла. Бит S_IFDIR устанавливается, если pathname определяет директорий; бит S_IFREG устанавливается, если pathname ссылается на обычный файл. Биты чтения/записи устанавливаются пользователем в соответствии с режимом доступа к файлу. Пользователь выполняет установку битов, используемых для расширения имени файла.

// Проверка на католог (st.st_mode & S_IFDIR)

if((st.st_mode&S_IFMT)==S_IFDIR&&struc->d_name[0]!='.')

*/

nlink_t st_nlink; /* число жестких ссылок. У каждого файла есть хотябы одна жесткая ссылка*/

uid_t st_uid; /* идентификатор пользователя владельца */

gid_t st_gid; /* идентификатор группы владельца */

dev_t st_rdev; /* идентификатор устройства (для специальных файлов) */

off_t st_size; /* общий размер в байтах */

blksize_t st_blksize; /* предпочтительный размер блока для эффективного ввода-вывода в файловой системе */

blkcnt_t st_blocks; /* количество выделенных блоков файловой системы, связанных с данным файлом. Это значения меньше значения в поле st_size, если в файле есть дыры (т.е. это разряженные файлы)*/

time_t st_atime; /* время последнего доступа */

time_t st_mtime; /* время последней модификации */

time_t st_ctime; /* время последнего изменения статуса */

};

S_IFMT	0170000	битовая маска для полей типа файла
S_IFSOCK	0140000	сокет
S_IFLNK	0120000	символьная ссылка
S_IFREG	0100000	обычный файл
S_IFBLK	0060000	блочное устройство
S_IFDIR	0040000	каталог
S_IFCHR	0020000	символьное устройство
S_IFIFO	0010000	канал FIFO
S_IRWXU	00700	маска для прав доступа пользователя
S_IRUSR	00400	пользователь имеет право чтения
S_IWUSR	00200	пользователь имеет право записи
S_IXUSR	00100	пользователь имеет право выполнения
S_IRWXG	00070	маска для прав доступа группы
S_IRGRP	00040	группа имеет права чтения
S_IWGRP	00020	группа имеет права записи
S_IXGRP	00010	группа имеет права выполнения

S_IRWXO	00007	маска прав доступа всех прочих (не находящихся в группе)
S_IROTH	00004	все прочие имеют права чтения
S_IWOTH	00002	все прочие имеют права записи
S_IXOTH	00001	все прочие имеют права выполнения

Возвращаемые значения

0-успех

-1 – ошибка. errno может принимать след. значения:

EACCESS

У вызывающего процесса отсутствуют разрешения на поиск для одного из каталогов в пути path (только для функций stat() и 1 stat()).

EBADF

Недопустимое значение fd (только для функции fstat()).

EFAULT

Параметр path или but содержит недопустимый указатель.

ELOOP

Параметр path включает слишком много символических ссылок (только для функций stat() и lstat()).

ENAMETOOLONG

Слишком длинное значение параметра path (только для функций stat()).

ENOENT

Компонент пути path не существует (только для функций stat()).

ENOMEM

Недостаточно памяти для выполнения данного запроса.

ENOTDIR

Компонент пути path не является каталогом (только для функций stat()).

Задание 1 – показать размер файла, выбранного в командной строке.

Изменение разрешения

#include <sys/types.h>

#include <sys/stat.h>

int chmod (const char *path, mode_t mode);

int fchmod (int fd, mode_t mode);

Возвращаемые значения

0-успех

-1 – ошибка. errno может принимать след. значения:

EACCESS

У вызывающего процесса нет разрешений на поиск для компонента пути path (только для chmod O).

EBADF

Недопустимый дескриптор файла fd (только для f chmod O).

EFAULT

Параметр path содержит недопустимый указатель (только для chmod O).

EIO

В файловой системе произошла внутренняя ошибка ввода-вывода. Это очень плохая ситуация, которая может указывать на повреждение диска или файловой системы.

ELOOP

Во время разрешения пути path ядро встретило слишком много символических ссылок (только для chmod ()).

ENAMETOOLONG

Слишком длинное значение параметра path (только для chmodO).

ENOENT

Путь path не существует (только для параметра chmodO).

ENOMEM

Недостаточно памяти для выполнения данного запроса.

ENOTDIR

Компонент пути path не является каталогом (только для chmodO).

EPERM

Действительный идентификатор вызывающего процесса не соответствует владельцу файла, и у процесса отсутствует характеристика CAPFOWNER.

EROFS

Файл находится в файловой системе, доступной только для чтения.

Изменение владения

#include <sys/types.h>

#include <unistd.h>

int chown (const char *path, uid_t owner, gid_t group);

int fchown (int fd, uid_t owner, gid_t group);

Возвращаемые значения

0-успех

-1 – ошибка. errno может принимать след. значения:

EACCESS

У вызывающего процесса отсутствуют разрешения на поиск для компонента пути path (только для chown O и lchown O).

EBADF

Недопустимое значение fd (только для f chown O).

EFAULT

Недопустимое значение path (только для chown () и 1 chown ()).

EIO

Внутренняя ошибка ввода-вывода (очень плохо).

ELOOP

Во время разрешения пути path ядро встретило слишком много

символических ссылок (только для chown O и lchown O).

ENAMETOOLONG

Слишком длинное значение path (только для chown () и 1 chown()).

ENOENT

Файл не существует.

ENOMEM

Недостаточно памяти для выполнения данного запроса.

ENOTDIR

Компонент пути path не является каталогом (только для chown() и IchownC)).

EPERM

У вызывающего процесса отсутствуют необходимые права для изменения

владельца или группы в соответствии с запросом.

EROFS

Файловая система доступна только для чтения.

Атрибуты файлов – ознакомиться самостоятельно.

Каталоги

Проверка текущего рабочего каталога

#include <unistd.h>

char * getcwd (char *buf, size_t size); //Обычно используется для сохранения текущего

Успешный вызов getcwd () копирует текущий рабочий каталог в форме абсолютного пути в буфер длиной size, на который указывает параметр buf, и возвращает указатель на buf. В случае ошибки вызов возвращает значение NULL и присваивает переменной errno одно из следующих значений:

EFAULT

Параметр buf содержит недопустимый указатель.

EINVAL

Значение size равно 0, но значение buf не равно NULL.

ENOENT

Текущий рабочий каталог более не действителен. Это может происходить

при удалении текущего рабочего каталога.

ERANGE

Значение size слишком мало, чтобы можно было сохранить текущий рабочий каталог в буфере buf. Необходимо выделить буфер большего размера и повторить попытку. Можно указывать NULL и 0.Стистема сама выделит, сколько надо, но это не по стандарту POSIX. Однако, в некоторых версиях Linux работает. Пример,

char *cwd;

cwd = getcwd (NULL, 0);

Изменение текущего рабочего каталога

#include <unistd.h>

int chdir (const char *path);

int fchdir (int fd);

Если происходит ошибка, chdirO также присваивает переменной еггпо одно

из следующих значений:

EACCESS

У вызывающего процесса отсутствуют разрешения на поиск для компонента

пути path.

EFAULT

Параметр path содержит недопустимый указатель.

EIO

Произошла внутренняя ошибка ввода-вывода.

ELOOP

Во время разрешения пути path ядру встретилось слишком много

символических ссылок.

ENAMETOOLONG

Слишком длинное значение path.

ENOENT

Каталог, на который указывает параметр path, не существует.

ENOMEM

Недостаточно памяти для выполнения данного запроса.

ENOTDIR

Один или несколько компонентов пути path не являются каталогами.

fchdirO может присваивать переменной errno следующие значения:

EACCESS

У вызывающего процесса отсутствуют разрешения на поиск в каталоге с дескриптором файла, указанным при помощи параметра fd (то есть бит исполнения не установлен). Такая ситуация случается, когда каталог более высокого уровня доступен для чтения, но не для исполнения, вызов open O выполняется успешно, но fchdirO завершается ошибкой.

EBADF

Значение параметра fd не является открытым дескриптором файла.

Лучше, однако, открывать текущий каталог вызовом open(), чтобы потом возвращаться в него при помощи fchdir(). Этот подход быстрее, так как ядро не хранит полный путь текущего рабочего каталога в памяти; хранится только структура inode. В конце нужно вызвать close (swd_fd);

Создание каталогов

#include <sys/stat.h>

#include <sys/types.h>

int mkdir (const char *path, mode_t mode);

В случае ошибки mkdir() возвращает значение -1 и присваивает переменной errno одно из следующих значений:

EACCESS

У текущего процесса нет прав на запись в родительский каталог, или один или несколько компонентов в пути path недоступны для поиска.

EEXIST

Путь path уже существует (и не обязательно является каталогом).

EFAULT

Параметр path содержит недопустимый указатель.

ELOOP

Во время разрешения пути path ядру встретилось слишком много символических ссылок.

ENAMETOOLONG

Слишком длинное значение path.

ENOENT

Компонент пути path не существует или является символической ссылкой, указывающей на несуществующий объект.

ENOMEM

Недостаточно памяти ядра для выполнения данного запроса.

ENOSPC

На устройстве, где находится path, недостаточно пространства либо превышена дисковая квота для пользователя.

ENOTDIR

Один или несколько компонентов пути path не являются каталогами.

EPERM

Файловая система, которой принадлежит path, не поддерживает создание каталогов.

EROFS

Файловая система, которой принадлежит path, подмонтирована с доступом

только на чтение.

Удаление каталогов

#include <unistd.h>

int rmdir (const char *path);

В случае ошибки rmdirO возвращает -1 и присваивает переменной errno

одно из следующих значений:

EACCESS

Запрещен доступ на запись в родительский каталог каталога path или одна

или несколько составляющих path недоступны для поиска.

EBUSY

Каталог path в данный момент используется системой и удалить его

невозможно. В Linux это может произойти только в том случае, если path является точкой монтирования или корневым каталогом.

EFAULT

Параметр path содержит недопустимый указатель.

EINVAL

Последним компонентом пути path является каталог точка.

ELOOP

При разрешении пути path ядру встретилось слишком много символических

ссылок.

ENAMETOOLONG

Слишком длинное значение path.

ENOENT

Компонент пути path не существует или представляет собой символическую

ссылку, указывающую на несуществующий объект.

ENOMEM

Недостаточно памяти ядра для выполнения данного запроса.

ENOTDIR

Один или несколько компонентов пути path не являются каталогами.

ENOTEMPTY

Каталог path содержит другие записи, кроме специальных каталогов точка и точка-точка.

EPERM

Для каталога, являющегося предком каталога path, установлен бит закрепления в памяти (sticky bit, SISVTX), но действительный идентификатор пользователя процесса не совпадает ни с идентификатором пользователя указанного родителя, ни с идентификатором самого каталога path, а также процесс не обладает характеристикой CAPFOWNER. Также возможно, что файловая система, которой принадлежит path, не поддерживает удаление каталогов.

EROFS

Файловая система, которой принадлежит path, подмонтирована в режиме только для чтения.

Чтение содержимого каталога

#include <sys/types.h>

#include <dirent.h>

DIR * opendir (const char *name);

Поток каталога — это всего лишь дескриптор файла, представляющий открытый каталог, некоторое количество метаданных и буфер для записи содержимого каталога. Следовательно, имея поток каталога, можно получить дескриптор файла для соответствующего каталога:

#define _BSD_SOURCE /* или _SVID_SOURCE */

#include <sys/types.h>

#include <dirent.h>

int dirfd (DIR *dir);

Чтение из потока каталога

#include <sys/types. h>

#include <dirent.h>

struct dirent * readdir (DIR *dir);

Успешный вызов readdir() возвращает очередную запись в каталоге, указанном при помощи параметра dir. Запись каталога представляется структурой dirent.

struct dirent {

ino_t dino; /* номер inode */

off_t doff; /* смещение к следующей записи dirent */

unsigned short d_reclen; /* длина данной записи */

unsigned char d_type; /* тип файла */

char d_name[256]; /* имя файла */ };

В стандарте POSIX обязательным является только поле d_name, то есть имя файла в каталоге. Прочие поля не обязательны или уникальны для Linux. В приложениях, стремящихся к хорошей переносимости на другие системы или к полному соблюдению POSIX, следует обращаться только к полю d_name.

Закрытие потока каталога

finclude <sys/types.h>

include <dirent.h>

int closedir (DIR *dir);

Ссылки – ознакомиться самостоятельно.

Копирование

В POSIX нет системного вызова для копирования

Перемещение

#include <stdio.h>

int rename (const char *oldpath, const char *newpath);

Успешный вызов rename переименовывает путь к файлу oldpath в путь newpath. Содержимое файла и его inode не меняются. И oldpath, и newpath должны принадлежать одной файловой системе; если это не так, то вызов завершается ошибкой.

Задание 2 – поиск файла (поданного в аргументах командной строки) в каталоге

Задание 3 – обход каталога