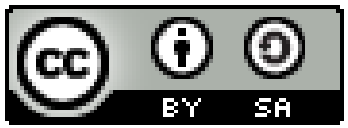
Системные вызовы для доступа к каталогам

Основные понятия и действия



December 19, 2013

Отличия каталога от обычного файла

Под специальный файл типа “каталог” (directory) место в файловой системе распределяется так же, как и для обычного файла. Но работа с содержимым каталога отличается от работы с содержимым файла:

- формат записи о файле в каталоге зависит от файловой системы, и программы пользователя не могут, и не должны знать внутреннее её устройство;
- если разрешить пользователю произвольную запись в каталог, то он сможет разрушить всю файловую систему.

Вывод: для работы с каталогами требуются отдельные системные вызовы.

Чтение и запись в каталог

Открыть каталог как обычный файл нельзя.

Запись в каталоги должна производится при изменении метаинформации файловой системы, а именно, при:

- создании файла (open, mknod, mkdir, symlink, socket);
- создании нового имени файла (link);
- удалении имени файла (unlink);
- удалении каталога (rmdir);

Чтение из каталога позволяет получить имена файлов, “содержащихся” в этом каталоге (opendir, readdir . . .).

Открытие каталога

Для открытия каталога для чтения (аналогично открытию файла) используется функция **opendir**:

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

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

name — имя каталога.
Возвращает:
0 — ошибка открытия каталога, код ошибки — в errno.
!=0 — указатель на DIR (структуру), содержащую все необходимые элементы для доступа к каталогу.
На самом деле действие реализуется системным вызовом open со специальными параметрами.

Закрытие каталога

Как и файл, каталог следует закрывать после использования, во избежание утечки ресурсов. Используется функция **closedir**.

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

int closedir(DIR *dirp);
```

Возвращает:
0 — закрытие прошло без ошибок.
−1 — ошибка, причина — в errno (часто игнорируется).

При завершении процесса каталоги, как и файлы, закрываются автоматически.

Чтение записи каталога

Прочитать одну запись каталога, получив имя одного файла, можно с помощью функции **readdir**.

```
#include <dirent.h>

struct dirent *readdir(DIR *dirp);

int readdir_r(DIR *dirp, struct dirent *entry,
              struct dirent **result);
```

dirp — указатель на DIR, (полученный от opendir).
Возвращает:
0 — при конце каталога и при ошибке (различать по изменению errno).
!=0 — указатель на структуру dirent, содержащую данные об очередном файле из каталога. Сама структура принадлежит функции readdir (static).

Есть одноимённый системный вызов readdir (т.е. man 2 readdir).

Чтение записи каталога
Структура dirent

Стандарт POSIX.1 определяет наличие в структуре dirent двух полей:

```
struct dirent {
  ino_t d_ino;
  // возможны необязательные поля
  char d_name[NAME_MAX+1];
};
```

Поле **d_name** — имя файла. Максимальный размер имени файла в каталоге — NAME_MAX байт (обычно 255, но использовать NAME_MAX). Пользователю обычно необходимо только это поле.
Поле **d_ino** — номер индексного дескриптора. Можно получить, используя **stat**.
Использование других полей приводит к непереносимости программы и трудноуловимым ошибкам.

Пример чтения каталога

```
dirent *de;
DIR *d = opendir("/");
if( !d ) {
  fprintf( stderr, "fail to open dir </>: %s\n",
    "/", strerror(errno);
  // return 1;
}

while( (de = readdir(d)) != 0 ) {
  printf( "%s\n", de->d_name );
}

closedir(d);
```

Перемещение в каталоге

Перейти к началу каталога можно с помощью **rewinddir**, при этом каталог читается заново.

```
void rewinddir(DIR *dirp);
```

Запомнить текущую позиции можно с помощью **telldir**, а восстановить – **seekdir**

```
long telldir(DIR *dirp); // возвращает
     // текущую позицию в каталоге
void seekdir(DIR *dirp, long offset);
     // перемещает на заданную
```

Не гарантируется правильность смещений при изменении каталога.

Создание нового имени файла

В пределах одной файловой системы файлу можно дать еще имена (создать жёсткие ссылки). Используется системный вызов **link**

```
#include <unistd.h>

int link(const char *oldpath, const char *newpath);
```

oldpath — существующее имя файла;
newpath — новое имя файла.
Возвращает:
-1 — ошибка, причина — в errno.
0 — новое имя создано.
Старое и новое имена совершенно равноправны!
Создавать жёсткие ссылки на каталоги может только суперпользователь.

Удаление имён файлов

Для удаления имени файла (но не каталога) используется системный вызов **unlink**.

```
#include <unistd.h>

int unlink(const char *pathname);
```

Возвращает:
−1 — ошибка, причина — в errno. 0 — имя удалено.

Файл удаляется, если:

- все имена файла удалены (st.st_nlinks==0)
- файл не открыт ни одним процессом.

Создание и удаление каталогов

Так как создание и удаление каталогов требует создания и удаления нескольких имён в файловой системе, то и системные вызовы — специальные:

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

int mkdir(const char *pathname, mode_t mode);
int rmdir(const char *pathname);   // удалить
```

pathname — имя каталога, **mode** — права доступа (младшие 9 бит).

Возвращает:

-1 — ошибка, причина — в errno.

0 — Ok.

Удалять системным вызовом можно только пустой каталог.

Создание файла устройств

Файлы устройств создаются с помощью системного вызова **mknod**

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

int mknod(const char *pathname, mode_t mode,
          dev_t dev);
```

pathname — имя файла, **mode** — ТИП и права доступа.
dev — major и minor файла устройства (см. makedev).
Тип: S_IFREG, S_IFCHR, S_IFBLK, S_IFIFO or S_IFSOCK.
Возвращает:
−1 — ошибка, причина — в errno. 0 — Ok.

Создание символической ссылки

Специальный файл типа “символическая ссылка” можно создать с помощью системного вызова **symlink**.

```
#include <unistd.h>

int symlink(const char *target, const char *link);
```

target — имя, на которое ссылка ссылается;
link — имя нового файла — символической ссылки.

Можно создавать символические ссылки на несуществующие файлы, файлы в других файловых системах. Права ссылки игнорируются.
Прочитать ссылку — readlink.

```
ssize_t readlink(const char *path, char *buf,
                 size_t bufsiz);
```

Смена прав файла

Права к файлу изменяются с помощью системного вызова **chmod**

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

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

path — путь к файлу, **mode** — права доступа (12 бит), **fd** — файловый дескриптор.

Возвращает: -1 — ошибка, причина — в errno. 0 — Ok.

Доступен только владельцу и суперпользователю.

Смена владельца файла

Владелец и группа файла изменяются с помощью системного вызова **chown**

```
#include <unistd.h>

int chown(const char *path, uid_t owner, gid_t gid);
int fchown(int fd, uid_t owner, gid_t group);
int lchown(const char *path, uid_t owner,gid_t gid);
```

path — путь к файлу, **owner** — UID владельца, **gid** — GID группы (−1 — не изменять), **fd** — файловый дескриптор.

Возвращает: −1 — ошибка, причина — в errno. 0 — Ok.

Суперпользователь может менять владельца.
Пользователь может менять группу на любую из своих.

Монтирование файловых систем

Файловые системы монтируются с помощью системного вызова **mount**

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

int mount(const char *source, const char *target,
    const char *filesystemtype,
    unsigned long mountflags,
    const void *data);
int umount(const char *target);
```

source — что монтировать (файл устройства, файл образа . . .) **target** — куда (каталог), **filesystemtype** — тип файловой системы или “auto” **mountflags** — флаги монтирования, **data** — дополнительные данные. Доступен только владельцу и суперпользователю.