ОБЩИЙ ПРИНЦИП РАБОТЫ С ФАЙЛАМИ.

После открытие файла возвращается файловый дескриптор (>0 это номер из таблицы файлов), который используется как ядром, так и пользовательским пространством, а последующие операции (считывание, запись и т. п.) принимают файловый дескриптор как основной аргумент.

Каждая запись в списке содержит информацию об открытом файле, включая указатель на находящуюся в памяти копию inode файла и связанные метаданные, такие, как позиция в файле и режимы доступа.

Если только процесс явно не закроет их, у каждого процесса по определению есть по крайней мере три открытых файловых дескриптора:

- 0 (STDIN_FILENO) - стандартный ввод (standard in, stdin);

- 1(STDOUT_FILENO) - это стандартный вывод (standard out, stdout);

- 2 (STDERR_FILENO) - стандартная ошибка (standard error, stderr).

Открытие файлов

Read(), write(), open(), creat(), close()

СИСТЕМНЫЙ ВЫЗОВ OPEN

Возвращает файловый дескриптор. В качестве файловой позиции указывается его начало.

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

int open (const char *name, int flags);

int open (const char *name, int flags, mode_t mode);

Флаги для системного вызова open()

O_RDONLY (только чтение), O_WRONLY (только запись) и O_RDWR (и чтение и запись).

А также (ИЛИ |). МОЖНО посмотреть **man open**

(стр. 56-57)

O_APPEND - файл открывается в режиме присоединения (append mode). Это означает, что перед каждой записью указатель позиции в файле устанавливается на конец файла.

O_CREAT

Если файл, указанный при помощи аргумента name, не существует, то ядро создает его. Если файл уже существует, то этот флаг не оказывает никакого эффекта, если только одновременно не используется флаг O_EXCL.

O_DIRECT

Файл открывается для прямого ввода-вывода

O_EXCL

Когда этот флаг добавляется к флагу O_CREAT, то вызов open() завершается ошибкой, если файл, указанный при помощи аргумента name, уже существует.

O_SYNC

Файл открывается для синхронного ввода-вывода. Ни одна операция записи не завершается, пока данные физически не записываются на диск; обычные операции считывания и так выполняются синхронно, поэтому данный флаг никак не влияет на чтение файлов.

O_TRUNC

Если файл существует и это обычный файл, а указанные флаги допускают запись, то файл усекается до нулевой длины.

O_NONBLOCK

Если возможно, файл открывается в режиме без блокировки. Ни вызов open(), ни любые другие операции не приводят к блокировке (засыпанию) процесса во время ввода-вывода. Это поведение может определяться только для конвейеров FIFO.

Права доступа новых файлов (атрибут mode_t mode)

Аргумент mode игнорируется во всех случаях, когда файл не создается; он необходим, когда передается флаг O_CREAT. Если не передать – результат будет непредсказуем (**стр. 59**). Комбинация с побитовым или:

- S_IRWXU — владелец имеет право на чтение, запись и исполнение файла;
- S_IRUSR — владелец имеет право на чтение;
- S_IWUSR — владелец имеет право на запись;
- S_IXUSR — владелец имеет право на исполнение;
- S_IRWXG — владеющая группа имеет право на чтение, запись и исполнение файла;
- S_IRGRP — владеющая группа имеет право на чтение;
- S_IWGRP — владеющая группа имеет право на запись;
- S_IXGRP — владеющая группа имеет право на исполнение;
- S_IRWXO — любой пользователь имеет право на чтение, запись и исполнение файла;
- S_IROTH — любой пользователь имеет право на чтение;
- S_IWOTH — любой пользователь имеет право на запись;
- S_IXOTH — любой пользователь имеет право на исполнение.

ФУНКЦИЯ CREAT

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

int creat (const char *name, mode_t mode);

В принципе не нужна (используется для совместимости, так как в последнее время убрали из системных вызовов, и creat стало просто библиотечной функцией):

int creat (const char *name, int mode)

{

return open (name, O_WRONLY | O_CREAT | O_TRUNC, mode);

}

Возвращаемые значения и коды ошибок

В случае успешного завершения open и create - возвращается дескриптор файла, в случае ошибки -1 и устанавливает переменную errno (таблица 1.2 стр. 51).

ЧТЕНИЕ ФАЙЛА ПРИ ПОМОЩИ СИСТЕМНОГО ВЫЗОВА READ

#include <unistd.h>

ssize_t read (int fd, void *buf, size_t len);

Каждый вызов считывает до len байтов в буфер buf, начиная с текущего смещения в файле, который указывается при помощи аргумента fd. В случае успеха возвращается число записанных в buf байтов. В случае ошибки вызов возвращает значение -1 и устанавливает переменную errno. Позиция в файле увеличивается на число байтов, считанных из fd. Если объект, представляемый fd, не поддерживает поиск (например, это может быть файл устройства посимвольного ввода-вывода), то чтение всегда начинается с «текущей» позиции.

Простой пример (не совсем правильный):

unsigned long word;

ssize_t nr;

nr = read (fd, &word, sizeof (unsigned long));

if (nr == -1) /*ошибка*/

Возможные возвращаемые значения:

1. Вызов возвращает значение, равное len. Все len считанных байтов сохраняются в buf. Именно это нам и требовалось.

2. Если nr вернул результат значение <len, но >0. То не понятно почему, может ошибка, может сигнал прерывает незавершенную операцию считывания, может все нормально, просто в файле меньше байт.

3. Если nr=EOF(0). Это означает, что файл закончен.

4. Вызов блокируется, поскольку в текущий момент данные недоступны. Этого не происходит в неблокирующем режиме.

5. Nr=-1. Errno= EINTR (Системный вызов был прерван, раньше чем получили доступ к байтам) или Errno= EAGAIN (Повторите попытку. Может возникнуть когда не используется режим блокировки. flags = O_NONBLOCK). В этом случае нужно просто повторить попытку чтения.

6. Более серьезная ошибка.

Считывание всех байтов, с учетом описанного выше

```
ssize_t ret;

while (len ! = 0 && (ret = read (fd, buf, len)) != 0) {

lf (ret ==- 1) {

if (errno == EINTR) continue;

perror ("read");break;

}

len -= ret;

buf += ret;

}
```

Считывание без блокировки

```
char buf[BUFSIZ];

ssize_t nr;

start:

nr = read (fd, buf, BUFSIZ).

if (nr == -1) {

if (errno == EINTR)

goto start;

if (errno == EAGAIN)

/* повторить позже, сначала можно выполнить что-то другое */

else

/* ошибка */

}
```

Прочие значения ошибки

EBADF

Данный файловый дескриптор недопустим или не открыт для чтения.

EFAULT

Указатель, предоставленный вызову при помощи buf, находится за пределами адресного пространства вызывающего процесса.

EINVAL

Файловый дескриптор соответствует объекту, не допускающему чтение.

EIO

Произошла низкоуровневая ошибка ввода-вывода.

Лимиты размера для вызова read

Типы size_t и ssize_t определены в POSIX. Тип size_t используется для хранения значений, обозначающих размеры, в байтах. Тип ssize_t — это версия size_t со знаком (отрицательные значения обозначают ошибки). В 32-разрядных системах дополнительные типы C — это обычно unsigned int и int соответственно. Максимальное значение типа size_t — это SIZE_MAX; максимальное значение типа ssizet — SSIZE_MAX. Если значение len больше SSIZE_MAX, то результаты вызова read() не определены.

Не стоит забывать об этом ограничении: if (len > SSIZE_MAX) len = SSIZF_MAX.

ЗАПИСЬ ПРИ ПОМОЩИ СИСТЕМНОГО ВЫЗОВА WRITE

#include <unistd.h>

ssize_t write (int fd, const void *buf, size_t count);

Начинает писать с места, где установлена позиция. Если записывать из разных потоков, нужна синхронизация.

Далее все аналогично read, с тем отличием, что для обычных файлов с очень большой вероятностью частичной записи не будет. Для специальных файлов лучше делать запись через цикл, аналогичный тому, что был в read.

Режим присоединения (flag = O_APPEND)

Запись начинается не с текущей позиции, а с конца. Удобно использовать если запись идет из разных потоков, например ведется файл «журнал» из разных потоков.

Запись без блокировки

Аналогично чтению.

Прочие коды ошибок

Прочие полезные значения errno включают в себя:

EBADF

Указанный файловый дескриптор недопустим или не открыт для записи.

EFAULT

Указатель в аргументе buf указывает за пределы адресного пространства процесса.

EFBIG

Данная операция записи сделала бы файл больше максимального допустимого размера для процесса или больше внутренних пределов реализации.

EINVAL

Указанный файловый дескриптор соответствует объекту, не допускающему запись.

EIO

Произошла низкоуровневая ошибка.

ENOSPC

В файловой системе, откуда взят данный дескриптор файла, недостаточно свободного пространства.

EPIPE

Данный файловый дескриптор связан с конвейером или сокетом, считывающая сторона которого закрыта.

Ограничения размера для вызова write

Если значение count превышает значение SSIZEMAX, то результат вызова write() не определен.

Если выполнить write() со значением count, равным нулю, то вызов мгновенно вернет значение 0.

СИНХРОНИЗИРОВАННЫЙ ВВОД-ВЫВОД

Если ничего не предпринимать, то будет использоваться отложенная запись – так работает обыкновенный write. Т.е. сначала все помещается в память, а уже потом на диск. Сделано это, так как работа с памятью в любом случае быстрее чем работа с жестким диском.

При использовании write, данные сразу же не записываются на диск, а помещаются в буфер ядра (это сделано для оптимизации и увеличения скорости чтобы физически записывать данные большими пачками) т.е. нет гарантии что сразу же после write данные окажутся на жёстком диске.

FSYNC() И FDATASYNC()

#include <unistd.h>

int fsync (int fd); - запись на диск и данных и метаданных. Завершается, когда и то и другое записано.

int fdatasync (int fd); - запись на диск только данных и «существенных» метаданных, например размер файла. Потенциально быстрее, чем fsync.

int ret;

ret = fsync (fd);

if (ret == -1) /* ошибка */

Ни одна из этих функций не гарантирует, что все обновившиеся записи каталогов, в которых содержится файл, будут синхронизированы на диске. Имеется в виду, что если ссылка на файл недавно была обновлена, то информация из данного файла может успешно попасть на диск, но еще не отразиться в ассоциированной с файлом записи из того или иного каталога. В таком случае файл окажется недоступен. Чтобы гарантировать, что на диске окажутся и все обновления, касающиеся записей в каталогах, fsync() нужно вызвать и к дескриптору файла, открытому для каталога, содержащего интересующий нас файл.

Возвращаемые значения и коды ошибок

Если успешно, то 0, иначе -1. Бывает случаи, что fsync не реализован в системе (в последнее время реализован).

EBADF

Указанный файловый дескриптор является недопустимым или не открыт для записи.

EINVAL

Указанный файловый дескриптор соответствует объекту, не поддерживающему синхронизацию.

EIO

Во время синхронизации произошла низкоуровневая ошибка ввода-вывода.

М.Б. что fsync завершился из-за того (ошибка EINVAL), что он не реализован. Можно попробовать вызвать fdatasync.

SYNC()

#include <unistd.h>

void sync (void);

У этой функции нет параметров, и она не возвращает никакое значение. Она всегда завершается успешно, и возврат значения указывает на то, что все буферы — как с

данными, так и с метаданными — были гарантированно записаны на диск. Может работать ОЧЕНЬ ДОЛГО.

Флаг O_SYNC для OPEN

Передается в OPEN.

Можно представлять себе это так: O_SYNC заставляет систему неявно выполнять вызов fsync() после каждой операции write() до того, как вызов на запись возвращает значение. В действительности семантика именно такова, хотя в ядре Linux работа O_SYNC реализуется немного более эффективно. По сравнению с fsync() и fdatasync() сильно увеличивается задержки.

Флаги O_DSYNC и O_RSYNC **для OPEN**

Флаг O_DSYNC указывает, что после каждой операции должны синхронизироваться только обычные данные, но не метаданные.

Флаг O_RSYNC регламентирует, что все побочные эффекты операции считывания также должны быть синхронизированы.

ПРЯМОЙ ВВОД-ВЫВОД

Когда вы добавляете флаг O_DIRECT в вызов open(), ядро минимизирует управление вводом-выводом и ввод-вывод осуществляется напрямую из буферов в пользовательском пространстве в устройство, обходя страничный кэш. Все операции ввода-вывода синхронизируются и не возвращают значение до тех пор, пока полностью не завершаются.

При использовании прямого ввода-вывода длина запроса, выравнивание буфера и смещения файлов — все должны быть целыми числами, кратными размеру сектора соответствующего устройства.

ЗАКРЫТИЕ ФАЙЛОВ

#include <unistd.h>

int close (int fd);

Отсоединятся дескриптор от файла. Если все хорошо возвращает 0. Завершение выполнения функции close не означает, что файл сброшен на диск, чтобы в этом убедиться, необходимо использовать функции синхронизированного ввода/вывода.

Ошибки:

EBADF - данный дескриптор файла недопустим

EIO – низкоуровневая ошибка

ПОИСК ПРИ ПОМОЩИ LSEEK

#include <sys/types.h>

#include <unistd.h>

off_t lseek (int fd, off_t pos, int origin);

Аргумент pos может содержать отрицательное, положительное и нулевое значение.

Поведение lseek() зависит от аргумента origin, который может принимать следующие значения:

SEEK_CUR

Текущая позиция в файле fd увеличивается на значение аргумента pos.

SEEK_END

В качестве текущей позиции в файле fd устанавливается значение, равное текущей длине файла плюс значение pos.

SEEK_SET

Текущая позиция в файле fd приравнивается значению аргумента pos.

В случае успеха вызов возвращает новую позицию в файле. В случае ошибки он возвращает значение -1 и соответствующим образом устанавливает переменную errno.

Чаще всего lseek () используют для поиска начала файла, конца файла или определения текущей позиции в файле, связанном с данным дескриптором файла.

Поиск за пределами конца файла

ret = lseek (fd, (off_t) 1688, SEEK_END).После записи дырка заполнится нулями. НО!! Не занимает отдельного места на диске.

Ограничения off_t обычно long. Если превысили errno= EOVERFLOW

Еще есть ошибки

EBADF - данный дикриптор файла недопустим

EINVAL – значение origin не равно SEEK_CUR, SEEK_END, SEEK_SET или результирующая позиция <0.

Позиционное чтение и запись

pread и pwrite

#define _XOPEN_SOURCE 500

#include <unistd.h>

ssize_t pread (int fd, void *buf, sizet count, off_t pos);

ssize_t pwrite (int fd, const void *buf, size_t count, off_t pos);

Делают все тоже самое, только на вход подается позиция в файле + они не меняют позицию после выполнения. Их легче использовать, чем lseek если работа ведется в разных потоках.

Возвращают число прочитанных или записанных байт. -1, если ошибка.

Значения errno в случае ошибки такие же, как и у read + lseek или write + lseek

УСЕЧЕНИЕ ФАЙЛОВ

#include <unistd.h>

#include <sys/types.h>

int ftruncate (int fd, off_t len); fd – дескиптор, должен открыт для записи.

int truncate (const char *path, off_t len); path – имя файла, должен открыт для записи.

Оба системных вызова усекают указанный файл до длины len. Можно «усекать» до размера большей длинны файла. Если все хорошо, то возвращают 0, иначе -1.

Пример.

БЫЛО: файл 123.txt: 123456789

ret = truncate ("./122.txt", 3); Получили: 123

МУЛЬТИПЛЕКСИРОВАННЫЙ ВВОД-ВЫВОД

Мультиплексированный ввод-вывод (multiplexed I/O) позволяет приложению одновременно фиксироваться на нескольких файловых дескрипторах и получать уведомления, когда один из них становится доступным для чтения или записи без блокировки.

select, poll и epoll.

SELECT

Синхронный мультиплексированный ввод-вывод. Select – ожидает пока какой-нибудь файл будет готов к использованию без блокировки или пока истечет время.

#include <sys/time.h>

#include <sys/types.h>

#include <unistd.h>

int select (int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);

FD_CLR(int fd, fd_set *set)- удаляет дескриптор из указанной группы;

FD_ISSET(int fd, fd_set *set) - проверяет принадлежит ли дескриптор к группе. Если возвращает 0, то не принадлежит, иначе возвращает значение >0;

FD_SET(int fd, fd_set *set)- добавляет дескриптор к набору;

FD_ZERO(fd_set *set)- удаляет все дескриптора файла из набора.

#include <sys/time.h>

struct timeval {

long tvsec; /* секунды */

long tvusec; /* микросекунды */

};

Параметры:

readfds – набор дескрипторов фалов, для которых нужно ожидать, чтобы они стали доступны для чтения

writefds – набор дескрипторов фалов, для которых нужно ожидать, чтобы они стали доступны для записи

exceptfds – отлавливать значение исключений (нужны только для сокетов).

Если какая-то группа отсутствует, то передается NULL.

n – максимальное значение из всех дескрипторов+1

Timeout – сколько ждать.

Если Timeout – NULL, то время не ограничено.

После завершения select() в наборе остаются только те, которые доступны.

Возвращаемые значения и коды ошибок

В случае успеха вызов select() возвращает число дескрипторов файла во всех трех наборах, готовых к вводу-выводу. Иначе -1. Если закончился Timeout, то 0. В результате в Timeout записывается значение изначального Timeout-сколько времени прошло.

EBADF - в одном из наборов содержится недопустимы дескриптор

EINVAL – значение n<0 или указанное значение таймаута недопустимо

ENOMEM – недостаточно памяти для выполнения запроса

Пример. Взять один файл и ждать на чтение.

Краткое засыпание при помощи select

struct timeval tv;

tv tv_sec = 0;

tv.tv_usec = 500;

select (0, NULL, NULL, NULL, &tv);

ВЫЗОВ PSELECT

#define _X0PEN_S0URCE 600

#include <sys/select.h>

int pselect (int n, fd_set *readfds, fdset *writefds, fdset *exceptfds, const struct timespec *timeout, const sigset_t *sigmask);

FD_CLR(int fd, fd_set *set);

FD_ISSET(int fd, fd_set *set);

FD_SET(int fd, fd_set *set);

FD_ZERO(fd_set *set);

Отличия:

- в timespec – наносекунды, a timeval – микросекунды

- в pselect timeout не меняется, в select – меняется

- в pselect есть sigmask – ожидание между дескрипторами файла и сигналами (Подробнее потом). Если не использовать, то устанавливается значение NULL.

Обычно используют select.

СИСТЕМНЫЙ ВЫЗОВ POLL

#include <sys/poll.h>

int poll (struct pollfd *fds, unsigned int nfds, int timeout);

nfds – количество элементов в структуре

timeout – время в миллисекундах

struct pollfd {

int fd; /* дескриптор файла */

short events; /* запрошенные события для наблюдения */

short revents; /* возвращаемые случившиеся события */

};

Возвращаемые значения и коды ошибок как у select.

Поле events в структуре представляет собой битовую маску наблюдаемых событий для данного дескриптора файла. Это поле устанавливает пользователь. Поле revents — это битовая маска событий, которые встретились для данного дескриптора файла. Его устанавливает ядро, когда вызов возвращает значение. Все события, запрошенные в поле events, могут быть возвращены в поле revents. Допустимы следующие значения:

- POLLIN — имеются данные для считывания;
- POLLRDNORM — имеются обычные данные для считывания;
- POLLRDBAND — имеются приоритетные данные для считывания;
- POLLPRI — имеются срочные данные для считывания;
- POLLOUT — запись блокироваться не будет;
- POLLWRNORM — запись обычных данных блокироваться не будет;
- POLLWRBAND — запись приоритетных данных блокироваться не будет;
- POLLMSG — доступно сообщение SIGPOLL.

Кроме того, в поле revents могут быть возвращены следующие события:

- POLLER — возникла ошибка на указанном файловом дескрипторе;
- POLLHUP — событие зависания на указанном файловом дескрипторе;
- POLLNVAL — указанный файловый дескриптор не является допустимым.

Сочетание POLLIN | POLLPRI эквивалентно событию считывания в вызове select(), а событие POLLOUT | POLLWRBAND идентично событию записи в вызове select(). Значение POLLIN эквивалентно POLLRDNORM |POLLRDBAND, а POLLOUT эквивалентно POLLWRNORM

Возвращаемые значения и коды ошибок

В случае успеха poll() возвращает количество файловых дескрипторов, чьи структуры содержат ненулевые поля revents. В случае возникновения задержки до каких-либо событий этот вызов возвращает 0. При ошибке возвращается –1, а errno устанавливается в одно из следующих значений:

EBADF — для одной или нескольких структур были заданы недопустимые файловые дескрипторы;

EFAULT — значение указателя на файловые дескрипторы находится за пределами адресного пространства процесса;

EINTR — до того как произошло какое-либо из запрошенных событий, был выдан сигнал; вызов можно повторить;

EINVAL — параметр nfds превысил значение RLIMIT_NOFILE;

ENOMEM — для выполнения запроса оказалось недостаточно памяти.

Пример использование poll() для одновременной проверки двух условий: не возникнет ли блокировка при считывании с stdin и записи в stdout.

Системный вызов ppoll()

#define _GNU_SOURCE

#include <sys/poll.h>

int ppoll (struct pollfd *fds, nfds_t nfds, const struct timespec *timeout, const sigset_t *sigmask);

Отличия такие же как и у pselect от select