Собрал Мигалин Сергей

Расписать назначение, аргументы и возвращаемые значения следующих системных вызовов:

1. alarm()

Описание

Посылает процессу сигнал побудки (функция, настраивающая таймер на подачу сигнала).

alarm настраивает таймер на подачу сигнала SIGALRM процессу через seconds секунд. Если значение seconds равно нулю, то сигнал alarm не будет послан. В любом случае, предыдущее значение alarm не запоминается.

Возвращаемое значение

alarm возвращает число секунд, оставшихся до запланированного сигнала, или 0, если сигнал не был задан.

Прототип

```
#include <unistd.h>
unsigned int alarm(unsigned int seconds);
```

2. chdir()

Описание

Функция смены рабочего каталога (устанавливает текущий каталог, указанный в аргументе path)

Возвращаемое значение

После успешного выполнения возвращаемое значение становится нулевым. При ошибке возвращаемое значение равно -1, а переменной errno присваивается код ошибки.

Прототип

```
#include <unistd.h>
int chdir(const char *path);
```

3. chroot()

Описание

Функция установки нового корневого каталога. **chroot** изменяет имя корневого каталога на указанный в аргументе path. Этот каталог - для файлов, имена которых начинаются со знака "/". Корневой каталог наследуется всеми "потомками" текущего процесса. Только суперпользователь может изменять корневой каталог. Этот системный вызов не изменяет текущий рабочий каталог, поэтому "." может находиться вне дерева каталогов, имя которого начинается со знака "/".

Возвращаемое значение

После успешного выполнения возвращаемое значение становится нулевым. При ошибке возвращаемое значение равно -1, а переменной errno присваивается код ошибки.

Прототип

```
#include <unistd.h>
int chroot(const char *path);
```

4. chmod()

Описание

Функция изменяет режим доступа к файлу, заданному параметром path.

chmod изменяет бит режима файла каждого заданного файла в соответствии с mode, который может быть либо символическим представлением об изменениях.

С помощью операции или можно сразу задавать следующие режимы:

S_ISUID 04000 установить пользователю права на выполнение

S_ISGID 02000 установить группе права на выполнение.

Возвращаемое значение

После успешного выполнения возвращаемое значение становится нулевым. При ошибке возвращаемое значение равно -1, а переменной errno присваивается код ошибки.

Прототип

```
#include <unistd.h>
int chmod(const char *path, mode_t mode);
```

5. fchmod()

Описание

Функции изменяют режим доступа к файлу, заданному описателем файла fildes в соответствии с mode (см. предыдущий пункт)

Возвращаемое значение

После успешного выполнения возвращаемое значение становится нулевым. При ошибке возвращаемое значение равно -1, а переменной errno присваивается код ошибки.

Прототип

```
#include <unistd.h>
int fchmod(int fildes, mode_t mode);
```

6, 7, 8. chown, fchown, lchown

Описание

chown, fchown, lchown - изменить владельца файла

Изменяет владельца для файла, задаваемого параметрами path или fd. Только суперпользователь может изменять владельца файла. Владелец файла может изменять группу файла на любую группу, к которой он принадлежит. Суперпользователь может произвольно изменять группу.

В чем разница?

Chown работает с аргументом имени файла, **fchown** работает с открытым файлом, а **lchown** работает с символической ссылкой вместо файла, на который эта ссылка указывает.

chown() changes the ownership of the file specified by path, which is dereferenced if it is a symbolic link.
fchown() changes the ownership of the file referred to by the open file descriptor fd.
lchown() is like chown(), but does not dereference symbolic links.

Если параметр owner или group заданы как -1, то соответствующий идентификатор не изменяется.
Когда владелец или группа исполняемого файла изменяются не-суперпользователем, то очищаются биты S_ISUID и S_ISGID. POSIX не требует, чтобы это происходило, когда суперпользователь выполняет chown; в этом случае поведение зависит от версии ядра Linux. Если в правах доступа к файлу не установлен бит исполнения группой (S_IXGRP), то бит S_ISGID означает принудительную блокировку на этом файле и не очищается функцией chown.

Возвращаемое значение

После успешного выполнения возвращаемое значение становится нулевым. При ошибке возвращаемое значение равно -1, а переменной errno присваивается код ошибки.

Прототип

```
#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);
int lchown(const char *path, uid_t owner, gid_t group);
```

9. access()

Описание

access - проверить права доступа пользователя к файлу
access проверяет, имеет ли процесс права на чтение или запись, или же просто проверяет, существует ли файл (или другой объект файловой системы), с именем pathname. Если pathname является символьной ссылкой, то проверяются права доступа к файлу, на который она ссылается.
mode -- это маска, состоящая из одного или более флагов R_OK, W_OK, X_OK и F_OK.
R_OK, W_OK и X_OK запрашивают соответственно проверку существования файла и возможности его чтения, записи или выполнения. F_OK просто проверяет существование файла.
Результаты проверки зависят от прав доступа к каталогам, находящимся по пути к файлу, заданному параметром pathname, и от прав доступа к каталогам и файлам, на которые ссылаются символьные ссылки, встреченные по пути.
Проверка осуществляется, используя реальные, а не эффективные идентификаторы пользователя и группы. Эффективные идентификаторы будут использоваться при действительной попытке выполнения той или иной операции. Это дает setuid-программам простой способ проверить права доступа настоящего пользователя.
Проверяются только биты прав доступа, а не тип файла или его содержимое. Таким образом, если каталог имеет "возможность записи", это, вероятно, означает, что в нем можно создавать файлы, а не что в этот каталог можно писать так же, как в обычный файл. Подобно этому файл из DOS может показаться "выполняемым", но системный вызов execve завершится неудачно.
Если процесс имеет соответствующие привелегии, то некоторые реализации могут показать успех для X_OK даже если права на файл не содержат бит, разрешающий выполнение.

Возвращаемое значение

В случае успеха (есть все запрошенные права) возвращается нуль. При ошибке (по крайней мере один запрос прав из mode был неудовлетворен, или случилась другая ошибка), возвращается -1, а errno устанавливается должным образом.

Прототип

```
#include <unistd.h>
int access(const char *pathname, int mode);
```

10. brk()

Описание

Brk устанавливает конец сегмента данных в значение, указанное в аргументе end_data_segment, когда это значение является приемлимым, система симулирует нехватку памяти и процесс не достигает своего максимально возможного размера сегмента данных

Возвращаемое значение

В случае успеха brk возвращает ноль, в случае ошибки возвращается -1.

Прототип

```
#include <unistd.h>
int brk(void *end_data_segment);
```

11. sbrk()

Описание

Sbrk увеличивает пространство данных программы на increment байт. Sbrk не является системным вызовом, он просто является обёрткой (wrapper), которую использует библиотека C. Вызов sbrk с инкрементом 0 может быть использован, чтобы найти текущее местоположения прерывания программы.

Возвращаемое значение

Sbrk возвращает указатель на начало новой области. В случае ошибки возвращается -1.

Прототип

```
#include <unistd.h>
void *sbrk(intptr_t increment);
```

12. exit()

Описание

exit - обычное завершение работы программы

Функция exit() приводит к обычному завершению программы, и величина status & 0377 возвращается процессу-родителю. Все данные всех открытых потоков сохраняются и потоки закрываются. Стандарт C описывает два определения EXIT_SUCCESS и EXIT_FAILURE, которые могут быть переданы exit() для обозначения соответственно успешного и неуспешного завершения.

Возвращаемое значение

Функция exit() не возвращает никаких значений.

Прототип

```
#include <stdlib.h>
void exit(int status);
```

13. __exit()

Описание

_exit, _Exit - функция, завершающая работу программы

_exit "немедленно" завершает работу программы. Все дескрипторы файлов, принадлежащие процессу, закрываются; все его дочерние процессы начинают управляться процессом 1 (init), а родительскому процессу посылается сигнал SIGCHLD.

Значение status возвращается родительскому процессу как статус завершаемого процесса; он может быть получен с помощью одной из функций семейства wait.

Функция _Exit эквивалентна функции _exit.

Эти функции никогда не возвращают управление вызвавшей их программе.

Возвращаемое значение

См. выше.

Прототип

```
#include <unistd.h>
#include <stdlib.h>
void _Exit (int status);
```

14, 15. getpid, getppid()

Описание

getpid, getppid - получение идентификатора процесса

getpid возвращает идентификатор ID текущего процесса. (Это часто используется функциями, которые генерируют уникальные имена временных файлов).

getppid возвращает идентификатор ID родительского процесса.

Возвращаемое значение

См. выше.

Прототип

```
#include <sys/types.h>
#include <unistd.h>
pid_t getpid(void);
pid_t getppid(void);
```

16, 17. getuid, geteuid

Описание

getuid, geteuid - получить идентификатор пользователя.

getuid возвращает фактический идентификатор ID пользователя в текущем процессе.

geteuid возвращает эффективный идентификатор ID пользователя в текущем процессе.

Фактический ID соответствует ID пользователя, который вызвал процесс. Эффективный ID соответствует установленному setuid биту на исполняемом файле.
Эти функции всегда завершаются успешно.

Возвращаемое значение
См. выше.

Прототип
```
#include <unistd.h>
#include <sys/types.h>
uid_t getuid(void);
uid_t geteuid(void);
```

18, 19. getgid, getegid

Описание
getgid, getegid - получить группу процесса
getgid возвращает действительный идентификатор группы текущего процесса.
getegid возвращает еффективный идентификатор группы текущего процесса.
Действительный идентификатор соответствует идентификатору вызывающего процесса. Эффективный идентификатор соответствует биту setuid на исполняемом файле.
Эти функции всегда завершаются успешно.

Возвращаемое значение
См. выше.

Прототип
```
#include <unistd.h>
#include <sys/types.h>
gid_t getgid(void);
gid_t getegid(void);
```

20. ioctl

Описание
ioctl - управление устройствами
Функция ioctl манипулирует базовыми параметрами устройств, представленных в виде специальных файлов. В частности, многими оперативными характеристиками специальных символьных файлов (например, терминалов) можно управлять через ioctl запросы. В качестве аргумента d должен быть указан открытый файловый дескриптор. Второй аргумент является кодом запроса, который зависит от устройства. Третий аргумент является указателем на память, который не имеет типа. Традиционно это char *argp (до тех пор пока в C не появился void *).
Ioctl запрос request кодирует в себе либо аргумент, который является параметром in либо аргумент, который является параметром out и кроме того размер аргумента argp в байтах. Макросы и определения, используемые в специальных ioctl запросах request находятся в файле <sys/ioctl.h>.

Возвращаемое значение

Обычно в случае успеха возвращается ноль. Некоторые ioctl используют возвращаемое значение как выходной параметр и возвращают в случае успеха неотрицательное значение. В случае ошибки возвращается -1 и значение errno устанавливается соответствующим образом.

Прототип

```
#include <sys/ioctl.h>
int ioctl(int d, int request, ...);
```

21. indir

Описание

Выполняется системный вызов в месте вызова. Выполнение возобновляется после вызова **indir.** Основная цель indir - разрешить программе хранить аргументы в системных вызовах и выполнять их вне строки в сегменте данных. Это сохраняет чистоту текстового сегмента.

Если indir выполняется косвенно, это не-op. Если команда в непрямой локации не является системным вызовом.

Возвращаемое значение

indir возвращает код ошибки EINVAL

Прототип

ASSEMBLER

(indir = 0.)

sys indir; call

Описание(оригинал)

```
The system call at the location call is executed.  Execution
resumes after the indir call.

       The  main  purpose  of  indir is to allow a program to store
arguments in system calls and
       execute them out of line in the data segment.  This
preserves the purity of the text  seg-
       ment.

       If  indir is executed indirectly, it is a no-op.  If the
instruction at the indirect loca-
       tion is not a system call, indir returns error code EINVAL
```


22. kill

Описание

Системный вызов **kill** может быть использован для посылки какого-либо сигнала какому-либо процессу или группе процесса.

Если значение *pid* является положительным, сигнал *sig* посылается процессу с идентификатором *pid*.

Если *pid* равен 0, то *sig* посылается каждому процессу, который входит в группу текущего процесса.

Если *pid* равен -1, то *sig* посылается каждому процессу, за исключением процесса с номером 1 (init), но есть нюансы, которые описываются ниже.

Если *pid* меньше чем -1, то *sig* посылается каждому процессу, который входит в группу процесса *-pid*.

Если *sig* равен 0, то никакой сигнал не посылается, а только выполняется проверка на ошибку.

Возвращаемое значение

В случае успеха, возвращается ноль. При ошибке, возвращается -1 и значение *errno* устанавливается соответствующим образом.

Прототип

#include <sys/types.h**>**

#include <signal.h**>**

int kill(pid_t *pid***, int** *sig***);**

23. umask

Описание

Наложение маски выполняет системный вызов **umask.** Данный вызов устанавливает и возвращает маску прав доступа для вновь создаваемых файлов. Umask используется open, mkdir и другими системными вызовами, которые создают файлы для изменения разрешений, размещенных на вновь созданных файлах или каталогах.

Возвращаемое значение

Возвращает предыдущее значение маски (возврат кодов ошибок не предусмотрен)

Так как каждый процесс имеет маску и любая комбинация из 9 бит считается допустимой, *umask* не предусматривает выход по ошибке. Он всегда возвращает прежнее значение маски. Чтобы узнать текущее значение маски не изменяя ее, нужно выполнить два последовательных вызова *umask*. Первый – чтобы получить текущее значение маски, второй – чтобы восстановить ее в первоначальное состояние.

Прототип

#include< sys/types.h **>**
#include < sys / stat.h **>**
mode_t umask (*маска* mode_t);

24. link

Описание

link создает новую ссылку (также известную как "жесткая" ссылка) на существующий файл.

Если *newpath* существует, он *не* будет перезаписан.

Это новое имя может использоваться точно так же, как и старое, для любых операций; оба имени ссылаются на один и тот же файл (то есть имеют те же права доступа и владельца) и невозможно сказать, какое имя было "настоящим".

Возвращаемое значение

В случае успеха возвращается ноль. При ошибке возвращается -1, а *errno* устанавливается должным образом.

Прототип

#include <unistd.h**>**

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

25. unlink

Описание

unlink удаляет имя из файловой системы. Если это имя было последней ссылкой на файл и больше нет процессов, которые держат этот файл открытым, данный файл удаляется и место, которое он занимает освобождается для дальнейшего использования. Если имя было последней ссылкой на файл, но какие-либо процессы всё ещё держат этот файл открытым, файл будет оставлен пока последний файловый дескриптор, указывающий на него, не будет закрыт.

Если имя указывает на символьную ссылку, ссылка будет удалена.

Если имя указывает на сокет, FIFO или устройство, имя будет удалено, но процессы, которые открыли любой из этих объектов могут продолжать его использовать.

Возвращаемое значение

В случае успеха возвращается ноль. В случае ошибки возвращается -1 и значение *errno* устанавливается соответствующим образом.

Прототип

#include <unistd.h**>**

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

26. lseek

Описание

Функция **lseek** устанавливает смещение для файлового дескриптора *fildes* в значение аргумента *offset* в соответствии с директивой *whence* которая может принимать одно из следующих значений:

SEEK_SET

Смещение устанавливается в *offset* байт (от начала файла -- прим. пер.).

SEEK_CUR

Смещение устанавливается как текущее смещение плюс *offset* байт.

SEEK_END

Смещение устанавливается как размер файла плюс *offset* байт.

Функция **lseek** позволяет задавать смещения, которые будут находиться за существующим концом файла (но это не изменяет размер файла). Если позднее по этому смещению будут записаны данные, то последующее чтение в промежутке от конца файла до этого смещения, будет возвращать нулевые байты (пока в этот промежуток не будут фактически записаны данные).

Возвращаемое значение

При успешном выполнеии **lseek** возвращает получившееся в результате смещение в батах от начала файла. В противном случае, возвращается значение (off_t)-1 и *errno* показывает ошибку.

Прототип

#include<sys/types.h**>**
#include <unistd.h**>**

off_t lseek(int *fildes*, off_t *offset*, int *whence*);

27. mknod

Описание

Системный вызов **mknod** создат запись inode в файловой системе (обычный файл, файл устройства или именованый канал) с именем *pathname*, с атрибутами, определяемыми параметрами *mode* и *dev*.

Параметр *mode* задает как права доступа, так и тип создаваемой записи. В этом параметре содержится битовая комбинация (используя побитовое логическое сложение OR) одного из нижеперечисленных типов файлов и прав доступа для нового узла. Права доступа изменяются, при этом значение **umask** процесса используется обычным способом: права доступа становятся равны **(mode & ~umask)**. Тип файла должен быть **S_IFREG**,**S_IFCHR**, **S_IFBLK**, **S_IFIFO** or **S_IFSOCK** для определения обычного файла (он будет создан пустым), особого символьного файла, особого блочного файла, FIFO (именованного канала) или доменного сокета Unix соответственно. (Пустой тип файла эквивалентно указанию типа S_IFREG.)

Если тип файла - **S_IFCHR** или **S_IFBLK**, то *dev* задает основной и вспомогательный номера создаваемого файла устройства; в остальных случаях параметр игнорируется. Если *pathname* уже существует, или является символьной ссылкой, то вызов завершается с ошибкой EEXIST. Созданный файл будет принадлежать фактическому владельцу процесса. Если в правах доступа к каталогу, в котором находится файл, установлен бит setgid, или если файловая система смонтирована с семантикой групп BSD, то новый файл унаследует группу-владельца от своего родительского каталога; в противном случае группой-владельцем станет фактическая группа процесса.

Возвращаемое значение

При успешном завершении работы **mknod** возвращаемое значение равно нулю, в случае ошибки оно равно -1 и переменная *errno* приобретает соответствующее значение.

Прототип

#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*);

28. mkdir

Описание

mkdir пытается создать каталог, который называется *pathname*.

mode задает права доступа, которые получит свежесозданный каталог. Эти права стандартным образом модифицируются с помощью **umask**: права доступа оказываются равны **(mode & ~umask)**.

Свежесозданный каталог принадлежит фактическому владельцу процесса. Если на родительском каталоге установлен флаг setgid, или файловая система смонтирована с семантикой групп в стиле BSD, то новый каталог унаследует группу-владельца от своего родительского каталога; в противном случае группой-владельцем станет фактическая группа процесса.

Если у родительского каталога установлен бит setgid, то он будет установлен также и у свежесозданного каталога.

Возвращаемое значение

mkdir возвращает ноль при успешном завершении или -1, если произошла ошибка (в этом случае *errno* устанавливается должным образом).

Прототип

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

#include <sys/types.h**>**

int mkdir(const char ***_pathname_, mode_t** _mode_**);**

29. rmdir

Описание

rmdir удаляет каталог, который должен быть пустым.

Возвращаемое значение

В случае успеха возвращается ноль. При ошибке возвращается -1, а *errno* устанавливается должным образом.

Прототип

#include <unistd.h**>**
int rmdir(const char ***_pathname_);**

30. mkfifo

Описание

Функция **mkfifo** создает особый FIFO-файл с названием *pathname*. *mode* определяет уровни доступа для FIFO. Они меняются с помощью процесса **umask** обычным путем: уровни доступа для созданного файла есть **(***mode* **& ~umask)**.

Особый FIFO-файл похож на обычный канал, только он создается другим путем. Вместо того, чтобы быть анонимным каналом связи, особый FIFO-файл подключается к системе с помощью вызова **mkfifo**.

Как только таким образом создан особый FIFO-файл, любой процесс может открыть его для чтения или записи так же, как и любой обычный файл. Тем не менее, он должен быть открытым в обоих состояниях одновременно, прежде чем Вы захотите провести в нем операции ввода или вывода. Однако, открытие FIFO для чтения обычно блокирует его до тех пор, пока другой процесс не откроет этот же FIFO для записи, и наоборот.

Возвращаемое значение

Обычно возвращаемое значение при успешном завершении работы *mkfifo* равно **0**. В случае ошибки возвращается **-1**, при этом значение переменной *errno* изменяется соответственно ошибке.

Прототип

#include <sys/types.h**>**
#include <sys/stat.h**>**
int mkfifo (const char ***_pathname_, mode_t** _mode_**);**

31. mount

Описание

mount подключает файловую систему, определяемую *source* (обычно это имя является названием устройства, но может также быть именем каталога или dummy), к каталогу, заданному в *target*.

umount и **umount2** отключает (самую последнюю) файловую систему, подключенную к *target*.

Подключать и отключать файловые системы может только суперпользователь. Начиная с Linux 2.4 одна файловая система может быть видна на нескольких точках подключения, а множество подключений может быть собрано в одной точке.

Значения для аргумента *filesystemtype* поддерживаемые ядром, перечислены в */proc/filesystems* (например: "minix", "ext2", "msdos", "proc", "nfs", "iso9660" и т.п.). остальные типы могут стать доступными, если будут загружены соответствующие модули.

Возвращаемое значение

При удачном завершении вызова возвращаемое значение равно нулю. При ошибке возвращается -1, а переменной *errno* присваивается номер ошибки.

Прототип

#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*);**
int umount2(const char ****target*, int** *flags***);**

32. umount

См.31

33. nice

Описание

nice добавляет значение аргумента *inc* к значению приоритета nice вызывающего процесса (большее значение nice означает меньший приоритет). Только суперпользователь может задавать отрицательные значения или определять увеличение приоритета.

Возвращаемое значение

При удачном завершении вызова возвращаемое значение равно нулю. При ошибке оно равно -1, а переменной *errno* присваивается номер ошибки.

Прототип

#include <unistd.h**>**
int nice(int *inc***);**

34. pause

Описание

После вызова функции **pause** вызывающий процесс (или подзадача) приостанавливается до тех пор, пока не получит сигнал. Данный сигнал либо остановит процесс, либо заставит его вызвать функцию обработки этого сигнала.

Возвращаемое значение

Функция **pause** возвращается только тогда, когда сигнал был перехвачен и произошел возврат из функции обработки сигнала. В этом случае она возвращает -1, а значение переменной *errno* становится равным **EINTR**.

Прототип

#include <unistd.h**>**

int pause(void);

35. profil

Описание

Эта функция позволяет определить, какая часть программы занимает наибольшую часть программного времени. Аргумент *buf* указывает на *bufsiz* байтов ядра. Каждые виртуальные 10 миллисекунд проверяется пользовательский счетчик (PC): вычитается *offset*, а результат умножается на *scale* и делится на 65536. Если результирующая величина меньше, чем *bufsiz*, то запись в *buf* увеличивается соответственно. Если величина *buf* равна NULL, то профилирование отключено.

Возвращаемое значение

Всегда возвращается 0.

Прототип

#include <unistd.h**>**

int profil(u_short ****buf*, size_t** *bufsiz***, size_t** *offset***, u_int** *scale***);**

36. ptrace

Описание

ptrace - это средство, позволяющее родительскому процессу наблюдать и контролировать протекание другого процесса, просматривать и изменять его данные и регистры. Обычно эта функция используется для создания точек прерывания в программе отладки и отслеживания системных вызовов.

Родительский процесс может начать трассировку, сначала вызвав функцию fork, а затем получившийся дочерний процесс может выполнить PTRACE_TRACEME, за которым (обычно) следует выполнение exec. С другой стороны, родительский процесс может начать отладку существующего процесса при помощи PTRACE_ATTACH.

При трассировке дочерний процесс останавливается каждый раз при получении сигнала, даже если этот сигнал игнорируется. (Исключением является SIGKILL, работающий обычным образом.) Родительский процесс будет уведомлен об этом при вызове wait, после которого он может просматривать и изменять содержимое дочернего процесса до его запуска. После этого родительский процесс разрешает дочернему

продолжать работу, в некоторых случаях игнорируя посылаемый ему сигнал или отправляя вместо этого другой сигнал).

По окончании трассировки родительский процесс может прекратить работу дочернего при помощи функции PTRACE_KILL или дать ему возможность работать в обычном, нетрассируемом, режиме, используя PTRACE_DETACH.

Значение аргумента *request* определяет действие, выполняемое функцией.

Возвращаемое значение

При удачном завершении работы функции запросы PTRACE_PEEK* возвращают данные памяти, а остальные запросы возвращают нулевое значение. При ошибке все запросы возвращают -1, а переменной *errno* присваивается номер ошибки. Значение, возвращаемое при удачном завершении PTRACE_PEEK*, может равняться -1, поэтому после вызова данной функции необходимо проверить содержимое *errno*, чтобы узнать, вернула ли функция ошибку или запрос был удовлетворен.

Прототип

#include <sys/ptrace.h**>**

long ptrace(enum __ptrace_request *request***, pid_t** *pid***, void *** *addr***, void *** *data***);**

37. setsid

Описание

setsid() создает новый сеанс, если вызывающий процесс не создает группу. Вызывающий процесс становится ведущим в группе, ведущим процессом нового сеанса и не имеет контролирующего терминала. Идентификаторы группы процессов и сеанса при установке будут равными идентификатору вызывающего процесса. Вызывающий процесс будет единственным в этой группе и сеансе.

Возвращаемое значение

Возвращает идентификатор сеанса вызывающего процесса.

Прототип

#include <unistd.h**>**

pid_t setsid(void);

38. setpgid

Описание

setpgid присваивает идентификатор группы процессов *pgid* тому процессу, который был определен *pid*. Если значение *pid* равно нулю, то процессу присваивается идентификатор текущего процесса. Если значение *pgid* равно нулю, то используется идентификатор процесса, указанный *pid*. Если **setpgid** используется для перевода процесса из одной группы в другую, то обе группы должны быть частью одной сессии. В этом случае *pgid*указывает на существующую группу процессов, с которой должен ассоциироваться процесс, а идентификатор сессии этой группы должен соответствовать идентификатору сессии присоединяющегося процесса. **getpgid** возвращает идентификатор группы процессов, к которой принадлежит процесс, указанный *pid*. Если значение *pid* равно нулю, то используется идентификатор текущего процесса. Вызов **setpgrp()**эквивалентен **setpgid(0,0)**.

Аналогично, значение **getpgrp()** эквивалентно **getpgid(0)**. Каждая группа процессов является компонентом сеанса, и, соответственно, каждый процесс является членом того сеанса, компонентом которого является его группа процессов. Группы процессов используются для доставки сигнала и как терминалы для разрешения запросов на ввод данных: процесс, который принадлежит той же группе, что и терминал, имеет преимущество над процессами других групп. Эти вызовы используются такими программами, как csh для создания групп процессов в целях осуществления контроля над процессами. Вызовы **TIOCGPGRP** и **TIOCSPGRP** описаны в termios и используются для получения/установления групп процессов управляющего терминала. Если у сеанса есть управляющий терминал, то CLOCAL не устанавливается и соединение разрывается, затем программе, инициализировавшей сеанс, посылается сигнал SIGHUP. Если эта программа закрывается, то сигнал SIGHUP будет послан каждому процессу в группе управляющего терминала. Если инициализирующая программа закрыта, а один из ее процессов остановлен, то каждому процессу в группе посылается сигнал SIGHUP, а за ним SIGCONT.

Возвращаемое значение

При удачном выполнении **setpgid** и **setpgrp** возвращаемое значение равно нулю. При ошибке возвращается -1, а переменной *errno* присваивается номер ошибки. **getpgid** возвращает идентификатор текущей группы процессов. При ошибке возвращается -1, а переменной *errno* присваивается номер ошибки. **getpgrp** всегда возвращает идентификатор группы процессов.

Прототип

#include <unistd.h**>**

int setpgid(pid_t *pid***, pid_t** *pgid***);**
pid_t getpgid(pid_t *pid***);**
int setpgrp(void);
pid_t getpgrp(void);

39. setuid

Описание

setuid устанавливает фактический идентификатор владельца текущего процесса. Если фактический пользователь, вызвавший эту функцию является суперпользователем, то также устанавливаются действительный и сохраненный идентификаторы.

В Linux **setuid** реализован, как и в стандарте POSIX с возможностью _POSIX_SAVED_IDS. Это позволяет setuid-программам (не setuid-root) сбрасывать все привилегии, делать непривилегированную работу, а затем безопасным путем возвращать себе исходный идентификатор эффективного пользователя.

Если пользователь root или программа установлена как setuid root, при работе требуется особая осторожность. Функция **setuid** проверяет идентификатор вызвавшего эффективного пользователя и, если это суперпользователь, то все устанавливаемые значения идентификаторов равны*uid*. После этого программа уже никаким образом не сможет вернуть права пользователя root.

Таким образом, программа setuid-root, собирающаяся временно сбросить права root, на время сделаться другим пользователем, а затем восстановить права root, не сможет вернуть **setuid**. Требуемого результата можно достичь с помощью вызова **seteuid**,; этот вызов не описан в POSIX, но описан в BSD.

Возвращаемое значение

При успешном завершении возвращается нулевое значение. В случае ошибки возвращается -1, а переменной *errno* присваивается номер ошибки.

Прототип

#include<sys/types.h**>**
#include <unistd.h**>**

int setuid(uid_t *uid***);**

40. setgid

Описание

setgid устанавливает идентификатор эффективной группы текущего процесса. Если функция вызвана суперпользователем, то устанавливаются идентификаторы действительной и сохраненной группы. В Linux **setgid** реализован так же, как в стандарте POSIX, где есть возможность _POSIX_SAVED_IDS. Это позволяет setgid-программам (не являющимся программами суперпользователя) сбросить привилегии группы, проделать непривилегированную работу и вернуть исходный идентификатор эффективной группы в полной сохранности. Если пользователь является суперпользователем или установленная программа является setgid root, то необходимо быть особенно осмотрительным. Функция **setgid** проверяет идентификатор эффективной группы вызвавшего процесса, и если это суперпользовательский процесс, то устанавливаемое значение всех идентификаторов его группы становится равным *gid*. После этого программе невозможно вернуть привилегии суперпользователя. Таким образом, программа setgid-root, собирающаяся временно сбросить привилегии суперпользователя, попасть в не-root группу, а потом восстановить привилегии суперпользователя, не может использовать **setgid**. Желаемого результата можно добиться с помощью системного вызова **setegid**, которого нет в стандарте POSIX, но он есть в BSD.

Возвращаемое значение

При успешном завершении вызова возвращается нулевое значение. При ошибке возвращается -1, а переменной *errno* присваивается номер ошибки.

Прототип

#include<sys/types.h**>**
#include <unistd.h**>**

int setgid(gid_t *gid***);**

41. seteuid

Описание

seteuid устанавливает действующий идентификатор пользователя текущего процесса. Непривилегированные пользовательские процессы могут менять действующий идентификатор пользователя только на действительный, действующий или сохраненный идентификатор пользователя. То-же самое справедливо при работе **setegid** для групп.

Возвращаемое значение

При нормальном завершении работы возвращается 0. При ошибках возвращается -1, а переменной *errno* присваивается код соответствующей ошибки.

Прототип

```
#include <sys/types.h>
#include <unistd.h>
int seteuid(uid_t euid);
int setegid(gid_t egid);
```


42. setegid

см. 41

43-45. signal, sigset, sigaction (привести таблицу сигналов с действиями по умолчанию и причинами посылки сигнала)

Описание

Системный вызов **signal**() устанавливает новый обработчик сигнала с номером *signum* в соответствии с параметром *sighandler*, который может быть функцией пользователя, **SIG_IGN** или **SIG_DFL**. При получении процессом сигнала с номером *signum* происходит следующее: если устанавливаемое значение обработчика равно**SIG_IGN**, то сигнал игнорируется; если оно равно **SIG_DFL**, то выполняются стандартные действия, связанные с сигналом (см. **signal**(7)). Наконец, если обработчик установлен в функцию *sighandler*, то сначала устанавливает значение обработчика в SIG_DFL или выполняется зависимая от реализации блокировка сигнала, а затем вызывается функция *sighandler* с параметром *signum*.

Использование функции-обработчика сигнала называется "перехватом сигнала". Сигналы **SIGKILL** и **SIGSTOP** не могут быть "перехвачены" или игнорированы.

Возвращаемое значение

Функция **signal**() возвращает предыдущее значение обработчика сигнала или **SIG_ERR** при ошибке

Прототип

```
#include <signal.h>
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
```

Описание

Системный вызов sigset задает реакцию системы на сигнал sig. Эта реакция заключается либо в вызове обработчика сигнала процесса func, либо в выполнении определенного системой действия.
Параметру sig может быть присвоено любое из приведенных значений, за исключением значения SIGKILL. Сигналы, зависящие от аппаратной или программной реализации, не приведены (см. "Примечания" ниже). Каждое значение sig является макросом, определенным в файле <signal.h> и который может быть представлен в виде выражения-константы.

Прототип

sighandler_t sigset(int *sig*, sighandler_t *disp*);

Описание

Системный вызов sigaction используется для изменения действий процесса при получении соответствующего сигнала. Параметр signum задает номер сигнала и может быть равен любому номеру, кроме SIGKILL и SIGSTOP. Если параметр act не равен нулю, то новое действие, связянное с сигналом signum, устанавливается соответственно act. Если oldact не равен нулю, то предыдущее действие записывается в oldact.

Возвращаемое значение

Функции sigaction, sigprocmask и sigpending возвращают 0 при удачном завершении работы функции и -1 при ошибке.

Прототип

#include <signal.h>
int sigaction(int *signum*, **const struct sigaction** **act*, **struct sigaction** **oldact*);

Сигнал	Номер	Действие	Комментарии
SIGINT	2	Term	Прерывание с клавиатуры
SIGQUIT	3	Core	Прекратить работу с клавиатурой
SIGILL	4	Core	Некорректная инструкция от процессора
SIGABRT	6	Core	Сигнал о прекращении, посланный *abort*(3)
SIGFPE	8	Core	Неправильная операция с "плавающей" запятой
SIGKILL	9	Term	Сигнал Kill
SIGSEGV	11	Core	Некорректное обращение к памяти
SIGPIPE	13	Term	Запись в канале, не имеющем считывающих процессов
SIGALRM	14	Term	Сигнал таймера от *alarm*(2)
SIGTERM	15	Term	Сигнал снятия
SIGUSR1	30,10,16	Term	Определяемый пользователем сигнал #1
SIGUSR2	31,12,17	Term	Определяемый пользователем сигнал #2
SIGCHLD	20,17,18	Ign	Дочерний процесс остановлен или прерван
SIGCONT	19,18,25	Продолжить в случае остановки	
SIGSTOP	17,19,23	Stop	Процесс остановлен
SIGTSTP	18,20,24	Stop	Остановка с помощью клавиатуры
SIGTTIN	21,21,26	Stop	Запрос на ввод с терминала для фонового процесса
SIGTTOU	22,22,27	Stop	Запрос на вывод с терминала для фонового процесса

46. stat

Описание

stat возвращает информацию о файле *file_name* и заполняет буфер *buf*.

Все эти функции возвращают структуру *stat*, которая содержит следующие поля:

```
struct stat {
    dev_t     st_dev;     /* устройство */
    ino_t     st_ino;     /* inode */
    mode_t    st_mode;    /* режим доступа */
    nlink_t   st_nlink;   /* количество жестких ссылок */
    uid_t     st_uid;     /* идентификатор пользователя-владельца */
    gid_t     st_gid;     /* идентификатор группы-владельца */
    dev_t     st_rdev;    /* тип устройства */
                          /* (если это устройство) */
    off_t     st_size;    /* общий размер в байтах */
    blksize_t st_blksize; /* размер блока ввода-вывода */
                          /* в файловой системе */
    blkcnt_t  st_blocks;  /* количество выделенных блоков */
    time_t    st_atime;   /* время последнего доступа */
    time_t    st_mtime;   /* время последней модификации */
    time_t    st_ctime;   /* время последнего изменения */
};
```

Прототип

#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
int stat(const char *_file_name_, struct stat *_buf_);

47. fstat

Описание

fstat идентична **stat**, только возвращается информация об открытом файле, на который указывает *filedes*(возвращаемый **open**(2)), а не о *file_name*.

Прототип

int fstat(int _filedes_, struct stat *_buf_);

48. stime

Описание

stime - устанавливает системное время

Возвращаемое значение

При удачном завершении возвращается 0. При ошибке возвращается -1, а переменной *errno* присваивается номер ошибки

Прототип

#define _SVID_SOURCE /* необходим glibc2 */
#include <time.h>
int stime(time_t *t);

49. symlink

Описание

symlink создает символьную ссылку, которая называется *frompath* и содержит строку *topath*.

Символьные ссылки интерпретируются "на лету", как будто бы содержимое ссылки было подставлено вместо пути, по которому идет поиск файла или каталога.
Символьные ссылки могут содержать такие компоненты пути, как **..** которые, (если используются в начале ссылки), ссылаются на родительский каталог того каталога, в котором находится ссылка.
Символьная ссылка (также известная как "мягкая ссылка") может указывать как на существующий, так и на несуществующий файлы; в последнем случае такая ссылка называется "висячей".

Возвращаемое значение

В случае успеха возвращается ноль. При ошибке возвращается -1, а значение *errno* устанавливается должным образом.

Прототип

#include <unistd.h>
int symlink(const char *topath, const char *frompath);

50. sync

Описание

Принудительно и экстренно сохраняет блоки данных на дисках, обновляет суперблок.

Прототип

#include <unistd.h>
void sync (void);

51. time

Описание

Функция time возвращает время в секундах, прошедшее с начала этой эпохи (00:00:00 UTC, 1 Января 1970 года). Если t не равно нулю, то возвращаемое значение будет также сохранено в памяти структуры t.

Возвращаемое значение

При удачном завершении работы функции возвращается время в секундах, прошедшее с начала этой эпохи. При ошибке возвращается ((time_t)-1), а переменной *errno* присваивается номер ошибки.

Прототип

#include <time.h>
time_t time(time_t *t);

52. ftime

Описание

Возвращает текущую дату и время в виде *tp*, определенного следующим образом:

```
struct timeb {
    time_t   time;
    unsigned short millitm;
    short    timezone;
    short    dstflag;
};
```

Возвращаемое значение

Функция всегда возвращает 0.

Прототип

```
#include <sys/timeb.h>
int ftime(struct timeb *tp);
```

53. times

Описание

Функция **times**() записывает времена текущего процесса в структуру **struct tms** на которую указывает аргумент *buf*. Функция *struct tms* определена в <*sys/times.h*> следующим образом:

```
struct  tms  {
clock_t tms_utime;  /* пользовательское время */
clock_t tms_stime;  /* системное время */
clock_t tms_cutime; /* пользовательское время дочерних процессов */
clock_t tms_cstime; /* системное время дочерних процессов */
};
```

Возвращаемое значение

Функция **times** возвращает количество тиков часов с определенного момента времени. В Linux это количество тиков часов с момента запуска системы.

Прототип

#include <sys/times.h>
clock_t times(struct tms *buf);

54. vfork

Описание

Функция *vfork*() аналогична *fork*(), **vfork()** отличается от fork тем, что родительский процесс блокируется до тех пор, пока дочерний процесс не вызовет **execve**(2) или **_exit**(2). Дочерний процесс разделяет всю память с родительским, включая стек, до тех пор, пока не вызовет *execve()*. Дочерний процесс не должен выходить из текущей функции или вызывать *exit()*, но может вызвать *_exit()*.

Прототип

#include <sys/types.h>
#include <unistd.h>
pid_t vfork(void);

55. waitpid

Описание

Функция **waitpid** приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс, указанный в параметре *pid*, не завершит выполнение, или пока не появится сигнал, который либо завершает текущий процесс либо требует вызвать функцию-обработчик. Если указанный дочерний процесс к моменту вызова функции уже завершился (так называемый "зомби"), то функция немедленно возвращается. Системные ресурсы, связанные с дочерним процессом, освобождаются.

Возвращаемое значение

Возвращает идентификатор дочернего процесса, который завершил выполнение, или ноль, если использовался**WNOHANG** и ни один дочерний процесс пока еще недоступен, или -1 в случае ошибки (в этом случае переменной*errno* присваивается соответствующее значение)

Прототип

pid_t waitpid(pid_t *pid***, int ****status***, int** *options***);**

56. execlp

Описание

Семейство функций **exec** заменяет текущий образ процесса новым образом процесса.

Начальным параметром этих функций будет являться полное имя файла, который необходимо исполнить. Параметр *const char *arg* и аналогичные записи в функциях **execl**, **execlp**, и **execle** подразумевают параметры *arg0*,*arg1*, ..., *argn*. Все вместе они описывают один или нескольких указателей на строки, заканчивающиеся NULL, которые представляют собой список параметров, доступных исполняемой программе. Первый параметр, по соглашению, должен указывать на имя, ассоциированное с файлом, который надо исполнить. Список параметров*должен* заканчиваться **NULL**.

!!! Функции **execlp** и **execvp** дублируют действия оболочки, относящиеся к поиску исполняемого файла, если указанное имя файла не содержит символ черты (/). Путь поиска определяется в окружении переменной **PATH** .Если эта переменная не определена, то используется путь поиска ":/bin:/usr/bin" по умолчанию. Дополнительно обрабатываются некоторые ошибки.

!!! Если запрещен доступ к файлу (при попытке исполнения **execve** была возвращена ошибка **EACCES**), то эти функции будут продолжать поиск вдоль оставшегося пути. Если не найдено больше никаких файлов, то по возвращении они установят значение глобальной переменной *errno* равным **EACCES**.
Если заголовок файла не распознается (при попытке выполнения функции **execve** была возвращена ошибка**ENOEXEC**), то эти функции запустят оболочку (shell) с полным именем файла в качестве первого параметра. (Если и эта попытка будет неудачна, то дальнейший поиск не производится.)

Возвращаемое значение

Возвращение значения какой-либо из функций **exec** приведет к ошибке. При этом возвращаемым значением будет -1 и глобальной переменной *errno* будет присвоен код соответствующей ошибки.

Прототип

#include <unistd.h>

extern char **environ;
int execl(const char *_path_, const char *_arg_, ...);
int execlp(const char *_file_, const char *_arg_, ...);
int execle(const char *_path_, const char *_arg_ , ..., char * const _envp_[]);
int execv(const char *_path_, char *const _argv_[]);
int execvp(const char *_file_, char *const _argv_[]);

57. execle

см. 56

58. execvp

см. 56

59. execv

см. 56

60. execve

см. 56

61. popen

Описание

Сначала popen() создает канал, затем порождает процесс, исполняющий команду, заданную первым аргументом, и перенаправляет ее стандартный ввод или вывод. Функция popen() реализована с использованием pipe() для создания канала, fork() для запуска подпроцесса и dup() для перенаправления стандартного ввода или вывода в канал.
Аргумент *command* - это указатель на строку символов, содержащую любую правильную команду интерпретатора shell.
Аргумент *type* - "r" для чтения и "w" для записи. Выбор "r" или "w" определяется тем, как программа будет использовать полученный указатель на файл.

- type должен быть "r", если программа хочет читать вывод, производимый стандартным выводом command
- type должен быть "w", если вывод в полученный указатель на файл должен быть стандартным вводом command

Так как канал задается однонаправленным, аргумент *type* может указать только на режим чтения или записи, но не на оба одновременно.

Возвращаемое значение

Возвращаемое значение popen() - это обычный буферизованный поток ввода-вывода (указатель на стандартный библиотечный тип FILE) (за исключением того, что он должен быть закрыт только функцией pclose() ,а не fclose()). Функция popen возвращает NULL, если вызовы fork или pipe завершились ошибкой или если невозможно выделить необходимый для этого объем памяти.

Прототип

```
#include <stdio.h>
FILE *popen(const char *command, const char *type);
```

62. pclose

Описание

Закрывает канал, открытый popen и ждет завершения порожденного процесса. Функция pclose() использует close(), чтобы закрыть канал, и wait() для того, чтобы дождаться завершения порожденного процесса.
Аргумент *stream* - указатель на FILE, полученный при вызове popen()

Возвращаемое значение

Функция **pclose** возвращает -1, если была обнаружена какая-либо ошибка, в случае успеха - статус подпроцесса.

Прототип

```
#include <stdio.h>
int pclose(FILE *stream)
```

63. sigprocmask

Описание

С помощью данного системного вызова процесс может запрашивать и создавать/изменять сигнальную маску процесса.
new - определяет набор сигналов, которые будут добавлены (1) в сигнальную маску вызывающего процесса или удалены из нее
cmd - указывает, как значение new должно быть использовано:

- SIG_SETMASK - заменяет сигнальную маску процесса значением, указанным в аргументе new
- SIG_BLOCK - добавляет сигналы, указанные в аргументе new в сигнальную маску процесса
- SIG_UNBLOCK - удаляет из сигнальной маски процесса сигналы, указанные в аргументе new

Если new = 0, то аргумент cmd игнорируется, и сигнальная маска текущего процесса не изменяется.
old - переменная, которой присваивается сигнальная маска процесса до вызова функции sigprocmask()

Возвращаемое значение

В случае неудачи возвращает -1, в случае удачи 0.

Прототип

```
#include <signal.h>
int sigprocmask(int cmd, const sigset_t *new, sigset_t *old);
```

64. sigpending

Описание

Системный вызов **sigpending** позволяет определить наличие ожидающих сигналов (полученных заблокированных сигналов). Маска ожидающих сигналов помещается в *set*.

Возвращаемое значение

В случае неудачи возвращает -1, в случае удачи 0.

Прототип

```
#include <signal.h>
int sigpending(sigset_t *set);
```

65. setitimer

Описание

Считывает или устанавливает значение таймера равным величине, указанной в *new_value*. Система предоставляет каждому процессу три таймера, значение каждого из которых уменьшается на единицу по истечении установленного времени. Когда на одном из таймеров истекает время, то процессу отправляется сигнал и таймер (обычно) перезапускается.
which - указывает таймер, значение которого надо изменить:

- ITIMER_REAL - уменьшается в режиме реального времени и подает сигнал SIGALRM, когда значение таймера становится равным 0.
- ITIMER_VIRTUAL - уменьшается только во время работы процесса и подает сигнал SIGVTALRM, когда значение таймера становится равным 0.
- ITIMER_PROF - уменьшается во время работы процесса и когда система выполняет что-либо по заданию процесса. Когда значение таймера становится равным 0, подается сигнал SIGPROF.

new_value - определяет новое значение таймера
Если величина *old_value* не равна нулю, то в нее записывается прежнее значение таймера.

Возвращаемое значение
В случае неудачи возвращает -1, в случае удачи 0.

Прототип
```
#include <sys/time.h>
int setitimer(int which, const struct itimerval *new_value, struct itimerval *old_value);
```

66. getitimer

Описание
Позволяет получить текущее значение указанного таймера.
which - указывает таймер, значение которого надо получить
curr_value - структура, которая заполняется текущим значением таймера, указанного в which.

Возвращаемое значение
В случае неудачи возвращает -1, в случае удачи 0.

Прототип
```
#include <sys/time.h>
int getitimer(int which, struct itimerval *curr_value);
```

67. msgget

Описание
Получить доступ к очереди сообщений.
В качестве ключа key может быть использовано любое длинное целое. Ключ
может быть получен использованием ftok() или значения IPC_PRIVATE.
msgflg управляет созданием и правами доступа очереди. Его значение получается побитовым ИЛИ следующих констант:
. IPC_CREAT - если не существует очереди с этим ключом, создает ее.
. IPC_EXCL - только вместе с IPC_CREAT. Очередь создается тогда и только тогда, когда ее не существует. Иными словами, когда заданы IPC_CREAT | IPC_EXCL, и уже существует очередь с заданным ключом, системный вызов возвратит неуспех.
. Девять младших бит msgflg используются для задания прав доступа - право чтения определяет возможность получать сообщения, а право записи - посылать их.

Возвращаемое значение

В случае неудачи возвращает -1, в случае удачи идентификатор очереди сообщений.

Прототип

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgget(key_t key, int msgflg);
```

68. msgsnd

Описание

Послать сообщение в очередь сообщений.
msgid задает очередь, в которую нужно поместить сообщение;
msgp указывает на объект типа struct msgbuf, в котором содержится сообщение;
msgsz - число байтов, которые нужно послать;
Если в очереди недостаточно места, то по умолчанию msgsnd блокируется.
msgflg задает действия на случай, если сообщение не может быть поставлено в очередь.
Параметр *msgflg* состоит из комбинации следующих флагов:

- *PC_NOWAIT* - если он указан, то если в очереди недостаточно места, msgsnd завершается с -1
- MSG_NOERROR - используется для урезания текста сообщения, размер которого больше максимального количества байтов, которые могут стоять в очереди

Возвращаемое значение

В случае неудачи возвращает -1, в случае удачи 0.

Прототип

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

69. msgrcv

Описание

Получить сообщение из очереди.
msgid - определяет очередь, из которой будет получено сообщение;
msgp - указывает на область памяти, в которую сообщение будет записано. Эта область памяти должна содержать long для типа сообщения, за которым должен следовать буфер, достаточный для самого большого сообщения, которое процесс хотел бы получить.
msgsz - показывает максимальный размер сообщения, которое процесс хотел бы получить. Это значение должно быть меньше или равно размера буфера, который следует за long.
Если длина текста сообщения больше, чем *msgsz*, и флаг *msgflg* установлен в MSG_NOERROR, то текст сообщения будет урезан (а урезанная часть потеряна), иначе сообщение не удаляется из очереди, а системный вызов возвращает ошибку.
Параметр *msgtyp* задает тип сообщения следующим образом:
если msgtyp = нулю, то используется первое сообщение из очереди;
если msgtyp >нуля, то из очереди берется первое сообщение типа msgtyp.
Если msgtyp < нуля, то из очереди берется первое сообщение со значением, меньшим, чем абсолютное значение msgtyp.

Параметр *msgflg* состоит из комбинации следующих флагов:

- *PC_NOWAIT*, который указывает на немедленный возврат из функции, если в очереди нет сообщений необходимого типа. При этом системный вызов возвращает ошибку.
- *MSG_EXCEPT* - используется (если *msgtyp* больше, чем **0**) для чтения первого сообщения, тип которого не равен *msgtyp*;
- MSG_NOERROR, используемый для урезания текста сообщения, размер которого больше *msgsz* байтов.

Возвращаемое значение

В случае неудачи возвращает -1, в случае удачи количество байтов, записанных в элемент структуры, на которую указывает msgp (количество прочитанных байтов).

Прототип

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int
msgflg);
```

70. msgctl

Описание

Эта функция выполняет операцию, заданную в *cmd*, над очередью сообщений с идентификатором *msqid* (определить состояние очереди; изменить хозяина или права доступа к ней; изменить максимальный размер очереди или удалить ее).
Возможные значения *cmd*:
IPC_STAT - записывает состояние очереди в структуру, на которую указывает buf.
IPC_SET - используется для установки пользователя, группы и
права чтения/записи для пользователя, группы и других. Значения берутся из структуры, на которую указывает buf.
IPC_RMID - удаляет очередь сообщений, ее дескриптор и связанную с ним структуру данных, "разбудив" все процессы, ожидающие записи или чтения этой очереди
(еще есть другие, но используются в основном только эти)

Возвращаемое значение

В случае неудачи возвращает -1, в случае удачи при указанных значениях cmd возвращается 0 (при других значениях cmd возвращаемое значение зависит от того, что указано в cmd, в тех самых редких случаях).

Прототип

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

71. semget

Описание

Создать набор из одного или нескольких семафоров, или получить доступ к существующему набору.
key - ключ доступа к набору. Похож на имя файла. В качестве ключа может использоваться любое целое значение. Различные пользователи набора должны договориться об уникальном значении ключа. Ключ может быть создан библиотечной функцией ftok(). Если необходим приватный ключ, может быть использовано значение IPC_PRIVATE.
nsems - количество семафоров в наборе. Это значение должно быть больше или равно 1. Семафор задается идентификатором набора и индексом в этом наборе. Индекс меняется от нуля до nsems-1. Аргумент *nsems* может быть равен **0**, если набор семафоров не создается.
semflg - биты прав доступа и флаги, используемые при создании набора. Девять младших битов задают права доступа для владельца, группы и других пользователей. Для набора семафоров определены права чтения и изменения. Флаги таковы:
IPC_CREAT - если этот флаг установлен и набор не существует, он будет создан. Если же набор с таким ключом уже существует и не задан флаг IPC_EXCL, то semget() возвратит его идентификатор.
IPC_EXCL - этот флаг используется только вместе с IPC_CREAT. Он используется для того, чтобы создать набор только тогда, когда такого набора еще не существует.
Если он используется совместно с IPC_CREAT при наличии уже созданного набора семафоров, вызов semget завершится с ошибкой.

Возвращаемое значение

В случае неудачи возвращает -1, в случае удачи - идентификатор набора семафоров.

Прототип

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semget(key_t key, int nsems, int semflg);
```

72. semop

Описание

Функция позволяет выполнять операции над одним или более семафоров из набора с идентификатором *semid*. Операции увеличивают или уменьшают значение семафора на заданную величину, или ожидают, пока семафор не станет нулевым.
sops - адрес массива с элементами типа struct sembuf, задающих операции над семафорами. По этому адресу должно размещаться nsops таких структур. Эта структура определена следующим образом:

```
sys/sem.h:
struct sembuf {
   ushort sem_num; /* semaphore */
   short sem_op;   /* semaphore operation */
   short sem_flg;  /* operation flags */
};
```

Возвращаемое значение

В случае неудачи возвращает -1, в случае удачи 0.

Прототип

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semop(int semid, struct sembuf *sops, unsigned nsops);
```

73. semctl

Описание

Функция позволяет выполнять операции, определенные в *cmd* над набором семафоров с идентификатором *semid* или над семафором с номером *semnum* из этого набора. (Семафоры нумеруются, начиная с 0). Тип и наличие последнего параметра зависит от значения команды cmd.

Возвращаемое значение

В случае неудачи возвращает -1, в случае удачи возвращаемое значение зависит от cmd.

Прототип

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semctl(int semid, int semnum, int cmd, ...);
```

74. shmget

Описание

Создать или получить доступ к сегменту разделяемой памяти.
Присваивает идентификатор разделяемому сегменту памяти. Значение key может быть получено с использованием ftok() или установлено в IPC_PRIVATE.
size - размер сегмента в байтах, shmflg управляет созданием и правами доступа к сегменту.

Возвращаемое значение

При удачном завершении вызова возвращается неотрицательный идентификатор разделяемого сегмента, и -1 при ошибке.

Прототип

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key, size_t size, int shmflg);
```

75. shmat

Описание

Присоединяет сегмент разделяемой памяти к адресному пространству вызывающего процесса.
shmid является идентификатором разделяемого сегмента, полученным от успешного вызова shmget().
shmaddr, вместе с shmflg, определяет адрес, по которому будет помещен присоединенный сегмент. Если shmaddr равен нулю, система определяет адрес сама. Если shmaddr не равен нулю, его значение будет взято в качестве адреса.

Возвращаемое значение

При удачном завершении вызова возвращается виртуальный адрес присоединенного сегмента, и -1 при ошибке.

Прототип

```
#include <sys/ipc.h>
#include <sys/shm.h>
void *shmat(int shmid, void *shmaddr, int shmflg);
```

76. shmdt

Описание

Отсоединяет разделяемый сегмент, присоединенный по адресу shmaddr.

Возвращаемое значение

В случае неудачи возвращает -1, в случае удачи 0.

Прототип

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmdt (void *shmaddr);
```

77. shmctl

Описание

Операции по управлению разделяемой памятью.
shmid - это идентификатор разделяемого сегмента, полученный от успешного вызова shmget().
Команда cmd может принимать несколько значений. (прим. IPC_STAT - копирует информацию о состоянии разделяемого сегмента в структуру, на которую указывает buf).

Возвращаемое значение

В случае неудачи возвращает -1, в случае удачи 0.

Прототип

```
#include <sys/ipc.h>
#include <sys/shm.h>
```

```
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

78. socket

Описание

Socket создает конечную точку соединения - сокет (программный интерфейс для обеспечения обмена данными между процессами) и возвращает его описатель (дескриптор).
Параметр *domain* задает домен соединения: выбирает набор протоколов, которые будут использоваться для создания соединения. Сокет имеет тип *type*, задающий семантику коммуникации. Параметр *protocol* задает конкретный протокол, который работает с сокетом.

Возвращаемое значение

В случае неудачи возвращает -1, в случае удачи дескриптор нового сокета.

Прототип

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```


79. bind

Описание

Bind привязывает к сокету *sockfd* локальный адрес *addr* длиной *addrlen*. Традиционно, эта операция называется присваивание сокету имени. Когда сокет только что создан с помощью socket(), он существует в пространстве имён (семействе адресов), но не имеет назначенного имени.

Возвращаемое значение

В случае неудачи возвращает -1, в случае удачи 0.

Прототип

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

80. listen

Описание

Listen отмечает сокет, на который ссылается sockfd как пассивный сокет, то есть как сокет, который будет использоваться для приема входящих запросов на соединение с использованием accept().
Параметр *backlog* задает максимальную длину, до которой может расти очередь ожидающих соединений.

Возвращаемое значение

В случае неудачи возвращает -1, в случае удачи 0.

Прототип

```
#include <sys/types.h>
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

81. accept

Описание

Функция **accept** используется с сокетами, ориентированными на устанавление соединения (**SOCK_STREAM**, **SOCK_SEQPACKET** и **SOCK_RDM**). Эта функция извлекает первый запрос на соединение из очереди ожидающих соединений, создаёт новый подключенный сокет почти с такими же параметрами, что и у *s*, и выделяет для сокета новый файловый дескриптор, который и возвращается.

Возвращаемое значение

Этот системный вызов возвращает -1 в случае ошибки. При успешном завершении возвращается неотрицательное целое, являющееся дескриптором сокета.

Прототип

#include <sys/types.h>
#include <sys/socket.h>
int accept(int *s***, struct sockaddr ****addr***, socklen_t ****addrlen***);**

82. connect

Описание

connect - инициирует соединение на сокете. Файловый дескриптор *sockfd* должен ссылаться на сокет. Если сокет имеет тип **SOCK_DGRAM**, значит, адрес *serv_addr* является адресом по умолчанию, куда посылаются датаграммы, и единственным адресом, откуда они принимаются. Если сокет имеет тип **SOCK_STREAM** или **SOCK_SEQPACKET**, то данный системный вызов попытается установить соединение с другим сокетом. Другой сокет задан параметром *serv_addr*, являющийся адресом длиной *addrelen* в пространстве коммуникации сокета. Каждое пространство коммуникации интерпретирует параметр *serv_addr* по-своему.
Обычно сокеты с протоколами, основанными на соединении, могут устанавливать соединение только один раз; сокеты с протоколами без соединения могут использовать **connect** многократно, чтобы изменить адрес назначения

Возвращаемое значение

Если соединение или привязка прошла успешно, возвращается нуль. При ошибке возвращается -1, а *errno* устанавливается должным образом.

Прототип

#include <sys/types.h>
#include <sys/socket.h>
int connect(int *sockfd***, const struct sockaddr ****serv_addr***, socklen_t** *addrlen***);**

83-84. send, sendto

Описание

send, **sendto** используются для пересылки сообщений в другой сокет. **send** можно использовать, только если сокет находится в состоянии *соединения*, тогда как **sendto** можно использовать в любое время. Адрес получателя задается параметром *to* длиной *tolen*. Длина сообщения задается параметром *len*. Если сообщение слишком длинное, чтобы быть отосланным протоколом нижнего уровня, возвращается ошибка **EMSGSIZE**, а сообщение не отсылается.

Возвращаемое значение

Эти системные вызовы возвращают количество отправленных символов или -1, если произошла ошибка.

Прототип

#include <sys/types.h>
#include <sys/socket.h>
ssize_t send(int *s*, **const void ***msg*, **size_t** *len*, **int** *flags*);**
ssize_t sendto(int *s*, **const void ***msg*, **size_t** *len*, **int** *flags*, **const struct sockaddr ***to*, **socklen_t** *tolen*);**

85-86. recv, recvfrom

Описание

Системные вызовы **recvfrom** используются для получения сообщений из сокета, и могут использоваться для получения данных, независимо от того, является ли сокет ориентированным на соединения или нет.
Если параметр *from* не равен **NULL**, а сокет не является ориентированным на соединения, то адрес отправителя в сообщении не заполняется. Аргумент *fromlen* передается по ссылке, в начале инициализируется размером буфера, связанного с *from*, а при возврате из функции содержит действительный размер адреса.
Вызов **recv** обычно используется только на *соединенном* сокете (см. **connect**(2)) и идентичен вызову **recvfrom** с параметром *from*, установленным в **NULL**.
Все две функции возвращают длину сообщения при успешном завершении. Если сообщение слишком длинное и не поместилось в предоставленный буфер, лишние байты могут быть отброшены, в зависимости от типа сокета, на котором принимаются сообщения .

Возвращаемое значение

Эти системные вызовы возвращают количество принятых байт или -1, если произошла ошибка.

Прототип

#include <sys/types.h>
#include <sys/socket.h>
int recv(int *s*, **void ***buf*, **size_t** *len*, **int** *flags*);**
int recvfrom(int *s*, **void ***buf*, **size_t** *len*, **int** *flags*, **struct sockaddr ***from*, **socklen_t ***fromlen*);**

87. shutdown

Описание

Системный вызов **shutdown** приводит к закрытию всего полнодуплексного соединения или его части в сокете, связанном с описателем *s*. Если параметр *how* равно **SHUT_RD**, то запрещен прием данных. Если *how* равно **SHUT_WR**, то запрещена передача данных. Если *how* равно **SHUT_RDWR**, то запрещены как прием, так и передача данных.

Возвращаемое значение

В случае успешного завершения вызова возвращается нулевое значение. При ошибке возвращается -1, а переменной *errno* присваивается номер ошибки.

Прототип

#include <sys/socket.h>
int shutdown(int *s*, **int** *how*);**

88-89. mmap,munmap

Описание

Функция **mmap** отражает *length* байтов, начиная со смещения *offset* файла (или другого объекта), определенного файловым описателем *fd*, в память, начиная с адреса *start*. Последний параметр (адрес) необязателен, и обычно бывает равен 0. Настоящее местоположение отраженных данных возвращается самой функцией **mmap**, и никогда не бывает равным 0. Аргумент *prot* описывает желаемый режим защиты памяти (он не должен конфликтовать с режимом открытия файла). *fd* должно быть корректным описателем файла. *offset* должен быть пропорционален размеру страницы. Адрес *start* должно быть кратен размеру страницы.
Системный вызов **munmap** удаляет все отражения из заданной области памяти, после чего все ссылки на данную область будут вызывать ошибку "неправильное обращение к памяти" (invalid memory reference). Отражение удаляется автоматически при завершении процесса. С другой стороны, закрытие файла не приведет к снятию отражения

Возвращаемое значение

При удачном выполнении **mmap** возвращает указатель на область с отраженными данными. При ошибке возвращается значение MAP_FAILED (-1), а переменная *errno* приобретает соответствующее значение. При удачном выполнении **munmap** возвращаемое значение равно нулю. При ошибке возвращается -1, а переменная *errno* приобретает соответствующее значение.

Прототип

#include <unistd.h>
#include <sys/mman.h>
#ifdef _POSIX_MAPPED_FILES
void * mmap(void **start*, size_t *length*, int *prot* , int *flags*, int *fd*, off_t *offset*);
int munmap(void **start*, size_t *length*);
#endif

90. msync

Описание

msync записывает на диск изменения, внесенные в файл, отраженный в память при помощи функции **mmap** (2). Если не использовать эту функцию, то нет никакой гарантии, что изменения будут записаны в файл до вызова **munmap** (2). Если быть точнее, то на диск записывается часть файла, начинающаяся в памяти с адреса *start* длиной *length*.

Возвращаемое значение

При удачном завершении вызова возвращаемое значение равно нулю. При ошибке оно равно -1, а переменной *errno* присваивается номер ошибки.

Прототип

#include <unistd.h>
#include <sys/mman.h>
#ifdef _POSIX_MAPPED_FILES
#ifdef _POSIX_SYNCHRONIZED_IO
int msync(void **start*, size_t *length*, int *flags*);
#endif
#endif

91. opendir

Описание

Функция **opendir()** открывает поток каталога, соответствующий каталогу *name*, и возвращает указатель на этот поток. Поток устанавливается на первой записи в каталоге.

Возвращаемое значение

Функция **opendir()** возвращает указатель на поток каталога или NULL в случае ошибок.

Прототип

#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char *_name_);

92. closedir

Описание

Функция **closedir()** закрывает поток, связанный с каталогом *dir*. Описатель потока *dir* будет недоступен после вызова этой функции.

Возвращаемое значение

Функция **closedir()** возвращает 0 в случае удачного завершения работы, или -1 при ошибке.

Прототип

```
#include <sys/types.h>
#include <dirent.h>
int closedir(DIR *dir);
```

93. readdir

Описание

Функция **readdir()** возвращает указатель на следующую запись каталога в структуре dirent, прочитанную из потока каталога. Каталог указан в *dir*. Функция возвращает NULL по достижении последней записи или если была обнаружена ошибка. Данные, возвращаемые **readdir()** могут быть переписаны последующими вызовами **readdir()** для того же каталожного потока.

Возвращаемое значение

Функция **readdir()** возвращает указатель на структуру dirent или NULL в случае ошибки или по достижении последней записи.

Прототип

```
#include <sys/types.h>
#include <dirent.h>
struct dirent *readdir(DIR *dir);
```

94. rewinddir

Описание

Функция **rewinddir()** устанавливает новую позицию потока каталога *dir* в начале каталога.

Возвращаемое значение

Функция **rewinddir()** не возвращает значений.

Прототип

```
#include <sys/types.h>
#include <dirent.h>
void rewinddir(DIR *dir);
```

95. scandir

Описание

Функция **scandir()** производит поиск элементов в каталоге *dir*, посылая каждому элементу вызов **select()**. Элементы, которым **select()** возвращает ненулевое значение, записываются в строках и размещаются в памяти при помощи **malloc()**; они сортируются посредством функции **qsort()** и функции сравнения **compar()**, а затем накапливаются в массиве *namelist* , который размещается в памяти функцией **malloc()**. Если *select* равен NULL, то выбираются все записи.

Возвращаемое значение

Функция **scandir()** возвращает количество выбранных записей или -1, если произошла ошибка.

Прототип

```
#include <dirent.h>
int scandir(const char *dir, struct dirent ***namelist,
      int(*select)(const struct dirent *),
      int(*compar)(const struct dirent **, const struct dirent **));
```

96-97. cfsetospeed, cfsetispeed

Описание

cfsetospeed() устанавливает скорость вывода данных, сохраненную в структуре **termios**, на которую указывает *termios_p*, в *speed*, которая должна быть равна одной из констант.
cfsetispeed() устанавливает значение скорости ввода данных, сохраненной в структуре **termios**, равное *speed*. Если скорость ввода данных меняется на ноль, то скорость ввода данных будет равна скорости вывода данных

Возвращаемое значение

Эти функции возвращают 0 при нормальном завершении работы, -1 при ошибках (при этом значение переменной *errno* меняется на значение, соответствующее ошибке).

Прототип

#include <termios.h>
#include <unistd.h>
int cfsetispeed(struct termios *termios_p, speed_t speed);
int cfsetospeed(struct termios *termios_p, speed_t speed);

98-99. cfgetospeed, cfgetispeed

Описание

cfgetospeed() возвращает скорость вывода данных, сохраненную в структуре **termios**, на которую указывает *termios_p*.
cfgetispeed() возвращает скорость ввода данных, сохраненную в структуре **termios.**

Возвращаемое значение

cfgetispeed() возвращает скорость ввода данных, сохраненную в структуре **termios**.
cfgetospeed() возвращает скорость вывода данных, сохраненную в структуре **termios**

Прототип

#include <termios.h>
#include <unistd.h>
speed_t cfgetispeed(struct termios *_termios_p_);
speed_t cfgetospeed(struct termios *_termios_p_);

100-101. tcgetattr, tcsetattr

Описание

tcgetattr() : получить параметры, связанные с объектом, на который ссылается *fd*, и сохранить их в структуре **termios**, на которую ссылается *termios_p*. Эта функция может быть запущена из фонового процесса; однако, атрибуты терминала могут в дальнейшем изменяться основным процессом.
tcsetattr(): меняет параметры, связанные с терминалом (если требуется поддержка используемого оборудования, которая недоступна), и параметры структуры **termios**, связанной с *termios_p*.

Возвращаемое значение

Эти функции возвращают 0 при нормальном завершении работы, -1 при ошибках (при этом значение переменной *errno* меняется на значение, соответствующее ошибке).
tcsetattr() сообщает об успешном завершении, если *хотя бы одно* из запрошенных изменений может быть успешно выполнено. Поэтому, при необходимости одновременного изменения нескольких параметров, может понадобится после этой функции вызвать **tcgetattr()** для того, чтобы убедиться, что все изменения были выполнены успешно.

Прототип

#include <termios.h>
#include <unistd.h>
speed_t cfgetispeed(struct termios *_termios_p_);
speed_t cfgetospeed(struct termios *_termios_p_);