Лабораторная работа №1

1. **Что такое внутренние и внешние команды Shell-интерпретатора? Приведите примеры внутренних команд.**

Как правило, при получении команды интерпретатор определяет имя файла, содержащего программу, которую необходимо выполнить, и порождает процесс, который работает под управлением заданной программы.

Однако, существует ряд функций, выполнение которых в качестве отдельного процесса либо неэффективно, либо невозможно.

Команды, выполняющиеся непосредственно интерпретатором (без порождения процесса), называются встроенными командами языка. С точки зрения пользователя встроенные команды практически не отличаются по своим свойствам от остальных команд системы, за исключением того, что для них обычно нельзя переопределить стандартные файлы ввода/вывода.

exit [**N**]

Прерывание выполнение текущего процесса. Сообщается код завершения **N**. Если параметр **N** отсутствует, то используется код завершения последней выполненной команды.

times

Выдается время, затраченное пользователем и системой на выполнение процесса.

wait [**N**]

Ожидает окончания выполнения процесса с номером (**N**) и присваивает его код завершения макропеременной **?**.

cd [**справочник**]

Объявить указанный справочник текущим. Если параметр не задан, в качестве имени справочника используется значение макропеременной HOME. Синонимом команды cd является команда chdir.

: Эта команда не выполняет никаких действий, ее код завершения равен нулю. Тем не менее производится подстановка значений макропеременных.

2. **Какие существуют средства группирования команд? Приведите примеры использования.**

; и <перевод строки>	последовательное выполнение команд;
&	асинхронное (фоновое) выполнение предшествующей команды; Символ &, завершающий строку, указывает интерпретатору команд, что следует перейти к обработке следующей команды без ожидания окончания трансляции.
&&	выполнение последующей команды при условии нормального завершения предыдущей, иначе игнорировать;
||	выполнение последующей команды при ненормальном завершении предыдущей, иначе игнорировать.

- cmd1 arg ...; cmd2 arg ...; ... cmdN arg ... - последовательное выполнение команд;
- cmd1 arg ...& cmd2 arg ...& ... cmdN arg ... - асинхронное выполнение команд;
- cmd1 arg ... && cmd2 arg ... - зависимость последующей команды от предыдущей таким образом, что последующая команда выполняется, если предыдущая выдала нулевое значение;
- cmd1 arg ... || cmd2 arg ... - зависимость последующей команды от предыдущей таким образом, что последующая команда выполняется, если предыдущая выдала ненулевое значение;

3. Как осуществляется перенаправление ввода-вывода?

- cmd > file - стандартный вывод направлен в файл file;
- cmd >> file - стандартный вывод направлен в конец файла file;
- cmd < file - стандартный ввод выполняется из файла file;
- cmd1 | cmd2 - конвейер команд, в котором стандартный вывод команды cmd1 направлен на стандартный вход команды cmd2.

4. В чем сущность конвейера команд? Приведите примеры использования.

Стандартный ввод (вход) - "stdin" в ОС UNIX осуществляется с клавиатуры терминала, а стандартный вывод (выход) - "stdout" направлен на экран терминала. Существует еще и стандартный файл диагностических сообщений - "stderr". Стандартные файлы имеют номера: 0 - stdin, 1 - stdout и 2 - stderr.

Команда, которая может работать со стандартным входом и выходом, называется ФИЛЬТРОМ. Средство, объединяющее стандартный выход одной команды со стандартным входом другой, называется КОНВЕЙЕРОМ и обозначается вертикальной чертой "|".

Например, в результате работы команд:

ls | wc -l

список файлов текущего каталога будет направлен на вход команды "wc", которая на экран выведет число строк каталога.

В конвейере каждая команда работает как параллельный независимый процесс, информация передается в одном направлении. Если входные данные процесса-получателя еще не готовы, или выходные данные процесса-источника не успевают обрабатываться, то выполнение соответствующего процесса приостанавливается.

5. Как средствами Shell выполнить арифметические действия над Shell-переменной?

Команда "expr" работает с shell-переменными как с целыми числами, а не как со строками:

n=`expr $n + 1`

при каждом выполнении значение "n" увеличивается на 1.

6. Каковы правила генерации имен файлов?

При генерации имен используют метасимволы:

*****	произвольная (возможно пустая) последовательность символов;
?	один произвольный символ;
[...]	любой из символов, указанных в скобках перечислением и/или с указанием диапазона;
cat f*	выдаст все файлы каталога, начинающиеся с "f";
cat *f*	выдаст все файлы, содержащие "f";
cat program.?	выдаст файлы данного каталога с однобуквенными расширениями, скажем "program.c" и "program.o", но не выдаст "program.com";
cat [a-d]*	выдаст файлы, которые начинаются с "a", "b", "c", "d". Аналогичный эффект дадут и команды "cat [abcd]*" и "cat [bdac]*".

7. Как выволняется подстановка результатов выполнения команд?

Shell-интерпретатор дает возможность выполнять **подстановку результатов** выполнения команд в Shell-программах. Если команда заключена в одиночные обратные кавычки, то интерпретатор Shell выполняет эту команду и подставляет вместо нее полученный результат.

8. Как интерпретировать строку cmd1 & cmd2 & ?

Команды cmd1 и cmd2 и последующие будут выполнены асинхронно, т.е. интерпретатор перейдет к обработке следующей команды без ожидания окончания трансляции.

9. Как интерпретировать строку cmd1 && cmd2 & ?

Команда cmd2 будет выполнена, если команда cmd1 завершилась успешно (вернула 0) и последующие будут выполнены асинхронно, т.е. интерпретатор перейдет к обработке следующей команды без ожидания окончания трансляции.

10. Как интерпретировать строку cmd1 || cmd2 & ?

Команда cmd2 будет выполнена, если команда cmd1 завершилась неуспешно (вернула 1) и последующие будут выполнены асинхронно, т.е. интерпретатор перейдет к обработке следующей команды без ожидания окончания трансляции.

11. В каком режиме выполняется интерпретатор команд Shell?

set [**-ekntuvx** [**параметр** ...]]

Устанавливает режимы работы интерпретатора языка **SHELL**. Могут задаваться следующие ключи:

-e В неинтерактивном режиме вызывает немедленное прерывание процесса при обнаружении ошибки в выполнении команды.

-k Все определенные макропеременные экспортируются всреду запускаемых процессов.

-n Производит только синтаксический контроль команд.

-t Прерывает выполнение процесса после того, как будет считана и выполнена одна команда.

-u Устанавливает режим диагностики ошибки при попытке использовать неопределенные макропеременные.

-v Устанавливает режим печати вводимых строк.

-x Распечатывает команды и их параметры по мере выполнения.

- Отменяет ключи **-v** и **-x**.

Установленные ключи содержатся в макропеременной **-**.

Остальным параметрам команды **set** присваиваются значения позиционных параметров "1, 2, ...". Если параметры не заданы, печатается список значений всех макропеременных.

12. Кем и в каком режиме осуществляется чтение потока символов с терминала интерпретатором Shell?

Чтение осуществляется в интерактивном? режиме (при обнаружении ошибки работы продолжается). Режим стандартным файлом stdin, имеющим номер 0.

Лабораторная работа №2

1. Что представляет собой суперблок?

Суперблок – содержит самую общую информацию о ФС (размер ФС, размер области индексных дескрипторов, их число, список свободных блоков, свободные индексные дескрипторы и т. д.). Суперблок всегда находится в оперативной памяти.

В суперблоке находятся:

1. Счетчик числа свободных блоков.
   Переменная s_nfree.
2. Список свободных блоков.
   Массив s_free[].
3. Счетчик числа описателей файлов.
   Переменная s_ninode.
4. Список описателей файлов.
   Массив s_inode[].

2. Что представляет собой список свободных блоков?

Когда процесс записывает данные в файл, ядро выделяет из ФС дисковые блоки под информационные блоки файла в прямой или косвенной адресации, связывая их с описателем файла в таблице индексных дескрипторов. Суперблок ФС содержит массив s_free, используя его для хранения номеров свободных дисковых блоков.

Программа ядра mkfs организует хранение не используемых в данный момент дисковых блоков в виде списковой структуры, первым элементом которой является массив s_free. Каждый элемент списка хранит номера свободных дисковых блоков аналогично s_free, причём самый первый номер в каждом элементе является ссылкой на следующий элемент списковой структуры.

Mkfs упорядочивает номера свободных дисковых блоков в списковой структуре с целью ускорения поиска блоков. Впоследствии, в связи с непредсказуемостью выделения/высвобождения блоков, mkfs списковых структур не перестраивает.

3. Что представляет собой список свободных описателей файлов?

Выделение свободных блоков для размещения файлов производится с конца списка супреблока. Когда в списке остается единственный элемент, ядро интерпретирует его как указатель на блок, содержащий продолжение списка. В этом случае содержимое этого блока считывается в суперблок и блок становится свободным. Такой подход позволяет использовать дисковое пространство под списки, пропорциональное свободному месту в файловой системе. Другими словами, когда свободного места практически не остается, список адресов свободных блоков целиком помещается в суперблоке.

Поскольку число свободных inode и блоков хранения данных может быть значительным, хранение двух последних списков целиком в суперблоке непрактично. Например, для индексных дескрипторов хранится только часть списка. Когда число свободных inode в этом списке приближается к 0, ядро просматривает ilist и вновь формирует список свободных inode. Для этого ядро анализирует поле di_mode индексного дескриптора, которое равно 0 у свободных inode.

4. Как производится выделение свободных блоков под файл?

Если массив s_inode[] в суперблоке не пуст, ядро выделяет очередной описатель файла и назначает его некоторому файлу.

Если массив пуст, ядро просматривает таблицу описателей файлов, выбирая из неё номера свободных описателей файлов, и заполняет ими массив s_inode[], причём запоминается та точка таблицы описателей файлов, на которой закончен просмотр с тем, чтобы при следующем заполнении массива s_inode[] начать с этой точки.

Алгоритм alloc:

1. если суперблок заблокирован, приостановиться до тех пор пока не будет снята блокировка
2. удалить блок из списка свободных блоков суперблока
3. если из списка удалён последний блок:
   - заблокировать суперблок
   - прочитать блок, только что взятый из списка свободных блоков
   - скопировать номера блоков, хранимых в этом блоке, в массив s_free суперблока
   - снять блокировку суперблока
4. переписать в буфер блок, удалённый из массива s_free
5. уменьшить общее число свободных блоков
6. пометить суперблок, как изменённый
7. возвратить буфер, содержащий выделенный блок.

5. Как производится освобождение блоков данных, занятых под файл?

1. If(список в суперблоке не полон) номер освобождаемого блока включается в список s_free
2. if(список полон) освобождённый блок включается в списковую структуру и ядро записывает в этот блок содержимое массива s_free
3. Массив s_free очищается и его елинственным элементом становится номер этого блока.

6. Каким образом осуществляется монтирование дисковых устройств?

Mount <устройство монтирования> <точка монтирования>

Для выполнения монтирования требуются права суперпользователя.

1. Если пользователь не root, возвратить ошибку.
2. Получить описатель файла для блочного специального файла.
3. Проверить допустимость значений параметров.
4. Получить описатель файла для каталога, где производится монтирование.
5. Если описатель файла не принадлежит каталогу или счётчик ссылок больше единицы:
   - освободить описатель файлов;

- возвратить ошибку.

6. Найти свободное место в таблице монтирования.
7. Запустить процедуру открытия блочного устройства для данного драйвера.
8. Получить свободный буфер из буферного кэша.
9. Считать суперблок в свободный буфер.
10. Проинициализировать поля суперблока.
11. Получить корневой описатель файла монтируемой системы, сохранив его в таблице монтирования.
12. Сделать пометку, что описатель файла каталога, являющегося вторым аргументом является точкой монтирования.
13. Снять блокировку с описателя файла каталога, который является точкой монтирования.

7. Каково назначение элементов структуры stat?

Структура **stat** содержит информацию о файле. Содержится в <sys/stat.h>. Возвращается функцией stat(char *name, stat *info). Описание:

```
struct stat
{
dev_t st_dev;    /* устройство, содержащее файл */
ino_t st_ino;    /* индекс */
ushort st_mode;  /* биты режима */
short st_nlink;  /* число связей файла */
ushort st_uid;   /* пользовательский ID */
ushort st_gid;   /* ID группы */
dev_t st_rdev;   /* для спец. файлов */
off_t st_size;   /* размер файла */
time_t st_atime; /* время последнего чтения */
time_t st_mtime; /* время последнего редактирования */
time_t st_ctime; /* время последнего изменения статуса */
}
```

Поле st_mode содержит флаги, описывающие файл. Флаги несут следующую информацию:

S_IFMT 0170000 - тип файла
S_IFDIR 0040000 - каталог
S_IFCHR 0020000 - байт-ориентированный специальный файл
S_IFBLK 0060000 - блок-ориентированный специальный файл
S_IFREG 0100000 - обычный файл
S_IFFIFO 0010000 - дисциплина FIFO
S_ISUID 04000 - идентификатор владельца
S_ISGID 02000 - идентификатор группы
S_ISVTX 01000 - сохранить свопируемый текст
S_ISREAD 00400 - владельцу разрешено чтение
S_IWRITE 00200 - владельцу разрешена запись
S_IEXEC 00100 - владельцу разрешено выполнение.

8. Каким образом осуществляется защита файлов в ОС UNIX?

Каждый файл или каталог имеет права доступа. Права доступа определяют, КТО и ЧТО может делать с содержимым файла. Существуют три группы прав доступа: для владельца файла, для членов группы, для всех остальных (см. табл.)

Право	Обозна-чение	Файл	Каталог
Чтение	r	Файл можно посмотреть и скопировать	Можно посмотреть список входящих файлов

Запись	w	Файл можно изменить и переименовать	Можно создавать и удалять файлы
Выпол-нение	x	Файл можно “выполнить”(скрип-ты и программы)	Можно входить, делать текущим

Примеры

-rw-r- -r- - , ... где (-) - тип файла (крайнее левое поле),
(r w -) - права доступа владельца файла,
(r - -) - права доступа группы владельца файла,
(r - -) - права доступа всех остальных.

В поле тип файла символ (-) обозначает файл, а символ (d) - каталог. В остальных полях символ (-) обозначает отсутствие прав доступа. В приведенном примере владелец имеет право читать и изменять файл, члены группы могут читать файл, все остальные могут только читать файл.

9. Каковы права доступа к файлу, при которых владелец может выполнять все операции (r, w, x), а прочие пользователи - только читать?

Права доступа могут быть заданы в команде не только в символьном виде, но и в цифровой форме (восьмеричное значение). Связь между цифровой и символьной формами приведена в табл. 2.2

Таблица 2.2

ЦИФРОВАЯ ФОРМА		**СИМВОЛЬНАЯ ФОРМА**
двоичная	**восьмеричная**	
111	**7**	**rwx**
110	**6**	**rw-**
101	**5**	**r-x**
100	**4**	**r--**
011	**3**	**-wx**
010	**2**	**-w-**
001	**1**	**- -x**
000	**0**	**- - -**

Для владельца(7)
Для прочих (4)
Итого: 744

10. Что выполняет системный вызов lseek(fd, (off_t)0, SEEK_END)?

Установит смещение файлового дескриптора fd, как размер файла//+(off_t)0.

int **fseek** (resource handle, int offset [, int whence])

Устанавливает смещение в файле, на который ссылается *handle*. Новое смещение, измеряемое в байтах от начала файла, получается путём прибавления параметра *offset* к позиции, указанной в параметре *whence*, значения которого определяются следующим образом:

SEEK_SET - Устанавливает смещение в *offset* байт.

SEEK_CUR - Устанавливает смещение в текущее плюс *offset*.

SEEK_END - Устанавливает смещение в размер файла плюс *offset*. (Чтобы перейти к смещению перед концом файла, вы должны передать отрицательное значение в параметр *offset*.)

Если *whence* не указан, по умолчанию он устанавливается в **SEEK_SET**.

В случае успеха возвращает 0; в противном сучае возвращает -1. Обратите внимание, что переход к смещению за концом файла не считается ошибкой.

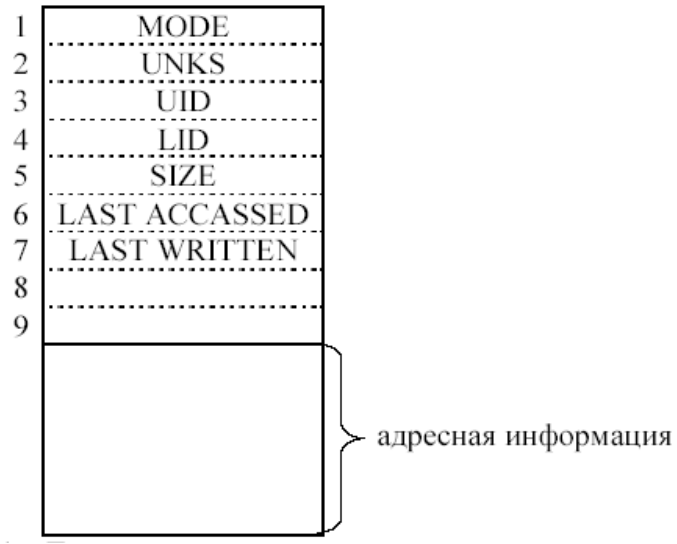
Лабораторная работа №3

1.Какова структура описателей файлов, таблицы файлов, таблицы открытых файлов процесса?

inode

Вся информация о файлах, кроме их содержимого и имени, находится в так называемых описателях файлов. Каждому файлу соответствует один описатель.

Описатель имеет фиксированный формат и располагается непрерывным массивом, начиная со второго блока. Общее число описателей (максимальное число файлов) задаётся в момент создания ФС. Описатели нумеруются натуральными числами. Первый описатель закреплён за «файлом» плохих блоков. Второй описывает корневой каталог ФС. Назначение прочих описателей не имеет фиксированного предназначения. Зная номер и размер описателя нетрудно вычислить его координаты на диске.



1. Тип и права доступа.
2. Число ссылок (счётчик числа ссылок на файл).
3. Идентификатор владельца.
4. Идентификатор группы, к которой принадлежит владелец.
5. Размер файла в байтах.
6. Время последнего доступа.
7. Время последней записи.
8. Время последней модификации inode.
9. Размер файла в блоках.

Адресная информация указывает на месторасположение содержимого файла на диске, состоит из прямой и косвенной адресной информации.

Информацию о файле необходимо получать с помощью системных вызовов stat (fstat), поскольку именно информация, хранящаяся в описателе файла, в основном и помещается системным вызовом stat (fstat) в стуктуру, специфицированную его вторым выходным параметром.

```
struct stat
{
dev_t st_dev;    /* устройство, содержащее файл */
ino_t st_ino;    /* индекс */
ushort st_mode;  /* биты режима */
short st_nlink;  /* число связей файла */
ushort st_uid;   /* пользовательский ID */
ushort st_gid;   /* ID группы */
dev_t st_rdev;   /* для спец. файлов */
off_t st_size;   /* размер файла */
time_t st_atime; /* время последнего чтения */
time_t st_mtime; /* время последнего редактирования */
time_t st_ctime; /* время последнего изменения статуса */
}
```

Таблица описателей файлов – структура, элементами которой являются копии описателей файлов: по одной на каждый файл, к которому осуществлена попытка доступа.

Каждый элемент таблицы файлов содержит информацию о режиме открытия файла, специфицированным при открытии файла, а также информацию о положении указателя чтения-записи.

Элемент таблицы открытых файлов процесса содержит номер дескриптора файла и ссылку на элемент таблицы файлов.

2.Какова цепочка соответствия дескриптора файла, открытого процессом, и файлом на диске?

Алгоритм creat:

1. Получить описатели файлов по данному имени файла (для указанного имени файла).
2. Если файл существует, и доступ к нему не разрешён, освободить описатель файла и возвратить код ошибки. Иначе считаем, что файл не существует:
   - назначить свободный описатель файла;
   - создать новую точку входа в родительский каталог;
   - включить имя нового файла в оглавление.
3. Выделить для описателя файла запись в таблице описателей файлов и инициализировать счётчик и смещение.
4. Выделить запись в таблице файлов в соответствии с режимом открытия, инициализировать счётчик и смещение, установить указатель на запись в таблице описателей файлов.
5. Выделить запись в таблице открытых файлов процесса, инициализировать счётчик и смещение, установить указатель на запись в таблице файлов.
6. Возвратить пользователю дескриптор файла.

3.Опишите функциональную структуру операции ввода-вывода (пулы, ассоциация их с драйверами, способы передачи информации и т.д.).

4. Каким образом осуществляется поддержка устройств ввода-вывода в ОС UNIX?

В ОС UNIX все устройства рассматриваются как некоторые виртуальные (специальные) файлы, что дает возможность использовать общий набор базовых операций ввода-вывода для любых устройств независимо от их специфики.

Драйверы устройств обеспечивают интерфейс между ядром UNIX и аппаратной частью компьютера. Благодаря этому от остальной части ядра скры!ты архитектурные особенности компьютера.

В UNIX существует большое количество драйверов. Часть из них обеспечивает доступ к физическим устройствам, например, жесткому диску, принтеру или терминалу, другие предоставляют услуги. Примером последних могут служить драйверы для работы с виртуальной памятью ядра представляющий "нулевое" устройство.

Драйвер устройства является частью кода ядра операционной системы и обеспечивает взаимодействие других подсистем UNIX с физическими или псевдоустройствами. Традиционно для встраивания драйвера в ядро UNIX требуется перекомпиляция ядра и перезапуск системы (в том числе момент загрузки)

Различают следующие типы драйверов:

1. Символьные - побайтовый обмен,
2. Блочные - обмен фиксированными порциями (блоками).
3. низкого уровня – битами, минуя буферный кэш.

Пул - это просто блок памяти выделенный в ядре. Размер пула зависит от типа драйвера.

Наиболее важной особенностью блокориентированного интерфейса является буферизация передаваемых блоков данных на основе поддерживаемого ядром *буферного пула (кэша)*, с емкостью от 20 до 40 буферов. Размер буферного пула задается при генерации OS UNIX. На время выполнения операций ввода-вывода буфера пула ассоциируются с драйверами внешних устройств по мере необходимости. Например, если выполняется операция ввода, то, прежде чем осуществить считывание очередного логического блока с внешнего устройства, производится проверка его наличия в буферном пуле. Реальное считывание происходит только при отрицательном результате этой проверки, для чего с драйвером ассоциируется неиспользованный буфер в пуле. Если в пуле нет свободного буфера, то занимается один из буферов, ассоциированных ранее с другим драйвером, который не используется в данный момент.

5.Какова структура таблиц открытых файлов, файлов и описателей файлов после открытия файла?

таблица описателей файлов +1
таблица файлов +1
таблица открытых файлов процесса +1 (для процесса, производящего открытие)

6.Какова структура таблиц открытых файлов, файлов и описателей файлов после закрытия файла?

таблица описателей файлов -1
таблица файлов -1
таблица открытых файлов процесса -1(для процесса, производящего закрытие)

7.Какова структура таблиц открытых файлов, файлов и описателей файлов после создания канала?

таблица описателей файлов +2
таблица файлов +2
таблица открытых файлов процесса +2 (для процесса, создавшего канал)

8.Какова структура таблиц открытых файлов, файлов и описателей файлов после создания нового процесса?

таблица описателей файлов =0
таблица файлов =0

таблица открытых файлов процесса =0

Лабораторная работа №4

1. Каким образом может быть порожден новый процесс? Какова структура нового процесса?

Системный вызов fork() – единственное средство порождения процессов.

Порождённый процесс является почти полной копией исходного процесса и отличается от него только номером (идентификатором) процесса. Созданный процесс наследует от родительского процесса контекст среды, включая дескрипторы файлов, каналы. Системный вызов fork возвращает в исходный процесс идентификатор исходного процесса, а в порождённый процесс – 0. В случае неуспешной попытки порождения процесса в родительский процесс возвращается -1.

2. Если процесс-предок открывает файл, а затем порождает процесс-потомок, а тот, в свою очередь, изменяет положение указателя чтения-записи файла, то изменится ли положение указателя чтения-записи файла процесса-отца?

Да, изменится.

С таблицей описателей файлов тесно связана таблица файлов. Каждый элемент таблицы файлов содержит информацию о режиме открытия файла, специфицированным при открытии файла, а также информацию о положении указателя чтения-записи. При каждом открытии файла в таблице файлов появляется новый элемент.

Один и тот же файл ОС UNIX может быть открыт несколькими не связанными друг с другом процессами, при этом ему будет соответствовать один элемент таблицы описателей файлов и столько элементов таблицы файлов, сколько раз этот файл был открыт.

Однако из этого правила есть одно исключение: оно касается случая, когда файл, открытый процессом, потом открывается процессом-потомком, порожденным с помощью системного вызова fork(). При возникновении такой ситуации операции открытия файла, осуществленной процессом-потомком, будет поставлен в соответствие тот из существующих элементов таблицы файлов (в том числе положение указателя чтения-записи), который в свое время был поставлен в соответствие операции открытия этого файла, осуществленной процессом-предком.

3. Что произойдет, если процесс-потомок завершится раньше, чем процесс-предок осуществит системный вызов wait()?

До момента вызова wait() ничего не произойдет.

В тот момент, когда процесс-отец получает информацию о причине смерти потомка, паспорт умершего процесса наконец вычеркивается из таблицы процессов и может быть переиспользован новым процессом. До того, он хранится в таблице процессов в состоянии "zombie" - "живой мертвец". Только для того, чтобы кто-нибудь мог узать статус его завершения.

4. Могут ли родственные процессы разделять общую память?

Могут. А почему бы и нет?...

5. Каков алгоритм системного вызова fork()?

1. Проверить доступность ресурсов ядра.
2. Получить свободное место в таблице процессов и уникальный идентификатор процесса.
3. Проверить, не запустил ли пользователь слишком много процессов (не превышено ли ограничение).
4. Сделать пометку, что порождённый процесс находится в состоянии создания.
5. Скопировать информацию в таблицу процессов из записи, соответствующей родительскому процессу, в запись, соответствующую порождаемому процессу.

6. Увеличить значение счётчика открытых файлов в таблице файлов.
7. Сделать копию контекста родительского процесса
8. Если в данный момент выполняется родительский процесс, то:
   - перевести порождаемый процесс в состояние готовности;
   - возвратить идентификатор процесса.
9. Если выполняется порождённый процесс, то:
   - записать начальные значения в поля синхронизации адресного пространства.

6.Какова структура таблиц открытых файлов, файлов и описателей файлов после создания процесса?

таблица описателей файлов =0
таблица файлов =0
таблица открытых файлов процесса =0

7.Каков алгоритм системного вызова exit()?

1. Игнорировать все сигналы.
2. Закрыть все открытые файлы.
3. Освободить области и память, ассоциированные с процессом.
4. Создать запись учётной информации.
5. Прекратить существование процесса.
6. Назначить всем процессам-потомкам в качестве родителя процесс init().
7. Если какой-либо из потомков прекратил существование, то послать процессу init сигнал гибели потомка.
8. Переключить контекст.

8.Каков алгоритм системного вызова wait()?

1. Если процесс, который вызвал wait, не имеет потомков, то возвратить код ошибки.
2. В бесконечном цикле:
   * Если процесс, вызвавший wait, имеет потомков, прекративших существование:
   - выбрать произвольного потомка;
   - передать его родителю информацию об использовании потомком ресурсов;
   - освободить в таблице процессов место, занятое процессом;
   - выдать идентификатор процесса, код возврата status из системного вызова exit, вызванного потомком.
   * Приостановиться с приоритетом, допускающим прерывание, до завершения потомка.

9.В чем разница между различными формами системных вызовов типа exec()?

Запуск программ, находящихся в отдельных файлах, в рамках текущего процесса возможен при использовании следующих системных вызовов:
int execl (char *path, char* arg1, ...);
Запуск программы path с параметрами arg1, arg2… Последний параметр должен иметь значение NULL.
int execv (char *path, char* argv[]);
Запуск программы path с параметрами argv[i]. Последний параметр должен иметь значение NULL.
int execlp (char *file, char *arg1, ...);
Запуск программы file с параметрами arg1, arg2… Последний параметр должен иметь значение NULL. Если file не содержит символов /, поиск программы ведётся, используя переменную окружения PATH.
int execvp (char *file, char* argv[]);
Запуск программы file с параметрами argv[i]. Последний параметр должен иметь значение NULL. Если file не содержит символов /, поиск программы ведётся, используя переменную окружения PATH.

10.Для чего используются сигналы в ОС UNIX?

Сигналы - это программное средство, с помощью которого может быть прервано функционирование процесса в ОС UNIX. Механизм сигналов позволяет процессам реагировать на различные события, которые могут произойти в ходе функционирования процесса внутри него самого или во внешнем мире. Каждому сигналу ставятся в соответствие номер сигнала и строковая константа, используемая для осмысленной идентификации сигнала.

11.Какие виды сигналов существуют в ОС UNIX?

Количество стандартных сигналов зависит от версии системы. Ниже приведен перечень, используемый в большинстве диалектов ОС.

SIGABRT	Сигнал генерируется вызовом функции *abort( )* и вызывает аварийное завершение процесса.
SIGALARM	Сигнал посылается по истечении времени, установленного функцией *alarm( )*.
SIGBUS	Сигнал возникает при сбое оборудования.
SIGCONT	Сигнал позволяет возобновить выполнение процесса, прерванного по сигналу **SIGSTOP**.
SIGFPE	Ошибка при выполнении арифметической операции с действительными числами.
SIGILL	Неверная инструкция процессора при выполнении программы.
SIGINT	Сигнал возникает при вводе с терминала <Ctrl+C>.
SIGIO	Завершение асинхронной операции ввода-вывода.
SIGIOT	Ошибочное завершение асинхронной операции ввода-вывода.
SIGKILL	Немедленно завершить процесс.
SIGPIPE	Ошибки при записи в канал межпроцессного ввода-вывода (pipe) (см. 1.7.2.).
SIGSEGV	Ошибка при использовании процессом неверного адреса.
SIGSTOP	Приостановить выполнение процесса.
SIGSYS	Ошибка при выполнении системного вызова.
SIGTERM	Завершить процесс.
SIGTRAP	Аппаратная ошибка при выполнении процесса.
SIGTSTP	Ввод с клавиатуры <Ctrl+Z> (приостановить процесс).

Программа, получив сигнал, может обработать его одним из трех способов:

1. Процесс может проигнорировать сигнал. Это невозможно только для SIGKIL и SIGSTOP. Эти два сигнала позволяют администратору системы UNIX прерывать или приостанавливать процессы в случае необходимости.

2. Процесс может зарегистрировать собственную функцию обработки сигнала. Рекомендуется, по возможности, обрабатывать сигналы, приводящие к преждевременному завершению программы.
3. Если не задана функция обработки сигнала, ядро системы выполняет действия, предусмотренные для его обработки по умолчанию. Если пришел сигнал, связанный с программными и аппаратными ошибками, процесс, как правило, завершается с созданием в текущей директории файла с именем "core". В последний помещается содержимое области оперативной памяти, которая была занята задачей в момент прихода сигнала.

12.Для чего используются каналы?

Для обмена данными между процессами.

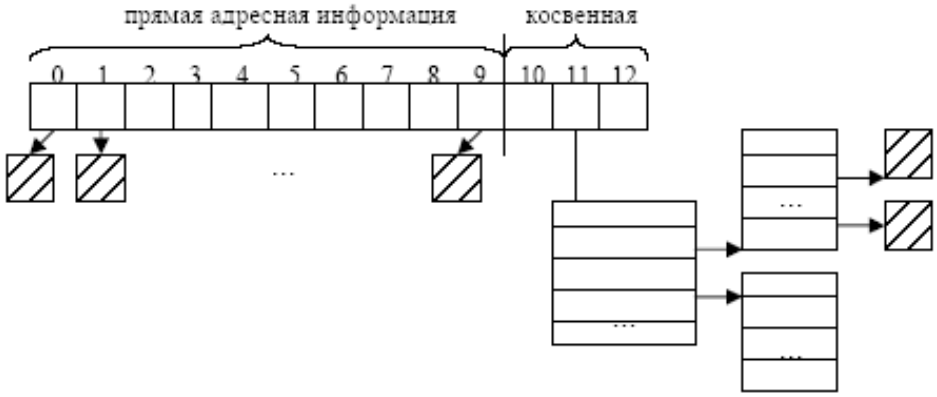
13.Какие требования предъявляются к процессам, чтобы они могли осуществлять обмен данными посредством каналов?

Процессы должны быть родственными.

14.Каков максимальный размер программного канала и почему?

Важно заметить, что на практике размер буфера канала конечен. Другими словами, только определенное число байтов может находиться в канале, прежде чем следующий вызов fdwrite будет заблокирован. Минимальный размер, определенный стандартом POSIX, равен 512 байтам. В большинстве существующих систем это значение намного больше.

При программировании важно знать максимальный размер канала для системы, так Как он влияет на оба вызова fdwrite и fdread. Он зависит от файловой системы.



10 – первого уровня

11 – второго уровня

12 – третьего уровня

На указатель отводится 4 байта.

(80 кб. + 2048)*8192 – максимальный размер файла.

Если блок 8 кб, то максимальная длина файла, использующего прямую адресацию, равна 80 кб.

Косвенная адресация первого уровня ссылается на блок. На указатель отводится 4 байта. Всего в блоке может быть 2048 ссылок. Косвенная адресация второго уровня: блок содержит 2048 ссылок на блоки, имеющие ссылки.

Лабораторная работа №7

1.В чем разница между двоичным и общим семафорами?

Есть два типа семафоров: двоичные и общие (или счётные).

Как видно из названия, двоичный семафор может принимать только два значения - 0 или 1.

Счётный семафор может принимать значения в диапазоне от 0 до 255, 65535 или 4294967295, в зависимости от того, семафор какой разрядности используется - 8, 16 или 32 бит соответственно. Это значение зависит от того, какое ядро используется.

UNIX поддерживает общие семафоры (как расширение двоичных семафоров).

Достаточность в общем случае двоичных семафоров доказана. Известен алгоритм реализации семафоров общего вида на основе двоичных.

2.Чем отличаются P() и V()-операции от обычных операций увеличения и уменьшения на единицу?

V - операция (V(S))

- это операция с одним аргументом, который должен быть семафором S. Эта операция увеличивает значение аргумента на 1.

P - операция (P(S))

- это операция с одним аргументом, который должен быть семафором S. Ее назначение - уменьшить величину аргумента на 1, если только результирующее значение не становится отрицательным.

P- и V- операции являются неделимыми, т.е. решение о том, что настоящий момент является подходящим для выполнения операциия и последующее собственно выполнение операции, должно рассматриваться как неделимая операция.

3.Для чего служит набор программных средств IPC?

IPC (Inter-Process Communication Facilities) обеспечивает:

- синхронизацию процессов при доступе к совместно используемым ресурсам (семафоры - semaphores);
- посылку процессом сообщений другому произвольному процессу (очереди сообщений - message queries);
- наличие общей для процессов памяти (сегменты разделяемой памяти - shared memory segments).

4.Для чего введены массовые операции над семафорами в ОС UNIX?

Основным системным вызовом для манипулирования семафором является semop:

oldval = semop(id, oplist, count);

id - это ранее полученный дескриптор группы семафоров,

oplist - массив описателей операций над семафорами группы,

Каждый элемент массива oplist имеет следующую структуру:

- номер семафора в указанном наборе семафоров;
- операция;
- флаги.

count - размер этого массива.

Значение, возвращаемое системным вызовом, является значением последнего обработанного семафора.

Основным поводом для введения массовых операций над семафорами было стремление дать программистам возможность избегать тупиковых ситуаций в связи с семафорной синхронизацией. Это обеспечивается тем, что системный вызов semop, каким бы длинным он не был (по причине потенциально неограниченной длины массива oplist) выполняется как атомарная операция, т.е. во время выполнения semop ни один другой процесс не может изменить значение какого-либо семафора.

5.Какие операции над семафорами существуют в ОС UNIX?

Набор операций над семафорами System V IPC отличается от классического набора операций {P, V}, предложенного Дейкстрой. Он включает три операции:

- A(S, n) – увеличить значение семафора S на величину n;
  - D(S, n) – пока значение семафора S < n, процесс блокируется. Далее S = S - n;
  - Z(S) – процесс блокируется до тех пор, пока значение семафора S не станет равным 0.

Изначально все IPC-семафоры инициируются нулевым значением.
Мы видим, что классической операции P(S) соответствует операция D(S,1), а классической операции V(S) соответствует операция A(S,1). Аналогом ненулевой инициализации семафоров Дейкстры значением n может служить выполнение операции A(S,n) сразу после создания семафора S, с обеспечением атомарности создания семафора и ее выполнения посредством другого семафора.
Мы показали, что классические семафоры реализуются через семафоры System V IPC. Обратное не является верным. Используя операции P(S) и V(S), мы не сумеем реализовать операцию Z(S).

6.Каково назначение механизма очередей сообщений?

Обмен данными между процессами посредством сообщений.

7.Каково назначение системного вызова msgget()?

По системному вызову msgget() в ответ на ключ (key) и набор флагов (полностью аналогичны флагам в системном вызове semget()) ядро либо создает новую очередь сообщений и возвращает пользователю идентификатор созданной очереди, либо находит элемент таблицы очередей сообщений, содержащий указанный ключ, и возвращает соответствующий идентификатор очереди:

int msgqid = msgget(key_t key, int flag).
Прототип функции описан в файлах
#include <sys/ipc.h>
#include <sys/msg.h>

8.Какие условия должны быть выполнены для успешной постановки сообщения в очередь?

Условиями успешной постановки сообщения в очередь являются:

- наличие прав процесса по записи в данную очередь сообщений;
- не превышение длиной сообщения заданного системой верхнего предела;
- положительное значение указанного в сообщении типа сообщения.

Если же оказывается, что новое сообщение невозможно буферизовать в ядре по причине превышения верхнего предела суммарной длины сообщений, находящихся в данной очереди сообщений (флаг IPC_NOWAIT при этом отсутствует), то обратившийся процесс откладывается (усыпляется) до тех пор, пока очередь сообщений не разгрузится процессами, ожидающими получения сообщений.

9.Как получить информацию о владельце и правах доступа очереди сообщений?

Системный вызов msgctl()

int msgctl (int msgqid, int command, struct msqid_ds *msg_stat)
используется с command = IPC_STAT для опроса состояния описателя очереди сообщений () и помещения его в структуру msg_stat, которая содержит в себе помимо прочего информацию о владельце и правах доступа очереди сообщений.

msg_perm.uid uid владельца очереди.

msg_perm.gid	gid владельца очереди.
msg_perm.mode	режим доступа к файлу в очереди.
msg_stime	время отправки последнего сообщения в очередь.
msg_rtime	время получения последнего сообщения из очереди.
msg_ctime	время последнего изменения очереди.
msg_qnum	количество сообщений в очереди, ожидающих прочтения.
msg_qbytes	количество байт, пространства, доступного в очереди в данный момент для хранения отправленных сообщений, пока они не будут получены.
msg_lspid	pid процесса, отправившего последнее сообщение в очередь.
msg_lrpid	pid процесса, отправившего последнее сообщение из очереди.

10.Каково назначение системного вызова shmget()?
Системный вызов

int shmid = shmget (key_t key, size_t size, int flag)
на основании параметра size определяет желаемый размер сегмента в байтах. Если в таблице разделяемой памяти находится элемент, содержащий заданный ключ, и права доступа не противоречат текущим характеристикам обращающегося процесса, то значением системного вызова является идентификатор существующего сегмента. В противном случае создается новый сегмент с размером не меньше установленного в системе минимального размера сегмента разделяемой памяти и не больше установленного максимального размера. Создание сегмента не означает немедленного выделения под него основной памяти и это действие откладывается до выполнения первого системного вызова подключения сегмента к виртуальной памяти некоторого процесса.

IPC_CREAT	Semget создает новый сегмент для данного ключа. Если флаг IPC_CREAT не задан, а сегмент с указанным ключом уже существует, то обращающийся процесс получит идентификатор существующего сегмента.
IPC_EXLC	Флаг IPC_EXLC вместе с флагом IPC_CREAT предназначен для создания (и только для создания) сегмента. Если сегмент уже существует, Semget возвратит -1, а системная переменная errno будет содержать значение EEXIST.

Лабораторная работа №8

1.Какова структура IP-адреса?
IP-адрес имеет длину 4 байта и обычно записывается в виде четырех чисел, представляющих значения каждого байта в десятичной форме, и разделенных точками, например:

128.10.2.30 - традиционная десятичная форма представления адреса.

IP-адрес состоит из двух логических частей: номера сети и номера узла. Какая часть адреса относится к номеру сети, а какая – к номеру узла, определяется значениями первых трёх битов адреса.
Всего существует 5 классов IP-адресов: A,B,C,D,E, но основными являются первые три.
Класс A:
Первый бит – 0, далее – номер сети, номер узла в сети.

Если адрес начинается с двоичного нуля, то сеть относится к классу A, номер сети занимает 1 байт, остальные 3 байта интерпретируются как номер узла в сети. Сети класса A имеют номера в диапазоне 1÷126 (0 не используется), номер 127 зарезервирован. Максимальное число узлов в сети класса A не может превышать величины 16777216 (2^{24}).

Класс B.

Первое двоичное число – 1, второе – 0. Первые 2 байта определяют номер сети, следующие 2 – номер узла в сети.

Диапазон адресов в сети класса B: 128.0.0.0-191.255.255.255. Максимальное количество узлов в сети класса B не может превышать 65536 (2^{16}).

Класс C:

Если адрес начинается с последовательности 110, то это сеть класса C. Под адрес сети отводится 3 байта, под номер компьютера – 1 байт. Сеть класса C имеет номера в диапазоне 192.0.0.0-223.255.255.255. Количество узлов в сети класса C не превышает 256 (2^6).

Класс D:

Первые 4 цифры 1110. Деления на номер сети и номер узла нет. Адрес является адресом группы Multicast, обозначает особый групповой адрес. Если в пакете в качестве адреса назначения указать адрес класса D, то такой пакет должны получить все узлы, которым присвоен данный адрес. Диапазон адресов: 224.0.0.0-239.255.225.255.

Класс E: В начале стоит комбинация 11110; зарезервирован (в настоящий момент не используется). Диапазон: 240.0.0.0-247.255.255.255.

В таблице приведены диапазоны номеров сетей, соответствующих каждому классу сетей.

Класс	**Наименьший адрес**	**Наибольший адрес**
A	01.0.0	126.0.0.0
B	128.0.0.0	191.255.0.0
C	192.0.1.0	223.255.255.0
D	224.0.0.0	239.255.255.255
E	240.0.0.0	247.255.255.255

2.Как поместить и извлечь IP-адрес из структуры сокета?

Сокеты для связи через сеть, определеннные в файле <netinet/in.h>, содержат стркуктуру struct in_addr:

```
struct sockaddr_in
{
u_char sin_len;         /*Длина поля sockaddr_in
                    (для FBSD)*/
u_char sin_family;      /*Семейство адресов (домен)*/
u_short sin_port;       /*Номер порта*/
struct in_addr sin_addr; /*IP-адрес*/
```

```
char sin_zero[8]; };     /*Поле выравнивания*/
где
struct in_addr
{
n_int32_t s_addr
};
```

При программировании TCP-соединения должны быть созданы сокеты (системный вызов socket()) и в программе сервера и в программе клиента, при этом в обеих программах сокеты связываются с адресом машины, на которую будет установлена программа сервера. Но, если в программе сервера для определения IP-адреса в структуре сокета может быть использована переменная INADDR_ANY, то в программе клиента для занесения в структуру сокета IP-адреса машины сервера необходимо использовать системный вызов inet_addr().

Сетевые вызовы inet_addr() и inet_ntoa() выполняют преобразования IP-адреса из формата текстовой строки "x.y.z.t" в структуру типа in_addr и обратно.

```
#include <arpa/inet.h>
in_addr_t inet_addr (const char *ip_address);
char * inet_ntoa(const struct in_addr in);
```

Для того чтобы процесс мог ссылаться на адрес своего компьютера, файле <netinet/in.h> определена переменная INADDR_ANY, содержащая локальный адрес компьютера в формате in_addr_t.

3.В чем разница между моделями TCP-соединения и дейтаграмм ?

User Datagarm Protocol, UDP — протокол стека протокола TCP/IP, в котором для адресации используется протокол IP, но средства контроля доставки пакетов протокола TCP отброшены.

Transmission Control Protocol, TCP протокол управления передачей данных) — протокол передачи пакетов данных по сети с гарантированной доставкой.

User Datagram Protocol (дословный перевод - протокол дайтограмм пользователя) предназначен для передачи данных между прикладными процессами и обменом дейтаграммами между компьютерами входящими в единую сеть.

Длина пакета в UDP измеряется в октетах, дейтаграммы пользователя включают заголовок и данные. Это означает, что минимальная величина длины четыре байта.

Протокол UDP является транспортным и он не устанавливает логического соединения, а также не упорядочивает пакеты данных. То есть пакеты могут прийти не в том порядке в котором они были отправлены и UDP не обеспечивает достоверность доставки пакетов. Но данные, отправляемые через модуль UDP, достигают места назначения как единое целое. Главная особенность UDP заключается в том, что он сохраняет границы сообщений и никогда не объединяет несколько сообщений в одно.

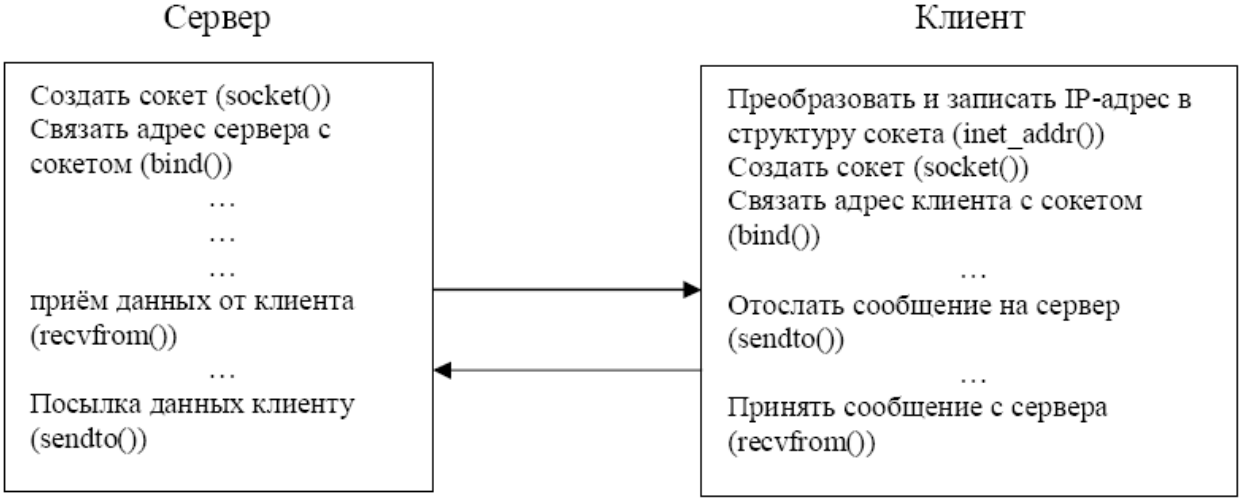
Взаимодействие между процессами и модулем UDP осуществляется через UDP-порты. Адресом назначения является номер порта прикладного сервиса. В протоколе так же используется IP который является адресом узла . У UDP так же как и у TCP существуют зарезервированные порты. Присвоением сервисам собственных номеров занимается организация IANA (Internet Assigned Numbers Authority). Всего в UDP используется от 0 до 65535 портов. При этом от 0 до 1023 главные порты, от 1024 до 49151 порты выделенные под крупные проекты и частные порты, от 49152 до 65535 предусмотрены для любого программиста который захочет использовать данный протокол.

4.Каковы основные шаги межпроцессоного взаимодействия в модели TCP-соединения?

Структура серверного процесса:
создать сокет (socket())
связать адрес сервера с сокетом (bind())
включить приём TCP-соединений (listen())
установить соединение с клиентом (accept())
организовать приём данных от клиента (recv())
преобразовать поступившие данные
отослать ответ клиенту (send())
закрыть TCP-соединение (close())

Структура клиентского процесса:
преобразовать и записать IP-адрес в структуру сокета (inet_addr())
создать сокет (socket())
подключиться к серверу, связав сокет с адресом сервера (connect())
послать данные (send())
принять двнные (recv())
<закрыть TCP-соединение (close())

5.Каковы основные шаги межпроцессоного взаимодействия в модели дейтаграмм?



6.Как занести в структуру сокета IP-адрес своего компьютера?

Для того чтобы процесс мог ссылаться на адрес своего компьютера, файле <netinet/in.h> определена переменная INADDR_ANY, содержащая локальный адрес компьютера в формате in_addr_t.

7.Каким образом извлечь информацию о клиенте после установки TCP-соединения?

Дополнительный сокет для работы с соединением создается при помощи вызова accept(), принимающего очередное соединение.

```
#include<sys/types.h>
#include<sys/socket.h>
int accept (int sfd, struct sockaddr *addr, size_t *add_l);
```

- sfd - дескриптор сокета, для которого ведется прием соединений;
- addr - указатель на обобщенную структуру адреса сокета с информацией о клиенте; так как связь использует соединение адрес клиента знать не обязательно и допустимо задавать параметр address значением NULL;
- add_l - размер структуры адреса, заданной параметром address, если значение address не равно NULL.

Возвращаемое значение соответствует идентификатору нового сокета, который будет использоваться для связи. До тех пор, пока от клиента не поступил запрос на соединение, процесс, выдавший системный вызов **accept()** переводится в состояние ожидания.

8.Какова реакция системных вызовов посылки и приема сообщений в модели TCP-соединения при разрыве связи?

```
#include<sys/types.h>
#include<sys/socket.h>
ssize_t recv (int skfd, void *buff, size_t l, int fl);
ssize_t send (int sfd, const void *buff, size_t l, int fl);
```

- sfd - дескриптор сокета, через который читаются или записываются данные;
- buff - буфер, в который они помещаются или откуда отсылаются через сокет;
- l - размер буфера;
- fl - поле дополнительных опций при получении или передаче данных.

В случае успешного чтения/записи системные вызовы send() и recv() возвращают число прочитанных/отосланных байт, или -1 в случае ошибки; в случае разорванной связи (клиент разорвал TCP-соединение) вызов recv() (или read()) возвращают нулевое значение; если процесс пытается записать данные через разорванное TCP-соединение посредством write() или send(), то он получает сигнал SIGPIPE, который можно обработать, если предусмотрена обработка данного сигнала.