15. Асинхронный ввод/вывод в POSIX и уведомления о завершении операций.

Лекция 19

Интерфейс POSIX AIO состоит из следующих функций:

int aio\_read(struct aiocb \*aiocbp);

Функция aio\_read() ставит в очередь запрос ввода-вывода, описанный в буфере, на который указывает aiocbp. Эта функция является асинхронным аналогом вызова read(2).

int aio\_write(struct aiocb \*aiocbp);

Функция aio\_write() ставит в очередь запрос ввода-вывода, описанный в буфере, на который указывает aiocbp. Эта функция является асинхронным аналогом вызова write(2).

int lio\_listio( int mode,

struct aiocb \*const aiocb\_list[],

int nitems,

struct sigevent \*sevp);

Функция lio\_listio() запускает на выполнение список операций ввода-вывода, описанных

в массиве aiocb\_list.

int aio\_error(const struct aiocb \*aiocbp);

Функция aio\_error() возвращает состояние ошибки запроса асинхронного ввода-вывода

для указанного блока управления aiocbp.

ssize\_t aio\_return(struct aiocb \*aiocbp);

Функция aio\_return() возвращает окончательное значение завершения запроса асин-

хронного ввода-вывода, задаваемого указателем на контрольный блок aiocbp.

Структура sigevent используется в различных программных интерфейсах для описания

способа, которым нужно уведомлять процесс о событии (например, окончание асинхронного

запроса, истечение таймера или поступление сообщения).

struct sigevent {

int sigev\_notify; // метод уведомления

int sigev\_signo; // сигнал уведомления

union sigval sigev\_value; // данные, передаваемые с уведомлением

// функция, используемая для нити уведомления (SIGEV\_THREAD)

void (\*sigev\_notify\_function)(union sigval);

void \*sigev\_notify\_attributes; // атрибуты для нити уведомления (SIGEV\_THREAD)

pid\_t sigev\_notify\_thread\_id; // ID нити для уведомления (SIGEV\_THREAD\_ID)

};

Уведомления приходят по окончании ввода/вывода и являются сигнализацией окончания.

16. Основной интерфейс асинхронного ввода/вывода в POSIX и списки запросов

Лекция 19 21 и дальше.

Интерфейс POSIX AIO состоит из следующих функций:

int aio\_read(struct aiocb \*aiocbp);

Функция aio\_read() ставит в очередь запрос ввода-вывода, описанный в буфере, на который указывает aiocbp. Эта функция является асинхронным аналогом вызова read(2).

int aio\_write(struct aiocb \*aiocbp);

Функция aio\_write() ставит в очередь запрос ввода-вывода, описанный в буфере, на который указывает aiocbp. Эта функция является асинхронным аналогом вызова write(2).

int lio\_listio( int mode,

struct aiocb \*const aiocb\_list[],

int nitems,

struct sigevent \*sevp);

Функция lio\_listio() запускает на выполнение список операций ввода-вывода, описанных

в массиве aiocb\_list.

Значение операции mode может быть одним из следующих:

LIO\_WAIT — вызов не завершается до тех пор, пока не будут выполнены все операции. Аргумент sevp при этом игнорируется.

LIO\_NOWAIT — операции ввода-вывода ставятся в очередь на обработку и вызов сразу завершается. После выполнения всех операций ввода-вывода посылается асинхронное уведомление, задаваемое в аргументе sevp. Если значение sevp равно NULL, то асинхронные уведомления не посылаются.

Аргумент aiocb\_list представляет собой массив указателей на структуры aiocb, в которых

описаны операции ввода-вывода.

Эти операции выполняются в произвольном порядке

int aio\_error(const struct aiocb \*aiocbp);

Функция aio\_error() возвращает состояние ошибки запроса асинхронного ввода-вывода

для указанного блока управления aiocbp.

ssize\_t aio\_return(struct aiocb \*aiocbp);

Функция aio\_return() возвращает окончательное значение завершения запроса асин-

хронного ввода-вывода, задаваемого указателем на контрольный блок aiocbp.

17. Сокеты — основные концепции, типы и адреса.

Лекция 20 3-дальше

Когда создается сокет, необходимо указать тип связи, который предполагается использовать, и тип протокола, который должен ее реализовать.

Тип связи

«Тип (стиль) связи» сокета определяет семантику отправки и получения данных на

уровне пользователя через сокет. Выбор типа связи определяет ответы на следующие вопросы:

- Каковы единицы передачи данных?

- Могут ли данные быть потеряны во время нормальной работы?

Тип протокола

Для осуществления связи необходимо выбрать «протокол».

Протокол определяет, какой низкоуровневый механизм используется для передачи и

приема данных. Нужно знать о протоколах следующее:

1) Для связи между двумя сокетами они должны указать один и тот же протокол.

2) Каждый протокол имеет смысл с определенными комбинациями стиля/пространства

имен и не может использоваться с несоответствующими комбинациями. Например, протокол TCP соответствует только стилю связи с потоком байтов и пространству имен Internet.

3) Для каждой комбинации стиля и пространства имен существует «протокол по умолчанию», который можно запросить, указав 0 в качестве номера протокола. И это то, что обычно следует делать — использовать в качестве номера протокола значение по умолчанию.

SOCK\_STREAM

Тип SOCK\_STREAM похож на канал (pipes и FIFO). Он работает через соединение с кон-

кретным удаленным сокетом и надежно передает данные в виде потока байтов.

SOCK\_DGRAM

Тип SOCK\_DGRAM используется для ненадежной отправки пакетов с индивидуальной адресацией. Это полная противоположность SOCK\_STREAM.

SOCK\_RAW

Этот тип обеспечивает доступ к сетевым протоколам и интерфейсам низкого уровня.

Обычные пользовательские программы обычно не нуждаются в использовании этого стиля.

SOCK\_SEQPACKET

Обеспечивает работу последовательного двустороннего канала для передачи дейтаграмм

на основе соединений. Дейтаграммы имеют постоянный размер. От получателя требуется

за один раз прочитать целый пакет.

SOCK\_RDM

Обеспечивает надежную доставку дейтаграмм без гарантии, что они будут расположены

по порядку.

Имя сокета обычно называется «адресом».

Функции и символы для работы с адресами сокетов были названы непоследовательно,

иногда с использованием термина «имя», а иногда с использованием термин «адрес». Когда

речь идет о сокетах, можно рассматривать эти термины как синонимы.

18. Пять основных системных вызовов для работы с сокетами и их использование.

Лекция 20 28- дальше

int socket( int socket\_family, // семейство протоколов, которое будет использоваться

int socket\_type, // семантика соединения (stream, datagram, raw, ...)

int protocol); // протокол, используемый с сокетом

int bind( int sockfd, // файловый дескриптор сокета

const struct sockaddr \*addr, // адрес сокета

socklen\_t addrlen); // длина адреса

int listen( int sockfd, // файловый дескриптор сокета

int backlog); // максимальный размер очереди ожидающих соединений

int accept( int sockfd, // файловый дескриптор слушающего сокета

struct sockaddr \*addr, // адресс сокета или NULL

socklen\_t \*addrlen); // длина адреса

или же

int connect( int sockfd, // сокет инициатора соединения

const struct sockaddr \*addr, // адрес соединения

socklen\_t addrlen); // размер addr

send()/recv()

На стороне сервера:

1) Вызывается socket для создания объекта и связанного с ним файлового дескриптора.

2) Вызывается bind, для назначения сокету адреса (имени) .

3) Вызывается listen, чтобы пометить сокет, как предназначенный для приема запросов

на соединение.

4) Вызывается accept, чтобы дождаться и принять входящее соединение. После этого

вызов accept создает второй сокет с новым файловым дескриптором, который и использу-

ется для обмена данными.

5) В операциях ввода-вывода (системные вызовы read и write) участвует созданный

(второй) файловый дескриптор.

На стороне клиента:

1) Вызывается socket для создания объекта и связанного с ним файлового дескриптора.

2) Для установления соединения с сервером вызывается connect, которому в качестве

аргумента передается имя сервера, (этот шаг, не обязательно должен быть синхронизирован

со вторым шагом на стороне сервера).

3) Файловый дескриптор сокета используется для выполнения операций ввода-вывода.

19. Ввод/вывод с сокетами, функции отправки и получения сообщений

Лекция 20 21-дальше

Ввод/вывод осуществляется путём отправки и получения сообщений. Сокет отправляет сообщение, другой же сокет изначально ждёт принятия сообщения вызовом recv().

ожидает готовности операций ввода-вывода

Вызов select() позволяет программам отслеживать изменения нескольких файловых

дескрипторов ожидая, когда один или более файловых дескрипторов станут «готовы» для операции ввода-вывода определённого типа (например, ввода).

int select( int nfds, // количество всех файловых дескрипторов

fd\_set \*readfds, // набор дескрипторов для чтения

fd\_set \*writefds, // набор дескрипторов для записи

fd\_set \*exceptfds, // набор дескрипторов для обнаружения

// исключительных условий.

struct timeval \*timeout); // время ожидания

ssize\_t send( int sockfd, // отправляющий сокет

const void \*buf, // буфер с данными

size\_t len, // длина данных

int flags); //

Системные вызовы send(), sendto() и sendmsg() используются для пересылки сообщений в

другой сокет. Вызов send() можно использовать, только если сокет находится в состоянии соединения, то есть если известен получатель.

ssize\_t recv( int sockfd, // получающий сокет

void \*buf, // буфер с данными

size\_t len, // длина данных

int flags);

Системные вызовы recv(), recvfrom() и recvmsg() используются для получения сообщений из сокета. Они могут использоваться для получения данных, независимо от того, является ли сокет ориентированным на соединения или нет.

Вызов recv() отличается от read(2) только наличием аргумента flags. Если значение flags

равно нулю, то вызов recv() эквивалентен read(2) с некоторыми отличиями.

20. Сокеты для локального межпроцессного взаимодействия AF\_LOCAL, три типа адреса.

Лекция 20

AF\_UNIX — сокеты для локального межпроцессного взаимодействия

unix\_socket = socket(PF\_UNIX, type, 0);

error = socketpair(PF\_UNIX, type, 0, int \*sv);

Семейство сокетов AF\_UNIX (AF\_LOCAL) используется для эффективного взаимодействия между процессами на одной машине.

Допустимые типы сокета для домена UNIX:

SOCK\_STREAM — потоковый сокет;

SOCK\_DGRAM — датаграмный сокет, сохраняющий границы сообщений в большинстве

реализаций UNIX. Датаграмные сокеты домена UNIX всегда надёжны и не меняют порядок датаграмм.

SOCK\_SEQPACKET — сокет, ориентированный на соединение. Задает последовательность пакетов, сохраняет границы сообщений и доставляет сообщения в том же порядке, в каком они были отправлены.

Адрес доменного сокета UNIX представляет собой следующую структуру:

struct sockaddr\_un {

sa\_family\_t sun\_family; // AF\_UNIX

char sun\_path[108]; // имя пути

};

В sockaddr\_un структуре различают три типа адресов:

1) с именем пути (путевые сокеты) — доменный сокет UNIX может быть привязан к имени

пути (с завершающимся null) в файловой системе с помощью bind(2). При возврате адреса имени пути сокета (одним и системных вызовов, упомянутых выше), его длина равна offsetof(struct sockaddr\_un, sun\_path)4 + strlen(sun\_path) + 1 и sun\_path содержит путь, оканчивающийся нулем.

В Linux, указанное выше выражение offsetof() равно sizeof(sa\_family\_t), но в некоторых

реализациях включаются другие поля перед sun\_path, поэтому выражение offsetof() описывает размер адресной структуры более переносимым способом).

4) offsetof(type, member)2) безымянный — потоковый сокет, который не привязан к имени пути с помощью bind(), не

имеет имени.

Два сокета, создаваемые socketpair(), также не имеют имён. При возврате адреса сокета его

длина равна sizeof(sa\_family\_t), а значение sun\_path не используется.

3) абстрактный: абстрактный адрес сокета отличается (от имени пути сокета) тем, что значением

sun\_path[0] является байт \0.

21. Безымянные сокеты AF\_LOCAL и функция socketpair(), путевые сокеты.

Лекция 20 21 и дальше

Безымянный сокет — потоковый сокет, который не привязан к имени пути с помощью bind(), не имеет имени.

Два сокета, создаваемые socketpair(), также не имеют имён. При возврате адреса сокета его

длина равна sizeof(sa\_family\_t), а значение sun\_path не используется.

int socketpair(int domain, int type, int protocol, int sv[2]);

Вызов socketpair() создает пару неименованых присоединённых сокетов в заданном семействе адресации (домене) domain заданного типа взаимодействия type, используя (при необходимости) заданный протокол protocol.

Данные аргументы имеют тот же смысл и значения, что и для системного вызова socket(2).

Файловые дескрипторы, используемые как ссылки на новые сокеты, возвращаются в sv[0] и

sv[1]. Никаких различий между этими двумя сокетами нет.

путевые сокеты — доменный сокет UNIX может быть привязан к имени

пути (с завершающимся null) в файловой системе с помощью bind(2).

При привязке сокета к пути для максимальной переносимости и простоте кодирования нужно

учесть несколько правил:

- Имя пути в sun\_path должно завершаться null.

- Длина имени пути, включая завершающий байт null, не должна превышать размер sun\_path.

- Аргумент addrlen, описывающий включаемую структуру sockaddr\_un, должен содержать значение, как минимум: offsetof(struct sockaddr\_un, sun\_path) + strlen(addr.sun\_path) + 1 или, проще говоря, для addrlen можно использовать sizeof(struct sockaddr\_un).

22. Вспомогательные сообщения, их передача и прием с использованием функций \*msg().

Вспомогательные сообщения

Вспомогательные данные отправляются и принимаются с помощью sendmsg(2) и

recvmsg(2). В силу исторических причин перечисленные типы вспомогательных сообщений относятся к типу SOL\_SOCKET, даже если они относятся к AF\_UNIX.

Для того, чтобы их отправить, следует установить значение поля cmsg\_level структуры

cmsghdr равным SOL\_SOCKET, а в значении поля cmsg\_type указать его тип.

Дополнительная информация приведена в cmsg(3).

SCM\_RIGHTS

Передать или принять набор открытых файловых дескрипторов из другого процесса. Часть с данными содержит целочисленный массив файловых дескрипторов.

Эта операция эквивалентна дублированию (dup(2)) файлового дескриптора в таблицу файловых дескрипторов другого процесса.

SCM\_CREDENTIALS

Передать или принять учётные данные UNIX. Может быть использована для аутентификации. Учётные данные передаются в виде структуры struct ucred вспомогательного сообщения.

SCM\_SECURITY

Получить контекст безопасности SELinux (метку безопасности) однорангового сокета. Полученные вспомогательные данные представляют собой строку (с null в конце) с контекстом безопасности.

Получатель должен выделить не менее NAME\_MAX байт под эти данные в в части данных вспомогательного сообщения.

При отправке вспомогательных данных с помощью sendmsg(2) посылаемое сообщение может

содержать только по одному элементу каждого типа, из представленных выше. При отправке вспомогательных данных по крайней мере должен быть отправлен один байт реальных данных.

При отправке вспомогательных данных через дейтаграммный доменный сокет UNIX в Linux

необязательно отправлять какие-либо реальные сопровождающие данные. Однако переносимые приложения должны также включать, по крайней мере, один байт реальных данных при отправке вспомогательных данных через дейтаграммный сокет.

При получении из потокового сокета вспомогательные данные формируют своего рода барьер для полученных данных.

23. Права на каталоги и файлы, функции umask(), chown() и chmod().

С божьей помощью

mode\_t umask(mode\_t mask);

Системный вызов umask() устанавливает в вызывающем процессе значение маски (umask) режима доступа к файлу равным mask & 0777 (т.е. из mask используются только биты прав доступа к файлу) и возвращает предыдущее значение маски.

Значение umask используется в open(2), mkdir(2) и других системных вызовах, которые создают файлы, для изменения прав, назначаемых на создаваемые файлы или каталоги.

int chown(const char \*pathname, uid\_t owner, gid\_t group);

chown() изменяет владельца для файла, задаваемого параметром pathname, который разыменовывается, если является символьной ссылкой.

Только привилегированный процесс может сменить владельца файла.

Владелец файла может сменить группу файла на любую группу, в которой он числится.

Привилегированный процесс может задавать произвольную группу.

int chmod(const char \*pathname, mode\_t mode);

Вызов chmod() изменяет режим файла, задаваемого путём из параметра pathname, который разыменовывается, если является символьной ссылкой.

Новый режим файла указывается в mode и представляет собой битовую маску, создаваемую побитовым сложением нуля или более следующих констант — S\_ISUID, S\_ISGID, S\_ISVTX, S\_IRUSR, S\_IWUSR, S\_IXUSR, S\_IRGRP, S\_IWGRP, S\_IXGRP, S\_IROTH, S\_IWOTH, S\_IXOTH.

24. Семиуровневая модель взаимодействия открытых систем.

Лекция 22



25. Общие функции уровней семиуровневой модели взаимодействия открытых систем.

Лекция 22

*Первый, физический уровень (physical layer, L1)*

Он отвечает за обмен физическими сигналами между физическими устройствами, «железом». Устройства физического уровня оперируют битами. Они передаются по кабелям (например, через оптоволокно) или без — например, через Bluetooth или IRDA, Wi-Fi, GSM, 4G и так далее.

*Второй уровень, канальный (data link layer, L2)*

Второй уровень решает проблему адресации при передаче информации. Канальный уровень получает биты и превращает их в кадры (frame, также «фреймы»). Задача здесь — сформировать кадры с адресом отправителя и получателя, после чего отправить их по сети. У канального уровня есть два подуровня — это MAC и LLC. MAC (Media Access Control, контроль доступа к среде) отвечает за присвоение физических MAC-адресов, а LLC (Logical Link Control, контроль логической связи) занимается проверкой и исправлением данных, управляет их передачей. На втором уровне OSI работают коммутаторы, их задача — передать сформированные кадры от одного устройства к другому, используя в качестве адресов только физические MAC-адреса.

На канальном уровне активно используется протокол ARP (Address Resolution Protocol — протокол определения адреса).

*Третий уровень, сетевой (network layer, L3)*

На третьем уровне появляется новое понятие — маршрутизация. Для этой задачи были созданы устройства третьего уровня — маршрутизаторы (их еще называют роутерами). Маршрутизаторы получают MAC-адрес от коммутаторов с предыдущего уровня и занимаются построением маршрута от одного устройства к другому с учетом всех потенциальных неполадок в сети.

*Четвертый уровень, транспортный (transport layer, L4)*

Все семь уровней модели OSI можно условно разделить на две группы:

* Media layers (уровни среды),
* Host layers (уровни хоста).

Четвертый уровень — это посредник между Host Layers и Media Layers, относящийся скорее к первым, чем к последним. Его главной задачей является транспортировка пакетов. Естественно, при транспортировке возможны потери, но некоторые типы данных более чувствительны к потерям, чем другие.

*Пятый уровень, сеансовый (session layer, L5)*

Пятый уровень оперирует чистыми данными. Помимо пятого, чистые данные используются также на шестом и седьмом уровне. Сеансовый уровень отвечает за поддержку сеанса или сессии связи. Пятый уровень оказывает услугу следующему: управляет взаимодействием между приложениями, открывает возможности синхронизации задач, завершения сеанса, обмена информации.

*Шестой уровень, представления данных (presentation layer, L6)*

О задачах уровня представления вновь говорит его название. Шестой уровень отвечает за преобразование протоколов и кодирование/декодирование данных. Шестой уровень также занимается представлением картинок (в JPEG, GIF и т.д.), а также видео-аудио (в MPEG, QuickTime). А помимо этого → шифрованием данных, когда при передаче их необходимо защитить.

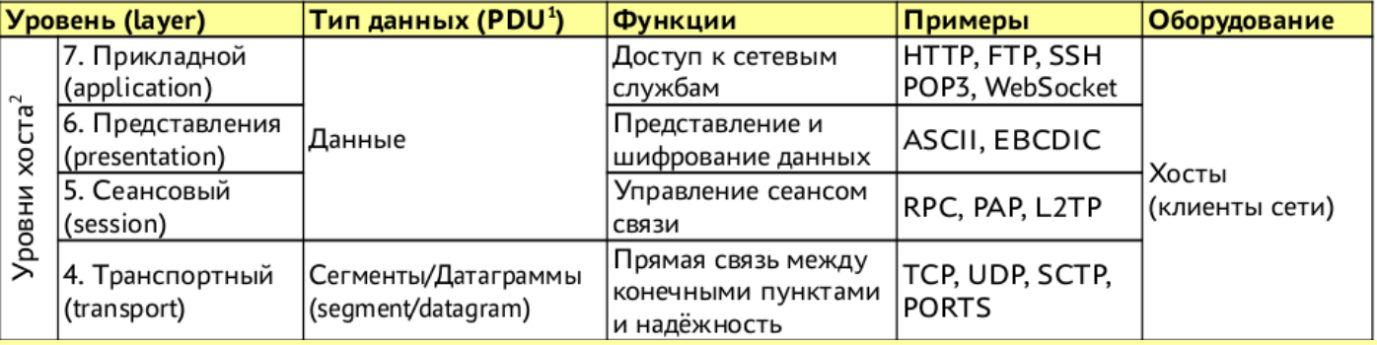
*Седьмой уровень, прикладной (application layer)*

Седьмой уровень иногда еще называют уровень приложений, но чтобы не запутаться можно использовать оригинальное название — application layer. Прикладной уровень — это то, с чем взаимодействуют пользователи, своего рода графический интерфейс всей модели OSI, с другими он взаимодействует по минимуму.

Все услуги, получаемые седьмым уровнем от других, используются для доставки данных до пользователя. Протоколам седьмого уровня не требуется обеспечивать маршрутизацию или гарантировать доставку данных, когда об этом уже позаботились предыдущие шесть. Задача седьмого уровня — использовать свои протоколы, чтобы пользователь увидел данные в понятном ему виде.

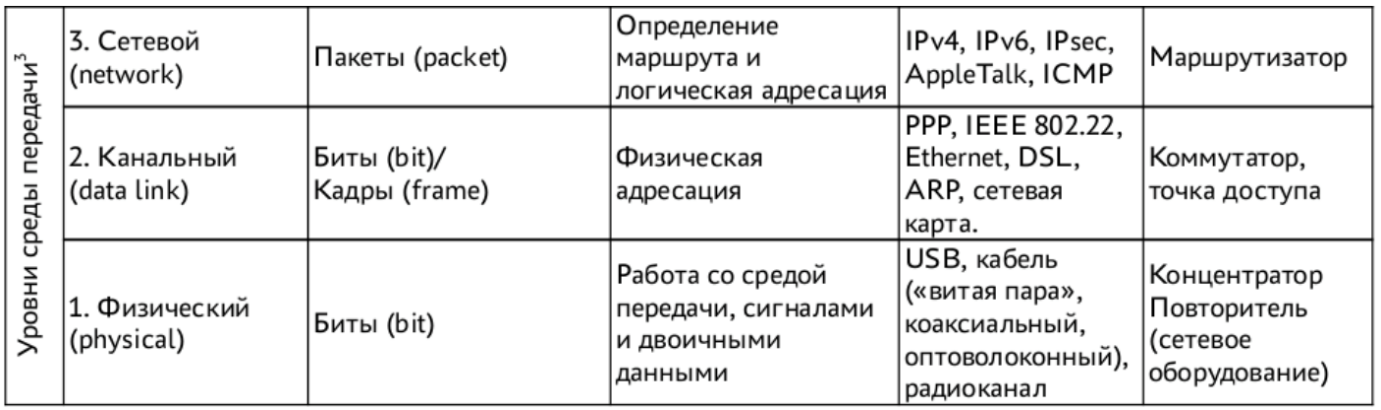
26. Уровни хоста семиуровневой модели взаимодействия открытых систем, типы данных и функции.

Лекция 22



27. Уровни среды передачи семиуровневой модели взаимодействия открытых систем, типы данных, функции и примеры оборудования.

Лекция 22



28. Классификация IP-сетей — классовая и бесклассовая адресация.

С божьей помощью лекции Глеца

Для адресации IP-сетей используется 4 байта.

Существуют два типа адресации IP-сетей — классовая и бесклассовая.

Классовая адресация

При классовой адресации сетевой адрес IPv4 разделяется на узловой и сетевой компоненты по байтовой границе следующим образом:

***Класс A***

На данный тип адреса указывает **0 на месте старшего бита** (сетевой порядок байтов) адреса.

Адрес сети (сетевой адрес) содержится в самом старшем байте, а адреса узлов занимают оставшиеся три байта.

***Класс B***

На данный тип адреса указывает **двоичное значение 10 на месте двух самых старших битов** (сетевой порядок байтов) адреса.

Адрес сети содержится в двух старших байтах, а адреса узлов занимают оставшиеся два байта.

***Класс C***

На данный тип адреса указывает д**воичное значение 110 на месте самых трех старших битов** (сетевой порядок байтов) адреса.

Адрес сети содержится в первых трёх старших байтах, а адреса узлов занимают оставшийся байт.

Классовая адресация в настоящее время устарела и была заменена на бесклассовую адресацию (CIDR), при которой компоненты сети и узла в адресе могут занимать произвольное число битов (а не байтов).

**Бесклассовая адресация** — метод IP-адресации, позволяющий гибко управлять пространством IP-адресов, не используя жёсткие рамки классовой адресации. Использование этого метода позволяет экономно использовать ограниченный ресурс IP-адресов, поскольку возможно применение различных масок подсетей к различным подсетям.

IP-адрес трактуется, как массив бит. Маска подсети задаёт какие биты в IP-адресе являются адресом сети.

Блок адресов задается указанием начального адреса и маски подсети.

Бесклассовая адресация основывается на **переменной длине маски подсети** (англ. variable length subnet mask, VLSM), в то время, как в классовой адресации длина маски подсети имела всего лишь 3 фиксированных значения.

Пример подсети 192.0.2.16/28 с применением бесклассовой адресации:



В маске подсети 28 бит слева — единицы. В таком случае говорят о длине префикса подсети в 28

бит и указывают через косую черту (символ /) после базового адреса.

192.0.2.16 — адрес подсети

192.0.2.17...30 — адреса хостов (14 штук)

192.0.2.31 — широковещательный адрес

192.0.2.32— адрес следующей подсети

29. Сокеты AF\_INET и их типы. Адреса IPv4, их типы и формат адреса.

Лекция 22….. С божьей помощью

AF\_INET — реализация протокола IPv4 в Linux

tcp\_socket = socket(AF\_INET, **SOCK\_STREAM**, 0);

udp\_socket = socket(AF\_INET, **SOCK\_DGRAM**, 0);

raw\_socket = socket(AF\_INET, **SOCK\_RAW**, protocol);

В Linux реализован протокол Интернета.

Программный интерфейс совместим с интерфейсом сокетов BSD (*socket(), bind(), listen(), connect(), accept(), gethostbyname(), gethostbyaddr()*).

Сокет IP создаётся с помощью **системного вызова socket()**:

socket(AF\_INET, socket\_type, protocol);

socket\_type — возможные типы сокета:

SOCK\_STREAM — для открытия сокета tcp(7);

SOCK\_DGRAM — для открытия сокета udp(7);

SOCK\_RAW — для открытия сокета raw(7) с прямым доступом к протоколу IP

protocol — задаётся протокол IP, который указывается в IP-заголовке принимаемых или отправляемых пакетов. Допустимые значения для параметра protocol:

0 и IPPROTO\_TCP — для сокетов TCP,

0 и IPPROTO\_UDP — для сокетов UDP.

Адрес IP-сокета определяется как комбинация IP-адреса интерфейса и номера порта.

В самом протоколе IP нет номеров портов, они реализуются протоколами более высокого уровня, например udp(7) и tcp(7).

У неструктурированных (raw) сокетов номер протокола IP указывается в sin\_port.

struct sockaddr\_in {

sa\_family\_t sin\_family; // семейство адресов: AF\_INET

in\_port\_t sin\_port; // порт сокета в сетевом порядке байт

struct in\_addr sin\_addr; // Интернет-адрес

};

// Интернет-адрес

struct in\_addr {

uint32\_t s\_addr; // адрес в сетевом порядке байт

};

Значение sin\_family всегда устанавливается в AF\_INET. Это обязательно.

В sin\_port указывается номер порта в сетевом порядке байт.

Порты, номера которых меньше 1024, называются привилегированными портами (зарезервированными портами). Только привилегированные процессы могут быть связаны с этими сокетами с помощью bind.

30. Динамическое связывание, преимущества и недостатки по сравнению со статическим.

С одной стороны программы не могут изменять код — для операционной системы исполняемый код считается «только для чтения».

С другой стороны, программы имеют большие объемы общего кода, поэтому вместо того, чтобы реплицировать его для каждого исполняемого файла, это общий код может совместно использоваться многими исполняемыми файлами.

Виртуальная память позволяет это сделать легко — на физические страницы памяти (страничные блоки), в которые загружен общий код (библиотеки), можно легко ссылаться с любого количества виртуальных страниц в любом количестве адресных пространств.

Таким образом, несмотря на то, что в памяти системы существует только одна физическая копия кода библиотеки, каждый процесс может иметь доступ к этому коду библиотеки по любому виртуальному адресу, который ему нравится.

**Преимущества библиотек динамической компоновки:**

- только один экземпляр библиотеки динамической компоновки загружается в память, если

несколько программ, работающих одновременно, используют одну и ту же библиотеку.

- библиотеку динамической компоновки можно обновлять без изменения исполняемого файла.

- библиотеку динамической компоновки можно вызвать из большинства языков программирования, таких как Pascal, C# и Visual Basic. Можно даже вызвать из Java, но сложно.

**Недостатки динамически подключаемых библиотек:**

- вся библиотека загружается в память, даже если требуется лишь небольшая ее часть.

- загрузка библиотеки динамической компоновки требует дополнительного времени при загрузке

исполняемого файла программы.

- вызов функции в библиотеке динамической компоновки менее эффективен, чем вызов в статической библиотеке, из-за дополнительных накладных расходов на вызов и из-за менее эффективного использования кэша кода.

- библиотека динамической компоновки должна распространяться вместе с исполняемым файлом.

- несколько программ, установленных на одном компьютере, должны использовать одну и ту же версию библиотеки динамической компоновки, если разные версии имеют одинаковое имя, что может вызвать множество проблем с совместимостью.