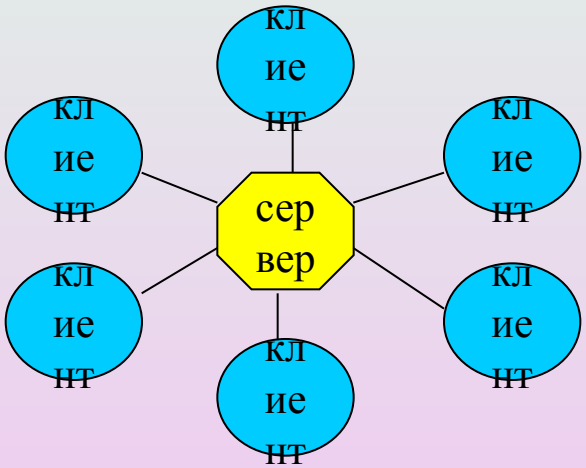
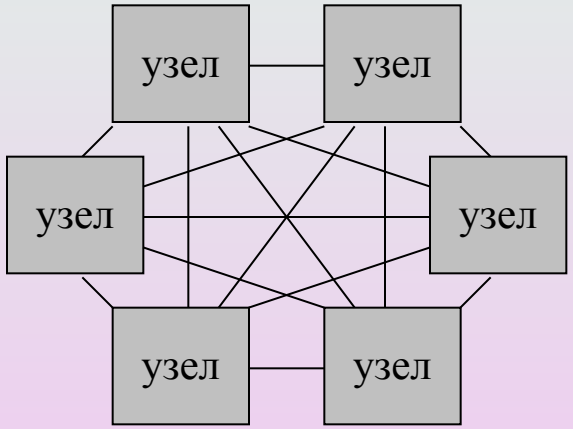
Технология клиент-сервер

- Архитектура распределённой вычислительной системы, в которой логические компоненты приложения разделены на клиентский и серверный процессы.
- В отличие от одноранговых распределённых вычислительных систем (peer-to-peer) функциональность компонентов приложения чётко разделена между клиентом и сервером.





Rev. 1.0 / 1.10.2007

Технология клиент-сервер

Особенности клиента:

- инициатор запроса;
- обычно предусматривает соединение с небольшим количеством серверов;
- обычно непосредственно взаимодействует с конечным пользователем (посредством GUI).

Особенности сервера:

- пассивно ожидает запроса клиента;
- обычно предусматривает соединение с большим количеством клиентов;
- обычно непосредственно не взаимодействует с пользователем.

Технология клиент-сервер

- Двухуровневая архитектура: клиент и сервер.

- Трёхуровневая (многоуровневая) архитектура:
  клиент,
  сервер приложений (обрабатывает данные для клиентов),
  сервер баз данных (хранит данные для сервера приложений).

+	-
лучше масштабируется, возможно улучшение производительности и надёжности	увеличение сетевого трафика, сложнее наладка и тестирование

Технология клиент-сервер

Преимущества (по сравнению с peer-to-peer):

- инкапсуляция
- повышение безопасности (контроль доступа со стороны сервера)
- централизованное обновление данных
- большое количество готовых решений
- возможность создания разных по функциональности клиентов

Недостатки (по сравнению с peer-to-peer):

- неравномерная нагрузка на коммуникационные каналы
- система в целом уязвима к отказам сервера

Inter-Process Communication

- Файл (file)
- Сигнал (signal)
- Сокет (socket)
- Конвейер (pipe), тж. буфер FIFO
- Семафор (semaphore)
- Разделяемая память (shared memory)
- Очередь сообщений (message queue)

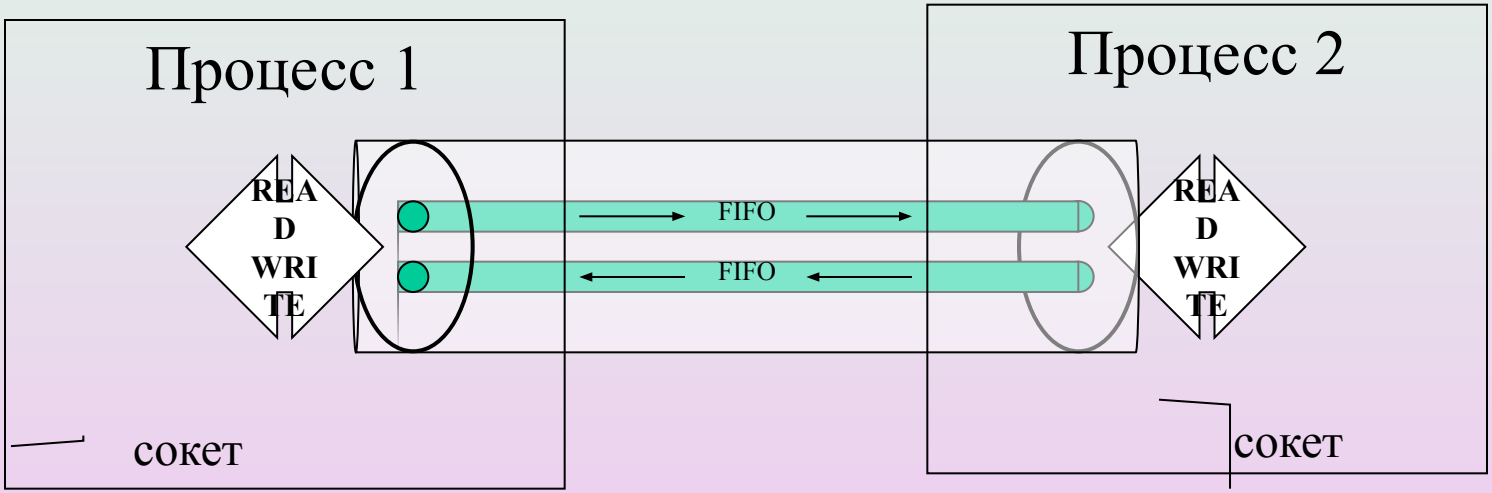
и др.

Сокеты

Конечная точка двунаправленного упорядоченного потока байт между двумя процессами (нитями).

Примечания:

1) Могут быть однонаправленными (если постараться).
2) Могут не гарантировать надёжность доставки или упорядоченность.
3) Ничто не мешает использовать «оба конца» такого потока внутри одного процесса – только нет смысла.



Сокеты

Сетевые сокеты (network sockets или Internet sockets)

- TCP-сокеты
- UDP-сокеты
- raw-IP-сокеты

«Имя» сокета = IP-адрес и порт.

Локальные сокеты (Unix domain sockets или POSIX Local IPC Sockets)

«Имя» сокета = имя спец. файла.

Berkeley Sockets API (1983) – большинство ОС, в. ч. в Windows – Winsock (особенность Winsock – функции для асинхронной работы с сокетами)
Альтернатива: Transport Layer Interface (TLI) или X/Open Transport Interface (XTI) в Solaris, Mac OS

Berkeley Sockets API

Заголовочные файлы:

- **<sys/socket.h>** - ключевые функции работы с сокетами и базовые структуры данных
- **<netinet/in.h>** - структуры данных специфические для сетевых сокетов
- **<sys/un.h>** - структуры данных специфические для локальных сокетов
- **<arpa/inet.h>** - функции для манипуляции IP-адресами
- **<netdb.h>** - функции для перевода названий протоколов и хостов в численные значения (в т.ч. DNS-запросы)

Berkeley Sockets API

Преобразование в (из) сетевой порядок байт

```
#include <netinet/in.h>

uint32_t htonl(uint32_t hostlong);
  /* host-to-network-long */

uint16_t htons(uint16_t hostshort);
  /* host-to-network-short */

uint32_t ntohl(uint32_t netlong);
  /* network-to-host-long */

uint16_t ntohs(uint16_t netshort);
  /* network-to-host-short */
```

Berkeley Sockets API

Действия сервера:

- Создать сокет
- Задать имя сокету
- Перевести сокет в режим прослушивания
- Принять соединение (создаётся новый сокет)
- Обслужить клиента
- Закрыть сокеты

Действия клиента:

- Создать сокет
- Инициировать соединение с сервером
- Запросить у сервера данные и принять их
- Закрыть сокет

Berkeley Sockets API

Создание сокета

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

domain: PF_UNIX/PF_LOCAL, PF_INET, PF_INET6 и др.
type: SOCK_STREAM, SOCK_DGRAM, SOCK_RAW и др.
protocol (<netinet/in.h>): IPPROTO_TCP, IPPROTO_UDP и др.

Результат: -1 при ошибке, в противном случае – целое неотрицательное число (дескриптор сокета).
Пример:

```
int sock;
sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
if (sock < 0) {
   perror("socket() failed");
   exit(1);
}
```

Berkeley Sockets API

Создание сокета

Ошибки:
EPROTONOSUPPORT – не поддерживается указанный протокол
EAFNOSUPPORT – не поддерживается указанное семейство адресов
EINVAL – неизвестный протокол или семейство адресов
EACCES – доступ на создание сокета данного типа запрещён
ENFILE, EMFILE, ENOBUFS, ENOMEM – недостаточно системных ресурсов
и др.

Berkeley Sockets API

Задание имени сокету

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(
   int sockfd,           /*дескриптор*/
   struct sockaddr *my_addr,  /*адрес («имя»)*/
   socklen_t addrlen);     /*размер my_addr*/
```

Результат: 0 – при успешном завешении, -1 – при ошибке.
Форматы адресов различаются для различных семейств протоколов и различных семейств адресов. Структура sockaddr – это 'родовая' структура:

```
struct sockaddr {
   unsigned short int sa_family;
   unsigned char sa_data[14];
};
```

Berkeley Sockets API

Задание имени сокету

Для семейства PF_INET:

```
struct sockaddr_in {
   sa_family_t sin_family; /*семейство: AF_INET*/
   in_port_t sin_port;     /*порт*/
   struct in_addr sin_addr;   /*IP-адрес*/
   unsined char sin_zero[8];
};
struct in_addr {in_addr_t s_addr;};
```

Пример:

```
struct sockaddr_in saddr;
saddr.sin_family = AF_INET;
saddr.sin_port = htons(80); /*HTTP-порт*/
saddr.sin_addr.s_addr = htonl(INADDR_ANY); /*0.0.0.0*/
if (bind(sock,(struct sockaddr*)&saddr,sizeof(saddr))) {
   perror("bind() failed");
   exit(1);
}
```

Berkeley Sockets API

Задание имени сокету

Ошибки:
EINVAL – у сокета уже есть «имя»
EACCES – попытка непривилегированного процесса привязаться к порту с номером меньше 1024
EADDRINUSE – сокет с указанным именем уже (ещё) существует
EADDRNOTAVAIL – указанный адрес не связан ни с одним локальным сетевым интерфейсом
ENOBUFS, ENOMEM – не хватает системных ресурсов
EBADF, ENOTSOCK – указан «левый» дескриптор

Berkeley Sockets API

Перевод сокета в режим прослушивания (SOCK_STREAM)

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

sockfd: дескриптор сокета
backlog: максимальная длина очереди соединений, ожидающих подтверждения (во многих ОС не может превышать 5, в Linux – предлагается значение SOMAXCONN=128)

Результат: 0 – при успешном завешении, -1 – при ошибке.

Пример:

```
if (listen(sock, SOMAXCONN)) {
   perror("listen() failed");
   exit(1);
}
```

Berkeley Sockets API

Перевод сокета в режим прослушивания

Ошибки:
EADDRINUSE – сокет с указанным именем уже (ещё) существует и находится в режиме прослушивания
EBADF, ENOTSOCK – указан «левый» дескриптор
EOPNOTSUPP – неправильный тип сокета (должен быть SOCK_STREAM или SOCK_SEQPACKET)

Berkeley Sockets API

Приём (подтверждение) соединения

```
#include <sys/types.h>
#include <sys/socket.h>
int accept(
   int sockfd,          /*дескриптор*/
   struct sockaddr *addr,  /*адрес («имя»)*/
   socklen_t *addrlen);       /*размер my_addr*/
```

Результат: -1 – при ошибке, в противном случае – целое неотрицательное число (дескриптор нового сокета). Новый сокет не находится в состоянии прослушивания, атрибуты сокета *sockfd* (например O_NONBLOCK или O_ASYNC) им не наследуются. Состояние исходного сокета *sockfd* не меняется. Вызов может быть блокирующим.

Пример:

```
struct sockaddr_in remote_addr;
socklen_t len = sizeof(remote_addr);
int csock;
csock=accept(sock,(struct sockaddr*)&remote_addr,&len);
if (csock<0) { …   /*ошибка*/
```

Berkeley Sockets API

Приём (подтверждение) соединения

Ошибки:
EBADF, ENOTSOCK – указан «левый» дескриптор
EAGAIN, EWOULDBLOCK – неблокирующий сокет, нет соединения
EOPNOTSUPP – неправильный тип сокета (должен быть SOCK_STREAM)
EINTR – вызов был прерван сигналом
EINVAL – сокет не находился в состоянии прослушивания
ECONNABORTED – разрыв соединения
ENFILE, EMFILE, ENOBUFS, ENOMEM – недостаточно системных ресурсов
и др.

Berkeley Sockets API

Запрос соединения

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(
   int sockfd,           /*дескриптор*/
   struct sockaddr *addr, /*адрес («имя»)*/
   socklen_t addrlen);    /*размер my_addr*/
```

Результат: 0 – при успешном соединении, -1 – при ошибке. Если у сокета *sockfd* нет локального «имени», то функция его задаёт. Вызов может быть блокирующим.

Пример:

```
struct sockaddr_in serv_addr;
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(80);  /*HTTP-порт*/
serv_addr.sin_addr.s_addr =
   htonl(INADDR_LOOPBACK); /*127.0.0.1*/
if (connect(sock, (struct sockaddr*) &serv_addr,
   sizeof(serv_addr))) { …/*ошибка*/
```

Berkeley Sockets API

Запрос соединения

Ошибки:
EBADF, ENOTSOCK – указан «левый» дескриптор
EISCONN – сокет уже был соединён
ECONNREFUSED – на «другой» стороне указанный порт никто не слушает
ETIMEDOUT – таймаут
ENETUNREACH – сеть недоступна
EINPROGRESS – неблокирующий сокет, соединение ещё устанавливается
EALREADY – неблокирующий сокет, предыдущая попытка ещё не закончилась
EACCES, EPERM – попытка широковещательной рассылки без установки соответствующей опции сокета
и др.

Berkeley Sockets API

Передача данных в сокет

```
#include <sys/types.h>
#include <sys/socket.h>
size_t sendto(
   int sockfd,          /*дескриптор сокета*/
   const void *buf,     /*что передаём*/
   size_t len,       /*размер передаваемых данных*/
   int flags,           /*флаги*/
   const struct sockaddr *to, /*куда передаём*/
   socklen_t tolen);       /*адрес структуры to*/
size_t send(
   int sockfd,
   const void *buf,
   size_t len,
   int flags);

#include <unistd.h>
size_t write(int fd, const void *buf, size_t len);
```

Berkeley Sockets API

Передача данных в сокет

```
write(sock,buf,len) <=>
send(sock,buf,len,0) <=>
sendto(sock,buf,len,0,NULL,0);
```

Флаги:
MSG_OOB – послать данные out-of-band
MSG_DONTROUTE – не посылать данные через шлюз
MSG_DONTWAIT – неблокирующая операция (возвращается EAGAIN)
MSG_NOSIGNAL – не посылать сигнал SIGPIPE, если соединение разорвано (будет фиксироваться ошибка EPIPE)

Результат: -1 – при ошибке, в противном случае возвращается количество переданных байт. Вызов может быть блокирующим.

send() и write() можно использовать для сокетов в состоянии установленного соединения (тип SOCK_STREAM), sendto() – для всех остальных (SOCKET_DGRAM)

Berkeley Sockets API

Передача данных в сокет

Ошибки:
EBADF, ENOTSOCK – указан «левый» дескриптор
EAGAIN, EWOULDBLOCK – неблокирующий сокет, передача данных заблокировала бы его
EISCONN – сокет уже был соединён, а в sendto() указан получатель
ECONNRESET – «другая» сторона сбросила соединение
ENOTCONN, EDESTADDRREQ – сокет не соединён, а адрес получателя в sendto() не указан
EINTR – вызов был прерван сигналом
ENOBUFS, ENOMEM – недостаточно системных ресурсов
EFAULT – недействительный адрес буфера
EOPNOTSUPP – недействительное сочетание флагов для данного сокета
EPIPE – с локальной стороны сокет был закрыт для передачи (также посылается сигнал SIGPIPE, если не выставлена опция MSG_NOSIGNAL)
и др.

Berkeley Sockets API

Приём данных из сокета

```
#include <sys/types.h>
#include <sys/socket.h>
size_t recvfrom(
   int sockfd,          /*дескриптор сокета*/
   const void *buf,     /*куда принимаем*/
   size_t len,             /*размер буфера приёма*/
   int flags,           /*флаги*/
   const struct sockaddr *from, /*откуда принимаем*/
   socklen_t fromlen);      /*адрес структуры from*/
size_t recv(
   int sockfd,
   const void *buf,
   size_t len,
   int flags);

#include <unistd.h>
size_t read(int fd, const void *buf, size_t len);
```

Berkeley Sockets API

Приём данных из сокета

```
read(sock,buf,len) <=>
recv(sock,buf,len,0) <=>
recvfrom(sock,buf,len,0,NULL,0);
```

Флаги:
MSG_OOB – принять данные out-of-band
MSG_PEEK – получить данные, но не удалять их из входного буфера
MSG_WAITALL – ожидать данные до тех пор, пока не будет получено запрошенное количество байт
MSG_DONTWAIT – неблокирующая операция (возвращается EAGAIN)

Результат: -1 – при ошибке, в противном случае возвращается количество принятых байт. Если возвращается 0 – сокет закрыт с «другой» стороны. Последующие вызовы приведут к SIGPIPE. Вызов может быть блокирующим. recv() и read() можно использовать для сокетов в состоянии установленного соединения (тип SOCK_STREAM), recvfrom() – для всех остальных (SOCKET_DGRAM)

Berkeley Sockets API

Приём данных из сокета

Ошибки:
EBADF, ENOTSOCK – указан «левый» дескриптор
EAGAIN, EWOULDBLOCK – неблокирующий сокет, приём данных заблокировал бы его
ENOTCONN – сокет не соединён
EINTR – вызов был прерван сигналом
ENOBUFS, ENOMEM – недостаточно системных ресурсов
EFAULT – недействительный адрес буфера
и др.

Berkeley Sockets API

Закрытие сокета

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

how:
SHUT_RD (0) – прекратить приём данных из сокета
SHUT_WR (1) – прекратить передачу данных в сокет
SHUT_RDWR (2) – прекратить приём и передачу
Только для сокетов, ориентированных на соединение (SOCK_STREAM).
Функция не освобождает дескриптор. Сокет должен быть закрыт вызовом:

```
#include <unistd.h>
int close(int fd);
```

Результат: 0 – при успешном соединении, -1 – при ошибке.

Berkeley Sockets API

Функции манипуляции IP-адресом

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
```

```
int inet_aton(const char *cp, struct in_addr *inp);
```

Преобразует IP-адрес, задаваемый первым аргументом (ASCIIZ-строка), из стандартной формы в виде десятичных чисел, разделенных точками, в бинарную форму в сетевом порядке байтов. Результат преобразования помещается в структуру, на которую указывает второй параметр. Функция возвращает ненулевое значение при успешном преобразовании, 0 – при ошибке.

Пример:

```
struct sockaddr_in addr;
inet_aton("192.168.0.1", &addr.sin_addr);
```

Berkeley Sockets API

Функции манипуляции IP-адресом

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
```

```
in_addr_t inet_addr(const char *cp);
```

Преобразует IP-адрес, задаваемый аргументом (ASCIIZ-строка), из стандартной формы в виде десятичных чисел, разделенных точками, в бинарную форму в сетевом порядке байтов. При ошибке возвращается значение INADDR_NONE (-1), которое может также интерпретироваться как адрес 255.255.255.255, поэтому предпочтительно использование inet_aton().

Пример:

```
struct sockaddr_in addr;
addr.sin_addr.s_addr = inet_addr("192.168.0.1");
```

Berkeley Sockets API

Функции манипуляции IP-адресом

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
char *inet_ntoa(struct in_addr in);
```

Преобразует IP-адрес, задаваемый аргументом, из бинарной формы в сетевом порядке байт в стандартную форму в виде десятичных чисел, разделенных точками. Сформированная ASCIIZ-строка хранится в статическом буфере, который при последующих вызовах переписывается.

Пример:

```
struct sockaddr_in raddr;
socklen_t len = sizeof(raddr);
int csock;
csock = accept(sock, (struct sockaddr*) &raddr, &len);
if (csock<0) { /*ошибка*/ }
printf("Peer address: %s:%d\n",
   inet_ntoa(raddr.sin_addr),
   ntohs(raddr.sin_port));
```

Berkeley Sockets API

Функции манипуляции IP-адресом

Стандарт POSIX 1003.1-2001 рекомендует вместо функции inet_aton() использовать указанную ниже, поскольку она поддерживает и IPv4, и IPv6:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
int inet_pton(int af, const char *src, void *dst);
```

af: AF_INET (*dst*: struct in_addr*), AF_INET6 (*dst*: struct in6_addr*)

Функция inet_pton() преобразует IP-адрес, задаваемый аргументом *src*, из строковой формы в бинарную в сетевом порядке байт в соответствии с указанным семейством адресов *af*.
Функция возвращает положительное значение при успешном завершении, 0 – если строка не может быть интерпретирована как IP-адрес и отрицательное значение, если *af* содержит неподдерживаемое значение семейства..

Berkeley Sockets API

Функции манипуляции IP-адресом

Стандарт POSIX 1003.1-2001 рекомендует вместо функции inet_ntoa() использовать указанную ниже, поскольку она поддерживает и IPv4, и IPv6:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
const char *inet_ntop(
   int af, const void *src, char *dst, size_t cnt);
```

af: AF_INET (*src*: struct in_addr*), AF_INET6 (*src*: struct in6_addr*)

Функция inet_ntop() преобразует IP-адрес, задаваемый аргументом *src*, из бинарной формы в сетевом порядке байт в строковую форму. Сформированная ASCIIZ-строка помещается в указанный пользователем буфер *dst*. Пользователь должен зарезервировать по крайней мере INET_ADDRSTRLEN (или INET6_ADDRSTRLEN) байт под буфер *dst*. Функция возвращает значение указателя *dst* или NULL – при ошибке.

Berkeley Sockets API

Функции работы с БД узлов сети (DNS)

```
#include <netdb.h>
#include <sys/socket.h>

struct hostent *gethostbyname(const char *cp);
struct hostent *gethostbyaddr(
   const char *addr, int len, int type);
```

Функция выполняет поиск в БД узлов (DNS) информации об указанном хосте. Для gethostbyname() хост задаётся доменным именем или IP-адресов в десятично-точечной нотации. Для gethostbyaddr() хост задаётся IP-адресом в бинарном виде в сетевом порядке байт (*addr*), второй аргумент определяет длину адреса, а третий должен быть AF_INET или AF_INET6).
При ошибке возвращается NULL.
Функции могут искать информацию в /etc/hosts, в DNS, в LDAP, в NIS и др. Способ поиска информации и порядок опроса служб определяются Name Service Switch Configuration (/etc/nsswitch.conf)

Berkeley Sockets API

Функции работы с БД узлов сети (DNS)

```
struct hostent {
   char *h_name;     /*официальное имя хоста*/
   char **h_aliases;    /*массив псевдонимов*/
   int h_addrtype;  /*тип адреса – AF_INET*/
   int h_length;     /*длина адреса*/
   char **h_addr_list; /*массив адресов*/
};
```

Массивы *h_aliases* и *h_addr_list* ограничены элементом со значением 0. Адреса хранятся в сетевом порядке байт. Функции gethostbyname() и gethostbyaddr() возвращают указатель на структуру, размещаемую в области данных ядра ОС (не надо отводить место под структуру – только под указатель). Если для gethostbyname() в качестве параметра указать IP-адрес хоста, то он будет просто скопирован в поле *h_name* (DNS-запрос не будет выполняться). Если нужно получить доменное имя по IP-адресу, используйте gethostbyaddr().

Berkeley Sockets API

Функции работы с БД узлов сети (DNS)

Пример:

```
struct hostent *he;
int i;
char buf[INET_ADDRSTRLEN]="";
he = gethostbyname("iq.karelia.ru");
if (!he) { /*ошибка*/ }
printf("\nOfficial name: %s", he->h_name);
i=0;
while (he->h_aliases[i])
   printf("\nAlias: %s", he->h_aliases[i++]);
if (he->h_addrtype == AF_INET) {
   if (inet_ntop(AF_INET, he->h_addr_list[0],
      buf, INET_ADDRSTRLEN))
         printf("\nIP: %s\n", buf);
}
```

Berkeley Sockets API

Функции работы с БД узлов сети (DNS)

Если требуется выполнить ряд запросов к DNS, имеет смысл объединить их в сеанс работы с DNS на основе одного соединения.

```
void sethostent(int stayopen);
```

Если *stayopen*=1, TCP-соединение с DNS не будет закрываться между последовательными запросами. Если *stayopen*=0, каждый запрос к DNS будет выполнен в виде отдельной UDP-дейтаграммы.

Завершение сеанса работы с DNS на основе TCP-соединения:

```
void endhostent(void);
```

Berkeley Sockets API

Функции работы с БД узлов сети (DNS)

При ошибке функции работы с БД узлов сети помещают код ошибки в переменную h_error. Например:
HOST_NOT_FOUND – информация о хосте не найдена
NO_ADDRESS, NO_DATA – с запрошенным именем не связан IP-адрес
NO_RECOVERY – ошибка при работе с БД узлов сети
TRY_AGAIN – временная ошибка

Можно получить текстовое сообщение об ошибке при помощи функции:

```
const char *hstrerror(int err);
```

у которой в качестве параметра передаётся значение h_error. Или вывести сообщение об ошибке на стандартное диагностическое устройство stderr:

```
void herror(const char *s);
```

s – дополнительный комментарий, выводимый перед сообщением об ошибке.

Berkeley Sockets API

Функции работы с БД информации о сетевых службах

```
#include <netdb.h>

struct servent *getservbyname(
   const char *name, const char *proto);
struct servent *getservbyport(
   int port, const char *proto);
```

Функция getservbyname() выполняет поиск в БД (/etc/services) информации (номера порта) об указанной сетевой службе. Функция getservbyport() возвращает информацию о сетевой службе по номеру её порта (*port* – в сетевом порядке байт). Параметр *proto* может быть NULL, тогда поиск осуществляется для любого протокола.

При ошибке возвращается NULL.

Berkeley Sockets API

Функции работы с БД информации о сетевых службах

```
struct servent {
   char *s_name;     /*официальное имя службы*/
   char **s_aliases;   /*массив псевдонимов*/
   int s_port;       /*номер порта*/
   char *s_proto;    /*протокол*/
};
```

Массив *s_aliases* ограничен элементом со значением 0. Номер порта *s_port* хранятся в сетевом порядке байт. Функции getservbyname() и getservbyport() возвращают указатель на структуру, размещаемую в области данных ядра ОС (не надо отводить место под структуру – только под указатель).

Berkeley Sockets API

Функции работы с БД информации о сетевых службах

Пример:

```
int i;
struct servent *se;
se = getservbyname("www", NULL);
/* или так: se = getservbyport(htons(80), "tcp"); */
if (!se) {
   fprintf(stderr, "\nservice not found!\n");
       exit(1);
}
printf("\nOfficial name: %s", se->s_name);
i=0;
while (se->s_aliases[i])
   printf("\nAlias: %s", se->s_aliases[i++]);
printf("\n%s:%d\n", se->s_proto, ntohs(se->s_port));
```

Berkeley Sockets API

Функции работы с БД информации о протоколах (/etc/protocols)

```
#include <netdb.h>

struct protoent *getprotobyname(const char *name);

struct protoent *getprotobynumber(int proto);

struct protoent {
   char *p_name;    /*официальное имя*/
   char **p_aliases;   /*псевдонимы*/
   int p_proto;     /*номер протокола*/
};
```

Berkeley Sockets API

Опции сокета

```
#include <sys/types.h>
#include <sys/socket.h>
int getsockopt(int s, int level, int optname, void
*optval, socklen_t *optlen);
int setsockopt(int s, int level, int optname, const void
*optval, socklen_t optlen);
```

Функции возвращают 0 при успешном возврате или -1 при ошибке.
level: SOL_SOCKET
optname:
SO_KEEPALIVE (параметр – int) – посылать пакеты “keep-alive” для TCP
SO_REUSEADDR (параметр – int) – менее строгие правила именования сокета при выполнении bind (можно «привязать» сокет к порту, если для него ещё остались открытые соединения)

man 7 socket