Системное программное обеспечение локальных компьютерных сетей

Программный интерфейс взаимодействия сокетов Беркли

Денис Пынькин

2013 – 2014

e-mail: denis.pynkin@bsuir.by

http://goo.gl/32cTB

СЧАСТЬЕ ДЛЯ ВСЕХ, ДАРОМ, И ПУСТЬ НИКТО НЕ УЙДЕТ ОБИЖЕННЫЙ!

(c)Стругацкие, Пикник на обочине

В стандартах TCP/IP не даны подробные сведения о том, каким образом прикладное ПО должно взаимодействовать с ПО протоколов TCP/IP; в них описаны только необходимые функциональные средства, а возможность определять конкретные требования к реализации API-интерфейса предоставлена системным проектировщикам.
Интерфейс между TCP/IP и приложениями, в которых используются эти протоколы, определены неформально, в виде рекомендаций, а не требований.

Неформальная спецификация

К преимуществам относятся гибкость и широкая применяемость, что позволяет проектировщикам реализовывать протоколы TCP/IP для любых операционных систем. Также проектировщики имеют возможность использовать органихацию интерфейса, наиболее подходящую для операционной системы (например процедурную или основанную на передаче сообщений).

Неформальная спецификация

К преимуществам относятся гибкость и широкая применяемость, что позволяет проектировщикам реализовывать протоколы TCP/IP для любых операционных систем. Также проектировщики имеют возможность использовать органихацию интерфейса, наиболее подходящую для операционной системы (например процедурную или основанную на передаче сообщений).
Недостатком неформальной спецификации является то, что ее применение может привести к появлению различий в отдельных деталях реализации интерфейса для каждой ОС.

На практике существует лишь небольшое количество API-интерфейсов, позволяющих использовать протоколы TCP/IP во всех приложениях. На данный момент получили наиболее широкое распространение 3 интерфейса:

- сокеты Беркли (сокеты, интерфейс сокетов), разработанный в ун-те Беркли;
- Windows Sockets, разработанный в компании Microsoft;
- TLI (Transport Layer Interface), разработанный AT&T для системы Unix версии System V.

Функциональные средства интерфейса

- распределение локальных ресурсов для связи;

Функциональные средства интерфейса

- распределение локальных ресурсов для связи;
- задание локальной и удаленной оконечных точек связи;

Функциональные средства интерфейса

- распределение локальных ресурсов для связи;
- задание локальной и удаленной оконечных точек связи;
- инициирование соединения (клиент);

Функциональные средства интерфейса

- распределение локальных ресурсов для связи;
- задание локальной и удаленной оконечных точек связи;
- инициирование соединения (клиент);
- передача дейтаграммы (клиент);

Функциональные средства интерфейса

- распределение локальных ресурсов для связи;
- задание локальной и удаленной оконечных точек связи;
- инициирование соединения (клиент);
- передача дейтаграммы (клиент);
- ожидание входящего запроса на установление соединения (сервер);

Функциональные средства интерфейса

- распределение локальных ресурсов для связи;
- задание локальной и удаленной оконечных точек связи;
- инициирование соединения (клиент);
- передача дейтаграммы (клиент);
- ожидание входящего запроса на установление соединения (сервер);
- передача или прием данных;

Функциональные средства интерфейса

- распределение локальных ресурсов для связи;
- задание локальной и удаленной оконечных точек связи;
- инициирование соединения (клиент);
- передача дейтаграммы (клиент);
- ожидание входящего запроса на установление соединения (сервер);
- передача или прием данных;
- определение момента поступления данных;

Функциональные средства интерфейса

- распределение локальных ресурсов для связи;
- задание локальной и удаленной оконечных точек связи;
- инициирование соединения (клиент);
- передача дейтаграммы (клиент);
- ожидание входящего запроса на установление соединения (сервер);
- передача или прием данных;
- определение момента поступления данных;
- выработка срочных данных;

Функциональные средства интерфейса

- распределение локальных ресурсов для связи;
- задание локальной и удаленной оконечных точек связи;
- инициирование соединения (клиент);
- передача дейтаграммы (клиент);
- ожидание входящего запроса на установление соединения (сервер);
- передача или прием данных;
- определение момента поступления данных;
- выработка срочных данных;
- обработка входящих срочных данных;

Функциональные средства интерфейса

- распределение локальных ресурсов для связи;
- задание локальной и удаленной оконечных точек связи;
- инициирование соединения (клиент);
- передача дейтаграммы (клиент);
- ожидание входящего запроса на установление соединения (сервер);
- передача или прием данных;
- определение момента поступления данных;
- выработка срочных данных;
- обработка входящих срочных данных;
- корректное завершение соединения;

Функциональные средства интерфейса

- распределение локальных ресурсов для связи;
- задание локальной и удаленной оконечных точек связи;
- инициирование соединения (клиент);
- передача дейтаграммы (клиент);
- ожидание входящего запроса на установление соединения (сервер);
- передача или прием данных;
- определение момента поступления данных;
- выработка срочных данных;
- обработка входящих срочных данных;
- корректное завершение соединения;
- обработка запроса на завершение соединения от удаленного участника соединения;

Функциональные средства интерфейса

- распределение локальных ресурсов для связи;
- задание локальной и удаленной оконечных точек связи;
- инициирование соединения (клиент);
- передача дейтаграммы (клиент);
- ожидание входящего запроса на установление соединения (сервер);
- передача или прием данных;
- определение момента поступления данных;
- выработка срочных данных;
- обработка входящих срочных данных;
- корректное завершение соединения;
- обработка запроса на завершение соединения от удаленного участника соединения;
- аварийное прекращение связи;

Функциональные средства интерфейса

- распределение локальных ресурсов для связи;
- задание локальной и удаленной оконечных точек связи;
- инициирование соединения (клиент);
- передача дейтаграммы (клиент);
- ожидание входящего запроса на установление соединения (сервер);
- передача или прием данных;
- определение момента поступления данных;
- выработка срочных данных;
- обработка входящих срочных данных;
- корректное завершение соединения;
- обработка запроса на завершение соединения от удаленного участника соединения;
- аварийное прекращение связи;
- устранение последствий аварийных ситуаций или аварийного прекращения связи;

Функциональные средства интерфейса

- распределение локальных ресурсов для связи;
- задание локальной и удаленной оконечных точек связи;
- инициирование соединения (клиент);
- передача дейтаграммы (клиент);
- ожидание входящего запроса на установление соединения (сервер);
- передача или прием данных;
- определение момента поступления данных;
- выработка срочных данных;
- обработка входящих срочных данных;
- корректное завершение соединения;
- обработка запроса на завершение соединения от удаленного участника соединения;
- аварийное прекращение связи;
- устранение последствий аварийных ситуаций или аварийного прекращения связи;
- освобождение локальных ресурсов после завершения связи.

Сокеты Berkeley

Разработчики решили применять уже существующие системные вызовы и вводить новые только для поддержки функций TCP/IP, которые нельзя легко включить в существующий набор функций. Результаты этого проекта получили широкую известность под названием “API-интерфейса сокетов” или просто интерфейса сокетов, а разработанная система стала называться Berkeley UNIX или просто UNIX . Протокол TCP впервые появился в выпуске 4.1 дистрибутива BSD (Berkeley Software Distribution).

Проектировщики из BSD предусмотрели возможность применения различных семейств протоколов связи, далеко выходящую за рамки протоколов TCP/IP.
Среди них протоколы TCP/IP представлены единственным семейством AF_INET. Они также предусмотрели, чтобы в приложениях необходимые операции связи определялись путем указания на тип требуемой службы, а не выбора имени протокола. *Поэтому вместо указания на то, что ему требуется соединение TCP, приложение запрашивает службу типа потоковой передачи семейства протоколов Интернет.*
API-интерфейс сокетов предоставляет обобщенные функции, которые поддерживают сетевую связь с помощью многих возможных протоколов. В вызовах функций сокетов все протоколы TCP/IP упоминаются как одно семейство протоколов. Эти вызовы позволяют программистам указывать тип требуемой службы, а не имя конкретного протокола.

Сетевой адрес

Под сетевым адресом понимается видимый в пределах сети идентификатор, используемый для обозначения оконечных точек сети. Адреса есть у определенных оконечных точек хостов, могут они быть и у хостов в целом.

Сетевой адрес

Под сетевым адресом понимается видимый в пределах сети идентификатор, используемый для обозначения оконечных точек сети. Адреса есть у определенных оконечных точек хостов, могут они быть и у хостов в целом.
Процесс присвоения сетевого адреса оконечной точке называется связыванием, или привязкой, а обратное действие – освобождением, или отменой привязки.
Обычно оконечной точкой служит аппаратный сетевой интерфейс, посредством которого данные передаются и принимаются, однако с таким интерфейсом как loopback, никакой аппаратуры не ассоциировано.

Сетевой адрес

Под сетевым адресом понимается видимый в пределах сети идентификатор, используемый для обозначения оконечных точек сети. Адреса есть у определенных оконечных точек хостов, могут они быть и у хостов в целом.
Процесс присвоения сетевого адреса оконечной точке называется связыванием, или привязкой, а обратное действие – освобождением, или отменой привязки.
Обычно оконечной точкой служит аппаратный сетевой интерфейс, посредством которого данные передаются и принимаются, однако с таким интерфейсом как loopback, никакой аппаратуры не ассоциировано.
При взаимодействии процессов оконечными точками служат сокеты, они трактуются стандартом POSIX-2001 как отдельный тип файлов.

Данные о хостах как узлах сети хранятся в сетевой базе, допускающей и последовательный, и случайный доступ с возможностью поиска по именам и адресам хостов. Поддерживается база данных маршрутизации, используемая при выборе сетевого интерфейса для передачи порции данных (сетевого пакета).

Порядок байт

Данные передаются по сети в виде последовательности октетов (восьмибитных беззнаковых величин). Если некоторый элемент данных (например, адрес или номер порта) состоит более чем из восьми бит, для его передачи и хранения требуется несколько октетов. ***Сетевым*** *называется порядок октетов (байт), при котором первый (с наименьшим адресом) октет содержит старшие (наиболее значимые) биты.*

Порядок байт

Данные передаются по сети в виде последовательности октетов (восьмибитных беззнаковых величин). Если некоторый элемент данных (например, адрес или номер порта) состоит более чем из восьми бит, для его передачи и хранения требуется несколько октетов. ***Сетевым*** *называется порядок октетов (байт), при котором первый (с наименьшим адресом) октет содержит старшие (наиболее значимые) биты.*
Последовательности октетов – неудобный объект обработки на хостах, где предпочтительнее аппаратно поддерживаемые типы, в особенности целочисленные. Значения этих типов обычно хранятся с другим порядком байт, называемым **хостовым**, поэтому вполне возможно, что старшего бита не окажется в первом байте и вообще будет использоваться некое неочевидное распределение бит по байтам.

Порядок байт

Для преобразования значений типов `uint16_t` и `uint32_t` из хостового порядка байт в сетевой служат функции `htons` и `htonl`, функции `ntohs` и `ntohl` осуществляют обратную операцию.

Сокет

В API-интерфейсе сокетов реализовано новое абстрактное понятие для сетевой связи – **сокет**. Как и файл, сокет обозначается целым числом, называемым дескриптором сокета. Операционная система размещает дескрипторы сокетов в той же таблице дескрипторов, что и дескрипторы файлов. Поэтому в приложении не может присутствовать и дескриптор файлов и дескриптор сокетов с одним и тем же значением.

Создание нового дескриптора

В ОС предусмотрена отдельная системная функция `socket`, вызываемая приложением для создания сокета (функция `open` используется только для создания файлов). Общий замысел, который лёг в основу разработки интерфейса сокетов, состоял в том, чтобы для создания любого сокета было достаточно одного системного вызова. Для указания точных сведений о назначении сокета нужно выполнить еще несколько системных вызовов.

Адрес сокета

Под адресом сокета, как (удаленной) оконечной точки понимается структура, включающая идентификатор адресного семейства и адресную информацию, специфичную для данного семейства. Последняя может состоять из нескольких компонентов, в том числе сетевого адреса хоста и идентификатора конкретной оконечной точки.

Сокет – универсальное средство ввода/вывода

Сокет может применяться для любой связи. Поэтому приложение должно указывать, как он будет использоваться. В частности необходимо указать номера портов протокола и адреса локального и удаленного компьютеров.

Общая структура адреса I

Для обозначения типа адреса используется структура:

```
1 struct sockaddr{
2   u_char sa_len;
3   u_short sa_family;
4   char sa_data[14];
5 }
```

- sa_len – общая длина (только с BSD 4.4);
- sa_family – тип адреса;
- sa_data[14] – значение адреса.

AF_INET

Для обеспечения свободы выбора представлений используемых адресов, в каждом семействе протоколов спецификация сокета определяет семейство адресов для каждого типа адреса. Во всех протоколах TCP/IP применяется единственное представление адреса, а семейство адресов обозначается как AF_INET.

Семейства адресов

Адресное семейство соответствует конкретной среде взаимодействия. Стандарт POSIX-2001 определяет три таких семейства:

- AF_UNIX – Адресное семейство UNIX поддерживает межпроцессное взаимодействие в пределах одной системы.
- AF_INET – Адресное семейство, поддерживающее взаимодействие по протоколам IPv4.
- AF_INET6 – Взаимодействие по протоколам IPv6 (необязательная возможность).

Семейства адресов: socket.h

```
1  /* Supported address families. */
2  #define AF_UNSPEC 0
3  #define AF_UNIX 1 /* Unix domain sockets  */
4  #define AF_LOCAL 1 /* POSIX name for AF_UNIX */
5  #define AF_INET 2 /* Internet IP Protocol  */
6  #define AF_AX25 3 /* Amateur Radio AX.25  */
7  #define AF_IPX 4 /* Novell IPX  */
8  #define AF_APPLETALK 5 /* AppleTalk DDP  */
9  #define AF_NETROM 6 /* Amateur Radio NET/ROM  */
10 #define AF_BRIDGE 7 /* Multiprotocol bridge  */
11 #define AF_ATMPVC 8 /* ATM PVCs */
12 #define AF_X25 9 /* Reserved for X.25 project  */
13 #define AF_INET6 10 /* IP version 6 */
14 ...
15 #define AF_MAX 39 /* For now.. */
```

Структура данных для AF_INET

```
1 struct sockaddr_in{
2   u_char sin_len;
3   u_short sin_family;
4   u_short sin_port;
5   struct in_addr sin_addr;
6   char sin_zero[8];
7 }
```

- sin_len – общая длина (только с BSD 4.4);
- sin_family – тип адреса;
- sin_port – номер порта протокола;
- sin_addr – IP-адрес (иногда = u_long);
- sin_zero[8] – не используется (=0).

Типы сокетов

В пределах каждого адресного семейства могут существовать сокеты нескольких типов. В стандарте POSIX-2001 их четыре:

- SOCK_STREAM – Сокеты данного типа поддерживают надежные, упорядоченные, полнодуплексные потоки октетов в режиме с установлением соединения.
- SOCK_SEQPACKET – Аналог SOCK_STREAM с дополнительным сохранением границ между записями.
- SOCK_DGRAM – Передача данных в виде датаграмм в режиме без установления соединения.
- SOCK_RAW – Аналог SOCK_DGRAM с дополнительной возможностью доступа к протокольным заголовкам и другой информации нижнего уровня. Также известны,как “неструктурированные” или “сырые” сокеты.

1 тип сокета != 1 протокол в адресном семействе

Для каждого адресного семейства каждый тип сокета может поддерживаться одним или несколькими протоколами. В частности, в адресном семействе AF_INET для сокетов типа SOCK_STREAM подразумеваемым является протокол с именем IPPROTO_TCP, а для типа SOCK_DGRAM – IPPROTO_UDP; посредством неструктурированных сокетов (SOCK_RAW) можно воспользоваться протоколом ICMP, задав имя IPPROTO_ICMP, и т.д.

Сетевой и хостовый порядок байт

Преобразование значений типов `uint16_t` и `uint32_t` из хостового порядка байт в сетевой выполняется посредством функций `htons` и `htonl`, а функции `ntohs` и `ntohl` осуществляют обратную операцию.

```
1 #include <arpa/inet.h>
2 uint32_t htonl (uint32_t hostlong);
3 uint16_t htons (uint16_t hostshort);
4 uint32_t ntohl (uint32_t netlong);
5 uint16_t ntohs (uint16_t netshort);
```

Получение имен хостов

Структура hostent определена в файле `netdb.h`:

```
1  #include <netdb.h>
2  struct hostent {
3    char  *h_name;          /* official name of host */
4    char **h_aliases;       /* alias list */
5    int    h_addrtype;      /* host address type */
6    int    h_length;        /* length of address */
7    char **h_addr_list;     /* list of addresses */
8  }
9  #define h_addr h_addr_list[0] /* for backward compatibility */
```

- h_name – официальное имя хоста;
- h_aliases – массив псевдонимов хоста;
- h_addrtype – тип адреса (на данный момент `AF_INET` или `AF_INET6`;
- h_length – длина адреса в байтах;
- h_addr_list – массив указателей на сетевые адреса хоста;

Получение имен хостов

Доступ к сетевой базе имен хостов осуществляется с помощью вызовов:

```
1  #include <netdb.h>
2  void sethostent (int stayopen);
3  struct hostent *gethostent (void);
4  void endhostent (void);
5
6  struct hostent *gethostbyname(const char *name);
7
8  #include <sys/socket.h>       /* for AF_INET */
9  struct hostent *gethostbyaddr(const void *addr,
10 socklen_t len, int type);
```

Получение имен хостов

- `sethostent` – устанавливает соединение с базой, остающееся открытым после вызова `gethostent`, если значение аргумента `stayopen` отлично от нуля.
- `gethostent` – последовательно читает элементы базы, возвращая результат в структуре типа `hostent`.
- `endhostent` – закрывает соединение с базой.
- `gethostbyname` – возвращает результат в структуре типа `hostent` для заданного имени хоста.
- `gethostbyaddr` – возвращает результат в структуре типа `hostent` для заданного в структуре `addr`.

Пример содержимого файла /etc/hosts

```
127.0.0.1       alien.home      localhost.localdomain   localhost
::1     alien.home      localhost6.localdomain6 localhost6
```

Получение списка хостов из локальной БД

```
1   #include <stdio.h>
2   #include <netdb.h>
3
4   int main (void) {
5    struct hostent *pht;
6    char *pct;
7    int i, j;
8
9    sethostent (1);
10
11   while ((pht = gethostent ()) != NULL) {
12     printf ("Официальное имя хоста: %s\n", pht->h_name);
13     printf ("Альтернативные имена:\n");
14     for (i = 0; (pct = pht->h_aliases [i]) != NULL; i++) {
15       printf ("  %s\n", pct);
16     }
17     printf ("Тип адреса хоста: %d\n", pht->h_addrtype);
18     printf ("Длина адреса хоста: %d\n", pht->h_length);
19     printf ("Сетевые адреса хоста:\n");
20     for (i = 0; (pct = pht->h_addr_list [i]) != NULL; i++) {
21       for (j = 0; j < pht->h_length; j++) {
22     printf (" %d", (unsigned char) pct [j]);
23       }
24       printf ("\n");
25     }
26   }
27
28   endhostent ();
29
30   return 0;
```

Результат работы программы

```
Официальное имя хоста: alien.home
Альтернативные имена:
  localhost.localdomain
  localhost
Тип адреса хоста: 2
Длина адреса хоста: 4
Сетевые адреса хоста:
 127 0 0 1
Официальное имя хоста: alien.home
Альтернативные имена:
  localhost6.localdomain6
  localhost6
Тип адреса хоста: 2
Длина адреса хоста: 4
Сетевые адреса хоста:
  127 0 0 1
```

Произвольный доступ

Произвольный доступ по ключам — именам и адресам хостов с помощью функций `gethostbyname` и `gethostbyaddr` считаются устаревшим и эти функции могут быть исключены из новой версии стандарта POSIX. Вместо них необходимо использовать функции `getnameinfo` и `getaddrinfo()`.

Произвольный доступ

```
1  #include <sys/socket.h>
2  #include <netdb.h>
3
4  void freeaddrinfo (struct addrinfo *ai);
5
6  int getaddrinfo
7    (const char *restrict nodename,
8    const char *restrict servname,
9    const struct addrinfo *restrict hints,
10   struct addrinfo **restrict res);
11
12 int getnameinfo
13   (const struct sockaddr *restrict sa,
14   socklen_t salen, char *restrict node,
15   socklen_t nodelen, char *restrict service,
16   socklen_t servicelen, int flags);
```

Произвольный доступ

Аргумент `hints` позволяет передать дополнительную информацию об опрашиваемом сервисе - адресное семейство, тип сокета, протокол, флаги. Согласно стандарту, структура `addrinfo`, описанная в заголовочном файле `<netdb.h>`, должна содержать по крайней мере следующие поля:

```
int              ai_flags; /* Входные флаги */
int              ai_family; /* Адресное семейство сокета */
int              ai_socktype; /* Тип сокета */
int              ai_protocol; /* Протокол сокета */
socklen_t        ai_addrlen; /* Длина адреса сокета */
struct sockaddr *ai_addr; /* Адрес сокета */
char            *ai_canonname; /* Официальное имя узла сети */
struct addrinfo *ai_next; /* Указатель на следующий элемент списка */
```

Пример: доступ к произвольным именам хостов с помощью getaddrinfo

```
1   #include <stdio.h>
2   #include <netdb.h>
3   #include <arpa/inet.h>
4
5   int main (void) {
6       struct addrinfo hints = {AI_CANONNAME, AF_INET, SOCK_STREAM, IPPROTO_TCP,0,
            NULL, NULL, NULL};
7       struct addrinfo *addr_res;
8       int res=0;
9
10     if (res=getaddrinfo ("google.by", "http", &hints, &addr_res) != 0) {
11        perror ("GETADDRINFO");
12        printf ("GETADDRINFO: %s\n", gai_strerror(res));
13     } else {
14           printf ("Результаты для сервиса http\n");
15        /* Пройдем по списку возвращенных структур */
16           do {
17           printf ("Адрес сокета: Порт: %d IPадрес—: %s\n",
18             ntohs (((struct sockaddr_in *) addr_res->ai_addr)->sin_port),
19          inet_ntoa (((struct sockaddr_in *) addr_res->ai_addr)->sin_addr));
20       printf ("Официальное имя хоста: %s\n", addr_res->ai_canonname);
21        } while ((addr_res = addr_res->ai_next) != NULL);
22     }
23
24       return 0;
25  }
```

Результат работы программы

```
Результаты для сервиса http
Адрес сокета: Порт: 80 IP-адрес: 209.85.148.105
Официальное имя хоста: fra07s07-in-f105.1e100.net
Адрес сокета: Порт: 80 IP-адрес: 209.85.148.106
Официальное имя хоста: (null)
Адрес сокета: Порт: 80 IP-адрес: 209.85.148.147
Официальное имя хоста: (null)
Адрес сокета: Порт: 80 IP-адрес: 209.85.148.99
Официальное имя хоста: (null)
Адрес сокета: Порт: 80 IP-адрес: 209.85.148.103
Официальное имя хоста: (null)
Адрес сокета: Порт: 80 IP-адрес: 209.85.148.104
Официальное имя хоста: (null)
```

Отображение имен

Для преобразования IP-адресов из текстового представления в числовое и наоборот можно использовать функции:

```
1  #include <arpa/inet.h>
2  in_addr_t inet_addr (const char *cp);
3  char *inet_ntoa (struct in_addr in);
4  int inet_pton (int af, const char
5    *restrict src, void *restrict dst);
6  const char *inet_ntop (int af, const void
7    *restrict src, char *restrict dst,
8    socklen_t size);
```

Доступ к базе данных сетевых протоколов

Локальная база данных сетевых протоколов располагается по адресу `/etc/protocols`, а содержимое этого файла выглядит следующим образом:

```
ip 0 IP # internet protocol, pseudo protocol number
icmp 1 ICMP # internet control message protocol
igmp 2 IGMP # internet group management protocol
tcp 6 TCP # transmission control protocol
udp 17 UDP # user datagram protocol
```

Структура protoent

Структура protoent содержит по крайней мере следующие поля:

- char *p_name – официальное имя протокола;
- char **p_aliases – массив указателей на альтернативные имена протокола, завершаемый пустым указателем;
- int p_proto – номер протокола.

Функции для работы

Доступ к базе данных сетевых протоколов осуществляется с помощью следующих вызовов:

```
1  #include <netdb.h тест>
2  void setprotoent (int stayopen);
3  struct protoent *getprotoent (void);
4  struct protoent *getprotobyname
5            (const char *name);
6  struct protoent *getprotobynumber
7  (int proto);
8  void endprotoent (void);
```

```c
#include <stdio.h>
#include <netdb.h>

int main (void) {
 struct protoent *pht;
 char *pct;
 int i;

 setprotoent (1);

 while ((pht = getprotoent ()) != NULL) {
   printf ("Официальное имя протокола: %s\n", pht->p_name);
   printf ("Альтернативные имена:\n");
   for (i = 0; (pct = pht->p_aliases [i]) != NULL; i++) {
     printf ("  %s\n", pct);
   }
   printf ("Номер протокола: %d\n\n", pht->p_proto);
 }

 if ((pht = getprotobyname ("ipv6")) != NULL)
   printf ("Номер протокола ipv6: %d\n\n", pht->p_proto);
 else fprintf (stderr, "Протокол ip в базе не найден\n");

 if ((pht = getprotobyname ("IPV6")) != NULL)
   printf ("Номер протокола IPV6: %d\n\n", pht->p_proto);
 else fprintf (stderr, "Протокол IPV6 в базе не найден\n");

 endprotoent ();

 return 0;
}
```

Результат работы программы:

```
Официальное имя протокола: ip
Альтернативные имена:
  IP
Номер протокола: 0

Официальное имя протокола: icmp
Альтернативные имена:
  ICMP
Номер протокола: 1

Официальное имя протокола: igmp
Альтернативные имена:
  IGMP
Номер протокола: 2
```

Доступ к базе данных сетевых сервисов

Локально база данных сетевых сервисов располагается по адресу /etc/services, а содержимое этого файла выглядит следующим образом:

```
# service-name  port/protocol  [aliases \ldots]   [# comment]
domain 53/tcp # Domain Name Server
domain 53/udp # Domain Name Server
bootps 67/tcp # Bootstrap Protocol Server
bootps 67/udp # Bootstrap Protocol Server
bootpc 68/tcp # Bootstrap Protocol Client
bootpc 68/udp # Bootstrap Protocol Client
```

Структура servent

Структура servent содержит следующие поля:

- char *s_name – Официальное имя сервиса
- char **s_aliases – Массив указателей на альтернативные имена сервиса, завершаемый пустым указателем
- int s_port – Номер порта, соответствующий сервису (в сетевом порядке байт)
- char *s_proto – Имя протокола для взаимодействия с сервисом

Функции

Доступ к базе данных сетевых сервисов осуществляется с помощью следующих вызовов:

```
1  #include <netdb.h>
2  void setservent (int stayopen);
3  struct servent *getservent (void);
4  struct servent *getservbyname
5      (const char *name, const char *proto);
6  struct servent *getservbyport
7      (int port, const char *proto);
8  void endservent (void);
```

Пример (начало)

```c
#include <stdio.h>
#include <netdb.h>

int main (void) {
 struct servent *pht;
 char *pct;
 int i;

 setservent (1);

 while ((pht = getservent ()) != NULL) {
   printf ("Официальное имя сервиса: %s\n", pht->s_name);
   printf ("Альтернативные имена:\n");
   for (i = 0; (pct = pht->s_aliases [i]) != NULL; i++) {
     printf ("  %s\n", pct);
   }
   printf ("Номер порта: %d\n", ntohs ((in_port_t) pht->s_port));
   printf ("Имя протокола: %s\n\n", pht->s_proto);
 }
```

Пример (продолжение)

```
1   ОфициальноеимясервисаАльтернативныеименаНомерпортаИмяпротокола
2    if ((pht = getservbyport (htons ((in_port_t) 21), "udp")) != NULL) {
3      printf ("Официальное имя сервиса: %s\n", pht->s_name);
4      printf ("Альтернативные имена:\n");
5      for (i = 0; (pct = pht->s_aliases [i]) != NULL; i++) {
6        printf ("  %s\n", pct);
7      }
8      printf ("Номер порта: %d\n", ntohs ((in_port_t) pht->s_port));
9      printf ("Имя протокола: %s\n\n", pht->s_proto);
10   } else {
11     perror ("GETSERVBYPORT");
12   }
13
14   if ((pht = getservbyport (htons ((in_port_t) 21), (char *) NULL)) != NULL) {
15     printf ("Официальное имя сервиса: %s\n", pht->s_name);
16     printf ("Альтернативные имена:\n");
17     for (i = 0; (pct = pht->s_aliases [i]) != NULL; i++) {
18       printf ("  %s\n", pct);
19     }
20     printf ("Номер порта: %d\n", ntohs ((in_port_t) pht->s_port));
21     printf ("Имя протокола: %s\n\n", pht->s_proto);
22   } else {
23     perror ("GETSERVBYPORT");
24   }
25
26   endservent ();
27
28   return 0;
29  }
```

Результат работы программы

```
Официальное имя сервиса: http
Альтернативные имена:
  www
  www-http
Номер порта: 80
Имя протокола: tcp

Официальное имя сервиса: http
Альтернативные имена:
  www
  www-http
Номер порта: 80
Имя протокола: udp

Официальное имя сервиса: kerberos
Альтернативные имена:
  kerberos5
  krb5
Номер порта: 88
Имя протокола: tcp
```

Спасибо за внимание!
Вопросы?