Сергей Юрьевич Шилов

Системное программирование для современных платформ

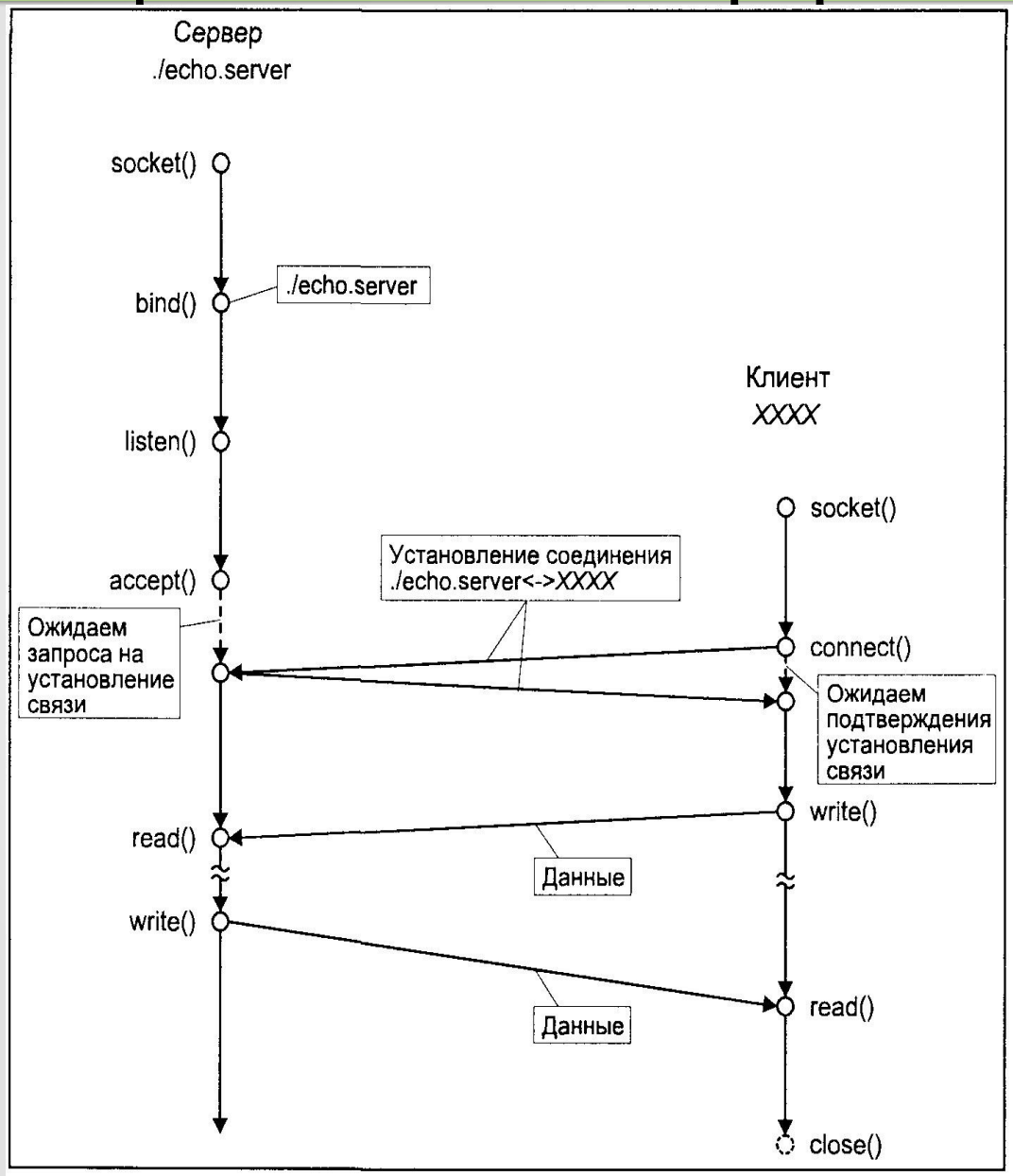
Системное программирование для современных платформ

6. Сокеты

Системное программирование для современных платформ

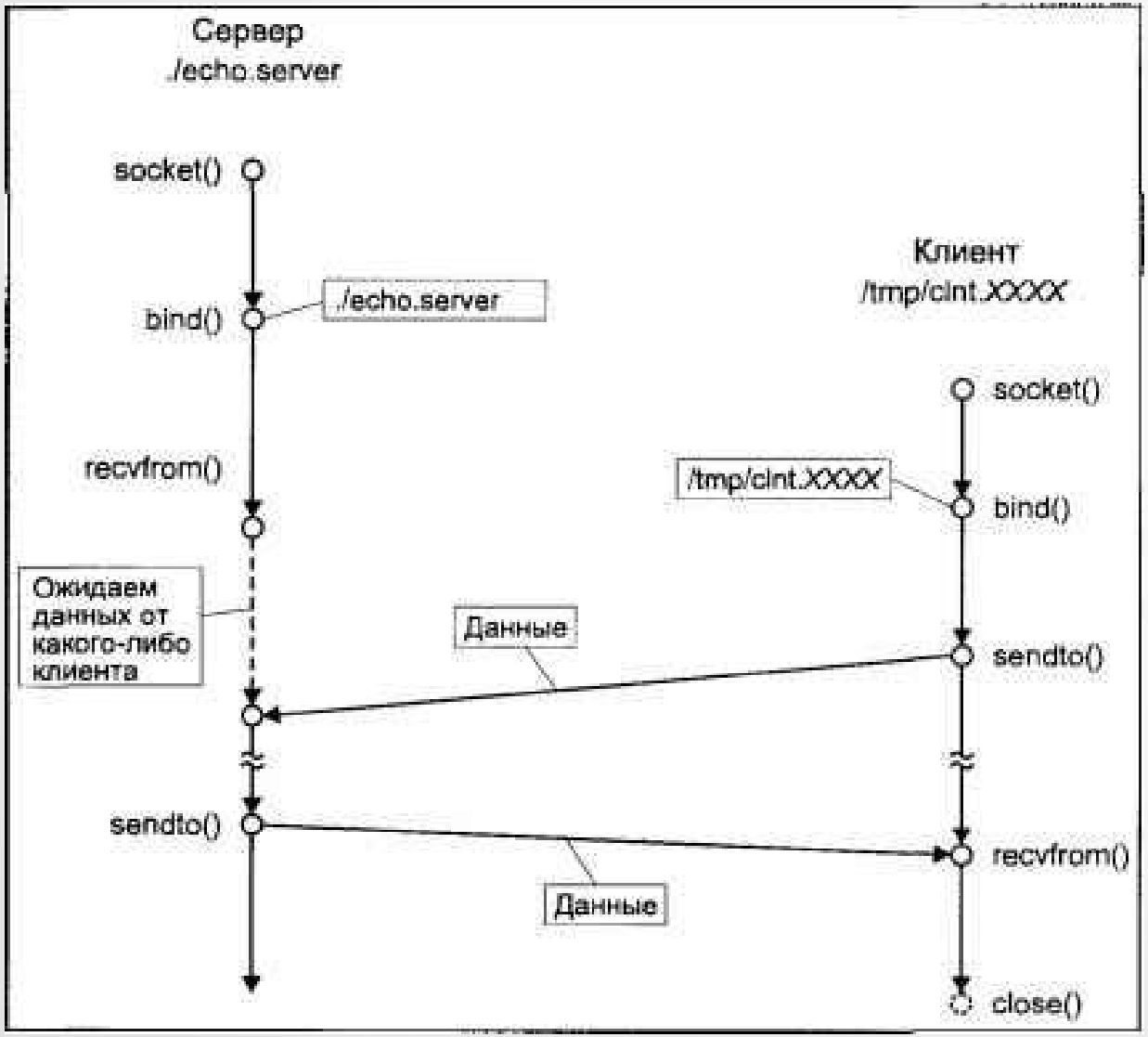
- программный интерфейс, обеспечивающий передачу данных между процессами, запущенными на одном (AF_UNIX) или разных хостах (AF_INET)
- удалённые сокеты используют протоколы TCP (надежный) или UDP (ненадёжный), возможно использование других транспортных протоколов
- данные передаются as-is
- интерфейс сокетов реализует сеансовый уровень модели ISO/OSI
- разница между файлами и сокетами в том, что дескриптор файла указывает на существующий файл, который является точкой назначения (endpoint), сокет же сначала создаётся, а затем связывается с соответствующей endpoint

Системное программирование для современных платформ



TCP – сокеты

Системное программирование для современных платформ



UDP – сокеты

Системное программирование для современных платформ

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

ОПИСАНИЕ

socket создает конечную точку соединения и возвращает ее дескриптор.
Параметр **domain** задает домен соединения: выбирает набор протоколов, которые будут использоваться для создания соединения. Такие наборы описаны в **<sys/socket.h>**. В частности, поддерживаются следующие форматы:

Название	Назначение	Страница руководства
PF_UNIX, PF_LOCAL	локальное соединение	**unix(7)**
PF_INET IPv4	протоколы Интернет	**ip(7)**

Сокет имеет тип **type**, задающий семантику коммуникации. В частности, определены следующие типы:

SOCK_STREAM - Обеспечивает создание двусторонних надежных и последовательных потоков байтов, поддерживающих соединения. Может также поддерживаться механизм передачи внепоточных данных.
SOCK_DGRAM - Поддерживает датаграммы (ненадежные сообщения с ограниченной длиной и не поддерживающие соединения).
SOCK_RAW - Обеспечивает доступ к низкоуровневому сетевому протоколу.
Некоторые типы сокетов могут быть не включены в определенные наборы протоколов

Системное программирование для современных платформ

Параметр **protocol** задаёт конкретный протокол, который работает с сокетом. Обычно существует только один протокол, задающий конкретный тип сокета в определённом семействе протоколов, в этом случае **protocol** может быть определено, как **0**. Однако, возможно существование нескольких таких протоколов (в этом случае и используется данный параметр). Номер протокола зависит от используемого коммуникационного домена, см. protocols(5). См. тж. getprotoent(3), где описано, как соотносить имена протоколов с их номерами. В связи с тем, что для разных платформ соответствие между коммуникационным доменом и набором поддерживающих протоколов может оказаться различным, для написания переносимых программ этот параметр рекомендуется указывать явно.

Сокеты типа **SOCK_STREAM** являются соединениями полнодуплексных байтовых потоков, похожими на каналы. Они не сохраняют границы записей. Потоковый сокет должен быть в состоянии соединения перед тем, как из него можно будет отсылать данные или принимать их в нем. Соединение с другим сокетом создаётся с помощью системного вызова **connect(2)**. После соединения данные можно передавать, с помощью системных вызовов **read(2)**, **write(2)** или одного из вариантов следующих системных вызовов: **send(2)**, **recv(2)**. Когда сеанс закончен, выполняется команда **close(2)**. Внепоточные данные могут передаваться, как описано в **send(2)**, а приниматься, как описано в **recv(2)**.

Коммуникационные протоколы, которые реализуют **SOCK_STREAM**, следят, чтобы данные не были потеряны или дублированы. Если часть данных, для которых имеется место в буфере протокола, не может быть передана за определённое время, соединение считается "мёртвым". Когда в сокете включён флаг **SO_KEEPALIVE**, протокол каким-либо способом проверяет, не отключена ли ещё другая сторона. Сигнал **SIGPIPE** появляется, если процесс посылает или принимает данные, пользуясь "разорванным" потоком; это приводит к тому, что процессы, не обрабатывающие сигнал, завершаются.

Сокеты **SOCK_DGRAM** и **SOCK_RAW** позволяют посылать датаграммы принимающей стороне, заданной при вызове **send(2)**. Датаграммы обычно принимаются с помощью вызова **recvfrom(2)**, который возвращает следующую датаграмму с соответствующим обратным адресом.

Системное программирование для современных платформ

Системный вызов **fcntl(2)** с аргументом **F_SETOWN** может использоваться для задания группы процессов, которая будет получать сигнал **SIGURG**, когда прибывают внепоточные данные; или сигнал **SIGPIPE**, когда соединение типа **SOCK_STREAM** неожиданно обрывается. Этот вызов также можно использовать, чтобы задать процесс или группу процессов, которые получают асинхронные уведомления о событиях ввода-вывода с помощью **SIGIO**.

Когда сеть сообщает модулю протокола об ошибке (например, в случае **IP**, используя **ICMP-**сообщение), то для сокета устанавливается флаг ожидающей ошибки. Следующая операция этого сокета вернёт код ожидающей ошибки. Некоторые протоколы поддерживают очереди ошибок в сокете и позволяют получить детальную информацию об ошибке; см. **IP_RECVERR** в **ip(7)**.

Операции сокетов контролируются их параметрами. Эти параметры описаны в **<sys/socket.h>**. Функции **setsockopt(2)** и **getsockopt(2)** используются, чтобы установить и получить необходимые параметры соответственно.

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ

В случае ошибки возвращается **-1**; в противном случае возвращается дескриптор сокета.

Системное программирование для современных платформ

тип	название	протокол	PF_UNIX	PF_INET
Сокет датаграмм	SOCK_DGRAM	IPPROTO_UDP	+	+
Сокет потока	SOCK_STREAM	IPPROTO_TCP	+	+
Сокет низкого уровня	SOCK_RAW	IPPROTO_RAW, IPPROTO_ICMP	-	+

Системное программирование для современных платформ

Примеры:

```
tcp_socket = socket(AF_INET, SOCK_STREAM, 0);
udp_socket = socket(AF_INET, SOCK_DGRAM, 0);
raw_socket = socket(AF_INET, SOCK_RAW, protocol);

unix_socket = socket(AF_UNIX, type, 0);
```

Системное программирование для современных платформ

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

ОПИСАНИЕ

bind привязывает к сокету, управляемому дескриптором **sd**, локальный адрес **my_addr** длиной **addrlen**. Традиционно, эта операция называется «присваивание сокету имени». Когда сокет только что создан с помощью **socket(2)**, он существует в пространстве имён (семействе адресов), но не имеет назначенного имени.
Обычно сокету типа **SOCK_STREAM** требуется назначить локальный адрес с помощью **bind**, перед тем, как он сможет принимать соединения (см. **accept(2)**).

Правила, используемые при привязке имён, разные в разных семействах адресов. Обратитесь к соответствующему руководству в секции 7 за дальнейшей информацией. Для **AF_INET** смотри **ip(7)**, для **AF_UNIX** смотри **unix(7)**.

```
struct sockaddr_in {
     sa_family_t    sin_family; /* address family: AF_INET */
    in_port_t      sin_port;   /* port in network byte order */
    struct in_addr sin_addr;   /* internet address in network byte order*/ };

struct sockaddr_un {
     sa_family_t sun_family;               /* AF_UNIX */
     char sun_path[UNIX_PATH_MAX=108];  /* pathname */  };
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

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

Системное программирование для современных платформ

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

ОПИСАНИЕ

Для открытого и связанного сокета **listen** выражает готовность принимать входящие соединения и задаёт размер очереди. Системный вызов **listen** применим только к сокетам типа **SOCK_STREAM** или **SOCK_SEQPACKET**.
Параметр **backlog** задает максимальную длину, до которой может расти очередь ожидающих соединений. Если приходит запрос на соединение, а очередь полна, то клиент получит ошибку **ECONNREFUSED** или, если соответствующие протокол поддерживает повторную передачу, запрос может быть игнорирован, чтобы попытаться ответить на повторный запрос.

ЗАМЕЧАНИЯ

Параметр **backlog** на **TCP-сокетах** задает размер очереди для полностью установленных соединений, ожидающих, пока процесс примет их. Максимальный размер очереди для неоконченных сокетов может быть задан через переменную ядра **tcp_max_syn_backlog**.
Когда разрешено использование **syncookies**, логическая максимальная длина отсутствует и настройка этого **sysctl** игнорируется. См. **tcp(7)** за дальнейшей информацией.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

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

Системное программирование для современных платформ

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

ОПИСАНИЕ

Функция **accept** используется с сокетами, ориентированными на устанавление соединения (**SOCK_STREAM**). Эта функция извлекает первый запрос на соединение из очереди ожидающих соединений, создаёт новый подключенный сокет почти с такими же параметрами, что и у **sd**, и выделяет для сокета новый дескриптор, который и возвращается. Новый сокет более не находится в слушающем состоянии. Исходный сокет **sd** не изменяется при этом вызове. Флаги дескрипторов которые можно установить с помощью параметра **F_SETFL** функции **fcntl**, типа неблокированного состояния или асинхронного ввода-вывода не наследуются новым файловым дескриптором после **accept**.
Аргумент **sd** - это сокет, который был создан с помощью **socket(2)**, привязан к локальному адресу с помощью **bind(2)**, и слушает соединения после **listen(2)**.

Аргумент **addr** - указатель на структуру **sockaddr**. В эту структуру помещается адрес другой стороны, в том виде, в каком он известен на коммуникационном уровне. Точный формат адреса, передаваемого в параметре **addr**, определяется "семейством" сокета. Аргумент **addrlen** является параметром, передаваемым по ссылке: перед вызовом он содержит размер структуры, на которую ссылается **addr**, а после вызова - действительную длину адреса в байтах. Если **addr** равен **NULL**, он не заполняется.

Если в очереди нет запросов на соединение, и на сокет не установлен флаг, что он является неблокирующим, **accept** блокирует вызвавшую программу до появления соединения. Если сокет является неблокирующим, а в очереди нет запросов на соединение, то **accept** возвращает **EAGAIN**.

Для того, чтобы получать уведомления о входящих сооединениях на сокете, можно использовать **select(2)** или **poll(2)**. В этом случае, когда придёт запрос на новое соединение, будет доставлено событие "можно читать", и после этого вы можете вызвать **accept**, чтобы получить сокет для этого соединения. Можно также настроить сокет так, чтобы он посылал сигнал **SIGIO**, когда на нём происходит какая-либо активность; см. **socket(7)**, где описаны детали.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

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

Системное программирование для современных платформ

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

ОПИСАНИЕ

Функция **connect** инициирует клиентское соединение через сокет, на который указывает дескриптор **sd**. Если сокет имеет тип **SOCK_DGRAM**, адрес **serv_addr** является адресом по умолчанию, куда посылаются датаграммы, и единственным адресом, откуда они принимаются. Если сокет имеет тип **SOCK_STREAM** или **SOCK_SEQPACKET**, то данный системный вызов попытается установить соединение с другим сокетом. Другой сокет задан параметром **serv_addr**, являющийся адресом длиной **addrelen** в пространстве коммуникации сокета. Каждое пространство коммуникации интерпретирует параметр **serv_addr** по-своему. Обычно сокеты с протоколами, основанными на соединении, могут устанавливать соединение только один раз; сокеты с протоколами без соединения могут использовать **connect** многократно, чтобы изменить адрес назначения. Сокеты без поддержки соединения могут прекратить связь с другим сокетом, установив член **sa_family** структуры **sockaddr** в **AF_UNSPEC**.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

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

Системное программирование для современных платформ

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

ssize_t sendto(int s, const void *msg, size_t len, int flags, const struct sockaddr *to, socklen_t
tolen);
```

ОПИСАНИЕ

sendto используется для пересылки сообщений в другой сокет. Адрес получателя задаётся параметром **to** длиной **tolen**. Длина сообщения задаётся параметром **len**. Если сообщение слишком длинное, чтобы быть отосланным протоколом нижнего уровня, возвращается ошибка **EMSGSIZE**, а сообщение не отсылается. Неудачная отправка не показывается с помощью **send**. Локальные ошибки принимают значение **-1**. Когда сообщение не помещается в буфер отправки сокета, **send** обычно дожидается завершения отправки, если только сокет не находится в неблокирующем режиме. Если сокет находится в неблокирующем режиме, то в этом случае возвращается **EAGAIN**. Системный вызов **select(2)** можно использовать для выяснения, возможно ли отправлять данные.

Параметр **flags** является битовой маской и может содержать такие флаги:

MSG_OOB - Посылает внепотоковые данные, если сокет это поддерживает (как, например, сокеты типа **SOCK_STREAM**); протокол более низкого уровня также должен поддерживать внепотоковые данные.
MSG_DONTROUTE - Не использовать маршрутизацию при отправке пакета, а посылать его только на хосты в локальной сети. Обычно это используется в диагностических программах и программах маршрутизации. Этот флаг определён только для маршрутизируемых семейств протоколов; пакетные сокеты не используют маршрутизацию.
MSG_DONTWAIT - Включает режим **non-blocking**; если операция должна была заблокировать программу, возвращается **EAGAIN** (этот режим также можно задать с помощью опции **O_NONBLOCK**, команды **F_SETFL** и функции **fcntl(2)**).
MSG_NOSIGNAL - Требует не посылать сигнал **SIGPIPE**, если при работе с ориентированным на поток сокетом другая сторона обрывает соединение. Код ошибки **EPIPE** возвращается в любом случае.

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

Системное программирование для современных платформ

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

int recvfrom(int s, void *buf, size_t len, int flags, struct sockaddr *from,
socklen_t *fromlen);
```

ОПИСАНИЕ

Системный вызов **recvfrom** используются для получения сообщений из сокета, и могут использоваться для получения данных, независимо от того, является ли сокет ориентированным на соединения или нет. Если параметр **from** не равен **NULL**, а сокет не является ориентированным на соединения, то адрес отправителя в сообщении не заполняется. Аргумент **fromlen** передается по ссылке, в начале инициализируется размером буфера, связанного с **from**, а при возврате из функции содержит действительный размер адреса.

Функция возвращает длину сообщения при успешном завершении. Если сообщение слишком длинное и не поместилось в предоставленный буфер, лишние байты могут быть отброшены, в зависимости от типа сокета, на котором принимаются сообщения (см. **socket(2)**).

Если на сокете не доступно ни одного сообщения, то обсуждаемые функции ожидают их прибытия, если сокет не помечен как неблокирующий (см. **fcntl(2)**), в противном случае возвращается значение **-1**, а внешняя переменная **errno** устанавливается в значение **EAGAIN**. Все эти функции обычно возвращают уже доступные данные вплоть до запрошенного объема, и не ждут, пока появятся данные полной запрошенной длины.

Системные вызовы **select(2)** или **poll(2)** можно использовать для определения появления новых данных.

Системное программирование для современных платформ

Аргумент **flags** системного вызова **recv** формируется конкатенацией одного или более нижеследующих значений:

MSG_OOB - Этот флаг запрашивает приём внепотоковых данных, которые в противном случае не были бы получены в обычном потоке данных. Некоторые протоколы помещают данные повышенной срочности в начало обычной очереди данных, и поэтому этот флаг не может использоваться с такими протоколами.
MSG_PEEK - Этот флаг заставляет выбрать данные из начала очереди, но не удалять их оттуда. Таким образом, последующий вызов функции вернёт те же самые данные.
MSG_WAITALL - Этот флаг просит подождать, пока не придёт полное запрошенное количество данных. Однако, этот вызов все равно может вернуть меньше данных, чем было запрошено, если был пойман сигнал, произошла ошибка или разрыв соединения, или если начали поступать данные другого типа, не того, который был сначала.
MSG_TRUNC - Возвращает реальную длину пакета, даже если она была больше, чем предоставленный буфер. Этот флаг можно использовать только с пакетными протоколами.

Системное программирование для современных платформ

MSG_ERRQUEUE - Получить пакет из очереди ошибок.
MSG_NOSIGNAL - Этот флаг отключает возникновение сигнала **SIGPIPE** на потоковых сокетах, если другая сторона вдруг исчезает.
MSG_ERRQUEUE - Указание этого флага позволяет получить из очереди ошибок сокета накопившиеся ошибки. Каждая ошибка передаётся во вспомогательном сообщении, чей тип зависит от протокола (для **IPv4** этим типом является **IP_RECVERR**). Пользователь должен предоставить буфер достаточной длины. См. **cmsg(3)** и **ip(7)**, где приведена дополнительная информация. Содержимое исходного пакета, который привел к ошибке, передаётся в виде обычных данных с помощью **msg_iovec**. Исходный адрес назначения датаграммы, которая вызвала ошибку, передаётся с помощью **msg_name**. Для локальных ошибок адрес не передаётся (это можно выяснить, проверив поле **cmsg_len** структуры **cmsghdr**). Для ошибок при приёме в **msghdr** устанавливается **MSG_ERRQUEUE**. После того, как ошибка передана программе, следующая ошибка в очереди ошибок становится ожидающей ошибкой и передаётся программе при следующей операции на сокете.
Ошибка хранится в структуре **sock_extended_err (см.)**.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

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

Системное программирование для современных платформ

```
/* A simple server in the internet domain using TCP. The port number is passed as an argument */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

void error(char *msg) {
    perror(msg);
    exit(1);
    }

int main(int argc, char *argv[]) {
     int sockfd, newsockfd, portno, clilen;
     char buffer[256];
     struct sockaddr_in serv_addr, cli_addr;
     int n;
     if (argc < 2) {
         fprintf(stderr,"ERROR, no port provided\n");
         exit(1); }
     if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) error("ERROR opening socket");
     bzero((char *) &serv_addr, sizeof(serv_addr));
     portno = atoi(argv[1]);
     serv_addr.sin_family = AF_INET;
     serv_addr.sin_addr.s_addr = INADDR_ANY;
     serv_addr.sin_port = htons(portno);
     if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) error("ERROR on binding");
     listen(sockfd,5);
     clilent = sizeof(cli_addr);
     if ((newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen)) < 0) error("ERROR on accept");
     bzero(buffer,256);
     if ((n = read(newsockfd,buffer,255);) < 0) error("ERROR reading from socket");
     printf("Here is the message: %s\n",buffer);
     if ((n = write(newsockfd,"I got your message",18)) < 0) error("ERROR writing to socket");
     return 0;
     }
```

Системное программирование для современных платформ

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

void error(char *msg) {
    perror(msg);
    exit(0);
    }

int main(int argc, char *argv[]) {
    int sockfd, portno, n;

    struct sockaddr_in serv_addr;
    struct hostent *server;

    char buffer[256];
    if (argc < 3) { fprintf(stderr,"usage %s hostname port\n", argv[0]); exit(0); }
    portno = atoi(argv[2]);
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) error("ERROR opening socket");
    server = gethostbyname(argv[1]);
    if (server == NULL) { fprintf(stderr,"ERROR, no such host\n"); exit(0); }
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr, (char *)&serv_addr.sin_addr.s_addr, server->h_length);
    serv_addr.sin_port = htons(portno);
    if (connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr)) < 0) error("ERROR connecting");
    printf("Please enter the message: ");
    bzero(buffer,256);
    fgets(buffer,255,stdin);
    if (write(sockfd,buffer,strlen(buffer) < 0) error("ERROR writing to socket");
    bzero(buffer,256);
    if (read(sockfd,buffer,255) < 0) error("ERROR reading from socket");
    printf("%s\n",buffer);
    return 0;
    }
```

Системное программирование для современных платформ

7. API каталогов

Системное программирование для современных платформ

```
#include <sys/dir.h>

struct DIR {
  long d_ino;
  unsigned short d_reclen; /*size of current instance of struvcture*/
  char d_name [MAXNAMLEN];
};

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

struct dirent {
  long d_ino;
  off_t d_off;
  unsigned short d_reclen; /*size of current instance of struvcture*/
  char d_name[];
};
```

Системное программирование для современных платформ

```
#include <sys/dir.h>

DIR *opendir(const char *name);
int closedir(DIR *dir);
```

ОПИСАНИЕ

Функция **opendir()** открывает структуру каталога, соответствующий каталогу **name**, и возвращает указатель на этот поток. Связанный со структурой маркер **dirent** будет указывать на первую запись в каталоге.
Функция **closedir()** закрывает структуру, связанную с каталогом **dir**.

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ

Функция **opendir()** возвращает указатель на поток каталога или NULL в случае ошибок.
Функция **closedir()** возвращает **0** в случае удачного завершения работы, или **-1** при ошибке.

Системное программирование для современных платформ

```
#include <sys/types.h>
#include <dirent.h>

struct dirent *readdir(DIR *dir);
```

ОПИСАНИЕ

Функция **readdir()** возвращает указатель на следующую запись каталога в структуре **dirent**, прочитанную из структуры каталога, на которую указывает **dir**. Функция возвращает **NULL** по достижении последней записи или если была обнаружена ошибка.
В соответствие с **POSIX**, структура **dirent** содержит поле **char d_name[]** неопределенной длины, с максимальным количеством символов, предшествующих конечному нулевому символу, равным **NAME_MAX**. Использование других полей отрицательно повлияет на переносимость ваших программ. В **POSIX-2001** как расширение **XSI** описано поле **ino_t d_ino**. Данные, возвращаемые **readdir()** могут быть переписаны последующими вызовами **readdir()** для того же каталожного потока.

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ

Функция **readdir()** возвращает указатель на структуру **dirent** или **NULL** в случае ошибки или по достижении последней записи. Переменная **errno** устанавливается в соответствующее значение. По этому значению можно определить точную причину возврата **NULL**.

Системное программирование для современных платформ

```
#include <sys/types.h>
#include <dirent.h>

void rewinddir(DIR *dir);
```

ОПИСАНИЕ

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

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ

Функция rewinddir() не возвращает значений.

Системное программирование для современных платформ

```
#include <dirent.h>

off_t telldir(DIR *dir);
void seekdir(DIR *dir, off_t offset);
```

ОПИСАНИЕ

Функция **telldir()** возвращает текущее положение в каталоге, задаваемом структурой **dir**. Не является **POSIX**-совместимой.
Функция **seekdir()** устанавливает в каталоге позицию, с которой начнет работу следующий вызов **readdir()**. Позиция **seekdir()** должна возвращаться вызовом **telldir()**. Не является **POSIX**-совместимой.

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ

Функция **telldir()** возвращает текущее положение в потоке каталога или **-1** при ошибках.
Функция **seekdir()** не возвращает значений.

Системное программирование для современных платформ

Спасибо за внимание!

:-)