Разработка программного обеспечения ОС UNIX

Удаленный вызов процедур.

Взаимодействие через сеть

Двери. Создание

```
#include <door.h>

int door_create( void (*server_procedure) (void *cookie, char *argp, size_t
    arg_size, door_desc_t *dp, uint_t n_desc), void *cookie, uint_t attributes);
```

Связывание с именем в файловой системе:

```
fd = door_create( serverproc, 0, 0 );
unlink( name );
close( open( name, O_CREAT|O_RDWR, mode );
fattach( fd, name );
close( fd );
```

Связывание с пулом потоков:

```
int door_bind(int did);
int door_unbind(void);
```

Вызов двери. Передача параметров

```
typedef struct {
    char *data_ptr;            /* Argument/result buf ptr*/
    size_t data_size;          /* Argument/result buf size */
    door_desc_t *desc_ptr; /* Argument/result descriptors */
    uint_t desc_num;           /* Argument/result num desc */
    char *rbuf;                /* Result buffer */
    size_t rsize;              /* Result buffer size */
} door_arg_t;

typedef struct {
    door_attr_t d_attributes; /* Describes the parameter */
    union {
        struct {
            int d_descriptor; /* Descriptor */
            door_id_t d_id; /* Unique door id */
        } d_desc;
    } d_data;
} door_desc_t;

int door_call(int d, door_arg_t *params);
```

Информация о двери

```
int door_cred(door_cred_t *info);

struct door_cred_t {
    uid_t dc_euid;          /* Effective uid of client */
    gid_t dc_egid;          /* Effective gid of client */
    uid_t dc_ruid;          /* Real uid of client */
    gid_t dc_rgid;          /* Real gid of client */
    pid_t dc_pid;           /* pid of client */
};

int door_info(int d, struct door_info *info);
struct door_info {
    pid_t       di_target;      /* door server pid */
    door_ptr_t  di_proc;        /* server function */
    door_ptr_t  di_data;        /* data cookie for invocation */
    door_attr_t di_attributes;  /* door attributes */
    door_id_t   di_uniquifier;  /* unique id among all doors */
};
```

Передача информации через дверь

В серверной функции return использовать нельзя!

Возврат из двери:

```
int door_return( char *data_ptr, size_t data_size, door_desc_t *desc_ptr,
                 uint_t num_desc);
```

Отключает доступ через дескриптор:

```
int door_revoke(int d);
```

Пример использования. Клиент

```
#include <door.h>

int fd;
long ival, oval;
door_arg_t arg;
fd = open(“door_name.door”, O_RDWR);
ival = …
arg.data_ptr = (char*)&ival;
arg.data_size = sizeof( ival );
arg.desc_ptr = NULL;
arg.desc_num = 0;
arg.rbuf = (char*)&oval;
arg.rsize = sizeof( oval );
door_call( fd, &arg );
```

Пример использования. Сервер

```
void server_procexample( void *cookie, char *argp, size_t arg_size,
                                                    door_desc_t *dp, uint_t n_desc) {
    long arg, result;
    arg = *((long*)argp);
    result = arg | 0xA5A5A5A5;
    door_return( (char*)&result, sizeof(result), NULL, 0 );
}


…
    int fd;
    fd = door_create( server_procexample, NULL, 0 );
    unlink( “door_name.door” );
    close( open(“door_name.door”, O_CREAT|O_RDWR, 0660 );
    fattach( fd, “door_name.door” );
    for (;;);
    pause();
…
```

RPC – вызов удалённых процедур

#include <rpc/rpc.h>
#include <netconfig.h> Библитека libnsl

Вызов функции из адресного пространства процесса другой машины!

«Подводные камни»:

- Передача данных по сети.
- Идентификация.
- Типы данных.
- Особенности архитектуры (порядок байт).

Упрощение процедуры вызова:

- IDL – intermediate data language.
- rpcgen – компилятор с языка IDL.
- XDR – external data representation.

Пример вызова функции

---------- Исходная функция:

```
int calcul( int a, int b ) {
        return a | (b << 4);
}
```

---------- Файл IDL calcul.x

```
program CALCULPROG
{
        version CALCULVER
        {
                int CALCUL( int, int ) = 1;
        } = 1;
} = 0x20023450;
```

---------- Используем rpcgen:

```
rpcgen –C calcul.x
gcc –c client.c;        gcc –c calcul_clnt.c;        gcc –c calcul_xdr.c
gcc –o client client.o calcul_clnt.o calcul_xdr.o -lnsl
```

Серверная и клиентская части

```
#include “calcul.h”
int* calcul_1( int* a. int* b, struct svc_req *rqstp ) {
        static result = *a | (*b << 4);
        return &result;
}

#include “calcul.h”
int main( int argc, char ** argv) {
CLIENT *cl;
int     a = 10;
int     b = 20;
int     *r;
cl = clnt_create( argv[1], CALCULPROG, CALCULVER, “tcp” );
if ( ! (r = calcul_1( &a, &b, cl ))) {
        clnt_perror( cl, argv[1] );
} else {
        printf(“r == %d\n”, *r );
```

XDR-преобразования

Типы данных:

- bool
- char
- int
- long
- hyper == long long
- float
- double
- quadruple
- enum
- string var<n>
- opaque var[n]
- opaque var<n>
- struct
- union

Если используется структура преобразовывается каждое поле:

xdr_u_int(…
xdr_vector(…
xdr_pointer(…
xdr_double(…

Сериализация

В файле data.x

```
struct data {
        short     shortdata;
        string    vs<128>;          //строка переменной длины
        short     vsh[32];          //массив
};
```

rpcgen –C data.x → data.h, data.c (функция xdr_data(…)).

Запись в файл

```
XDR xhandle, data out;
char* buff = malloc(BUFFSIZE);
xdrmem_create( &xhandle, buff, BUFFSIZE, XDR_ENCODE );
        //создали поток в памяти
if ( xdr_data( &xhandle, &out ) != TRUE ) …
size = xdr_getpos( &xhandle ); //Сколько места заняли
write( STDOUT_FILENO, buff, size );
```

… и десериализация

Чтение из файла

```
XDR xhandle;
data in;
int n;
char* buff = malloc(BUFFSIZE);
n = read( STDIN_FILENO, buff, BUFFSIZE );
xdrmem_create( &xhandle, buff, n, XDR_DECODE );
memset( &in, 0, sizeof(in) );
if ( xdr_data( &xhandle, &in ) != TRUE ) …
```

xdr_free(…) – использовать для освобождения памяти

Вычисление размера занимаемой памяти

RNDUP:

```
size = RNDUP( sizeof(s.a) ) + RNDUP( sizeof(s.b) ) + RNDUP( sizeof(s.c) );
```

s – структура. (обычно выравнивание по гр. 4).

API RPC – создание клиента

```
#include <rpc/rpc.h>

bool_t clnt_control(CLIENT *clnt, const uint_t req, char *info);

CLIENT *clnt_create(const char *host, const rpcprog_t prognum, const rpcvers_t
    versnum, const char *nettype);

CLIENT *clnt_create_vers (const char *host,rpcvers_t *const rpcprog_t
    prognum,vers_outp,const rpcvers_t vers_low,const rpcvers_t vers_high, char
    *nettype);

CLIENT *clnt_dg_create(const int fildes,const struct netbuf *svcaddr, const rpcprog_t
    prognum,const rpcvers_t versnum, const uint_t sendsz,const uint_t recsz);

void clnt_pcreateerror(const char *s);

CLIENT *clnt_raw_create(const rpcprog_t prognum,const rpcvers_t versnum);
char *clnt_spcreateerror(const char *s);

CLIENT *clnt_tli_create(const int fildes,const struct netconfig *netconf, const struct netbuf
    *svcaddr,
const rpcprog_t prognum, const rpcvers_t versnum,const uint_t sendsz, const uint_t
    recsz);

CLIENT *clnt_tp_create(const char *host,const rpcprog_t prognum, const rpcvers_t
    versnum, const struct netconfig *netconf);

CLIENT *clnt_door_create(const rpcprog_t prognum, const rpcvers_t versnum, const
    uint_t sendsz);
```

API RPC – вызов клиента

```
#include <rpc/rpc.h>
enum clnt_stat clnt_call(CLIENT *clnt, const rpcproc_t procnum,
const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc,
caddr_t out, const struct timeval tout);

enum clnt_stat clnt_send (CLIENT *clnt, const u_long
procnum, const xdrproc_t proc, const caddr_t in);

bool_t clnt_freeres(CLIENT *clnt, const xdrproc_t outproc,
caddr_t out);

void clnt_geterr(const CLIENT *clnt, struct rpc_err *errp);

void clnt_perrno(const enum clnt_stat stat);

void clnt_perror(const CLIENT *clnt, const char *s);

enum clnt_stat rpc_broadcast(const rpcprog_t prognum,const rpcvers_t
    versnum, const rpcproc_t procnum,const xdrproc_tinproc, const caddr_t in,
const xdrproc_t outproc, caddr_t out, const resultproc_t eachresult, const char
    *nettype);
```

API RPC – серверные функции

bool_t svc_control(SVCXPRT *svc, const uint_t req, void *info);

int svc_create(const void (*dispatch)const struct svc_req *, const SVCXPRT *), const rpcprog_t prognum, const rpcvers_t versnum, const char *nettype);

void svc_destroy(SVCXPRT *xprt);

SVCXPRT *svc_raw_create(void);

SVCXPRT *svc_tli_create(const int fildes, const struct netconfig *netconf, const struct t_bind *bind_addr, const uint_t sendsz, const uint_t recvsz);

SVCXPRT *svc_tp_create(const void (*dispatch)const struct svc_req *, const SVCXPRT *), const rpcprog_t prognum, const rpcvers_t versnum, const struct netconfig *netconf);

SVCXPRT *svc_vc_create(const int fildes, const uint_t sendsz, const uint_t recvsz);

SVCXPRT *svc_door_create(void (*dispatch)(struct svc_req *, SVCXPRT *), const rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz);

API RPC – серверные функции

```
int svc_dg_enablecache(SVCXPRT *xprt, const uint_t cache_size);

int svc_done(SVCXPRT *xprt);

void svc_exit(void);

bool_t svc_freeargs(const SVCXPRT *xprt, const txdrproc_t inproc,caddr_t in);

bool_t svc_getargs(const SVCXPRT *xprt, const xdrproc_t inproc,caddr_t in);

int svc_getcallerucred(const SVCXPRT *xprt, ucred_t **ucred);

struct netbuf *svc_getrpccaller(const SVCXPRT *xprt);

void svc_run(void);

bool_t svc_sendreply(const SVCXPRT *xprt, const xdrproc_t outproc,caddr_t
    outint svc_max_pollfd);
```

Передача данных по сети

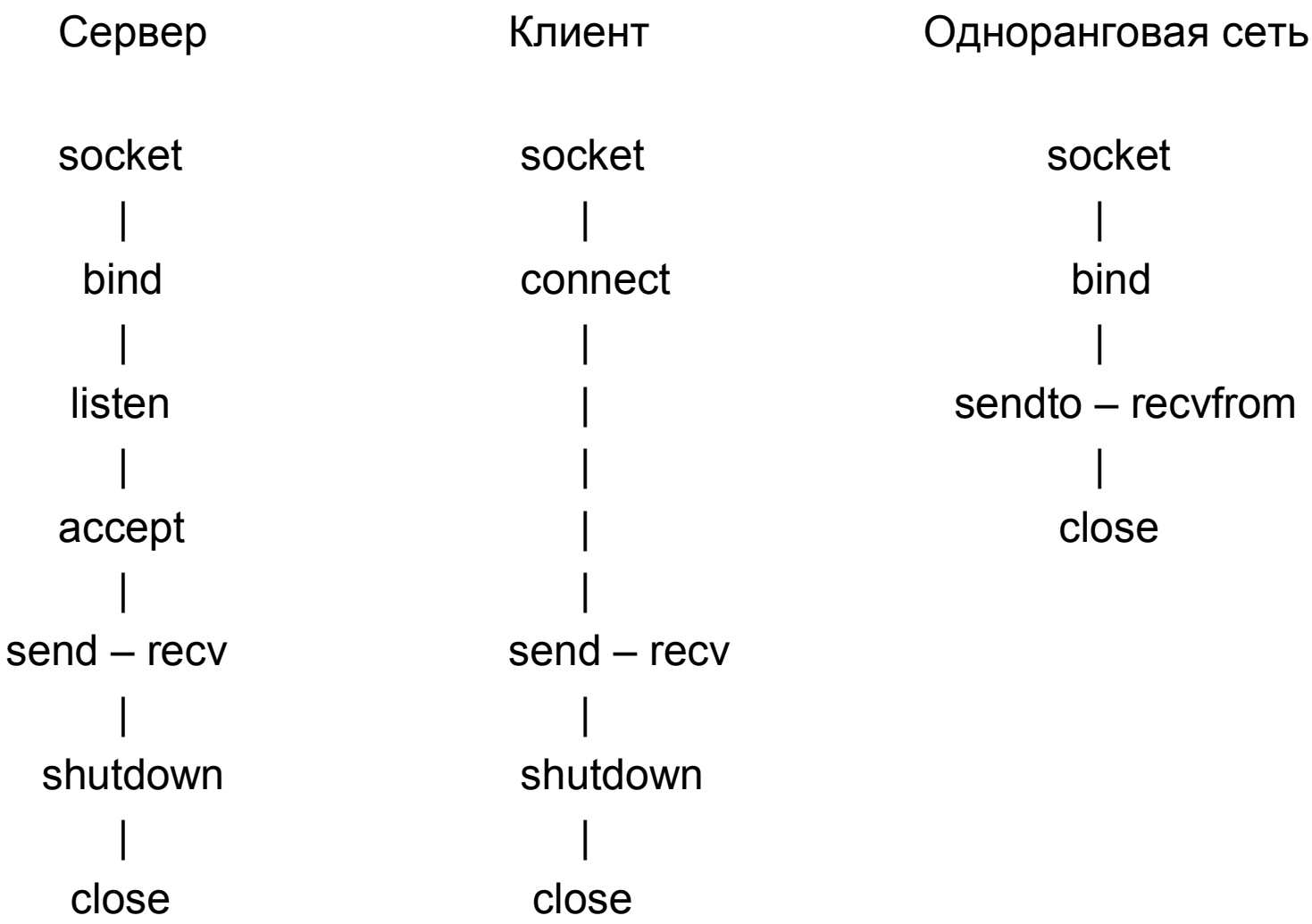
Сокеты – sockets – предложены в BSD UNIX

<fcntl.h>
<netinet/tcp.h>
<sys/socket.h>
<sys/stat.h>
<sys/uio.h>
<sys/un.h>

<arpa/inet.h>
<netdb.h>
<netinet/in.h>
<unistd.h>

-lnsl -lsocket

Схема применения функций API



Создание сокета

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

int socket( int domain, int type, int protocol );
```

domain

AF_UNIX	AF_INET	AF_INET6

type

SOCK_STREAM	SOCK_DGRAM	SOCK_SEQPACKET

protocol

0

Возвращает дескриптор или -1.

Связывание с именем

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

int bind( int socket, const struct sockaddr *address, socklen_t address_len );

struct sockaddr {
      sa_family_t  sa_family;
      char         sa_data[1];
};
struct sockaddr_un {
      sa_family_t  sun_family;
      char         sun_path[1];
};
-------------------------------------- Пример ----------------------------------------------------
int sfd;      struct sockaddr_un my_addr;
sfd = socket( AF_UNIX, SOCK_STREAM, 0);
if (sfd == -1) ….
memset(&my_addr, '\0', sizeof( struct sockaddr_un ));
my_addr.sun_family = AF_UNIX;
strncpy( my_addr.sun_path, "/somepath", sizeof(my_addr.sun_path) -1 );
if (bind(sfd, (struct sockaddr *) &my_addr, sizeof(struct sockaddr_un)) == -1) …
```

Связывание с адресом

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

IP v4

```
struct sockaddr_in {
        sa_family_t     sin_family;   // AF_INET
        in_port_t       sin_port;     // Port number
        struct in_addr  sin_addr;     // IP address
};
```

IP v6

```
struct sockaddr_in6 {
        sa_family_t     sin6_family;     // AF_INET6.
        in_port_t       sin6_port;       // Port number.
        uint32_t        sin6_flowinfo;   //IPv6 traffic class and flow info
        struct in6_addr sin6_addr;       //IPv6 address.
        uint32_t        sin6_scope_id;   //  Set of interfaces for a scope.
};
```

Преобразование в серверный сокет

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

int listen(int socket, int backlog);       0 / -1
                          SOMAXCONN
```

Подключение клиента

```
int accept( int socket, struct sockaddr *address, socklen_t *address_len );
```

Завершение работы

```
int shutdown(int socket, int how);      SHUT_RD,   SHUT_WR,    SHUT_RDWR
int close(int fildes);
```

Подключение к серверу

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

int connect(  int     socket,
              const struct sockaddr *address,
              socklen_t address_len);
```

Возвращает 0 / -1

Если SOCK_DGRAM – задаёт адрес по умолчанию.

Опции сокета:

```
int getsockopt( int socket, int level, int option_name, void *option_value,
              socklen_t *option_len);

int setsockopt( int socket, int level, int option_name, const void *option_value,
              socklen_t option_len);
```

Опции сокета

Socket-Level Options (SOL_SOCKET)

Option	Parameter Type	Parameter Meaning
SO_ACCEPTCONN	**int**	Non-zero indicates that socket listening is enabled (*getsockopt*() only).
SO_BROADCAST	**int**	Non-zero requests permission to transmit broadcast datagrams (SOCK_DGRAM sockets only).
SO_DEBUG	**int**	Non-zero requests debugging in underlying protocol modules.
SO_DONTROUTE	**int**	Non-zero requests bypass of normal routing; route based on destination address only.
SO_ERROR	**int**	Requests and clears pending error information on the socket (*getsockopt*() only).
SO_KEEPALIVE	**int**	Non-zero requests periodic transmission of keepalive messages (protocol-specific).
SO_LINGER	**struct linger**	Specify actions to be taken for queued, unsent data on *close*(): linger on/off and linger time in seconds.
SO_OOBINLINE	**int**	Non-zero requests that out-of-band data be placed into normal data input queue as received.
SO_RCVBUF	**int**	Size of receive buffer (in bytes).
SO_RCVLOWAT	**int**	Minimum amount of data to return to application for input operations (in bytes).
SO_RCVTIMEO	**struct timeval**	Timeout value for a socket receive operation.
SO_REUSEADDR	**int**	Non-zero requests reuse of local addresses in *bind*() (protocol-specific).
SO_SNDBUF	**int**	Size of send buffer (in bytes).
SO_SNDLOWAT	**int**	Minimum amount of data to send for output operations (in bytes).
SO_SNDTIMEO	**struct timeval**	Timeout value for a socket send operation.
SO_TYPE	**int**	Identify socket type (*getsockopt*() only).

Передача сообщений

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

ssize_t send( int       socket,
              const void *buffer,
              size_t        length,
              int              flags);       //MSG_EOR MSG_OOB  MSG_NOSIGNAL


ssize_t write(int fildes, const void *buf, size_t nbyte);

ssize_t sendto( int    socket,
              const void *message,
              size_t        length,
              int              flags,
              const struct sockaddr *dest_addr,
              socklen_t    dest_len);
```

Прием сообщений

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

ssize_t recv( int    socket,
              void *     buffer,
              size_t     length,
              int        flags); // MSG_PEEK  MSG_OOB  MSG_WAITALL

ssize_t read(int fildes, void *buf, size_t nbyte);

ssize_t recvfrom( int        socket,
                  void *  buffer,
                  size_t  length,
                  int        flags,
                  struct sockaddr *address,
                  socklen_t *address_len);
```

Адресация IPv4: порт

Задать номер порта (до 1024 – привилегированные).

Использовать сетевой порядок байт:

```
#include <arpa/inet.h>

uint32_t htonl( uint32_t hostlong );
uint16_t htons( uint16_t hostshort );
uint32_t ntohl( uint32_t netlong );
uint16_t ntohs( uint16_t netshort );
```

Пример:

```
    portnumber = 8015;
    struct sockaddr_in  sa;
    sa.sin_family = AF_INET;
    sa.sin_port = htons( portnumber );
```

Адресация IPv4: адрес машины

Сформировать структуру адреса по IP и наоборот.

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

```
int inet_aton( const char *cp, struct in_addr *addr );
char *inet_ntoa( const struct in_addr in );
```

Формат текстового описания – двоично-десятичный: “x.x.x.x”.

Пример (продолжение).

```
    inet_aton( “194.67.66.174”, &sa.sin_addr );
```

```
int inet_lnaof(const struct in_addr in);        // Номер  локальной сети (подсети)
int inet_netof(const struct in_addr in);        // Номер сети
```

Адресация IPv4 и IPv6: адрес машины

```
const char *inet_ntop( int af, const void *addr, char *cp,size_t size );
int inet_pton( int af, const char *cp, void *addr );
```

Размер буфера:

```
#define INET_ADDRSTRLEN 16
#define INET6_ADDRSTRLEN 46
```

Формат адреса:

IPv6 (sa.sin_family = AF_INET6)

x:x:x:x:x:x:x:x, например, 1080:0:0:0:8:800:200C:417A

x:x:x:x:x:x:d.d.d.d для смешанного окружения.

IPv4

d.d.d.d

Использование DNS для адресации

Имя локальной машины

```
int gethostname(char *name, size_t namelen);  // 0/-1, не более 255 симв.
```

```
#include <netdb.h>
struct hostent *gethostbyname( const char *name );
struct hostent *gethostbyaddr( const char *addr, int len, int type );
```

Структура описания машины

```
struct hostent {
        char *h_name;           /* имя машины */
        char **h_aliases;       /* список дополнительных имен */
        int h_addrtype;         /* тип адреса */
        int h_length;           /* длина адреса */
        char **h_addr_list;     /* список адресов */
};
```

Пример адресации по имени

```
struct sockaddr_in mk_address( const char* hostname, int portnum )
{
    struct sockaddr_in sa;
    sa.sin_family = AF_INET;
    if ( !hostname ) {
        sa.sin_addr.s_addr = INADDR_ANY;
    } else {
        struct hostent *hp;
        hp = gethostbyname( hostname );
        if ( !hp ) {
            Не нашли…
        }
        memcpy( (char*)&sa.sin_addr, hp->h_addr, hp->h_length );
    }
    sa.sin_port = htons( portnum );
    return sa;
}
```

Ожидание событий, связанных с описателями

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

int select( int nfds, fd_set *readfds, fd_set *writefds, fd_set *errorfds,
                        struct timeval *timeout);

int pselect( int nfds, fd_set *readfds, fd_set *writefds, fd_set *errorfds,
                        const struct timespec *timeout, const sigset_t *sigmask);
```

nfsd – наибольший дескриптор+1

Возвращает общее количество готовых дескрипторов, устанавливает готовые и сбрасывает остальные.

Заполнение массива дескрипторов:

```
void FD_SET( int fd, fd_set *fdset );      установка указанного
int FD_ISSET( int fd, fd_set *fdset );     проверка установки
void FD_CLR( int fd, fd_set *fdset );      сброс указанного
void FD_ZERO( fd_set *fdset );             сброс всех
```

Ожидание событий, альтернатива

```
#include <poll.h>

    int poll(struct pollfd fds[], nfds_t nfds, int timeout);

    struct pollfd {
            int    fd;
            short  events;          // Что интересует
            short  revents;         // что произошло
    };
```

POLLIN	Data other than high-priority data may be read without blocking.
POLLRDNORM	Normal data may be read without blocking.
POLLRDBAND	Priority data may be read without blocking.
POLLPRI	High priority data may be read without blocking.
POLLOUT	Normal data may be written without blocking.
POLLWRNORM	Equivalent to POLLOUT.
POLLWRBAND	Priority data may be written.
POLLERR	An error has occurred (revents only).
POLLHUP	Device has been disconnected (revents only).
POLLNVAL	Invalid fd member (revents only).

Пример использования poll

```
#include <poll.h>
struct pollfd fds[2];
int timeout_msecs = 500;
int ret;    int i;

     fds[0].fd = …
     fds[1].fd = …
     fds[0].events = POLLOUT | POLLWRBAND;
     fds[1].events = POLLOUT | POLLWRBAND;
     ret = poll(fds, 2, timeout_msecs);
     if (ret > 0) {      /* что-то случилось */
     for ( i=0; i<2; i++ ) {
            if (fds[i].revents & POLLWRBAND)  /* Возможен приоритетный вывод. */
            if (fds[i].revents & POLLOUT)          /* Возможна запись */
      }
      if (fds[i].revents & POLLHUP)  /* Разрыв */
```

Интерфейс транспортного уровня TLI/XTI

1986 – AT&T TLI; 1990 – включены протоколы TCP/IP; 1988 – X/Open XTI

Заголовочный файл

#include <xti.h>

#include <tiuser.h> для TLI

Вспомогательные функции:

int t_sysconf(intname); _SC_T_IOV_MAX

t_errno используют функции XTI

int t_error(const char *errmsg); ошибка функции интерфейса

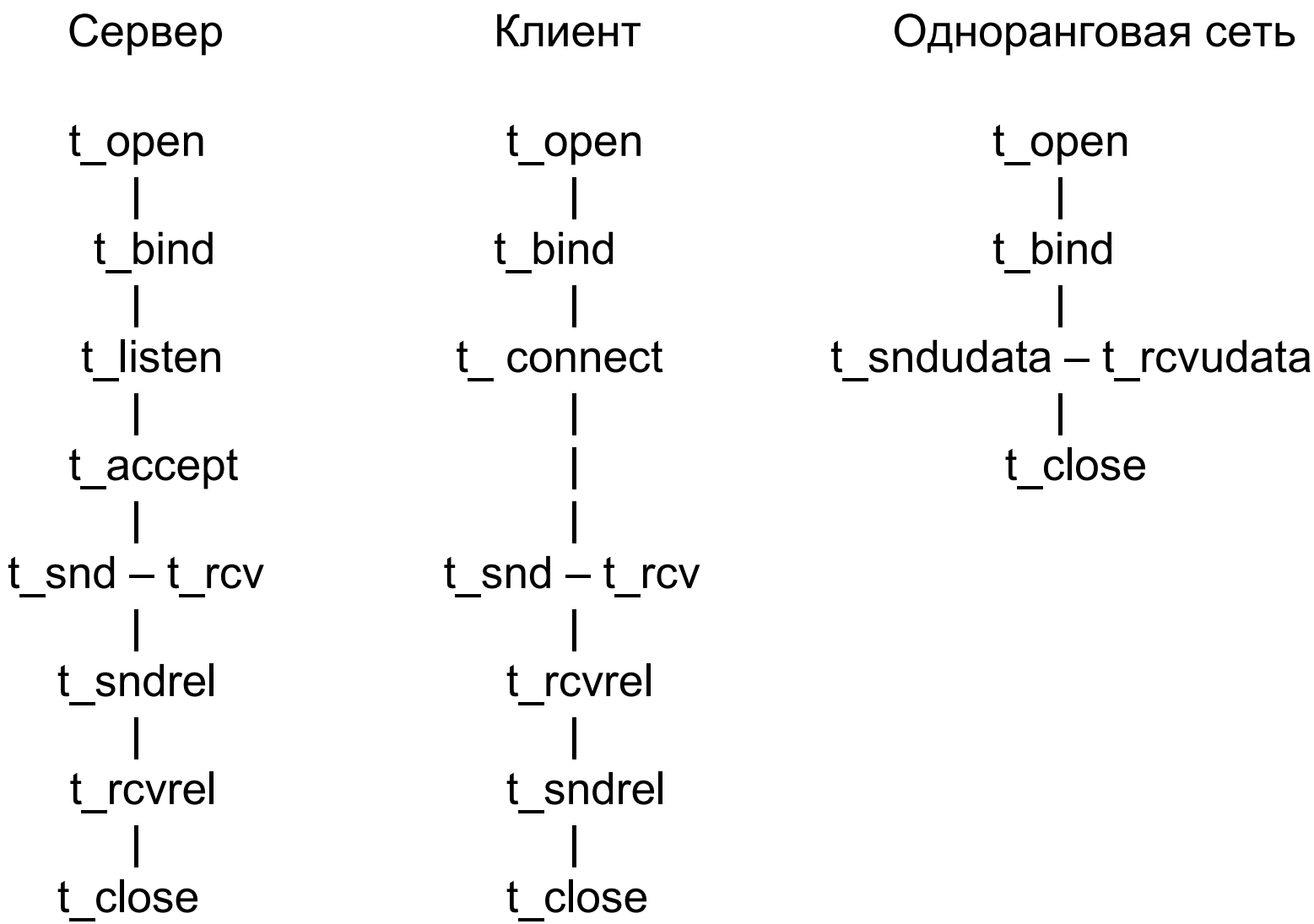
void *t_alloc(int fd, int struct_type, int fields); выделение памяти для структур

struct_type		struct_type	
T_OPTMGMT	struct t_optmgmt	T_BIND	struct t_bind
T_UNITDATA	struct t_unitdata	T_DIS	struct t_discon
T_INFO	struct t_info	T_UDERROR	struct t_uderr
		T_CALL	struct t_call

fields: T_ADDR, T_OPT, T_UDATA, T_ALL

int t_free(void *ptr, int struct_type); освобождение памяти

Схема применения функций API



Создание конечной точки транспортировки

int t_open(const char *name, int oflag, struct t_info *info); // дескриптор / -1

name:

Виртуальный канал с установлением соединения

“/dev/ticotsord” T_COTSRD

Дейтаграммное соединение

“/dev/ticlts” T_CLTS

oflag: O_RDWR, O_NONBLOCK

----- info возвращает характеристики провайдера транспорта -----

```
struct t_info {
    t_scalar_t    addr;     /* max size of the transport protocol address */
    t_scalar_t    options;  /* max number of bytes of protocol-specific options */
    t_scalar_t    tsdu;     /* max size of a transport service data unit (TSDU) */
    t_scalar_t    etsdu;    /* max size of an expedited transport service data unit */
    t_scalar_t    connect;  /* max amount of data allowed on connection establishment */
    t_scalar_t    discon;   /* max amount of data allowed on t_snddis() and t_rcvdis() */
    t_scalar_t    servtype; /* service type supported by the transport provider */
    t_scalar_t    flags;    /* other info about the transport provider */
};
```

Задание адреса точки транспортировки

```
int t_bind( int fd, const struct t_bind *req, struct t_bind *ret );

  struct t_bind {
     struct netbuf   addr;
     unsigned        qlen;
  };
  struct netbuf {
     unsigned int    maxlen;
     unsigned int    len;
     char*           buf;
};
```

Как задать адрес?

Локальный - HOST_SELF

```
#include <netdir.h>
     char *taddr2uaddr(struct netconfig *config, struct netbuf *addr);
     struct netbuf *uaddr2taddr(struct netconfig *config, char *uaddr);
```

Формат адреса: “x.x.x.x.x.x”

Ожидание пакета и установление соединения

int t_listen(int fd, struct t_call *call); 0/-1

В синхронном режиме, если не установлен флаг O_NONBLOCK

```
struct t_call {
    struct netbuf addr;      адрес клиентской точки транспортировки
    struct netbuf opt;       параметры, зависящие от протокола
    struct netbuf udata;     необязательные пользовательские данные
    int sequence;            идентификатор соединения
};
```

Принимает клиентский запрос

int t_accept(int fd, int newfd, const struct t_call *call);

newfd – новая точка транспортировки для связи с клиентом

Пример установления соединения

```
struct t_call *call=(struct t_call*) t_alloc( fd, T_CALL T_ALL );
while ( t_listen( fd, call ) == 0 ) {
    switch ( fork() ) {
        case -1:  perror( “fork” ); break;
        default: break;
        case 0:
            if ( (newfd = t_open( “/dev/ticotsord”, O_RDWR, 0 )) == -1 ||
            t_bind( newfd, 0, 0 ) == -1 ) {
                t_error(“”);
            } else if (t_accept( fd, newfd, call ) == -1 ) {
                t_error(“”);
            } else {
                t_close( fd );
                        Осуществляем связь с клиентом через newfd
            }
    }
}
```

Соединение со стороны клиента

int t_connect(int fd, const struct t_call *sndcall, struct t_call *rcvcall);

В sndcall – адрес сервера.

Если в неблокирующем режиме t_errno == TNODATA

Проверить успешность соединения

int t_rcvconnect(int fd, struct t_call *call);

Пример.

```
struct netconfig *nconf;
struct t_call *call = (struct t_call *)t_alloc( fd, T_CALL, T_ALL );
netbuf* A = uaddr2taddr( nconf, “168.202.3.4.50.6);
memcpy( call->addr.buf? A->buf, A->len );
t_connect( fd, call, call );
if ( t_errno == T_LOOK ) {
    if ( n = t_look( fd ) == T_DISCONNECT ) { …
```

Передача и прием данных

```
int t_snd(int fd, void *buf, unsigned int nbytes, int flags);

int t_rcv(int fd, void *buf, unsigned int nbytes, int *flags);

    T_EXPEDITED  T_MORE    T_PUSH


int t_sndudata(int fd, const struct t_unitdata *unitdata);

int t_rcvudata(int fd, struct t_unitdata *unitdata, int *flags);

 struct t_unitdata {
    struct netbuf addr;        адрес, куда передавать
    struct netbuf opt;         опции провайдера
    struct netbuf udata;       сообщение
 };
```

Завершение приема/передачи данных

Нормальное завершение:

Извещение о завершении передачи (может продолжать прием данных)

```
int t_sndrel(int fd);
```

Подтверждение сообщения о завершении (может отправлять данные)

```
int t_rcvrel(int fd);
```

Аварийное завершение:

Запрос на разрыв соединения или отказ от соединения

```
int t_snddis(int fd, const struct t_call *call);
```

Принятие информации о разрыве (событие T_DISCONNECT)

```
int t_rcvdis(int fd, struct t_discon *discon);
struct t_discon {
        struct netbuf       udata;      // данные пользователя
        int                 reason;     // код причины завершения
        int                 sequence;   //
};
```

Определение текущего события в соединении — `int t_look(int fd);`

Закрытие конечной точки транспортировки — `int t_close(int fd);`

Разработка ПО ОС UNIX

Лекционный материал завершён.

СПАСИБО ЗА ВНИМАНИЕ