

Обмен данными между процессами (продолжение)

Неименованные каналы

```
ewgenij@dew:~$ history | grep mount
441 sudo mount whirl:~/ ~/STUDIO/COUETTE_CUDA/whirl
507 history | grep mount
```

В операционных системах UNIX/LINUX все процессы являются узлами одного дерева процессов с корневым процессом *init*. Соответственно запущенные нами процессы *history* и *grep* являются дочерними по отношению к родительскому процессу *bash*. Обнаружив при разборе командной строки вертикальную линию оболочка создает неименованный канал и передает дескрипторы соответствующим дочерним процессам.

Фрагмент вывода команды “ps -fax” :

<i>PID</i>	<i>STAT</i>	<i>TIME</i>	<i>COMMAND</i>
.....			
<i>1</i>	<i>Ss</i>	<i>0:01</i>	<i>/sbin/init</i>
.....			
<i>2087</i>	<i>Ss+</i>	<i>0:00</i>	<i>_ /bin/bash</i>
<i>2292</i>	<i>Ss</i>	<i>0:00</i>	<i>_ /bin/bash</i>
<i>2311</i>	<i>Sl+</i>	<i>5:18</i>	<i> _ gnuplot</i>
<i>4321</i>	<i>Ss</i>	<i>0:00</i>	<i>_ /bin/bash</i>
<i>4340</i>	<i>S+</i>	<i>0:00</i>	<i>_ ssh whirl</i>

Рассмотрим реализацию этой технологии на платформе MS Windows:

```
#include <windows.h>
```

```
#include <math.h>
```

```
int main()
```

```
{
```

```
HANDLE hPipeIn, hPipeOut;
```

```
SECURITY_ATTRIBUTES sa;
```

```
STARTUPINFO si;
```

```
PROCESS_INFORMATION pi;
```

```
char buff[80];
```

```
unsigned long dw;
```

```
double x;
```

```
sa.nLength=sizeof(sa);
```

```
sa.lpSecurityDescriptor=NULL;
```

```
sa.bInheritHandle=TRUE; //дочерние процессы наследуют
```

```
//дескрипторы
```

Файл *p1.c*

```
if(!CreatePipe(&hPipeIn,&hPipeOut,&sa,0)){
    printf("The pipe could not be created\n");
    exit(1);
}// Создание канала с дескриптором ввода hPipeIn и
// вывода hPipeOut
memset(&si,0,sizeof(si));
si.cb=sizeof(si);
si.dwFlags=STARTF_USESTDHANDLES;//использовать поля
//для стандартных потоков
si.hStdInput=hPipeIn; //перенаправление потока ввода
si.hStdOutput=GetStdHandle(STD_OUTPUT_HANDLE);
si.hStdError=GetStdHandle(STD_ERROR_HANDLE);
if(!CreateProcess(NULL,"p2",NULL,NULL,TRUE,0,NULL,NULL,
&si,&pi)){
    printf("Could not create process, %i\n",GetLastError());
    exit(1);
}
```

Структура **STARTUPINFO**, содержащая информацию о начальных свойствах окна:

```
typedef struct _STARTUPINFO {  
    DWORD cb;  
    LPTSTR lpReserved;  
    LPTSTR lpDesktop;  
    LPTSTR lpTitle;  
    DWORD dwX;  
    DWORD dwY;  
    DWORD dwXSize;  
    DWORD dwYSize;  
    DWORD dwXCountChars;  
    DWORD dwYCountChars;  
    DWORD dwFillAttribute;  
    DWORD dwFlags;  
    WORD wShowWindow;  
    WORD cbReserved2;  
    LPBYTE lpReserved2;  
};
```

```
HANDLE hStdInput;  
HANDLE hStdOutput;  
HANDLE hStdError; } STARTUPINFO, *LPSTARTUPINFO;
```

```
BOOL CreatePipe(                                PHANDLE hReadPipe, //  
дескриптор для чтения                            PHANDLE hWritePipe,  
// дескриптор для записи LPSECURITY_ATTRIBUTES  
lpPipeAttributes, // атриб. безоп. DWORD nSize // размер  
буфера канала );
```

```
CloseHandle(hPipeIn);
```

```
for(x=0.0;x<3.1416;x+=0.1){  
    sprintf(buff,"%g  %g\n",x,sin(x));  
    WriteFile(hPipeOut, buff, strlen(buff), &dw, NULL);  
}
```

```
*buff=(char)26;  
WriteFile(hPipeOut, buff, 1, &dw, NULL);
```

```
CloseHandle(hPipeOut);
```

```
return 0;
```

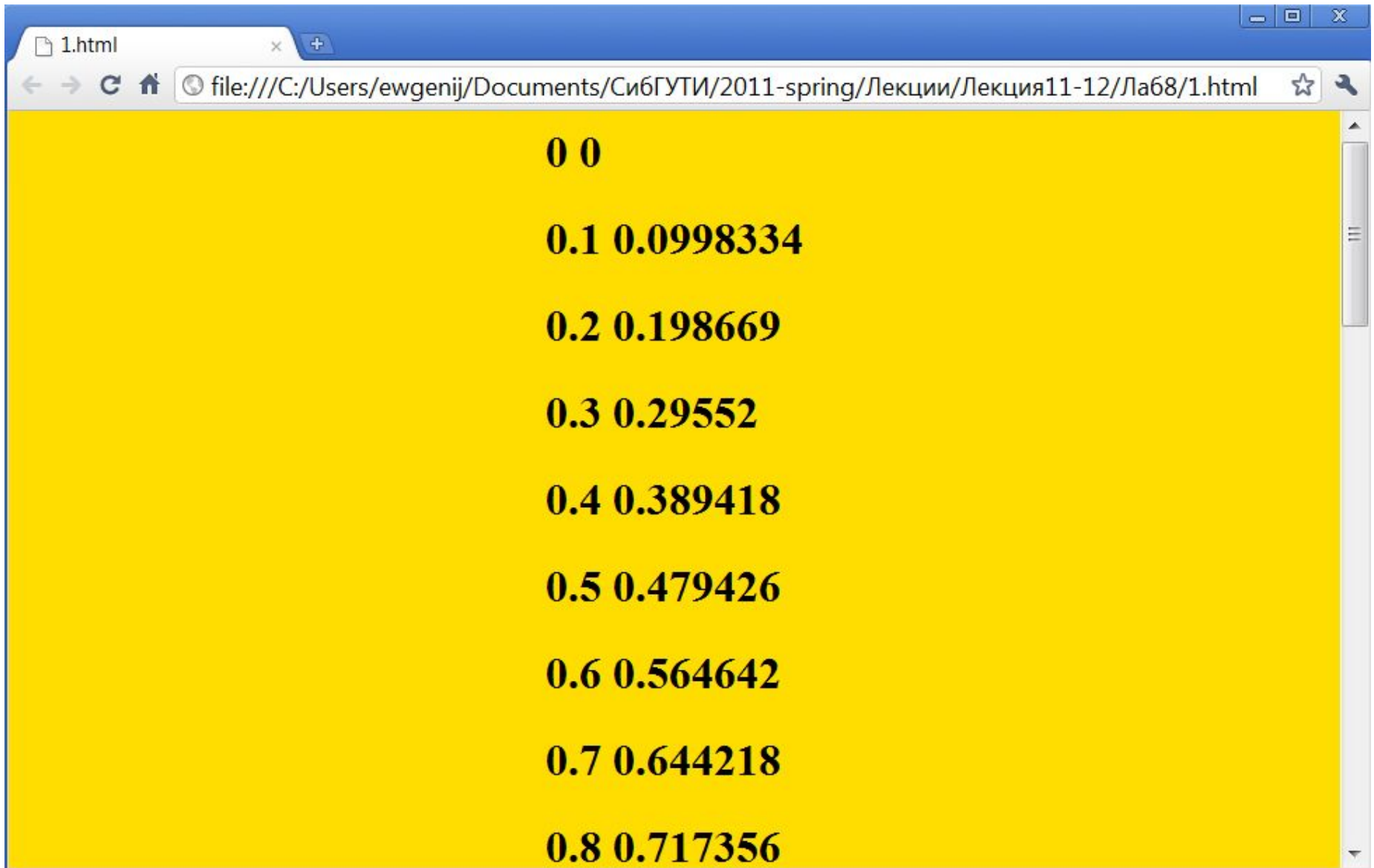
```
}
```



```
    fprintf(stdout, "</H1>");
    fprintf(stdout, "</TD>");
    fprintf(stdout, "</TR>");
}
    fprintf(stdout, "</TABLE></CENTER>");
    fprintf(stdout, "</BODY>");
    fprintf(stdout, "</HTML>");

return 0;
}
```

C:\...\Лекции\Лекция11-12\Лаб8>p1 > 1.html



Именованные каналы:

Сервер:

```
#include <stdio.h>
#include <windows.h>

void main()
{
HANDLE hPipe;
LPTSTR lpPipeName = TEXT("\\\\.\\pipe\\MyPipe");
char buff[255];
DWORD iBytesToRead = 255, i;
```

Файл *pr1.c*

```
hPipe = CreateNamedPipe(  
    lpPipeName,          // имя канала  
    PIPE_ACCESS_DUPLEX, // чтение и запись из канала  
    PIPE_TYPE_MESSAGE | // передача сообщений по каналу  
    PIPE_READMODE_MESSAGE //режим чтения сообщений  
    PIPE_WAIT,          // синхронная передача сообщений  
    PIPE_UNLIMITED_INSTANCES, //число экземпляров  
    4096,               // размер выходного буфера  
    4096,               // размер входного буфера  
    NMPWAIT_USE_DEFAULT_WAIT, // тайм-аут клиента  
    NULL);             // защита по умолчанию
```

```
if (hPipe == INVALID_HANDLE_VALUE) {  
    printf("CreatePipe failed: error code %d\n",  
        (int)GetLastError());  
    return;  
}
```

```
if((ConnectNamedPipe(hPipe, NULL))==0)
{
printf("client could not connect\n");
return;
}
ReadFile(hPipe, buff, iBytesToRead, &iBytesToRead, NULL);
for(i=0; i<iBytesToRead; i++) printf("%c",buff[i]);
}
```

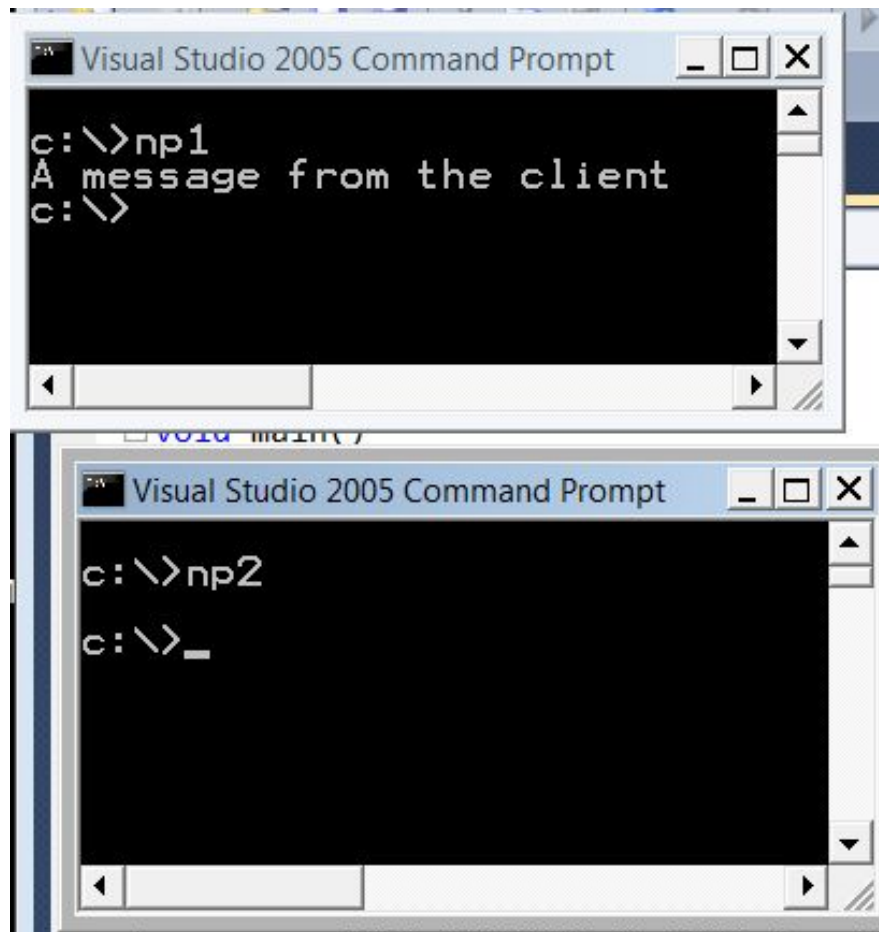
Клиент:

```
#include <stdio.h>
#include <windows.h>
void main(){
HANDLE hPipe;
LPTSTR lpPipeName = TEXT("\\\\.\\pipe\\MyPipe");
//TEXT("\\\\.\\WANDERER\\pipe\\MyPipe");
DWORD iBytesToWrite;
char buff[] = "A message from the client";
```

Файл *np2.c*

```
hPipe = CreateFile(  
    lpPipeName, // имя канала  
    GENERIC_READ | // чтение и запись в канал  
    GENERIC_WRITE,  
    0,           // нет разделяемых операций  
    NULL,       // защита по умолчанию  
    OPEN_EXISTING, // открытие существующего канала  
    0,         // атрибуты по умолчанию  
    NULL);    // нет шаблона атрибутов
```

```
WriteFile(hPipe, buff, strlen(buff), &iBytesToWrite, NULL);  
CloseHandle(hPipe);  
}
```



Упражнение 1: протестировать разобранные программы.

Упражнение 2: написать «чат» один-к-одному в локальной сети.

Тема курсовой №3: написать многопользовательский «чат» в локальной сети (после темы Многопоточные приложения в MS Windows).

Сокеты

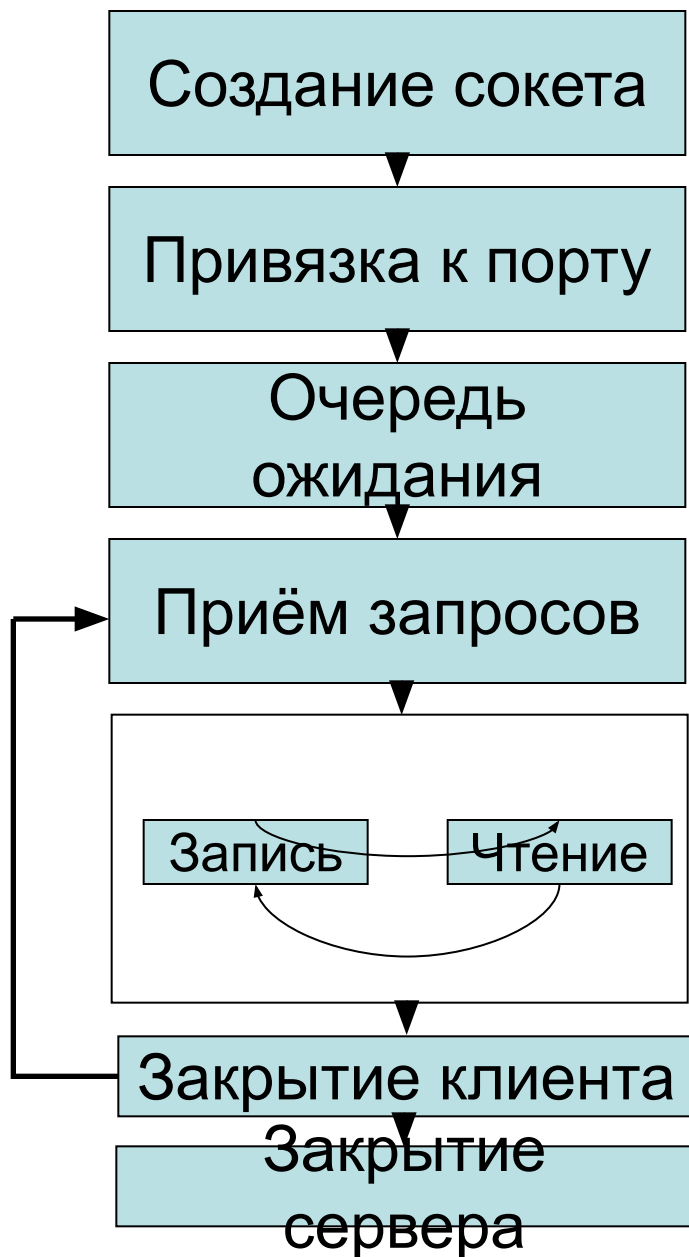
(как средство обмена данными между процессами)

Сокет – это устройство двунаправленного действия, предназначенное для передачи данных между процессами на одном или разных узлах компьютерной сети.

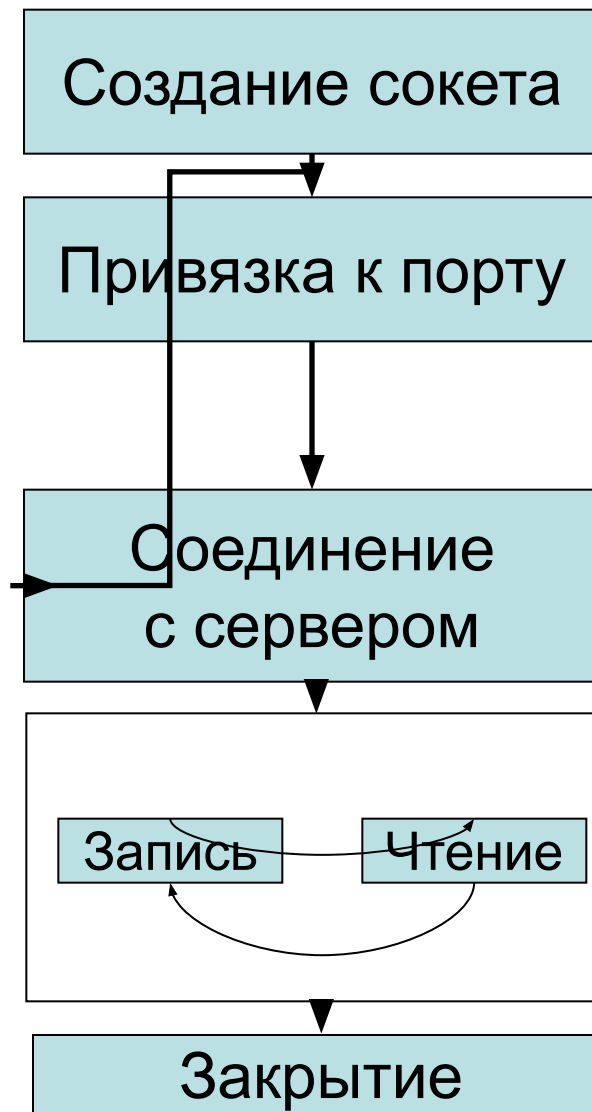
Для обмена данными между процессами даже на одном компьютере необходимо, чтобы на нем была установлена *сетевая карта* и *сетевые протоколы*. Чаще всего, но не обязательно, используется стек протоколов TCP/IP.

При создании сокета задаются три параметра: *пространство имен*, *тип взаимодействия* и *протокол*.

сокет-сервер



сокет-клиент



Простой *http*-клиент. Отличия сокетов Беркли от *Windows* сокетов.

```
//#include <sys/socket.h>  
//#include <netinet/in.h>  
//#include <arpa/inet.h>  
#include <Winsock2.h>  
#include <string.h>  
#include <stdio.h>  
#define SERVERADD "94.100.191.203"  
int main(){  
    SOCKET socket_fd; //int socket_fd;  
    struct sockaddr_in name;  
    char buff[1024];  
    char buffer[10000];  
    size_t num_char;  
    struct hostent* host;
```

Файл *s1.c*

```
//Инициализация библиотеки Winsock2
if (WSAStartup(0x202,(WSADATA *)&buff[0]))
{
    printf("WSAStart error %d\n",WSAGetLastError());
    return -1;
}
```

```
host = gethostbyname( "www.mail.ru" );
```

```
socket_fd=socket(PF_INET, SOCK_STREAM, 0);
```

```
name.sin_family=AF_INET;
```

```
name.sin_port=htons(80);
```

Пространство имен (каким образом записываются адреса):

Значение	Описание
PF_INET	Протоколы семейства IPv4; TCP/IP (в настоящее время может использоваться синоним AF_INET)
PF_LOCAL	Локальные именованные каналы в стиле BSD
PF_IPX	Протоколы Novell
PF_INET6	Протоколы семейства IPv6; TCP/IP

Тип взаимодействия:

Значение	Описание
SOCK_STREAM	Протокол последовательной передачи данных в виде байтового потока с подтверждением доставки (TCP)
SOCK_RDM	Протокол пакетной передачи данных с подтверждением доставки
SOCK_DGRAM	Протокол пакетной передачи данных без подтверждения доставки (UDP)
SOCK_RAW	Протокол передачи низкоуровневых данных без подтверждения доставки

```
//inet_aton("195.189.238.9",&name.sin_addr);
if (inet_addr("94.100.191.203")!=INADDR_NONE)
    name.sin_addr.s_addr=inet_addr("94.100.191.203");
else {
    printf("no such address\n");
return -2;
}

if(connect(socket_fd,(struct sockaddr*)&name,sizeof(name))==
        SOCKET_ERROR){
    printf("Couldn't connect to server\n");
    return -1;
}
sprintf(buffer,"GET ^n");

send(socket_fd,buffer,strlen(buffer),0);
```

```
while(1){
    num_char=recv(socket_fd,buffer,sizeof(buffer) - 1,0);
    if(num_char==0){
        closesocket(socket_fd);
        WSACleanup();
        return 1;
    }
    fwrite(buffer,sizeof(char),num_char,stdout);
}
closesocket(socket_fd);
WSACleanup();
return 0;
}
```

Компиляция:

```
>cl s1.c Ws2_32.lib
```

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

```
#include <winsock2.h>
```

```
#include <windows.h>
```

```
#define MY_PORT 1952
```

```
int main(int argc, char* argv[])
```

```
{
```

```
char buff[1024];
```

```
SOCKET mysocket, client_socket;
```

```
struct sockaddr_in local_addr, client_addr;
```

```
int client_addr_size=sizeof(client_addr);
```

```
if (WSAStartup(0x0202,(WSADATA *) &buff[0]))
```

```
{
```

```
printf("Error WSAStartup %d\n",
```

```
WSAGetLastError());
```

```
return -1;
```

```
}
```

Файл *srv.c*

Сокет - сервер


```
if ((mysocket=socket(AF_INET,SOCK_STREAM,0))<0)
{
    printf("Error socket %d\n",WSAGetLastError());
    WSACleanup();
    return -1;
}
local_addr.sin_family=AF_INET;
local_addr.sin_port=htons(MY_PORT);
local_addr.sin_addr.s_addr=0;

if (bind(mysocket,(struct sockaddr *)&local_addr,
        sizeof(local_addr))) {
    printf("Error bind %d\n",WSAGetLastError());
    closesocket(mysocket);
    WSACleanup();
    return -1;
}
```

```
if (listen(mysocket, 0x100))
{
    printf("Error listen %d\n",WSAGetLastError());
    closesocket(mysocket);
    WSACleanup();
    return -1;
}
printf("Waiting for calls\n");

while((client_socket=accept(mysocket, (struct sockaddr *)
    &client_addr, &client_addr_size)))
{
    struct hostent *hst;
    int bytes_recv;
    hst=gethostbyaddr((char *)&client_addr.sin_addr.s_addr,4,
        AF_INET);
```

```
printf("+%s [%s] new connect!\n",
(hst)?hst->h_name:"", inet_ntoa(client_addr.sin_addr));

send(client_socket,"Hello a new client!\n",
      sizeof("Hello a new client!\n"),0);

while( ( bytes_recv=recv(client_socket, &buff[0], sizeof(buff),0) )
      && bytes_recv !=SOCKET_ERROR )
      send(client_socket, &buff[0], bytes_recv, 0);

printf("Client was disconnected\n");
closesocket(client_socket);
}

return 0;
}
```

```
#include <stdio.h>
#include <string.h>
#include <winsock2.h>
#include <windows.h>

#define PORT 1952
#define SERVERADDR "127.0.0.1"

int main(int argc, char* argv[])
{
    char buff[1024];
    struct sockaddr_in dest_addr;
    SOCKET my_sock;
    struct hostent *hst;
    int nsize;
```

Файл *cln.c*

Сокет - клиент

```
//инициализация библиотеки Winsock
```

```
if (WSAStartup(0x202,(WSADATA *)&buff[0]))  
{  
    printf("WSAStart error %d\n",WSAGetLastError());  
    return -1;  
}
```

```
// создание сокета
```

```
my_sock=socket(AF_INET,SOCK_STREAM,0);  
if (my_sock < 0){  
    printf("Socket() error %d\n",WSAGetLastError());  
    return -1;  
}
```

```
//установка соединения
```

```
dest_addr.sin_family=AF_INET;  
dest_addr.sin_port=htons(PORT);
```

```
// преобразование IP адреса из символического в сетевой формат
if (inet_addr(SERVERADDR)!=INADDR_NONE)
    dest_addr.sin_addr.s_addr=inet_addr(SERVERADDR);
else
// получение IP адреса по доменному имени сервера
if (hst=gethostbyname(SERVERADDR))
    ((unsigned long *)&dest_addr.sin_addr)[0]=
    ((unsigned long **)hst->h_addr_list)[0][0];
else
{
    printf("Invalid address %s\n",SERVERADDR);
    closesocket(my_sock);
    WSACleanup();
    return -1;
}
```

//установка соединения

```
if (connect(my_sock,(struct sockaddr *)&dest_addr,  
           sizeof(dest_addr))){  
    printf("Connect error %d\n",WSAGetLastError());  
    return -1;  
}  
printf("Connection with %s was established\n\  
      Type quit for quit\n\n",SERVERADDR);
```

// чтение и передача сообщений

```
while((nsize=recv(my_sock, &buff[0], sizeof(buff)-1,0))  
      !=SOCKET_ERROR){  
    buff[nsize]=0;  
    printf("ServerToClient:%s",buff);  
  
    printf("ClientToServer:");  
    fgets(&buff[0], sizeof(buff) - 1, stdin);
```

```
if (!strcmp(&buff[0], "quit\n")) {  
    printf("Exit...");  
    closesocket(my_socket);  
    WSACleanup();  
    return 0;  
}
```

```
send(my_socket, &buff[0], nsize, 0);  
}
```

```
printf("Recv error %d\n", WSAGetLastError());  
closesocket(my_socket);  
WSACleanup();
```

```
return -1;
```

```
}
```


Упражнение 3: протестировать разобранные сетевые программы.

Тема курсовой №4: написать многопользовательский «чат» на основе сокетов (после темы Многопоточные приложения в MS Windows).

Тема курсовой №5: написать многопользовательский http-сервер (web-сервер) (после темы Многопоточные приложения в MS Windows).