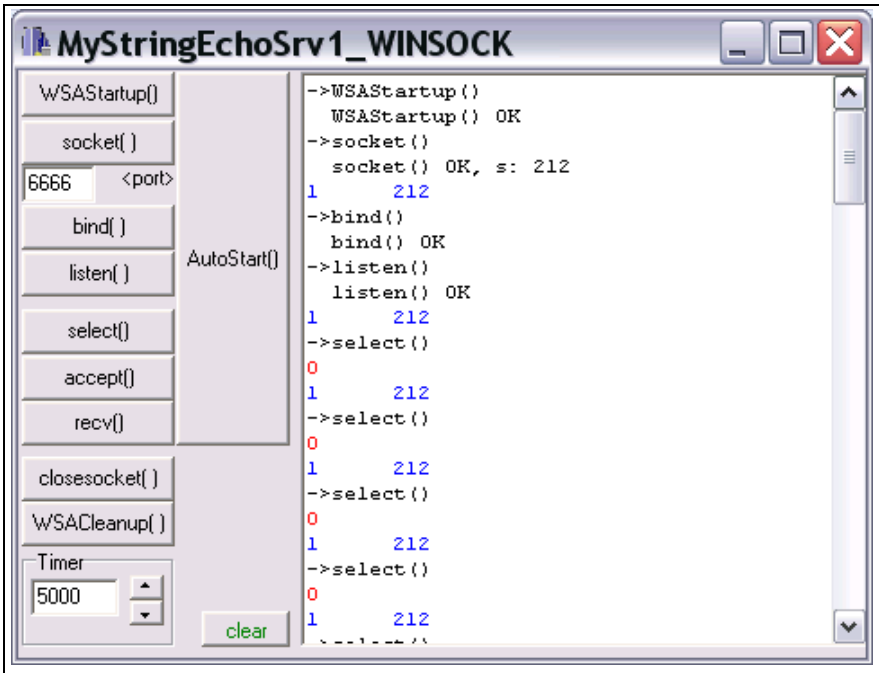
1.1 Ziel des Projektes

- Es soll ein String-Echo-Server auf WINSOCK-API aufgesetzt werden.
- Der Srv-Socket nimmt kommende Verbindungen an, empfängt Daten und sendet diese zurück.
- Der Socket soll blockierend arbeiten. Nach dem Aufruf der Funktion listen() wartet der Srv-Socket auf ankommende Verbindungswünsche von Clients, indem er die Funktion accept() aufruft. →Kommt z.B. kein Verbindungswunsch an, blockiert der Socket, die Serveranwendung kann dann nicht normal terminieren.
- Deshalb wird vor dem Aufruf der blockierenden Funktionen accept(), recv() mittels der NICHT-BLOCKIERENDEN FUNKTION select() geprüft, ob ein Accept- oder Read-Ereignis vorliegt.
- Eingangs zu diesem Beispiel werden select(), die Struktur fd_set und Funktionen auf fd_set nochmals besprochen. Programmierbeispiele sind dazu MyFD_SET1 und MyFD_SET2.

1.2 Grundlagen zu select()

Siehe Anlage A.

1.3 Realisierung des Projektes MyStringEchoSrv1_WINSOCK



- Erzeugen Sie ein neues Projekt "MyStringEchoSrv_WINSOCK" im gleichnamigen Order.
- Editieren Sie die Oberfläche entsprechend der nebenstehenden Abbildung.
- Erzeugen Sie Schritt für Schritt den Programmcode, entsprechend dem Beispiel-Code.

Autostart ruft die Funktionen WSAStartup(), socket(), bind(), listen(), select() auf.
Es wird ein die Klasse TTimer verwendet. Läuft der Timer ab, wird die Funktion select() aufgerufen.

☝ Beachten Sie: der mit Zeilennummern versehene Programmcode wird im Abschnitt 1.4 kommentiert.

```
//---------------------------------------------------------------------------
#include <vcl.h>
#include <winsock2.h> //erforderlich
#include <string>
#pragma hdrstop
#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
u_int s, as;                                    //socket descriptor
int ret_len;                                    //für send() und recv()
char recv_buf[4095];                            //recv()
struct sockaddr_in xAddr; int xAddrLen; //für bind(),accept()
int spalte,zeile;
fd_set readfds; //struktur für select()
```

```
fd_set mySocks; //lokale struktur für alle aktiven sockets, auch srvsocket
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
            : TForm(Owner)
{
 Edit1->Text=IntToStr(this->Timer1->Interval);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::WSAStartup_Click(TObject *Sender)
{
 WSADATA wsaD;
//an DLL anmelden
  RichEdit1->Lines->Add("->WSAStartup()");
  FD_ZERO(&mySocks);
  if  (WSAStartup (MAKEWORD(2,0),&wsaD) ==0) //MAKEWORD(maj,min)
      RichEdit1->Lines->Add("  WSAStartup() OK");
  else
      RichEdit1->Lines->Add("  WSAStartup() ERR");
}
//---------------------------------------------------------------------------
void __fastcall TForm1::socket_Click(TObject *Sender)
{
   RichEdit1->Lines->Add("->socket()");
  s = socket(AF_INET,SOCK_STREAM,0);
  if (s!=0){
      RichEdit1->Lines->Add("  socket() OK, s: "+IntToStr(s));
//den sockDescriptor des srvSock in mySocks eintragen
      FD_SET(s,&mySocks);
      Zeige_mySocks(Form1);
      }
  else
      RichEdit1->Lines->Add("  socket() ERR");
}
//---------------------------------------------------------------------------
void __fastcall TForm1::bind_Click(TObject *Sender)
{
   xAddr.sin_family=AF_INET;
   xAddr.sin_port=htons(port->Text.ToInt());
   xAddr.sin_addr.S_un.S_addr=inet_addr("0.0.0.0");
   RichEdit1->Lines->Add("->bind()");
      if (bind (s,(struct sockaddr*)&xAddr,sizeof(sockaddr))==0)
       {
       RichEdit1->Lines->Add("  bind() OK");
     as=s;//srvSocketDescriptor in as
            }
      else
       RichEdit1->Lines->Add("->connect() ERR: "+IntToStr(WSAGetLastError()));
}
//---------------------------------------------------------------------------
1   void __fastcall TForm1::listen_Click(TObject *Sender)
2   {
3   RichEdit1->Lines->Add("->listen()");
4   if (listen(s,5)==0)
5   RichEdit1->Lines->Add("  listen() OK");
6   else RichEdit1->Lines->Add("  listen() ERR: "+IntToStr(WSAGetLastError()));
7   }
//---------------------------------------------------------------------------
8   void __fastcall TForm1::accept_Click(TObject *Sender)
9   {
10    xAddrLen=sizeof(xAddr);
11    RichEdit1->Lines->Add("->accept()");
12    as=accept(s,(struct sockaddr*) &xAddr, &xAddrLen);
13    RichEdit1->Lines->Add("accepted :"+ IntToStr(as));
14    if (as!=SOCKET_ERROR) {
15    FD_SET(as,&mySocks);
16    Zeige_mySocks(Form1);
17    }
18  }
```

```
//---------------------------------------------------------------------------
19  void __fastcall TForm1::recv_Click(TObject *Sender)
20  {
21     RichEdit1->Lines->Add("->recv()");
22     ret_len = recv(as, recv_buf, sizeof(recv_buf),0);
23     switch (ret_len) {
24     case -1 : ;break;
25     case 0  : {FD_CLR(as,&mySocks);
26                       RichEdit1->Lines->Add("closed: "+(AnsiString)as);
27                       break; }
28     default : {
29                       AnsiString text=((AnsiString)recv_buf).SubString(0,ret_len);
30                       RichEdit1->Lines->Add("received: "+text);
31                       RichEdit1->Lines->Add("->send(): "+text);
32                       ret_len=send(as,recv_buf,ret_len,0);
33                        }
34     }
35  }
//---------------------------------------------------------------------------
36  void __fastcall TForm1::select_Click(TObject *Sender)
37  {   int retWert,i;
38          FD_ZERO(&readfds);//löschen
39          readfds=mySocks;  //alle aktiven Socks eintragen
40          timeval time;     //blockierungszeit für select() festlegen
41          time.tv_sec=1; time.tv_usec=0;
42          RichEdit1->SetFocus();
43          Zeige_mySocks(Form1);
44          RichEdit1->Lines->Add("->select()");
45          retWert=select(0,&readfds,NULL,NULL,&time);
46          Zeige_readfds(Form1);
47  //retWert=-1: Serversocketfehler
48          if (retWert==-1) {RichEdit1->Lines->Add("Error:
                              "+IntToStr(WSAGetLastError()));goto stop;}
49  //retWert=0: keine Aktionen erforderlich
50          if (retWert==0)  {goto stop;}
51  //retWert>1: wenn ServerSocket in readfds dann accept(), für andere recv()
52          if (retWert>0)
53            {
54            for (i=0; i<retWert ; i++)
55                {
56                if (readfds.fd_array[i]==s) accept_Click(Form1);
57                  if ((readfds.fd_array[i]!=s)&&(readfds.fd_array[i] >0))
58                    { as=readfds.fd_array[i]; recv_Click(Form1);}
59                }//end-for
60            }//end-if
61   stop:
62  }
//---------------------------------------------------------------------------
63  void __fastcall TForm1::closesocket_Click(TObject *Sender)
64  {  int x;
65     if (mySocks.fd_count >0) //--alle sockets schließen
66    {
67     for (int i= mySocks.fd_count; i>0 ; i--)
68      {
69      if (closesocket(mySocks.fd_array[i-1] )!=SOCKET_ERROR)
70      { FD_CLR(mySocks.fd_array[i-1],&mySocks);
71          RichEdit1->Lines->Add("->closesocket()");  }
72      else {RichEdit1->Lines->Add("  closesocket() ERR: "
73            +IntToStr(WSAGetLastError()));}
74      }//end-for
75    Timer1->Enabled=False;
76    }
} //---------------------------------------------------------------------------
void __fastcall TForm1::WSACleanup_Click(TObject *Sender)
{
       Timer1->Enabled=False;
       RichEdit1->Lines->Add("->WSACleanup()");
       WSACleanup();
```

```
       RichEdit1->Lines->Add("  WSACleanup() OK");
}
//---------------------------------------------------------------------------
void __fastcall TForm1::clear_Click(TObject *Sender)
{
   RichEdit1->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::automatisch_Click(TObject *Sender)
{
        WSAStartup_Click(Form1);
        socket_Click(Form1);
        bind_Click(Form1);
        listen_Click(Form1);
        Timer1->Enabled=True;
}
//---------------------------------------------------------------------------


void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
        Edit1->SetFocus();
        Timer1->Enabled=True;
        select_Click(Form1);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Zeige_mySocks(TObject *Sender)
{   //Anzeige per BitBtn Zeige_mySocks
      RichEdit1->SelAttributes->Color=clBlue;
      AnsiString txt;
      txt=IntToStr(mySocks.fd_count);
            if (mySocks.fd_count!=0) {
                  for (u_int i=0;i<mySocks.fd_count;i++ ) {
                  txt=txt+"\t"+IntToStr(mySocks.fd_array[i]);
                  }
            }
      RichEdit1->Lines->Add(txt);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Zeige_readfds(TObject *Sender)
{   //Anzeige von readfds nach select()
      RichEdit1->SelAttributes->Color=clRed;
      AnsiString txt;
      txt=IntToStr(readfds.fd_count);
            if (readfds.fd_count!=0) {
                  for (u_int i=0;i<readfds.fd_count;i++ ) {
                  txt=txt+"\t"+IntToStr(readfds.fd_array[i]);
                  }
            }
      RichEdit1->Lines->Add(txt);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  if (Button == 0) Timer1->Interval= Timer1->Interval+1000;
  if (Button==1) {Timer1->Interval= Timer1->Interval-1000; }
  Edit1->Text=IntToStr(Timer1->Interval);
}
//---------------------------------------------------------------------------
```

1.4 Kommentierung der Funktionen listen(), accept(), recv(), send(), select() und closesocket()

```
//---------------------------------------------------------------------------
1   void __fastcall TForm1::listen_Click(TObject *Sender)
2   {
3   RichEdit1->Lines->Add("->listen()");
4   if (listen(s,5)==0)
5   RichEdit1->Lines->Add("  listen() OK");
6   else RichEdit1->Lines->Add("  listen() ERR: "+IntToStr(WSAGetLastError()));
7   }
```

4 durch listen(s,5)wird der Socket mit Descriptor s zum Serversocket. Der Parameter 5 gibt an, dass am ServerSocket bis zu 5 Sockets warten können, während dieser einen Verbindungswunsch akzeptiert.

```
//---------------------------------------------------------------------------
8   void __fastcall TForm1::accept_Click(TObject *Sender)
9   {
10    xAddrLen=sizeof(xAddr);
11    RichEdit1->Lines->Add("->accept()");
12    as=accept(s,(struct sockaddr*) &xAddr, &xAddrLen);
13    RichEdit1->Lines->Add("accepted :"+ IntToStr(as));
14    if (as!=SOCKET_ERROR) {
15    FD_SET(as,&mySocks);
16    Zeige_mySocks(Form1);
17    }
18  }
19  }
```

12 der ServerSocket akzeptiert neue Verbindungen. accept() ist eine blockierende Funktion.
14, 15 wenn as!=SOCKET_ERROR ist, dann wird der SocketDescriptor in die Struktur mySocks eingetragen. In mySocks stehen die SocketDescriptoren aller angenommener Verbindungen. Dieses Wissen wird in der Funktion select() benötigt um abzufragen, welcher Socket lesbar ist.

```
//---------------------------------------------------------------------------
20  void __fastcall TForm1::recv_Click(TObject *Sender)
21  {
22     RichEdit1->Lines->Add("->recv()");
23     ret_len = recv(as, recv_buf, sizeof(recv_buf),0);
24     switch (ret_len) {
25     case -1 : ;break;
26     case 0  : {FD_CLR(as,&mySocks);
27                     RichEdit1->Lines->Add("closed: "+(AnsiString)as);
28                     break; }
29     default : {
30                AnsiString text=((AnsiString)recv_buf).SubString(0,ret_len);
31                     RichEdit1->Lines->Add("received: "+text);
32                     RichEdit1->Lines->Add("->send(): "+text);
33                     ret_len=send(as,recv_buf,ret_len,0);
34                      }
35     }
36  }
```

23 die blockierende Funktion recv() wird aufgerufen. In as steht der Socketdescriptor.
24 Es wird untersucht, welchen Rückgabewert recv() lieferte.
25 Ist er -1 liegt ein Fehler vor. Die Fehlerauswertung ist hier weggelassen worden.
26 Ist der Wert 0, wurde die Verbindung abgebaut. Der SocketDescriptor muss aus mySocks gelöscht werden
29 … der empfangene Text wird mittels send() zurück gesendet.

```
//---------------------------------------------------------------------------
37  void __fastcall TForm1::select_Click(TObject *Sender)
38  {   int retWert,i;
39          FD_ZERO(&readfds);//löschen
40          readfds=mySocks;  //alle aktiven Socks eintragen
41          timeval time;     //blockierungszeit für select() festlegen
42          time.tv_sec=1; time.tv_usec=0;
43          RichEdit1->SetFocus();
44          Zeige_mySocks(Form1);
45          RichEdit1->Lines->Add("->select()");
46          retWert=select(0,&readfds,NULL,NULL,&time);
47          Zeige_readfds(Form1);
48  //retWert=-1: Serversocketfehler
49          if (retWert==-1) {RichEdit1->Lines->Add("Error:
                              "+IntToStr(WSAGetLastError()));goto stop;}
50  //retWert=0: keine Aktionen erforderlich
51          if (retWert==0)  {goto stop;}
52  //retWert>1: wenn ServerSocket in readfds dann accept(), für andere recv()
53          if (retWert>0)
54            {
55            for (i=0; i<retWert ; i++)
56                {
57                if (readfds.fd_array[i]==s) accept_Click(Form1);
58                else  {as=readfds.fd_array[i];recv_Click(Form1);}
59                }//end-for
60            }//end-if
61   stop:
62  }
```

39, 40 die Struktur read_fds wird gelöcht und anschließend dieser die Einträge aus mySocks zugewiesen
41, 42 select() kann man durch eine Variable vom Typ timeval mitteilen, wie lange die Funktion auf Aktionen wartet, bevor sie zurück kehrt. Hier wird 1sec festgelegt. Liegen aber beim Aufruf von select() schon Ereignisse vor, kehrt die Funktion sofort zurück. Übergibt man einen Wert 0, blockiert die Funktion.
46 select() wird aufgerufen, &read_fds ist die Adresse auf eine Struktur fd_set in der alle SocketDescriptoren stehen, die auf Lesbarkeit geprüft werden sollen. NULL, NULL meint, es sollen keine Sockets auf Beschreibbarkeit bzw. Ausnahmen geprüft werden
48 Ist der Wert -1, ligt ein ServerSocketFehler vor, der wird angezeigt.
50 Ist der Wert 0, ligt für keinen der Sockets etwas vor
53 ist der retWert 1 oder größer, sind 1 oder mehrere Sockets lesbar
55 in der for-Schleife werden alle lesbaren Sockets ausgewertet, es wird accept() oder recv()aufgerufen
57 ein lesbarer Socket ist der Serversocket. Deshalb wird acceptClick() aufgerufen.
58 ein lesbarer Socket ist nicht der Serversocket. Deshalb wird recvClick() aufgerufen.

```
    //---------------------------------------------------------------------------
63  void __fastcall TForm1::closesocket_Click(TObject *Sender)
64  {  int x;
65     if (mySocks.fd_count >0) //--alle sockets schließen
66    {
67     for (int i= mySocks.fd_count; i>0 ; i--)
68      {
69      if (closesocket(mySocks.fd_array[i-1] )!=SOCKET_ERROR)
70      { FD_CLR(mySocks.fd_array[i-1],&mySocks);
71          RichEdit1->Lines->Add("->closesocket()");  }
72      else {RichEdit1->Lines->Add("  closesocket() ERR: "
73            +IntToStr(WSAGetLastError()));}
74      }//end-for
75    Timer1->Enabled=False;
76    }
```

65, … wenn es noch aktive Sockets gibt, müssen diese erst geschlossen werden, bevor der ServerSocket geschlossen wird. Beachte: in mySocks stehen alle aktiven Sockets. In mySocks.fd_array[0] steh der Serversocket, dieser muss zum Schluss geschlossen werden. Deshalb geht die for-Schleife von oben nach unten!

ANLAGE A: Details zu fd_set, select(), FD_CLEAR, FD_SET, …

Mittels select() kann man vor dem Aufruf einer blockierenden Funktion prüfen, ob ein relevantes Ereignis vorliegt. Wenn ja, ruft man diese blockierenden Funktionen auf, sonst nicht. Wenn man an einem blockierenden Serversocket beispielsweise accept() aufruft und es kommt kein Verbindungswunsch an, blockiert die Anwendung bist zum jüngsten Tag. Gleiches gilt für recv() usw.

```
int WSAAPI     select (
               IN        int                 nfds,
               IN OUT    fd_set FAR*         readfds,
               IN OUT    fd_set FAR*         writefds,
               IN OUT    fd_set FAR*         exceptfds,
               IN        const struct timeval FAR*    timeout);
```

Mittels select() kann man ermitteln, für welche Sockets Ereignisse vorliegen. select() nutzt dazu Strukturen vom Typ fd_set

nfds	bei WSAAPI =0
readfds	Zeiger auf Struktur vom Typ fd_set (file descriptor set), welche Sockets sind lesbar?
writefds	Zeiger auf Struktur vom Typ fd_set welche Sockets sind beschreibbar?
exceptfds	Zeiger auf Struktur vom Typ fd_set, für welche Sockets liegt Ausnahme vor
timeout	Zeiger auf Struktur vom Typ timeval; Zeit in sec:msec, die man warten will
Return Value int	wenn **0** =für keinen der Sockets liegt etwas vor wenn >0 =Anzahl der Sockets, die gelesen, akzeptiert werden müssen oder ein Fehler vorliegt wenn **-1** =Fehler, mit **int WSAAPI WSAGetLastError()** Nummer des Fehlers ermitteln.

C	int retWert,i;
	fd_set **readfds**, **mySocks**; //array für alle lesbaren und aktiven sockets anlegen
	timeval time; //blockierungszeit für select() festlegen
	time.tv_sec=0; time.tv_usec=1000;
	readfds = **mySocks**; //alle aktiven Socks aus mySock in readfds eintragen
	retWert=**select**(0,&**readfds**,NULL,NULL,&time);
	//--auswertung, ob accept() und/oder recv() ausgeführt werden muß
	if (retWert==-1){ anzErr());goto stop;}
	if (retWert==0) { goto stop;}
	if (retWert>0) { for (i=0; i<retWert ; i++)
	{ if (**readfds**.fd_array[i]==srvSock)**accept**();
	if ((**readfds**.fd_array[i]!=s)&&(**readfds**.fd_array[i] >0))
	{ as=**readfds**.fd_array[i]; **recv**(as);}
	}
	}

fd_set

The **fd_set** structure is used by various Windows Sockets functions and service providers, such as the **select** function, to place sockets into a "set" for various purposes, such as testing a given socket for readability using the *readfds* parameter of the **select** function.

```
typedef struct fd_set {
  u_int fd_count;                 // Number of sockets in the set
  SOCKET fd_array[FD_SETSIZE];    // Array of sockets that are in the set, FD_SETSIZE=64
} fd_set;
```

→ Diese Struktur besteht aus einem Zähler und einem Array für SocketDescriptoren (auch FileDescriptoren genannt). Dieses Array hat eine Standardgröße von 64 .

Damit man mit diesen Strukturen vernünftig arbeiten kann, werden einige nützliche Funktionen durch das API bereitgestellt: **FD_ZERO(), FD_SET(), FD_CLR(), FD_ISSET().**

BEISPIELE für Umgang mit Strukturen vom Typ fd_set:

```
fd_set mySocks;         //Erzeugen der Variablen mySocks
FD_ZERO(&mySocks);      //in mySocks steht: 0  0      0      0      0      ...
FD_SET(1800,&mySocks); //in mySocks steht: 1  1800   0      0      0      ...
FD_SET(1801,&mySocks); //in mySocks steht: 2  1800   1801   0      0      ...
FD_SET(1802,&mySocks); //in mySocks steht: 3  1800   1801   1802   0      ...

int i=FD_ISSET(1802,&mySocks); //i=1, da 1802 im SET enthalten ist
int i=FD_ISSET(1810,&mySocks); //i=0, da 1810 nicht im SET enthalten ist

FD_CLR(1800,&mySocks); //in mySocks steht: 2  1801   1802   1802   0      ...
FD_CLR(1802,&mySocks); //in mySocks steht: 1  1801   1802   1802   0      ...
```

→siehe auch Programmbeispiel MyFD_SET1/2 ..

In select() können drei Strukturen übergeben werden (siehe Funktionsbeschreibung):
- readfds zum Prüfen ob aktive Sockets lesbar sind,
- writwfds zum Prüfen ob aktive Sockets beschreibbar sind
- exceptfds zum Prüfen ob aktive Sockets Fehler/Ausnahmen melden.

Bei select() wird hauptsächlich nur die Struktur readfds genutzt.

		Inhalt von mySocks
1	**fd_set mySocks;** **timeval time;** **time.tv_sec=0; time.tv_usec=200;**//Blockierungszeit für select()	0 0 0 …
2	**int sd=socket(AF_INET,SOCK_STREAM,0);** **FD_SET(sd,&mySocks);**	1 1800 0 …

3	**int i=select(0,&mySocks,NULL,NULL,&time);**
	//wenn i=0, kein Verbindungswunsch
	//wenn i=1, ein ankommender Verbindungswunsch liegt vor, accept() aufrufen
		1 1800 0 …
		0 1800 0 …
		1 1800 0 …

4	**int as=accept(s,(struct sockaddr*) &xAddr, &xAddrLen);** **FD_SET(as,&mySocks);**	2 1800 1802 …

5	**int i=select(0,&mySocks,NULL,NULL,&time);**
	//wenn i=0, kein Verbindungswunsch, nichts zu lesen
	//wenn i=2, ein ankommender Verbindungswunsch, ein Socket zu lesen
	//wenn i=1, und 1800, Verbindungswunsch liegt vor
	//wenn i=1, und 1802, Socket ist lesbar
		2 1800 1802 …
		0 1800 1802 …
		2 1800 1802 …
		1 1800 1802 …
		1 1802 1802 …