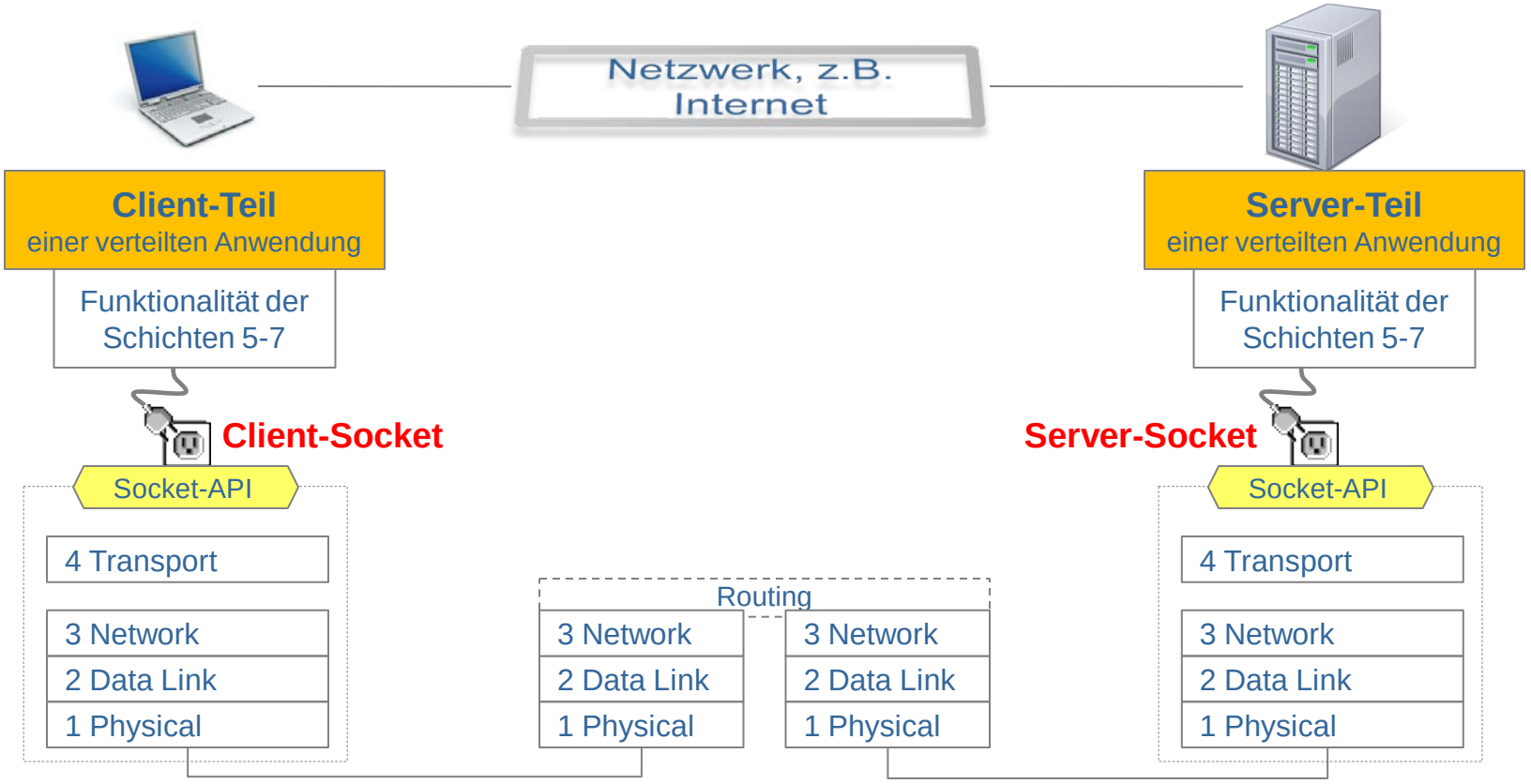
© Prof. Dr.-Ing. habil. Lutz Winkler, Hochschule Mittweida (FH) – University of Applied Sciences, Fakultät Informationstechnik & Elektrotechnik
win@hs-mittweida.de http://www.telecom.hs-mittweida.de

- Ziel der Vorlesung:
  - Vermittlung von Grundkenntnissen zum Socket-API und dessen Nutzung unter Windows.
  - Anhand von Programmierbeispielen auf BSD-Niveau und unter Nutzung von Socket-Komponenten werden wesentliche Konzepte praktisch vertieft.
  - Es werden vorwiegend Konzepte erläutert, und nicht deren minimalistische Programmierung.
- Inhaltsübersicht der Vorlesung:

Übersicht - Was ist eine Socket?

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

- Socket's (Steckdose, Anschluss) sind **Kommunikationsendpunkte**, mittels derer verteilte Anwendungen (z.B. Webbrowser mit Webserver oder E-Mail-Client mit E-Mail-Server) Transportdienste (Schichten 1 bis 3/4) von Kommunikationsnetzen nutzen können.
- Der Serverteil einer Anwendung errichtet einen Serversocket, der auf Assoziationen wartet, der Clientteil errichtet einen Client-Socket, der Assoziationen zu einem Serversocket aktiv herstellt..



Übersicht - Was ist eine Socket?

- Socket-APIs können verschiedene Netzwerkprotokolle unterstützten:
  - **AF_INET** →`Address Family InterNET, Version 4` `//dominante Anwendung`
  - AF_INET6 →`Address Family InterNET, Version 6`
  - AF_UNIX, →`Address Family UNIX`
  - AF_IPX, ….
- Sockets sind für viele Betriebssysteme verfügbar. Die Funktionen werden über ein API (application programming interface) bereit gestellt:
  - in UNIX, LINUX als Teil des Betriebssystems,
  - in Windows als Bibliothek - winsock.dll, wsock32.dll usw.
- Unterscheidung:
  - ServerSocket →wartet auf Assoziationen,
  - Clientsocket →stellt Assoziationen aktiv zu ServerSockets her.
- Die Socketnutzung basiert auf Unix-Ein-Ausgabe-Paradigma:
  open → read/write →close
- Das Windows-API, welches Sockets unterstützt, heißt:
  →Windows-Sockets-API (WSA, WS-API)

- Socket-API entstand 1982 (University of California at Berkeley) als Teil von BSD[1)]-Unix.
- Windows-Socket-API (WSA) unterstützt alle BSD-Socket-Funktionen (mit marginalen Unterschieden) und darüber hinaus weitere Funktionen.
- WSA-Versionen:
  - 1992-WSA 1.0
  - 1993-WSA 1.1
  - 1995-WSA 2.0
  - 1997-WSA 2.2.2 →Standard umfasst ca. 300 Seiten

- DLL‘s:
  - **Version 1.1**:
    - Für 16-Bit-Systeme (Windows 3.11) …… →**winsock.dll**
    - Für 32-Bit-Systeme (W’95, W’98, W‘NT, W‘XP)… →**wsock32.dll**
  - **Version 2**:
    - Für 16-Bit-Systeme ……..…….. →**ws2_16.dll**
    - Für 32-Bit-Systeme ……..…….. →**ws2_32.dll**

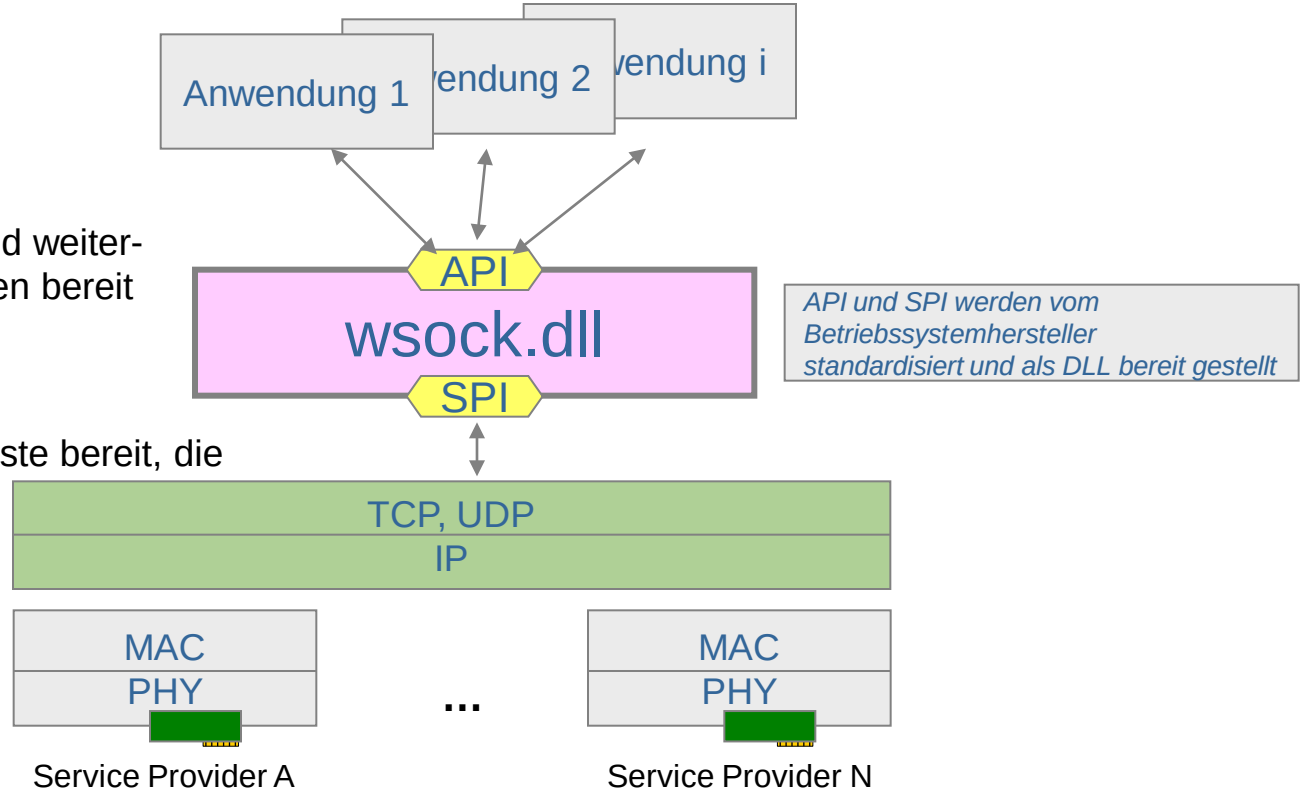
1) **BSD - Berkeley Software Distribution** (**BSD**), ist eine Version des Betriebssystems Unix

- WOSA allgemein→DLL mit zwei Interfaces:
  - **API** (Application Programming Interface),
    →Standard[1)] des Betriebssystemherstellers für Anwendungsprogrammierer.
  - **SPI** (Service Provider Interface),
    →Standard[1)] des Betriebssystemherstellers für Dienstbereitsteller.

- WOSA für Sockets:

Windows Sockets 2 API:
Stellt BSD-Socket-Funktionen und weitergehende (WSA-) Socketfunktionen bereit

Windows Sockets 2 SPI:
Hier stellen Serviceprovider Dienste bereit, die dem WSOCK-SPI genügen.



1) Diese Standards können Sie downloaden von www.telecom.hs-mittweida.de >Lehre DI-Direktstudium

0.1. Document Organization

The complete Windows Sockets 2 specification consists of four separate documents:

1. ***Windows Sockets 2 Application Programming Interface*** →***ca. 300 Seiten***
2. ***Windows Sockets 2 Protocol-Specific Annex***
3. ***Windows Sockets 2 Service Provider Interface***
4. ***Windows Sockets 2 Debug-Trace DLL***

This document (***Windows Sockets 2 Application Programming Interface***) is divided into four main sections and four appendices.

Section 1	Introductory material about the specification as a whole
Section 2	Summary of additions and changes in going from Windows Sockets 1.1 to Windows Sockets 2
Section 3	Windows Sockets Programming Considerations
Section 4	Detailed reference information for the data transport functions
Section 5	Introductory material and detailed reference information for the name resolution and registration functions
Appendix A	Information on WinSock header files, error codes and data type definitions
Appendix B	Details on multipoint and multicast features in Windows Sockets 2
Appendix C	The WinSock Lame List
Appendix D	A bibliography for those seeking additional information

The ***Windows Sockets 2 Protocol-Specific Annex*** contains information specific to a number of transport protocols that are accessible via Windows Sockets 2.
The ***Windows Sockets 2 Service Provider Interface*** specifies the interface that transport providers must conform to in order to be accessible via Windows Sockets 2.
Windows Sockets 2 Debug-Trace DLL describes the files and mechanics of the debug-trace DLL. This is a useful tool for application developers and service providers alike, that shows API and SPI calls in and out of the WinSock 2 DLL..

WinSock-2-Architektur

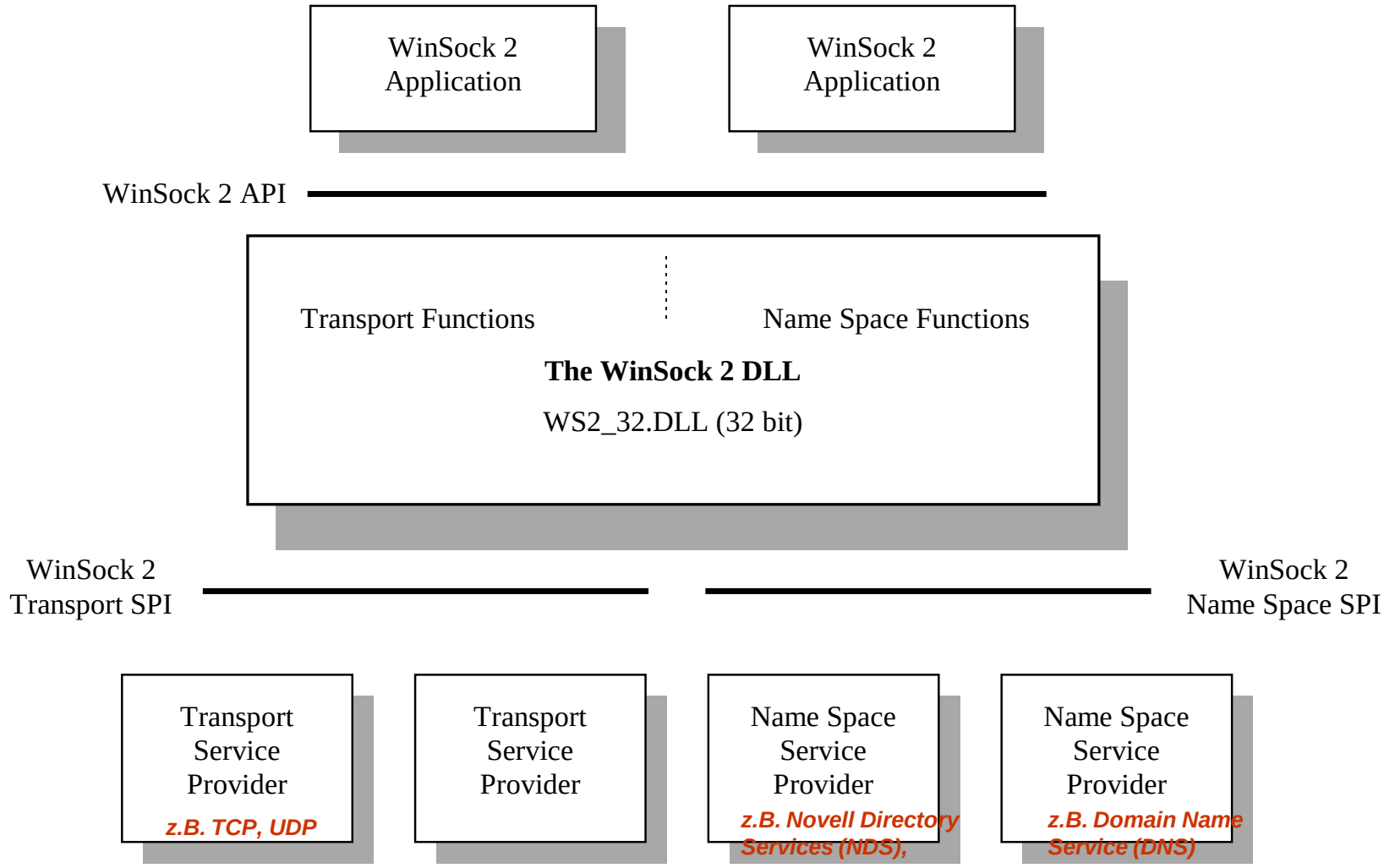


Figure 1 WinSock 2 Architecture,
Windows Sockets 2, Application Programming Interface, Revision 2.2.2, August 7, 1997

WinSock-1.1.-Architektur

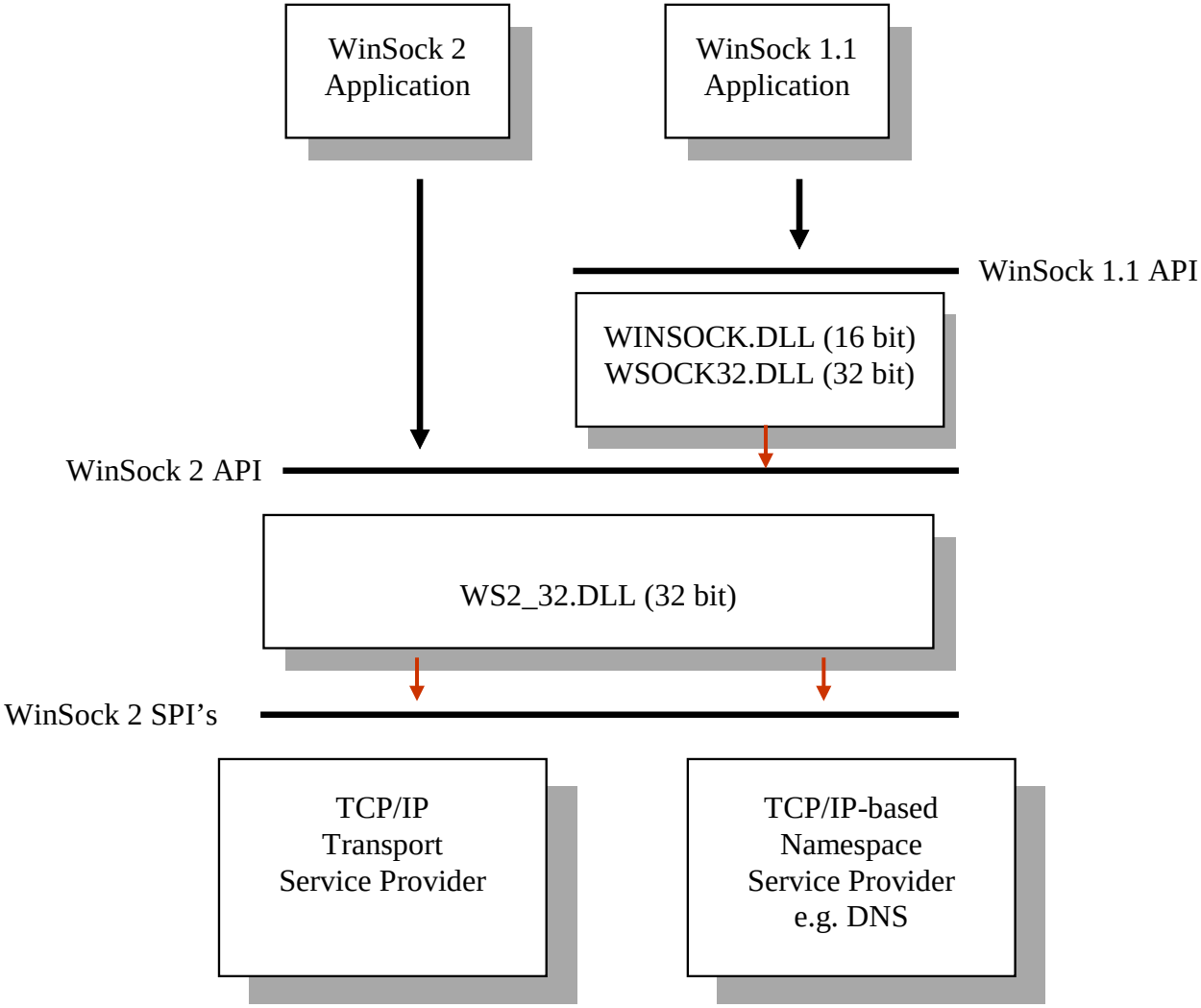
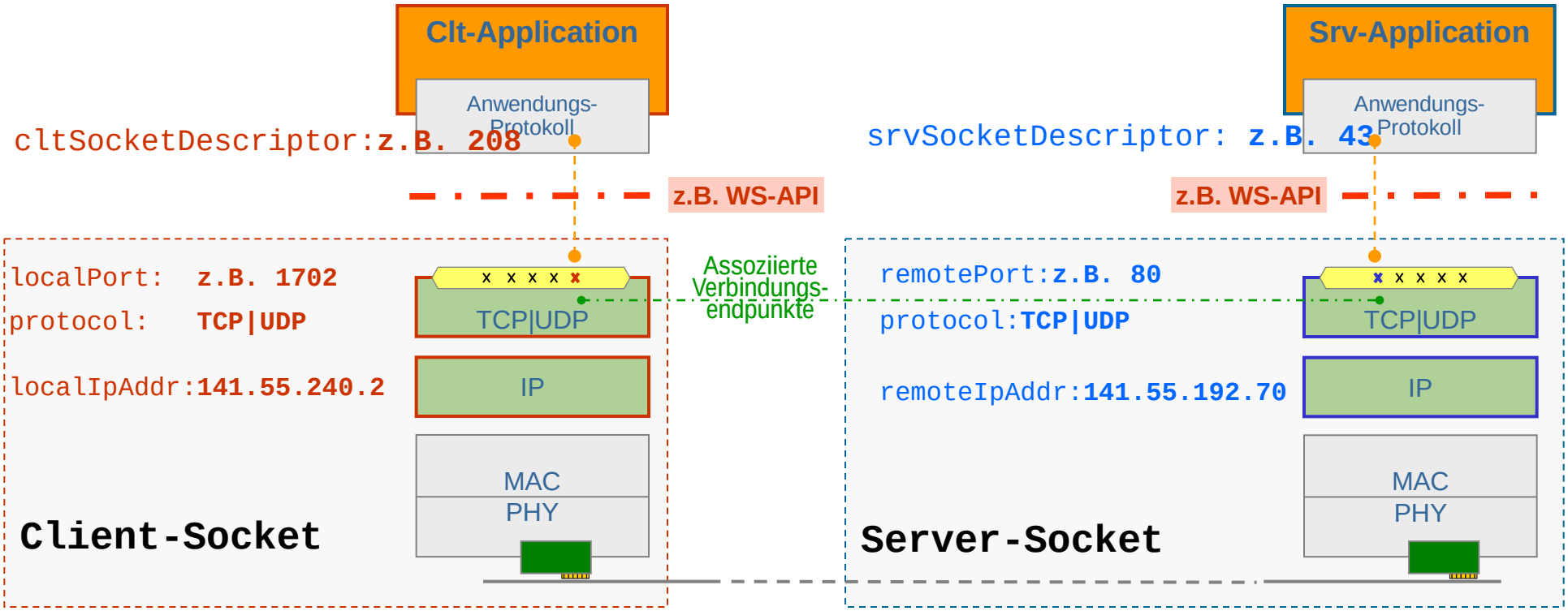


Figure 2 WinSock 1.1 Compatibility Architecture,
Windows Sockets 2, Application Programming Interface, Revision 2.2.2, August 7, 1997

- Anwendungsteile referenzieren die Assoziation durch einen →socketDescriptor.
- Der Kommunikationsstack referenziert die Assoziation durch Verbindungsendpunkte (Sockets), beschrieben durch das Quadtupel:

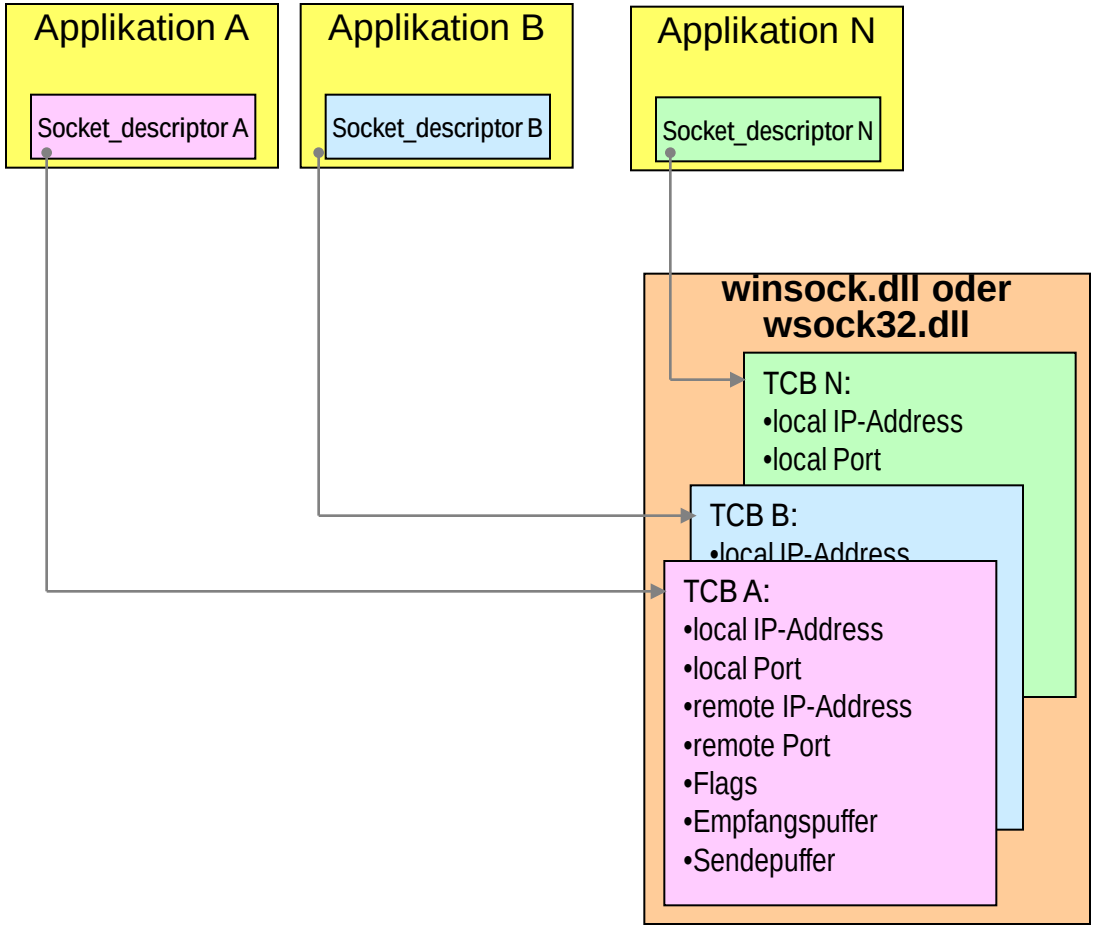
<VerbindungsTyp>	
<localPort>	<remotePort>
<localIpAddress>	<remoteIpAddress>



Begriffe: Socket Descriptor, Transmission Control Block

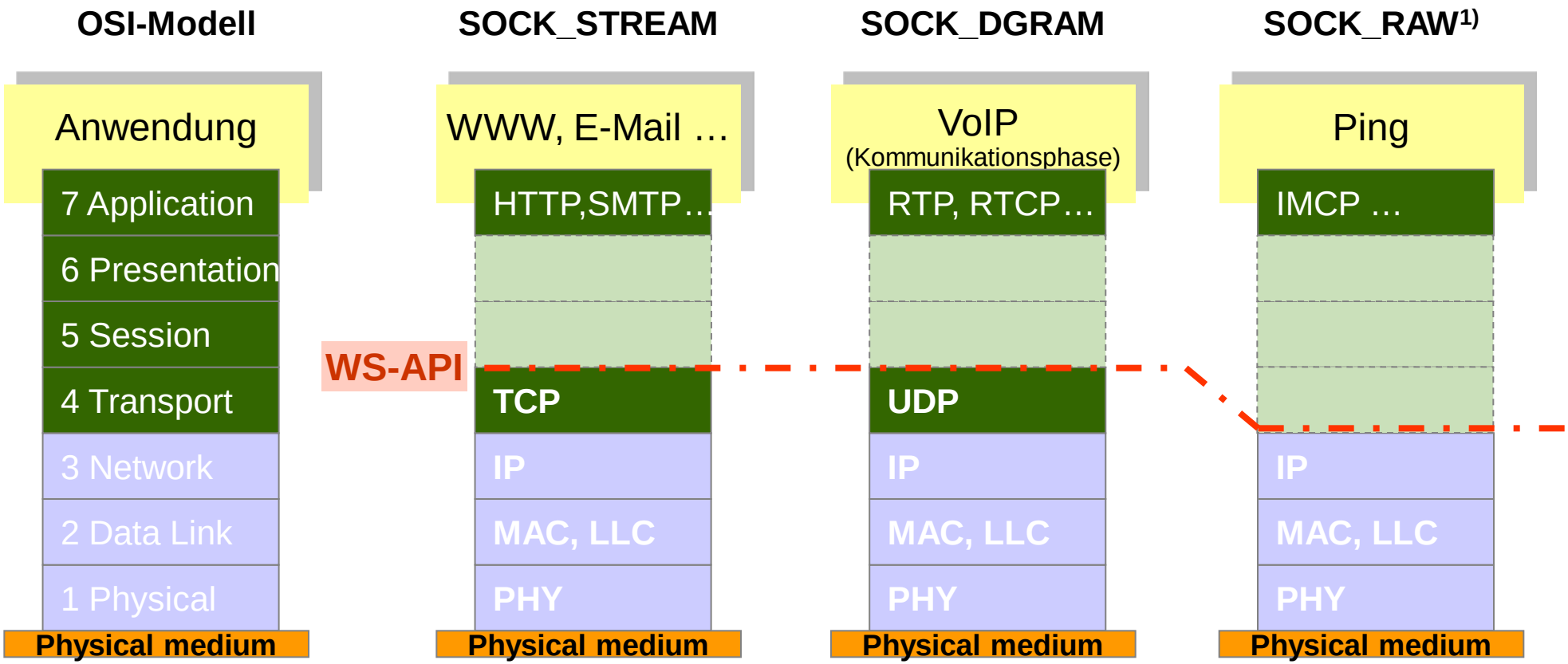
Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

- Ein Socket Call gibt einen u_int-Wert zurück: socketDescriptor (socketHandle)
- Mit dem socketDescriptor kann man den so genannten transmission control block (TCB) referenzieren.
- TCB hat u.a. folgende Parameter:
  - Local IP address
  - Local port
  - Remote IP address
  - Remote Port
  - Protocol (TCP, UDP)
  - Send buffer size
  - Receive buffer size
  - Bei TCP, current state usw.



Begriffe: Verbindungstypen

- Für AF_INET werden 3 Verbindungstypen unterstützt
  - Applikation-TCP-IP-Netzwerkinterface →SOCK_STREAM
  - Applikation-UDP-IP-Netzwerkinterface →SOCK_DGRAM
  - Applikation-IP-Netzwerkinterface →SOCK_RAW



1) WINSOCK-Service-Provider können, aber müssen diesen Verbindungstyp nicht anbieten. Also probieren ,ob dieser Typ unterstützt wird.

Begriffe: Verbindungstypen

- **SOCK_STREAM**:
  - Der Ablauf: →Socket errichten →Verbindungsherstellung →Kommunikation →Verbindungsabbau
  - Zwischen den Anwendungen existiert eine verbindungsorientierte Assoziation (connection-oriented - co).
  - Eigenschaft:
    - Bytes werden sequenzweise, basierend auf TCP, übertragen.
    - Jede Sequenz wird nummeriert und vom Partner quittiert. Bleibt Quittung aus, werden Daten erneut gesendet.
    - Anhand der Sequenznummern kann die ursprüngliche Reihenfolge hergestellt werden.
    - Verbindungsauf-, Verbindungsabbau sowie Quittierung verringern den Durchsatz, aber alles kommt an.

- **SOCK_DGRAM**:
  - Der Ablauf: →Socket errichten →Kommunikation.
  - Zwischen den Anwendungen existiert eine verbindungslose Assoziation (connection-less – cl).
  - Eigenschaft:
    - Bytes werden datagramm-weise, basierend auf UDP, übertragen.
    - Aufgrund der fehlenden Nummerierung keine Datensicherung möglich.
    - Sehr effektive Übertragung bei geringen Datenmengen oder einmaligen Daten oder bei Übertragung von Echtzeitdaten (Wiederholung ist hier sinnlos, Zeit läuft ja weiter).
- **SOCK_RAW**:
  - Direkte Nutzung der verbindungslosen IP-Datagram-Übertragung, z.B. für Echtzeitanwendungen.
  - Dieser Verbindungstyp kann, muss aber nicht unter Windows durch einen Service Provider unterstützt werden.

- Auf einem Rechner gibt es genau eine TCP- und UDP-Instanz. Damit mehrere Anwendungen gleichzeitig TCP- oder UDP-Kommunikation machen können, hat man Portnummern eingeführt.
- Für TCP und UDP gibt es jeweils 2^{16}=65536 Portnummern, von 0 bis 65535.

- 3 Portnummernbereiche, verwaltet durch IANA (Internet Assigned Numbers Authority):
  - **Well Known Ports**: 0.. 1023, für allgemein anerkannte Dienste
  - **Registered Ports**: 1024..49151, für proprietäre Dienste
  - **Private Ports**: 49152..65535, für private Anwendungen
- Bis 1992 ging die Well Known Ports lediglich von 0..256.
- Portnummern findet man unter:
  - http://www.ietf.org/rfc/rfc1700.txt
  - **C:\WINDOWS\system32\drivers\etc\services**

Ermitteln Sie:

service	echo			http	pop3	imap	ldap	https
port	7	23	25					
protocol(s)	tcp,udp	tcp,udp	tcp					

Socket-Dienste *(BSD-Funktionen, bereitgestellt durch WS-API)*

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

Call	Initialisierung und Statusabfrage	Endpunkt
socket()	Legt einen neuen Socket an (Adressenformat, Typ)	Server|Client
bind()	Socket wird an lokale IP-Adresse und an eine vom Nutzer gewünschte lokale Portnummer gebunden (hauptsächlich bei Serveranwendungen aber auch auf Clientseite anwendbar).	Server|Client
select()	Überprüfung des Status eine gegebenen Menge von Sockets	Server|Client
	Verbindungsbereitschaft herstellen und Verbindungen akzeptieren	
listen()	▪ Konvertiert Socket in Serversocket (wartet auf Verbindungen→connections), ▪ Legt Warteschlangengröße für kommende Verbindungen an	Server
accept()	▪ Annahme von Client-Verbindungen ▪ Es wird ein neuer Socket zum anrufenden Client errichtet ▪ ServerSocket ist danach zur Entgegennahme weiterer Anrufe bereit	Server
	Verbindungsherstellung und Datenübertragung	
connect()	Stellt eine Assoziation zwischen localHost/localPort und remoteHost/remotePort her Beim TCP-Socket wird die Assoziation in Form einer Verbindung errichtet, beim UDP-Socket werden lediglich localName (IP und Port) sowie remoteName (IP und Port) festgelegt.	Client
send()	Daten in einen (vollständig adressierten) Socket senden	Server|Client
recv()	Daten aus einem (vollständig adressierten) Socket empfangen	Server|Client
sendto()	Daten in einen Socket unter Angabe der remoteAddress senden	Server|Client
recvfrom()	Daten aus einem Socket empfangen. In der Struktur SOCKADDR_IN steht nach Rückkehr der Funktion die Adresse des Absenders (zweckmäßig, wenn man sofort antworten will).	Server|Client
	Verbindung halbseitig beenden, Socket schließen	
shutdown ()	Beendet eine Halbverbindung einer dx-Verbindung (nur bei TCP möglich)	Server|Client
closesocket()	Verbindung beenden und Socket schließen. →BSD nur **close()**	Server|Client
	Steuerungsfunktionen	
getsockopt()	Abfrage von Socketoptionen (is socket listening, type of socket, …)	Server|Client
setsockopt()	Setzen von Socketoptionen	Server|Client
ioctlsocket()	Steuerung der Socketbetriebsart (blocking, non-blocking, …). →BSD nur **ioctl()**	Server|Client

Socket-Hilfs-Dienste *(BSD-Funktionen, bereitgestellt durch WS-API)*

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

Call	Konvertierungsfunktionen
`inet_addr()`	Macht aus IP-Adresse in Punktnotation (141.55.192.70) einen Wert, den man in das Feld s_addr der Struktur in_addr eintragen kann. Bsp: **`sAddr.sin_addr.s_addr=inet_addr("141.55.192.70")`**
`inet_ntoa()`	Ermittelt aus einer Struktur sockaddr_in die IP-Adresse und gibt diese als String aus.
`ntohl()`	Konvertiert 32-Bit-Integer von Network-Byte-Order in die Host-Byte-Orde →(**n**etwork-**to**-**h**ost-**l**ong)
`ntohs()`	Konvertiert 16-Bit-Integer von Network-Byte-Order in die Host-Byte-Order →(**n**etwork-**to**-**h**ost-**s**hort)
`htonl()`	Konvertiert 32-Bit-Integer von Host-Byte-Order in die Network-Byte-Order →(**h**ost-**to**-**n**etwork-**l**ong)
`htons()`	Konvertiert 16-Bit-Integer von Host-Byte-Order in die Network-Byte-Order →(**h**ost-**to**-**n**etwork-**s**hort)
	Datenbankfunktionen
`getpeername()`	Liefert die zugewiesene Remote-Address eines Socket
`getsockname()`	Liefert die zugewiesene Local-Address eines Socket
`gethostbyaddr()`	Liefert den Name eines Hosts, dessen IP-Adressenstring gegeben ist
`gethostbyname()`	Liefert den IP-Adressenstring des Hosts, dessen Name gegeben ist
`getprotbyname()`	Liefert die Portnummer eines Anwendungs-Protokolls (z.B. HTTP, FTP, SMTP)
`getprotbynumber()`	Liefert das Anwendungsprotokoll zu einer Portnummer
`getservbyname()`	Liefert Infos (Port u.a.) zu einem Internet-Dienst (z.B. FTP, SMTP, IMAP, …), dessen Name gegeben ist
`getservbyport()`	Liefert Infos zu einem Internet-Dienst, dessen Portnummer gegeben ist

Nebenbetrachtung: Host- und Network-Reihenfolge *(order)*

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

- Wenn man dem WS-API Verbindungsparameter (Portadressen) übergeben will, geschieht dies in der →Netz-Reihenfolge:
  - Funktionen: Host-Order→Netz-Order: **`htonl() htons()`**
  - Funktionen: Netz-Order→Host-Order: **`ntohl() ntohs()`**

- **Host-Order** oder Little endian (Intel CPUs, DEC alpha) :
  - Der am wenigsten signifikante Teil eines Wertes kommt zuerst →das Dünne zuerst.
- **Network-Order** oder Big endian (Motorola MC68000, SPARC CPUs (SUN)):
  - Der signifikanteste Teil eines Wertes kommt zuerst → das dicke Ende zuerst.

- Beispiel: Darstellung eines 32-Bit-Integers mit dem Wert (04 03 02 01)h

host-order *(little endian)*		***network-order*** *(big endian)*	
auf Adresse	steht Wert	auf Adresse	steht Wert
`"..00"`	`01`	`"..00"`	`04`
`"..01"`	`02`	`"..01"`	`03`
`"..10"`	`03`	`"..10"`	`02`
`"..11"`	`04`	`"..11"`	`01`

Durch Invertierung der letzten beiden Adressbits:
→ Umschaltung zwischen den Systemen.

- Weiteres Beispiel:

Auf Adresse	*stehe Wert*	*Speicherabbild* ***host-order*** *(little endian)*	*Speicherabbild* ***network-order*** *(big endian)*
XX_h	0102_h	**Ab XX_h steht:** **02 01 03 07 06 05 04 08**	**Ab XX_h steht:** **01 02 03 04 05 06 07 08**
$XX_h + 02_h$	03_h		
$XX_h + 03_h$	$0405\ 0607_h$		
$XX_h + 07_h$	08_h		

Datenstrukturen (DS): Übersicht

- Socketfunktionen nutzen **Datenstrukturen**:
  - um z.B. Adressenparameter zu übergeben (`bind(), connect(), sendto()`),
  - um z.B. Adressenparameter zu bekommen (`rcvfrom()` und `accept()`).

- Folgende **Datenstrukturen** sind von Bedeutung:
  - **`sockaddr`** ist eine generische Struktur für die Socketadressierung. Diese Struktur ist für verschiedene Arten von Netzwerkadressen verwendbar (IPX-Adressen, IP-Adressen usw.).
  - ***`sockaddr_in`*** ist die Anpassung der generischen Struktur für die Internet-Adressierung.
    - ***`in_addr`*** ist eine Teilstruktur von ***`sockaddr_in`***, wo die IP-Adresse steht.
  - ***`hostent`*** ist Struktur zur Adressauflösung (`gethostbyname(), gethostbyaddr()`).
  - ***`servent`*** ist Struktur zur Dienstauflösung (`getservbyname(), getservbyport()`).
  - ***`fd_set`*** ist Struktur, die zur Zustandsabfrage (`select()`)verwendet wird.
  - ***`WSAData`*** ist Struktur, die bei der Anmeldung am API verwendet wird (`WSAStartup()`).

- Alle Datenstrukturen werden in C-Notation besprochen →deshalb zur Wiederholung die Basis-Typdefinitionen:

```
/*=== Basic system type definitions, taken from the BSD file sys/types.h ===*/
typedef unsigned char   u_char;  //8 bit
typedef unsigned short  u_short; //16 bit
typedef unsigned int    u_int;   //32 bit
typedef unsigned long   u_long;  //32 bit
```

Datenstrukturen (DS): Nutzungsübersicht

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

Socketfunktionen (bind, …, getservbyport) nutzen Datenstrukturen (sockaddr_in, …, servent) zur Übergabe/zum Erhalt von Parameteren

bind()
connect()
sendto()

rcvfrom()
accept()

```
struct sockaddr_in
    u_short   sin_family
    u_short   sin_port
    struct in_addr  sin_addr
    char  sin_zero[8]
```

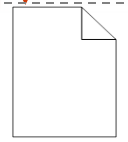
gethostbyname()
gethostbyaddr()

```
struct hostent
    FAR-Ptr  h_name
    FAR-Ptr  h_aliases
    u_short  h_addrtype
    u_short  h_length
    FAR-Ptr  h_addr_list
```

Domain Name Server

getservbyname()
getservbyport()

```
struct servent
    FAR-Ptr  s_name
    FAR-Ptr  s_aliases
    2-Byte   s_port
    FAR-Ptr  s_proto
```



\WINDOWS\system32\drivers\etc**services**

- Struktur zur Übergabe von Adressen zum/vom API:
  - an den Socket (bind, …)
  - vom Socket (rcvfrom, …).

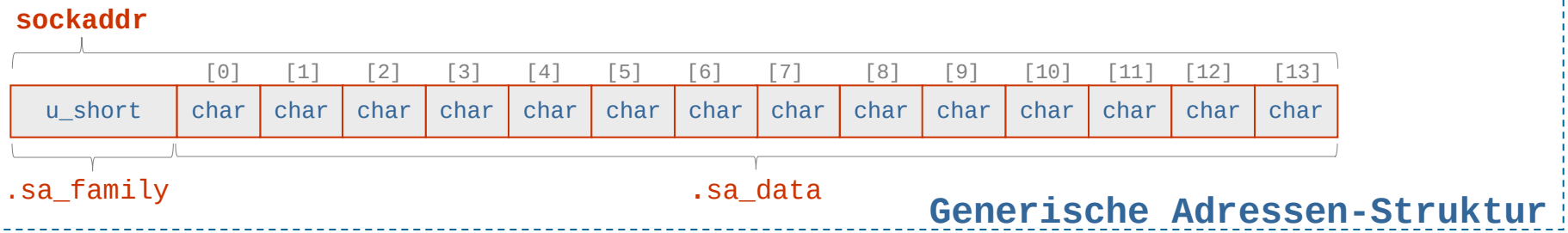
- Struktur zur Auflösung von:
  - Name auf Adresse (gethostbyname)
  - Adresse auf Name (gethostbyaddr)

- Struktur zur Auflösung von:
  - Dienst auf Port (getservbyname)
  - Port auf Dienst (getservbyport)

DS: struct sockaddr (Generische Adressen-Struktur)

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

```
/*== in winsock.h ist deklariert ==================================*/
struct sockaddr {
u_short sa_family; //Adressen-Familie,AF_INET,AF_IPX,AF_UNIX, …
char sa_data[14];  //die Adresse innerhalb einer AF
};
```



Generische Adressen-Struktur

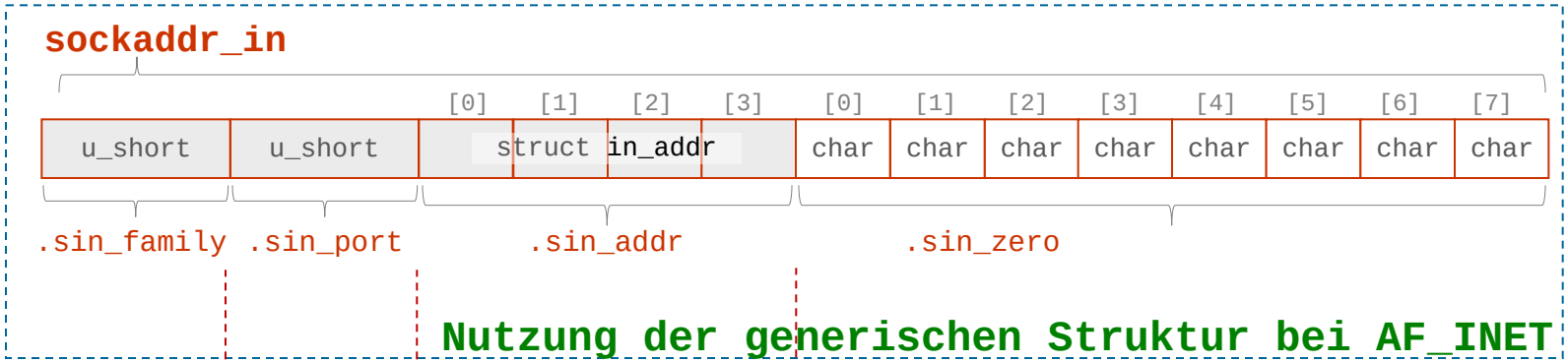
- Diese generische (grundlegende) Adressenstruktur ist für verschiedene Netzwerke gedacht. In sa_family steht Adressentyp, im Rest die Adresse. Beispielsweise für:

```
/*=== in winsock.h sind für sa_family beispielsweise deklariert ==== =====*/
#define AF_UNSPEC       0       //unspecified
#define AF_UNIX         1       //local to host (pipes, portals)
#define AF_INET         2       //interworking: UDP, TCP, etc.
#define AF_IPX          6       //IPX and SPX
#define AF_ISO          7       //ISO protocols
#define AF_ECMA         8       //ECMA protocols
#define AF_CCITT        10      //CCITT protocols, X.25 etc.
#define AF_SNA          11      //IBM SNA
#define AF_DECnet       12      //DECnet
#define AF_APPLETALK    16      //AppleTalk
```

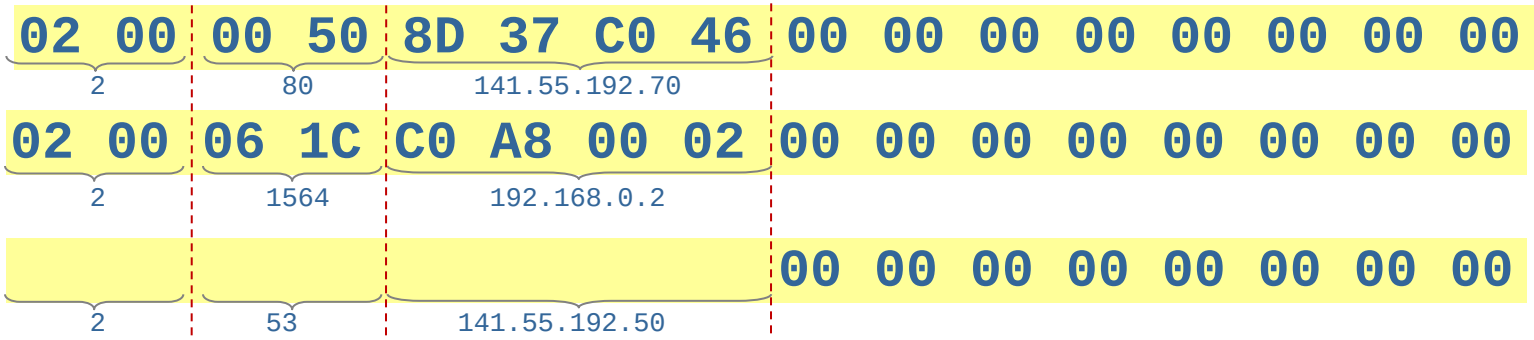
- Für die Adressenfamilie Internet (AF_INET) wurde eine zugeschnittene Struktur sockaddr_in definiert →siehe nächste Folie und folgende.

DS: struct sockaddr_in (Internetverwendung der generischen Struktur)


```
/*== in winsock.h ist deklariert ==========================================*/
struct sockaddr_in {
u_short         sin_family; //hier steht die Adressfamilie AF_INET | 2
u_short         sin_port;   //hier steht die Portadresse
struct in_addr  sin_addr;   //hier steht die IP-Adresse
char            sin_zero[8];//zum Auffüllen der generischen Struktur auf 16 Byte
};
```



- Beispiele:

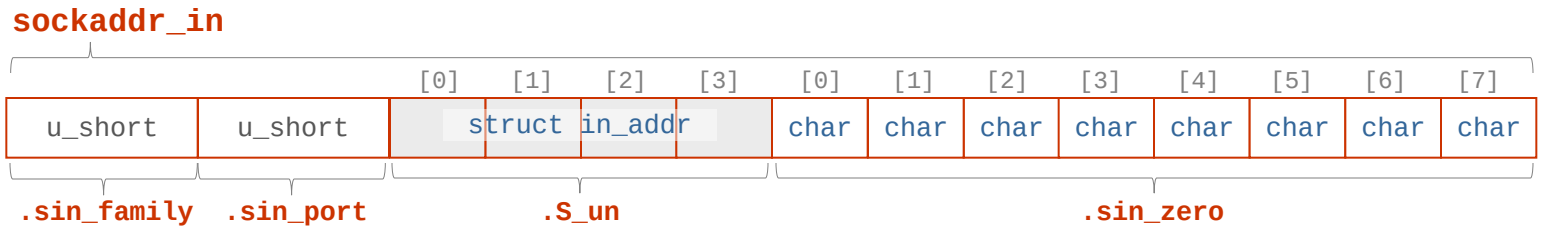


sin – **s**ocket **in**ternet

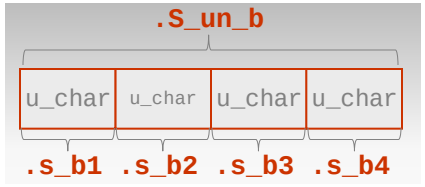
DS: struct in_addr im struct sockaddr_in

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

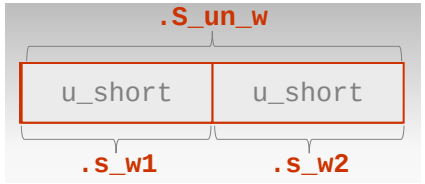
```
/*== in winsock.h ist deklariert ==========================================*/
struct in_addr {
      union {
              struct {u_char  s_b1, s_b2, s_b3, s_b4;} S_un_b;
              struct {u_short s_w1, s_w2;} S_un_w;
              u_long S_addr;
             } S_un;
}
```



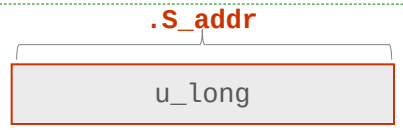
union erlaubt unterschiedliche Zugriffsarten



//entweder **byteweise** Zugriff auf 32-Bit-Adresse



//oder **wordweise** Zugriff auf 32-Bit-Adresse



//oder **dwordweise** Zugriff auf 32-Bit-Adresse

- **Bsp.1:** Für connect() soll in xAddr eingetragen werden: <family> AF_INET, <port> 5432, <ip-addr> 192.168.0.1

(1)xAddr deklarieren	`sockaddr_in  xAddr;`	Variable xAddr vom Typ sockaddr_in deklarieren
(2)<family>	`xAddr.sin_family=AF_INET;`	AF mittels Konstantendeklaration eintragen
(2)<family>	`xAddr.sin_family=2;`	***oder*** AF mittels Wert eintragen
(3)<port>	`xAddr.sin_port=htons(5432);`	Portnr. mittels htons() von Host- in Netzorder
(4)<ip-addr>	`xAddr.sin_addr.S_addr=inet_addr("192.168.0.1");`	IP-Eintrag mittels Funktion inet_addr() →einfachste Möglichkeit
(4)<ip-addr>	`xAddr.sin_addr.S_addr=htonl(0xC0A80001);`	***oder*** IP-Eintrag mittels Funktion htonl()

(4)<ip-addr>	`xAddr.sin_addr.S_un.S_un_b.s_b1=192;`
	`xAddr.sin_addr.S_un.S_un_b.s_b2=168;`
	`xAddr.sin_addr.S_un.S_un_b.s_b3=0;`
	`xAddr.sin_addr.S_un.S_un_b.s_b4=1;`
		oder IP-Eintrag: Byte für Byte

(4)<ip-addr>	`xAddr.sin_addr.S_un.S_un_w.s_w1=MAKEWORD(192,168);` `xAddr.sin_addr.S_un.S_un_w.s_w2=MAKEWORD(0,1);`	***oder*** IP-Eintrag: Word für Word
(4)<ip-addr>	`xAddr.sin_addr.S_un.S_addr=` `MAKELONG(MAKEWORD(192,168),MAKEWORD(0,1));`	***oder*** IP-Eintrag: als Long

Das steht dann in xAddr:

```
02 00 15 38 C0 A8 00 01 00 00 00 00 00 00 00 00
```

sin_family sin_port struct sin_addr sin_zero

DS: Nutzung von **sockaddr_in**

- **Bsp.2:** Für connect() soll in yAddr eingetragen werden: <family> AF_INET, <port> 8080, <ip-addr> 127.0.0.1

(1)yAddr deklarieren	`sockaddr_in yAddr;`	Variable yAddr vom Typ sockaddr_in deklarieren
(2)<family>	`yAddr.sin_family=AF_INET;`	AF mittels Konstantendeklaration eintragen
(2)<family>	`yAddr.sin_family=2;`	***oder*** AF mittels Wert eintragen
(3)<port>	`yAddr.sin_port=htons(8080);`	Portnr. mittels htons() von Host- in Netzorder
(4)<ip-addr>	`yAddr.sin_addr.s_addr=inet_addr("127.0.0.1");`	IP-Eintrag mittels Funktion inet_addr() →einfachste Möglichkeit
(4)<ip-addr>	`yAddr.sin_addr.s_addr=htonl(INADDR_LOOPBACK);`	***oder*** IP-Eintrag mittels Funktion htonl() #define INADDR_LOOPBACK 0x7f000001
(4)<ip-addr>	`yAddr.sin_addr.s_addr=htonl(0x7F000001);`	***oder*** IP-Eintrag mittels Funktion htonl()

(4)<ip-addr>	`yAddr.sin_addr.S_un.S_un_b.s_b1=127;`
	`yAddr.sin_addr.S_un.S_un_b.s_b2=0;`
	`yAddr.sin_addr.S_un.S_un_b.s_b3=0;`
	`yAddr.sin_addr.S_un.S_un_b.s_b4=1;`
		oder IP-Eintrag: Byte für Byte

(4)<ip-addr>	`yAddr.sin_addr.S_un.S_un_w.s_w1=MAKEWORD(127,0);` `yAddr.sin_addr.S_un.S_un_w.s_w2=MAKEWORD(0,1);`	***oder*** IP-Eintrag: Word für Word
(4)<ip-addr>	`yAddr.sin_addr.S_un.S_addr=` `MAKELONG(MAKEWORD(192,168),MAKEWORD(0,1));`	***oder*** IP-Eintrag: als Long

Das steht dann in yAddr:

```
02 00 50 50 7F 00 00 01 00 00 00 00 00 00 00 00
```

sin_family sin_port struct sin_addr sin_zero

- **Bsp.3: Für bind() soll in bAddr eingetragen werden: <family> AF_INET, <port> 80, <ip-addr> 0.0.0.0 →this computer**

(1)bAddr deklarieren	`sockaddr_in  bAddr;`	Zu bindende Adresse bAddr vom Typ sockaddr_in deklarieren
(2)<family>	`bAddr.sin_family=AF_INET;`	AF mittels Konstantendeklaration eintragen
(2)<family>	`bAddr.sin_family=2;`	***oder*** AF mittels Wert eintragen
(3)<port>	`bAddr.sin_port=htons(80);`	Portnr. mittels htons() von Host- in Netzorder
(4)<ip-addr>	`bAddr.sin_addr.s_addr=inet_addr("0.0.0.0");`	IP-Eintrag mittels Funktion inet_addr() →einfachste Möglichkeit
(4)<ip-addr>	`bAddr.sin_addr.s_addr=htonl(INADDR_ANY);`	***oder*** IP mittels htonl() und Konstante INADDR_ANY #define INADDR_ANY (u_long)0x00000000
(4)<ip-addr>	`bAddr.sin_addr.s_addr=htonl(0x000000);`	***oder*** IP-Eintrag mittels Funktion htonl()

(4)<ip-addr>	`bAddr.sin_addr.S_un.S_un_b.s_b1=0;`
	`bAddr.sin_addr.S_un.S_un_b.s_b2=0;`
	`bAddr.sin_addr.S_un.S_un_b.s_b3=0;`
	`bAddr.sin_addr.S_un.S_un_b.s_b4=0;`
		oder IP-Eintrag: Byte für Byte

	`usw.`	

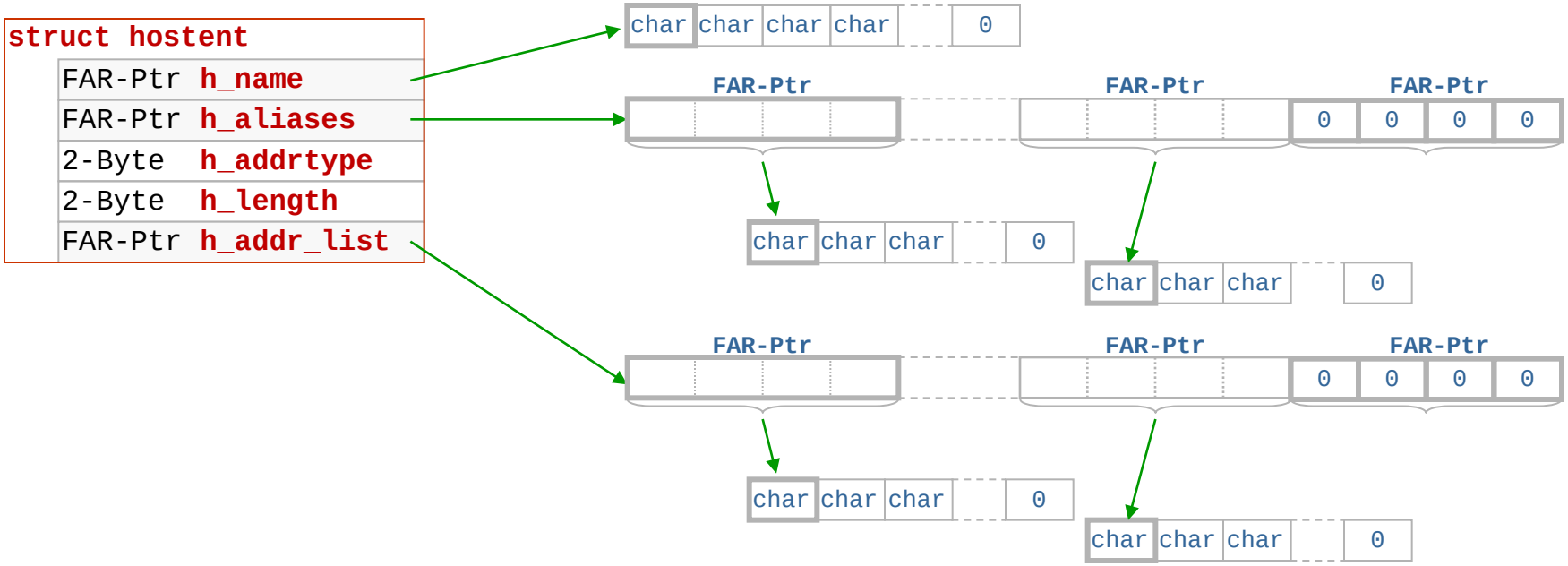
Das steht dann in bAddr:

```
02 00 00 50 00 00 00 00 00 00 00 00 00 00 00 00
```

sin_family sin_port sin_addr sin_zero

DS: struct hostent – für gethostbyname(),gethostbyaddr()

```
/*== in winsock2.h ist deklariert ========================================*/

struct hostent {
        char   FAR* h_name;             //Zeiger auf String (0-term.) mit Hostname
        char   FAR* FAR* h_aliases;     //Zeiger auf Zeigerarray auf Nicknames
        short  h_addrtype;              //hier steht immer AF_INET|2
        short  h_length;                //hier steht immer 4
        char   FAR* FAR* h_addr_list; //Zeiger auf Zeigerarray auf IP-Adressen

#define h_addr h_addr_list[0];          //h_addr enthält die erste IP-Adresse
}
```



- **Bsp.1:** Für www.htwm.de soll mittels gethostbyname() die IP-Adresse ermittelt und anschließend alle Parameter angezeigt werden!

Vorbedingung	`#include <winsock2.h>`	
Zeiger **host** deklarieren	`struct hostent* host;`	Zeigervariable host auf eine Struktur vom Typ hostent anlegen
Funktionsruf DNS-Name	`host=gethostbyname("www.htwm.de");` `host=gethostbyname(Edit1->Text.c_str());`	Mittels gethostbyname soll IP-Adr. von www.htwm.de ermittelt werden. Zeiger-variable host zeigt auf Struktur vom Typ hostent, wo die IP-Adresse steht.
IP-Adresse in **sAddr** übernehmen	`sockaddr_in  sAddr;` `sAddr.sin_addr.S_un.S_addr=*(Cardinal*)host->h_addr;`	sAddr anlegen und dort die IP-Adresse aus host->h_addr eintragen
Auswertung **h_name**	`RichEdit1->Lines->Add ("h_name:"+(AnsiString)host->h_name);`	Anzeige des aufgelösten DNS-Host-Namens in RichEdit-Komponente. Typecasting auf AnsiString.

Auswertung **h_aliases**	`while (*(host->h_aliases))`
	`{RichEdit1->Lines->Add ("h_aliases:"`
	`+(AnsiString)*(host->h_aliases)); (host->h_aliases++);}`
		while (*(host->h_aliases)): solange der Inhalt >0 ist, Anzeige vorhandener Alias-DNS-Host-Namen

Auswertung **h_addrtype h_length**	`RichEdit1->Lines->Add("h_addrtype:"`
	`+IntToStr(host->h_addrtype));`
	`RichEdit1->Lines->Add("  h_length:"`
	`+IntToStr(host->h_length));`	

xAddr deklarieren und Auswertung **h_addr_list**	`struct sockaddr_in xAddr;`
	`while(*(host->h_addr_list))`
	`{xAddr.sin_addr=*(struct in_addr*) *host->h_addr_list;`
	`RichEdit1->Lines->Add("h_addr:"`
	`+(AnsiString)inet_ntoa(xAddr.sin_addr.S_un.S_addr));`
	`host->h_addr_list++;}`
		xAddr ist eine Hilfsstruktur zur Anzeige der IP-Adressen. Durch Nutzung dieser Struktur, kann man die Konvertiertungsfunktion inet_ntoa() des WSA-API nutzen

Auswertung **h_addr_list**	`while (*(host->h_addr_list))`
	`{ char* ipaddr= *(host->h_addr_list);`
	`RichEdit1->Lines->Add("  h_addr    :"`
	`+(AnsiString)(u_char)ipaddr[0]+"."`
	`+(AnsiString)(u_char)ipaddr[1]+"."`
	`+(AnsiString)(u_char)ipaddr[2]+"."`
	`+(AnsiString)(u_char)ipaddr[3]);`
	`host->h_addr_list++;}`
		alternative Auswertung der IP-Adresse(n)

- **Bsp.2:** Für 141.55.192.70 soll mittels gethostbyaddr() der DNS-Hostname ermittelt und alle Parameter angezeigt werden!

Vorbedingung	`#include <winsock2.h>`	
Zeiger host und xAddr deklarieren	`struct hostent* host;` `struct sockaddr_in xAddr;`	Zeiger host auf eine Struktur vom Typ hostent anlegen. xAddr ist Hilfsstruktur für das Eintragen der IP-Adresse.
aufzulösende IP-Adresse direkt eintragen	`xAddr.sin_addr.S_addr=inet_addr("141.55.192.70");`	Adresse mittels **inet_addr()**in xAddr eintragen. Der Adressenstring wird direkt eingetragen.
aufzulösende IP-Adresse indirekt eintragen	`xAddr.sin_addr.S_addr=inet_addr(ComboBox1->Text.c_str());` `xAddr.sin_addr.S_addr=inet_addr(Edit1->Text.c_str());`	Adresse mittels **inet_addr()**in xAddr eintragen. Der Adressenstring steht in der Eigenschaft Text einer CombBox- bzw. Edit-Komponente.
Funktionsruf	`host=gethostbyaddr((char*)&xAddr,sizeof(xAddr),AF_INET))`	bei Borland C++-Builder, wenn aufzulösende Adresse in xAddr steht
Auswertung h_name	`RichEdit1->Lines->Add ("h_name:" +(AnsiString)host->h_name);`	Anzeige des aufgelösten DNS-Host-Namens in einer RichEdit-Komponente.

Auswertung h_aliases	`while (*(host->h_aliases))`
	`{RichEdit1->Lines->Add ("h_aliases:"`
	`+(AnsiString)*(host->h_aliases));(host->h_aliases++);}`
		Anzeige vorhandener Alias-DNS-Host-Namen

Auswertung h_addrtype h_length	`RichEdit1->Lines->Add("  h_addrtype:"`
	`+IntToStr(host->h_addrtype));`
	`RichEdit1->Lines->Add("  h_length:"`
	`+IntToStr(host->h_length));`	

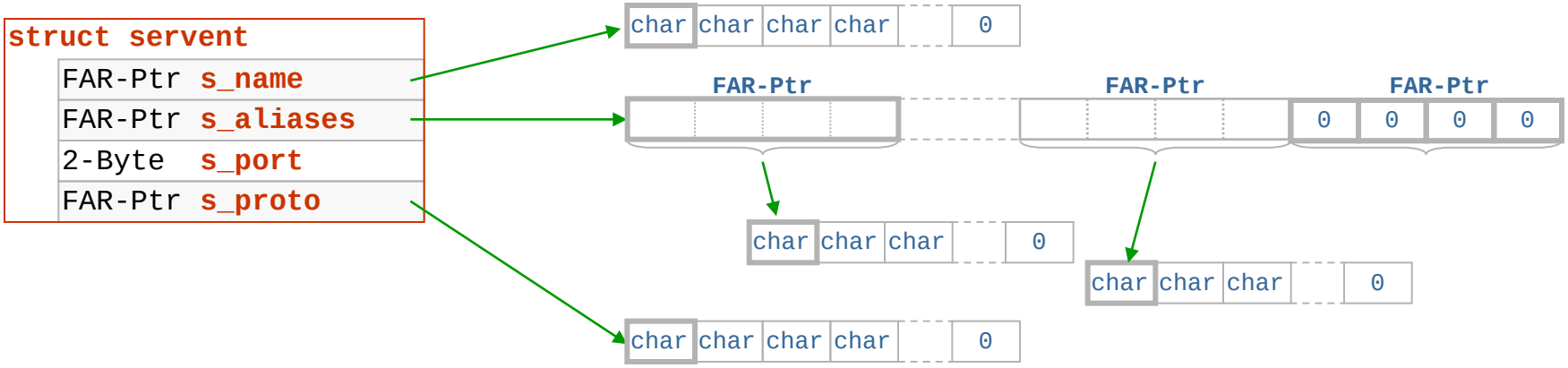
xAddr deklarieren und Auswertung **h_addr_list**	`struct sockaddr_in xAddr;`
	`while(*(host->h_addr_list))`
	`{xAddr.sin_addr=*(struct in_addr*) *host->h_addr_list;`
	`RichEdit1->Lines->Add("h_addr:"`
	`+(AnsiString)inet_ntoa(xAddr.sin_addr.S_un.S_addr));`
	`host->h_addr_list++;}`
		xAddr ist eine Hilfsstruktur zur Anzeige der IP-Adressen. Durch Nutzung dieser Struktur, kann man die Konvertiertungsfunktion inet_ntoa() des WSA-API nutzen

DS: struct servent – für getservbyname(), getservbyport()

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

```
/*== in winsock2.h ist deklariert ========================================*/

struct servent {
        char   FAR*       s_name;     //Zeiger auf String (0-term.) mit Servicename
        char   FAR* FAR*  s_aliases;  //Zeiger auf Zeigerarray auf Nickservnames
        int               s_port;     //Port[1], über den der Service erreichbar ist
        char   FAR*       s_proto;    //Zeiger auf das Transportprotokoll (tcp,udp)
};
```

[1] Port wird in network-order angegeben!



- **Bsp.2: Für http soll das TCP-Port ermittelt und alle Parameter der Struktur hostent angezeigt werden!**

Vorbedingung	`#include <winsock2.h>`	
Zeiger **serv** deklarieren	`struct servtent* serv;`	Zeiger serv auf eine Struktur vom Typ servent anlegen.
Funktionsruf	`serv=getservbyname("http","tcp");`	
Funktionsruf	`serv=getservbyname(ComboBox1->Text.c_str(),` `ComboBox2->Text.c_str());`	bei Borland C++-Builder, wenn das Anwendungsprotokoll in ComboBox1 und das Transportprotokoll in ComboBox2 stehen.
Funktionsruf	`serv=getservbyname(Edit1->Text.c_str(),` `Edit2->Text.c_str());`	bei Borland C++-Builder, wenn das Anwendungsprotokoll in Edit1 und das Transportprotokoll in Edit2 stehen.
Auswertung **s_name**	`RichEdit1->Lines->Add("s_name:"` `+(AnsiString) serv->s_name);`	Anzeige des Dienstnamens, z.B. http, smtp usw.

Auswertung **s_aliases**	`while(*(serv->s_aliases))`
	`{`
	`RichEdit1->Lines->Add("s_aliases:"`
	`+(AnsiString)*(serv->s_aliases));`
	`}`
		Anzeige der zulässigen Alias-Dienstnamen, bei http sind dies z.B. www, www-http

Auswertung **s_port**	`RichEdit1->Lines->Add("s_port:“` `+IntToStr(ntohs(serv->s_port)));`	Anzeige der Portnummer des Dienstes

- **Bsp.2:** Für das TCP-Port 143 soll der Dienst ermittelt und alle Parameter der Struktur hostent angezeigt werden!

Vorbedingung	`#include <winsock2.h>`	
Zeiger **serv** deklarieren	`struct servtent* serv;`	Zeiger serv auf eine Struktur vom Typ servent anlegen.
Funktionsruf	`serv=getservbyport(htons(143),"tcp");//`	Port muss in network order (big endian) übergeben werden. Nutzung der Funktion htons()
Funktionsruf	`serv=getservbyport(htons(ComboBox1->Text.ToInt(),` `ComboBox2->Text.c_str());`	bei Borland C++-Builder, wenn das Port in ComboBox1 und das Transportprotokoll in ComboBox2 stehen.
Funktionsruf	`serv=getservbyport(Edit1->Text.ToInt(),` `Edit2->Text.c_str());`	bei Borland C++-Builder, wenn das Port in Edit1 und das Transportprotokoll in Edit2 stehen.
Auswertung **s_name**	`RichEdit1->Lines->Add("s_name:"` `+(AnsiString) serv->s_name);`	Anzeige des Dienstnamens, z.B. http, smtp usw.

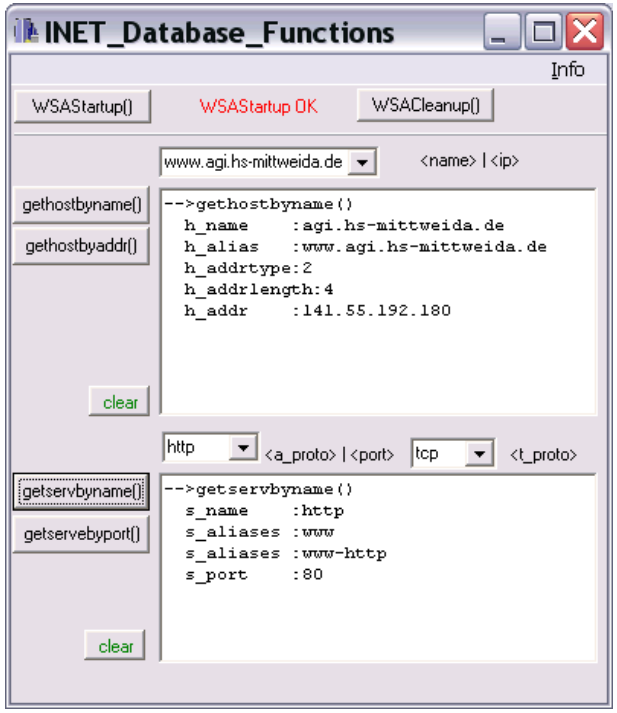
Auswertung **s_aliases**	`while(*(serv->s_aliases))`
	`{RichEdit1->Lines->Add("s_aliases:"`
	`+(AnsiString)*(serv->s_aliases));`
	`serv->s_aliases++; }`
		Anzeige der zulässigen Alias-Dienstnamen, bei Port 80 sind dies z.B. www, www-http

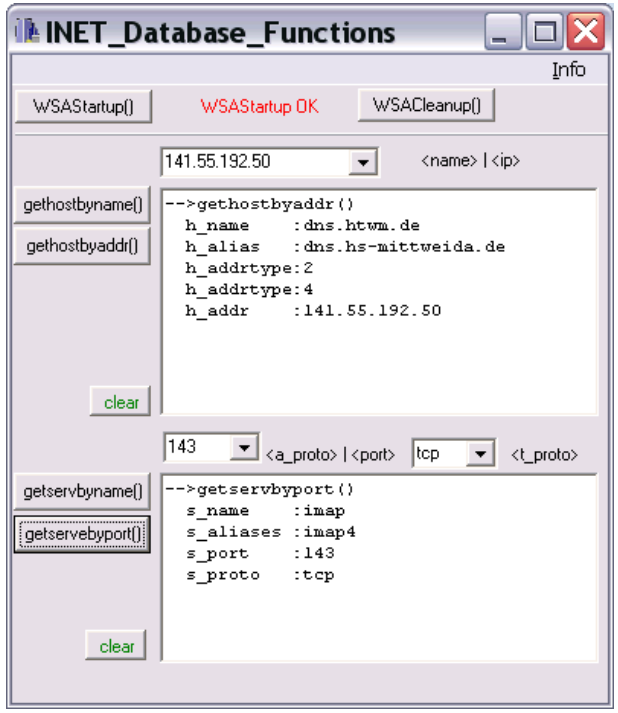
Auswertung **s_port**	`RichEdit1->Lines->Add("s_port:"` `+IntToStr(ntohs(serv->s_port)));`	Anzeige der Portnummer des Dienstes
Auswertung **s_proto**	`RichEdit1->Lines->Add("s_proto:"` `+(AnsiString) serv->s_proto);`	Anzeige des Transportprotokolls

DS: Programmierbeispiel zu `struct` **`hostent`**, `struct` **`servent`**

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

- Beispiel **INET_Database_Functions**: auf www.telecom.hs-mittweida.de, unter >Lehre DI-Direktstudium >Socketprogrammierung >Beispiele …
- Downloaden Sie zuerst die INET_Database_Functions.EXE und probieren deren Funktionalität (siehe Abbildungen)!





Implementieren Sie diese Anwendung selber, wie unter INET_Database_Functions.PDF beschrieben

DS: struct WSAData – Eigenschaften von WSA (<64 Bit-Version)

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

```
struct WSAData {    WORD            wVersion;
                    WORD            wHighVersion;
                    char            szDescription[WSADESCRIPTION_LEN+1];
                    char            szSystemStatus[WSASYSSTATUS_LEN+1];
                    unsigned short  iMaxSockets;
                    unsigned short  iMaxUdpDg;
                    char FAR*       lpVendorInfo; };
```

wVersion	Die Version von WinSocket, die vom API als angeforderte betrachtet wird (z.B. 1.1|2.0|2.2).
wHighVersion	Höchste WinSocket-Version, die die aktuell installierte DLL unterstützen kann (z.B. 2.2).
szDescription	Dieser Text kann bis 255 char lang sein. Real steht hier: WinSock 2.0 (das ist die Bezeichnung des Standards)
szSystemStatus	Ein null-terminierter ASCII-String, in den die WinSock-DLL den relevanten Status kopieren kann (z.B. running)
iMaxSockets	Bei der Socket-Version 1 wird hier die max. Socketanzahl eingetragen (z.B. 32767). Bei der Socket-Version 2 ist dieser Eintrag leer und ohne Bedeutung.
iMaxUdpDg	Bei der Socket-Version 1 wird hier die max. UDP-Datagram-Größe eingetragen (z.B. 65467). Bei Socket-Version 2 ist dieser Eintrag leer. *For the actual maximum message size specific to a particular WinSock service provider and socket type, applications should use **getsockopt()** to retrieve the value of option **SO_MAX_MSG_SIZE** after a socket has been created.*
lpVendorInfo	Bei allen Socket-Versionen wird hier nichts mehr durch die Socket-DLL eingetragen. Erweiterte Infos lassen sich durch ***getsockopt()*** und den Parameter ***VD_CONFIG*** ermitteln.

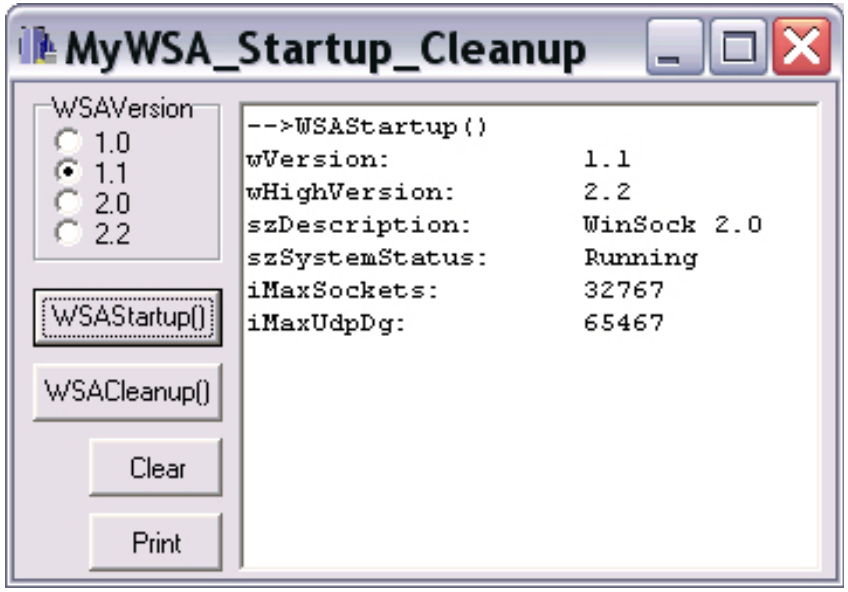
```
typedef struct WSAData {
                    WORD                wVersion;
                    WORD                wHighVersion;
          #ifdef _WIN64
                    unsigned short      iMaxSockets;
                    unsigned short      iMaxUdpDg;
                    char FAR *          lpVendorInfo;
                    char                szDescription[WSADESCRIPTION_LEN+1];
                    char                szSystemStatus[WSASYS_STATUS_LEN+1];
          #else
                    char                szDescription[WSADESCRIPTION_LEN+1];
                    char                szSystemStatus[WSASYS_STATUS_LEN+1];
                    unsigned short      iMaxSockets;
                    unsigned short      iMaxUdpDg;
                    char FAR *          lpVendorInfo;
          #endif
} WSADATA, FAR * LPWSADATA;
```

DS: Programmierbeispiel zu `struct WSAData`

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

- Beispiel **MyWSA_Startup_Cleanup**: auf www.telecom.hs-mittweida.de, unter >Lehre DI-Direktstudium >Socketprogrammierung – Beispiele …

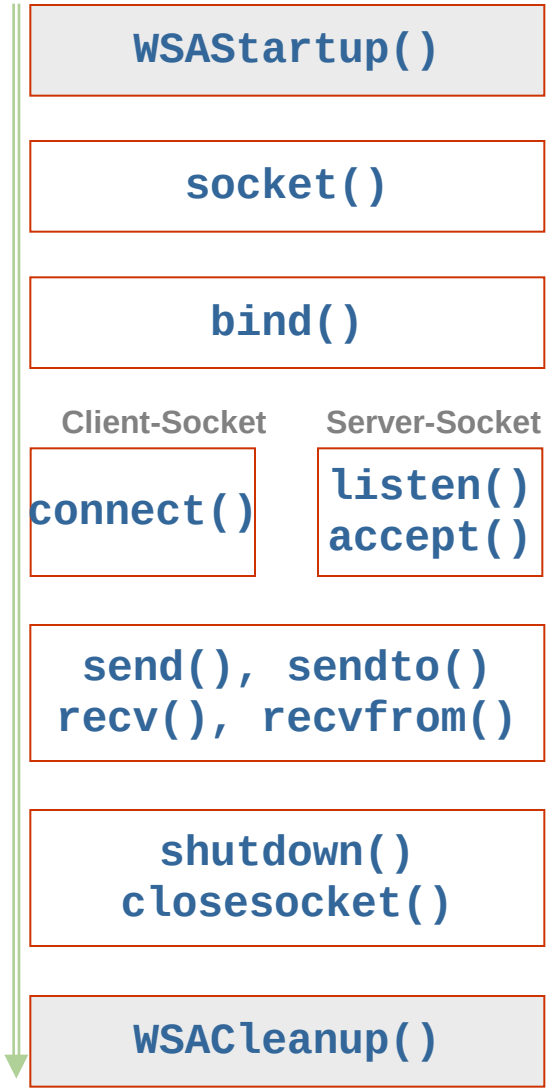
- Downloaden Sie zuerst die MyWSA_Startup_Cleanup.EXE und probieren deren Funktionalität (siehe Abbildung)!



Implementieren Sie diese Anwendung selber, wie unter MyWSA_Startup_Cleanup.PDF beschrieben

Allgemeiner Ablauf einer Socket-Nutzung unter Windows

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de



(1) Anmeldung an der DLL: Speicherplatzreservierung, Abfrage der Version und anderer Parameter

(2) Socket errichten: Festlegung der Adressierungsfamilie (hier AF_INET), des Socket-Typs (SOCK_STREAM, SOCK_DGRAM, SOCK_RAW) und optional das Protokoll

(3) Lokale Socket-Parameter festlegen: lokalen Port, ev. auch lokale IP-Adresse. Bei Srv-Sockets notwendig, bei Clt-Sockets möglich.

(4) Clt-Socket, connect(): Assoziation zu einem Server-Socket herstellen
(4) Srv-Socket, listen(): Assoziationsbereitschaft herstellen,
(4) Srv-Socket, accept(): Assoziationswünsche annehmen.

(5) Socket nutzen: in den Socket schreiben, aus dem Socket lesen

(6) Richtungsabhängiger Abbau einer Verbindung: shutdown()
(6) Socket schließen: closesocket()

(7) Abmeldung an der DLL: Speicherplatzfreigabe

WSAStartup() – an API anmelden

```
int WSAAPI      WSAStartup (
                IN      WORD        wVersionRequested,
                OUT     LPWSADATA   lpWSAData );
```

WSAStartup() ist eine WSA-Erweiterung gegenüber BSD. Damit erfolgt eine Anmeldung an der DLL. Der Aufruf dieser Funktion ist bei Windows unbedingt erforderlich. Die Abmeldung erfolgt mit **WSACleanup()**.

wVersionRequested	Erforderliche DLL-Versionsnummer, die die Anwendung benötigt (1.1, 2.0, …). Im high byte steht die minor version, im low byte die major version.
lpWSAData	LongPointer auf eine **structur** vom Typ WSAData. Diese Struktur ist vordekla-riert (winsock.h|winsock.dll) und wird auf der nächsten Folie besprochen.
Return Value int	wenn **0**, Aufruf fehlerfrei wenn **<>0**, Fehler (Fehlercodes siehe Anhang)

Pascal	**var** retCode:integer; **wsaD**:WSADATA; majVers,minVers:integer;
	begin
	majVers:=2; minVers:=0;
	{1} **retCode:=WSAStartup(MakeWord(majVers,minVers),wsaD);**
	if retCode=0 **then** anzOK() **else** anzERR();
	end;
	{2} **retCode:=(WSAStartup(MakeWord(2,0),wsaD);**
	{3} **retCode:=(WSAStartup(MakeWord(1,1),wsaD);**
	{4} **retCode:=(WSAStartup($0020,wsaD);**

C	#include <winsock2.h>
	WSADATA **wsaD**; //wsaD vom Typ WSADATA anlegen, zur Parameterübergabe API->Applikation
	if (WSAStartup(MAKEWORD(2,0),&wsaD)==0) {anzOK()**} else {**anzErr()**};**
	//MAKEWORD(majVers,minVers)

WSACleanup() – Abmelden an der DLL

int WSAAPI WSACleanup (void);	
WSACleanup() ist eine WSA-Erweiterung gegenüber BSD. Damit erfolgt eine Abmeldung an der DLL. Alle gebundenen Ressourcen werden frei gegeben.	
Return Value ***int***	wenn **0**, Aufruf fehlerfrei wenn **<>0**, Fehler (Fehlercodes siehe Anhang)

Pascal	**var** retCode:integer;
	begin
	  retCode:=WSACleanup();
	  if retCode=0 **then** anzOK() **else** anzERR();
	end;

C	**if (WSACleanup() {**anzOK()**} else {**anzErr()**};**

socket() – einen Socket errichten

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

```
int  WSAAPI      socket (
                 IN      int      af,
                 IN      int      type,
                 IN      int      protocol );
```

socket() erzeugt einen Socket, der auf eine Adressenfamilie, ein Transportprotokoll und u.U. auf ein Protokoll (was diesen Socket nutzt) festgelegt ist. Parameter ***protocol*** kann 0 sein.

af	Legt die Adressenfamilie fest, z.B. **AF_UNIX|1, AF_INET|2, AF_IPX|6.**
type	Legt den Sockettyp fest: **SOCK_STREAM|1, SOCK_DGRAM|2, SOCK_RAW|3**
protocol	Angabe, welches Protokoll IP nutzt. Üblicherweise wird hier 0 eingesetzt. SOCK_STREAM zulässige Werte: **IPPROTO_TCP|6|0,** SOCK_DGRAM zulässige Werte: **IPPROTO_UDP|17|0,** SOCK_RAW zulässige Werte: **IPPROTO_IP|0** bzw. weitere wie: **IPPROTO_ICMP|IPPROTO_IGMP**
Return Value int	wenn **0** = Fehler, mit **int WSAAPI WSAGetLastError()** Fehlernummer ermitteln. wenn **>0** = ist es der SocketDescriptor (auch Handle) der den Socket bei allen künftigen Funktionsaufrufen referenziert.

Pascal	**var** s:Integer;//socket descriptor
	begin
	{1} **s:=socket(AF_INET,SOCK_STREAM,0);**
	if s <> INVALID_SOCKET anzOK() **else** anzERR();
	end;
	{1.1 **s:=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);**}
	{1.2 **s:=socket(AF_INET,SOCK_STREAM,6);**}
	{2 **s:=socket(AF_INET,SOCK_DGRAM,0);**}
	{3 **s:=socket(AF_INET,SOCK_RAW,33);**}
	{4 **s:=socket(AF_INET,SOCK_RAW,IPPROTO_ICMP);**}

C	**int s=socket(AF_INET,SOCK_STREAM,0);** **if** (s != INVALID_SOCKET) **{**anzOK()**}** **else {**anzErr()**};** //INVALID_SOCKET hat den Wert 0

ioctlsocket() – Input/Output-Mode einstellen oder abfragen

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

```
int WSAAPI      ioctlsocket (
                IN      SOCKET          s,
                IN      long            cmd,
                IN OUT  u_long FAR*     argp);
```

TCP, UDP: wenn ein Socket errichtet wurde, ist er automatisch blockierend gesetzt. Diese Blockade wirkt sich auf die Funktionen accept(), connect(), recv(), rcvfrom(), send(), sendto(), closesocket() in unterschiedlicher Art und Weise aus, siehe Folie. Blockieren bedeutet, das z.B. die Funktionen recv() oder accept() solange warten, bis aus dem Socket etwas zu lesen ist oder ein neuer Verbindungswunsch zu akzeptieren ist. Ruft man also recv() auf und für diesen Socket kommt nie etwas zum Lesen an, blockiert damit die gesamte Anwendung.

(Lösung 1) Der Socket wird mittels Kommando FBIONIO in nichtblockierend gesetzt. Anschließend muß man diese Funktionen im Pollingmode aufrufen, was zu einer hohen Prozessorlast führt.

(Lösung 2) Bevor man recv() bzw. accept() aufruft, prüft man mittels der Funktion select(), welcher Socket lesbar ist bzw. ob am Serbversocket ein neuer Verbindungswunsch vorliegt, der akzeptiert werden soll.

s	Socket descriptor
cmd	Kommandos, welches auf den Socket *s* angewendet werden soll. **FIONBIO** (File IOctl Nonblocking I/O), zum Setzen in nichtblockierend|blockierend Mode **FIONREAD** (File IOctl Read), zur Abfrage, wieviel Bytes durch ein *recv()* gelesen werden können. Handelt es sich um einen SOCK_DGRAM, erhält man in argp die Größe des zuerst empfangenen Datagrams.
argp	Zeiger auf einen Parameter, der *cmd* parametrisiert
Return Value int	**0** =Anzeige, das der Funktionsruf erfolgreich war **>0** =Anzeige des Fehlercodes

Pascal	**var** retCode:**integer**; para: **word**;
	begin
	  para:=1;//jeder Wert >0 ist zulässig
	  retCode:=ioctlsocket(s, FIONBIO, para);
	  if retCode <> SOCKET_ERROR **then** anzOK() **else** anzERR();
	end;

C	**int** retCode; **u_int** para;
	//socket nonBlocking setzen; para muss Wert >0 haben, will man Socket wieder blockieren: par=0!
	  para=1; **if (retCode=ioctlsocket(s,FIONBIO,&para)>0) {**anzOK()**} else** {anzERR();}
	//Abfrage wie viel Byte aus s lesbar sind, Bytezahl steht dann in para
	  if (retCode=ioctlsocket(s,FIONREAD,&para)>0) {anzOK("socket(s) lesbar")**}**
	  else {anzERR("kein socket lesbar");}

bind() – Socket eine lokale Adresse|Port zuweisen

```
int WSAAPI      bind (
                IN      SOCKET                          s,
                IN      const struct sockaddr FAR*      name,
                IN      int                             namelen  );
```

bind() dient dazu, einem Socket lokale Adressenparameter (IP-Adresse, Port) zuzuweisen. Diese Funktion muss man nutzen, wenn man einen ServerSocket errichten will um die Portnummer zuzuweisen. Die Anwendung von bind() ist aber auch bei einem Client-Socket möglich, wenn dieser beispielsweise auf eine bestimmte lokale Portnummer festgelegt werden soll.

s	Socketdescriptor (int), erhalten als Rückgabeparameter bei socket().
name	Zeiger[1)] auf Name der Adress vom Typ SOCKADDR_IN. Dort trägt man üblicherweise ein: -AF_INET|2, -lokales Port (z.B. 80) in network-order, also unter Verwendung der function **htons**(80) -lokale IP-Adresse (üblicherweise 0.0.0.0 →this computer).
namelen	Länge der Adresse, zu ermitteln mit **SizeOf(addr).**
Return Value ***int***	wenn **-1**= Fehler, mit **int WSAAPI WSAGetLastError()** Fehlernummer ermitteln. wenn **0** = OK

Pascal	**var** retCode:Integer; **sAddr**: SOCKADDR_IN;
	begin
	  sAddr.sin_family:=AF_INET;
	  sAddr.sin_port:=htons(80);
	  sAddr.sin_addr.S_un.S_addr:=inet_addr('0.0.0.0');
	{1} **retCode:= bind(s,sAddr,sizeOf(sAddr));**
	  if retCode <> SOCKET_ERROR **then** anzOK() **else** anzERR();
	end;

C	SOCKADDR_IN **sAddr**; //serveradresse definieren und ausfüllen
	  sAddr.sin_family=AF_INET;
	  sAddr.sin_port=htons(6666);
	  sAddr.sin_addr.S_un.S_addr=inet_addr("0.0.0.0");
	if (bind(s,(struct sockaddr*)&sAddr,sizeof(sAddr))<>SOCKET_ERROR) {anzOK()**} else {**anzErr()**};**

[1)] Ab winSock 2, kann auch die Struktur selber übergeben werden

listen() – Socket in Servermodus setzen

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

```
int WSAAPI        listen (
                  IN      SOCKET          s,
                  IN      int             backlog, );
```

Mit listen() wird aus einem Socket ein Verwaltungssocket. Ankommende Verbindungen müssen durch accept() bestätigt werden. Die Anzahl max. wartender Verbindungen ist auf 5 begrenzt, auch wenn man höhere Werte in ***backlog*** übergibt.

s	Socket descriptor, der einen gebundenen, aber nicht verbundenen Socket identifiziert.
backlog	Max. Anzahl der Warteplätze einkommender Verbindungen. Der MaxWert ist 5 und in der Konstanten **SOMAXCONN** festgelegt. Dieser Wert wird nicht überschritten. Einkommende Verbindungen werden mit accept() angenommen.
Return Value int	**0** =Socket ist im Zustand "Listen" **-1** =Fehler, mit **int WSAAPI WSAGetLastError()** Nummer des Fehlers ermitteln.

Pascal	**var** retCode:Integer;
	begin
	  retCode:=listen(s,1|2|..|5);
	  if retCode <> SOCKET_ERROR **then** anzOK() **else** anzERR();
	end;

C	**if (listen(s,5)!=SOCKET_ERROR) {**anzOK()**} else {**anzErr()**};**

accept() – Assoziationswünsche akzeptieren

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

```
int WSAAPI      accept (
                IN      SOCKET                  s,
                OUT     struct sockaddr FAR*    addr,
                OUT     int  FAR*               addrlen );
```

Nach **listen()** ruft man **accept()** auf. Damit kann man ankommende Assoziationen von Clients annehmen. Diese Funktion blockiert bis zur Ankunft einer Assoziation, es sei denn, man hat mittels **ioctrsocket()** non-blocking eingestellt. Für jede erfolgreiche Annahme wird ein neuer Socket kreiert. Die socketDescriptoren muss man sich in einem Array (z.B. **FD_SET** mySocks;) selber merken. Wenn man die remoteAdresse erfahren will, muss man einen Zeiger auf eine Adressenstruktur vom Typ **SOCKADDR_IN** und deren Länge übergeben. Benötigt man diese Informationen nicht, übergibt man für **addr, addrlen** den Wert 0. Mit **select()** kann man ermitteln, ob für das listenPort Assoziationswünsche vorliegen. Jeder Socket muss mit **closesocket()** einzeln beendet werden.

s	Socket descriptor, eines Socket der im Zustand **listen** ist.
addr	Zeiger auf Name der Adressenstruktur vom Typ SOCKADDR_IN, auch 0 zulässig.
addrlen	Länge der Adresse, zu ermitteln mit **sizeof(aAddr),** auch 0 zulässig.
Return Value int	>**0** = socketDescriptor der angenommenen Verbindung →muss gespeichert werden **-1** = Fehler, mit **int WSAAPI WSAGetLastError()** Fehlernummer ermitteln.

Pascal	**var** as, aAddrLen:**Integer**; aAddr: **SOCKADDR_IN**; mySocks: **FD_SET;**
	begin
	aAddrLen:=sizeof(aAddr);
	{1} **as:=accept(s,sAddr,aAddrLen);**
	if as <> SOCKET_ERROR **then begin** anzOK();save_as(as);save_addr(aAddr);**end else** anzERR();**end;**
	
	{2} **as:=accept(s,0,0);**
	if as <> SOCKET_ERROR **then begin** anzOK();save_as(as); **end else** anzERR();
	end;

C	**sockaddr_in** aAddr;**int** as, aAddrLen; **FD_SET** mySocks; aAddrLen=sizeof(aAddr); {1} **if (as = accept(s,(struct sockaddr *)&aAddr,&aAddrLen))!=SOCKET_ERROR** **{**anzOK();**FD_SET**(as,&mySocks);**} else {**anzErr()**};** {2} **if (as = accept(s,0,0)!=SOCKET_ERROR)** **{**anzOK(); FD_SET(as,&mySocks);**} else {**anzErr()**};**

connect() – eine Assoziation herstellen

```
int WSAAPI      connect (
                IN      SOCKET                          s,
                IN      const struct sockaddr FAR*      name,
                IN      int                             namelen );
```

TCP-Socket: Wird diese Funktion auf einen TCP-Socket angewendet, wird versucht eine Assoziation zwischen Clt und Srv in Form einer Verbindung (connection) herzustellen (→SYN ←SYN,ACK →ACK).

UDP-Socket: Einem UDP-Socket kann man lokale Parameter mit bind() zuweisen, z.B. eine bestimmtes localPort. Ruft man nach bind() connect() auf, werden die Parameter remoteAddress und remotePort dem Socket zugewiesen. Damit ist die Assoziation vollständig beschrieben: (localAddress, localPort) ←→ (remoteAddress, remotePort). Hat man kein bind() angewendet, erhält der UDP-Socket alle Parameter mit dem connect()-Aufruf.

s	Socket descriptor, erhalten bei socket()
name	Ist ein Zeiger auf eine Adressenstruktur vom Typ SOCKADDR_IN
namelen	Länge der Adresse, zu ermitteln mit **sizeof(addr)**
Return Value int	wenn **0** =Verbindung hergestellt wenn **-1** =Fehler, mit **int WSAAPI WSAGetLastError()** Nummer des Fehlers ermitteln.

Pascal	**var** retCode:**Integer**; **sAddr**:SOCKADDR_IN;
	begin
	  sAddr.sin_family:=AF_INET;
	  sAddr.sin_port:=htons(6666);
	  sAddr.sin_addr.s_addr:=inet_addr('127.0.0.1');
	{1} **retCode:=connect(sd,sAddr,sizeof(sAddr));**
	  if retCode <> SOCKET_ERROR **then** anzOK() **else** anzERR();
	end;

C	SOCKADDR_IN **sAddr**; //serveradresse definieren und ausfüllen
	sAddr.sin_family=AF_INET;
	sAddr.sin_port=**htons**(6666);
	sAddr.sin_addr.S_un.S_addr=**inet_addr**("127.0.0.1");
	if **(connect (s, (struct sockaddr*)&sAddr,sizeof(sAddr))==0) {**anzOK()**} else {**anzErr()**};**

select() – Status eines oder mehrerer Sockets ermitteln

```
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);
```

Mit select() überprüft man den Status von Socketinstanzen. select() nutzt dazu bis zu drei Strukturen vom Datentyp fd_set. Die Funktion kehrt zurück, wenn für mindestens eine Socketinstanz ein Ereignis vorliegt oder die **timeout**-Zeit abgelaufen ist.

nfds	bei WSAAPI =0
readfds[1)]	Zeiger auf Struktur vom Typ fd_set (file descriptor set), welche Sockets sind lesbar? Lesbarkeit eines Serversockets bedeutet: Verbindungswunsch liegt vor, accept()aufrufen. Lesbarkeit eines Clientsockets bedeutet: Daten liegen vor, receive()aufrufen.
writefds[1)]	Zeiger auf Struktur vom Typ fd_set (file descriptor set), welche Sockets sind beschreibbar.
exceptfds	Zeiger auf Struktur vom Typ fd_set (file descriptor 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 Socketinstanzen, die gelesen, akzeptiert werden müssen oder ein Fehler vorliegt wenn **-1** =Fehler, mit **int WSAAPI WSAGetLastError()** Nummer des Fehlers ermitteln.
Pascal	

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=200;`
	`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);}`
	`}`
	`} stop:`

[1)] siehe auch Beispiele zu FD_SET und select() weiter hinten

send() – Daten in einen (vollständig adressierten) Socket senden

```
int WSAAPI        send (
                  IN      SOCKET            s,
                  IN      const char FAR *  buf,
                  IN      int               len,
                  IN      int               flags );
```

TCP: nach erfolgreichem **connect()** kann man **send()** nutzen. Als Rückgabeparameter erhält man die Anzahl real gesendeter Bytes.

UDP: Beim SOCK_DGRAM wird mittels **connect()** der Socket adressiert und damit eine Assoziation aber keine Verbindung hergestellt. Anschließend kann man mit **send()** Daten in den adressierten Socket senden. Als Rückgabeparameter erhält man die Anzahl real gesendeter Bytes. Bei SOCK_DGRAM verwendet man besser **sendto()**.

ALLGEMEIN: **send()** ist eine blockierende Funktion, z.B. wenn der Socket momentan keine Sendedaten entgegen nehmen kann.

ALLGEMEIN: Der Rückgabewert enthält die Anzahl der tatsächlich gesendeten Bytes. Wurden nicht alle übergebenen Byte gesendet, muss man die noch nicht gesendeten erneut senden.

s	Socket descriptor
buf	Zeiger auf das erste Byte, der zu sendenden Daten
len	Anzahl der zu sendenden Bytes.
flags	**0** →für normale TCP-Daten, **MSG_OOB** (out-of-band[1]) →für **URG**end (dringende) TCP-Daten.
Return Value int	>**0** =Anzahl gesendeter Bytes **-1** = Fehler, mit **int WSAAPI WSAGetLastError()** Fehlernummer ermitteln.

Pascal	**var** slen:**integer**; buf: **array**[0..1459]**of char**;
	begin
	  buf:='dieser text soll gesendet werden '+#0;
	{1} s**len:=send(s,&buf,StrLen(buf),0);**
	  if slen <> SOCKET_ERROR **then** anzOK() **else** anzERR();
	end;
	{2 **len:=send(sd,buf,StrLen(buf),MSG_OOB);**} //Daten werden URGent gesendet, nur bei SOCK_STREAM

C	**char** *buf="GET /hsm/index.htm HTTP/1.0\r\n" **if (int slen=send(s,buf,strlen(buf))) {**anzOK()**} else {**anzErr()**};**

1) damit kann man dringende Daten, abgegrenzt vom TCP_Datenstrom senden. Im TCP-Header wird das URG-Flag gesetzt. Der URG-Pointer gibt an, bis wohin die dringenden Daten gehen (URG-Daten stehen ab Headerende bis vor den Offset-Wert im URG-Pointer, danach können "normale" Daten kommen).

sendto() – Daten zu einer remoteAddr senden

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

```
int WSAAPI      sendto (
                IN      SOCKET                      s,
                IN      const char FAR *            buf,
                IN      int                         len,
                IN      int                         flags,
                IN      const struct sockaddr FAR * to,
                IN      int                         tolen );
```

UDP: **sendto()** wird normalerweise bei **SOCK_DGRAM** verwendet, um Daten an eine remoteAddr zu senden. Hat man bereits mit **connect()** eine remoteAddr zugewiesen, wird diese Adresse ignoriert und die übergebene verwendet. Rückgabeparameter erhält Zahl real gesendete Bytes.

TCP: verwendet man **sendto()** bei einem **SOCK_STREAM**, wirkt diese Funktion wie **send()**, d.h. die Parameter **to** und **tolen** werden ignoriert. Als Rückgabeparameter erhält man die Anzahl real gesendeter Bytes.

ALLGEMEIN: **sendto()** ist eine blockierende Funktion, z.B. wenn der Socket momentan keine Sendedaten entgegen nehmen kann.

ALLGEMEIN: Der Rückgabewert enthält die Anzahl der tatsächlich gesendeten Bytes. Wurden nicht alle übergebenen Byte gesendet, muss man die noch nicht gesendeten erneut senden.

s	Socket descriptor
buf	Zeiger auf das erste Byte, der zu sendenden Daten
len	Anzahl der zu sendenden Bytes.
to	Ist ein Zeiger auf eine Adressenstruktur vom Typ SOCKADDR_IN
tolen	Größe der Adresse in to, z.B. ermittelt mit sizeOf(to)
Return Value *int*	>**0** =Anzahl gesendeter Bytes **-1** = Fehler, mit **int WSAAPI WSAGetLastError()** Fehlernummer ermitteln.

Pascal	**var** slen: integer; buf:**array**[0..1459]of **char**; sAddr:**TSockAddr**;
	begin
	buf:='Dieser Text soll gesendet werden'+#0;
	sAddr.sin_family:=AF_INET;
	sAddr. sin_addr.s_addr:=inet_addr('127.0.0.1');
	sAddr.sin_port:=htons(6666);
	slen:=sendto(s,&buf,StrLen(buf),0,&sAddr,sizeof(sAddr));
	if slen <> SOCKET_ERROR **then** anzOK() **else** anzERR();
	end;

C	**char** *buf="Dieser Text soll gesendet werden"; **SOCKADDR_IN** **sAddr**; **sAddr**.sin_family=AF_INET; **sAddr**.sin_addr.S_un.S_addr=**inet_addr**("127.0.0.1"); **sAddr**.sin_port=**htons**(6666); **if** (**int slen=sendto(s,buf,strlen(buf),0,&sAddr,sizeof(sAddr))**) {anzOK()} **else** {anzErr()};

recv() – Daten aus einem (vollständig adressierten) Socket lesen

```
int WSAAPI      recv (
                IN      SOCKET          s,
                OUT     char FAR*       buf,
                IN      int             len,
                IN      int             flags   );
```

TCP: nach erfolgreichem **connect()** kann man **recv()** nutzen. Der Rückgabewert enthält die Anzahl der in **buf** übergebenen Bytes.Die übergebenen Bytes werden aus dem Empfangspuffer der SockDLL gelöscht.

ACHTUNG: Ist beim SOCK_STREAM der Rückgabeparameter ="0", hat der remotePartner die Verbindung beendet, es kommt also nichts mehr.

UDP: Beim SOCK_DGRAM wird mittels **connect()** oder **bind()** der Socket adressiert und damit eine Assoziation aber keine Verbindung hergestellt. Anschließend kann man mit **recv()** Daten aus dem Socket lesen. Der Rückgabewert ist die Anzahl der in **buf** übergebenen Bytes.

ALLGEMEIN: **recv()** ist eine blockierende Funktion.

ALLGEMEIN: Ist die Größe von **buf** nicht ausreichend, sind die Daten weg. Der Rückgabeparameter kann also > **len** sein!!! Deshalb ausreichende buf-Größe festlegen (z.B. 1460 Byte, bei Ethernets).

s	Socket descriptor
buf	Zeiger auf das erste Byte des Empfangsbuffers
len	Größe des Empfangsbuffers
flags	**0**, für normale TCP-Daten empfangen und diese im Socket-Puffer löschen, **MSG_PEEK**, für normale TCP-Daten empfangen und diese im Socket-Puffer nicht löschen **MSG_OOB,** (out-of-band) für dringende Daten empfangen.
Return Value ***int***	**0** = Anzeige, das die Verbindung von der Gegenseite beendet wurde (nur SOCK_STREAM) >**0** = Anzahl der Bytes, die in buf übergeben werden **-1** = Fehler, mit **int WSAAPI WSAGetLastError()** Fehlernummer ermitteln.

Pascal	**var** rlen:**integer**; buf: **array**[0..1459]**of char**;
	begin
	{1} **rlen:=recv(s,&buf,sizeof(buf),0);**
	if retCode <> SOCKET_ERROR **then** anzOK() **else** anzERR();
	end;
	{2 **rlen:=recv(s,buf,sizeof(buf),MSG_OOB);**} //Empfang von URGent-Daten, nur bei SOCK_STREAM möglich
	{3 **rlen:=recv(s,buf,sizeOf(buf),MSG_PEEK);**} //Datenempfang, die aber im SockDLL-Puffer verbleiben

C	**char** buf[1460]; **if** (**int rlen=recv(s,buf,sizeof(buf),0)>**0) {//Empfangsdaten verarbeiten} **else** {anzERR();}

recvfrom() – Daten u. Absenderadresse aus Socket lesen

```
int WSAAPI        recvfrom (
                  IN       SOCKET                 s,
                  OUT      char FAR*              buf,
                  IN       int                    len,
                  IN       int                    flags,
                  OUT      struct sockaddr FAR*   from,
                  IN OUT   int FAR*               fromlen,);
```

TCP: nach erfolgreichem **connect()**kann man **recvfrom()** nutzen. Als Rückgabeparameter erhält man die Anzahl empfangener Bytes und die Adresse des Absenders in **from**. Besonderheiten siehe → **recv().**

UDP: Beim SOCK_DGRAM wird mittels **connect()** oder **bind()** der Socket adressiert und damit eine Assoziation aber keine Verbindung hergestellt. Anschließend kann man mit **recvfrom()** Daten aus dem addressierten Socket lesen. Der Rückgabewert enthält die Anzahl der in **buf** übergebenen Bytes und in **from** die Absenderadresse.

Will man zusätzlich wissen, von wem die Emfangs-Daten sind, verwende man **rcvfrom()**.

s	Socket descriptor
buf	Zeiger auf das erste Byte des Empfangsbuffers
len	Größe des Empfangsbuffers
flags	**0**|**MSG_PEEK**|**MSG_OOB** →Siehe **recv()**
from	Ist ein Zeiger auf eine Adressenstruktur vom Typ SOCKADDR_IN
fromlen	Größe der Adresse in from, z.B. ermittelt mit sizeof(from)
Return Value int	**0** =Anzeige, das die Verbindung von der Gegenseite beendet wurde (nur SOCK_STREAM) >**0** =Anzahl der Bytes, die in buf übergeben werden **-1** =Fehler, mit **int WSAAPI WSAGetLastError()** Fehlernummer ermitteln.

Pascal	**var** rlen:**integer**; buf: **array**[0..1459]**of char**; from:**TSockAddr**;
	begin
	{1} **rlen:=recvfrom(s,&buf,sizeof(buf),0,&from, sizeof(from));**
	if retCode <> SOCKET_ERROR **then** anzOK() **else** anzERR();
	end;
	{2 **rlen:=recvfrom(s,buf,sizeof(buf),MSG_OOB,&from,sizeoff(from));**} //URGent-Daten, nur bei SOCK_STREAM
	{3 **rlen:=recvfrom(s,buf,sizeOf(buf),MSG_PEEK,&from,sizeoff(from);**} //Datenempfang, die aber im SockDLL- Puffer verbleiben

C	**char** buf[1460];**SOCKADDR_IN from**; **if** (**int rlen=recvfrom(s,buf,sizeof(buf),0,from,sizeof(from))>**0) {//Empfangsdaten verarbeiten} **else** {anzERR();}

closesocket() – Socket schließen

int WSAAPI **closesocket (IN SOCKET** *s* **);**	
Der Socket wird geschlossen	
s	Socket descriptor
Return Value int	**0** =Anzeige, Socket erfolgreich geschlossen
Pascal	**var** retWert:**integer**; **begin** {1} **retwert:=closesocket(s);** **if** retCode = 0 **then** anzOK() **else** anzERR(); **end**;
C	**int retWert=closesocket(s);** **if** (retWert==0) **{**anzOK()**} else {**anzErr()**};**

Programmierbeispiel: MyTime1..3

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

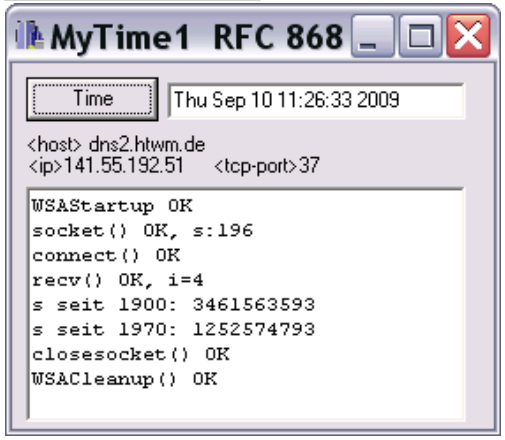
- Im RFC868 (Mai 1983) wird ein Zeitserver wie folgt spezifiziert:

```
When used via TCP the time service works as follows:
      S: Listen on port 37 (45 octal).
      U: Connect to port 37.
      S: Send the time as a 32 bit binary number.
      U: Receive the time.
      U: Close the connection.
      S: Close the connection.
The time is the number of seconds since 00:00 (midnight) 1 January 1900 GMT, such that the
time 1 is 12:00:01 am on 1 January 1900 GMT; this base will serve until the year 2036.
```

- Auf www.telecom.hs-mittweida.de,
  >Lehre DI-Direktstudium >Socketprogrammierung – Beispiele, finden Sie MyTime1..3.

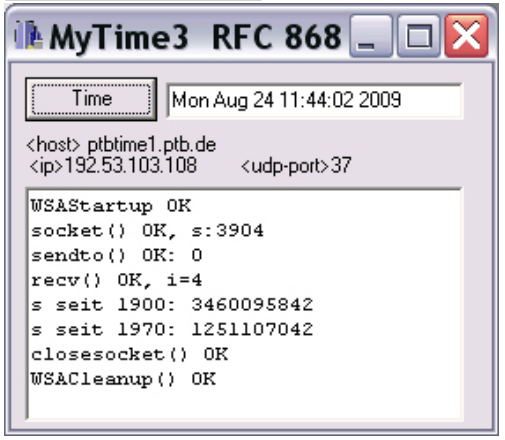
- Downloaden Sie zuerst die jeweilige EXE probieren deren Funktionalität (siehe Abbildungen)!

SOCK_STREAM



SOCK_STREAM

SOCK_DGRAM



Implementieren Sie diese Anwendungen selber, wie in den PDFs beschrieben.

Programmierbeispiel: MyEchoClient1 WINSOCK_STREAM 1(2)

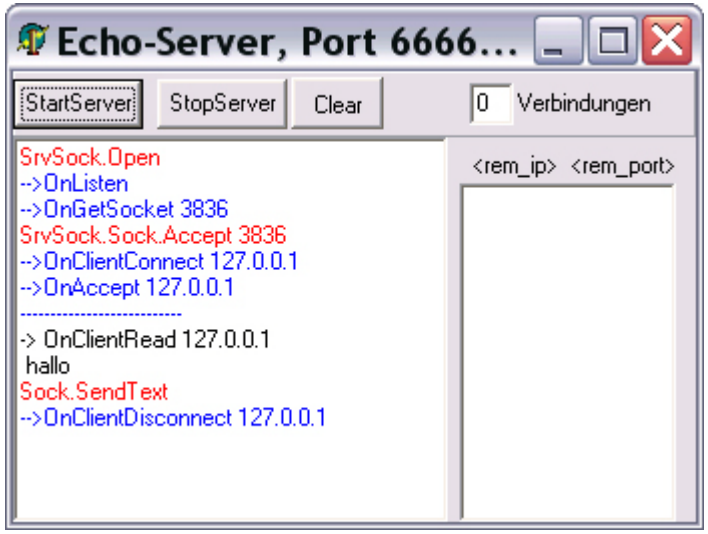
Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

- Ziel:
  - Programmieren einer Echo-Clientanwendung, die den Text 'hallo' zu einem Echoserver sendet.
  - Der Echo-Server wandelt den Text in Großbuchstaben und sendet diesen zurück an den Client.

- Voraussetzung
  - Unter https://www.telecom.hs-mittweida.de finden Sie unter >Lehre DI-Direktstudium > Socketprogrammierung das Serverprogramm SOCK_STREAM_Echo_Srv.exe

  1. Downloaden und Starten Sie die SrvAnwendung.
  2. Tragen Sie die gewünschte Srv-Portnummern ein, z.B. 6666
  3. Starten sie den Server.

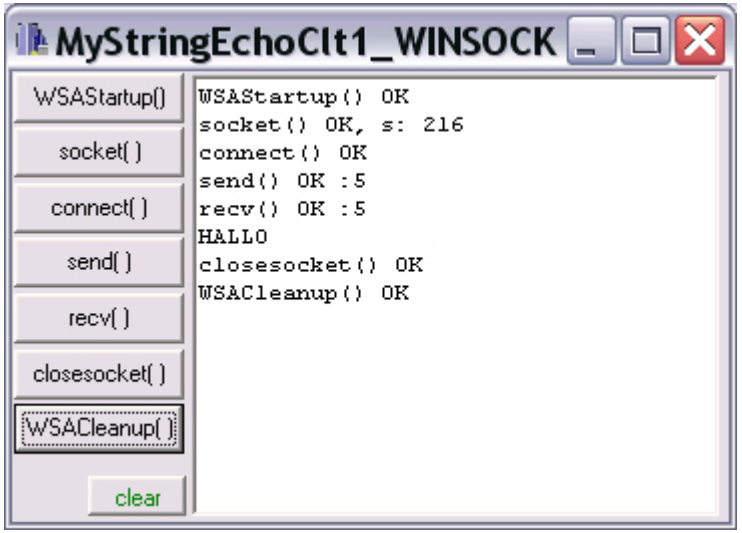
- Downloaden und starten Sie dann die Beispiel.EXE "MyEchoClient1", um die Funktionsweise zu erfassen. Sie sollten dann im Server das Nebenstehende sehen.



Funktion des Servers

- Der Server akzeptiert ankommende Verbindungen und zeigt die `<IP-Adresse>` an.
- Der Server zeigt den empfangenen String an und sendet diesen UpperCase zurück an den Sender.
- Der Server zeigt das Beenden von Verbindungen an.

Mit Borland C++ Builder/Explorer ist nebenstehende Oberfläche für den Client zu programmieren.



- Die ANLEITUNG und eine Beispiel-EXE finden Sie unter: www.telecom.hs-mittweida.de
  >Lehre DI-Direktstudium > Socketprogrammierung
- *Beachten Sie: Server- und Clientanwendung laufen auf dem gleichen Host!*
- *Im nächsten Beispiel werden Clint- und Serveranwendung auf verschiedenen Hosts zum Ablauf kommen.*

- Im RichEdit soll der Ablauf der Socketnutzung zu sehen sein:

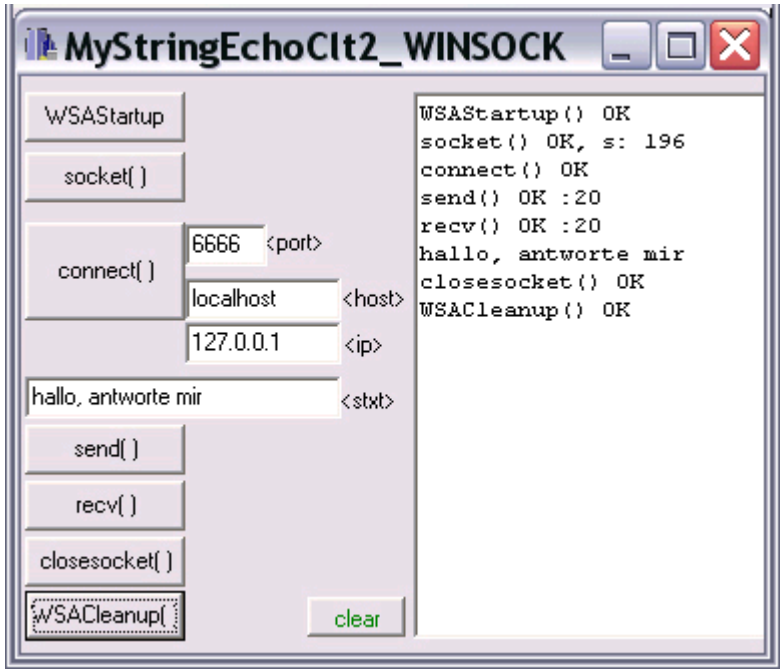
```
WSAStartup() //an DLL anmelden
socket()     //Socket errichten und SocketDescriptor anzeigen
connect()    //Verbindung zum Server herstellen
send()       //FixString “hallo“ senden, Rückgabewert: 5 Byte gesendet
recv()       //Echo vom Server empfangen, Rückgabewert: 5 Byte empfangen
closesocket()//Socket schließen
WSACleanup() //von DLL abmelden
```

- Ziel:
  - Der EchoClient soll Eingabemöglichkeiten für die Serveradresse und einen frei wählbaren Text erhalten.
  - Die Adressenauflösung ist zu programmieren.

- Voraussetzung:
  - Alle Kommilitonen sollten einen Echo-Server auf Port 6666 starten. Erfragen Sie die IP-Adressen und nutzen Sie Echo-Server, die jetzt auf einem beliebigen anderen Host laufen.
  - Downloaden und Starten Sie zuerst die Beispiel-EXE, damit Sie die Funktionalität erfassen.

Mit Borland C++ Builder/Explorer ist nebenstehende Oberfläche für den Client zu programmieren.

  - Die ANLEITUNG und eine Beispiel-EXE finden Sie unter: www.telecom.hs-mittweida.de
    >Lehre DI-Direktstudium > Socketprogrammierung



- Sockets können auf BLOCKING oder NON-Blocking eingestellt ist.
  - Wird ein Socket errichtet, ist er automatisch auf BLOCKING eingestellt.
  - Mit **`ioctlsocket()`** kann die Betriebsart BLOCKING | NON-BLOCKING (für alle Funktionen) gesetzt werden.
- Wenn der Socket auf BLOCKING eingestellt ist, blockieren folgende Funktionen:

Funktion	*Verhalten*
accept()	Blockiert, bis ein Verbindungswunsch ankommt.
connect()	SOCK_STREAM: blockiert solange, bis die Verbindung zustande kommt oder festgestellt wird das dies nicht möglich ist. SOCK_DGRAM: blockiert nicht, da hier nur übergebene Adressenparameter als remoteAddr-Parameter eingetragen werden
recv() **rcvfrom()**	Blockiert, bis Daten von der Gegenstelle vorliegen.
send() **sendto()**	Blockiert dann, wenn der WSOCK-DLL-Sende-Puffer voll ist. Erst wenn der Socket vorherige Sende-Daten aus dem Pufferbereich senden konnte, werden die mit send(), sendto() neu übergebenen Daten übernommen und die Funktion kehrt zurück.
closesocket()	Blockiert solange, bis alle noch im Puffer vorhandenen Daten gesendet wurden.

- Das Blockieren von Socketfunktionen kann:
  - ***gut sein***: **`recv(), recvfrom()`** kehrt z.B. erst zurück, wenn wirklich was empfangen wurde.
  - ***schlecht sein***: weil die gesamte Anwendung während dieser Zeit steht. Im Extremfall muss die Anwendung gekillt werden, wenn z.B. nach Aufruf von **`recv()`** keine Daten mehr ankommen.

- Es gibt 4 Verfahren mit dem Blockieren umzugehen:

(1) NON-BLOCKING per **`ioctlsocket()`** setzen, und dann zyklisch die Funktionen aufrufen (pollen) bis Erfolg eintritt.

(2) Die CallBack-Möglichkeit der WSOCK-DLL **`WSAAsyncSelect()`** nutzen.
Die DLL meldet, das z.B. neuer Verbindungswunsch vorliegt, gelesen werden kann, Out-Of_Band-Daten vorliegen usw.

(3) Die Funktion `select()` nutzen:

i. um bei einem Serversocket zu ermitteln, ob ein Verbindungswunsch vorliegt, um diesen dann mit **`accept()`** zu bedienen,

ii. um zu ermitteln, welche Sockets beschreibbar bzw. lesbar sind, um dann erst die Funktionen **`send(), recv()`** nutzen.

(4) Blockierende Aufrufe in einem THREAD starten →es blockiert der Thread, aber nicht die Anwendung, die im Haupt-Thread läuft.

- In diesem Script werden die Verfahren (3) und (4) besprochen, da Polling nicht sehr elegant ist und Call-Back nicht von allen API's unterstützt wird.

Funktion select() und die Datenstruktur fd_set

- Mittels der API-Funktion **`select()`** kann man den Status eines oder mehrerer Socket-Instanzen bestimmen. Dies ist insbesondere bei Serveranwendungen erforderlich.
- Weiter vorn wurde die Funktion select() schon einmal beschrieben. Beim Aufruf der Funktion kann man Zeiger auf drei Strukturen vom Typ fd_set übergeben.

```
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);
```

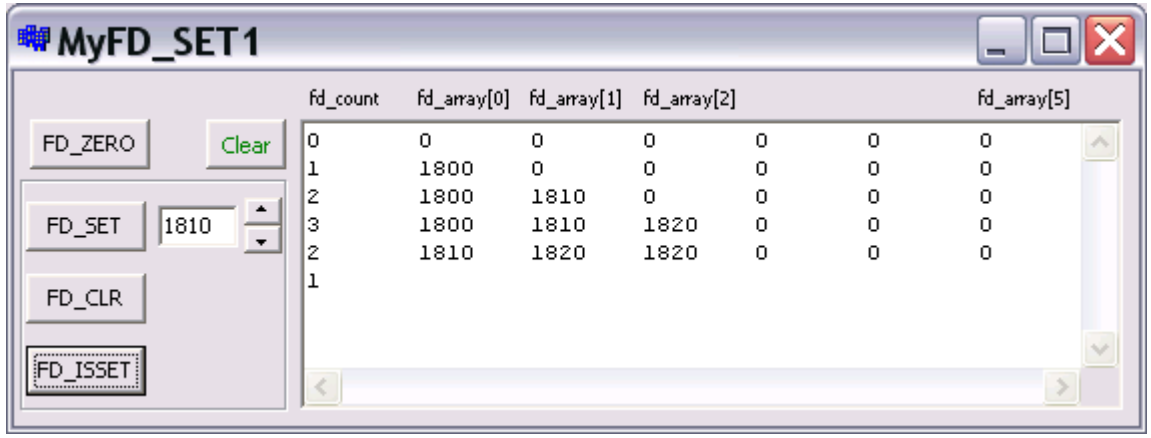
- Hier der Aufbau der drei Strukturen `readfds, writefds, excepfds` aus winsock2.h:

```
typedef u_int           SOCKET;

#define FD_SETSIZE 64

typedef struct fd_set {
u_int fd_count;                 //Anzahl der gültigen Einträge
SOCKET fd_array[FD_SETSIZE];  //Ein Array von SocketDescriptoren, hier 64
} fd_set;
```

- Ziel:
  - Grundsätzlicher Umgang mit Strukturen vom Typ fd_set
  - Anwendung der API-Funktionen: FD_ZERO(), FD_SET(), FD_CLR(), FD_ISSET()

- Mit Borland C++ Builder/Explorer ist unten stehende Oberfläche zu programmieren.

  - Die ANLEITUNG und eine Beispiel-EXE finden Sie unter: www.telecom.hs-mittweida.de >Lehre DI-Direktstudium > Socketprogrammierung



In dem Beispiel werden der Zähler gültiger Einträge (fd_count) und die ersten 6 SocketDescriptorEinträge angezeigt

Im Beispiel ist folgendes passiert:

•nach dem Programmstart ist die Struktur mit 0 besetzt

•Mittels FD_SET wurden nacheinander die Descriptoren 1800, 1810, 1820 eingetragen

•Danach wurde mittels FD_CLR der Descriptor 1800 gelöscht

•Dann wurde mittels FD_ISSET abgefragt, ob der Descriptor 1810 im Set enthalten ist.

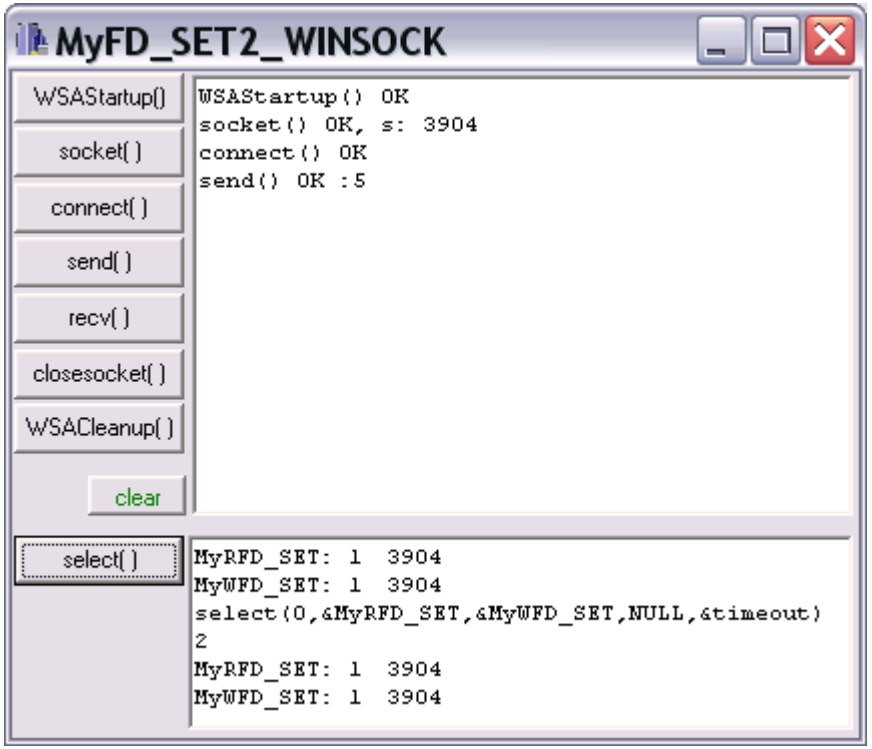
Funktion select() und die Datenstruktur fd_set

Struktur	Beim ClientSocket	Beim ServerSocket
readfds	Für Sockets, deren Descriptoren in dieser Struktur stehen, liegen Daten vor, die mit `recv()` oder `rcvfrom()` blockierungsfrei gelesen werden können. Bei SOCK_STREAM wird damit auch ein Verbindungsabbau signalisiert , wenn man beim Lesen des Sockets 0 Byte zurückbekommt.	Ist der Serversocketdescriptor in readfds eingetragen, liegen ein oder mehrere Verbindungswünsche vor, die mit `accept()` blockierungsfrei akzeptiert werden können.
writefds	In Sockets, deren Descriptoren in dieser Struktur stehen kann blockierungsfrei geschrieben werden mit `send()` oder `sendto().` War der Socket im Zustand connecting, wird mit einem Eintrag angezeigt, das die Verbindung hergestellt wurde und in den Socket geschrieben werden kann.	-
exceptfds	Ist der Socket im Zustand connecting, wird mit einem Eintrag angezeigt, das ein Fehler aufgetreten ist. Dieser kann mit getsockopt() ermittelt werden. Im Zustand connected wird mit einem Eintrag des Socketdescriptors angezeigt, das Out-Of-Band-Daten lesbar sind. Dies sind Daten, die in einem TCP-Segment mit gesendet werden können (siehe Vorlesung Internet (2)).	-

- Nachdem der Umgang mit der Struktur fd_set im ersten Beispiel im Mittelpunkt stand, soll hier die Funktion **`select()`** genutzt werden, um Socketereignisse zu ermitteln:
  - Socket lesbar?
  - Socket beschreibbar?

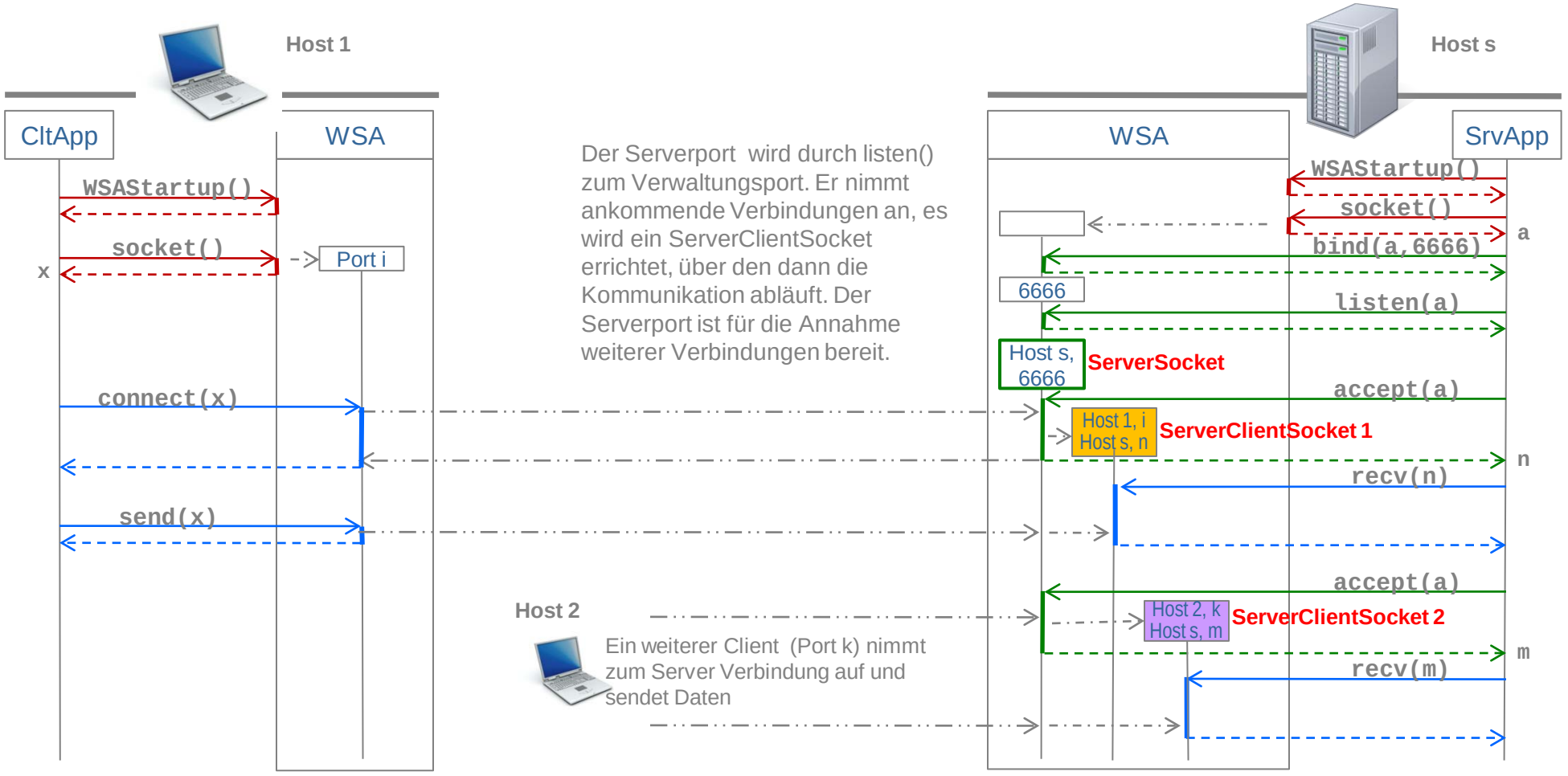
- Ziel: Grundlegender Umgang mit **`select()`**
- Umsetzung:
  - Wir nutzen erst mal einen Clientsocket. Hier sind die Verhältnisse übersichtlich.
  - Kopieren Sie das Beispiel MyEchoClt1_WINSOCK in den Ordner MyFD_SET2.
  - Benennen Sie das Projekt entsprechend um.
  - Vor dem Schreiben, bzw. Lesen wird mit der Funktion **`select()`** die Lesbarkeit, Beschreibbarkeit geprüft.

- Mit Borland C++ Builder/Explorer sind die nebenstehenden Oberflächen zu programmieren.
  - Die ANLEITUNG und eine Beispiel-EXE finden Sie unter: www.telecom.hs-mittweida.de >Lehre DI-Direktstudium > Socketprogrammierung



Nach WSAStartup(), socket(), connect(), send() wurde select() aufgerufen. Es wurden zwei Strukturen (Lesbar?, Schreibbar?) übergeben. Aus dem Rückgabewert 2 ist ersichtlich, das zwei Ereignisse vorliegen. Hier bedeutet dies, der Socket 3904 ist sowohl beschreibbar als auch lesbar.

- Bisher wurden nur Clients betrachtet. Serveranwendungen und damit Sersockets sind etwas komplizierter, da ein Server i.d.R. parallel mehrere Assoziationen zu Clients hat.
- ServerSockets sind **Verwaltungs-Sockets** zur Annahme von Verbindungen. Bei Annahme wird jeweils ein Socket errichtet, über den dann real kommuniziert wird.



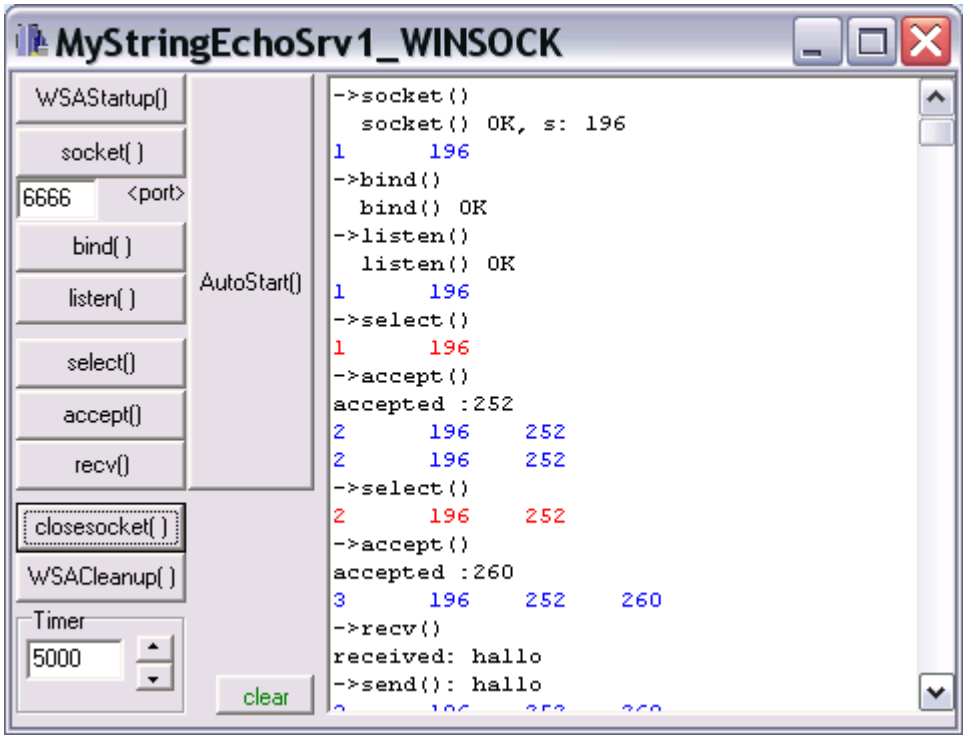
Programmierbeispiel: MyEchoSrv1_WINSOCK_Select

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

- Ziel:
  - Einfachste Implementierung eines Echo-Servers,
  - Der Socket soll blockieren,
  - Nutzung von select() um zu entscheiden, ob accept() oder recv() genutzt werden muss.
- Hinweis:
  - Es wird die Klasse TTimer genutzt.
  - Läuft der Timer ab, wird select() aufgerufen.

Mit Borland C++ Builder/Explorer ist nebenstehende Oberfläche für den Server zu programmieren.

- Die ANLEITUNG und eine Beispiel-EXE finden Sie unter: www.telecom.hs-mittweida.de
  >Lehre DI-Direktstudium
  >Socketprogrammierung



Im obigen Beispiel wurden zwei Strukturen vom Typ fd_set genutzt:
Blau: alle aktiven Sockets die in readfds übergeben werden
Rot: readfds nach dem Aufruf von **`select()`**.
Der Socketdescriptor (sd) des Serversockets ist 196.

1. **`select(): 1 Socketereignis, da für 196 → accept()`**
2. **`select(): 2 Ereignisse, da für 196 → accept(),`**
   `da für 252 → recv()`

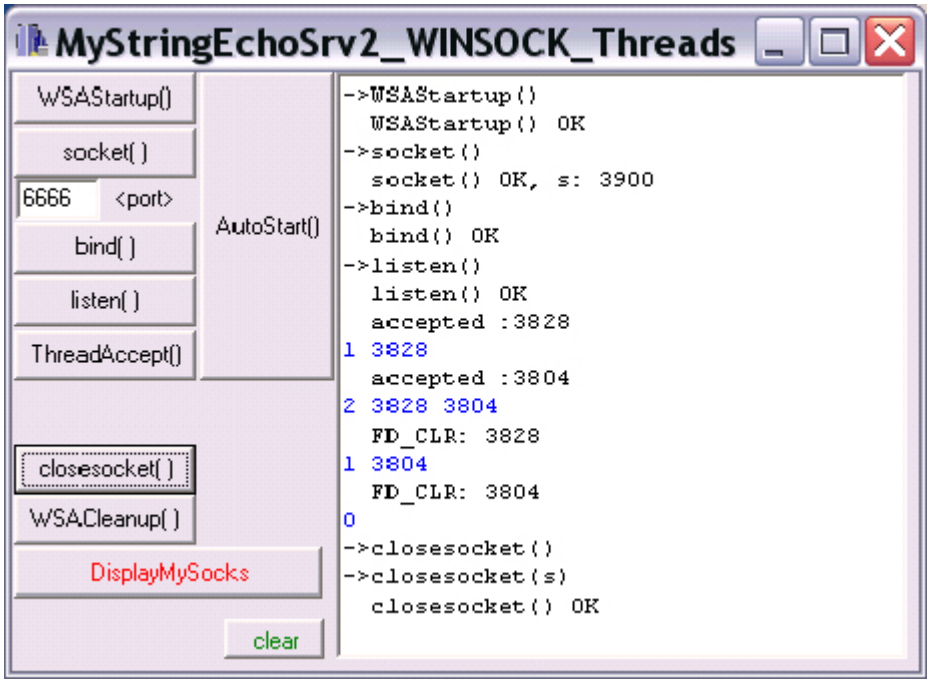
Programmierbeispiel: MyEchoSrv2_WINSOCK_Threads

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

- Ziel:
  - Implementierung eines Echo-Servers, basierend auf Threads.
  - Die blockierenden Funktionen , accept() und recv() werden in einem Thread ausgeführt
- Hinweis:
  - Informieren Sie sich zuerst unter http://docs.embarcadero.com/products/rad_studio/cbuilder6/DE/dg.pdf
    Teil1, Kapitel 11 über den Umgang mit Threads

Mit Borland C++ Builder/Explorer ist nebenstehende Oberfläche für den Server zu programmieren.

- Die ANLEITUNG und eine Beispiel-EXE finden Sie unter: www.telecom.hs-mittweida.de
  >Lehre DI-Direktstudium
  >Socketprogrammierung



Im obigen Beispielablauf ist folgendes dargestellt:

•Ein Socket (Descriptor=3900) wird errichtet, gebunden und mit listen() zum ServerSocket.

•Zwei Client-Verbindungen werden akzeptiert (3828, 3804).

•Die Clients bauen mit closesocket() die Verbindung ab, die Descriptoren werden deshalb gelöscht

•Mit closesocket() wird der Serversocket beendet.

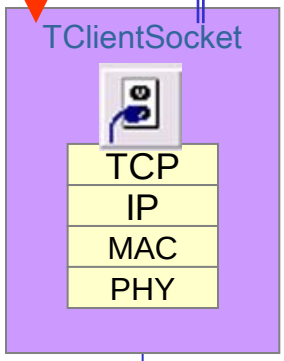
- Die Borland-IDE bietet mehrere Client- und Serversocket-Komponenten an. Klassiker sind TClientSocket, TServerSocket.
- Sollten diese Komponenten nicht vorhanden sein, findet man hier eine Installationsanleitung.

Methoden der Komponente TClientSocket (Auswahl):

```
ClientSocket1->Open() //Öffnet die Socketverbindung
ClientSocket1->Close() //Schließt die Socketverbindung
ClientSocket1->Socket.SendText(string) //String wird in Socket geschrieben
string ClientSocket1->Socket.ReceiveText //String wird aus Socket gelesen
```

Ereignisse der Komponente TClientSocket (Auswahl):

```
OnConnect     //Der Socket ist verbunden
OnConnecting  //Die Verbindung wird hergestellt
OntDisconnect //Verbindung beendet
OnError       //Fehlermeldung, Bedienung durch ErrorCode:=0
OnLookup      //Versuch, den Hostname per DNS auf IP-Adresse aufzulösen
OnRead        //Daten wurden empfangen, können gelesen werden (ctNonBlocking)
OnWrite       //Socket ist beschreibbar (ctNonBlocking)
```



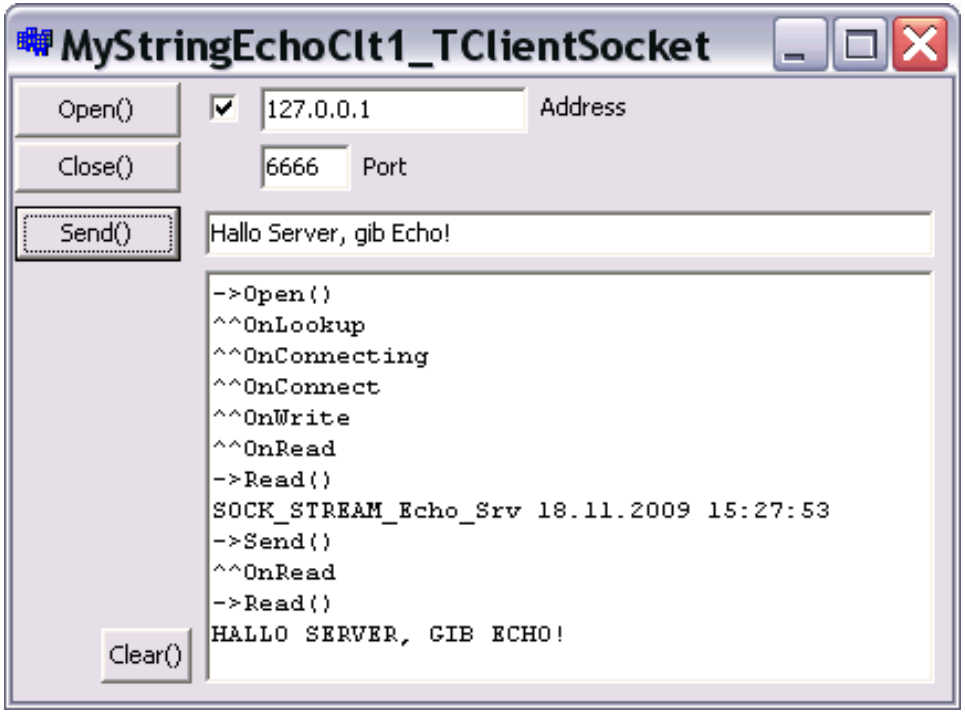
Properties:

```
Active      False|True //wenn true, wird beim Start open aufgerufen
Address     IP-Adresse //dominiert Host; z.B. 141.55.192.70 oder 127.0.0.1
ClientType  ctBlocking|ctNonBlocking //ctNonBlocking liefert OnX-Ereignisse
Host        Host-Name //z.B. www.hs-mittweida.de oder localhost
Port        21 | 80 | 25 | 6666 //dominiert service
Service     ftp | http | smtp
```

- Ziel:
  - Implementierung eines Echo-Clients mit der Komponente TClientSocket.
  - Adresse und Port sollen einstellbar sein.

- Mit Borland C++ Builder/Explorer ist nebenstehende Oberfläche für den Client zu programmieren.
- Es wurden folgende Objekte verwendet:
  - `TButton Open_, Close_, Send_, Clear_`
  - `TEdit Edit1, Edit2, Edit3`
  - `TCheckBox CheckBox1`
  - `TLabel Label1, Label2`
  - `TRichEdit RichEdit1`

- Die ANLEITUNG und eine Beispiel-EXE finden Sie unter: www.telecom.hs-mittweida.de
  >Lehre DI-Direktstudium
  >Socketprogrammierung



Im obigen Beispiel sieht man Folgendes:
Aufruf der Funktion open()
Die Ereignisse OnLookup … OnRead // der server sendet Begrüßungstext
Aufruf der Funktion Read()
Ausgabe des Begrüßungstextes
Aufruf der Funktion Send()
Aufruf der Funktion Read()
Ausgabe des Echotextes

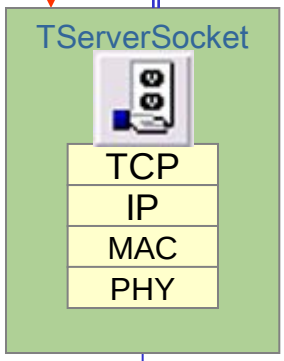
- Basis für dieses Programmierbeispiel ist die Komponente TServerSocket (TCP-basiert).
- Sollten diese Komponente nicht vorhanden sein, findet man hier eine Installationsanleitung.

Methoden der Komponente TServerSocket (Auswahl):

```
open //Aktiviert den Server_Socket, wartet auf Verbindungen
close //Schließt den Server_Socker
Socket.SendText(string) //String wird in Socket geschrieben
string Socket.ReceiveText //String wird aus Socket gelesen
//weitere Schreib- und Lesemethoden→siehe Hilfe
```

Ereignisse der Komponente TServerSocket (Auswahl):

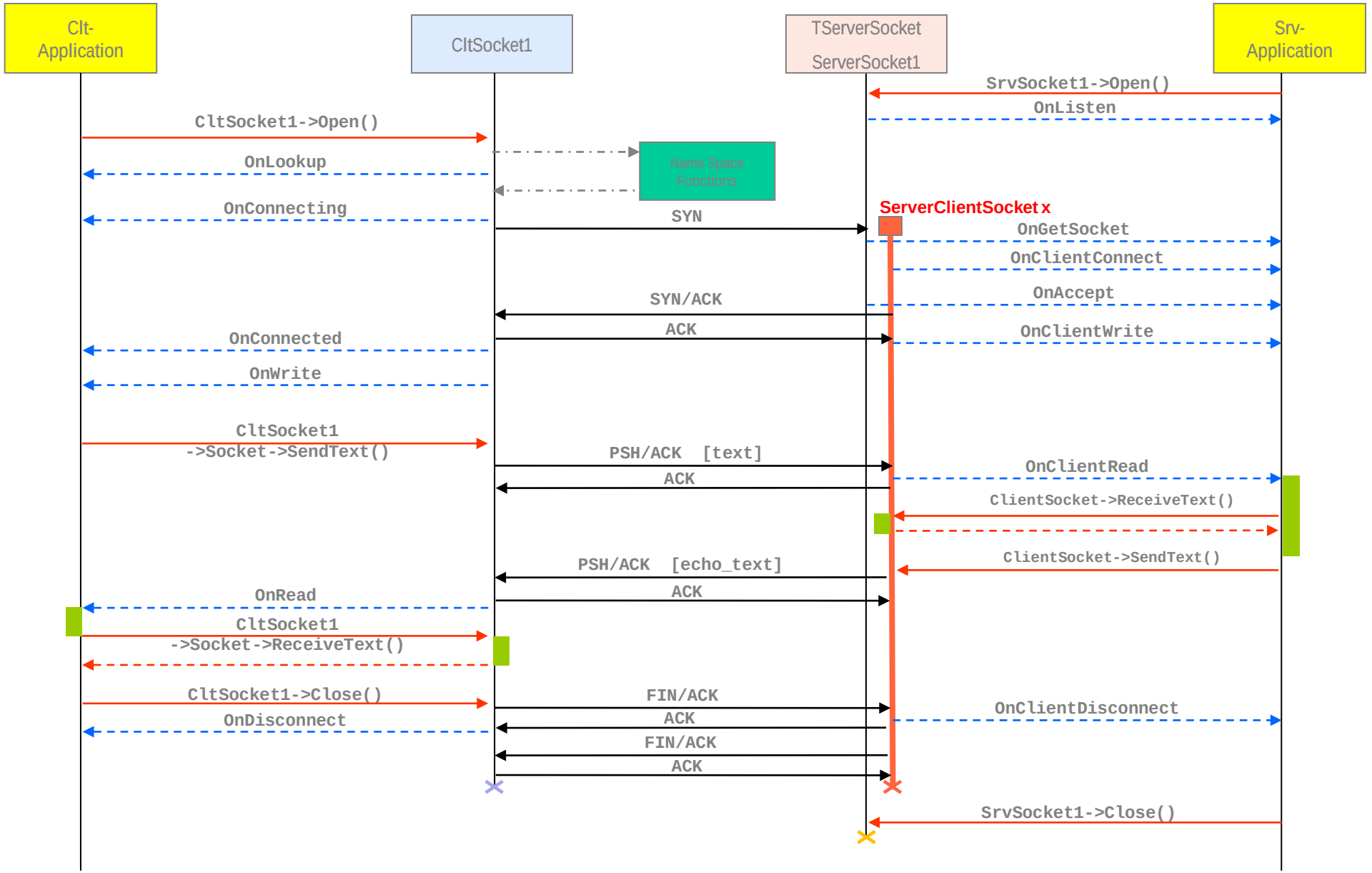
```
OnListen      //Der Socket befindet sich im Zustand Listening
OnGetSocket   //Neuer Clt-Verbindungswunsch
OnAccept      //Verbindungswunsch akzeptiert
OnClientConnect       //Client hat sich verbunden
OnClientDisconnect    //Verbindung zum Client beendet
OnClientWrite//Socket ist beschreibbar (ctNonBlocking)
OnClientRead //Daten wurden empfangen, können gelesen werden
OnError       //Fehlermeldung, Bedienung durch ErrorCode:=0
```



Properties von TServerSocket:

```
Active False |True //wenn true, wird beim Start Methode open aufgerufen
ServerType  ctBlocking | ctNonBlocking
   //bei Lese-/Schreibvorgang auf Vollendung warten | nicht warten
   //bei ctNonBlocking erzeugt Socket OnRead-, OnWrite-Ereignisse
Port    z.B. 21 | 80 | 25 | 5555
Service  z.B. ftp | http | smtp
```

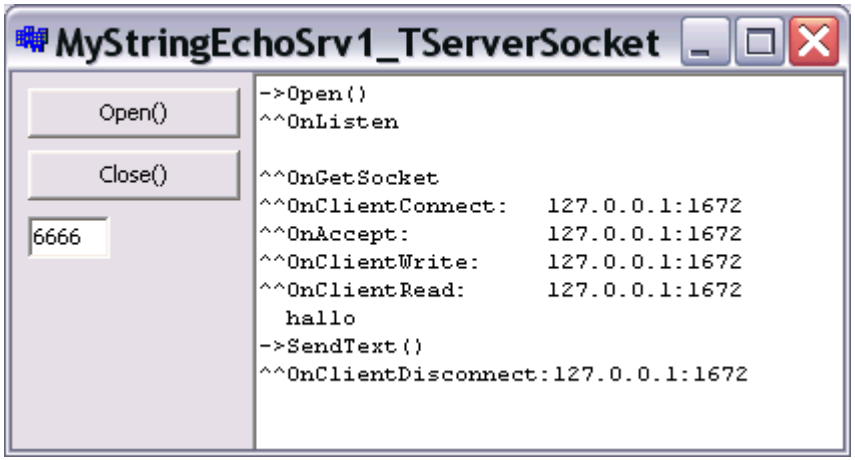
Was in den Sockets so abläuft



Programmierbeispiel: MyStringEchoSrv1_TServerSocket

Prof. L. Winkler, Hochschule Mittweida ::: win@hs-mittweida.de ::: www.telecom.hs-mittweida.de

- Ziel:
  - Implementierung eines Echo-Servers mit der Komponente TServerSocket.
- Mit Borland C++ Builder/Explorer ist nebenstehende Oberfläche für den Server zu programmieren.
- Es wurden folgende Objekte verwendet:
  - `TButton StartSrv, StopSrv,`
  - `TEdit Edit1`
  - `TRichEdit RichEdit1`
  - `TServerSocket ServerSocket1`
- Die ANLEITUNG und eine Beispiel-EXE finden Sie unter: www.telecom.hs-mittweida.de >Lehre DI-Direktstudium >Socketprogrammierung



Im obigen Beispiel sieht man Folgendes:
Aufruf der Funktion Open()
^^OnListen // Server wartet auf ankommende Verbindungen
^^OnGetSocket // Server hat für neue Verbindung ServerClientSocket erzeugt
^^OnClientConnect // ServerClientSocket meldet, Verbindung zum Clt aktiv
^^Accept// ServerSocket meldet, Verbindungswunsch eines Clients akzeptiert
^^OnClientWrite // ServerClientSocket meldet, ab jetzt Datensendung an Clt möglich
^^OnClientRead // ServerClientSocket meldet, Daten vom Clt sind da
Aufruf der Funktion SendText()
^^OnClientDisconnect // ServerClientSocket meldet, Clt hat Verbindung abgebaut

Fachbücher	
/TISCHER, JENNRICH/	**INTERNET Intern**, DATA BECKER 1997, ISBN: 3-8158-1160-0 //etwas teuer, aber ein sehr gutes Buch!
/STAINOV/	IPnG, das Internet-Protokoll der nächsten Generation, Intern. Thomson Publ., 1997, ISBN 3-8266-4018-7
/SIKORA/	**Technische Grundl. der Rechnerkommunikation**, Fachbuchverlag Leipzig, 2003, ISBN 3-446-22455-6
WWW	
/WINSOCK-REFERENCE/	http://msdn.microsoft.com/library/en-us/winsock/winsock/winsock_reference.asp?frame=true
/ERROR-CODES/	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winsock/winsock/windows_sockets_error_codes_2.asp //alle Fehlercodes in numerischer Reihenfolge und kurzen Erläuterungen
/SOCKETS.COM/	http://www.sockets.com/ //fast alles was man zur Socket-Programmierung benötigt, Stand 1998
/TANGENTSOFT/	http://tangentsoft.net/wskfaq/ //Nützliches zur Socket-Programmierung benötigt, Stand 2006
/C-WORKER/	http://www.c-worker.ch/
/HIGHSCORE.DE/	http://www.highscore.de/ //Online-Bücher zu Grundlagen der Programmierung in Java, C, Java-Script, UML, allerdings ohne Socketprogrammierung
/DELPHI-SOURCE/	http://www.dsdt.info/ //Delphi-Sprachgrundlagen, Tutorials und Tipps bis hin zu Insiderwissen
Standards	
/WSAPI/	**Windows Sockets 2, Application Programming Interface,** Revision 2.2.2, August 7, 1997
/WSSPI/	**Windows Sockets 2, Service Provider Interface,** Revision 2.2.2, August 7, 1997

Anlage A: Errors in Numerical Order

WSABASEERR (0) No Error
WSAEINTR (10004) Interrupted system call
WSAEBADF (10009) Bad file number
WSAEACCES (10013) Permission denied
WSAEFAULT (10014) Bad address
WSAEINVAL (10022) Invalid argument
WSAEMFILE (10024) Too many open files
WSAEWOULDBLOCK (10035) Operation would block
WSAEINPROGRESS (10036) Operation now in progress
WSAEALREADY (10037) Operation already in progress
WSAENOTSOCK (10038) Socket operation on non-socket
WSAEDESTADDRREQ (10039) Destination address required
WSAEMSGSIZE (10040) Message too long
WSAEPROTOTYPE (10041) Protocol wrong type for socket
WSAENOPROTOOPT (10042) Bad protocol option
WSAEPROTONOSUPPORT (10043) Protocol not supported
WSAESOCKTNOSUPPORT (10044) Socket type not supported
WSAEOPNOTSUPP (10045) Operation not supported on socket
WSAEPFNOSUPPORT (10046) Protocol family not supported
WSAEAFNOSUPPORT (10047) Address family not supported by protocol family
WSAEADDRINUSE (10048) Address already in use
WSAEADDRNOTAVAIL (10049) Can't assign requested address
WSAENETDOWN (10050) Network is down
WSAENETUNREACH (10051) Network is unreachable
WSAENETRESET (10052) Net dropped connection or reset
WSAECONNABORTED (10053) Software caused connection abort
WSAECONNRESET (10054) Connection reset by peer
WSAENOBUFS (10055) No buffer space available
WSAEISCONN (10056) Socket is already connected
WSAENOTCONN (10057) Socket is not connected
WSAESHUTDOWN (10058) Can't send after socket shutdown
WSAETOOMANYREFS (10059) Too many references, can't splice
WSAETIMEDOUT (10060) Connection timed out
WSAECONNREFUSED (10061) Connection refused
WSAELOOP (10062) Too many levels of symbolic links
WSAENAMETOOLONG (10063) File name too long
WSAEHOSTDOWN (10064) Host is down
WSAEHOSTUNREACH (10065) No Route to Host
WSAENOTEMPTY (10066) Directory not empty
WSAEPROCLIM (10067) Too many processes
WSAEUSERS (10068) Too many users
WSAEDQUOT (10069) Disc Quota Exceeded
WSAESTALE (10070) Stale NFS file handle
WSASYSNOTREADY (10091) Network SubSystem is unavailable
WSAVERNOTSUPPORTED (10092) WINSOCK DLL Version out of range
WSANOTINITIALISED (10093) Successful WSASTARTUP not yet performed
WSAEREMOTE (10071) Too many levels of remote in path
WSAHOST_NOT_FOUND (11001) Host not found
WSATRY_AGAIN (11002) Non-Authoritative Host not found
WSANO_RECOVERY (11003) Non-Recoverable errors: FORMERR, REFUSED, NOTIMP
WSANO_DATA (11004)* Valid name, no data record of requested type
WSANO_ADDRESS (11004)* No address, look for MX record

Anlage B: Wichtige Socket-Konstanten *(siehe winsock2.h)*

Gruppe	Konstante	Kommentar
Allg. Konstanten	SOMAXCONN=5	Max. Anzahl wartender Verbindungen beim SrvSock
	FD_SETSIZE=64	Arraygröße für Datenstruktur fd_set
Fehlermeldungen	SOCKET_ERROR=-1	
	INVALID_SOCKET=NOT(0)	
Adress-Familien	AF_UNIX=1	
	AF_INET=2	
Socket-Typen	SOCKET_STREAM=1	
	SOCKET_DGRAM=2	
	SOCKET_RAW=3	
Protokolle	IPPROTO_IP=0	
	IPPROTO_TCP=6	
	IPPROTO_UDP=17	
Adressen	INADDR_ANY=0	
	INADDR_LOOPBACK=$7F000001	
	INADDR_BROADCAST=-1	
Ports	IPPORT_ECHO=7	
	IPPORT_TELNET=23	
	IPPORT_TIMESERVER=37	
	IPPORT_NAMESERVER=42	