Windows Sockets

An Open Interface for Network Programming under Microsoft Windows

DI. Dr. Peter René Dietmüller

Institut für Informationsverarbeitung und Mikroprozessortechnik

Johannes Kepler Universität Linz

Überblick

- Geschichte
- Was ist Winsock?
- Kommunikation
- Winsock-Funktionen
- Beispiel
- Modi

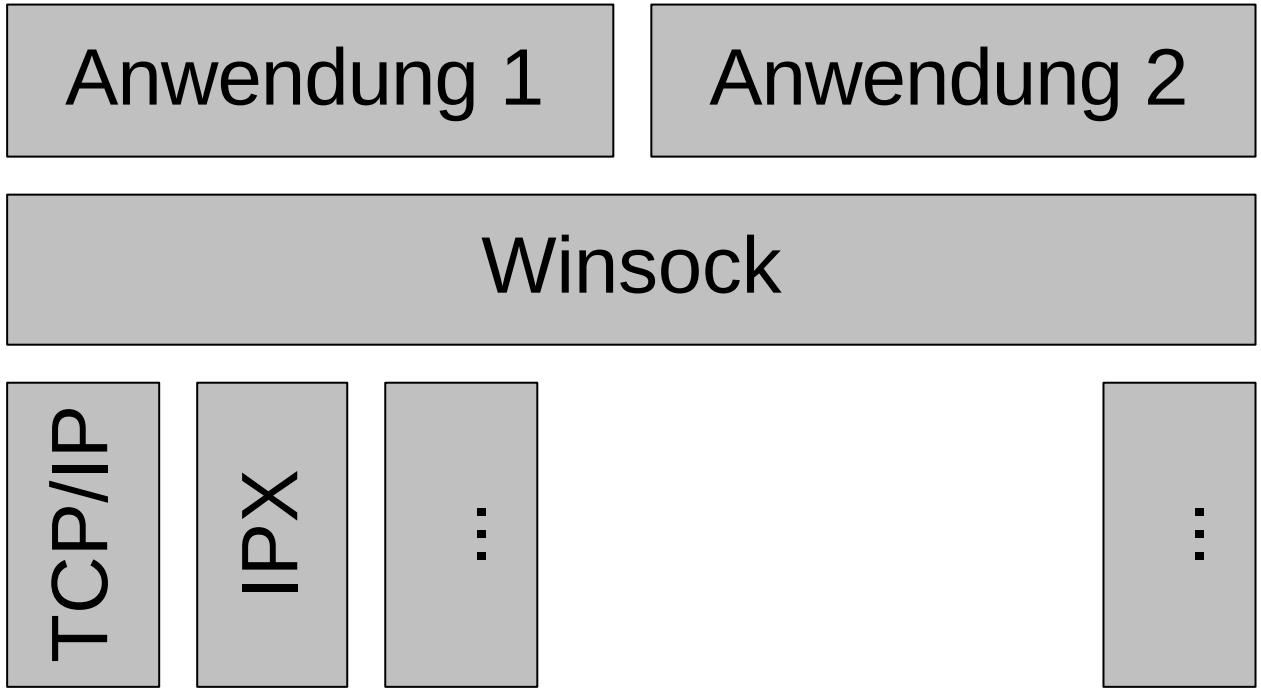
Geschichte

- Berkeley Sockets API
- Winsock Version 1.0 (Juni 1992)
- Winsock Version 1.1 (20.01.1993)
- Winsock Version 2.0 (Revision 2.0.8 vom 20.05.1995)

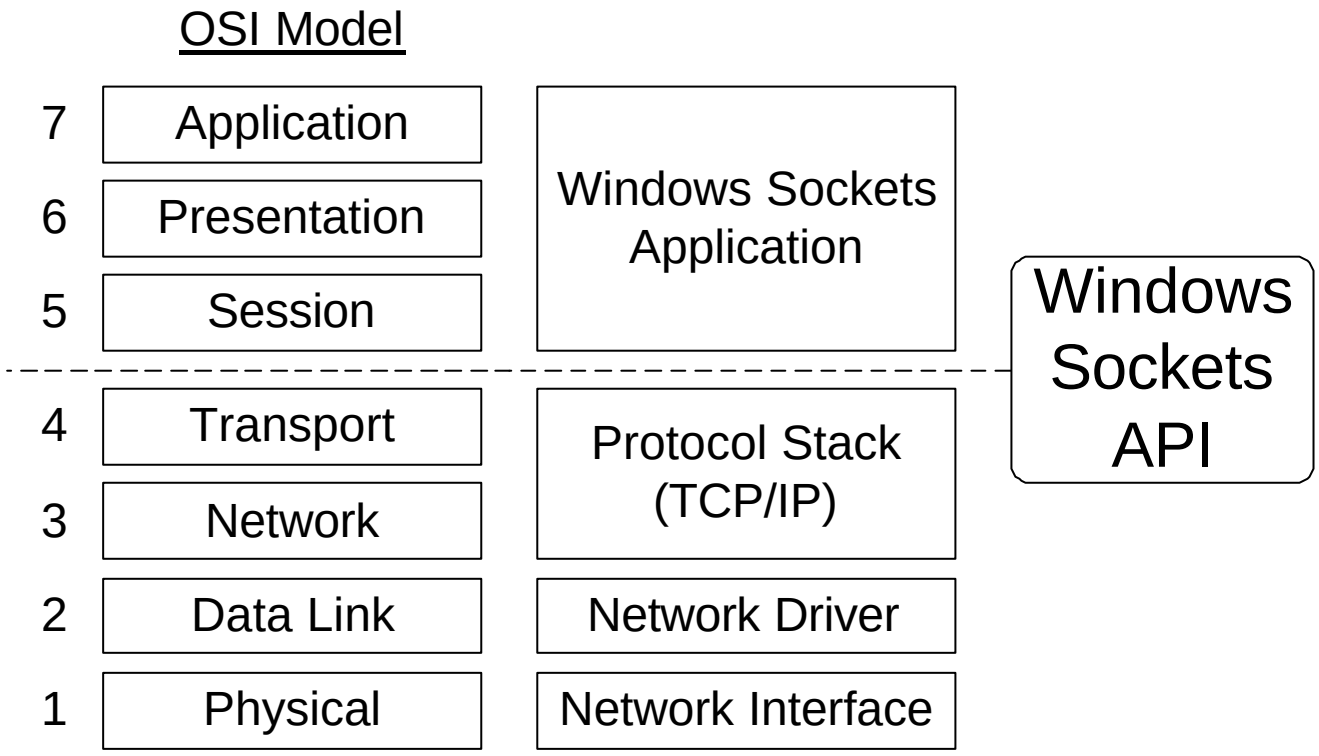
Was ist Winsock?

- Bibliothek von Funktionen zur Kommunikation (als DLL impl.)
- einheitliches API für Netzwerk-Anwendungen
- Schnittstelle zwischen Anwendungen und Protokolle
  - Die Anwendung ist für die Benutzerschnittstelle zuständig.
  - Das Protokoll übernimmt das Senden/Empfangen von Daten.
  - Winsock steht als Vermittler dazwischen.
- offene, frei verfügbare Schnittstelle ohne Lizenzgebühren
- Quellcode-kompatibel mit BSD-Sockets-Programmen
- Verfügbar: 16/32-Bit Windows, OS/2
- unterstützt verschiedene Protokolle

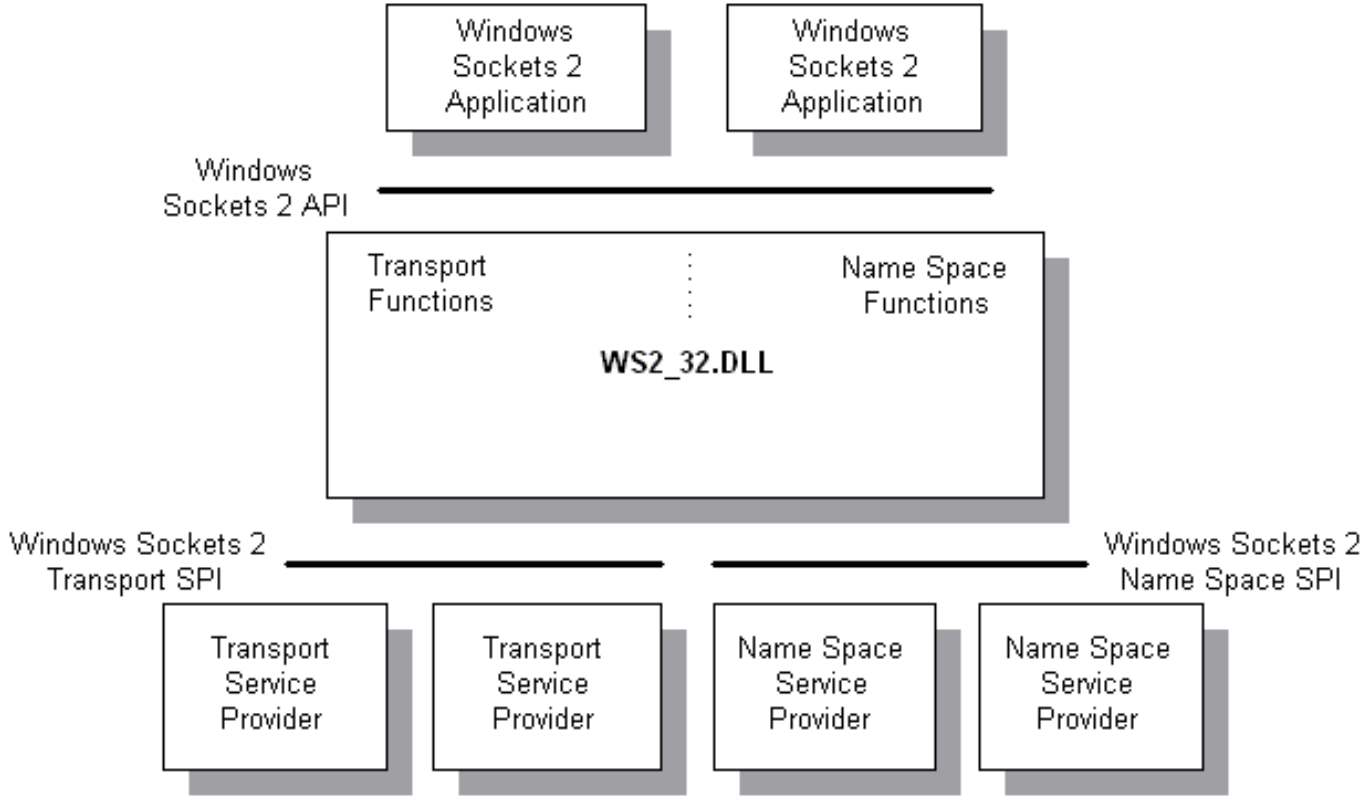
Was ist Winsock? (2)



Was ist Winsock? (3)



Winsock 2 - Architektur



Quelle: MSDN Oktober 2000

Sockets

- engl. Bezeichnung für Steckdose, Anschluß
- Sockets sind Endpunkte einer Kommunikation.
- Ein Programm kann ein oder mehrere solcher Anschlüsse öffnen.
- Mit einem Socket kann eine Verbindung zu einem anderen Socket aufgebaut werden oder auf eine Verbindung von einem anderen Socket gewartet werden.
- Die Adressierung eines Socket ist vom Protokoll abhängig.
- Im TCP/IP-Protokoll wird ein Socket durch die IP-Adresse des Rechners und durch eine Portnummer bestimmt.

Sockets (2)

Damit eine Kommunikation über einen Socket laufen kann, müssen folgende Informationen bereitgestellt werden:

- das Protokoll
- die lokale Adresse (für TCP: IP-Adresse, Portnummer) und
- die entfernte Adresse (für TCP: IP-Adresse, Portnummer).

<table>
<tr><td colspan="2">1) Protocol</td></tr>
<tr><td>2) Local IP
3) Local Port</td><td>4) Remote IP
5) Remote Port</td></tr>
</table>

Ports

Da auf einem Rechner viele verschiedene Programme laufen können, die über das Netzwerk kommunizieren, wird mit der Portnummer festgelegt, mit welchem Programm man kommunizieren möchte.

Eine Server-Anwendung wartet auf einem festgelegten Port auf eine Verbindung. Damit der Client mit dem Server kommunizieren kann, muß er genau auf diesem Port eine Verbindung zum Server aufbauen.

Einige Portnummer sind fix vergeben. Sie sind meist in einer Datei namens „services“ verzeichnet.

Standardisierte Ports

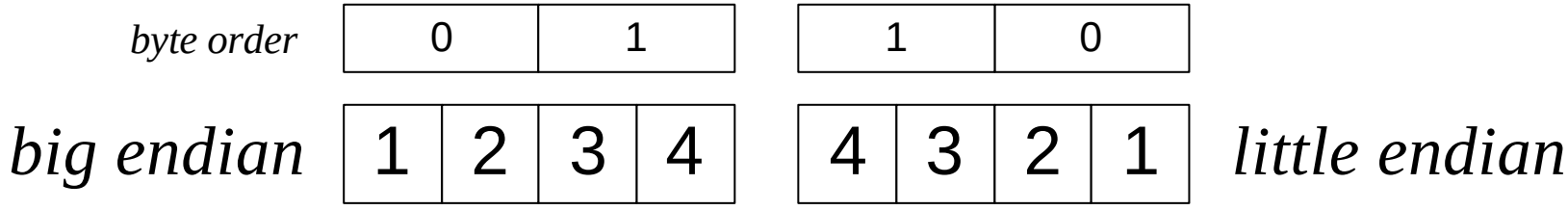
Keyword	Decimal	Description
echo	7/tcp	Echo
echo	7/udp	Echo
ftp-data	20/tcp	File Transfer [Default Data]
ftp-data	20/udp	File Transfer [Default Data]
ftp	21/tcp	File Transfer [Control]
ftp	21/udp	File Transfer [Control]
telnet	23/tcp	Telnet
telnet	23/udp	Telnet

…

Byte Ordering

Intel-Prozessoren speichern Multibyte-Werte in einer anderen Reihenfolge ab. Intel verwendet für seine Prozessoren little endian, während viele andere Herstelle big endian verwenden.

Über das Netzwerk wird in der Regel in „big endian“ versendet. Die meisten Winsock-Funktionen erwarten daher, daß Multibyte-Werte in big endian übergeben werden.



Kommunikation

- Streams über TCP
  - STD07: J. Postel. Transmission Control Protocol. September 1981. (RFC0793)
  - STD03: R. Braden. Host Requirements. October 1989. (RFC1122, RFC1123)
- Datagramme über UDP
  - STD06: J. Postel. User Datagram Protocol. August 1980. (RFC0768)
  - STD03: R. Braden. Host Requirements. October 1989. (RFC1122, RFC1123)

TCP-Kommunikation

Die Kommunikation mit Hilfe des Protokolls TCP kann mit Dateioperationen verglichen werden.

File I/O	Winsock
Datei öffnen	Socket öffnen
	Socket benennen
	Socket verbinden
Datei schreiben/lesen	Daten senden/empfangen
Datei schließen	Socket schließen

TCP-Kommunikation (2)

Die einzelnen Schritte unterscheiden sich, ob die Anwendung eine Server- oder Client-Anwendung ist.

Winsock	Client	Server
Socket öffnen	socket()	socket()
Socket benennen	-- / bind()	bind()
Socket verbinden	connect()	listen(), accept()
Daten senden/empfangen	send() / recv()	send() / recv()
Socket schließen	closesocket()	closesocket()

UDP-Kommunikation

Winsock	Client	Server
Socket öffnen	socket()	socket()
Socket benennen	--	-- / bind()
Socket verbinden	-- / connect()	--
Daten senden	sendto() / send()	sendto() / send()
Daten empfangen	recvfrom() / recv()	recvfrom() / recv()
Socket schließen	closesocket()	closesocket()

socket()

Diese Funktion öffnet einen Socket für ein bestimmtes Protokoll. Wenn die Funktion erfolgreich ist, gibt sie einen Socket Handle zurück, sonst den Wert `INVALID_SOCKET`.

```
SOCKET PASCAL FAR socket(
    int af          /* address family */
    int type        /* socket type    */
    int protocol); /* protocol name  */
```

Beispiel:

```
socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)
```

öffnet einen Socket für das Protokoll TCP

socket() (2)

Mit socket() wird ein Socket eröffnet und das Protokoll für diesen Socket spezifiziert.

1) Protocol	
2) Local IP 3) Local Port	4) Remote IP 5) Remote Port

sockaddr structure

Um ein Socket zu adressieren, wird folgende Struktur verwendet.

```
struct sockaddr {
  u_short sa_family;    /* address family */
  char    sa_data[14];  /* undefined      */
}
```

Für jedes Protokoll gibt es eine spezialisierte Version dieser Struktur, da die Adressierung der Protokolle nicht gleich ist. Für die Internetprotokolle gibt es die Struktur sockaddr_in.

sockaddr_in structure

```
struct sockaddr_in {
  short           sin_family;
  u_short         sin_port;
  struct in_addr  sin_addr;
  char            sin_zero[8];
}
```

sin_family: address family

sin_port: 16-Bit port number in network order

sin_addr: 32-Bit IP-address in network order

bind()

Mit bind() wird einem Socket die lokale Adresse des Socket mitgeteilt.

```
int PASCAL FAR bind(
  SOCKET s,                    /* socket handle */
  struct sockaddr FAR *addr, /* local addr. */
  int namelen);                /* structure length */
```

Um die lokale IP-Adresse nicht ermitteln zu müssen, kann statt der IP-Adresse die Konstante INADDR_ANY verwendet werden.

Eine Server-Anwendung muß diese Funktion aufrufen, um den Port zu setzen, auf dem sie auf einlangende Verbindungen wartet. Für eine Client-Anwendung ist es nicht notwendig, diese Funktion aufzurufen.

bind() (2)

Mit bind() wird die lokale Adresse des Socket spezifiziert.

1) Protocol	
2) Local IP **3) Local Port**	4) Remote IP 5) Remote Port

listen()

Mit listen() wird auf einem bestimmten Socket (und damit auf einem bestimmten Port) auf ankommende Verbindungen gewartet.

```
int PASCAL FAR listen(
  SOCKET s,
  int backlog);
```

backlog: Länge der wartenden Verbindungsanforderungen

connect()

Diese Funktion baut eine Verbindung zu einem anderen Socket auf.

```
int PASCAL FAR connect(
  SOCKET s,                    /* socket handle */
  struct sockaddr FAR *addr,  /* rem. addr. */
  int namelen);               /* structure length */
```

connect() (2)

Wenn bind() für diesen Socket nicht aufgerufen wurde, wird die lokale IP-Adresse und eine freie lokale Portnummer in den Socket eingetragen.

<table>
<tr><th colspan="2">1) Protocol</th></tr>
<tr><td>2) Local IP
3) Local Port</td><td>4) Remote IP
5) Remote Port</td></tr>
</table>

accept()

accept() wird nach der Funktion listen() aufgerufen und erzeugt einen neuen Socket für eine wartende Verbindung.

```
SOCKET PASCAL FAR accept(
  SOCKET s,              /* a listening socket */
  struct sockaddr FAR *addr, /* incoming s. */
  int FAR *addrlen);  /* length of sockaddr */
```

Der Socket, der auf eingehende Verbindung wartet, („listening socket“) bleibt von dieser Funktion unberührt und horcht weiterhin auf ankommende Verbindungen.

accept() (2)

accept() liefert einen neuen Socket, dessen lokale Adresse und Einstellungen mit dem „listening socket“ ident sind. Die entfernte Adresse wird aus dem Paket gewonnen, mit dem die Verbindung vom entfernten Rechner aufgebaut wird.

1) Protocol	
2) Local IP **3) Local Port**	**4) Remote IP** **5) Remote Port**

send()

send() sendet Daten über einen Socket. Aus der Funktion wird erst zurückgekehrt, wenn die Daten gesendet wurden.

```
int FAR PASCAL send(
  SOCKET s,
  const char FAR *buf,  /* Daten             */
  int len,              /* Länge der Daten */
  int flags);
```

sendto()

sendto() sendet Daten über einen „nicht verbundenen“ Socket. Aus der Funktion wird erst zurückgekehrt, wenn die Daten gesendet wurden.

```
int FAR PASCAL sendto(
  SOCKET s,
  const char FAR *buf,     /* Daten           */
  int len,                 /* Länge der Daten*/
  int flags,               /* Sendeoptionen  */
  struct sockaddr FAR *to,/* Zieladresse     */
  int tolen);              /* Länge der Ziel.*/
```

recv()

Empfängt ein Socket Daten, werden die Daten von der Winsock-Bibliothek in einer Queue zwischengespeichert. Mit recv() können die Daten vom Programm abgeholt werden.

```
int PASCAL FAR recv(
  SOCKET s,
  char FAR *buf,  /* Puffer für empf. Daten */
  int len,        /* Länge des Puffers      */
  int flags);
```

recvfrom()

Mit recvfrom() können Daten von einem „nicht verbundenen“ Socket gelesen werden. In die beiden letzten Parameter wird die Adresse des Absenders gespeichert.

```
int PASCAL FAR recvfrom(
  SOCKET s,
  char FAR *buf,                /* empf. Daten */
  int len,                      /* Länge der D.*/
  int flags,                    /* Optionen    */
  struct sockaddr FAR *from,    /* Senderadr.  */
  int FAR *fromlen);            /* Länge Sa.   */
```

closesocket()

closesocket() baut eine Verbindung zu einem anderen Socket ab und schließt den Socket.

```
int PASCAL FAR closesocket(SOCKET s);
```

shutdown()

Bevor man mit closesocket() eine Verbindung abbaut, sollte man mit shutdown() die Verbindung „teilweise“ abbauen. Der beteiligte Socket wird durch diese Funktion nicht geschlossen.

```
int PASCAL FAR shutdown(
  SOCKET s,
  int how);
```

how: Art des Shutdown

1.....Es werden keine Pakete mehr empfangen.

2.....Es können keine Pakete mehr gesendet werden.

3.....Beides.

WSAStartup()

WSAStartup() muß vor jeder anderen Winsock-Funktion aufgerufen werden, um die Winsock-Bibliothek zu initialisieren.

```
int PASCAL FAR WSAStartup(
  WORD wVersionRequired,
  LPWSADATA lpWSAData);
```

Über den Parameter *wVersionRequired* gibt eine Applikation an, welche Version der Winsock-Bibliothek benötigt wird. Im High-Byte wird die Minor- und im Low-Byte die Major-Versionsnummer angegeben.

Im Parameter *lpWSAData* liefert die Winsock-Bibliothek Details über die Implementierung der Bibliothek.

WSACleanup()

Muß als letzte Winsock-Funktion aufgerufen werden, damit die Winsock-Bibliothek „aufräumen“ kann. Für jeden Aufruf der Funktion WSAStartup() muß einen Aufruf von WSACleanup() geben.

```
int PASCAL FAR WSACleanup(void);
```

select()

Mit der Funktion select() kann man den Status eines oder mehrerer Sockets ermitteln. Damit ist es zum Beispiel möglich eine gewisse Zeit auf das Eintreffen von Daten zu warten und wenn keine Daten eintreffen einen Timeout auszulösen.

```
int PASCAL FAR select(
  int nfds,       /* not used, compatibility */
  fd_set FAR *readfds,          /* readable? */
  fd_set FAR *writefds,         /* writable? */
  fd_set FAR *exceptfds       /* exceptions? */
  struct timeval FAR *timeout);  /* timeout */
```

gethostbyname()

Diese Funktion liefert unter anderem die IP-Adresse eines Rechners.

```
struct hostent FAR * PASCAL FAR
  gethostbyname(char FAR * name);

struc hostent {
  char FAR * h_name;  /* official host name */
  char FAR * FAR * h_aliases;    /* aliases */
  short h_addrtype;        /* address family */
  short h_length;      /* length of address */
  char FAR * FAR * h_addr_list; /*addresses */
}
```

Beispiel WWWGet

WWWGet ist ein einfaches Windowsprogramm, das eine HTML-Seite von einem WWW-Server holt.

Dazu wird eine Verbindung zu einem WWW-Server auf Port 80 aufgebaut und das Kommando „GET …“ verschickt. Der WWW-Server sendet die im GET-Kommando angegebene HTML-Seite als Antwort zurück und schließt die Verbindung. Die empfangenen Daten werden ausgegeben.

WWWGet (2)

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

static int http_rcvGet(SOCKET s) {

  char buffer[4096];
  int  i;
  int  len;

  len = 1;
  while ((len != 0) && (len != SOCKET_ERROR)) {
    len = recv(s, buffer, sizeof(buffer), 0);
    for(i = 0; i < len; i++) putchar(buffer[i]);
  }

  return (len == 0);

}
```

WWWGet (3)

```
static int http_sendGet(SOCKET s, const char *url) {

  char cmd[200];

  strcpy(cmd, "GET ");
  strcat(cmd, url);
  strcat(cmd, " HTTP/1.0");
  strcat(cmd, "\n\n");
  send(s, cmd, strlen(cmd), 0);
  return 1;

}
```

WWWGet (4)

```
static void http_get(const char *server, const char *url) {

  int                err;
  struct hostent     *he;
  SOCKET             s;
  struct sockaddr_in sin;
  WORD               vr;
  WSADATA            wsaData;

  /* -- Winsock initialisieren -- */
  vr = MAKEWORD(2, 2);
  err = WSAStartup(vr, &wsaData);
  if (err != 0) {
    printf("Fehler beim Initialisieren der Winsock-Bibliothek.\n");
    return;
  }
```

WWWGet (5)

```
/* -- Namen auflösen -- */
he = gethostbyname(server);
if (!he) {
  printf("Der Server %s konnte nicht gefunden werden.\n", server);
  WSACleanup();
  return;
}

/* -- Socket anlegen -- */
s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s == INVALID_SOCKET) {
  printf("Ein Socket konnte nicht angelegt werden.\n");
  WSACleanup();
  return;
}
```

WWWGet (6)

```
  /* -- Verbindung aufbauen -- */
  memset(&sin, 0, sizeof(sin));
  sin.sin_family = PF_INET;
  sin.sin_port   = htons(80);
  memcpy(&sin.sin_addr, he->h_addr_list[0], sizeof(sin.sin_addr));
  err = connect(s, (LPSOCKADDR)&sin, sizeof(sin));
  if (err != SOCKET_ERROR) {
    http_sendGet(s, url);  /* Befehl an WWW-Server senden */
    if (!http_rcvGet(s)) { /* Seite empfangen und ausgeben */
      printf("Fehler beim Empfangen der URL %s.\n", url);
    }
    closesocket(s);        /* Verbindung schließen */
  } else {
    printf("Keine Verbindung: Server %s, Port %d.\n", server, 80);
  }
  WSACleanup();
}
```

WWWGet (7)

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

  printf("WWWGet V1.1\n\n");

  /* -- Parameter -- */
  if (argc != 3) {
    printf("Sie haben zu wenig Parameter angegeben.\n");
    Usage();
  } else {
    http_get(argv[1], argv[2]);
  }
  return 0;
}
```

Modi

- Blocking mode

  Die Winsock-Funktionen blockieren das Programm. In recv() wird zum Beispiel so lange gewartet, bis Daten empfangen wurden.

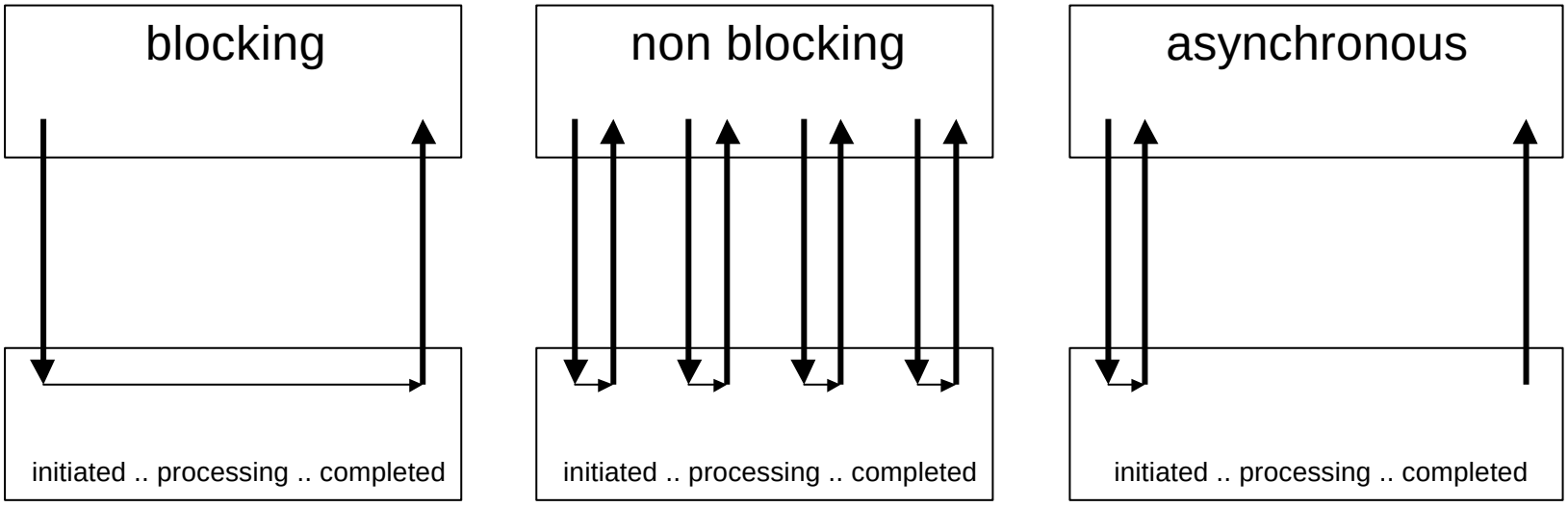
- Nonblocking mode (polling)

  Die Winsock-Funktionen blockieren das Programm nicht. Aus recv() wird zum Beispiel sofort zurückgekehrt, egal ob Daten empfangen wurden oder nicht.

- Asynchronous mode

  Die Winsock-Funktionen versenden Nachrichten beim Eintreffen bestimmter Ereignisse.

Modi (2)


blocking
initiated .. processing .. completed
non blocking
initiated .. processing .. completed
asynchronous
initiated .. processing .. completed

Quellen

- Bob Quinn, Dave Shute, „Windows Sockets Network Programming“, Addison Wesley, 1996
- Bob Quinn, „Winsock Development Information“, http://www.sockets.com
- J. Reynolds, J. Postel, „RFC1700 - ASSIGNED NUMBERS“, 1994
- MSDN Oktober 2000, Microsoft Corporation