Ingenieurschule Bern HTL

PASCAL Treiber für Novell-NetWare

Benutzerdokumentation

Diplomarbeit 1991

René Gilomen

2/12/2023 Version 1.1

Inhaltsverzeichnis

11

IPX_Send

12

IPX_Receive

14

IPX_Done

16

IPX_Internetwork_Address

19

IPX_To_Addr

1. Einleitung

Die hier beschrieben Turbo-Pascal Bibliothek ermöglicht es dem Benutzer, Daten über ein Netzwerk zu senden und zu empfangen.

Ausserdem wurden Aspekte des "Multitasking" berücksichtigt, d.h. der Benutzer ist während den Funktionen Senden und Empfangen in der Applikation nicht blockiert.

Es wurde darauf geachtet, dass die Routinen möglichst einfach einzusetzen sind.
Es wird jedoch angenommen, dass der Anwender das Betriebssystem, die Programmiersprache Turbo-Pascal und die Ethernetsoftware zu der verwendeten Hardware kennt. Grundlegende Kenntnisse von Netzwerken werden ebenfalls vorausgesetzt.

2. Novell-NetWare

Novell-NetWare [1] ist ein sehr verbreitetes Netzwerkbetriebssystem.
Es basiert auf dem Xerox Network System (XNS) [2].
Ein Grund für die gute Akzeptanz von NetWare ist die grosse Anzahl verschiedener Versionen, die erhältlich sind. Zum Beispiel Entry Level System (ELS) I und II speziell für kleine Netzwerke (4 - 8 Benutzer).
NetWare 386, Version 3.1 für 386 Mikroprozessoren, unterstützt bis zu 250 Benutzer.
Es sind ebenfalls Pakete erhältlich, die es erlauben, eine DEC-VAX als Server einzusetzen.
Die NetWare Komponenten können gut im OSI-Modell abgebildet werden [Bild 1].
Auf dem "Physical-" und "Data Link-Layer" unterstützt Novell-NetWare Ethernet, IEEE 802.3, IEEE 802.5, ARCNET und viele andere.
"Network-" und "Transport-Layer" werden durch das Internetwork Packet Exchange (IPX) und Sequenced Packet Exchange (SPX) abgedeckt.
Der "Session-Layer" ist durch das Network Basic Input Output System (NetBIOS) realisiert. "Presentation-" und "Application-Layer" werden durch NetWare Core Protocols (NCP), Value -Added Services, DOS-Shell und verschiedene Benutzerapplikationen repräsentiert.

Application

Presentation

NCP / DOS-Shell

Applikationen

Session

NetBIOS

Transport

SPX

Network

IPX

Data Link

CSMA/CD

Physical

802.3 / 802.5

Unterstüzte NW

Bild 1.

Anwendungen können auf verschiedene Arten auf das Netzwerk zugreifen

[Bild 2].
Das "Network Layer" Interface ist IPX, das eine verbindungslose Kommunikation zwischen Stationen erlaubt.
"Transport Layer" Interface ist SPX welches eine verbindungsorientierte Kommunikation unterstützt.
NetBIOS kompatible Anwendungen können auf den "Session Layer" direkt zugreifen.

Die "Workstation Shell" ist DOS File kompatibel. Die Kommunikation erfolgt von der Arbeitsstation zum Server. Eine direkte Kommunikation von Arbeitsstation zu Arbeitsstation ist so nicht möglich.

Workstation

Application

Shell
NetBIOS

Session

Emulator
SPX

Transport

IPX
Network

Das hier beschriebene Softwareprodukt **PASCAL Treiber für Novell-NetWare** ermöglicht dem Benutzer die Kommunikation mit Arbeitsstationen über ein Ethernet Netzwerk IEEE 802.3. Es werden dem Benutzer bestimmte Routinen zum Aufruf aus Turbo-Pascal zur Verfügung gestellt. Es ist eine Schnittstelle zum "Network-Layer" der bei Novell-NetWare mit IPX realisiert ist.

IPX ist ein verbindungsloses Protokoll, d.h. wenn eine Arbeitsstation mit einer anderen kommunizieren will, muss keine Verbindung mit der entsprechenden Station aufgebaut werden. IPX-Pakete enthalten eine Adresse und werden so zur Zielstation gesendet. Jedes Paket ist eine Einheit für sich und hat keine logische Beziehung zu einem anderen Paket.
IPX arbeitet mit sogenannten Kommunikationssockeln, einer Art Prozess, der vor dem Senden oder Empfangen aufgebaut werden muss. Beim Datentransfer muss immer die vollständige Netzwerkadresse angeben werden, inklusive der Sockelnummer.

Vorteile von IPX
-
-
aufgebaut
-

Nachteil
-

3. Benutzung der Turbo-Pascal Bibliothek

Die Bibliothek wird mit Turbo-Pascal V6.0 [3] benutzt. Mit älteren Versionen von Turbo-Pascal ist die Bibliothek nicht lauffähig.
Turbo-Pascal unterstützt sogenannte "UNITS", das sind Bibliotheken, die separat compiliert abgelegt sind und nur noch zum Benutzerprogramm dazu gelinkt werden müssen.
Die "UNIT" wird mit dem Turbo-Pascal Kommando "USES IPX" in den Code des Programmes eingebunden.

Es müssen mindestens zwei IBM kompatible PC´s mit eingebauter Ethernetkarte zur Verfügung stehen. Sie sollten über ein Netzwerk (IEEE 802.3) miteinander verbunden sein. Ein Server wird nicht benötigt, da es sich um eine Punkt-zu-Punkt Kommunikation handelt (Peer to Peer). Ebenfalls muss ein IPX-Treiber im Speicher geladen sein. Die Bibliothek wurde mit IPX Version 2.12 getestet.

4. Datenstrukturen

Für den Aufruf einiger der nachstehenden Routinen wird eine gemeinsame Datenstruktur verwendet, welche Daten und deren Länge in einem "RECORD" der Programmiersprache Turbo-Pascal enthält:

=

Data
: Data_Packet;

Length

END;

Für die Netzwerkadresse wird ebenfalls eine "RECORD" Datenstruktur benötigt.

RECORD

Network

Node
: S6Byte;

Socket

END;

Data_Packet, S4Byte und S6Byte sind Typen, die dem Benutzer zur Verfügung stehen. Data_Packet hat die Grösse von 546 Bytes, S4Byte 4 Bytes und S6Byte 6 Bytes.
Falls in der Beschreibung Halb-Oktetts als Parameter verlangt sind, werden keine ASCII-Zeichen akzeptiert. Es können also bei dieser Art von Parametern sämtliche Werte von Null ($00) bis 255 ($FF) als binäre Werte

in einem Oktett mitgegeben werden.

Zusätzlich stehen folgende globale Konstanten zur Verfügung:

- MAX_SOCKETS

- MAX_DATA_SIZE Maximale Länge der Daten.

- NET_LENGTH
Länge Netzwerkadresse (in Bytes).

- NODE_LENGTH

5. Funktionenbibliothek

FUNCTION IPX_Setup : BYTE;

Beschreibung:
Die Routine initialisiert die IPX-Software und deren Funktion.

Parameter:
Rückgabewert

Die Routine bereitet den IPX-Treiber vor und setzt alle nötigen Parameter innerhalb der Software. Es können danach alle Funktionen der Turbo-Pascal-Bibliothek angesprochen werden.
WICHTIG:
Programm aufgerufen werden.

Die Routine kann folgende Fehlercodes zurückgeben:

SUCCESS
Die Software konnte erfolgreich initialisiert werden.

DEVICE_SW_ERROR
Der Netzwerktreiber kann nicht angesprochen werden oder ist nicht geladen.

Bsp.

```
VAR
:
:
State := IPX_Setup;
IF State <> SUCCESS THEN
BEGIN


END;
:
:
```

FUNCTION IPX_Open_Socket (VAR Socket : WORD) : BYTE;

Beschreibung:
Die Routine öffnet einen Kommunikationssockel.

Parameter:
Socket
= Nummer des Sockels, der eröffnet
werden soll.

OUT
= Nummer des Sockels, der effektiv
geöffnet wurde.

Rückgabewert

Senden und Empfangen von Datenpaketen erfolgen immer über einen Kommunikationssockel. Als Parameter muss eine Sockelnummer angegeben werden (in hexadezimaler Form). Uebergibt man als Parameter Null ($0000), wird dem Sockel eine Nummer zugewiesen.
VORSICHT:
Kommunikationssockel
reserviert (siehe Novell Dokumentation [1]).

Die Routine kann folgende Fehlercodes zurückgeben:

SUCCESS
Der Kommunikationssockel konnte erfolgreich eröffnet werden.

PARAMETER_ERROR
Der Kommunikationssockel ist bereits eröffnet worden.

DEVICE_SW_ERROR
IPX ist nicht korrekt ansprechbar.

SOCKET_TABLE_FULL
Die maximale Anzahl Kommunikationssockel ist erreicht.

Bsp.

```
VAR

Socket : WORD;

:
:
 IPX_Setup ....
:
:
Socket := $8000;
:
State := IPX_Open_Socket (Socket);
IF State <> SUCCESS THEN
BEGIN


END;
:
:
```

FUNCTION IPX_Close_Socket (Socket : WORD) : BYTE;

Beschreibung:
Die Routine schliesst einen Kommunikationssockel.

Parameter:
Socket
= Nummer des Sockels, der
geschlossen werden soll.

OUT
= Fehlercode

Der Kommmunikationssockel wird geschlossen und die internen Variablen abgebaut (Sockelnummer in hexadezimaler Form).

Die Routine kann folgende Fehlercodes zurückgeben:

SUCCESS
Der Kommunikationssockel konnte erfolgreich geschlossen werden.

PARAMETER_ERROR
Die angegebe Sockelnummer existiert nicht.

Bsp.

```
VAR

Socket : WORD;

:
:
IPX_Setup ...
:
IPX_Open_Socket ...
:
State := IPX_Close_Socket (Socket);
IF State <> SUCCESS THEN
BEGIN


END;
:
:
```

```
FUNCTION IPX_Send ( Socket

: Network_Address;

 Buffer

        ) : BYTE;
```

Beschreibung:
Die Routine dient zum Senden von Daten an eine oder mehrere Gegenstationen.

Parameter:

Socket
= Sockelnummer, auf der gesendet werden soll.

Dest_Addr
= Vollständige Netzwerkadresse der Gegenstation(en).

Buffer
= Daten die gesendet werden und dessen Länge.

OUT
= Fehlercode

Die Sockelnummer muss in hexadezimaler Form angegeben werden. Die Daten können entweder an eine Gegenstation oder mittels "Broadcast" an alle Stationen gesendet werden. Die Netzwerkadresse muss vollständig angegeben werden. Die Felder "Network" und "Node" von "Dest_Addr" werden in Halb-Oktetts angegeben. Ist die Gegenstation auf demselben Netzwerk, kann das Feld "Network" mit Null ($0000) initialisiert werden. Die Daten dürfen eine maximale Länge von 546 Bytes haben. Die Länge muss spezifiziert werden.

Die Routine kann folgende Fehlercodes zurückgeben:

SUCCESS
Die verlangte Funktion wurde von IPX entgegengenommen.

PARAMETER_ERROR
Sockel existiert nicht, Netzwerkadresse nicht vollständig oder Buffer nicht korrekt.

NO_DESTINATION
Weg zur Gegenstation ist nicht bekannt.

Bsp.

```
VAR


Destination

Daten
: SData;

:
:
IPX_Setup ...
:
IPX_Open_Socket ...
:
WITH Destination DO
BEGIN









END;
:
WITH Daten DO
BEGIN


END;
```

```
:
State := IPX_Send (Socket,Destination, Daten);
IF State <> SUCCESS THEN
BEGIN



END;
:
:
```

FUNCTION IPX_Receive (Socket : WORD) : BYTE;

Beschreibung:
Die Routine dient zum Empfangen von Daten einer Gegenstation. Die Daten können, wenn das Kommando beendet ist, mit der Funktion IPX_Done vom Netzwerk abgeholt werden.

Parameter:
Socket
= Sockelnummer, auf der empfangen werden soll.

OUT
= Fehlercode

Die Sockelnummer muss in hexadezimaler Form angegeben werden. Mit der Funktion IPX_Done kann überprüft werden, ob Daten vom Netzwerk eingetroffen sind. Die Daten können ebenfalls mit dieser Funktion abgeholt werden.

Die Routine kann folgende Fehlercodes zurückgeben:

SUCCESS
Die verlangte Funktion wurde von IPX entgegengenommen.

PARAMETER_ERROR
Der angebene Kommunikationssockel existiert nicht.

DEVICE_SW_ERROR
IPX ist nicht korrekt ansprechbar.

Bsp.

```
VAR

Socket : WORD;

:
:
IPX_Setup ...
:
IPX_Open_Socket ...
:
:
State := IPX_Receive (Socket);
IF State <> SUCCESS THEN
BEGIN



END;
:
:
```

```
FUNCTION IPX_Done ( Socket
   : WORD;

    Code
   : BYTE;

    VAR Source_Addr


   : SData

         ) : BYTE;
```

Beschreibung:
Die Funktion liefert den Status einer vorher abgesetzten Routine. Zurückgegeben wird, ob die Routine schon beendet ist oder nicht sowie eventuelle Daten.

Parameter:

Socket
= Sockelnummer, auf der die Funktion ausgeführt werden soll.

Code
= Routine, deren Status überprüft werden soll.

OUT
= Vollständige Netzwerkadresse der Gegenstation, von der Daten eingetroffen sind.

Buffer
= Buffer, in dem eventuelle Daten abgelegt werden können.

Rückgabewert

Die Funktion liefert den Status einer vorher abgesetzten Routine. Es wird dabei zurückgegeben, ob die Routine bereits beendet ist oder nicht. Die Sockelnummer muss in hexadezimaler Form angegeben werden. ImPara-meter Source_Addr ist die vollständige Netzwerkadresse der Gegenstation gespeichert. Die Felder "Network" und "Node" sind in Halb-Oktetts abgelegt. Eventuelle Daten werden in "Buffer" abgelegt.
Das Feld "Code" definiert die abgesetzte Routine. "Code" kann dabei folgende Werte annehmen:

SEND
für IPX_Send
RECEIVE
für IPX_Receive

Die Routine kann folgende Fehlercodes zurückgeben:

SUCCESS
Die spezifizierte Funktion wurde erfolgreich beendet.

NOT_ENDED
Die Funktion ist noch nicht beendet.

PARAMETER_ERROR
Ungültiger Parameter beim Aufruf der Routine oder der Kommunikationssockel existiert nicht.

DEVICE_SW_ERROR
Die IPX-Software kann nicht angesprochen werden.

DEVICE_HW_ERROR
Es ist physikalisch unmöglich, ein Paket zu senden. Hardware oder Netzwerk defekt.

PACKET_BAD
Paketgrösse ist nicht korrekt (kleiner als 30 Byte oder grösser als 546 Byte).

PACKET_UNDELIVERIABLE
Paket kann nicht ausgeliefert werden (z.B. Zielstation nicht bekannt).

PACKET_OVERFLOW
Das empfangene Paket ist zu gross, um im Buffer abzulegen.

Bsp.

```
VAR
: BYTE;

Socket

Adresse

Daten
: SData;

:
:
IPX_Setup ...
:
IPX_Open_Socket ...
:
IPX_Send ...
:
State := IPX_Done (Socket, SEND, Adresse, Daten);
WHILE State = NOT_ENDED DO

:
:

oder

:
:
IPX_Receive ...
:
State := IPX_Done (Socket, RECEIVE, Adresse, Daten);
IF State = SUCCESS THEN
BEGIN

:
:
```

```
FUNCTION IPX_Internetwork_Address ( VAR Network

    VAR Node

  ) : BYTE;
```

Beschreibung:
Die Funktion liefert die Internetzwerkadresse der jeweiligen Station.

Parameter :

OUT
= Netzwerkadresse

Node
= Knotenadresse

Rückgabewert

Die Routine liefert die Internetzwerkadresse, der Station, auf der sie ausgeführt wird.

Die Routine kann folgende Fehlercodes zurückgeben:

SUCCESS
Die Funktion konnte erfolgreich ausgeführt werden.

Bsp.

```
VAR
: BYTE;

Net

Node
:
:
IPX_Setup ...
:
State := IPX_Internetwork_Address (Net,Node);
:
:
```

```
FUNCTION IPX_To_Addr ( Network
: String;

    Node
: String;

    Socket
: String;

    VAR Addr

   ) : BYTE;
```

Beschreibung :
Die Routine konvertiert die Eingabestrings in die Datenstruktur Network_Address.

Parameter :

IN
= Netzwerkadresse

Node
= Knotenadresse

Socket
= Sockelnummer

OUT
= Konvertierte
Internetzwerkadresse

Rückgabewert

Die Routine kann folgende Fehlercodes zurückgeben:

SUCCESS
Die Konversion konnte erfolgreich ausgeführt werden.

PARAMETER_ERROR
Die Eingansparameter konnten nicht konvertiert werden.

```
FUNCTION IPX_From_Addr ( Addr
   : Network_Address;

         VAR Network


   : String;


   : String;

        ) : BYTE;
```

Beschreibung :
Die Routine konvertiert die vollständige Netzwerkadresse in String´s.

Parameter :

IN
= Vollständige
Netzwerkadresse

OUT
= Netzwerkadresse

Node
= Knotenadresse

Socket
= Sockelnummer

Rueckgabewert

Die Routine kann folgende Fehlercodes zurückgeben:

SUCCESS
Die Konversion konnte erfolgreich ausgeführt werden.

PARAMETER_ERROR
Der Eingansparameter konnte nicht konvertiert werden.

6. Einschränkungen / Probleme

Bei der Benutzung der Bibliothek ist auf die folgenden Punkte besonders zu achten:

-
immer
einer
wenn
gemacht,

-
als

bremsen z.B. mittels delay - Funktion. Dadurch sinkt natürlich der Durchsatz.

Sockel
verschiedenen

-
nicht
möglich,
ohne dass es

Zeilstation
Applikation z.B. mit
werden.

-
Applikation sämtliche Kommunikationsockel explizit zu schliessen. Dies ist

Literaturverzeichnis

[1]
Part # 100-000569-001.

Division, Novell Part # 100-0005571-001.

[2]

[3]