Task-to-Task Kommunikation mit Ethernet

Dokumentations-Nr.: D930072H3-1085

Dieses Papier stellt die in der Betriebssystemversion MUNIX V.2/05 enthaltenen Möglichkeiten der Task-to-Task-Kommunikation mit Ethernet vor. Programmbeispiele sind angegeben.

Best.-Nr.: D930072H3-1085
DF: task task.text
Autoren-Kennzeichen: NS

Inhalt

* * * * *

1. Einleitung

Diese Beschreibung stellt die Schnittstellen zum Ethernet vor, mit denen eine *Task-to-Task-Kommunikation* aufgebaut werden kann.

Es gibt drei Varianten:

- MUNIX/NET - Schnittstelle

- Basic Block Port - Schnittstelle

- Ethernet Port - Schnittstelle

Zur Nutzung der einzelnen Schnittstellen müssen die entsprechenden Treiber im Betriebssystemkern generiert sein. Es ist für

- MUNIX/NET der MUNIX/NET-Treiber

- Basic Block Port der NEWCASTLE-Treiber

- Ethernet Port der NEWCASTLE-Treiber

im Systemkern der Version MUNIX V.2/05 nötig.

Im nächsten Kapitel werden die Gemeinsamkeiten der drei Schnittstellen beschrieben. In den folgenden Abschnitten werden die einzelnen Schnittstellen vorgestellt. Zuerst wird das Prinzip jeder Schnittstelle erläutert. Dann wird zu den einzelnen Protokollen das Paketformat angegeben und die Benutzung der jeweiligen Schnittstelle durch die Angabe von Beispielprogrammen erläutert.

2. Gemeinsame Merkmale der Schnittstellen

Alle drei Schnittstellen ermöglichen Prozessen, die auf verschiedenen Knoten im Netzwerk ablaufen, miteinander zu kommunizieren. Ein Prozeß kann Nachrichten von anderen, auf entfernten Rechnern laufenden Prozessen, empfangen und senden. Nachrichten werden in Form von Paketen über Ethernet gesendet.

Die *Task-to-Task-Kommunikation* mit Ethernet ist ein Datagramm Service. Es ist nicht sichergestellt, daß ein abgeschicktes Paket den adressierten Prozeß erreicht. Es können Pakete verlorengehen. Die einzelnen Pakete sind voneinander unabhängig. Zwischen den kommunizierenden Prozessen muß keine direkte Verbindung aufgebaut werden.

Die Adressierung der Prozesse erfolgt nicht direkt, sondern basiert auf dem Port-Konzept, das allerdings bei jeder Schnittstelle anders realisiert wurde. Prozesse senden Nachrichten an Ports und können von Ports Nachrichten empfangen. Um einen Port nutzen zu können, muß er eröffnet sein. Es wird immer nur ein empfangenes Paket am Port zwischengespeichert; für die restliche Ablaufkontrolle ist der Anwender zuständig. Kommt ein weiteres Paket am Port an, ohne daß das vorhergehende gelesen wurde, so geht es verloren.

Ports sind im Netz eindeutig durch Rechneradresse und lokale Portnummer adressierbar. Es gibt statisch bestimmte und dynamische Portnummern. Statisch bestimmt heißt, daß ein Prozeß einem Port eine vordefinierte Nummer zuordnet. Dadurch ist es möglich, daß Prozesse mit systemweit bekannten Portnummern existieren. Dynamisch heißt, daß ein Prozeß einen Port anfordert, dessen Nummer vom Betriebssystem zum Eröffnungszeitpunkt bestimmt wird.

Die *Task-to-Task-Kommunikation* erfolgt über Routinen der Standardbibliothek oder über Systemaufrufe. In jedem Fall muß ein Port vor der Benutzung erst eröffnet und entsprechend konfiguriert werden. Durch die Konfigurierung eines Ports wird entweder der Adressat einer Nachricht festgelegt oder der Empfang von Nachrichten gesteuert.

Zu den Daten, die der Anwender als Paket auf das Netz schickt, wird vom Betriebssystem noch Verwaltungsinformation beigefügt.

3. MUNIX/NET - Schnittstelle

3.1. Funktionsprinzip

Diese Schnittstelle steht innerhalb von MUNIX/NET zur Verfügung. MUNIX/NET verbindet einzelne MUNIX-Systeme (Knoten) über Ethernet zu einem virtuellen MUNIX-System.

Das Port-Konzept wurde über die MUNIX-Ports implementiert. Ein Prozeß muß nur für das Empfangen von Nachrichten einen Port besitzen, das Senden ist auch ohne Port möglich. Es kann immer nur ein Port einem bestimmten Prozeß zugeordnet sein. Eröffnet ein Prozeß einen weiteren Port, so geht ihm der zuvor geholte Port verloren. Ports sind innerhalb des Rechners (Knoten) durch ihre Portnummer, innnerhalb des Netzwerks durch Knotenadresse und Portnummer bestimmt.

Für das Transportieren der Nachrichten über die Ports stehen die Routinen:

open_uport
read_uport
write_uport
close_uport

in der Standardbibliothek zur Verfügung.

Mit **open_uport** kann ein Port eröffnet werden. Gleichzeitig kann der Senderkreis, von dem an diesem Port Nachrichten empfangen werden sollen, bestimmt werden. Ein Paket wird mit **read_uport** empfangen und mit **write_uport** gesendet. **Close_uport** schließt einen Port. Näheres über die Routinen entnehmen Sie bitte dem MUNIX Manual 1b (siehe: OPEN_UPORT(3U), READ_UPORT(3U), WRITE_UPORT(3U), CLOSE_UPORT(3U), UPORT(3U)) Die maximale, für den Anwender nutzbare Größe eines Pakets ist 1024 Bytes.

3.2. Paketformat

Ein mit *write_uport* gesendetes Paket wird im folgenden Format über Ethernet übertragen:

14 Bytes:	
6 Bytes	Zieladresse
6 Bytes	Absender
2 Bytes	Protokoll-Typ 0x0800
xx Bytes:	MUNIX/NET Header
0 - 1024 Bytes:	Frei für den Benutzer

Die 14 Bytes des Ethernet-Headers und des MUNIX/NET-Headers wird vom Betriebssystem den Benutzerdaten vorangestellt.

3.3. Programmbeispiele

3.3.1. Vom Port empfangen

```
# include <sys/types.h>
# include <sys/unison.h>
# include <sys/uport.h>

main()
{
 int i;
 char buf[512];
 ipnetaddr srcaddr; /* Platz fuer Absender */
 int srcptno; /* Platz fuer Senderport */

 /* Eroeffnet den Port mit der Nummer 111  */
 /* und akzeptiert Pakete von ueberall;   */
 /* durch die Angabe SETDYN anstelle von  */
 /* der Nummer 111 erhaelt man einen      */
 /* dynamischen Port                      */

 i = open_uport(111,ANYPORT,ANYADDR,0);
 if (i == -1) {
          printf("Port kann nicht eroeffnet werden\n");
          exit(-1);
         }

 /* Liest ein Paket von dem Port    */
 /* Nach dem Lesen steht die Sender-*/
 /* adresse                         */
 /* in srcaddr und die              */
 /* zugehoerige Portnummer          */
 /*  in srcptno                     */
 /* das Paket befindet sich in buf  */
 /* i enthaelt die Anzahl der       */
 /* uebertragenen Bytes             */
 /* nb enthaelt die maximale Anzahl */
 /* Bytes die gelesen werden können*/

 i = read_uport(buf,sizeof(buf),&srcaddr,&srcptno);
 if (i == -1) {
          printf("Fehler beim Lesen des Ports\n");
          exit(-1);
         }

 /* Schliesst den Port */

 close_uport();
}
```

3.3.2. An Port senden

```
# include <errno.h>
# include <sys/types.h>
# include <sys/unison.h>
# include <sys/uport.h>

char buf[] = "Hallo";

main()
{

 int i;
 ipnetaddr destaddr;
 int destptno,srcptno;


 /* Sendet ein Paket an den Port 111 auf dem Rechner */
 /* mit der IP - Adresse 0xc00c8001                  */
 /* das Paket befindet sich in buf                   */

 i = write_uport(buf,sizeof(buf),(ipnetaddr)0xc00c8001,111,0);
 if (i == -1) {
           printf("Port kann nicht beschrieben werden\n");
           exit(-1);
          }

}
```

4. Basic Block Port (BBP) - Schnittstelle

4.1. Funktionsprinzip

Die BBP - Schnittstelle wurde ursprünglich für den Cambridge Ring entwickelt und von PCS an Ethernet angepaßt. Mit BBP ist es möglich, daß verschiedene Prozesse auf unterschiedlichen Rechnern (oder auf dem gleichen) sich Nachrichten schicken. Ein Port ist gekennzeichnet durch die Struktur *portinfo* aus */usr/include/sys/port.h*:

```
struct portinfo {
short           pi_type;   /* type of port, unused for Ethernet */
unsigned short  pi_inport; /* bb_port number by which this port
                                    * is addressed by other stations
                                    */
short           pi_station;/* destination ring station */
unsigned short  pi_outport;/* destination bb_port number */
unsigned short  pi_accept; /* acceptable source station number */
etheradr        pi_ethadr; /* destination ethernet address */
};
```

Indem eine der Dateien */dev/bbp0*, */dev/bbp1* usw. eröffnet wird, kann ein Port geholt werden. Die maximale Anzahl der Ports im jeweiligen System steht in der Konstanten MAXPORTS in der Datei */usr/include/sys/param.h*. Die Anzahl der Ports, die ein Prozeß besitzen kann, ist auf die Anzahl der Dateien, die ein Prozeß eröffnen kann, begrenzt. Nach dem Eröffnen muß der Port durch ein *ioctl*-Kommando konfiguriert werden. Damit kann der Senderkreis, von dem emfangen werden soll, eingeschränkt werden. Nach dem Öffnen und dem Konfigurieren kann mit dem Port gearbeitet werden.

Der Datentransfer geschieht mit den *read* -und *write* -Systemaufrufen. Die maximale, für den Anwender nutzbare Paketgröße ist 1490 Bytes. Näheres siehe MUNIX Manual (BBP(4)).

4.2. Paketformat

Ein mit dem *write* -Systemaufruf gesendetes Paket hat folgendes Format:

14 Bytes:	
6 Bytes	Zieladdresse
6 Bytes	Absender
2 Bytes	Protokoll-Typ 0x4242
10 Bytes:	
2 Bytes	Sendestation
2 Bytes	Zielstation
2 Bytes	Absender-Port
2 Bytes	Ziel-Port
2 Bytes Größe der Daten	
0 - 1490 Bytes:	
	Frei für den Benutzer

Die 24 Bytes Header versorgt das Betriebssystem.

4.3. Programmbeispiele

4.3.1. Vom Port empfangen

```
# include <sys/port.h>
# include <fcntl.h>

/* Auf dem Rechner mit der Stationsnummer 1 liest das */
/* Programm Pakete auf Port 18, die von Port 19 des   */
/* gleichen Knotens gesendet werden                   */

struct portinfo aport = {0,18,1,19,1,{0x0800,0x2700,0x8001}};

main()
{
 char buf[512];
 int fd,i;


 /* Eroeffnen des Ports */

 fd = open("/dev/bbp0",O_RDWR);
 if (fd == -1) {
            printf("Fehler beim Eroeffnen\n");
            exit(-1);
           }

 /* Setzen des Ports wie oben angegeben */

 i = ioctl(fd,BBPSET,&aport);
 if (i == -1) {
           printf("Fehler beim Setzen\n");
           exit(-1);
          }

 /* Lesen eines Pakets */
 /* das Paket befindet */
 /* sich danach in buf */

 i = read(fd,&buf,sizeof(buf));
 if (i == -1) {
           printf("Fehler beim Lesen\n");
           exit(-1);
          }

 /* Schliessen des Ports */

 close(fd);
}
```

4.3.2. An Port senden

```
# include <sys/port.h>
# include <fcntl.h>

/* Auf dem Rechner mit der Stationsnummer 1 schreibt das */
/* Programm Pakete von Port 19 nach Port 18 des gleichen Knotens */

struct portinfo aport = {0,19,1,18,1,{0x0800,0x2700,0x8001}};

char buf[] = " Nachricht 1 ";

main()
{
 int fd,i;

  /* Eroeffnen des Ports */

  fd = open("/dev/bbp1",O_RDWR);
  if (fd == -1) {
              printf("Fehler beim Eroeffnen\n");
              exit(-1);
             }

  /* Setzen des Ports wie oben angegeben */

  i = ioctl(fd,BBPSET,&aport);
  if (i == -1) {
             printf("Fehler beim Setzen\n");
             exit(-1);
            }


  /* Senden eines Pakets */
  /* das Paket befindet  */
  /* sich in buf         */

  i = write(fd,&buf,sizeof(buf));
  if (i == -1) {
             printf("Fehler beim Schreiben des Ports\n");
             exit(-1);
            }

  /* Schliessen des Ports */

  close(fd);
}
```

5. Ethernet Port (ETHPT) - Schnittstelle

5.1. Funktionsprinzip

ETHPT ist eine einfache Schnittstelle zum Ethernet. Sie erlaubt die Anwendung von verschiedenen Protokollen und die Vernetzung von inhomogenen Rechnern. Die Kommunikation kann auch zwischen Prozessen stattfinden, die sich auf dem gleichen Rechner befinden.

Es können nur *write-only* oder *read-only* Ports eröffnet werden. Ein Port ist gekennzeichnet durch die Struktur *ethptinfo* aus */usr/include/sys/ethpt.h*:

```
struct ethptinfo {
unsigned short  ei_proto;   /* Ethernet protocol type */
char            ei_swapflag;/* swap bytes in Ethernet packets */
char            ei_rfuflag; /* reserved for future use */
etheradr        ei_remadr;  /* address of remote Ethernet station
                  * write-only port: destination address
                  * read-only  port: source address of
                  * last packet received
                  */
etheradr        ei_locadr;  /* address of this Ethernet station
                  * write-only port: local address
                  * read-only port: destination address
                  *       of
                  *        last packet received
                  */
}
```

In der Variablen *ei_proto* wird der Ethernet Protokolltyp festgelegt. Die Übermittlung von Nachrichten erfolgt nur zwischen Ports, die das gleiche Protokoll haben. Die Protokollarten 0x0800 und 0x0806 sind für MUNIX/NET und 0 bzw. 0x4242 für BBP reserviert. Die Identifikation auf dem Netz bilden die Ethernet-Adresse und der Protokolltyp. Es besteht keine Möglichkeit festzulegen, von welchen Ports man empfangen will. Ein Port kann aufgemacht werden, indem eine der Dateien */dev/ethpt/0*, */dev/ethpt/1* usw. eröffnet wird. Die maximale Anzahl der Ports ist auf **20** begrenzt. Nach dem Eröffnen der Datei, ist der Port mit dem Systemaufruf *ioctl* zu setzen. Hierbei darf der angegebene Protokolltyp auf dem Rechner noch bei keinem andern Port, mit dem gleichen Öffnungsmodus (*read-* oder *write-only)*, gesetzt sein.

Die Variable *ei_swapflag* berücksichtigt die Datenrepräsentation der einzelnen Rechner. Die Werte SWAP bzw. NOSWAP geben an, ob die Bytes der Benutzerdaten vertauscht werden sollen.

Der Datentransfer geschieht mit den ***read*** -und ***write*** -Systemcalls. Die maximale, für den Anwender nutzbare Paketgröße beträgt 1500 Bytes, die minimale 46. Die Ethpt - Schnittstelle ist im MUNIX Manual unter ethpt(4) näher beschrieben.

5.2. Paketformat

Ein mit dem ***write*** -Systemaufruf gesendetes Paket hat folgendes Format:

14 Bytes:	
6 Bytes	Zieladresse
6 Bytes	Absender
2 Bytes	Protokolltyp
46 - 1500 Bytes:	
	Frei für den Anwender

Dem Paket wird vor dem Senden vom Betriebssystem die 14 Bytes Header-Information mitgegeben.

5.3. Programmbeispiele

5.3.1. Vom Port empfangen

```
# include <fcntl.h>
# include <errno.h>
# include <sys/ethpt.h>

/* Empfangsport (read-only) mit Protokoll 0x00ab */

struct ethptinfo aport = {0x00ab,NOSWAP,0,{0x0,0x0,0x0},
                  {0x0,0x0,0x0}};

main()
{
 int fd,i;
 char buf[46];


  /* Eroeffnen des Ports */

  fd = open("/dev/ethpt/9",O_RDONLY);
  if (fd == -1) {
             printf("Fehler beim Eroeffnen\n");
             exit(-1);
            }

  /* Setzen des Ports */

  i = ioctl(fd,ETHPTSET,&aport);
  if (i == -1) {
            printf("Fehler beim Setzen\n");
            exit(-1);
           }

  /* Lesen eines Pakets  */
  /* das Paket befindet  */
  /* sich danach in buf  */

  i = read(fd,buf,46);
  if (i == -1) {
            printf("Fehler beim Lesen des Ports\n");
            exit(-1);
           }

  /* Schliessen des Ports */

  close(fd);
}
```

5.3.2. An Port senden

```
# include <sys/ethpt.h>
# include <errno.h>
# include <fcntl.h>

/* Sendeport (write-only) mit Protokoll 0x00ab */

struct ethptinfo aport = {0x00ab,NOSWAP,0,{0x0800,0x2700,0x8001},
                  {0x0,0x0,0x0}};

char buf[] = "Hallo, hier ist das Paket mit der Nummer 0001";

main()
{
 int fd,i;

  /* Eroeffnen des Ports          */
  /* modus = O_RDONLY oder O_WRONLY */

  fd = open("/dev/ethpt/7",O_WRONLY);
  if (fd == -1) {
             printf("Fehler beim Eroeffnen\n");
             exit(-1);
            }

  /* Setzen des Ports */

  i = ioctl(fd,ETHPTSET,&aport);
  if (i == -1) {
              printf("Fehler beim Setzen\n");
              exit(-1);
             }

  /* Senden eines Pakets */

  i = write(fd,buf,46);
  if (i == -1) {
            printf("Fehler beim Schreiben des Ports\n");
            exit(-1);
           }

  /* Schliessen des Ports */

  close(fd);
}
```

5.3.3. Eröffnen eines freien Ports

Anstatt einen Port – wie in den vorangegangenen Beispielen – über einen festen Dateinamen zu eröffnen, kann mit der Routine *dynport* dynamisch ein freier Port gesucht werden.

```
int dynport(modus)
int modus;
{
 int fd,i;

   /* Suchen eines freien Ports.*/
   /* Als Ergebnis wird der    */
   /* Dateideskriptor eines    */
   /* freien Ports oder -1     */
   /* (falls alle Ports belegt */
   /* sind) zurueckgegeben.    */

  static char ethpt[] = "/dev/ethpt/00";
     i = 0;
     ethpt[12] ='\0'; /* Initialisieren des Stringendes */
     do
     {
        if (i < 10)
        {
           ethpt[11]= i + '0';
        }
        else
        {
          ethpt[11]= (i/10)+'0';
          ethpt[12]= (i%10)+'0';
        }
        i++;
     }
     while ((fd = OPEN(ethpt, modus)) < 0 && errno == EACCES);

   return(fd);
}
```

NAME

bbp – Basic Block Port Interface

SYNOPSIS

#include <sys/port.h>

DESCRIPTION

The Basic Block Port Interface is a simple network interface, originally developed for the Cambridge Ring, and now adapted to Ethernet. The bbp provides a set of so-called ports, through which processes on different machines (or on the same) can talk to each other. The port is characterized by the structure portinfo in *<sys/port.h>* :

```
struct portinfo {
        short           pi_type;        /* port type,unused for Ethernet */
        unsigned int    pi_inport;      /* bb_port number by which this port
                                         * is addressed by other stations
                                         */
        short           pi_station;     /* destination ring station */
        unsigned int    pi_outport;     /* destination bb_port number */
        unsigned int    pi_accept;      /* acceptable source station number */
        etheradr        pi_ethadr;      /* destination ethernet address */
};
```

The field *pi_type* is unused for Ethernet. For the Cambridge ring this field specifies if the data transfers are protected by parity checks or not. The field *pi_inport* specifies the input port number, by which this port is addressed by other stations. Their output port number, *pi_outport*, must be equal to pi_inport, if they want to talk to this port. The field *pi_station* is a two-byte station number.

A station number is a unique identification of each machine attached to the net. This station number is more manageable than the six byte Ethernet address contained in the field *pi_ethadr*. The ethernet address is mainly used for the hardware address recognition, whereas the station number is used by upper level software. Both the station number and the ethernet address are fixed at system generation time. They are, like the ascii system name, a unique name of the system. Their values can be found in the file */usr/sys/name.c* and can be gotten by the system call *uname*(2).

The fields *pi_station*, and *pi_ethadr* together specify the destination machine; the field *pi_outport* is the port number on the destination machine, to which this port wants to talk. The field *pi_accept* specifies the machines from which this port is willing to receive. The values NOONE and ANYONE mean: accept packets from noone or anyone. Any other number means: accept packets only from the station with this number.

It is not necessary for a process to specify his own station number and ethernet address, as the system knows them already and they cannot change.

The portinfo structure is set by an ioctl system call with command BBPSET, and read with command BBPGET.

EXAMPLE

Machine alpha has the station number 3 and the ethernet address 333333333333. Machine beta has the station number 5 and the ethernet address 555555555555. A process on alpha wishes to receive on port

number 372. Another process on beta receives on port number 373; The process on alpha specifies

```
struct portinfo alphaport = { 0, 372, 5, 373, 5, {0x5555,0x5555,0x5555}};
        ioctl(fd,BBPSET,&alphaport);
```

whereas the process on beta specifies

```
struct portinfo betaport = { 0, 373, 3, 372, 3, {0x3333,0x3333,0x3333}};
        ioctl(fd,BBPSET,&betaport);
```

Both processes can now talk to each other by normal read and write system calls.

A port can be obtained by successively opening the files /dev/bbp0, /dev/bbp1 etc. If the open returns with errno ENXIO, the port does not exist, if errno is EACCES, the port is already opened. After the open the port must be configured with the command BBPSET. If the ioctl returns with errno EACCES, a port with the same *pi_inport* is already open. Just to get an unused port number, the value DYNAMIC can be given for *pi_inport*. The actual port number can then be gotten with the ioctl-command BBPGET.

EXAMPLE

```
struct portinfo aport = { 0, DYNAMIC, 5, 123, ANYONE, {0x5555,0x5555,0x5555}};
        ioctl(fd,BBPSET,&aport);
        ioctl(fd,BBPGET,&aport);  /* pi_inport contains a free port number */
```

Data is transferred with the normal read and write system calls. However, there is a limitation on the number of bytes that can be transferred with one write. On the Ethernet, the number 1024 is safe. The data of each write system call is sent as a packet over the net. The count of the read system call must be larger or equal than the size of the packet, otherwise the read returns with error EIO. read returns the size of the received packet. If the count for read or write is illegal, error EINVAL is returned.

At any time after BBPSET, the ioctl command BBPENQ will return the following structure, defined in *<sys/port.h>*:

```
struct portenq {
        short           pn_sender;      /* station number
                                           of sender of received block */
        short           pn_sendport;    /* port number
                                           of sender of received block */
        char            pn_xrslt;       /* last block transmission result */
        char            pn_blkavail;    /* a block is available to be read */
        etheradr        pn_sendadr;     /* ethernet address of sender */
};
```

The fields *pn_sender*, *pn_sendport*, and *pn_sendadr* specify the station number, port number, and ethernet address of the sender of a received packet. The field *pn_xrslt* contains the result of the last write. This is normally equal to BB_ACCEPTED on the ethernet, and equal to BB_ERROR only if excessive jams occurred on the net. The field *pn_blkavail* is unequal 0, if a packet has been received, but not yet read.

WARNING

The bbp contains no flow control. Incoming packets are simply discarded if they are not read fast enough. Protocols are entirely the responsibility of upper levels.

SEE ALSO

sbp(4)

FILES

/dev/bbp*