FUJITSU Software BS2000 POSIX-SOCKETS

Version 10.0A
Oktober 2016

Readme-Datei

1 Einleitung **3**
2 Produktbestandteile und Installation **4**
2.1 Installation im BS2000 4
2.2 Installation im POSIX-Dateisystem 4
3 Funktionen **6**
3.1 Zusätzliche Funktionen 6
3.2 Multihoming 8
3.3 Dateideskriptor-Übertragung 8
3.4 Funktionale Abhängigkeiten zu anderen Produkten 9
3.5 Entfallene Funktionen 9
4 Einschränkungen **10**
4.1 Verwendung von Sockets der Domäne AF_UNIX 10
4.2 Koexistenz von Sockets der Domänen AF_INET und AF_INET6 10

1 Einleitung

Die vorliegende Readme-Datei enthält die bekannten Änderungen zum Handbuch

POSIX
SOCKETS/XTI für POSIX
Ausgabe März 2005

2 Produktbestandteile und Installation

2.1 Installation im BS2000

Folgende Dateien können mit IMON im BS2000 installiert sein:

	MODE	SHARE	ACCESS	MIGRATE
$<TSOS>.SYSLIB.POSIX-SOCKETS.100	COM	Y	R	I
$<TSOS>.SKMLIB.POSIX-SOCKETS.100	X86	Y	R	I
$<TSOS>.SYSRME.POSIX-SOCKETS.100.D	COM	Y	R	I
$<TSOS>.SYSRME.POSIX-SOCKETS.100.E	COM	Y	R	I
$<TSOS>.SYSSII.POSIX-SOCKETS.100	COM	Y	R	A
$<TSOS>.SYSLNK.POSIX-SOCKETS.100.PTH	COM	Y	R	I
$<TSOS>.SKULNK.POSIX-SOCKETS.100.PTH	X86	Y	R	I

- Die SYSLIB.POSIX-SOCKETS.100 enthält die Header-Dateien und die Module im /390-Format zum Compilieren und Binden im BS2000 sowie zum Installieren der Sockets-Bibliothek /usr/lib/libsocket.a in das POSIX-Dateisystem.

- Die SKMLIB.POSIX-SOCKETS.100 enthält die Header-Dateien und die Module im X86-Format zum Compilieren und Binden im BS2000 sowie zum Installieren der Sockets-Bibliothek /usr/lib/libsocket.a bzw. /usr/lib/X86/libsocket.a in das POSIX-Dateisystem.

- Die S*xx*LNK.POSIX-SOCKETS.100.PTH enthalten die PThread-Varianten der Module (als LLM SOCKGM).
  Sie werden von bestimmten POSIX-SOCKETS-Anwendungen zur Laufzeit dynamisch nachgeladen. Diese Varianten werden nicht in das POSIX-Dateisystem installiert.

2.2 Installation im POSIX-Dateisystem

Die Installation erfolgt mit dem POSIX-Installationsprogramm (/START-POSIX-INSTALLATION, Funktion Paketinstallation). Siehe hierzu das Handbuch „POSIX Grundlagen“.

Für die Installation im POSIX-Dateisystem werden ca. 7 MB temporärer Speicher unter /tmp und ca. 7 MB permanenter Speicher pro installierter Variante unter /usr/lib bzw. /usr/lib/X86 benötigt.

Die Installation kann einige Zeit in Anspruch nehmen.

Falls die Installation nicht korrekt durchgeführt werden kann, wird die Textdatei /tmp/install.libsocket.err erstellt, die Hinweise auf die Fehlerursache enthält.

Installierte Bibliotheken:

Ist nur die SYSLIB.POSIX-SOCKETS.100 installiert, so wird nur die Bibliothek `/usr/lib/libsocket.a` mit der /390 Code Ausprägung im POSIX-Dateisystem erzeugt.
Sind verschiedene Varianten der BS2000-Bibliotheken installiert, so werden auch verschiedene Varianten der Sockets-Bibliotheken generiert:

`/usr/lib/libsocket.a`	mit /390 Code
`/usr/lib/X86/libsocket.a`	mit X86 Code

In diesem Fall erfolgt das Einbinden der X86 Variante dann durch Angabe der Compiler-Option
`-L /usr/lib/X86`

Installierte Dateien:

```
/usr/lib/libsocket.a
/usr/lib/libxnet.a
/usr/include/netdb.h
/usr/include/arpa/inet.h
/usr/include/arpa.inet.h
/usr/include/net/if.h
/usr/include/net.if.h
/usr/include/netinet/in.h
/usr/include/netinet.in.h
/usr/include/sys/byteorder.h
/usr/include/sys.byteorder.h
/usr/include/sys/netconfig.h
/usr/include/sys.netconfig.h
/usr/include/sys/socket.h
/usr/include/sys.socket.h
/usr/include/sys/sockio.h
/usr/include/sys.sockio.h
/usr/include/sys/un.h
/usr/include/sys.un.h
/usr/include/sys/xti_inet.h
/usr/include/sys.xti_inet.h
/usr/include/xti.h
```

Aktivierung von Multicast-Funktionen in `netinet/in.h` mit:
`#define IP_MULTICAST`

3 Funktionen

3.1 Zusätzliche Funktionen

Zusätzlich zu den im Handbuch beschriebenen Funktionen werden auch die folgenden unterstützt:

```
bindresvport( int sd , struct sockaddr_in *sin )

rcmd( char **ahost , u_short rport , char *locuser ,
          char *remuser , char *cmd , int *fd2p )

rcmd_af( char **ahost , u_short rport , char *locuser ,
          char *remuser , char *cmd , int *fd2p , int af )

rexec( char **ahost , int rport , char *name , char *pass ,
          char *cmd , int *fd2p )

s_fcntl( int des , int cmd , int arg ) /* same as fcntl() */
s_ioctl( int des , int cmd , int arg ) /* same as ioctl() */

void s_set_uppernameatgethostbyname( int mode ) /* bs2 special */

int t_rcvv(int, struct t_iovec *, unsigned int, int *)
int t_rcvreldata(int, struct t_discon *)
int t_rcvvudata(int, struct t_unitdata *, struct t_iovec *,
         unsigned int, int *)

int t_sndv(int, const struct t_iovec *, unsigned int, int)
int t_sndreldata(int, struct t_discon *)
int t_sndvudata(int, struct t_unitdata *, struct t_iovec *,
         unsigned int)

int t_sysconf(int)
```

Die Funktion `getsockopt()` unterstützt folgende im Manual nicht beschriebene Option:

```
getsockopt(fd, SOL_SOCKET, SO_BS2ERROR, &optval, &optlen);
```

Die Variable '`optval`' muss vom Typ '`struct so_bs2error`' sein. Der Aufruf liefert den internen Fehler-Code (BCAM-Returncode) und die entsprechende 'errno' des letzten für diesen Socket aufgetretenen Fehlers. Anders als bei '`getsockopt(..., SO_ERROR, ...)`' wird dieser Fehler-Code nach der Abfrage nicht gelöscht.
Sowohl '`SO_BS2ERROR`' als auch '`struct so_bs2error`' sind in `sys/socket.h` definiert.

Die Funktionen `getsockopt()` und `setsockopt()` unterstützen folgende im Manual nicht beschriebene Optionen:

```
getsockopt(fd, IPPROTO_TCP, TCP_KEEPIDLE,  &optval, &optlen);
getsockopt(fd, IPPROTO_TCP, TCP_KEEPINTVL, &optval, &optlen);
getsockopt(fd, IPPROTO_TCP, TCP_KEEPCNT,   &optval, &optlen);

setsockopt(fd, IPPROTO_TCP, TCP_KEEPIDLE,  &optval, optlen);
setsockopt(fd, IPPROTO_TCP, TCP_KEEPINTVL, &optval, optlen);
setsockopt(fd, IPPROTO_TCP, TCP_KEEPCNT,   &optval, optlen);
```

Der Parameter '`optval`' muss in allen Fällen vom Typ '`int *`' sein. Die Optionen beeinflussen das Verhalten, wenn das Senden von Kontrollnachrichten mit der Option SOL_SOCKET/SO_KEEPALIVE aktiviert ist.

TCP_KEEPIDLE
Diese Option legt die Zeit in Sekunden fest, die die Verbindung inaktiv sein muss, bevor die erste Kontrollnachricht gesendet wird. Gültiger Wertebereich: 120 bis 32767 Sekunden.

TCP_KEEPINTVL
Diese Option legt das Zeitintervall in Sekunden fest, nach welchem das Senden der Kontrollnachrichten wiederholt wird. Gültiger Wertebereich: 120 bis 32767 Sekunden.
Das Transportsystem BCAM unterstützt keine unterschiedlichen Werte für die Optionen TCP_KEEPIDLE und TCP_KEEPINTVL, deshalb wird dort der kleinere Wert der beiden Optionen benutzt, falls beide ungleich 0 sind.

TCP_KEEPCNT
Diese soll festlegen, nach wie vielen Kontrollnachrichten die Verbindung abgebaut werden soll. Gültiger Wertebereich: 1 bis 127.
Diese Option wird in POSIX-SOCKETS zwar formal unterstützt, aber sie hat keine Auswirkung auf das Verhalten des Transportsystems BCAM.

Die Funktion *ioctl()* unterstützt folgende im Manual nicht beschriebene Option (ab BCAM V23):

Request	***arg**	**Funktion**
SIOCGIFHWADDR	struct ifreq	Hardware-Adresse (MAC) des Interface auslesen

SIOCGIFHWADDR
Für das mit dem Element *ifr_name* spezifizierte Interface wird die Hardware-Adresse (MAC) im Element *ifr_hwaddr* zurückgeliefert.

Die bei der Funktion *ioctl()* für den Request *SIOCGLIFNETMASK* beschriebene Einschränkung für IPv6 trifft nicht mehr zu (ab BCAM V23):

SIOCGLIFNETMASK
Für das mit dem Element *lifr_name* spezifizierte Interface wird die IPv4-Subnetzmaske bzw. der IPv6-Präfix im Element *lifr_addr* zurückgeliefert.

Änderungen der Netzwerk-Interface-Konfiguration

Werden im laufenden Betrieb Änderungen an der Konfiguration der Netzwerk-Interfaces des Transportsystem BCAM vorgenommen, d.h. Hinzufügen oder Entfernen eines Interfaces oder Modifikation von Parametern eines Interfaces, ist kein Neustart des POSIX-Subsystems mehr notwendig, damit diese Änderungen für POSIX-Anwendungen wirksam werden.

Verbesserung der Close-Vorgänge von verbundenen Sockets

Die bisherigen Einschränkungen beim `'close()'` für verbundene Sockets wurden aufgehoben. Es wird nun ein „echter Graceful-Disconnect“ unterstützt und die Option SO_LINGER mit Timeout > 0 führt nicht mehr zu Verzögerungen beim `'close()'`.

3.2 Multihoming

Auf Rechnern mit mehreren IP-Interfaces (IP-Adressen) können POSIX-SOCKETS-Anwendungen gleichzeitig für jedes Interface einen eigenen Server-Socket auf denselben Port binden. Damit bearbeitet jeder Socket nur Anforderungen, die über das Interface eintreffen, für das er auf den Port gebunden ist.

Stattdessen (aber nicht gleichzeitig) kann ein Server-Socket auch wie bisher Anforderungen für einen Port unabhängig vom Interface bearbeiten (`INADDR_ANY`).

Einschränkung:

- Es können nicht gleichzeitig auf einen Port zwei Sockets gebunden werden, die Anforderungen unabhängig vom Interface nur für IPv4 (`INADDR_ANY`) und nur für IPv6 (`IN6ADDR_ANY`) bearbeiten.

3.3 Dateideskriptor-Übertragung

Die Funktionen `sendmsg()`/`recvmsg()` können benutzt werden, um Dateideskriptoren entweder mit Hilfe des Arrays `msg_accrights` oder mit Hilfe einer Control-Message vom Typ `SOL_SOCKET/SCM_RIGHTS` zu übertragen bzw. zu empfangen.

Nur Socket-Dateideskriptoren können übertragen werden.

Weitere Details sind der Beschreibung in der Header-Datei "`sys/socket.h`" zu entnehmen.

3.4 Funktionale Abhängigkeiten zu anderen Produkten

Die Funktionen `gethostbyname()` und `gethostbyaddr()` unterstützen DNS (Domain Name Service), wenn der DNS-Resolver aus dem Produkt InternetServices gestartet ist oder wenn das Subsystem SOC6 aus dem Produkt OpenNetServer gestartet ist.

Die Funktionen `getipnodebyname()` und `getipnodebyaddr()` unterstützen DNS für IPv4- und IPv6-Adressen, wenn das Subsystem SOC6 gestartet ist.

3.5 Entfallene Funktionen

Die Funktion `gethostname()` ist nicht mehr in POSIX-SOCKETS sondern nur in CRTE und CRTE-BASYS enthalten.

4 Einschränkungen

4.1 Verwendung von Sockets der Domäne AF_UNIX

Systemweit können maximal 500 dieser Sockets benutzt werden.

Die Einstellungen und Grenzwerte des Transportsystems BCAM gelten auch für diese Sockets.

4.2 Koexistenz von Sockets der Domänen AF_INET und AF_INET6

Allgemein können an einem Port Sockets der Domänen `AF_INET` und `AF_INET6` koexistieren.

Das gilt nicht, wenn ein Socket auf die Adresse `INADDR_ANY (0.0.0.0)` und ein anderer auf die Adresse `IN6ADDR_ANY (::0)` gebunden werden soll. In diesem Fall würde der zweite bind() mit `EADDRINUSE` abgewiesen werden.

Umgehung:

Die Anwendung kann einen Socket auf die Adresse `IN6ADDR_ANY (::0)` binden, welcher Anforderungen aus beiden Domänen verarbeitet.
Anforderungen aus der Domäne `AF_INET` werden diesem Socket dann mit IPv4-mapped-Adressen (`::ffff:a.b.c.d`) übergeben.