NORTi Version 4 ユーザーズガイド

補足説明書

2008 年 12 月版

MiSPO
株式会社ミスポ

はじめに

本書には、NORTi Version 4 ユーザーズガイドに記載されていない補足説明が記載されています。NORTi をご使用になる前に、および、お問い合せになる前に、本書をよくお読みいただけますようお願いします。

NORTi Version 4 に共通の事項については、次のドキュメントをご覧ください。

NORTi Ver.4 ユーザーズガイド カーネル編
NORTi Ver.4 ユーザーズガイド TCP/IP 編（注）

（注）第 5 版までは、「TCP/IP 編」でなく「ネットワーク編」という名称です。

2008 年 12 月版で改訂された内容

ページ	内容
11-12	TFN_NET_WRI_M_PKT、TFN_GET_NIF_NUM、TFN_NET_EXT を追加
13	lan_write_m_pkt、lan_ext_dev 関連を追加
14	lan_write_m_pkt1 を追加
15	lan_ext_dev、lan_ext_dev1 を追加
18-22	SET_TCP_DACK_TMO、SET_TCP_DUP_ACK を追加
19-20	解説を修正
26	tcp_set_opt、udp_set_opt の解説修正
27	解説を修正
28	「1.11.1 OS 資源」の解説修正
30	「1.13 ICMP パケットの送信」の解説修正
32-33	ファイル構成を修正
44-50	「2.11 SNTP クライアント」を追加
55-57	SET_TCP_DACK_TMO、SET_TCP_DUP_ACK を追加
58	arp_add_entry の解説修正
59	arp_del_entry の解説修正
60-65	「3.2 TCP 通信端点毎に変更可能になった設定値」を追加
66-67	「3.3 ARP REPLY の IP アドレスの重複検知」を追加

2005 年 10 月版で改訂された内容

ページ	内容
31	2.4 DHCP クライアントにリトルエンディアン使用時のマクロ定義を記述
34	2.5 DNS リゾルバにリトルエンディアン使用時のマクロ定義を記述

2005 年 5 月版で改訂された内容

ページ	内容
42	3.1.1 プロトコルスタックの終了をネットワーク I/F の終了に変更
42	3.1.1 の解説を追記

2005 年 2 月版で改訂された内容

ページ	内容
6	"1.1.概要"、"1.2 インストールを変更"
16	"1.9.3 ネットワーク I/F のオプション設定"を追加
17	"1.9.4 ネットワーク I/F のオプション設定(ネットワーク I/F 名指定)"を追加
18	"1.9.5 ネットワーク I/F オプション情報の取得"を追加
19	"1.9.6 ネットワーク I/F オプション情報の取得(ネットワーク I/F 名指定)"を追加
20	"1.9.7 ネットワーク I/F に設定されているアドレスの変更"を追加

21	"1.9.8 ネットワーク I/F に設定されているアドレスの変更(ネットワーク I/F 名指定)"を追加
22	"1.9.9 ネットワーク I/F 名からネットワーク I/F 制御ブロックを取得する"を追加
22	"1.9.10 チャネル番号からネットワーク I/F 制御ブロックを取得する"を追加
22	"1.9.11 デフォルトのネットワーク I/F 制御ブロックを取得する"を追加
22	"1.9.12 PPP のネットワーク I/F 制御ブロックを取得する"を追加
22	"1.9.13 IP アドレスからネットワーク I/F 制御ブロックを取得する"を追加
23	"TCP 通信端点のチャネル番号を指定する"の"解説"を修正
23	"UDP 通信端点のチャネル番号を指定する"の"解説"を修正
24	"1.10.1 TCP 編"の説明を変更
24	"1.10.2 UDP 編"の説明を変更
25	"1.10.3 IPV4_ADDRANY"の説明を変更
25	"1.11.3 DHCP"を追加
27	"1.12.4 Ethernet を 3 チャネル以上使用する"を追加
27	"1.13 ICMP パケットの送信"を追加
28	"1.15 ループバックインターフェース"を追加
31	"2.4.3 DHCP サーバーより構成情報を得る(ネットワークインターフェース名指定)"を追加
31	"2.4.4 DHCP サーバーより構成情報を再取得する(ネットワークインターフェース名指定)"を追加
33	"2.5.2 ドメイン名から IP アドレスを取得する(ネットワークインターフェース名指定)"を追加
41～	"3 章 新規に追加された機能"を追加
41	"3.1.1 プロトコルスタックの終了"を追加
41	"3.1.2 TCP 通信端点のオプション設定"を追加
42	"3.1.3 TCP 通信端点に設定されているオプション情報の取得"を追加
43	"3.1.4 UDP 通信端点のオプション設定"を追加
44	"3.1.5 UDP 通信端点に設定されているオプション情報の取得"を追加
45	"3.1.6 デフォルトネットワーク I/F のオプション設定"を追加
46	"3.1.7 デフォルトネットワーク I/F に設定されているオプション情報の取得"を追加
47	"3.1.8 デフォルトネットワーク I/F に設定されているアドレスの変更"を追加
49	"3.1.9 ARP 要求の送信"を追加
49	"3.1.10 ARP 要求の送信(ネットワークインターフェース名指定)"を追加
49	"3.1.11 ARP テーブルに情報を追加する"を追加
50	"3.1.12 ARP テーブルに情報を追加する(ネットワークインターフェース名指定)"を追加
50	"3.1.13 ARP テーブルから情報を削除する"を追加
50	"3.1.14 ARP テーブルから情報を削除する(ネットワークインターフェース名指定)"を追加

目次

第１章 マルチチャネル対応プロトコルスタック

1.1 概要

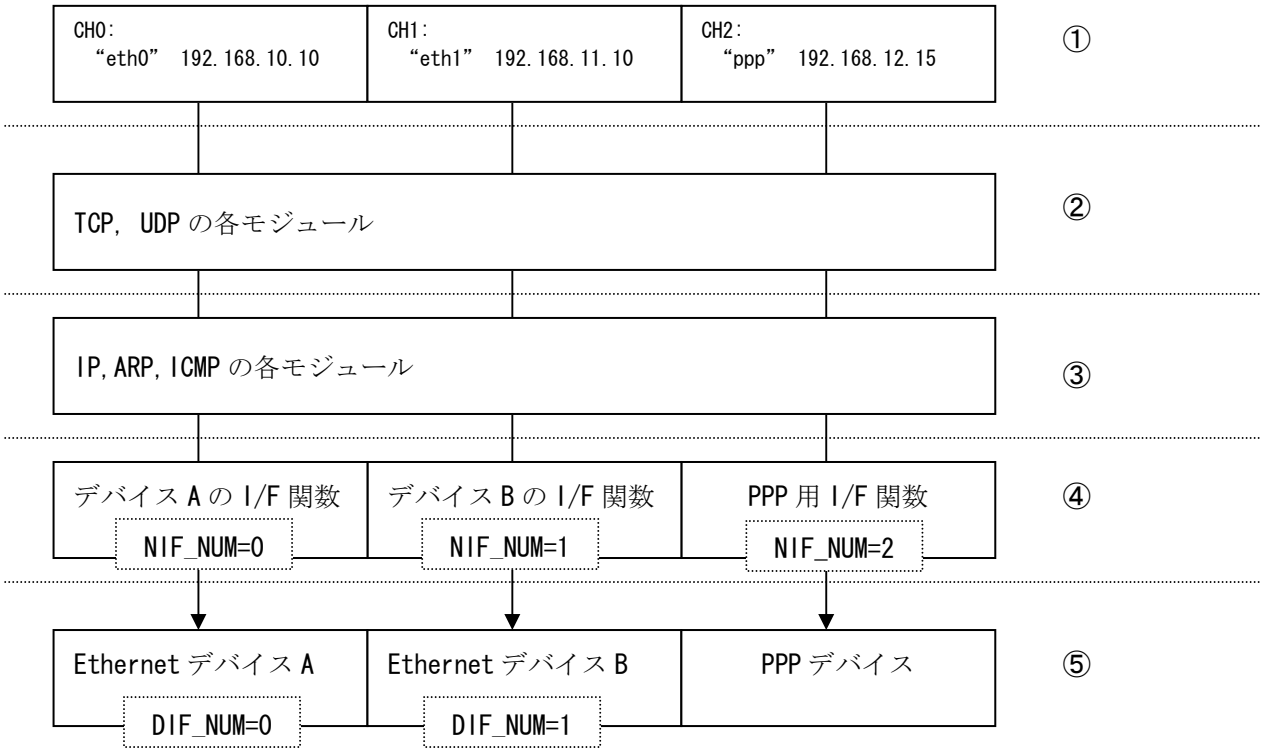
マルチチャネル対応プロトコルスタックは複数のネットワークインターフェース例えば複数の Ethernet や PPP と Ethernet の併用などに対応した TCP/IP プロトコルスタックです。NORTi TCP/IP プロトコルスタック Ver4.08.12 以降では、標準のプロトコルスタックで複数のネットワークインターフェースが使用可能になりました。複数のネットワークインターフェースを使用する場合は、コンフィグレーションやサービスコールの使用方法等が多少異なります。本章ではユーザーズガイドに記述されていない内容を説明します。

1.2 インストール

マルチチャネル対応プロトコルスタックの標準化に伴い、MPT フォルダは廃止になりました。標準のインストールでマルチチャネル対応プロトコルスタックがインストールされます。TCP/IP プロトコルスタックのライブラリ、ソースコード、ヘッダファイルはそれぞれ NORTi¥LIB,NORTi¥SRC,NORTi¥INC にインストールされます。

1.3 階層構造

以下は３つのネットワーク I/F を使用した階層構造の例です。



① 物理チャネル毎に IP アドレス、ゲートウェイ、サブネットマスクなどを設定できます。

② TCP、UDP の端点毎に使用しているチャネルが管理されます。

③ ARP 応答は設定したすべての IP アドレスに応答します。ICMP のリプライも同様です。

④ ネットワーク・ドライバ・インターフェースの入り口は使用するデバイスドライバ毎に定義します。

⑤ チャネル毎にデバイスドライバを定義します。

1.4 ネットワーク・ドライバ・インターフェース

ネットワーク・ドライバ・インターフェースでは、ネットワーク・ドライバとの物理的な処理結合がなされます。NORTi TCP/IP プロトコルスタックは、この結合が静的(コンパイル+リンク)に行われます。また、プロトコルスタックは、ネットワーク・ドライバ・インターフェースのメイン関数 lan_nif_dev を通して実行されます。

以下はインターフェース部の例です。

```
/************************************************************************
* Network Interface Function for LAN Controller
*
************************************************************************/

ER lan_nif_dev(T_NIF *nif,  FN fncd, ...)
{
    va_list plist;
    ER ercd;
    UB *macaddr;
    VP head;
    VP buf;
    int len, bufsz;
    TMO tmo;

    va_start(plist, fncd);

    switch (fncd){
    /* LAN コントローラの初期化 */
    case TFN_NET_INI:
        nif->type = NI_ETH;
        macaddr = va_arg(plist, UB*);
        ercd = lan_ini(macaddr);
        break;

    /* 受信割込み待ち */
    case TFN_NET_WAI_RCV:
        tmo = va_arg(plist, TMO);
        ercd = lan_wai_rcv(tmo);
        break;

    /* 送信割込み待ち */
    case TFN_NET_WAI_SND:
        tmo = va_arg(plist, TMO);
        ercd = lan_wai_snd(tmo);
        break;

    /* 受信パケット読み出し */
    case TFN_NET_RED_PKT:
        buf = va_arg(plist, VP);
        len = va_arg(plist, int);
        ercd = lan_read_pkt(buf, len);
        break;

    /* 受信パケット末尾まで読み出し */
    case TFN_NET_RED_PKT_END:
        buf = va_arg(plist, VP);
        len = va_arg(plist, int);
```

1.5 ネットワーク・ドライバ・インターフェースに必要な処理

ネットワーク・ドライバ・インタフェースのメイン関数

主(デフォルト)チャネル用

```
ER lan_nif_dev(T_NIF *nif,  FN fncd, ...);
```

副チャネル用

```
ER lan_nif_dev1(T_NIF *nif,  FN fncd, ...);
ER lan_nif_dev2(T_NIF *nif,  FN fncd, ...);
```

※副チャネルが 2 つ以上の場合には、リンクモジュール(nonelan.c)のカスタマイズが必要です。

```
ER lan_nif_dev(T_NIF *nif,  FN fncd [, argument]...)
  net                   ネットワークインターフェース制御ブロック
  fncd                  ファンクションコード
  [, argument]...       オプションの引数
```

TFN_NET_INI

デバイスの初期化を行います。lan_ini 相当を呼び出します。

TFN_NET_WAI_RCV

受信割り込み待ちを行います。lan_wai_rcv 相当を呼び出します。

TFN_NET_WAI_SND

送信割り込み待ちを行います。lan_wai_snd 相当を呼び出します。

TFN_NET_RED_PKT

受信パケットの読み出しを行います。lan_read_pkt 相当を呼び出します。

TFN_NET_RED_PKT_END

受信パケットを末尾まで読み出します。lan_read_pkt_end 相当を呼び出します。

TFN_NET_WRI_PKT

送信パケットの書き込みを行います。lan_write_pkt 相当を呼び出します。

TFN_NET_WRI_M_PKT

送信パケットの書き込みを行います。lan_write_m_pkt 相当を呼び出します。

TFN_NET_RCV_LEN

受信パケット長を得る。lan_received_len 相当を呼び出します。

TFN_NET_IGN_PKT

受信パケットを破棄する。lan_ignore_pkt 相当を呼び出します。

TFN_NET_ERR

ドライバのエラー処理を行います。 従来の lan_error 相当を呼び出します。

TFN_GET_NIF_NUM

割り当てるチャネル番号を返します。

TFN_NET_EXT

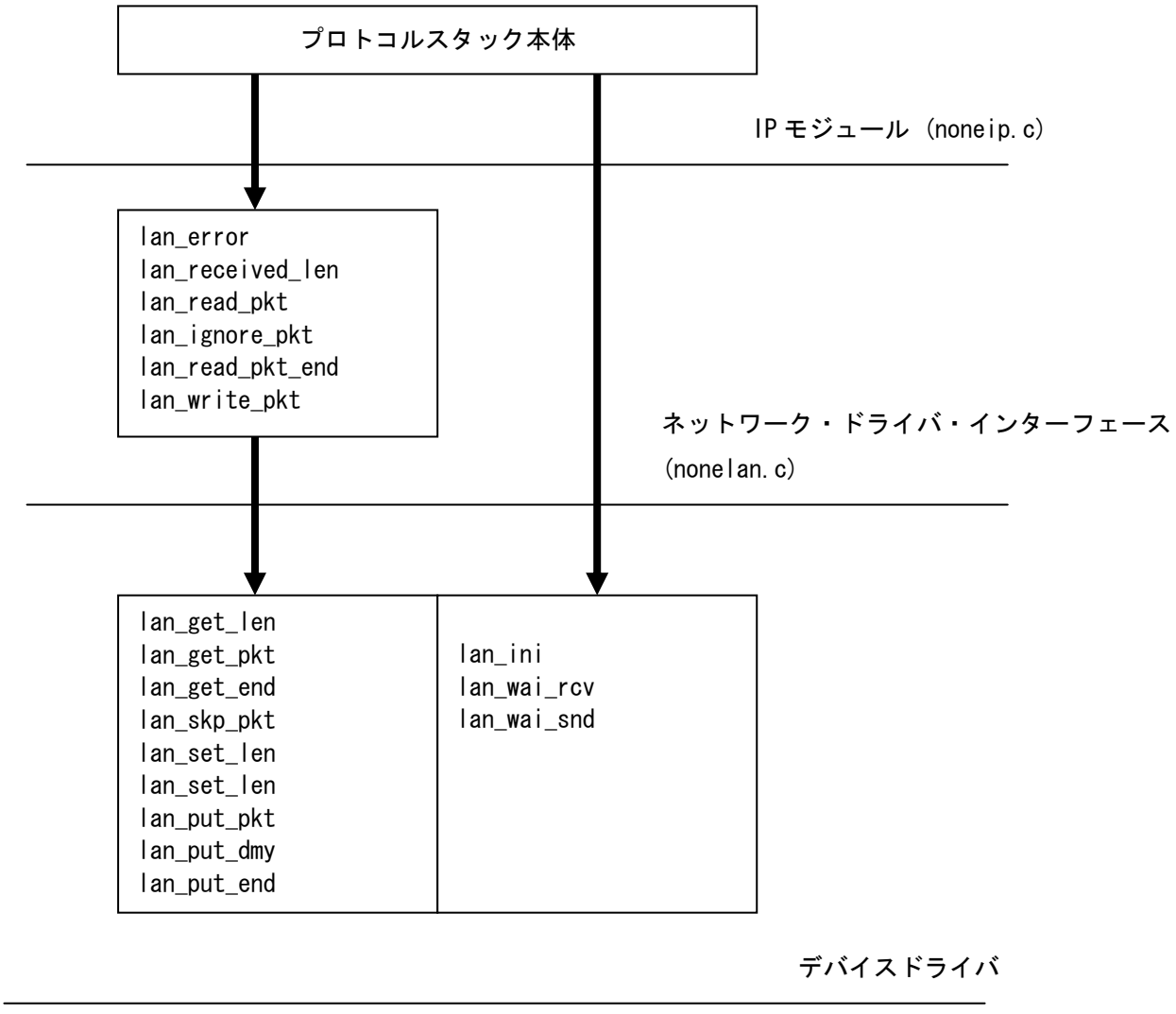
デバイスの終了処理を行います。 lan_ext_dev 相当を呼び出します。

※ 各ドライバ関数の使い方は「ユーザーズガイド」をご覧ください。

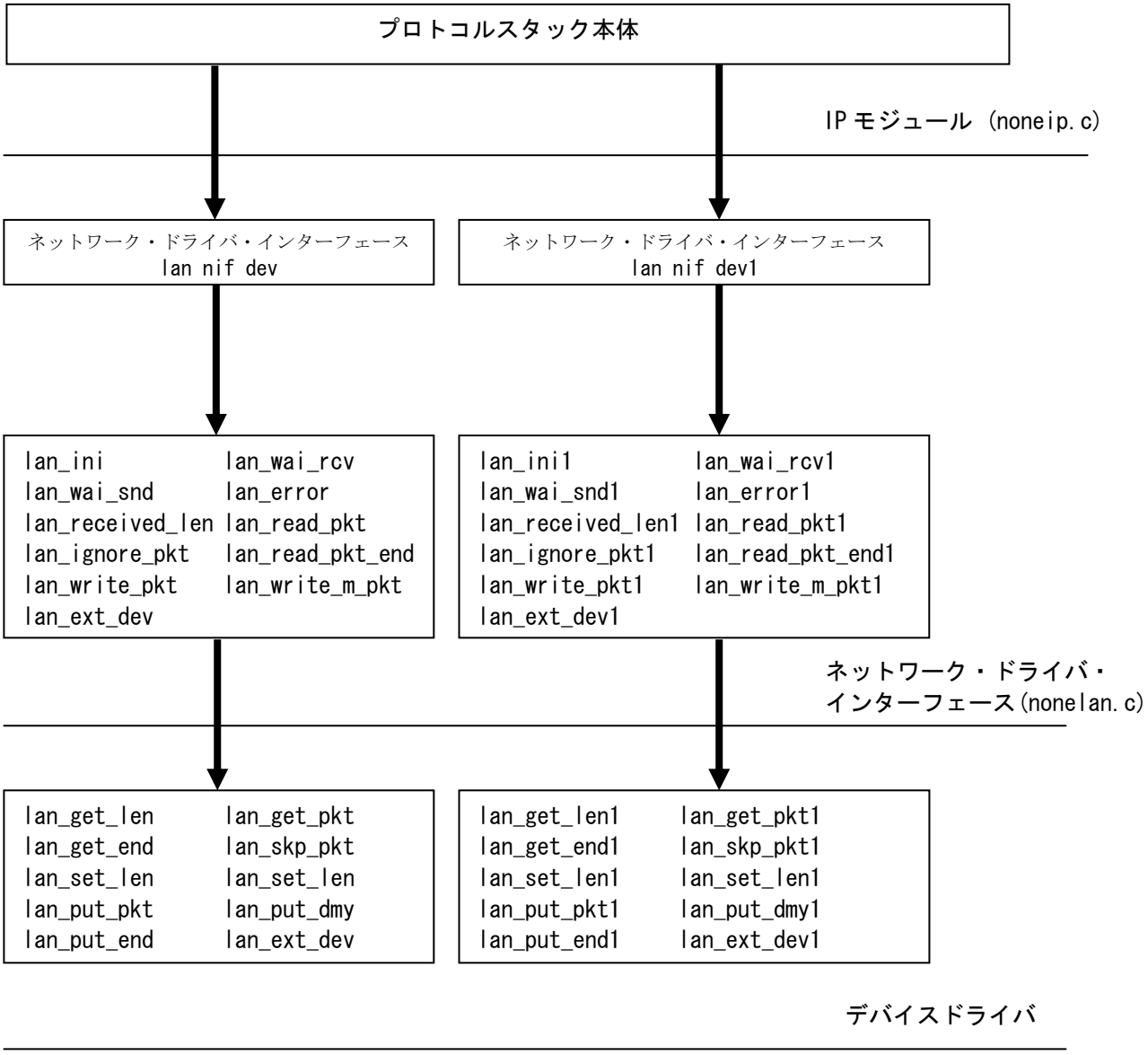
1.6 データリンクモジュールとデバイスドライバの統合

従来のプロトコルスタックの関数はデバイスドライバを直接的にコールしていましたが、マルチチャネル版プロトコルスタックでは、すべてネットワーク・ドライバ・インターフェースを経由してデバイスドライバがコールされるようになります。

従来の構成



マルチチャネル版構成


プロトコルスタック本体
IP モジュール (noneip.c)
ネットワーク・ドライバ・インターフェース
lan nif dev
ネットワーク・ドライバ・インターフェース
lan nif dev1
lan_ini
lan_wai_snd
lan_received_len
lan_ignore_pkt
lan_write_pkt
lan_ext_dev
lan_wai_rcv
lan_error
lan_read_pkt
lan_read_pkt_end
lan_write_m_pkt
lan_ini1
lan_wai_snd1
lan_received_len1
lan_ignore_pkt1
lan_write_pkt1
lan_ext_dev1
lan_wai_rcv1
lan_error1
lan_read_pkt1
lan_read_pkt_end1
lan_write_m_pkt1
ネットワーク・ドライバ・
インターフェース(nonelan.c)
lan_get_len
lan_get_end
lan_set_len
lan_put_pkt
lan_put_end
lan_get_pkt
lan_skp_pkt
lan_set_len
lan_put_dmy
lan_ext_dev
lan_get_len1
lan_get_end1
lan_set_len1
lan_put_pkt1
lan_put_end1
lan_get_pkt1
lan_skp_pkt1
lan_set_len1
lan_put_dmy1
lan_ext_dev1
デバイスドライバ

1.7 副チャネルのリンク方法

副チャネルを使用する場合には、アプリケーション側でネットワーク・ドライバ・インターフェースとドライバを相互に結合します。結合はネットワーク・ドライバ・インターフェースには、NIF_NUM と DIF_NUM マクロを設定することで実現します。

NIF_NUM は、ネットワーク・ドライバ・インタフェースの識別番号を指定します。1,2,未定義が指定可能です。このマクロが定義されない場合には、デフォルト I/F が指定されたことになります。

DIF_NUM は、ドライバ・インタフェースの識別番号を指定します。0,1,2,未定義が指定可能です。このマクロが定義されない場合には、デフォルト I/F が指定されたことになります。
※NIF_NUM が未定義の場合には、本定義にかかわらずデフォルトになります。
なお、ネットワーク・ドライバ・インターフェース(nonelan.c) は NORTi¥SRC に収録されています。

```
例)
(CC) (CFLAGS) -DNIF_NUM=1 -DDIF_NUM=0 NORTi¥SRC¥nonelan.c
CC      = コンパイルコマンド
CFLAGS  = コンパイラオプション
```

この例では、リンクモジュールには、次のシンボルが生成されます。

```
ER   lan_error1(ER);
UH   lan_received_len1(void);
BOOL lan_read_pkt1(void *, int);
void lan_ignore_pkt1(void);
BOOL lan_read_pkt_end1(void *, int, int);
BOOL lan_write_pkt1(const void *head, int hlen, const void *data, int dlen);
BOOL lan_write_m_pkt1(T_DATA *p_data, int num);
ER   lan_nif_dev1(T_NIF *nif,  FN fncd, ...);
```

1.8 副チャネル用のドライバ作成方法

デフォルトチャネルのインターフェースでは、次の処理があります。

```
ER lan_ini(UB *);
ER lan_wai_rcv(TMO);
ER lan_wai_snd(TMO);
ER lan_get_len(UH *);
ER lan_get_pkt(void *, int);
ER lan_skp_pkt(int);
ER lan_get_end(void);
ER lan_set_len(int);
ER lan_put_pkt(const void *, int);
ER lan_put_dmy(int);
ER lan_put_end(void);
ER lan_ext_dev(void);
```

副チャネルでは、これらシンボルのサフィックスにチャネル番号を付加します。

```
ER lan_ini1(UB *);
ER lan_wai_rcv1(TMO);
ER lan_wai_snd1(TMO);
ER lan_get_len1(UH *);
ER lan_get_pkt1(void *, int);
ER lan_skp_pkt1(int);
ER lan_get_end1(void);
ER lan_set_len1(int);
ER lan_put_pkt1(const void *, int);
ER lan_put_dmy1(int);
ER lan_put_end1(void);
ER lan_ext_dev1(void);
```

1.9 マルチチャネル用に追加／変更された関数と機能

1.9.1 プロトコルスタックの初期化

形式	ER tcp_ini(void);
戻値	E_OK　　正常終了 負の値　　内部で OS 資源の生成に失敗した
解説	プロトコルスタックの内部で使用する資源の生成とデータの初期化を行います。 各サービスコールを呼び出す前に必ず呼び出してください。 内部では tcp_nif_ini を使用してデフォルトネットワークインターフェースの初期化を行います。初期化の際にはデフォルトで以下の変数に設定した値が使用されます。 T_NIF　default_inet;　　デフォルトネットワーク I/F 制御ブロック const char default_inet_name[] = "eth0"; デフォルトネットワーク I/F 名 NIF_FP default_inet_func　= lan_nif_dev;デフォルトネットワーク I/F 関数 UB　　ethernet_addr[]　= { 0, 0, 0, 0, 0, 0 };　MAC アドレス UB　　default_ipaddr[]　= { 0, 0, 0, 0 };　　ローカル IP アドレス UB　　subnet_mask[]　　= { 255, 255, 255, 0 };　サブネットマスク UB　　default_gateway[] = { 0, 0, 0, 0 };　　デフォルトゲートウェイ PPP使用時には#include “nonetc.h” の前に#define PPPを定義するとPPPがデフォルトのネットワークインターフェースとして設定されます。 T_NIF　default_inet;　　デフォルトネットワーク I/F 制御ブロック const char default_inet_name[] = "ppp";　デフォルトネットワーク I/F 名 NIF_FP default_inet_func　= ppp_nif_dev; デフォルトネットワーク I/F 関数

1.9.2 ネットワークインターフェースの初期化

形式	ER tcp_nif_ini (T_NIF* nif, const char* name, NIF_FP func, T_NIF_ADDR *addr); nif　　ネットワークインターフェース制御ブロック name　ネットワークインターフェース名 func　ネットワークインターフェース関数名 addr　ネットワークインターフェースアドレス情報 typedef struct t_nif_addr { 　UB *hwaddr;　Ethernet アドレス 　UB *ipaddr;　IP アドレス 　UB *gateway;　デフォルトゲートウェイ 　UB *mask;　　サブネットマスク } T_NIF_ADDR;
戻値	正の値ならば、割り当てられたチャネル番号 E_OBJ　　チャネルが既に使用中 E_SYS　　管理ブロック用のメモリが確保できない
解説	ネットワークインターフェースの初期化を行います。 ネットワークインターフェース制御ブロックはネットワークインターフェース毎に必要な情報が保存されます。ネットワークインターフェース毎に領域を確保し、アプリケーション側で管理してください。中身は変更しないでください。 この関数は tcp_ini 呼出し後、ネットワークインターフェースが 1 つの場合、呼び出しの必要はありません。各 API を使用する前に呼び出す必要があります。 戻り値で得られるチャネル番号は使用するネットワークインターフェースのコンパイル時に指定する NIF_NUM と同じ値が返ります。

例	PPP のネットワーク I/F を追加する
	
	T_NIF net_ppp;
	UB ppp_addr[]　= { 0, 0, 0, 0, 0, 0 };
	UB ppp_ipaddr[]　= { 192, 168, 102, 11 }; /* IP アドレス */
	UB ppp_gateway[] = { 0, 0, 0, 0　};　　/* ゲートウェイ */
	UB ppp_mask[4]　= { 255, 255, 255, 0 };　/* サブネットマスク */
	
	/* PPP の初期化 */
	addr.hwaddr　= ppp_addr;
	addr.ipaddr　= ppp_ipaddr;
	addr.gateway = ppp_gateway;
	addr.mask　　= ppp_mask;
	ercd = tcp_nif_ini(&net_ppp, "ppp", ppp_nif_ini, &addr);

1.9.3 ネットワークI/Fのオプション設定

形式	ER netif_set_opt(T_NIF *nif, INT optname, const VP optval, INT optlen); nif　　ネットワーク I/F 制御ブロック optname オプションの種類 optval　オプション値が設定されているバッファのポインタ optlen　オプション値の長さ optname には以下が設定できます SET_IF_MTU　　MTU の値を設定します SET_IP_TTL　　TTL の値を設定します SET_IP_MTTL　　マルチキャストパケットの TTL の値を設定します SET_IP_REASM_TMO　　IP フラグメント再構築タイムアウトを設定します SET_TCP_SYN_RCNT　　SYN の再送回数を設定します SET_TCP_DAT_RCNT　　データの再送回数を設定します SET_TCP_RTO_INI　　TCP 再送時間の初期値を設定します SET_TCP_RTO_MIN　　TCP 再送タイムアウトの下位境界を設定します SET_TCP_RTO_MAX　　TCP 再送タイムアウトの上位境界を設定します SET_TCP_KEEPALIVE_TMO　キープアライブタイムアウトを設定します SET_TCP_KEEPALIVE_PRO　キープアライブプローブインターバルを設定します SET_TCP_KEEPALIVE_SUC　キープアライブプローブタイムアウトを設定します SET_TCP_DACK_TMO　　遅延 ACK の遅延時間を設定します SET_TCP_DUP_ACK　　重複 ACK をエラーとして扱う回数を設定します。 各オプションで指定するタイプは次のようになります SET_IF_MTU　　UH 型 SET_IP_TTL　　UB 型 SET_IP_MTTL　　UB 型 SET_IP_REASM_TMO　　UH 型 SET_TCP_SYN_RCNT　　UH 型 SET_TCP_DAT_RCNT　　UH 型 SET_TCP_RTO_INI　　UW 型 SET_TCP_RTO_MIN　　UW 型 SET_TCP_RTO_MAX　　UW 型 SET_TCP_KEEPALIVE_TMO　UW 型 SET_TCP_KEEPALIVE_PRO　UW 型 SET_TCP_KEEPALIVE_SUC　UW 型 SET_TCP_DACK_TMO　　UH 型 SET_TCP_DUP_ACK　　UH 型

戻値	E_OK　正常終了 E_OBJ　ネットワーク I/F 制御ブロックが不正
解説	ネットワーク I/F で使用するオプションを設定します。この関数はネットワークインターフェースの初期化後(tcp_ini 後)、通信が開始される前(TCP の場合は接続前)に使用してください。(通常は、tcp_ini 直後に使用してください。)

1.9.4 ネットワークI/Fのオプション設定（ネットワークI/F名指定）

形式	ER netif_set_byname(const char *name, INT optname, const VP optval, INT optlen); name　ネットワーク I/F 名("eth1" など) optname　オプションの種類 optval　オプション値が設定されているバッファのポインタ optlen　オプション値の長さ optname には以下が設定できます SET_IF_MTU　MTU の値を設定します SET_IP_TTL　TTL の値を設定します SET_IP_MTTL　マルチキャストパケットの TTL の値を設定します SET_IP_REASM_TMO　IP フラグメント再構築タイムアウトを設定します SET_TCP_SYN_RCNT　SYN の再送回数を設定します SET_TCP_DAT_RCNT　データの再送回数を設定します SET_TCP_RTO_INI　TCP 再送時間の初期値を設定します SET_TCP_RTO_MIN　TCP 再送タイムアウトの下位境界を設定します SET_TCP_RTO_MAX　TCP 再送タイムアウトの上位境界を設定します SET_TCP_KEEPALIVE_TMO　キープアライブタイムアウトを設定します SET_TCP_KEEPALIVE_PRO　キープアライブプローブインターバルを設定します SET_TCP_KEEPALIVE_SUC　キープアライブプローブタイムアウトを設定します SET_TCP_DACK_TMO　遅延 ACK の遅延時間を設定します SET_TCP_DUP_ACK　重複 ACK をエラーとして扱う回数を設定します。 各オプションで指定するタイプは次のようになります SET_IF_MTU　UH 型 SET_IP_TTL　UB 型 SET_IP_MTTL　UB 型 SET_IP_REASM_TMO　UH 型 SET_TCP_SYN_RCNT　UH 型 SET_TCP_DAT_RCNT　UH 型 SET_TCP_RTO_INI　UW 型 SET_TCP_RTO_MIN　UW 型

	SET_TCP_RTO_MAX　　UW 型 SET_TCP_KEEPALIVE_TMO　UW 型 SET_TCP_KEEPALIVE_PRO　UW 型 SET_TCP_KEEPALIVE_SUC　UW 型 SET_TCP_DACK_TMO　　UH 型 SET_TCP_DUP_ACK　　UH 型
戻値	E_OK　　正常終了 E_OBJ　ネットワーク I/F 制御ブロックが不正
解説	ネットワーク I/F で使用するオプションを設定します。この関数はネットワークインターフェースの初期化後(tcp_ini 後)、通信が開始される前(TCP の場合は接続前)に使用してください。(通常は、tcp_ini 直後に使用してください。)

1.9.5 ネットワークI/Fオプション情報の取得

形式	ER netif_get_opt(T_NIF *nif, INT optname, VP optval, INT optlen); nif　　ネットワーク I/F 制御ブロック optname　オプションの種類 optval　オプション値を取得するバッファのポインタ optlen　オプション取得するバッファのサイズ optname には以下が設定できます SET_IF_MTU　　MTU の値を取得します SET_IP_TTL　　TTL の値を取得します SET_IP_MTTL　　マルチキャストパケットの TTL の値を取得します SET_IP_REASM_TMO　IP フラグメント再構築タイムアウトを取得します SET_TCP_SYN_RCNT　SYN の再送回数を取得します SET_TCP_DAT_RCNT　データの再送回数を取得します SET_TCP_RTO_INI　TCP 再送時間の初期値を取得します SET_TCP_RTO_MIN　TCP 再送タイムアウトの下位境界を取得します SET_TCP_RTO_MAX　TCP 再送タイムアウトの上位境界を取得します SET_TCP_KEEPALIVE_TMO　キープアライブタイムアウトを取得します SET_TCP_KEEPALIVE_PRO　キープアライブプローブインターバルを取得します SET_TCP_KEEPALIVE_SUC　キープアライブプローブタイムアウトを取得します SET_TCP_DACK_TMO　遅延 ACK の遅延時間を取得します SET_TCP_DUP_ACK　重複 ACK をエラーとして扱う回数を取得します 各オプションで指定するタイプは次のようになります SET_IF_MTU　　UH 型

	SET_IP_TTL　　UB 型 SET_IP_MTTL　　UB 型 SET_IP_REASM_TMO　　UH 型 SET_TCP_SYN_RCNT　　UH 型 SET_TCP_DAT_RCNT　　UH 型 SET_TCP_RTO_INI　　UW 型 SET_TCP_RTO_MIN　　UW 型 SET_TCP_RTO_MAX　　UW 型 SET_TCP_KEEPALIVE_TMO　UW 型 SET_TCP_KEEPALIVE_PRO　UW 型 SET_TCP_KEEPALIVE_SUC　UW 型 SET_TCP_DACK_TMO　　UH 型 SET_TCP_DUP_ACK　　UH 型
戻値	E_OK　正常終了 E_OBJ　ネットワーク I/F 制御ブロックが不正
解説	ネットワーク I/F で設定されているオプションを取得します。

1.9.6 ネットワーク I/F オプション情報の取得（ネットワーク I/F 名指定）

形式	ER netif_get_byname(const char *name, INT optname, const VP optval, INT optlen); name　ネットワーク I/F 名 optname　オプションの種類 optval　オプション値を取得するバッファのポインタ optlen　オプション取得するバッファのサイズ optname には以下が設定できます SET_IF_MTU　　MTU の値を取得します SET_IP_TTL　　TTL の値を取得します SET_IP_MTTL　　マルチキャストパケットの TTL の値を取得します SET_IP_REASM_TMO　　IP フラグメント再構築タイムアウトを取得します SET_TCP_SYN_RCNT　　SYN の再送回数を取得します SET_TCP_DAT_RCNT　　データの再送回数を取得します SET_TCP_RTO_INI　　TCP 再送時間の初期値を取得します SET_TCP_RTO_MIN　　TCP 再送タイムアウトの下位境界を取得します SET_TCP_RTO_MAX　　TCP 再送タイムアウトの上位境界を取得します SET_TCP_KEEPALIVE_TMO　キープアライブタイムアウトを取得します SET_TCP_KEEPALIVE_PRO　キープアライブプローブインターバルを取得します SET_TCP_KEEPALIVE_SUC　キープアライブプローブタイムアウトを取得します SET_TCP_DACK_TMO　　遅延 ACK の遅延時間を取得します

	SET_TCP_DUP_ACK　　重複 ACK をエラーとして扱う回数を取得します 各オプションで指定するタイプは次のようになります SET_IF_MTU　　UH 型 SET_IP_TTL　　UB 型 SET_IP_MTTL　　UB 型 SET_IP_REASM_TMO　　UH 型 SET_TCP_SYN_RCNT　　UH 型 SET_TCP_DAT_RCNT　　UH 型 SET_TCP_RTO_INI　　UW 型 SET_TCP_RTO_MIN　　UW 型 SET_TCP_RTO_MAX　　UW 型 SET_TCP_KEEPALIVE_TMO　UW 型 SET_TCP_KEEPALIVE_PRO　UW 型 SET_TCP_KEEPALIVE_SUC　UW 型 SET_TCP_DACK_TMO　　UH 型 SET_TCP_DUP_ACK　　UH 型
戻値	E_OK　正常終了 E_OBJ　ネットワーク I/F 制御ブロックが不正
解説	ネットワーク I/F で設定されているオプションを取得します。

1.9.7 ネットワークI/Fに設定されているアドレスの変更

形式	ER netif_chg_ipa(T_NIF *nif, T_NIF_ADDR *addr, UB level);
	nif　　ネットワーク I/F 制御ブロック
	addr　ネットワーク I/F に設定するアドレス情報
	level　設定レベル
	
	typedef struct t_nif_addr {
	  UB *hwaddr;　　/* ハードウェアアドレス(未使用) */
	  UB *ipaddr;　　/* デフォルト IP アドレス */
	  UB *gateway;　/* デフォルトゲートウェイ */
	  UB *mask;　　/* サブネットマスク */
	} T_NIF_ADDR;

戻値	E_OK　　正常終了 E_OBJ　ネットワーク I/F 制御ブロックが不正
解説	設定されているネットワーク I/F のアドレスを変更します。設定レベルを 0 にすると、ネットワーク I/F で設定されている IP アドレス、デフォルトゲートウェイ、サブネットマスクを変更します。設定レベルを 1 にするとさらに通信端点で設定されている IP アドレスも変更します。この関数ではハードウェアアドレスは変更されません。 設定レベルを 2 に設定すると、処理中の各 API が E_ADDR(-191) でリターンします。待ち状態になっている処理は起床されます。

例	T_NIF *nif;
	T_NIF_ADDR new_addr;
	
	UB ipaddr[4]　= { 192, 168, 0, 99 };
	UB gateway[4]　= { 192, 168, 0, 1　};
	UB net_mask[4] = { 255, 255, 255, 0 };
	
	nif = getnif_from_name("eth0");
	
	new_addr.ipaddr　= ipaddr;
	new_addr.gateway = gateway;
	new_addr.mask　　= net_mask;
	
	ercd = netif_chg_ipa(nif, &new_addr, 0);

1.9.8 ネットワーク I/F に設定されているアドレスの変更（ネットワーク I/F 名指定）

形式	ER netif_ipa_byname(const char *name, T_NIF_ADDR *addr, UB level);
	name　ネットワーク I/F 名("eth1" など)
	addr　ネットワーク I/F に設定するアドレス情報
	level　設定レベル
	
	typedef struct t_nif_addr {
	  UB *hwaddr;　/* ハードウェアアドレス(未使用) */
	  UB *ipaddr;　/* デフォルト IP アドレス */
	  UB *gateway;　/* デフォルトゲートウェイ */
	  UB *mask;　/* サブネットマスク */
	} T_NIF_ADDR;

戻値	E_OK　正常終了 E_OBJ　ネットワーク I/F 制御ブロックが不正
解説	指定したネットワーク I/F に設定されているアドレスを変更します。設定レベルを 0 にすると、ネットワーク I/F で設定されている IP アドレス、デフォルトゲートウェイ、サブネットマスク変更します。設定レベルを 1 にするとさらに通信端点で設定されている IP アドレスも変更します。この関数ではハードウェアアドレスは変更されません。 設定レベルを 2 に設定すると、処理中の各 API が E_ADDR(-191) でリターンします。待ち状態になっている処理は起床されます。
例	T_NIF_ADDR new_addr; UB ipaddr[4]　= { 192, 168, 0, 99 }; UB gateway[4]　= { 192, 168, 0, 1　}; UB net_mask[4] = { 255, 255, 255, 0 }; new_addr.ipaddr　= ipaddr; new_addr.gateway = gateway; new_addr.mask　= net_mask; ercd = netif_ipa_byname(“eth0”, &new_addr, 0);

1.9.9 ネットワークI/F名からネットワークI/F制御ブロックを取得する

形式	T_NIF *getnif_from_name(const char *name); name　ネットワーク I/F 名("eth1" など)
戻値	NULL　≠ 正常終了(ネットワーク I/F 制御ブロック) NULL　＝ ネットワーク I/F 名が正しくないか、未初期化
解説	ネットワーク I/F 名からネットワーク I/F 制御ブロックを取得します。

1.9.10 チャネル番号からネットワークI/F制御ブロックを取得する

形式	T_NIF *getnif_from_ch(int ch); ch　tcp_ini,tcp_nif_ini で取得したチャネル番号
戻値	NULL　≠ 正常終了(ネットワーク I/F 制御ブロック) NULL　＝ チャネル番号が正しくないか、未初期化
解説	チャネル番号からネットワーク I/F 制御ブロックを取得します。

1.9.11 デフォルトのネットワークI/F制御ブロックを取得する

形式	T_NIF *getnif_default(void);
戻値	NULL　≠ 正常終了(ネットワーク I/F 制御ブロック)
解説	デフォルトのネットワーク I/F 制御ブロックを取得します。

1.9.12 PPPのネットワークI/F制御ブロックを取得する

形式	T_NIF *getnif_ppp(void);
戻値	NULL　≠ 正常終了(ネットワーク I/F 制御ブロック) NULL　＝ PPP 未使用または未初期化
解説	PPP のネットワーク I/F 制御ブロックを取得します。

1.9.13 IPアドレスからネットワークI/F制御ブロックを取得する

形式	T_NIF *getnif_addr(UW ipaddr); ipaddr　取得するネットワーク I/F 制御ブロックに設定されている IP アドレス
戻値	NULL　≠ 正常終了(ネットワーク I/F 制御ブロック) NULL　＝ ネットワーク I/F 制御ブロックが見つからないか未初期化
解説	ネットワーク I/F 制御ブロックに設定されている IP アドレスを指定し I/F 制御ブロックを取得します。

1.9.14 TCP 通信端点のチャネル番号を指定する

形式	ER tcp_set_opt(ID cepid, INT optname, const VP optval, INT optlen); cepid　TCP 通信端点 ID optname　IP_IF_NAME：ネットワーク I/F 名で指定 IP_IF_CH：チャネル番号で指定 optval　IP_IF_NAME 指定時は I/F 名(例：“eth0”) optlen　未使用(0 を指定してください)
戻値	E_OK　正常終了 E_ID　不正 ID 番号 E_NOEXS TCP 通信端点が未生成 E_OBJ　TCP 通信端点が使用中 E_PAR　指定したネットワークインターフェースが見つからない
解説	TCP の通信端点で使用するネットワーク I/F（チャネル）を指定します この関数は tcp_con_cep, tcp_acp_acp を呼び出す前に設定してください。 tcp_cre_rep で自分側の IP アドレスを指定しネットワーク I/F に割り当てることも可能ですが、本関数を使用し割り当てることも出来ます。

1.9.15 UDP通信端点のチャネル番号を指定する

形式	ER udp_set_opt(ID cepid, INT optname, const VP optval, INT optlen); cepid　UDP 通信端点 ID optname　IP_IF_NAME：ネットワーク I/F 名で指定 IP_IF_CH：チャネル番号で指定 optval　IP_IF_NAME 指定時は I/F 名(例：“eth0”) optlen　未使用(0 を指定してください)
戻値	E_OK　正常終了 E_ID　不正 ID 番号 E_OBJ　UDP 通信端点が使用中 E_NOEXS UDP 通信端点が未生成 E_PAR　指定したネットワークインターフェースが見つからない
解説	UDP の通信端点で使用するネットワーク I/F（チャネル）を指定します udp_cre_cep で自分側の IP アドレスを指定しネットワーク I/F に割り当てることも可能ですが、本関数を使用し割り当てることも出来ます。

1.10 通信端点とネットワークインターフェースの関係

1.10.1 TCP 編

通信端点は tcp_cre_cep による生成では、ネットワークインターフェースに割り当てが行われません。tcp_con_cep で自分側の IP アドレスを指定した場合、最初に IP アドレスが一致するネットワークインターフェースを検索し、見つかれば通信端点が割り当てられます。IP アドレスが一致するネットワークインターフェースが見つからない場合は、ネットワークアドレスが一致するネットワークインターフェースを検索し、最初に一致したネットワークインターフェースに割り当てられます。自分側の IP アドレスに各ネットワークインターフェースで使用されていないネットワークアドレスを指定すると E_PAR エラーが返ります。
tcp_acp_cep を呼び出した場合は、受付口生成時に指定した自分側の IP アドレスを使用する点以外は tcp_con_cep と同様です。

1.10.2 UDP 編

udp_cre_cep で自分側の IP アドレスを設定した場合、最初に IP アドレスが一致するネットワークインターフェースを検索し、見つかれば通信端点が割り当てられます。IP アドレスが一致するネットワークインターフェースが見つからない場合は、ネットワークアドレスが一致するネットワークインターフェースを検索し、最初に一致したネットワークインターフェースに割り当てられます。自分側の IP アドレスに各ネットワークインターフェースで使用されていないネットワークアドレスを指定すると E_OBJ エラーが返ります。

1.10.3 IPV4_ADDRANY

tcp_cre_rep で自分側の IP アドレスに IPV4_ADDRANY を指定し、tcp_acp_cep で受動接続を待ち受けたときに、いずれのインターフェースからの接続要求も受け付けることができます。tcp_con_cep で自分側の IP アドレスに IPV4_ADDRANY を指定した場合、指定した相手側の IP アドレスと同じネットワークアドレスを使用しているインターフェースに割り当てられます。それぞれのインターフェースで同一のネットワークアドレスを指定していた場合や、全てのインターフェースでネットワークアドレスが一致しない場合は、デフォルトチャネルに割り当てられます。
udp_cre_cep で自分側の IP アドレスに IPV4_ADDRANY を指定した場合、いずれのインターフェースからも udp_rcv_dat で受信を行うことができます。udp_snd_dat で自分側の IP アドレスに IPV4_ADDRANY を指定した場合は、tcp_con_cep と同様です。

1.11 注意事項

1.11.1 OS資源

従来のプロトコルスタックではタスク×2、メールボックス×2 分のＯＳ資源を使用してきましたが、マルチチャネルで使用する場合はタスク×2×チャネル数、メールボックス×2×チャネル数となります。カーネルのコンフィグレーションの際にご注意ください。tcp_ini や tcp_nif_ini でエラーが返る場合は XXXID_MAX が不足している可能性が高いです。
Ethernet パケット用メモリプールは全てのチャネル共通で使用します。ブロック数はデフォルトで 16 個に設定されています。チャネル数が多くなった場合は枯渇する可能性がありますので、使用するチャネル数を考慮して設定してください。設定する場合は、nonetc.h を#include する前に、ETH_QCNT マクロでブロック数を宣言してください。

1.11.2 PPP

PPP を複数の SIO チャネルで使用することはできません。

1.11.3 DHCP

tcp_ini()および tcp_nif_ini()の呼出し後に DHCP で IP アドレスの取得に失敗した場合は netif_chg_ipa()を使って IP アドレスを設定してください。

1.12 実装手順例

1.12.1 PPP と Ethernet を 1 チャネルずつ使用する場合

Ethernet をチャネル 0(デフォルト)に、PPP をチャネル 1 に設定します。デフォルトチャネルに対する設定などは特に必要ありません。設定は PPP を割当てるチャネル 1 のみです。

1)noneppp.c のコンパイル

```
(CC) (CFLAG) -def= NIF_NUM=1 ..¥..¥..¥PPP¥SRC¥noneppp.c
```

NIF_NUM=1 マクロを定義して PPP をチャネル 1 とします。ここで NIF_NUM=1 の定義を忘れると、NIF_NUM=0 となり、tcp_nif_ini 呼び出し時に重複エラーになります。

2)チャネル 1 のインターフェースの初期化
①T_NIF netif; ネットワークインターフェース制御ブロックをグローバル変数で定義します
②addr に IP アドレスなどを設定します。
③tcp_nif_ini に制御ブロック、インターフェース名、インターフェース関数、アドレス情報を指定して呼び出します。

```
ch = tcp_nif_ini(&netif, "ppp", ppp_nif_dev, &addr);
```

関数の戻り値にはコンパイル時に指定した NIF_NUM が設定されます。
以降、通信端点や受付口の割当てはこの関数で指定した、”ppp”と ch を指定して行います。
チャネル 0 にはデフォルトのインターフェース名、関数、アドレスが割当てられます。(1.9.1 プロトコルスタック初期化を参照)

1.12.2 Ethernet を 2 チャネル使用する

Ethernet を複数チャネルで使用する場合は、PPP の場合と少し異なります。Ethernet の場合はチャネル毎にドライバオブジェクトが必要になります。以下の例はドライバ A をデフォルトチャネル、ドライバ B をチャネル 1 に割当てます。 ドライバ A のコードはデフォルトチャネルに設定するので、関数名は標準の形式になっています。(ドライバの詳細はユーザーズガイド ネットワーク編 2.3 デバイスドライバを参照)
ドライバ B は副チャネルとして扱いますので、標準の関数名と別の名前で定義します。lan_ini0,lan_ini1 など関数名の後ろにつく番号がデバイス番号です。ここではドライバ B を lan_ini1 で作成してチャネル 1 に割当てます。

1) nonelan1.c のコンパイル

```
(CC) (CFLAG) -def= NIF_NUM=1,DIF_NUM=1 ..¥..¥..¥SRC¥nonelan1.c
```

これで、ドライバ B はチャネル１となり、チャネル１はドライバ B を使用することになります。
デフォルトチャネルに割当てられるインターフェースはプロトコルスタックのライブラリに入っていますので、再度コンパイルの必要はありません。

2) チャネル１のインターフェースの初期化
インターフェースの初期化は PPP とほぼ同じですが、インターフェース関数が異なります。
T_NIF netif; ネットワークインターフェース制御ブロックをグローバル変数で定義します
addr に IP アドレスなどを設定します。
tcp_nif_ini に制御ブロック、インターフェース名、インターフェース関数、アドレス情報を指定して呼び出します。

```
ch = tcp_nif_ini(&netif, "eth1", lan_nif_dev1, &addr);
```

NIF_NUM=1 と定義しましたので、インターフェース関数は lan_nif_dev1 を割当てます。 関数の戻り値にはコンパイル時に指定した NIF_NUM が設定されます。以降、通信端点や受付口の割当てはこの関数で指定した、”eth1”と ch を指定して行います。

1.12.3 新たに別のネットワークインターフェース(PPP,Ethernet 以外)を追加する

新たに PPP や Ethernet 以外のインターフェースを利用して、NORTi TCP/IP を使用する場合は、デバイスドライバと nonelan.c に相当するインターフェース関数を作成する必要があります。

1.12.4 Ethernet を 3 チャネル以上使用する

Ethernet を 3 チャネル以上使用する場合は nonelan1.c をコピーし、1.12.2 と同じ要領で NIF_NUM,DIF_NUM を変えて使用してください。

1.13 ICMPパケットの送信

icmp_snd_dat を使用して ICMP パケットを送信する場合、指定した自分側の IP アドレスと同じネットワーク I/F に割り当てられます。IP アドレスが一致するネットワーク I/F が見つからない場合は、ネットワークアドレスが一致する最初のネットワーク I/F に割り当てられます。各ネットワークインターフェースで使用されていないネットワークアドレスを指定すると E_OBJ エラーが返ります。

1.14 SNMPをマルチチャネル版で使用する場合

SNMP の通信はデフォルトのチャネルを使用して行います。複数チャネルからの受信を行うことはできません。SNMP を使用する場合はプロトコルスタックコンパイル時に SNMP マクロを定義する必要があります。

1.15 ループバックインターフェース

ループバックインターフェースを使用するとネットワーク上にパケットを配信せずに、自分自身に対してデータを送信(折り返し)することが出来ます。ループバックインターフェースはプロトコルスタックのライブラリにデフォルトで収録されており、Ethernet ドライバがリンクされなければデフォルトのインターフェースになります。

1.15.1 ループバックインターフェースの組込み方法

ループバックインターフェースと Ethernet や PPP を併用して使用することが出来ます。
ループバックインターフェースをサブチャネルとして使用する場合、nonelan1.c と nonloop.c を追加し次のようにコンパイルします。

```
(CC) (CFLAG) -def= NIF_NUM=1,DIF_NUM=1 ..¥..¥..¥SRC¥nonelan1.c
(CC) (CFLAG) -def= NIF_NUM=1,DIF_NUM=1 ..¥..¥..¥SRC¥nonloop.c
```

アプリケーションでは tcp_ini()を呼び出した後、次の方法で初期化を行います。

```
ER ercd;
T_NIF_ADDR addr;
UB ipaddr[4], gateway[4];

ipaddr[0] = 127;
ipaddr[1] = 0;
ipaddr[2] = 0;
ipaddr[3] = 1;
gateway[0] = 127;
gateway[1] = 0;
gateway[2] = 0;
gateway[3] = 1;

addr.hwaddr  = (const UB*)ethernet_addr;
addr.ipaddr  = (const UB*)ipaddr;
addr.gateway = (const UB*)gateway;
addr.mask    = (const UB*)subnet_mask;

if ((ercd = tcp_nif_ini(&net[1], "lo0", lan_nif_dev1, &addr)) < 0) {
    print("¥r¥nLoop Back Driver initialisation error ¥r¥n");
    return ercd;
}
```

第２章 アプリケーション・プロトコル・サンプル

2.1 はじめに

NORTi¥NETSMP にはアプリケーション・プロトコル・サンプルが収録されています。この章では DHCP クライアント、DNS リゾルバ、FTP サーバー、Telnet サーバー、LAN パケットダンプ機能、SNTP クライアントについて説明します。これらは netxxxx.c に使用例があります。

2.2 使用上の注意

NORTi¥NETSMP に収録されているファイルはサポート対象外のサンプル扱いとなっています。不具合修正の反映は、NORTi 本体のアップ毎に行われ即座には行えない場合があります。また、全ての機能をサポートしていないため、予告無く仕様を変更する可能性があります。

2.3 ファイル構成

NORTi¥NETSMP¥INC に収録されているファイル一覧

ファイル名	内容
noncons.h	コンソール入出力関数のヘッダ
nondhcp.h	DHCP クライアントのヘッダ
nonecho.h	Echo クライアント/サーバーのヘッダ
nonfile.h	簡易ファイルシステムのヘッダ
nonftp.h	FTP クライアント/サーバーのヘッダ
nonraws.h	RAW エコーサーバーのヘッダ
nonsntp.h	SNTP クライアントのヘッダ
nonteln.h	Telnet クライアント/サーバーのヘッダ
nontftp.h	TFTP サーバーのヘッダ
sntpcfg.h	SNTP コンフィグレーションヘッダ
sntptime.h	時刻変換関数のヘッダ

NORTi¥NETSMP¥INC に収録されているファイル一覧

ファイル名	内容
noncons.c	コンソール入出力関数のソース
nondhcp.c	DHCP クライアントのソース
nondump.c	LAN パケットダンプ用のソース
nonechc.c	Echo クライアントのソース
nonecho.c	Echo サーバーのソース
nonedns.c	DNS クライアントのソース
nonfile.c	簡易ファイルシステムのソース
nonftpc.c	FTP クライアントのソース
nonftpd.c	FTP サーバーのソース
nonping.c	ping コマンドのソース
nonramd.c	RAM ディスクのソース
nonraws.c	RAW エコーサーバーのソース
nonshel.c	コンソールと Telnet クライアントのシェルのソース
nonsntp.c	SNTP クライアントのソース

nontelc.c	Telnet クライアントのソース
nonteld.c	Telnet サーバーのソース
nontftp.c	TFTP サーバーのソース
sntptime.c	時刻変換関数のソース

2.4 DHCPクライアント

DHCP はホストのコンフィグレーションを行うために使用するアプリケーションプロトコルです。DHCP クライアントのサンプルは IP アドレス、サブネットマスク、デフォルトゲートウェイ、DNS サーバーアドレスを DHCP サーバーから取得し、それぞれ default_ipaddr、subnet_mask、default_gateway、dns_ipaddr に自動的に設定します。DHCP クライアントは内部で UDP の通信端点を 1 個使用します。
内部で使用しているバッファは Ethernet パケット用メモリプールから取得します。
リトルエンディアンモードのCPUを使用している場合、nondhcp.cをコンパイルする際にLITTLE_ENDIANマクロを定義してコンパイルしてください。

2.4.1 DHCP サーバーより構成情報を得る

形式	ER dhcp_get_data(ID cepid); cepid　内部で使用する UDP 通信端点の ID(内部で自動設定の場合は 0)
戻値	E_OK　正常終了 E_ID　通信端点 ID 番号が不正または不足 E_OBJ　UDP 通信端点が生成済みまたはポート番号既使用 E_TMOUT　通信タイムアウト(サーバーとの通信に失敗)
解説	DHCP サーバーよりデフォルトネットワーク I/F の IP アドレス、デフォルトゲートウェイ、サブネットマスク、DNS アドレスを得ます。 dhcp_get_data を呼び出した後、構成情報を再取得するには dhcp_reb_data を使用してください。

2.4.2 DHCPサーバーより構成情報を再取得する

形式	ER dhcp_reb_data(ID cepid); cepid　内部で使用する UDP 通信端点の ID(内部で自動設定の場合は 0)
戻値	E_OK　正常終了 E_ID　不正 ID 番号 E_OBJ　UDP 通信端点が生成済みまたはポート番号既使用 E_TMOUT　通信タイムアウト
解説	DHCP サーバーより構成情報を再取得します。

2.4.3 DHCPサーバーより構成情報を得る（ネットワークI/F名指定）

形式	ER dhcp_get_byname(ID cepid, const char *name); cepid　内部で使用する UDP 通信端点の ID(内部で自動設定の場合は 0) name　ネットワークインターフェース名(“eth1”等)
戻値	E_OK　正常終了 E_ID　通信端点 ID 番号が不正または不足 E_OBJ　UDP 通信端点が生成済みまたはポート番号既使用 E_TMOUT　通信タイムアウト(サーバーとの通信に失敗)
解説	DHCP サーバーより指定したネットワーク I/F の IP アドレス、デフォルトゲートウェイ、サブネットマスク、DNS アドレスを取得し内部で管理している各変数に設定されます。 dhcp_get_byname を呼び出した後、構成情報を再取得するには dhcp_reb_data を使用してください。

2.4.4 DHCPサーバーより構成情報を再取得する（ネットワークI/F名指定）

形式	ER dhcp_reb_byname(ID cepid, const char *name); cepid　内部で使用する UDP 通信端点の ID(内部で自動設定の場合は 0) name　ネットワークインターフェース名(“eth1”等)
戻値	E_OK　正常終了 E_ID　不正 ID 番号 E_OBJ　UDP 通信端点が生成済みまたはポート番号既使用 E_TMOUT　通信タイムアウト
解説	指定したネットワークインターフェースから DHCP サーバーより構成情報を再取得します。 取得した構成情報は指定したネットワーク I/F 用に設定されます。

2.4.5 DHCP 周期タスクの起動

形式	ER dhcp_sta_tsk(ID tskid, ID cepid, ID almid, DHCP_CALLBACK callback); tskid　　タスクで使用する ID(未指定の場合は 0) cepid　　内部で使用する UDP 通信端点の ID(未指定の場合は 0) almid　　内部で使用するアラームの ID(未指定の場合は 0) callback　　コールバック関数(未指定の場合は NULL)
戻値	E_OK　　正常終了 E_ID　　通信端点 ID 番号が不正またはタスク ID が範囲外 E_OBJ　　UDP 通信端点生成済みかポート番号が既使用またはタスクが生成済み
解説	リース切れになった IP アドレスを自動で更新するタスクを起動します。 コールバック関数は DHCP の更新が行われた場合と内部でエラーが発生した 場合に呼び出されます。

2.5 DNSリゾルバ

DNS リゾルバは DNS サーバーに mispo.co.jp などのドメイン名から IP アドレスを取得します。
DNSリゾルバは内部でUDP通信端点を 1 個使用します。内部で使用しているバッファはEthernetパケット用メモリプールから取得します。リトルエンディアンモードのCPUを使用している場合、nonedns.cをコンパイルする際にLITTLE_ENDIANマクロを定義してコンパイルしてください。

2.5.1 ドメイン名から IP アドレスを取得する

形式	ER dns_resolver(ID cepid, UW dns_ip, char *name, T_IPEP *host); cepid　使用する UDP 通信端点 ID(内部で自動設定の場合は 0) dns_ip　DNS サーバーの IP アドレス name　IP アドレスを取得するドメイン名 host　ホストのアドレス情報
戻値	E_OK　正常終了 E_ID　通信端点 ID 番号が不正または不足 E_OBJ　UDP 通信端点生成済みかポート番号が既使用またはタスクが生成済み E_TMOUT　通信タイムアウト(サーバーとの通信に失敗)
解説	name で指定したドメイン名を IP アドレスに変換するために dns_ip に問い合せます。

2.5.2 ドメイン名からIPアドレスを取得する（ネットワークI/F名指定）

形式	ER dns_resolver_byname(ID cepid, UW dns_ip, char *name, T_IPEP *host, const char *ch_name); cepid　使用する UDP 通信端点 ID(内部で自動設定の場合は 0) dns_ip　DNS サーバーの IP アドレス name　IP アドレスを取得するドメイン名 host　ホストのアドレス情報 ch_name　ネットワークインターフェース名(“eth1”等)
戻値	E_OK　正常終了 E_ID　通信端点 ID 番号が不正または不足 E_OBJ　UDP 通信端点生成済みかポート番号が既使用またはタスクが生成済み E_TMOUT　通信タイムアウト(サーバーとの通信に失敗)
解説	name で指定したドメイン名を IP アドレスに変換するために、dns_ip に問い合せます。 DNS パケットは指定したネットワークインターフェースから送信されます。

2.6 FTPサーバー

FTP はファイルを転送するためのプロトコルです。FTP サーバーは RAMDISK を使用した File System を使用して、FTP クライアントプログラムからの要求に応答します。

2.6.1 制約事項

- ディレクトリ階層をサポートしていません
- 複数クライアントの同時接続に対応していません
- Windows の DOS 窓などコマンドラインベースの FTP クライアントのみ対応しています

これらの機能は「NORTi File System Ver.4」に付属の FTP サーバーで対応しています。

2.6.2 FTPサーバー生成

形式	ER ftp_cre_srv(T_FTP *ftp, ID tskid, ID cepid1, ID cepid2, ID repid, FTP_CALLBACK callback); ftp　内部で使用する FTPd 用コントロールブロックのポインタ tskid　内部で使用するタスク ID(内部で自動設定の場合は 0) cepid1　内部で使用する TCP 通信端点 ID1(内部で自動設定の場合は 0) cepid2　内部で使用する TCP 通信端点 ID2(内部で自動設定の場合は 0) repid　内部で使用する TCP 受付口 ID(内部で自動設定の場合は 0) callback パスワードチェック用コールバック関数のアドレス
戻値	E_OK　正常終了 E_ID　ID 番号が不正または不足 E_OBJ　TCP 通信端点またはタスクが生成済み
解説	FTP サーバーで使用する通信端点とタスクの生成を行います。コントロールブロック ftp には、アプリケーションで静的変数として宣言した T_FTP 構造体のポインタを渡してください。callback はクライアントに対してユーザー名、パスワードを使った認証を行うことができます。この関数はアプリケーション側で定義してください。NULL で定義した場合、認証を行いません。

例	パスワードチェック関数の使用例
	
	char *ftp_passwd_check(T_FTP *ftp, const char *user, const char *pass)
	{
	　if ((strcmp(user, "mispo") == 0)
	　 && (strcmp(pass, "12345") == 0))
	　　return "230 Logged in¥r¥n";
	　else
	　　return NULL;
	}

2.6.3 FTPサーバー起動

形式	ER ftp_sta_srv(T_FTP *ftp); ftp　　内部で使用する FTPd 用コントロールブロックのポインタ
戻値	E_OK　　正常終了 E_NOEXS　FTP サーバーが未生成 E_OBJ　　FTP サーバーが既に起動されている
解説	ftp_cre_srv で生成したコントロールブロックのポインタを指定して、FTP サーバーを起動します。

2.6.4 対応コマンド一覧

コマンド	使い方	説明
USER	USER *username*	ユーザーを識別する
PASS	PASS *password*	パスワードを渡す
QUIT	QUIT	ログアウトする
PORT	PORT *host-port*	データ転送ホストの IP アドレスとポート番号を指定する
PASV	PASV	パッシブモードに移行する
TYPE	TYPE *type-code*	ファイルのタイプを指定する
APPE	APPE *pathname*	ファイルを新たに作成する。ない場合は新たに作成する
DELE	DELE *pathname*	ファイルを削除する
LIST	LIST [*pathname*]	ファイル／ディレクトリ情報を返す
NLST	NLIST [*pathname*]	ファイル名のリストを返す

2.7 Telnetサーバー

Telnet はリモートにあるホストを端末から操作するための仮想端末ソフトウェアです。Telnet サーバーはTelnetd タスク、Shell タスク(nonshel.c)の２つのタスクで構成されています。Telnet はTCP/IP 以外にも RS232-C を使って使用することができます。RS232-C を使用する場合は Console プログラム(noncons.c)と Shell タスクを組み合わせて使用します。Telnet サーバーのコールバック関数を拡張することで、独自のコマンドを簡単に追加できます。

2.7.1 Telnetサーバーの初期化

形式	ER telnetd_ini(T_TELNETD *t, ID tskid, ID mbfid, ID cepid, ID repid); t　内部で使用する Telnetd 用コントロールブロックのポインタ tskid　内部で使用するタスク ID(内部で自動設定の場合は 0) mbfid　内部で使用するメッセージバッファ ID(内部で自動設定の場合は 0) cepid　内部で使用する TCP 通信端点 ID(内部で自動設定の場合は 0) repid　内部で使用する TCP 受付口 ID(内部で自動設定の場合は 0)
戻値	E_OK　正常終了 E_ID　ID 番号が不正または不足 E_OBJ　TCP 通信端点、タスクまたはメッセージバッファが生成済み
解説	Telnet サーバーを初期化します。ここで指定した Telnetd 用コントロールブロックを使って、shell_ini 関数を呼び出しシェルタスクを起動すると Telnet サーバーが起動されます。

2.7.2 Shellタスクの起動

形式	ER shell_ini(VP t, ID tskid, ID mbfid, PASSWD_CHECK callback); t　内部で使用する Telnetd またはコンソール用コントロールブロックのポインタ tskid　内部で使用するタスク ID(内部で自動設定の場合は 0) mbfid　内部で使用するメッセージバッファ ID(内部で自動設定の場合は 0) callback パスワードチェック用コールバック関数のアドレス
戻値	E_OK　正常終了 E_ID　ID 番号が不正または不足 E_OBJ　タスク、メッセージバッファが生成済み
解説	Shell タスクを起動します。telnetd_ini 関数または console_ini 関数で指定したコントロールブロックを使って、この関数を呼び出すと Telnet サーバーが起動されます。

2.7.3 RS232-Cを使ったTelnetサーバーの初期化

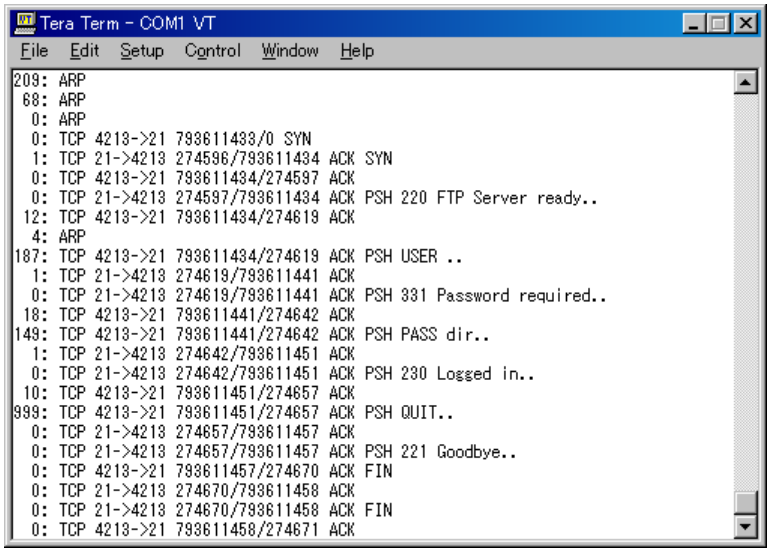
形式	ER console_ini(T_CONSOLE *t, ID tskid, ID mbfid, INT ch, const char *param); t　内部で使用するコンソール用コントロールブロックのポインタ tskid　内部で使用するタスク ID(内部で自動設定の場合は 0) mbfid　内部で使用するメッセージバッファ ID(内部で自動設定の場合は 0) ch　使用する RS232-C の論理チャネル番号 param　RS232-C の設定パラメータ
戻値	E_OK　正常終了 E_ID　ID 番号が不正または不足 E_OBJ　TCP 通信端点、タスクまたはメッセージバッファが生成済み
解説	RS232-C を使った Telnet サーバーを初期化します。ここで指定したコンソール用コントロールブロックを使って、shell_ini 関数を呼び出しシェルタスクを起動すると RS232-C を使った Telnet サーバーが起動されます。

2.7.4 Telnetサーバーのコールバック関数

Telnet サーバーにコマンドが送信されると、関数 BOOL telnetd_callback(T_TERMINAL *t, char *s) が呼び出されます。第 2 パラメータの’char *s’には入力された文字列が入っています。この文字列を解析して、コマンドに対応した処理を行うことができます。この関数はアプリケーション側で用意してください。telnetd_callback の使用例は netxxxx.c をご覧ください。

2.8 LANパケットダンプ機能

LAN パケットダンプ機能を使用すると Ethernet のパケットダンプを以下のように RS232-C に出力することができます。



2.8.1 LANパケットダンプ機能のコンフィグレーション

LAN パケットダンプ機能を有効にするには、NORTi¥SRC¥nonelan.c を、’DUMP’マクロを定義してコンパイルします。ダンプを出力する RS232-C のデフォルトチャネルは論理チャネル番号’1’です。それ以外のチャネルで使用する場合は、NORTi¥SRC¥ nondump.c コンパイル時にチャネル番号を指定してください。チャネル番号は’CH’マクロを使って指定します。使用可能なマクロは以下のとおりです。

マクロ名	デフォルト値	説明
CH	1	出力する RS232-C のチャネル番号
QCNT	10	ダンプで使用するキューの数
TCP	未使用	TCP のダンプのみを出力するためのフィルター
ARP	使用	ARP パケットをダンプする
DATA	未使用	詳細なダンプ表示を行う

2.8.2 LAN パケットダンプ機能の初期化

形式	landump_ini(ID tskid, ID mbxid, ID mpfid); tskid　内部で使用するタスク ID(内部で自動設定の場合は 0) mbfid　内部で使用するメッセージバッファ ID(内部で自動設定の場合は 0) mpfid　内部で使用する固定長メモリプール ID(内部で自動設定の場合は 0)
戻値	E_OK　正常終了 E_ID　ID 番号が不正または不足 E_OBJ　タスク、メッセージバッファまたは固定長メモリプールが生成済み
解説	LAN パケットダンプ機能の初期化を行います。この関数を呼び出す前に出力する RS232-C の初期化が必要です。

例	ercd = ini_sio(1, "38400 B8 PN S1");
	ercd = ctl_sio(1, TSIO_RXE|TSIO_TXE|TSIO_RTSON);
	ercd = landump_ini(0, 0, 0);

2.9 TFTPサーバー

TFTP は FTP の簡易版でユーザ名、パスワードの検証を必要としないファイル転送プロトコルです。
TFTP は FTP 同様に File System を使用して、TFTP クライアントプログラムからの要求に応答します。
TFTP では UDP を使用します。

2.9.1 TFTPサーバーの初期化

形式	ER tftpsrv_ini(ID tskid, ID cepid); tskid　内部で使用するタスク ID(内部で自動設定の場合は 0) cepid　内部で使用する TCP 通信端点 ID(内部で自動設定の場合は 0)
戻値	E_OK　正常終了 E_ID　ID 番号が不正または不足 E_OBJ　TFTP サーバーが既に起動されている
解説	TFTP サーバーを起動します。

2.10 TCP,UDP ECHOサーバー

ECHO プログラムは、単純に文字列を折り返すだけのプログラムです。ECHO サーバーはクライアントから受信した文字列をそのままクライアントに送信します。

2.10.1 TCP ECHOサーバーの初期化

形式	ER tcpecho_ini(ID tskid, ID cepid, ID repid); tskid　内部で使用するタスク ID(内部で自動設定の場合は 0) cepid　内部で使用する TCP 通信端点 ID(内部で自動設定の場合は 0) repid　内部で使用する TCP 受付口 ID(内部で自動設定の場合は 0)
戻値	E_OK　正常終了 E_ID　ID 番号が不正または不足 E_OBJ　TCP ECHO サーバーが既に起動されている
解説	TCP ECHO サーバーを起動します。

2.10.2 UDP ECHOサーバーの初期化

形式	ER udpecho_ini(ID tskid, ID cepid); tskid　内部で使用するタスク ID(内部で自動設定の場合は 0) cepid　内部で使用する UDP 通信端点 ID(内部で自動設定の場合は 0)
戻値	E_OK　正常終了 E_ID　ID 番号が不正または不足 E_OBJ　UDP ECHO サーバーが既に起動されている
解説	UDP ECHO サーバーを起動します。

2.11 SNTPクライアント

SNTP クライアントのサンプルプログラムを利用して、NTP サーバーから時刻を取得できます。

2.11.1 使用方法

使用するには以下が必要です。

1) DNS Resolver
   NORTi¥NETSMP¥SRC¥nonedns.c

2) 時刻変換関数のモジュール
   NORTi¥NETSMP¥INC¥sntptime.h
   NORTi¥NETSMP¥SRC¥sntptime.c

3) アプリケーションには以下のインクルードを追加
   #include　“nonsntp.h”
   #include　“sntpcfg.h”
   #include　“sntptime.h”

注) 単に時刻を設定するだけであれば、2) 時刻変換関数のモジュールがなくても動作します。

初回のリクエストは自分の時刻は 0 にセットされ、2 回目以降のリクエストは時刻が送信タイムスタンプに設定されます。

○ マクロ

NTP_SRV_MAX

NTP_SRV_MAX を定義することで NTP サーバーの指定数を設定できます。設定する場合は sntpcfg.h をインクルードする前に定義してください。デフォルトは 1 です。

LITTLE_ENDIAN

リトルエンディアンで使用する場合は LITTLE_ENDIAN マクロを定義してコンパイルします。

○ 制限事項

・IPv4 でのみ動作します
・デフォルトインターフェースから問い合わせを行います。

○ 時刻形式

```
typedef struct my_tm
{
        int tm_sec;      /* 秒  (0..61)      */
        int tm_min;      /* 分  (0..59)      */
        int tm_hour;     /* 時  (0..23)      */
        int tm_mday;     /* 日  (1..31)      */
        int tm_mon;      /* 月  (1 月が 0)     */
        int tm_year;     /* 年  (1900 年が 0)  */
        int tm_wday;     /* 曜日(日曜日が 0)  */
        int tm_yday;     /* 1 月 1 日からの日数 */
        int tm_isdst;    /* 季節時間の有無   */
} my_tm_t;

typedef long my_time_t;  /* 暦時間 */
typedef long my_clock_t; /* システムクロック */
```

2.11.2 使用例

```
#define  NTP_SRV_MAX  2
#include "nonsntp.h"
#include "sntpcfg.h"
#include "sntptime.h"

    int ercd;
    UW dns_ip;
    T_NTP_TIM *pk_tim;
    char dns_srv[] = "192.168.0.1";
    char *sntp_srv[] = {"ntp.nict.jp","ntp.ring.gr.jp"};

    dns_ip = ascii_to_ipaddr(dns_srv);
    ercd = sntp_start(sntp_srv, dns_ip, 1000/MSEC);

    pk_tim = NULL;
    ercd = sntp_get_tim(pk_tim);
```

2.11.3 SNTPクライアントの初期化

形式	ER sntp_start(char *sntp_srv, UW dns_ip, TMO tmout); sntp_srv　NTP サーバーのアドレスリスト 　　NTP_SRV_MAX(デフォルトは 1)の数だけサーバーを設定します dns_ip　DNS サーバーのアドレス tmout　タイムアウト 　　(0 とした場合は 1000/MSEC とみなします)
戻値	0 以上の値　初期化成功 負の値　失敗
解説	タイムアウトはパケットの送受信時のタイムアウトです。sntpcfg.h をインクルードする前に SNTP_TMOUT を定義しておけば、tmout を 0 とした場合のデフォルトの値を変更できます。

2.11.4 時刻の取得

形式	ER sntp_get_tim(T_NTP_TIM *pk_tim); pk_tim　NTP フォーマットの時刻を格納するポインタ
戻値	E_OK　時刻取得、設定成功 負の値　失敗
解説	時刻を NTP サーバーに問い合わせ、システムの時刻を校正します。問い合わせでエラーがあった場合は設定されている他の NTP サーバーがあればリトライを行います。

2.11.5 時刻を文字列に変換（出力バッファ指定なし）

形式	char *my_asctime(const my_tm_t *tms); tms　my_tm_t 形式の時刻
戻値	変換した文字列へのポインタ
解説	時刻を“Wday Mon dd hh:mm:ss yyyy¥n”形式の文字列に変換します。

2.11.6 時刻を文字列に変換（出力バッファ指定あり）

形式	char *my_asctime_r(const my_tm_t *tms, char *buf); tms　　my_tm_t 形式の時刻 buf　　変換結果を格納する文字列へのポインタ
戻値	変換した文字列へのポインタ
解説	時刻を“Wday Mon dd hh:mm:ss yyyy¥n”形式の文字列に変換します。文字列バッファは 26 文字分必要です。本関数はスレッドセーフです。

2.11.7 暦時間から現地時間の文字列を算出（出力バッファ指定なし）

形式	char *my_ctime(const my_time_t *timer); timer　暦時間を受け取る変数へのポインタ
戻値	変換した文字列へのポインタ
解説	暦時間を現地時間に変換し、文字列として返します。文字列は“Wday Mon dd hh:mm:ss yyyy¥n”形式となります。

2.11.8 暦時間から現地時間の文字列を算出（出力バッファ指定あり）

形式	char *my_ctime_r(const my_time_t *timer, char *buf); timer　暦時間を受け取る変数へのポインタ buf　　変換結果を格納する文字列へのポインタ
戻値	変換した文字列へのポインタ
解説	暦時間を現地時間に変換し、文字列として返します。文字列は“Wday Mon dd hh:mm:ss yyyy¥n”形式となります。文字列バッファは 26 文字分必要です。本関数はスレッドセーフです。

2.11.9 システムの現在の暦時間を取得

形式	my_time_t my_time(my_time_t *t); t　　暦時間を受け取る変数へのポインタ
戻値	暦時間の値
解説	システムの現在の暦時間を返します。暦時間は 1/Jan/1970 AM0:00 UTC からの経過秒数です。SNTP で時刻設定してご利用ください。暦時間を戻り値で受け取る場合、引数 t には NULL が指定できます。

2.11.10 起動時からのシステムクロックを取得

形式	my_clock_t my_clock(void);
戻値	システムクロックの下位 32 ビット
解説	起動時からのシステムクロックを返します。MSEC 単位の tick count の下位 32bit のみが返ります。

2.11.11 現地時間を暦時間に変換

形式	my_time_t my_mktime(my_tm_t *t); t　　my_tm_t 形式の時刻
戻値	変換出来た場合　　変換した暦時間 変換出来ない場合　-1 パラメータが無効　-1
解説	現地時間の年、月、日、時、分、秒を暦時間に変換します。暦時間は 1/Jan/1970 からの経過秒数となります。夏時間はサポートされていません。

2.11.12 暦時間をmy_tm_tに変換（格納先指定なし）

形式	my_tm_t *my_gmtime(const my_time_t *tms); tms　　暦時間
戻値	変換出来た場合　　変換した my_tm_t 形式の時刻へのポインタ 変換出来ない場合　NULL
解説	暦時間を、年、月、日、時、分、秒に変換します。

2.11.13 暦時間をmy_tm_tに変換（格納先指定あり）

形式	my_tm_t *my_gmtime_r(const my_time_t *tms, my_tm_t *tm_time); tms　　暦時間 tm_time　変換する my_tm_t 形式の時刻へのポインタ
戻値	変換出来た場合　　変換した my_tm_t 形式の時刻へのポインタ 変換出来ない場合　NULL
解説	暦時間を、年、月、日、時、分、秒に変換します。本関数はスレッドセーフです。

2.11.14 暦時間を現地時間に変換（格納先指定なし）

形式	my_tm_t *my_localtime(const my_time_t *tms); tms　　暦時間
戻値	変換出来た場合　　変換した my_tm_t 形式の時刻へのポインタ 変換出来ない場合　NULL
解説	暦時間を、現地時間に変換します。

2.11.15 暦時間を現地時間に変換（格納先指定あり）

形式	my_tm_t *my_localtime_r(const my_time_t *tms, my_tm_t *tm_time); tms　　暦時間 tm_time　変換する my_tm_t 形式の時刻へのポインタ
戻値	変換出来た場合　　変換した my_tm_t 形式の時刻へのポインタ 変換出来ない場合　NULL
解説	暦時間を、現地時間に変換します。本関数はスレッドセーフです。

2.11.16 タイムゾーンによる時差を設定

形式	ER set_timezone_ofst(W tz); tz　　タイムゾーンによる UTC から時間差(秒)
戻値	E_OK　　正常終了 E_PAR　　設定無効
解説	タイムゾーンによる標準時からの時差を設定します。設定できる範囲は-12*3600～+14*3600 です。my_localtime, my_localtime_r, my_mktime を利用する場合に設定が必要です。 例) W tz = 9 * 60 * 60; set_timezone_ofst(tz);　/* 日本の UTC からの時間差を設定 */

2.11.17 タイムゾーンによる時差を取得

形式	W get_timezone_ofst(void);
戻値	タイムゾーンによる標準時からの時差(秒) 無効な値の場合は-1
解説	タイムゾーンによる標準時からの時差を取得します。

第３章 新規に追加された機能

3.1 TCP/IPプロトコルスタックVersion 4.08.12

TCP/IP プロトコルスタック Version 4.08.12 以降次のサービスコールが追加されました。

3.1.1 ネットワークI/Fの終了

形式	ER tcp_ext(T_NIF *nif); nif　　ネットワーク I/F 制御ブロック
戻値	＝E_OK　　正常終了 ≠E_OK　　エラー
解説	ネットワーク I/F を終了し tcp_ini()および tcp_nif_ini()で確保したリソースを解放します。複数の I/F を使用していない場合、nif には NULL を設定してください。NULL を設定するとデフォルトチャネルが対象になります。この関数を呼び出した後で再び I/F を使用する場合、単一インターフェース使用時には tcp_ini() を複数インターフェース使用時には tcp_nif_ini()呼び出す必要があります。他のタスクで API がペンディング中の場合、E_LNK もしくは E_RLWAI（未接続で IPV4_ADDRANY 指定時）で API からリターンします。

3.1.2 TCP通信端点のオプション設定

形式	ER tcp_set_opt(ID cepid, INT optname, VP optval, INT optlen); cepid　TCP 通信端点 ID optname オプションの種類 optval　オプション値が設定されているバッファのポインタ optlen　オプション値の長さ optname には以下が設定できます SET_IP_TTL　通信端点で使用する TTL の値を設定します SET_IP_TOS　通信端点で使用する TOS の値を設定します SET_TCP_MTU 通信端点で使用する MTU サイズを設定します 各オプションで指定するタイプは次のようになります SET_IP_TTL　UB 型 SET_IP_TOS　UB 型 SET_TCP_MTU UW 型
戻値	E_OK　　正常終了 E_ID　　不正 ID 番号 E_NOEXS　通信端点が未生成 E_PAR　　パラメータエラー E_OBJ　　通信端点が使用中

解説	TCP 通信端点のオプションを設定します。 TCP の通信端点の生成後で、接続前にのみ変更ができます。
例	UB new_ttl = 128; ercd = tcp_set_opt(cepid, SET_IP_TTL, &new_ttl, sizeof(new_ttl));

3.1.3 TCP通信端点に設定されているオプション情報の取得

形式	ER tcp_get_opt(ID cepid, INT optname, VP optval, INT optlen); cepid　TCP 通信端点 ID optname オプションの種類 optval　オプション値を取得するバッファのポインタ optlen　オプション取得するバッファのサイズ optname には以下が設定できます SET_IP_TTL　通信端点で設定されている TTL の値を取得します SET_IP_TOS　通信端点で設定されている TOS の値を取得します SET_TCP_MTU 通信端点で設定されている MTU サイズを取得します 各オプションで指定するタイプは次のようになります SET_IP_TTL　UB 型 SET_IP_TOS　UB 型 SET_TCP_MTU UW 型
戻値	E_OK　　正常終了 E_ID　　不正 ID 番号 E_NOEXS　通信端点が未生成 E_PAR　　パラメータエラー E_OBJ　　通信端点が使用中
解説	設定されている TCP 通信端点のオプション値を取得します。

3.1.4 UDP通信端点のオプション設定

形式	ER udp_set_opt(ID cepid, INT optname, VP optval, INT optlen); cepid　UDP 通信端点 ID optname オプションの種類 optval　オプション値が設定されているバッファのポインタ optlen　オプション値の長さ optname には以下が設定できます SET_IP_TTL　通信端点で使用する TTL の値を設定します SET_IP_TOS　通信端点で使用する TOS の値を設定します 各オプションで指定するタイプは次のようになります SET_IP_TTL　UB 型 SET_IP_TOS　UB 型
戻値	E_OK　　正常終了 E_ID　　不正 ID 番号 E_NOEXS　通信端点が未生成 E_PAR　　パラメータエラー
解説	UDP 端点で使用するオプションを設定します。 UDP の通信端点の生成後で、接続前にのみ変更ができます。

3.1.5 UDP通信端点に設定されているオプション情報の取得

形式	ER udp_get_opt(ID cepid, INT optname, VP optval, INT optlen); cepid　UDP 通信端点 ID optname オプションの種類 optval　オプション値を取得するバッファのポインタ optlen　オプション取得するバッファのサイズ optname には以下が設定できます SET_IP_TTL　通信端点で設定されている TTL の値を取得します SET_IP_TOS　通信端点で設定されている TOS の値を取得します 各オプションで指定するタイプは次のようになります SET_IP_TTL　UB 型 SET_IP_TOS　UB 型
戻値	E_OK　　正常終了 E_ID　　不正 ID 番号 E_NOEXS　通信端点が未生成 E_PAR　　パラメータエラー
解説	設定されている UDP 通信端点のオプション値を取得します。 UDP の通信端点の生成後で、接続前にのみ変更ができます。

3.1.6 デフォルトネットワークI/Fのオプション設定

形式	ER net_set_opt(INT optname, VP optval, INT optlen); optname オプションの種類 optval　オプション値が設定されているバッファのポインタ optlen　オプション値の長さ optname には以下が設定できます SET_IF_MTU　　MTU の値を設定します SET_IP_TTL　　TTL の値を設定します SET_IP_MTTL　　マルチキャストパケットの TTL の値を設定します SET_IP_REASM_TMO　　IP フラグメント再構築タイムアウトを設定します SET_TCP_SYN_RCNT　　SYN の再送回数を設定します SET_TCP_DAT_RCNT　　データの再送回数を設定します SET_TCP_RTO_INI　　TCP 再送時間の初期値を設定します SET_TCP_RTO_MIN　　TCP 再送タイムアウトの下位境界を設定します SET_TCP_RTO_MAX　　TCP 再送タイムアウトの上位境界を設定します SET_TCP_KEEPALIVE_TMO　キープアライブタイムアウトを設定します SET_TCP_KEEPALIVE_PRO　キープアライブプローブインターバルを設定します SET_TCP_KEEPALIVE_SUC　キープアライブプローブタイムアウトを設定します SET_TCP_DACK_TMO　　遅延 ACK の遅延時間を設定します SET_TCP_DUP_ACK　　重複 ACK をエラーとして扱う回数を設定します。 各オプションで指定するタイプは次のようになります SET_IF_MTU　　UH 型 SET_IP_TTL　　UB 型 SET_IP_MTTL　　UB 型 SET_IP_REASM_TMO　　UH 型 SET_TCP_SYN_RCNT　　UH 型 SET_TCP_DAT_RCNT　　UH 型 SET_TCP_RTO_INI　　UW 型 SET_TCP_RTO_MIN　　UW 型 SET_TCP_RTO_MAX　　UW 型 SET_TCP_KEEPALIVE_TMO　UW 型 SET_TCP_KEEPALIVE_PRO　UW 型 SET_TCP_KEEPALIVE_SUC　UW 型 SET_TCP_DACK_TMO　　UH 型 SET_TCP_DUP_ACK　　UH 型

戻値	E_OK　　正常終了
解説	デフォルトネットワーク I/F で使用するオプションを設定します。

3.1.7 デフォルトネットワークI/Fに設定されているオプション情報の取得

形式	ER net_get_opt(INT optname, VP optval, INT optlen); optname オプションの種類 optval　オプション値を取得するバッファのポインタ optlen　オプション取得するバッファのサイズ optname には以下が設定できます SET_IF_MTU　　MTU の値を取得します SET_IP_TTL　　TTL の値を取得します SET_IP_MTTL　　マルチキャストパケットの TTL の値を取得します SET_IP_REASM_TMO　　IP フラグメント再構築タイムアウトを取得します SET_TCP_SYN_RCNT　　SYN の再送回数を取得します SET_TCP_DAT_RCNT　　データの再送回数を取得します SET_TCP_RTO_INI　　TCP 再送時間の初期値を取得します SET_TCP_RTO_MIN　　TCP 再送タイムアウトの下位境界を取得します SET_TCP_RTO_MAX　　TCP 再送タイムアウトの上位境界を取得します SET_TCP_KEEPALIVE_TMO　キープアライブタイムアウトを取得します SET_TCP_KEEPALIVE_PRO　キープアライブプローブインターバルを取得します SET_TCP_KEEPALIVE_SUC　キープアライブプローブタイムアウトを取得します SET_TCP_DACK_TMO　　遅延 ACK の遅延時間を設定します SET_TCP_DUP_ACK　　重複 ACK をエラーとして扱う回数を設定します。 各オプションで指定するタイプは次のようになります SET_IF_MTU　　UH 型 SET_IP_TTL　　UB 型 SET_IP_MTTL　　UB 型 SET_IP_REASM_TMO　　UH 型 SET_TCP_SYN_RCNT　　UH 型 SET_TCP_DAT_RCNT　　UH 型 SET_TCP_RTO_INI　　UW 型 SET_TCP_RTO_MIN　　UW 型 SET_TCP_RTO_MAX　　UW 型 SET_TCP_KEEPALIVE_TMO　UW 型 SET_TCP_KEEPALIVE_PRO　UW 型

	SET_TCP_KEEPALIVE_SUC　UW 型 SET_TCP_DACK_TMO　　UH 型 SET_TCP_DUP_ACK　　UH 型
戻値	E_OK　　正常終了
解説	設定されているデフォルトネットワーク I/F のオプション情報を取得します。

3.1.8 デフォルトネットワークI/Fに設定されているアドレスの変更

形式	ER net_chg_ipa(T_NIF_ADDR *addr, UB level); addr　ネットワーク I/F に設定するアドレス情報 level　設定レベル typedef struct t_nif_addr { 　UB *hwaddr;　　/* ハードウェアアドレス(未使用) */ 　UB *ipaddr;　　/* デフォルト IP アドレス */ 　UB *gateway;　/* デフォルトゲートウェイ */ 　UB *mask;　　/* サブネットマスク */ } T_NIF_ADDR;
戻値	E_OK　　正常終了
解説	設定されているデフォルトネットワークのアドレスを変更します。設定レベルを 0 にすると、デフォルトネットワーク I/F で設定されている IP アドレス、デフォルトゲートウェイ、サブネットマスク変更します。設定レベルを 1 にするとさらに通信端点で設定されている IP アドレスも変更します。この関数ではハードウェアアドレスは変更されません。設定レベルを 2 に設定すると、処理中の各 API が E_ADDR(-191) でリターンします。待ち状態になっている処理は起床されます。

例	T_NIF_ADDR new_addr;
	
	UB ipaddr[4]　= { 192, 168, 0, 99 };
	UB gateway[4]　= { 192, 168, 0, 1　};
	UB net_mask[4] = { 255, 255, 255, 0 };
	
	new_addr.ipaddr　= ipaddr;
	new_addr.gateway = gateway;
	new_addr.mask　　= net_mask;
	
	ercd = net_chg_ipa(&new_addr, 0);

3.1.9 ARPテーブルに情報を追加する

形式	ER arp_add_entry(UW ipaddr, UB *macaddr, UW type); ipaddr　登録する IP アドレス macaddr　登録する MAC アドレスが格納されたバッファへのポインタ type　タイプ(ARP_STATIC または ARP_DYNAMIC)
戻値	E_OK　正常終了 E_PAR　MAC アドレスへのポインタが NULL E_OBJ　IP アドレスまたはタイプが不正 E_NOMEM　ARP テーブルが一杯
解説	デフォルトネットワーク I/F で使用されている ARP テーブルに情報を追加します。ARP テーブルはデフォルトで 2 分間、登録アドレスと通信が行われない場合、クリアされます。ARP_STATIC 指定で追加されたアドレスは自動的にクリアが行われません。

3.1.10 ARPテーブルに情報を追加する（ネットワークI/F名指定）

形式	ER arp_add_byname(const char *name, UW ipaddr, UB *macaddr, UW type); name　ネットワークインターフェース名(“eth1”等) ipaddr　登録する IP アドレス macaddr　登録する MAC アドレスが格納されたバッファへのポインタ type　タイプ(ARP_STATIC または ARP_DYNAMIC)
戻値	E_OK　正常終了 E_PAR　MAC アドレスへのポインタが NULL E_OBJ　インターフェース名、IP アドレスまたはタイプが不正 E_NOMEM　ARP テーブルが一杯
解説	指定したネットワーク I/F で使用されている ARP テーブルに情報を追加します。ARP テーブルはデフォルトで 2 分間、そのアドレスと通信が行われない、クリアされます。ARP_STATIC 指定で追加されたアドレスは自動的にクリアが行われません。

3.1.11 ARPテーブルから情報を削除する

形式	ER arp_del_entry(UW ipaddr); ipaddr　削除する IP アドレス
戻値	E_OK　　正常終了 E_OBJ　　IP アドレスが不正か見つからない
解説	デフォルトネットワーク I/F の ARP テーブルから指定した IP アドレスを削除します。

3.1.12 ARPテーブルから情報を削除する（ネットワークI/F名指定）

形式	ER arp_del_byname(const char *name, UW ipaddr); name　　ネットワークインターフェース名(“eth1”等) ipaddr　削除する IP アドレス
戻値	E_OK　　正常終了 E_OBJ　　IP アドレスが不正か見つからない
解説	指定したネットワーク I/F の ARP テーブルから指定した IP アドレスを削除します。

3.2 TCP通信端点毎に変更可能になった設定値

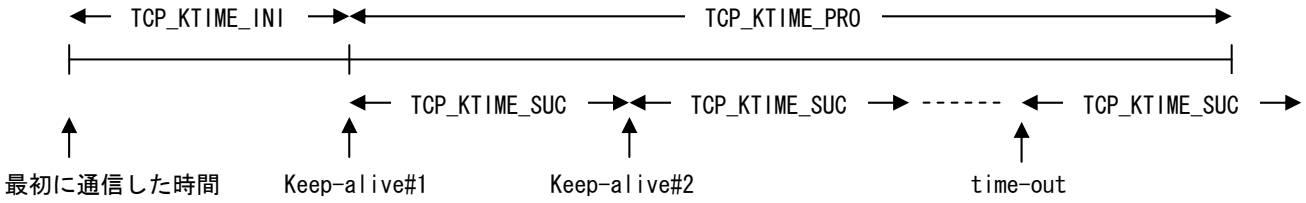
TCP/IP プロトコルスタック Version 4.24 以降、以下の設定値が通信端点毎に変更可能になりました。

TCP_DAT_RCNT	データの最大再送回数
TCP_RTO_INI	TCP 再送タイムアウトの初期値
TCP_RTO_UBOUND	TCP 再送タイムアウトの上位境界
TCP_RTO_LBOUND	TCP 再送タイムアウトの下位境界
TCP_KTIME_INI	キープアライブパケットが送信されるまでの時間
TCP_KTIME_PRO	キープアライブタイムアウト
TCP_KTIME_SUC	キープアライブパケットの送信インターバル

設定値の変更は tcp_set_opt を使用します。TCP 通信端点が接続状態のときにのみ変更可能です。TCP 通信端点が接続状態直後(tcp_acp_cep、tcp_con_cep で接続直後)で、データ通信を始める前に変更してください。それ以外のタイミングでの変更は動作保証外です。

3.2.1 キープアライブの最大送信回数について

キープアライブのパラメータの関係は以下となります。



以上の関係より、キープアライブの最大送信回数は TCP_KTIME_PRO と TCP_KTIME_SUC で設定します。式で表すと以下の関係となります。

キープアライブの最大送信回数 ＝ TCP_KTIME_PRO ／ TCP_KTIME_SUC

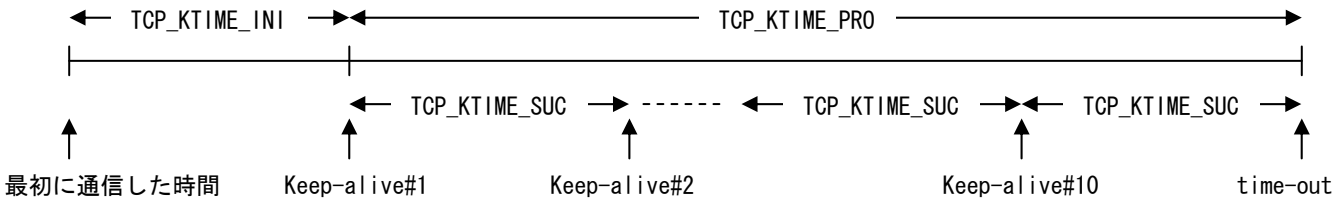
端数は切り捨てとなります。

例)

TCP_KTIME_PRO = 750

TCP_KTIME_SUC = 75

キープアライブの最大送信回数 = 10 = TCP_KTIME_PRO / TCP_KTIME_SUC



この場合、キープアライブの再送は 10 回実施し、11 回目のタイミングでタイムアウトとなります。

3.2.2 キープアライブのタイムアウト通知

TCP コールバック関数でキープアライブのタイムアウトを通知します。タイムアウト通知は、サービスコールの機能コードに TEV_TCP_TMO_KTIME(0x202)が設定されてコールバック関数が呼び出されます。

記述例)

```
const T_TCP_CCEP c_tcp_callback = { 0, ..., ..., ..., ..., (FP)tcp_callback };

ER tcp_callback(ID cepid, FN fncd, VP parblk)
{
    switch (fncd) {
    case TEV_TCP_TMO_KTIME:  /* キープアライブのタイムアウト通知 */
        break;
      :
    default:
        break;
    }

    return E_OK;
}

TASK MainTask(void)
{
     :
    ercd = tcp_ini();
    if (ercd < 0)
        goto END;
     :
    /* TCP 通信端点の生成 */
    ercd = tcp_cre_cep(cepid, c_tcp_callback);
    if (ercd != E_OK)
        goto ERR;
     :
}
```

3.2.3 制限・注意事項

・ タイムアウト時間値、リトライ回数の正当性チェック

タイムアウト時間値は、実装・環境等に依存するためプロトコルスタック側ではタイムアウト時間値の正当性チェックは行っていません。タイムアウト時間の変更はアプリケーション側の責任で行ってください。例えば、極端にタイムアウト時間を短くした場合、再送とタイムアウトを繰り返す無限ループに陥る可能性があります。この場合のタイムアウト時間の正当性チェックなどの責任はアプリケーション側で持つ必要があります。

・ タイムアウト値、リトライ回数の優先順位

TCP 通信端点毎のタイムアウト値、リトライ回数は以下ような動作となります。

a) 接続時には、ネットワーク I/F の値が設定される。その為、TCP 通信端点毎の設定をする場合は接続状態になった直後(ネットワーク I/F の値が設定された後)に設定する必要がある。TCP 通信端点毎の設定をしない場合は、ネットワーク I/F の値が使用される。

b) TCP 通信端点毎の設定値は、接続状態のときにのみ有効で次回接続のために値を保持しない。その為､一旦切断し再度接続する場合は TCP 通信端点毎の設定値を再度設定する必要がある。

c) TCP 通信端点毎のタイムアウト値、リトライ回数の優先順位は以下となる。
TCP 通信端点毎の値 > ネットーワーク I/F の値 > デフォルト値

3.2.4 TCP通信端点オプションの設定

形式	ER tcp_set_opt(ID cepid, INT optname, VP optval, INT optlen); cepid　TCP 通信端点 ID optname　オプションの種類 optval　オプションの値が設定されているバッファのポインタ optlen　オプションの値の長さ optname には以下が設定できます。 SET_TCP_DAT_RCNT　データの再送回数を設定します(単位：回) SET_TCP_RTO_INI　TCP 再送時間の初期値を設定します (単位：TMO(=ミリ秒/MSEC)) SET_TCP_RTO_MIN　TCP 再送タイムアウトの下位境界を設定します (単位：TMO(=ミリ秒/MSEC)) SET_TCP_RTO_MAX　TCP 再送タイムアウトの上位境界を設定します (単位：TMO(=ミリ秒/MSEC)) SET_TCP_KEEPALIVE_TMO　キープアライブタイムアウトを設定します(単位：秒) SET_TCP_KEEPALIVE_PRO　キープアライブプローブインターバルを設定します(単位：秒) SET_TCP_KEEPALIVE_SUC　キープアライブプローブタイムアウトを設定します(単位：秒) 各オプションで指定するタイプは次のようになります SET_TCP_DAT_RCNT　UH 型 SET_TCP_RTO_INI　UW 型 SET_TCP_RTO_MIN　UW 型 SET_TCP_RTO_MAX　UW 型 SET_TCP_KEEPALIVE_TMO　UW 型 SET_TCP_KEEPALIVE_PRO　UW 型 SET_TCP_KEEPALIVE_SUC　UW 型 各オプションとコンフィグレーション値の関係 SET_TCP_DAT_RCNT　TCP_DAT_RCNT SET_TCP_RTO_INI　TCP_RTO_INI SET_TCP_RTO_MIN　TCP_RTO_LBOUND SET_TCP_RTO_MAX　TCP_RTO_UBOUND SET_TCP_KEEPALIVE_TMO　TCP_KTIME_INI SET_TCP_KEEPALIVE_PRO　TCP_KTIME_PRO SET_TCP_KEEPALIVE_SUC　TCP_KTIME_SUC
戻値	E_OK　正常終了 E_ID　不正 ID 番号

	E_NOEXS　通信端点が未生成 E_PAR　パラメータエラー E_OBJ　通信端点が使用中 E_LNK　ネットワークインターフェイスが見つからない E_NOSPT　未サポートのパラメータが指定された
解説	設定値の変更を行います。設定値が変更可能になるのは、TCP 通信端点が通信先と接続後で且つデータ通信を始める前となります。キープアライブタイムアウト値(SET_TCP_KEEPALIVE_TMO)を 0 に設定するとキープアライブタイマーは停止(無効)します。キープアライブ関連のパラメータ(SET_TCP_KEEPALIVE_TMO, SET_TCP_KEEPALIVE_PRO, SET_TCP_KEEPALIVE_SUC)を変更すると、キープアライブ関連の内部変数とキープアライブタイマーは関連パラメータで初期化されます。TCP 再送関連のパラメータ(SET_TCP_DAT_RCNT, SET_TCP_RTO_INI, SET_TCP_RTO_MIN, SET_TCP_RTO_MAX)を変更すると、TCP 再送関連の内部変数は関連パラメータで初期化されます。

3.2.5 TCP通信端点オプションの参照

形式	ER tcp_get_opt(ID cepid, INT optname, VP optval, INT optlen); cepid　TCP 通信端点 ID optname　オプションの種類 optval　オプションの値が設定されているバッファのポインタ optlen　オプション取得するバッファのサイズ optname には以下が設定できます。 SET_TCP_DAT_RCNT　データの再送回数を設定します(単位：回) SET_TCP_RTO_INI　TCP 再送時間の初期値を設定します (単位：TMO(=ミリ秒/MSEC)) SET_TCP_RTO_MIN　TCP 再送タイムアウトの下位境界を設定します (単位：TMO(=ミリ秒/MSEC)) SET_TCP_RTO_MAX　TCP 再送タイムアウトの上位境界を設定します (単位：TMO(=ミリ秒/MSEC)) SET_TCP_KEEPALIVE_TMO　キープアライブタイムアウトを設定します(単位：秒) SET_TCP_KEEPALIVE_PRO　キープアライブプローブインターバルを設定します(単位：秒) SET_TCP_KEEPALIVE_SUC　キープアライブプローブタイムアウトを設定します(単位：秒) 各オプションで指定するタイプは次のようになります SET_TCP_DAT_RCNT　UH 型 SET_TCP_RTO_INI　UW 型 SET_TCP_RTO_MIN　UW 型 SET_TCP_RTO_MAX　UW 型 SET_TCP_KEEPALIVE_TMO　UW 型 SET_TCP_KEEPALIVE_PRO　UW 型

	SET_TCP_KEEPALIVE_SUC　UW 型 各オプションとコンフィグレーション値の関係 SET_TCP_DAT_RCNT　TCP_DAT_RCNT SET_TCP_RTO_INI　TCP_RTO_INI SET_TCP_RTO_MIN　TCP_RTO_LBOUND SET_TCP_RTO_MAX　TCP_RTO_UBOUND SET_TCP_KEEPALIVE_TMO　TCP_KTIME_INI SET_TCP_KEEPALIVE_PRO　TCP_KTIME_PRO SET_TCP_KEEPALIVE_SUC　TCP_KTIME_SUC
戻値	E_OK　正常終了 E_ID　不正 ID 番号 E_NOEXS　通信端点が未生成 E_PAR　パラメータエラー E_OBJ　通信端点が使用中 E_LNK　ネットワークインターフェイスが見つからない E_NOSPT　未サポートのパラメータが指定された
解説	設定値の変更を行います。設定値が変更可能になるのは、TCP 通信端点が通信先と接続後で且つデータ通信を始める前となります。キープアライブタイムアウト値(SET_TCP_KEEPALIVE_TMO)を 0 に設定するとキープアライブタイマーは停止(無効)します。キープアライブ関連のパラメータ(SET_TCP_KEEPALIVE_TMO, SET_TCP_KEEPALIVE_PRO, SET_TCP_KEEPALIVE_SUC)を変更すると、キープアライブ関連の内部変数とキープアライブタイマーは関連パラメータで初期化されます。TCP 再送関連のパラメータ(SET_TCP_DAT_RCNT, SET_TCP_RTO_INI, SET_TCP_RTO_MIN, SET_TCP_RTO_MAX)を変更すると、TCP 再送関連の内部変数は関連パラメータで初期化されます。

3.3 ARP REPLYのIPアドレスの重複検知

TCP/IP プロトコルスタック Version 4.24 以降、IP アドレスの重複検知を行い arp reply を返信するかどうかを ARP のコールバック関数で選択できるようになりました。

3.3.1 プログラム作成方法

コールバックは、自局宛て以外の ARP パケットと IP アドレスの重複検知で呼ばれます。また、そのパケットへの応答の判断は、コールバックの戻り値で判別されます。

NULL を返す　.....　そのパケットは、アプリケーションで処理されることを指定します。この時、pkt に示すバッファはアプリケーション側で返却するようにしてください。返却用 API は、arp_rel_pkt(pkt)です。

入力変数 pkt を　...　アプリケーションで処理を行わず、NORTi 実装に任せることを指定します。
そのまま返す

補足）重複検知した場合の戻り値の意味付け

・arp reply を返却する場合　　→　pkt のポインタ値をそのまま返す

・arp reply を返信しない場合　→　NULL を返す

NULL を返す場合は、返却用 API(arp_rel_pkt(pkt))を呼び出して、必ず受信パケットバッファを解放してください。受信パケットを解放しない場合、メモリープールが枯渇する可能性があります。

記述例）

```
T_ARP_CB arp_cb;

VP arp_callback(T_ARP *pkt)
{
    UW ipaddr;
    T_NIF *nif;

    nif = pkt->ctl.nif;
    ipaddr = byte4_to_long(pkt->spa);

    /* 送信元 IP アドレスと自 IP をチェックします */
    if (is_my_ipaddr(nif, ipaddr))  {
        arp_rel_pkt(pkt); /* パケットを廃棄する API を呼ぶ */
        return NULL;      /* arp reply を返信しない */
    }

    /* NORTi 仕様に任せる(arp reply を返信する) */
    return pkt;
}
```

```
TASK MainTask(void)
{
    :
    ercd = tcp_ini();
    if (ercd < 0)
        goto END;

    /* ARP コールバック関数を定義します。*/
    ercd = arp_def_cbk(&arp_cb, arp_callback);
    :
```

3.3.2 受信パケットを解放するAPI

形式	ER arp_rel_pkt(T_ARP *pkt); pkt　　ARP 受信パケットへのポインタ
戻値	E_OK　　正常終了 E_PAR,E_ID,E_NOEXS　　パケットが不正
解説	ARP のコールバックで渡されたパケットを解放します。ARP のコールバックで NULL を返す場合は、本 API を使用しパケットを解放してください。