TCP/IP プロトコルスタック（TINET）
ユーザズマニュアル（リリース 1.5.3）［2015/1/19］

1. TCP/IP プロトコルスタック（TINET）の概要

TINET は、TOPPERS/ASP と TOPPERS/JSP 用の TCP/IP プロトコルスタックである。

1. 1 機能一覧

以下に、TINET リリース 1.5 の機能一覧を示す。

(1) API

・ITRON TCP/IP API 仕様の標準機能
・暫定的な ITRON TCP/IP（バージョン 6）API 仕様の標準機能
・ITRON TCP/IP API 仕様の拡張機能

(2) TCP

・BSD の通信機能
・最大セグメントサイズ（MSS）オプション
・省コピー API
・ノンブロッキングコール（組込み選択可）
・タスクからの Time Wait 状態の TCP 通信端点分離機能（組込み選択可）
・送受信ウィンドバッファの省コピー機能（組込み選択可）
・TCP ヘッダのトレース出力機能（組込み選択可）

(3) UDP

・ノンブロッキングコール（組込み選択可）

(4) 近隣探索

・近隣探索要請の送受信
・近隣探索通知の送受信
・ルータ通知メッセージの受信
・ルータ要請メッセージの送信
・アドレス重複検出機能

(5) ICMPv4

・エコー要求・応答の送受信
・エラーの送信（組込み選択可）
・向け直しメッセージの受信（組込み選択可）

(6) ICMPv6

・エコー要求・応答の送受信
・エラーの送信（組込み選択可）
・向け直しメッセージの受信（組込み選択可）
・Path MTU

(7) IPv4

・静的経路表
・IP データグラムの分割・再構成（組込み選択可）
・IPSEC（組込み選択可、フックのみ実装）【リリース 1.5 新規】

(8) IPv6

・アドレスの自動設定
・静的経路表
・非 PC 系ディジタル機器への適用に向けた IPv6 最小要求仕様の IPv6 最小ホスト仕様に準拠
・拡張ヘッダのエラーの通知
・断片ヘッダ（組込み選択可）
・ホスト情報キャッシュ（組込み選択可）

(9) その他

・ARP 要求・応答の送受信
・ARP での IPv4 アドレス重複検出機能
・DHCP への対応
・SNMP 用管理情報ベース（MIB）の提供

1. 2 動作確認済みのシステム

1. 2. 1 開発環境

開発環境は、Windows 上の cygwin 上で、GCC を用いて開発を行った。GCC 等のバージョンを以下に示す。

(1) GCC 2.x.x 系

binutils-2.11.2
gcc-2.95.3
newlib-1.9.0

なお、cygwin の make のバージョン 3.81 では、コンパイルが正常に行えないため、これ以外のバージョンの make を使用すること。また、参考であるが、TOPPERS/ASP では、以下のバージョンでも動作を確認した。

(1) GCC 3.x.x 系

binutils-2.15
gcc-3.4.3
newlib-1.12.0

(2) GCC 4.x.x 系

binutils-2.23.2
gcc-4.7.3
newlib-1.20.0

1. 2. 2 動作確認済みのシステム（TOPPERS/ASP）

現在、動作を確認済みのシステムを以下に示す。

(1) 秋月電子通商製 H8/3069F ネット対応（イーサネット、ASP リリース 1.3 ～ 1.7）

また、動作を確認していないが参考実装のシステムを以下に示す。

(1) アルファプロジェクト製の AP-SH4-1A（イーサネット、ASP リリース 1.8）

1. 2. 3 動作確認済みのシステム（TOPPERS/JSP）

現在、動作を確認済みのシステムを以下に示す。

(1) 品川通信計装サービス製 NKEV-010H8（イーサネット、JSP リリース 1.4.2）
(2) 秋月電子通商製 H8/3068F ネット対応（イーサネット、JSP リリース 1.4.1）
(3) 秋月電子通商製 H8/3069F ネット対応（イーサネット、JSP リリース 1.4.1 ～ 1.4.4）

また、動作を確認していないが参考実装のシステムを以下に示す。

(1) 秋月電子通商製 H8/3048F（PPP、ループバック、JSP リリース 1.4.1）

1. 3 サポートしているネットワークインタフェース

現在、サポートしているネットワークインタフェースを以下に示す。

(1) イーサネット、NIC は NE2000 互換。

参考実装のネットワークインタフェースを以下に示す。

(1) ループバック。
(2) シリアルポートを用いた PPP で、直接接続とモデム接続。

なお、TINET は、上記のネットワークインタフェースを一種類のみ組み込むことができ、ユニット数も 1 である。

1. 4 制限事項

以下に、TINET リリース 1.5 の制約事項を述べる。

(1) 対応する TOPPERS/ASP カーネルはリリース 1.3.2 以降である。
(2) 対応する TOPPERS/JSP カーネルはリリース 1.4.1 以降である。
(3) ネットワーク層は IPv4 と IPv6 のどちらか一つのみ選択でき、同時に組込むことはできない。
(4) IPv6 では、ネットワークインタフェースはイーサネットのみ選択できる。
(5) IPv6 では、TCP と UDP の両方を選択するか、どちらか一つを選択しなければならない。また、UDP のみを選択した場合は、ノンブロッキング機能を組み込む必要がある。
(6) ノンブロッキングコールにおいても、通信端点の排他制御のため、短時間であるがブロックすることがある。
(7) IPv6 に関する ITRON TCP/IP API 仕様がないため、暫定的な ITRON TCP/IP（バージョン 6）API 仕様を定義した。
(8) ITRON TCP/IP API 部分のライブラリ化は行われているが、ライブラリとアプリケーションプログラムを別々に構築しておき、後でリンクする方法はサポートしていない。

(9) 設定と読出し可能な TCP 通信端点オプションは無いため、TCP 通信端点オプションの設定 API と読出し API の戻り値は E_PAR である。

(10) 設定と読出し可能な UDP 通信端点オプションは無いため、UDP 通信端点オプションの設定 API と読出し API の戻り値は E_PAR である。

1. 5 ディレクトリ構成

TINET のディレクトリは、JSP ルートディレクトリの下に置くことを想定しており、以下のディレクトリから構成されている。

`tinet`	TINET のルートディレクトリ
`tinet/cfg`	TINET コンフィギュレータ（TOPPERS/JSP 用のみ）
`tinet/doc`	ドキュメント類
`tinet/net`	汎用ネットワーク
`tinet/netapp`	サンプルのネットワークプログラム
`tinet/netdev`	ネットワークインタフェースのドライバ
`tinet/netdev/if_ed`	NE2000 互換ネットワークインタフェースのドライバ
`tinet/netinet`	TINET の本体
`tinet/netinet6`	IPv6 の本体

1. 6 ドキュメント類

ドキュメント類を以下に示す。全てのファイルは PDF でも提供している。

`tinet.txt`	ユーザズマニュアル
`tinet_config.txt`	コンパイル時コンフィギュレーション
`tinet_defs.txt`	プロセッサ、システム依存定義
`tinet_chg.txt`	変更メモ
`tinet-1.5.txt`	リリース 1.4 から 1.5 への移行
`tinet_ether.pdf`	TINET-1.4 におけるイーサネットの実装（PDF のみ）

2. TINET コンフィギュレータと TINET コンフィグレーションファイル

2. 1 TOPPERS/ASP 環境における TINET コンフィグレーションファイル

TOPPERS/ASP 環境では、TOPPERS/ASP 用コンフィギュレータを流用するため、TINET 独自のコンフィギュレータはない。

TOPPERS/ASP 用コンフィギュレータを流用して、TINET コンフィギュレーションファイル（標準は `tinet_$(APPLNAME).cfg`）から以下のファイルを生成する。

(1) `tinet_cfg.c`
TINET カーネル構成ファイルで、アプリケーションプログラム、TINET と共にコンパイルしてリンクする。

(2) `tinet_kern.cfg`
TINET 内部で使用するカーネルオブジェクトの静的 API が生成され、TOPPERS/ASP システムコンフィギュレーションファイル（標準は `$(APPLNAME).cfg`）にインクルードする。

(3) tinet_cfg.h
TINET 内部で使用するカーネルオブジェクトの ID 自動割付結果ファイルである。

2. 2 TOPPERS/JSP 環境における TINET コンフィグレーションファイル

TINET コンフィギュレータは tinet/cfg/tinet_cfg (cygwin では tinet_cfg.exe) であり、ターゲットには依存していない。TINET コンフィグレータの生成については「**8. TOPPERS/JSP 環境におけるインストールとファイルの作成・変更**」を参照すること。

TOPPERS/JSP 用 TINET コンフィギュレータは TINET コンフィギュレーションファイル（標準は tinet_$(UNAME).cfg）から以下のファイルを生成する。

(1) tinet_cfg.c
TINET カーネル構成ファイルで、アプリケーションプログラム、TINET と共にコンパイルしてリンクする。

(2) tinet_kern.cfg
TINET 内部で使用するカーネルオブジェクトの静的 API が生成され、TOPPERS/JSP システムコンフィギュレーションファイル（標準は $(UNAME).cfg）にインクルードする。

(3) tinet_id.h
TINET 内部で使用するカーネルオブジェクトの ID 自動割付結果ファイルである。

2. 3 暫定的なITRON TCP/IP（バージョン6）API仕様

IPv6 に関する ITRON TCP/IP API 仕様がないため、暫定的な ITRON TCP/IP（バージョン 6）API 仕様を定義した。

(1) 1.5.1 データ構造／データ型 (1) IP アドレス／ポート番号を入れるデータ構造

```
/* IPv6 アドレス */
struct t_in6_addr {
    union {
        uint8_t    __u6_addr8[16];
        uint16_t   __u6_addr16[8];
        uint32_t   __u6_addr32[4];
        } __u6_addr;
    } T_IN6_ADDR;

typedef struct t_ipv6ep {
    T_IN6_ADDR    ipaddr;        /* IPv6 アドレス */
    uint16_t           portno;        /* ポート番号    */
    } T_IPV6EP;
```

(2) 1.5.1 データ構造／データ型 (2) オブジェクト生成用のデータ構造

```
typedef struct t_tcp6_crep {
    /* 標準 */
    ATR         repatr;        /* 受付口属性        */
    T_IPV6EP    myaddr;        /* 自分のアドレス    */
    /* 実装依存 */
    } T_TCP6_CREP;
```

(3) 1.5.1 データ構造／データ型 (5) 特殊な IP アドレスとポート番号

```
#define IPV6_ADDR_UNSPECIFIED_INIT    \
    {{{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }}}
#define IPV6_ADDRANY        IPV6_ADDR_UNSPECIFIED_INIT
```

(4) 2.2 TCP 受付口の生成／削除

【静的 API】

```
TCP6_CRE_REP(ID repid, { ATR repatr,
                         { T_IN6_ADDR myipaddr, uint16_t myportno } } );
```

【API の機能】

myipaddr の型が T_IN6_ADDR になった以外は、ITRON TCP/IP API 仕様と同じである。myipaddr には IPV6_ADDRANY を指定できる。

(5) 2.4 接続／切断「接続要求待ち（受動オープン）」

【C 言語 API】

```
ER ercd = tcp6_acp_cep(ID cepid, ID repid,
                       T_IPV6EP *p_dstaddr, TMO tmout);
```

【API の機能】

p_dstaddr の型が、T_IPV6EP* になった以外は、ITRON TCP/IP API 仕様と同じである。

(6) 2.4 接続／切断「接続要求（能動オープン）」

【C 言語 API】

```
ER ercd = tcp6_con_cep(ID cepid, T_IPV6EP *p_myaddr,
                       T_IPV6EP *p_dstaddr, TMO tmout);
```

【機能】

p_myaddr と p_dstaddr の型が、T_IPV6EP* になった以外は、ITRON TCP/IP API 仕様と同じである。

(7) 3.2 UDP 通信端点の生成／削除

【静的 API】

```
UDP6_CRE_CEP(ID cepid, { ATR cepatr,
                         { T_IN6_ADDR myipaddr, uint16_t myportno },
                           FP callback } );
```

【機能】

myipaddr の型が T_IN6_ADDR になった以外は、ITRON TCP/IP API 仕様と同じである。myipaddr には IPV6_ADDRANY を指定できる。

(8) 3.3 データの送受信「パケットの送信」

【C 言語 API】

```
ER ercd = udp6_snd_dat(ID cepid, T_IPV6EP *p_dstaddr,
                       void *data, int_t len, TMO tmout);
```

【機能】

p_dstaddr の型が、T_IPV6EP* になった以外は、ITRON TCP/IP API 仕様と同じである。

(9) 3.4 データの送受信「パケットの受信」

【C 言語 API】

```
ER ercd = udp6_rcv_dat(ID cepid, T_IPV6EP *p_dstaddr,
                       void *data, int_t len, TMO tmout);
```

【機能】

p_dstaddr の型が、T_IPV6EP* になった以外は、ITRON TCP/IP API 仕様と同じである。

2. 4 サポートするオブジェクトの定義

サポートするオブジェクトの定義は、以下に示す ITRON TCP/IP API 仕様の静的 API、暫定的な ITRON TCP/IP（バージョン 6）API 仕様の静的 API、TINET 独自の静的 API、ファイルのインクルードである。

(1) TCP 受付口（IPv4）

【静的 API】

```
TCP_CRE_REP(ID repid, { ATR repatr,
                        { uint32_t myipaddr, uint16_t myportno } } );
```

【パラメータ】

パラメータについては、ITRON TCP/IP API 仕様と同じであり、実装依存の TCP 受付口属性はない。

【TCP 受付口数の定義】

TCP 受付口数を定義するプリプロセッサディレクティブであり、tinet_cfg.c に出力される。

```
#define TNUM_TCP_REPID <TCP受付口数>
```

【TCP 受付口 ID の最大値の変数の定義】

TCP 受付口 ID の最大値の変数の定義であり、tinet_cfg.c に出力される。

```
const ID tmax_tcp_repid =
          (TMIN_TCP_REPID + TNUM_TCP_REPID - 1);
```

(2) TCP 通信端点（IPv4）

【静的 API】

```
TCP_CRE_CEP(ID cepid, { ATR cepatr, void *sbuf, int_t sbufsz,
                                    void *rbuf, int_t rbufsz,
                                    FP callback } );
```

【パラメータ】

パラメータについては、ITRON TCP/IP API 仕様と同じであり、実装依存の TCP 通信端点属性はない。

【TCP 通信端点数の定義】

TCP 通信端点数を定義するプリプロセッサディレクティブであり、tinet_cfg.c に出力される。

```
#define TNUM_TCP_CEPID <TCP通信端点数>
```

【TCP 通信端点 ID の最大値の変数の定義】

TCP 通信端点 ID の最大値の変数の定義であり、tinet_cfg.c に出力される。

```
const ID tmax_tcp_cepid =
          (TMIN_TCP_CEPID + TNUM_TCP_CEPID - 1);
```

(3) UDP 通信端点（IPv4）

【静的 API】

```
UDP_CRE_CEP(ID cepid, { ATR cepatr,
                        { uint32_t myipaddr, uint16_t myportno },
                          FP callback } );
```

【パラメータ】

パラメータについては、ITRON UDP/IP API 仕様と同じであり、実装依存の UDP 通信端点属性はない。

【UDP 通信端点数の定義】

UDP 通信端点数を定義するプリプロセッサディレクティブであり、tinet_cfg.c に出力される。

```
#define TNUM_UDP_CEPID <UDP通信端点数>
```

【UDP 通信端点 ID の最大値の変数の定義】

UDP 通信端点 ID の最大値の変数を定義であり、tinet_cfg.c に出力される。

```
const ID tmax_udp_cepid =
          (TMIN_UDP_CEPID + TNUM_UDP_CEPID - 1);
```

(4) TCP 受付口（IPv6）

【静的 API】

```
TCP6_CRE_REP(ID repid, { ATR repatr,
                         { T_IN6_ADDR myipaddr, uint16_t myportno } } );
```

【パラメータ】

パラメータについては、myipaddr で指定する IP アドレスは IPv6 であり、IPv4 の IP_ADDRANY の代わりに、IPv6 では IPV6_ADDRANY を指定できる。これ以外は、ITRON TCP/IP API 仕様と同じであり、実装依存の TCP 受付口属性はない。

【TCP 受付口数の定義】

TCP 受付口数を定義するプリプロセッサディレクティブであり、tinet_cfg.c に出力される。

```
#define TNUM_TCP_REPID <TCP受付口数>
```

【TCP 受付口 ID の最大値の変数の定義】

TCP 受付口 ID の最大値の変数を定義であり、tinet_cfg.c に出力される。

```
const ID tmax_tcp_repid =
          (TMIN_TCP_REPID + TNUM_TCP_REPID - 1);
```

(5) TCP 通信端点（IPv6）

【静的 API】

```
TCP6_CRE_CEP(ID cepid, { ATR cepatr, void *sbuf, int_t sbufsz,
                                      void *rbuf, int_t rbufsz,
                                      FP callback } );
```

【パラメータ】

パラメータについては、ITRON TCP/IP API 仕様と同じであり、実装依存の TCP 通信端点属性はない。

【TCP 通信端点数の定義】

TCP 通信端点数を定義するプリプロセッサディレクティブであり、tinet_cfg.c に出力される。

```
#define TNUM_TCP_CEPID <TCP通信端点数>
```

【TCP 通信端点 ID の最大値の変数の定義】

最大の TCP 通信端点 ID の最大値の変数を定義であり、tinet_cfg.c に出力される。

```
const ID tmax_tcp_cepid =
          (TMIN_TCP_CEPID + TNUM_TCP_CEPID - 1);
```

(6) UDP 通信端点（IPv6）

【静的 API】

```
UDP6_CRE_CEP(ID cepid, { ATR cepatr,
                         { T_IN6_ADDR myipaddr, uint16_t myportno },
                           FP callback } );
```

【パラメータ】

パラメータについては、myipaddr で指定する IP アドレスは IPv6 であり、IPv4 の IP_ADDRANY の代わりに、IPv6 では IPV6_ADDRANY を指定できる。これ以外は、ITRON TCP/IP API 仕様と同じであり、実装依存の TCP 受付口属性はない。

【UDP 通信端点数の定義】

UDP 通信端点数を定義するプリプロセッサディレクティブであり、tinet_cfg.c に出力される。

```
#define TNUM_UDP_CEPID <UDP通信端点数>
```

【UDP 通信端点 ID の最大値の変数の定義】

UDP 通信端点 ID の最大値の変数を定義であり、tinet_cfg.c に出力される。

```
const ID tmax_udp_cepid =
          (TMIN_UDP_CEPID + TNUM_UDP_CEPID - 1);
```

(7) TCP 受付口の予約 ID（IPv4、TINET 独自）

【静的 API】

```
VRID_TCP_REP(ID repid);
```

【パラメータ】

```
ID          repid       予約するTCP受付口ID
```

(8) TCP 通信端点の予約 ID（IPv4、TINET 独自）

【静的 API】

```
VRID_TCP_CEP(ID cepid);
```

【パラメータ】

```
ID          repid       予約するTCP通信端点ID
```

(9) UDP 通信端点の予約 ID（IPv4、TINET 独自）

【静的 API】

```
VRID_UDP_CEP(ID cepid);
```

【パラメータ】

```
ID          repid       予約するUDP通信端点ID
```

(10) TCP 受付口の予約 ID（IPv6、TINET 独自）

【静的 API】

```
VRID_TCP6_REP(ID repid);
```

【パラメータ】

ID	repid	予約するTCP受付口ID

(11) TCP 通信端点の予約 ID（IPv6、TINET 独自）

【静的 API】

```
VRID_TCP6_CEP(ID cepid);
```

【パラメータ】

ID	repid	予約するTCP通信端点ID

(12) UDP 通信端点の予約 ID（IPv6、TINET 独自）

【静的 API】

```
VRID_UDP6_CEP(ID cepid);
```

【パラメータ】

ID	repid	予約するUDP通信端点ID

3. ITRON TCP/IP API 拡張機能

TINET リリース 1.3 までは、ITRON TCP/IP API の標準機能のみに対応していたが、リリース 1.4 からは、拡張機能にも対応した。ただし、応用プログラムから使用する場合は、以下に示すコンパイル時コンフィギュレーションパラメータを指定しなければならない。

(1) TCP_CFG_EXTENTIONS
ITRON TCP/IP API の TCP の拡張機能を有効にする。

(2) UDP_CFG_EXTENTIONS
ITRON TCP/IP API の UDP の拡張機能を有効にする。

3. 1 TCP の ITRON TCP/IP API 拡張機能

TCP_CFG_EXTENTIONS を指定することにより使用可能となる API を以下に示す。

・TCP 受付口の予約 ID【静的 API、VRID_TCP_REP】（IPv4、TINET 独自）
・TCP 受付口の予約 ID【静的 API、VRID_TCP6_REP】（IPv6、TINET 独自）
・TCP 通信端点の予約 ID【静的 API、VRID_TCP_CEP】（IPv4、TINET 独自）
・TCP 通信端点の予約 ID【静的 API、VRID_TCP6_CEP】（IPv6、TINET 独自）
・TCP 受付口の生成【動的 API、tcp_cre_rep】（IPv4）
・TCP 受付口の生成【動的 API、tcp6_cre_rep】（IPv6、TINET 独自）
・TCP 受付口の削除【動的 API、tcp_del_rep】
・TCP 通信端点の生成【動的 API、tcp_cre_cep】
・TCP 通信端点の削除【動的 API、tcp_del_cep】

・緊急データの送信【tcp_snd_oob】
・緊急データの受信【tcp_rcv_oob】
・TCP 通信端点オプションの設定【tcp_set_opt】
・TCP 通信端点オプションの読出し【tcp_get_opt】
・緊急データ受信【コールバック、TEV_TCP_RCV_OOB】

(1) TCP 受付口の生成と削除

この機能により、1 個の TCP 受付口を複数のタスクで共有することができる。ただし、1 回に使用できるのは 1 個のタスクに限定される。以下に標準的な使用方法を述べる。なお、煩雑になるため IPv6 に関しての説明は、一部省略している。

[1] TCP 受付口の予約 ID【静的 API、VRID_TCP_REP、VRID_TCP6_REP】により、TCP 受付口 ID を予約する。

VRID_TCP_REP の書式を以下に示す。

```
VRID_TCP_REP(ID repid);
```

パラメータ repid は予約する TCP 受付口 ID であり、一般的には、TINET コンフィグレーションファイルに以下のように指定する。

```
VRID_TCP_REP (TCP_RSV_REPID1);
```

これにより、TCP 受付口用のメモリ領域が確保され、TINET 内部で使用するカーネルオブジェクトの ID 自動割付結果ファイル（TOPPERS/ASP は tinet_cfg.h、TOPPERS/JSP は tinet_id.h）に、対応するマクロ定義が以下のように出力される。

```
#define TCP_RSV_REPID1  1
```

[2] TCP 受付口の生成【動的 API、tcp_cre_rep、tcp6_cre_rep】により、TCP 受付口を生成する。

まず、TCP 受付口生成情報構造体に情報を設定する。IPv4 の場合の例を以下に示す。

```
T_TCP_CREP crep;
crep.repatr = 0;
crep.myaddr.portno = 7;
crep.myaddr.ipaddr = IPV4_ADDRANY;
```

また IPv6 の場合の例を以下に示す。

```
T_TCP6_CREP crep;
crep.repatr = 0;
crep.myaddr.portno = 7;
memcpy(&crep.myaddr.ipaddr, &ipv6_addrany, sizeof(T_IN6_ADDR));
```

いずれも、受付ける自分の IP アドレスは規定値（全て）である。

次に、tcp_cre_rep の書式を示す。

```
ER ercd = tcp_cre_rep(ID repid, T_TCP_CREP *pk_crep);
```

パラメータ repid には [1] で予約した TCP 受付口 ID を指定し、pk_crep には上記で設定済みの TCP 受付口生成情報へのポインタを指定する。一般的な例を以下に示す。

```
ercd = tcp_cre_rep(TCP_RSV_REPID1, &crep);
```

これにより、VRID_TCP_REP で確保された TCP 受付口用のメモリ領域に TCP 受付口生成情報が書込まれる。

[3] 接続要求待ち（受動オープン）【tcp_acp_cep、tcp6_acp_cep】により、接続要求待ち（受動オープン）する。

tcp_acp_cep の書式を示す。

```
ER ercd = tcp_acp_cep(ID cepid, ID repid,
                      T_IPV4EP *p_dstaddr, TMO tmout);
```

パラメータ repid に [1] で予約した TCP 受付口 ID を指定する以外は、通常の呼び出しと同じである。

[4] TCP 受付口の削除【動的 API、tcp_del_rep】により、TCP 受付口を削除する。

通常は、接続要求待ち（受動オープン）が終了した後に、TCP 受付口を削除するが、接続要求待ち（受動オープン）中に、tcp_del_cep により、TCP 通信端点を削除することも可能である。この場合、tcp_acp_cep の戻り値には、E_DLT が返される。TCP 受付口を削除すると、他のタスクが同じ TCP 受付口 ID を利用できる。tcp_del_rep の書式を示す。

```
ER ercd = tcp_del_rep(ID cepid);
```

パラメータ repid には [1] で予約した TCP 受付口 ID を指定する。

(2) TCP 通信端点の生成と削除

この機能により、1 個の TCP 通信端点を複数のタスクで共有することができる。ただし、1 回に使用できるのは 1 個のタスクに限定される。以下に標準的な使用方法を述べる。なお、煩雑になるため IPv6 に関しての説明は、一部省略している。

[1] TCP 通信端点の予約 ID【静的 API、VRID_TCP_CEP、VRID_TCP6_CEP】により、TCP 通信端点 ID を予約する。

VRID_TCP_CEP の書式を以下に示す。

```
VRID_TCP_CEP(ID cepid);
```

パラメータ cepid は予約する TCP 通信端点 ID であり、一般的には、TINET コンフィグレーションファイルに以下のように指定する。

```
VRID_TCP_CEP (TCP_RSV_CEPID1);
```

これにより、TCP 通信端点用のメモリ領域が確保され、TINET 内部で使用するカーネルオブジェクトの ID 自動割付結果ファイル（TOPPERS/ASP は tinet_cfg.h、TOPPERS/JSP は tinet_id.h）に、対応するマクロ定義が以下のように出力される。

```
#define TCP_RSV_CEPID1  1
```

[2] TCP 通信端点の生成【動的 API、tcp_cre_cep】により、TCP 通信端点を生成する。

まず、TCP 通信端点生成情報構造体に情報を設定する。一般的な例を以下に示す。

```
T_TCP_CCEP ccep;
ccep.cepatr = 0;
ccep.sbufsz = TCP_ECHO_SRV_SWBUF_SIZE;
ccep.rbufsz = TCP_ECHO_SRV_RWBUF_SIZE;
ccep.sbuf = tcp_echo_srv_swbuf;
ccep.rbuf = tcp_echo_srv_rwbuf;
ccep.callback = (FP)callback_nblk_tcp_echo_srv;
```

次に、tcp_cre_cep の書式を示す。

```
ER ercd = tcp_cre_cep(ID cepid, T_TCP_CCEP *pk_ccep);
```

パラメータ cepid には [1] で予約した TCP 通信端点 ID を指定し、pk_ccep には上記で設定済みの TCP 通信端点生成情報へのポインタを指定する。一般的な例を以下に示す。

```
ercd = tcp_cre_cep(TCP_RSV_CEPID1, &ccep);
```

これにより、VRID_TCP_CEP で確保された TCP 通信端点用のメモリ領域に TCP 通信端点生成情報が書込まれる。

この後、TCP の各 API のパラメータ cepid に [1] で予約した TCP 通信端点 ID を指定する以外は、通常の TCP 通信端点と同じように使用できる。

[3] TCP 通信端点の削除【動的 API、tcp_del_cep】により、TCP 通信端点を削除する。

tcp_cls_cep を呼び出すまでは、TCP 通信端点を削除できないが、tcp_cls_cep の後は TCP 通信端点を削除でき、他のタスクが同じ TCP 通信端点 ID を利用できる。tcp_del_cep の書式を示す。

```
ER ercd = tcp_del_cep(ID cepid);
```

パラメータ cepid には [1] で予約した TCP 通信端点 ID を指定する。

(3) 緊急データの送受信

[1] 緊急データの送信【tcp_snd_oob】

tcp_snd_oob の書式を以下に示す。

```
ER_UINT ercd = tcp_snd_oob(ID cepid, void *data, int_t len, TMO tmout);
```

なお、以下に示すような制約がある。

・緊急データだからといって、すでに送信ウィンドバッファにある通常のデータより先に送信されるわけではない。
・tcp_snd_oob で、複数バイトのデータを送信しても（len > 1）、受信側で受信できるのは、送信した data の最後の 1 バイトのみである。また、これより前のデータは通常のデータとして受信される。

[2] 緊急データの受信【tcp_rcv_oob】

tcp_rcv_oob の書式を以下に示す。

```
ER_UINT     ercd = tcp_rcv_oob(ID cepid, void *data, int_t len);
```

なお、以下に示すような制約がある。

・緊急データ受信のコールバック関数内で呼び出すことを想定している。
・受信できるのは、緊急データの最後の 1 バイトのみである。従って、正常に tcp_rcv_oob から戻ってきた時の戻り値は、常に 1 である。

[3] 緊急データ受信【コールバック、TEV_TCP_RCV_OOB】

緊急データを受信した時、TCP 通信端点に指定されているコールバック関数を呼び出す。この時の事象の種類が TEV_TCP_RCV_OOB である。ただし、TCP 通信端点にコールバック関数が指定されていない場合、または、コールバック関数内で tcp_rcv_oob が呼び出されなければ、受信した緊急データは通常のデータとして受信する。

[4] コンパイル時コンフィギュレーションパラメータ

```
TCP_CFG_URG_OFFSET
```

緊急データの最後のバイトのオフセット、値が -1 の場合は BSD の実装と同じで、緊急ポインタは、緊急データの最後のバイトの次のバイトを差す。値が 0 の場合は RFC1122 の規定と同じで、緊急ポインタは、緊急データの最後のバイトを差す。既定値は -1 である。

(4) TCP 通信端点オプションの設定と読出し

設定可能な TCP 通信端点オプションは無いため、どちらの関数も戻り値として E_PAR が返される。

3. 2 UDP の ITRON TCP/IP API 拡張機能

UDP_CFG_EXTENTIONS を指定することにより使用可能となる API を以下に示す。

・UDP 通信端点の予約 ID【静的 API、VRID_UDP_CEP】（IPv4、TINET 独自）
・UDP 通信端点の予約 ID【静的 API、VRID_UDP6_CEP】（IPv6、TINET 独自）
・UDP 通信端点の生成【動的 API、udp_cre_cep】（IPv4）
・UDP 通信端点の生成【動的 API、udp6_cre_cep】（IPv6、TINET 独自）
・UDP 通信端点の削除【動的 API、udp_del_cep】
・UDP 通信端点オプションの設定【udp_set_opt】
・UDP 通信端点オプションの読出し【udp_get_opt】

(1) UDP 通信端点の生成と削除

この機能により、1 個の UDP 通信端点を複数のタスクで共有することができる。ただし、1 回に使用できるのは 1 個のタスクに限定される。以下に標準的な使用方法を述べる。なお、煩雑になるため IPv6 に関しての説明は、一部省略している。

[1] UDP 通信端点の予約 ID【静的 API、VRID_UDP_CEP、VRID_UDP6_CEP】により、UDP 通信端点 ID を予約する。

VRID_UDP_CEP の書式を以下に示す。

```
VRID_UDP_CEP(ID cepid);
```

パラメータ cepid は予約する UDP 通信端点 ID であり、一般的には、TINET コンフィグレーションファイルに以下のように指定する。

```
VRID_UDP_CEP (UDP_RSV_CEPID1);
```

これにより、UDP 通信端点用のメモリ領域が確保され、TINET 内部で使用するカーネルオブジェクトの ID 自動割付結果ファイル（TOPPERS/ASP は tinet_cfg.h、TOPPERS/JSP は tinet_id.h）に、対応するマクロ定義が以下のように出力される。

```
#define UDP_RSV_CEPID1  1
```

[2] UDP 通信端点の生成【動的 API、udp_cre_cep】により、UDP 通信端点を生成する。

まず、UDP 通信端点生成情報構造体に情報を設定する。通信相手からのデータの受信を待つ応用アプリケーションで、IPv4 の場合の例を以下に示す。

```
T_UDP_CCEP ccep;
ccep.cepatr = 0;
ccep.myaddr.portno = 7;
ccep.myaddr.ipaddr = IPV4_ADDRANY;
```

また IPv6 の場合の例を以下に示す。

```
T_UDP_CCEP ccep;
ccep.cepatr = 0;
ccep.myaddr.portno = 7;
memcpy(&ccep.myaddr.ipaddr, &ipv6_addrany, sizeof(T_IN6_ADDR));
```

いずれも、受付ける自分の IP アドレスは規定値（全て）である。

次に、udp_cre_cep の書式を示す。

```
ER ercd = udp_cre_cep(ID cepid, T_UDP_CCEP *pk_ccep);
```

パラメータ cepid には [1] で予約した UDP 通信端点 ID を指定し、pk_ccep には上記で設定済みの UDP 通信端点生成情報へのポインタを指定する。一般的な例を以下に示す。

```
ercd = udp_cre_cep(UDP_RSV_CEPID1, &ccep);
```

これにより、VRID_UDP_CEP で確保された UDP 通信端点用のメモリ領域に UDP 通信端点生成情報が書込まれる。

この後、UDP の各 API のパラメータ cepid に [1] で予約した UDP 通信端点 ID を指定する以外は、通常の UDP 通信端点と同じように使用できる。

[3] UDP 通信端点の削除【動的 API、udp_del_cep】により、UDP 通信端点を削除する。

UDP 通信端点はいつでも削除でき、他のタスクが同じ UDP 通信端点 ID を利用できる。なお、udp_snd_dat で送信待ちの時、または、udp_rcv_dat で受信待ちの時に、udp_del_cep により、UDP 通信端点を削除すると、それぞれの関数の戻り値には、E_DLT が返される。

次に、udp_del_cep の書式を示す。

```
ER ercd = udp_del_cep(ID cepid);
```

パラメータ cepid には [1] で予約した UDP 通信端点 ID を指定する。

(2) UDP 通信端点オプションの設定と読出し

設定可能な TCP 通信端点オプションは無いため、どちらの関数も戻り値として E_PAR が返される。

4. ルーティングの設定

ルーティングエントリには、静的ルーティングエントリと向け直し（ICMP）によるルーティングエントリがある。

静的ルーティングエントリは、予め決められたルーティング情報であり、ルーティング設定ファイル route_cfg.c のルーティング表エントリ配列に設定する。なお、ディフォルトゲートウェイのみのシンプルなネットワークでは、サンプルアプリケーション echos の route_cfg.c をそのまま流用できる。

向け直し（ICMP）によるルーティングエントリは、TINET コンフィギュレーション・パラメータ定義ファイルで、ルーティング表で予め確保するエントリ数を定義し、ルーティング設定ファイル route_cfg.c のルーティング表エントリ配列に、空のエントリとして確保する。

(1) ルーティング表のエントリ数の設定

エントリ数の設定するマクロは、TINET コンフィギュレーション・パラメータ定義ファイルで定義する。

[1] NUM_STATIC_ROUTE_ENTRY
ルーティング表の静的ルーティングエントリ数を指定する。

[2] NUM_REDIRECT_ROUTE_ENTRY
ルーティング表で予め確保する、向け直し（ICMP）によるルーティングエントリ数を指定する。0 を指定すると、向け直し（ICMP）を無視する。

(2) ルーティング表エントリ構造体（IPv4）

IPv4 では、#include <netinet/in_var.h> で定義されている。各フィールドの意味を以下に示す。

T_IN4_ADDR	targe	目標ネットワークのIPアドレス、 ディフォルトゲートウェイでは0を指定する。
T_IN4_ADDR	mask	目標ネットワークのサブネットマスク、 ディフォルトゲートウェイでは0を指定する。
T_IN4_ADDR	gateway	ゲートウェイの IP アドレス、 自ネットワーク内では0を指定する。

(3) ルーティング表エントリ構造体（IPv6）

IPv6 では、#include <netinet6/in6_var.h> で定義されている。各フィールドの意味を以下に示す。

T_IN6_ADDR	target	目標ネットワークアドレス
T_IN6_ADDR	gateway	ゲートウェイの IP アドレス
uint32_t	expire	有効時間が切れる時刻、0xffffffff を指定すること。
uint8_t	flags	フラグ、0x01 を指定すること。
uint8_t	prefix_len	プレフィックス長

IP アドレスは、{{{ と }}} で囲み、1 オクテット単位で指定する。例を以下に示す。

```
{ { { 0xfe, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0x41,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 } } }
```

(4) インクルードファイル

以下のインクルードファイルを指定すること。

[1] TOPPERS/ASP

```
#include <kernel.h>
#include <tinet_defs.h>
#include <tinet_config.h>
#include <netinet/in.h>
#include <netinet/in_var.h>
```

[2] TOPPERS/JSP

```
#include <s_services.h>
#include <t_services.h>
#include <tinet_defs.h>
#include <tinet_config.h>
#include <netinet/in.h>
#include <netinet/in_var.h>
```

(5) ルーティング表エントリ配列（IPv4）

以下のように指定すること。

```
T_IN4_RTENTRY routing_tbl[NUM_ROUTE_ENTRY] = {
    <ルーティング表エントリ構造体 1>,
    <ルーティング表エントリ構造体 2>,
    ...
    <ルーティング表エントリ構造体 n>,
    };
```

(6) ルーティング表エントリ配列（IPv6）

以下のように指定すること。

```
T_IN6_RTENTRY routing_tbl[NUM_ROUTE_ENTRY] = {
    <ルーティング表エントリ構造体 1>,
    <ルーティング表エントリ構造体 2>,
    ...
    <ルーティング表エントリ構造体 n>,
    };
```

(7) 探索順序

探索は、インデックスが大きな順、つまり、ルーティング表エントリ配列の最後の <ルーティング表エントリ構造体 n> から、最初の <ルーティング表エントリ構造体 1> に向って行われる。

5. TINET 独自機能

5. 1 タスクからの Time Wait 状態の TCP 通信端点分離機能

TCP 通信端点は、ソケットインタフェースにおけるファイルディスクリプタと異なり、TCP の接続状態が完全に終了するまで再利用可能とはならない。TCP/IP プロトコルの仕様に従うと、接続状態が完全に終了するまで数分かかる場合がある。問題になるのは、先に、TCP 通信端点のコネクション切断 API の tcp_sht_cep を呼出し、コネクションを切断する場合である。この時、tcp_sht_cep で指定

された TCP 通信端点は、最終的に Time Wait 状態になり、TCP 通信端点のクローズ API の tcp_cls_cep を呼出したタスクも、タイムアウト待ち状態になる。従って、サーバ側から切断する応用プログラム（WWW など）のタスクでは、タイムアウトするまで、次の接続要求を受信することができない。

これに対応するため、TINET は、終了待ちの TCP 通信端点をタスクから切り離すことにより、タスクが待ち状態にならないようにする機能を持っており、有効にするためには、コンパイル時コンフィギュレーションパラメータ NUM_TCP_TW_CEP_ENTRY を tinet_app_config.h 等に指定し、確保する TW 用 TCP 通信端点の数（1 以上の値）を定義する。

TCP 通信端点が Time Wait になると、TCP 通信端点から、Time Wait に必要な通信管理データを TW 用 TCP 通信端点にコピーし、元の TCP 通信端点を開放する。これに伴って、タスクも待ち状態から開放される。また、TW 用 TCP 通信端点には Time Wait に必要な通信管理データのみをコピーすることで、メモリの消費を抑えている。

ただし、この機能を有効にしていても、コネクションの同時切断のタイミングによっては、分離されない場合がある。

5. 2 受信ウィンドバッファの省コピー機能

ITRON TCP/IP API 仕様では、TCP 通信端点を生成する静的 API で、受信ウインドバッファの先頭アドレスの指定に、NADR を指定すると、プロトコルスタックで、受信ウインドバッファを確保することになっている。

TINET では、ネットワークバッファを、受信ウインドバッファとすることで、NADR の指定に対応している。さらに、ネットワークインタフェースで受信したプロトルコデータを保持するネットワークバッファを、そのまま受信ウインドバッファとすることで、ネットワークインタフェースと、TINET 内部で、データのコピーを省いている。特に、省コピー API を使用することにより、API におけるデータのコピーも行わないことも可能である。

この機能に関係するコンパイル時コンフィギュレーションパラメータを、以下に示す。

(1) TCP_CFG_RWBUF_CSAVE_ONLY
TCP 通信端点の受信ウィンドバッファの省コピー機能を組込み、この機能のみ使用する。TCP 通信端点を生成する静的 API で、受信ウインドバッファの先頭アドレスの指定に、応用プログラムが用意したバッファを指定しても無視する。

(2) TCP_CFG_RWBUF_CSAVE
TCP 通信端点の受信ウィンドバッファの省コピー機能を組込む。TCP 通信端点を生成する静的 API で、受信ウインドバッファの先頭アドレスの指定に、NADR を指定した場合は、受信ウィンドバッファの省コピー機能を使用するが、応用プログラムが用意したバッファを指定した場合は、受信ウィンドバッファの省コピー機能を使用しない。

(3) TCP_CFG_RWBUF_CSAVE_MAX_QUEUES
TCP 通信端点の受信ウィンドバッファの省コピー機能の、受信ウィンドバッファキューの最大エントリ数である。ただし、正常に受信したセグメントも破棄するため、再送回数が増加する。また、指定しないと制限しない。

なお、TCP_CFG_RWBUF_CSAVE_ONLY と TCP_CFG_RWBUF_CSAVE の、いずれも指定しない場合は、TCP 通信端点を生成する静的 API で、受信ウインドバッファの先頭アドレスの指定に、NADR を指定することができない。

5. 3 送信ウィンドバッファの省コピー機能

ITRON TCP/IP API 仕様では、TCP 通信端点を生成する静的 API で、受信ウインドバッファと同様に、送信ウインドバッファの先頭アドレスの指定に、NADR を指定すると、プロトコルスタックで、送信ウインドバッファを確保することになっている。

TINET では、ネットワークバッファを、送信ウインドバッファとすることで、NADR の指定に対応している。さらに、書込まれたデータの前に必要なヘッダを付加して、そのままネットワークインタフェースに渡すことにより、ネットワークインタフェースと、TINET 内部で、データのコピーを省いている。特に、省コピー API を使用することで、API におけるデータのコピーも行わないことも可能である。

ただし、イーサネット出力時に、NIC でネットワークバッファを開放する（コンパイル時コンフィギュレーションパラメータ ETHER_NIC_CFG_RELEASE_NET_BUF を、指定する必要がある）ディバイスドライバでは、この送信ウィンドバッファの省コピー機能を利用することはできない。

この機能に関係するコンパイル時コンフィギュレーションパラメータを、以下に示す。

(1) TCP_CFG_SWBUF_CSAVE_ONLY
TCP 通信端点の送信ウィンドバッファの省コピー機能を組込み、この機能のみ使用する。TCP 通信端点を生成する静的 API で、送信ウインドバッファの先頭アドレスの指定に、応用プログラムが用意したバッファを指定しても無視する。

(2) TCP_CFG_SWBUF_CSAVE
TCP 通信端点の送信ウィンドバッファの省コピー機能を組込む。TCP 通信端点を生成する静的 API で、送信ウインドバッファの先頭アドレスの指定に、NADR を指定した場合は、送信ウィンドバッファの省コピー機能を使用するが、応用プログラムが用意したバッファを指定した場合は、送信ウィンドバッファの省コピー機能を使用しない。

(3) TCP_CFG_SWBUF_CSAVE_MAX_SIZE
TCP 通信端点の送信ウィンドバッファの省コピー機能で、送信ウィンドバッファに使用するネットワークバッファの最大サイズであり、標準値は IF_PDU_SIZE である。

(4) TCP_CFG_SWBUF_CSAVE_MIN_SIZE
TCP 通信端点の送信ウィンドバッファの省コピー機能で、送信ウィンドバッファに使用するネットワークバッファの最小サイズであり、標準値は 0 である。

なお、TCP_CFG_SWBUF_CSAVE_ONLY と TCP_CFG_SWBUF_CSAVE の、いずれも指定しない場合は、TCP 通信端点を生成する静的 API で、送信ウインドバッファの先頭アドレスの指定に、NADR を指定することができない。

5. 4 ノンブロッキングコールの無効化

応用プロクラムで、ノンブロッキングコールを使用しない場合は、TCP と UDP のノンブロッキングコール機能を組込まないで、メモリを節約することができる。

この機能に関係するコンパイル時コンフィギュレーションパラメータを、以下に示す。

(1) TCP_CFG_NON_BLOCKING
TCP のノンブロッキングコール機能を組込む。

(2) UDP_CFG_NON_BLOCKING
UDP のノンブロッキングコール機能を組込む。

ただし、過去のリリースとの互換性のため、どちらのパラメータも、tinet/tinet_config.h に指定されており、既定では、ノンブロッキングコール機能が組込まれるようになっている。従って、組込まない場合は、tinet_app_config.h などで、#undef により、指定を解除しなければならない。

5. 5 TCP 受付口の無効化

応用プログラムが、クライアント機能のみで構成され、相手からの接続要求を受け付けないのであれば、TCP 受付口は不要であり、この TCP 受付口と、処理関数を組込まないで、メモリを節約することができる。

この機能に関係するコンパイル時コンフィギュレーションパラメータを、以下に示す。

(1) TCP_CFG_PASSIVE_OPEN
TCP の受動オープン機能を組込む。

ただし、過去のリリースとの互換性のため、このパラメータは、tinet/tinet_config.h に指定されており、既定では、TCP の受動オープン機能が組込まれるようになっている。従って、組込まない場合は、tinet_app_config.h などで、#undef により、指定を解除しなければならない。

5. 6 TCP ヘッダのトレース出力機能

送受信する TCP セグメントの TCP ヘッダと TCP 通信端点の情報を出力する機能である。なお、CONSOLE_PORTID で指定されるシリアルポートに直接出力するので、SYSLOG 出力が乱れることがある。受信時の出力例と意味を以下に示す。

```
<I 329.599=c: 4 s:CW f:60c00:--A---- a: 74461 s: 76082 w:58400 l:   0=
```

329.599	受信した時間、1/1000 秒単位、または 1 秒単位
c: 4	TCP 通信端点 ID
s:CW	TCP FSM 状態（tinet/netinet/tcp_fsm.h 参照）
f:60c00	TCP 通信端点の状態フラグ（16 進数、tinet/netinet/tcp_var.h 参照）
:--A----	TCP ヘッダのフラグフィールドの値（tinet/netinet/tcp.h 参照）
a: 74461	TCP ヘッダの確認応答番号（コネクション確立時からの相対値）
s: 76082	TCP ヘッダのシーケンス番号（コネクション確立時からの相対値）
w:58400	TCP ヘッダのウインドサイズ
l:0	受信ペイロードデータ数

送信時の出力例と意味を以下に示す。

```
=O 329.627=c: 4 s:CW f:60d20:--AP--- s: 74461 a: 76082 w: 2920 l:1460>
```

329.627	送信した時間、1/1000 秒単位、または 1 秒単位
c: 4	TCP 通信端点 ID
s:CW	TCP FSM 状態（tinet/netinet/tcp_fsm.h 参照）
f:60d20	TCP 通信端点の状態フラグ（16 進数、tinet/netinet/tcp_var.h 参照）
:--AP---	TCP ヘッダのフラグフィールドの値（tinet/netinet/tcp.h 参照）
s: 74461	TCP ヘッダのシーケンス番号（コネクション確立時からの相対値）
a: 76082	TCP ヘッダの確認応答番号（コネクション確立時からの相対値）
w: 2920	TCP ヘッダのウインドサイズ
l:1460	送信ペイロードデータ数

この機能に関係するコンパイル時コンフィギュレーションパラメータを、以下に示す。

(1) TCP_CFG_TRACE
TCP ヘッダのトレース出力機能を組込む。

(2) TCP_CFG_TRACE_IPV4_RADDR
トレース出力対象のリモートホストの IPv4 アドレスを指定する。IPV4_ADDRANY を指定すると、全てのホストを対象とする。

(3) TCP_CFG_TRACE_LPORTNO
トレース出力対象のローカルホストのポート番号を指定する。TCP_PORTANY を指定すると、全てのポート番号を対象にする。

(4) TCP_CFG_TRACE_RPORTNO
トレース出力対象のリモートホストのポート番号を指定する。TCP_PORTANY を指定すると、全てのポート番号を対象にする。

5. 7 TINETのライブラリ化

TINET のライブラリ化は、メモリ使用量を削減することを目的に実装している。このため、ライブラリ化されているのは ITRON TCP/IP API 部分のみであり、TINET のコア部分のライブラリ化は行われていない。また、コンパイル時オプションにより、処理内容が変わるため、ライブラリも再構築する必要がある。従って、ライブラリとアプリケーションプログラムを別々に構築しておき、後でリンクする方法はサポートしていない。

ITRON TCP/IP API 部分もライブラリ化させないためには、アプリケーションの Makefile に NO_USE_TINET_LIBRARY = true を指定する。

5. 8 IPv6におけるアドレス管理とPath MTUへの対応

TINET リリース 1.3 まで、IPv6 におけるアドレス管理は限定的な対応のみであり、Path MTU にも対応していなかったが、ホスト情報のキャッシュを実装することにより、TINET リリース 1.4 からは、ほぼ完全に対応した。

この機能に関係するコンパイル時コンフィギュレーションパラメータを、以下に示す。

(1) NUM_IN6_IFADDR_ENTRY
インタフェースのアドレスリスト（IPv6）のエントリ数である。

(2) NUM_ND6_DEF_RTR_ENTRY
ディフォルトルータリストのエントリ数で、最大値は 16 である。0 を指定するとルータ通知を受信しない。ただし、現在は、ルータ通知の受信以外にサイトローカルアドレス等を設定する方法がない。

(3) NUM_ND6_PREFIX_ENTRY
プレフィックスリストのエントリ数で、最大値は 16 である。

(4) NUM_IN6_HOSTCACHE_ENTRY
IPv6 用ホスト情報キャッシュのエントリ数で、0 を指定すると IPv6 用ホスト情報キャッシュを組込まない。また、この場合、Path MTU への対応も限定的になる。

6. TINET 独自 API

6. 1 ネットワーク統計情報

送受信オクテット数、送受信パケット数等の統計情報のカウンタ（net_count）が、単純変数、構造体、配列により組込まれている。

(1) ネットワーク統計情報の有効化

コンパイル時コンフィギュレーション・ファイルのいずれかで、プロトコル毎にネットワーク統計情報を有効にする事が必要である。有効にするためには、マクロ NET_COUNT_ENABLE に、プロトコル識別フラグ（インクルードファイル net/net.h で定義されている）をビット論理和により設定する。

(2) ネットワーク統計情報の標準データ型と標準構造体

いずれもインクルードファイル net/net_count.h に定義されている。

```
typedef UD T_NET_COUNT_VAL;

typedef struct t_net_count {
        T_NET_COUNT_VAL       in_octets;          /* 受信オクテット数      */
        T_NET_COUNT_VAL       out_octets;         /* 送信オクテット数      */
        T_NET_COUNT_VAL       in_packets;         /* 受信バケット数        */
        T_NET_COUNT_VAL       out_packets;        /* 送信バケット数        */
        T_NET_COUNT_VAL       in_err_packets;     /* 受信エラーバケット数  */
        T_NET_COUNT_VAL       out_err_packets;    /* 送信エラーバケット数  */
        } T_NET_COUNT;
```

(3) プロトコル毎のネットワーク統計情報

以下に、プロトコル毎のネットワーク統計情報の変数または配列を示す。（）内はインクルードファイル net/net.h に定義されているプロトコル識別フラグである。また、配列変数の場合、配列の内容は、インクルードファイル net/net_count.h を参照すること。

[1] PPP の HDLC（PROTO_FLG_PPP_HDLC）

標準構造体変数で、変数名は net_count_hdlc である。

[2] PPP の認証プロトコル（PROTO_FLG_PPP_PAP）

標準データ型変数で、変数名は、受信オクテット数が

```
net_count_ppp_upap_in_octets
```

送信オクテット数が

```
net_count_ppp_upap_out_octets
```

[3] PPP のリンク制御プロトコル（PROTO_FLG_PPP_LCP）

標準データ型変数で、変数名は、受信オクテット数が

```
net_count_ppp_lcp_in_octets
```

送信オクテット数が

```
net_count_ppp_lcp_out_octets
```

[4] PPP の IP 依存制御プロトコル（PROTO_FLG_PPP_IPCP）
標準データ型変数で、変数名は、受信オクテット数が

```
net_count_ppp_ipcp_in_octets
```

送信オクテット数が

```
net_count_ppp_ipcp_out_octets
```

[5] PPP 全体（PROTO_FLG_PPP）
PPP 全体のネットワーク統計情報は、標準構造体変数で、変数名は net_count_ppp である。また、PPP での net_buf の割当て失敗数は、標準データ型変数で、変数名は net_count_ppp_no_buf である。

[6] ループバックインタフェース（PROTO_FLG_LOOP）
標準構造体変数で、変数名は net_count_loop である。

[7] イーサネットディバイスドライバ NIC（PROTO_FLG_ETHER_NIC）
標準データ型配列変数で、変数名は net_count_ether_nic である。

[8] （PROTO_FLG_ETHER）
標準構造体変数で、変数名は net_count_ether である。

[9] （PROTO_FLG_ARP）
標準構造体変数で、変数名は net_count_arp である。

[10] （PROTO_FLG_IP4）
標準データ型配列変数で、変数名は net_count_ip4 である。

[11] （PROTO_FLG_IP6）
標準データ型配列変数で、変数名は net_count_ip6 である。

[12] （PROTO_FLG_ICMP4）
標準構造体変数で、変数名は net_count_icmp4 である。

[13] （PROTO_FLG_ICMP6）
標準データ型配列変数で、変数名は net_count_icmp6 である。

[14] （PROTO_FLG_ND6）
標準データ型配列変数で、変数名は net_count_nd6 である。

[15] （PROTO_FLG_UDP）
標準構造体変数で、変数名は net_count_udp である。

[16] （PROTO_FLG_TCP）
標準データ型配列変数で、変数名は net_count_tcp である。

[17] （PROTO_FLG_NET_BUF）
net_buf に関しては、特殊であるためサンプルアプリケーション nserv で使用している netapp/dbg_cons.c の関数 net_count を参照すること。

6. 2 SNMP 用管理情報ベース（MIB）

コンパイル時コンフィギュレーション・ファイルのいずれかで、マクロ SUPPORT_MIB を定義することにより、SNMP 用管理情報ベース（MIB）に準拠したネットワーク統計の取得が可能である。ただし、TINET 自体は、管理情報ベース（MIB）に準拠したネットワーク統計を提供するだけで，SNMP をサポートしていない。また、RFC1213、RFC2465、RFC2466 に定義されている全ての情報が取得できるわけではない。取得できる情報は、関係するインクルードファイルの構造体の定義を参照すること。

以下に、グループ、構造体を定義しているインクルードファイル、構造体名、構造体変数名を示す。

(1) TCP グループ

インクルードファイル	netinet/tcp_var.h
構造体名	T_TCP_STATS
変数名	tcp_stats

(2) UDP グループ

インクルードファイル	netinet/udp_var.h
構造体名	T_UDP_STATS
変数名	udp_stats

(3) ICMPv4 グループ

インクルードファイル	netinet/icmp_var.h
構造体名	T_ICMP_STATS
変数名	icmp_stats

(4) IPv4 グループ

インクルードファイル	netinet/ip_var.h
構造体名	T_IP_STATS
変数名	ip_stats

(5) ICMPv6 グループ

インクルードファイル	netinet/icmp6.h
構造体名	T_ICMP6_IFSTAT
変数名	icmp6_ifstat

(6) IPv6 グループ

インクルードファイル	netinet6/ip6_var.h
構造体名	T_IN6_IFSTAT
変数名	in6_ifstat

(7) ネットワークインタフェース（イーサネット）グループ

インクルードファイル	net/if_var.h
構造体名	T_IF_STATS
変数名	if_stats

6. 3 TINET 内部アクセス関数、サポート関数、全域変数とマクロ

応用プログラムから TINET 内部にアクセスするための関数、サポート関数、全域変数とマクロである。

(1) インタフェースに IPv4 アドレスを設定する関数

【C 言語 API】

```
ER ercd = in4_add_ifaddr (T_IN4_ADDR addr, T_IN4_ADDR mask);
```

【パラメータ】

T_IN4_ADDR	addr	IPアドレス
T_IN4_ADDR	mask	サブネットマスク

【リターンパラメータ】

ER	ercd	エラーコード（現在は常にE_OK）

【エラーコード】

E_OK	現在は常にE_OK

【インクルードファイル】

```
<netinet/in.h>
```

【機能】

インタフェースに IPv4 アドレスを設定する。ただし、ネットワークインタフェースがイーサネットで、ネットワーク層として IPv4 が指定されている場合のみ有効である。

(2) IPv4 用静的経路表に経路情報を設定する関数

【C 言語 API】

```
ER ercd = in4_add_route (int index, T_IN4_ADDR target,
                                    T_IN4_ADDR mask,
                                    T_IN4_ADDR gateway);
```

【パラメータ】

int	index	エントリのインデックス
T_IN4_ADDR	target	目標ネットワークのIPアドレス
T_IN4_ADDR	mask	目標ネットワークのサブネットマスク
T_IN4_ADDR	gateway	ゲートウェイのIPアドレス

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_PAR	引数indexの値が負か、コンパイル時コンフィギュレーションパラメータNUM_ROUTE_ENTRY以上のときE_PAR

【インクルードファイル】

```
<netinet/in.h>
```

【機能】

IPv4 用静的経路表に経路情報を設定する。ただし、ネットワークインタフェースがイーサネットで、ネットワーク層として IPv4 が指定されている場合のみ有効である。

(3) IPv4 アドレスをリテラル表現（文字列）に変換する関数

【C 言語 API】

```
char *p_retbuf = ip2str (char *p_buf, const T_IN4_ADDR *p_ipaddr);
```

【パラメータ】

char	buf	リテラル表現のIPv4アドレスを格納するバッファ
const T_IN4_ADDR	ipaddr	IPv4アドレス

【リターンパラメータ】

char	retbuf	リテラル表現の IPv4 アドレスが格納されたバッファ

【インクルードファイル】

```
<netinet/in.h>
```

【機能】

IPv4 アドレスをリテラル表現（文字列）に変換する。パラメータ buf には、最低 16 バイトの領域が必要である。また、パラメータ buf に、NULL を指定すると、TINET 内部で確保してあるバッファに IPv4 アドレスをリテラル表現に変換して書き込み、そのアドレスを返す。コンパイル時コンフィギュレーションパラメータ NUM_IPADDR_STR_BUFF によりバッファ数を指定することが出来る。ただし、バッファ数を超えて連続的に呼出すとバッファを上書きする。

(4) IPv6 アドレスをリテラル表現（文字列）に変換する関数

【C 言語 API】

```
char *p_retbuf = ipv62str (char *p_buf, const T_IN6_ADDR *p_ipaddr);
```

【パラメータ】

char	buf	リテラル表現のIPv6アドレスを格納するバッファ
const T_IN6_ADDR	ipaddr	IPv6アドレス

【リターンパラメータ】

char	retbuf	リテラル表現の IPv6 アドレスが格納されたバッファ

【インクルードファイル】

```
<netinet/in.h>
```

【機能】

IPv6 アドレスをリテラル表現（文字列）に変換する。パラメータ buf には、最低 40 バイトの領域が必要である。また、パラメータ buf に、NULL を指定すると、TINET 内部で確保してあるバッファに IPv6 アドレスをリテラル表現に変換して書き込み、そのアドレスを返す。コンパイル時コンフィギュレーションパラメータ NUM_IPADDR_STR_BUFF によりバッファ数を指定することが出来る。ただし、バッファ数を超えて連続的に呼出すとバッファを上書きする。

(5) ITRON TCP/IP API 機能コードを文字表現に変換する関数

【C 言語 API】

```
const char *p_str = in_strtfn (FN fncd);
```

【パラメータ】

FN	fncd	ITRON TCP/IP API 機能コード

【リターンパラメータ】

const char	str	ITRON TCP/IP API 機能コードの文字表現

【インクルードファイル】

```
<netinet/in.h>
```

【機能】

ITRON TCP/IP API 機能コードを文字表現に変換する。

(6) MAC アドレスをリテラル表現（文字列）に変換する関数

【C 言語 API】

```
char *p_retbuf = mac2str (char *p_buf, uint8_t *p_macaddr);
```

【パラメータ】

char	buf	リテラル表現のMACアドレスを格納するバッファ
uint8_t	macaddr	MACアドレス

【リターンパラメータ】

char	retbuf	リテラル表現の MAC アドレスが格納されたバッファ

【インクルードファイル】

```
<sil.h>      TOPPERS/ASP では必要
<net/net.h>
```

【機能】

MAC アドレスをリテラル表現（文字列）に変換する。パラメータ buf には、最低 18 バイトの領域が必要である。また、パラメータ buf に、NULL を指定すると、TINET 内部で確保してあるバッファに MAC アドレスをリテラル表現に変換して書き込み、そのアドレスを返す。コンパイル時コンフィギュレーションパラメータ NUM_MACADDR_STR_BUFF によりバッファ数を指定することが出来る。ただし、バッファ数を超えて連続的に呼出すとバッファを上書きする。

(7) IPv6 の IPV6_ADDRANY に対応する全域変数

【C 言語 API】

```
const T_IN6_ADDR ipv6_addrany;
```

【機能】

T_IPV6EP の ipaddr フィールドに、値 IPV4_ADDRANY を代入するとき、IPv4 では、

```
myaddr.ipaddr = IPV4_ADDRANY;
```

と指定できるが、IPv6 では、同様の指定ができないために用意した全域変数であり、以下のように、メモリ操作関数を呼び出してコピーする。

```
memcpy(&myaddr.ipaddr, &ipv6_addrany, sizeof(T_IN6_ADDR));
```

なお、この全域変数はマクロで定義している。

(8) TINET のバージョン情報マクロ

【C 言語 API】

```
TINET_PRVER
```

【ビット配分】

ビット12〜15　メジャーリリース（現在の値は1）
ビット4〜11　マイナーリリース（現在の値は5）
ビット3〜0　パッチレベル（現在の値は0）

【インクルードファイル】

```
<net/net.h>
```

(9) 8 ビット毎に指定した IPv4 アドレスを 32 ビットにするマクロ

【C 言語 API】

```
T_IN4_ADDR addr = MAKE_IPV4_ADDR(uint8_t a, uint8_t b,
                                 uint8_t c, uint8_t d);
```

【パラメータ】

uint8_t	a	IPv4アドレスのビット24〜31
uint8_t	b	IPv4アドレスのビット16〜23
uint8_t	c	IPv4アドレスのビット8〜15
uint8_t	d	IPv4アドレスのビット0〜7

【リターンパラメータ】

T_IN4_ADDR	addr	32ビットのIPv4アドレス

【インクルードファイル】

```
<netinet/in.h>
```

【機能】

各オクテットの値から IPv4 アドレスを生成する。

(10) 一般定数マクロ

TCP_REP_NONE	該当する TCP 受付口が無い。値は（0）。
TCP_CEP_NONE	該当する TCP 通信端点が無い。値は（0）。
UDP_CEP_NONE	該当する UDP 通信端点が無い。値は（0）。

6. 4 応用プログラムコールバック関数

TINET から呼出される応用プログラムコールバック関数であり、応用プログラム側で用意する必要がある。

(1) IPv4 アドレス重複検出時のコールバック関数

【C 言語 API】

```
boot_t reply = arp_callback_duplicated(uint8_t *shost);
```

【パラメータ】

uint8_t	shost	重複相手のMACアドレス

【リターンパラメータ】

bool_t	reply	重複の通知

【インクルードファイル】

<netinet/if_ether.h>

【コンパイル時コンフィギュレーションパラメータ】

ARP_CFG_CALLBACK_DUPLICATED

【機能】

戻り値に TRUE を指定すると、TINET で重複相手の MAC アドレスを syslog に出力し、重複相手にも重複したことを伝える。FALSE を指定すると何もしない。

7. TOPPERS/ASP 環境におけるインストールとファイルの作成・変更

7. 1 インストール手順

(1) TOPPERS/ASP をインストールする。TOPPERS/ASP のインストールに関しては、TOPPERS/ASP カーネルユーザズマニュアル（user.txt）の「**3.1. 開発環境の準備**」を参照すること。

(2) TOPPERS/ASP 用 TINET 配布ファイル（tinet-asp-1.5.tar.gz）を TOPPERS/ASP ルートディレクトリで展開する。

7. 2 ファイルの作成、設定

アプリケーションプログラムの他に、変更・新規作成すべきファイルと TINET コンフィグレータで生成されるファイルを以下に示す。全て TOPPERS/ASP ルートディレクトリからの相対パスであり、Makefile マクロの意味を以下に示す。

$(ARCH)	プロセッサ依存部
$(TARGET)	ターゲット依存部
$(NIC)	ネットワークインタフェース
$(APPLDIR)	アプリケーションプログラムのディレクトリ
$(APPLNAME)	アプリケーションプログラム名

(1) target/$(TARGET)/tinet_target_config.h【新規作成】
システムに依存する TINET コンフィギュレーション・パラメータを定義するファイルである。内容については tinet_config.pdf（tinet_config.txt）を参照すること。

(2) $(APPLDIR)/tinet_app_config.h【新規作成】
アプリケーションに依存する TINET コンフィギュレーション・パラメータを定義するファイルである。内容については tinet_config.pdf（tinet_config.txt）を参照すること。

(3) tinet/netdev/$(NIC)/tinet_nic_config.h【新規作成】
ネットワークインタフェースに依存する TINET コンフィギュレーション・パラメータを定義するファイルである。

(4) target/$(TARGET)/tinet_target_defs.h【新規作成】
プロセッサに依存する TCP/IP パラメータを定義するファイルである。内容については tinet_defs.pdf（tinet_defs.txt）を参照すること。

(5) tinet/netdev/$(NIC)/tinet_nic_defs.h【新規作成】
ネットワークインタフェースに依存する TCP/IP パラメータを定義するファイルである。内容については tinet_defs.pdf（tinet_defs.txt）を参照すること。

(6) $(APPLDIR)/tinet_$(APPLNAME).cfg【新規作成】
TINET コンフィギュレーションファイルである。内容については「**2.1 TOPPERS/ASP 環境における TINET コンフィグレーションファイル**」を参照すること。

(7) $(APPLDIR)/route_cfg.c【新規作成】
静的ルーティングを設定するファイルである。内容については「**4. ルーティングの設定**」を参照すること。なお、ディフォルトゲートウェイのみのシンプルなネットワークでは、サンプルアプリケーション echos の route_cfg.c をそのまま流用できる。

(8) $(APPLDIR)/$(APPLNAME).c【変更】
TINET を使用するために、以下のインクルードファイルを指定する必要がある。

```
#include "tinet_cfg.h"
#include <netinet/in.h>
#include <netinet/in_itron.h>
```

(9) $(APPLDIR)/$(APPLNAME).cfg【変更】
TINET 内部で使用するカーネルオブジェクトを取り込むために、TINET コンフィギュレーションファイルをインクルードする。

```
INCLUDE("../tinet/tinet_asp.cfg");
```

(10) $(APPLDIR)/Makefile【変更】
変更については「**7.3 アプリケーションの Makefile**」を参照すること。

(11) $(APPLDIR)/tinet_cfg.c【自動生成】
TCP 受付口、TCP 通信端点、及び UDP 通信端点に対応する構造体が生成されるファイルで、TOPPERS/ASP コンフィグレータにより生成される。アプリケーションプログラム、TINET と共にコンパイルしてリンクする。

(12) $(APPLDIR)/tinet_kern.cfg【自動生成】
TINET 内部で使用するカーネルオブジェクトの静的 API が生成されるファイルで、TOPPERS/ASP コンフィグレータにより生成される。TOPPERS/ASP のシステムコンフィギュレーションファイル（標準では $(APPLNAME).cfg）にインクルードする。

(13) $(APPLDIR)/tinet_cfg.h【自動生成】
TCP 受付口、TCP 通信端点、及び UDP 通信端点の ID 自動割付結果ファイルで、TOPPERS/ASP コンフィグレータにより生成される。

7. 3 アプリケーションの Makefile

7. 3. 1 アプリケーションの Makefile の修正

ここでは、既に存在するアプリケーションの Makefile に、TINET を組み込むための修正方法を述べる。TINET に付属するサンプルプログラムの構築については、「**9.1 サンプルアプリケーションの構築** 」を参照すること。

標準的な TOPPERS/ASP 環境におけるサンプルアプリケーションの Makefile の

```
#
#  アプリケーションプログラムに関する定義
#
APPLNAME = ...
   ... 途中略 ...
ifdef APPLDIR
  INCLUDES := $(INCLUDES) $(foreach dir,$(APPLDIR),-I$(dir))
endif
```

の後に、次に示す TINET 用の定義を追加する。

```
#
#  ネットワークサービスの定義
#

#  ネットワークインタフェースの選択

NET_IF = ether

#  イーサネット・ディバイスドライバの選択

NET_DEV = if_ed

#  ネットワーク層の選択、何れか一つ選択する。

SUPPORT_INET4 = true
#SUPPORT_INET6 = true

#  トランスポート層の選択

SUPPORT_TCP = true
#SUPPORT_UDP = true

#
#  システムサービスの Makefile のインクルード
#
include $(SRCDIR)/tinet/Makefile.tinet
```

7. 3. 2 アプリケーションの Makefile による TINET に組込む機能の指定

アプリケーションの Makefile による TINET に組込む機能を指定する方法を以下に示す。

(1) ネットワークインタフェースの選択
イーサネットのみ選択できる。

```
NET_IF = ether
```

(2) イーサネット・ディバイスドライバの選択

以下に示す通り、NET_DEV にイーサネット・ディバイスドライバを定義する。現在、TINET の配布ファイルでは、NE2000 互換の NIC のイーサネット・ディバイスドライバのみ提供している。

```
NET_DEV = if_ed
```

(3) ネットワーク層の選択

以下に示す定義から、IPv4 と IPv6 の何れか一つ選択する。

```
SUPPORT_INET4 = true
#SUPPORT_INET6 = true
```

(4) トランスポート層の選択

以下に示す定義から、必要なプロトコルを選択する。

```
SUPPORT_TCP = true
#SUPPORT_UDP = true
```

7. 3. 3 システムサービス用 Makefile.tinet の変数

TINET は、システムサービスとして TOPPERS/ASP カーネルと組合せてコンパイルしている。TINET を組込むアプリケーション用の Makefile では、TINET 用の Makefile.tinet を、以下に示すインクルードにより指定している。

```
#
#  システムサービスの Makefile のインクルード
#
include $(SRCDIR)/tinet/Makefile.tinet
```

ここでは、TINET 用の Makefile.tinet で定義している変数を以下に示す。

(1) SYSSVC_DIR

システムサービスのディレクトリを追加する。TINET では $(TINET_DIR)（値は $(TINET_ROOT)/net:$(TINET_ROOT)/netinet:$(TINET_ROOT)/netinet6）を追加している。

(2) SYSSVC_LCSRCS

システムサービスのライブラリ化するソースファイルを追加する。TINET では $(TINET_LCSRCS)（値は tcp_usrreq.c、udp_usrreq.c）を追加している。

(3) SYSSVC_COBJS

システムサービスの C 言語のオブジェクトファイルを追加する。TINET では、$(TINET_COBJS)（値は、多数のファイルのため、ここでは省略）を追加している。

(4) SYSSVC_CFLAGS

システムサービスをコンパイルするときのオプションである。TINET では $(TINET_CFLAGS)（値は未定義）を追加している。

(5) SYSSVC_LIBS

システムサービスのライブラリを追加する。TINET では、$(LIBTINET)（値は libtinet.a）と -lc を追加している。

(6) CLEAN_FILES
make clean で、一緒に消去するファイルを追加する。TINET では、$(TINET_CFG_OUT)（値は tinet_kern.cfg 、 tinet_cfg.h 、 tinet_cfg.c 、 tinet_cfg1_out.* ） と $(MAKE_TINET_LIB)（値は libtinet.a）を追加している。

8. TOPPERS/JSP 環境におけるインストールとファイルの作成・変更

8. 1 インストール手順

TOPPERS/JSP 環境では、"TINET コンフィギュレータを生成する。以下に手順を示す。

(1) TOPPERS/JSP をインストールする。TOPPERS/JSP のインストールに関しては、TOPPERS/JSP カーネルユーザズマニュアル（user.txt）の「**7. 開発環境・インストール・ポーティング** 」を参照すること。

(2) TOPPERS/JSP 用 TINET 配布ファイル（tinet-jsp-1.5.tar.gz）を TOPPERS/JSP ルートディレクトリで展開する。

(3) TINET コンフィギュレータ tinet_cfg（cygwin では tinet_cfg.exe）を生成する。

[1] TINET コンフィギュレータのディレクトリは、TOPPERS/JSP ルートディレクトリの下の tinet/cfg である。このディレクトリに移動する。

[2] make で TINET コンフィギュレータを生成する。

8. 2 ファイルの作成、設定

アプリケーションプログラムの他に、変更・新規作成すべきファイルと TINET コンフィグレータで生成されるファイルを以下に示す。全て TOPPERS/JSP ルートディレクトリからの相対パスであり、Makefile マクロの意味を以下に示す。

$(CPU)	ターゲット CPU
$(SYS)	ターゲットシステム
$(NIC)	ネットワークインタフェース
$(UNAME)	アプリケーションプログラム名

また、$(APP_DIR) は Makefile では定義されていないが、アプリケーションプログラムのディレクトリ名を意味している。

(1) config/$(CPU)/tinet_cpu_config.h【新規作成】
プロセッサに依存する TINET コンフィギュレーション・パラメータを定義するファイルである。内容については tinet_config.pdf（tinet_config.txt）を参照すること。

(2) config/$(CPU)/$(SYS)/tinet_sys_config.h【新規作成】
システムに依存する TINET コンフィギュレーション・パラメータを定義するファイルである。内容については tinet_config.pdf（tinet_config.txt）を参照すること。

(3) $(APP_DIR)/tinet_app_config.h【新規作成】
アプリケーションに依存する TINET コンフィギュレーション・パラメータを定義するファイルである。内容については tinet_config.pdf（tinet_config.txt）を参照すること。

(4) tinet/netdev/$(NIC)/tinet_nic_config.h【新規作成】
ネットワークインタフェースに依存する TINET コンフィギュレーション・パラメータを定義するファイルである。

(5) config/$(CPU)/tinet_cpu_defs.h【新規作成】
プロセッサに依存する TCP/IP パラメータを定義するファイルである。内容については tinet_defs.pdf（tinet_defs.txt）を参照すること。

(6) tinet/netdev/$(NIC)/tinet_nic_defs.h【新規作成】
ネットワークインタフェースに依存する TCP/IP パラメータを定義するファイルである。内容については tinet_defs.pdf（tinet_defs.txt）を参照すること。

(7) $(APP_DIR)/tinet_$(UNAME).cfg【新規作成】
TINET コンフィギュレーションファイルである。内容については「**2.2 TOPPERS/JSP 環境における TINET コンフィグレーションファイル**」を参照すること。

(8) $(APP_DIR)/route_cfg.c【新規作成】
静的ルーティングを設定するファイルである。内容については「**4. ルーティングの設定**」を参照すること。なお、ディフォルトゲートウェイのみのシンプルなネットワークでは、サンプルアプリケーション echos の route_cfg.c をそのまま流用できる。

(9) $(APP_DIR)/$(UNAME).c【変更】
TINET を使用するために、以下のインクルードファイルを指定する必要がある。

```
#include "tinet_id.h"
#include <netinet/in.h>
#include <netinet/in_itron.h>
```

(10) $(APP_DIR)/$(UNAME).cfg【変更】
TINET 内部で使用するカーネルオブジェクトを取り込むために、TINET コンフィギュレーションファイルをインクルードする。

```
#include "../tinet/tinet.cfg"
```

(11) $(APP_DIR)/Makefile【変更】
変更については「**8.3 アプリケーションの Makefile**」を参照すること。

(12) $(APP_DIR)/tinet_cfg.c【自動生成】
TCP 受付口、TCP 通信端点、及び UDP 通信端点に対応する構造体が生成されるファイルで、TINET コンフィグレータにより生成される。アプリケーションプログラム、TINET と共にコンパイルしてリンクする。

(13) $(APP_DIR)/tinet_kern.cfg【自動生成】
TINET 内部で使用するカーネルオブジェクトの静的 API が生成されるファイルで、TINET コンフィグレータにより生成される。TOPPERS/JSP のシステムコンフィギュレーションファイル（標準では $(UNAME).cfg）にインクルードする。

(14) $(APP_DIR)/tinet_id.h【自動生成】

TCP 受付口、TCP 通信端点、及び UDP 通信端点の ID 自動割付結果ファイルで、TINET コンフィグレータにより生成される。

8. 3 アプリケーションの Makefile

8. 3. 1 アプリケーションの Makefile の修正

ここでは、既に存在するアプリケーションの Makefile に、TINET を組み込むための修正方法を述べる。TINET に付属するサンプルプログラムの構築については、「**9.1 サンプルアプリケーションの構築**」を参照すること。

標準的な TOPPERS/JSP 環境におけるサンプルアプリケーションの Makefile の

```
#
#  アプリケーションプログラムに関する定義
#
    ... 途中略 ...
UTASK_LIBS =
```

の後に、次に示す TINET 用の定義を追加する。

```
#
#  ネットワークサービスの定義
#

#  ネットワークインタフェースの選択、何れか一つ選択する。

#NET_IF = loop
#NET_IF = ppp
NET_IF = ether

#  イーサネット・ディバイスドライバの選択

NET_DEV = if_ed

#PPP_CFG_MODEM = true    # PPP で、モデム接続の場合

#  ネットワーク層の選択、何れか一つ選択する。

SUPPORT_INET4 = true
#SUPPORT_INET6 = true

#  トランスポート層の選択

SUPPORT_TCP = true
#SUPPORT_UDP = true

#
#  ミドルウェアの Makefile のインクルード
#
include $(SRCDIR)/tinet/Makefile.tinet
```

8. 3. 2 アプリケーションの Makefile による TINET に組込む機能の指定

アプリケーションの Makefile による TINET に組込む機能を指定する方法を以下に示す。

(1) ネットワークインタフェースの選択

以下に示す定義から、ループバック、シリアルインタフェースを用いた PPP、イーサネットの何れか一つ選択する。ただし、ループバックとシリアルインタフェースを用いた PPP は参考実装である。

```
#NET_IF = loop
#NET_IF = ppp
NET_IF = ether
```

(2) イーサネット・ディバイスドライバの選択

以下に示す通り、NET_DEV にイーサネット・ディバイスドライバを定義する。現在、TINET の配布ファイルには、NE2000 互換の NIC のイーサネット・ディバイスドライバのみ提供している。

```
NET_DEV = if_ed
```

(3) モデム接続の定義

PPP で、モデム接続の場合は、以下の行を有効にする。ただし、シリアルインタフェースを用いた PPP は参考実装である。

```
#PPP_CFG_MODEM = true    # PPP で、モデム接続の場合
```

(4) ネットワーク層の選択

以下に示す定義から、IPv4 と IPv6 の何れか一つ選択する。

```
SUPPORT_INET4 = true
#SUPPORT_INET6 = true
```

(5) トランスポート層の選択

以下に示す定義から、必要なプロトコルを選択する。

```
SUPPORT_TCP = true
#SUPPORT_UDP = true
```

8. 3. 3 ミドルウェア用 Makefile.tinet の変数

TINET は、ミドルウェアとして TOPPERS/JSP カーネルと組合せてコンパイルしている。TINET を組込むアプリケーション用の Makefile では、TINET 用の Makefile.tinet を、以下に示すインクルードにより指定している。

```
#
#  ミドルウェアの Makefile のインクルード
#
include $(SRCDIR)/tinet/Makefile.tinet
```

ここでは、TINET 用の Makefile.tinet で定義している変数を以下に示す。

(1) MTASK_CFG
ミドルウェアのコンフィギュレーションファイル（ソース）を追加する。TINET では、$(TINET_CFG)（値は tinet_$(UNAME).cfg）を追加している。

(2) MTASK_KERNEL_CFG
ミドルウェアのコンフィギュレータから出力され、TOPPERS/JSP のシステムコンフィギュレーションファイルにインクルードされるファイルを追加する。TINET では、$(TINET_KERNEL_CFG)（値は、tinet_kern.cfg、tinet.cfg、TINET 内部で使用するカーネルオブジェクトを定義しているコンフィギュレーションファイル）を追加している。

(3) MTASK_DIR
ミドルウェアのディレクトリを追加する。TINET では $(TINET_DIR)（値は $(TINET_ROOT)/net:$(TINET_ROOT)/netinet:$(TINET_ROOT)/netinet6 ） を追加している。

(4) MTASK_LCSRCS
ミドルウェアのライブラリ化するソースファイルを追加する。TINET では $(TINET_LCSRCS)（値は tcp_usrreq.c、udp_usrreq.c）を追加している。

(5) MTASK_COBJS
ミドルウェアの C 言語のオブジェクトファイルを追加する。TINET では、$(TINET_COBJS)（値は、多数のファイルのため、ここでは省略）を追加している。

(6) MTASK_CFLAGS
ミドルウェアをコンパイルするときのオプションである。TINET では $(TINET_CFLAGS)（値は未定義）を追加している。

(7) MTASK_LIBS
ミドルウェアのライブラリを追加する。TINET では、$(LIBTINET)（値は libtinet.a）と -lc を追加している。

(8) MTASK_CLEAN_FILES
make clean で、一緒に消去するファイルを追加する。TINET では、$(TINET_CFG_OUT)（値は tinet_kern.cfg、tinet_id.h、tinet_cfg.c）と $(MAKE_TINET_LIB)（値は libtinet.a）を追加している。

9. サンプルアプリケーション

サンプルとして、以下のアプリケーションを提供している。【】内は【アプリケーション名、対応するネットワーク層】である。

(1) IPv6 TCP ECHO サーバ【echos6、IPv6】
ネットワークアプリケーションに必要な各ファイルの設定方法の参考となる、TCP エコーサーバ機能のみのシンプルなアプリケーションである。

(2) IPv4 TCP ECHO サーバ【echos4、IPv4】
ネットワークアプリケーションに必要な各ファイルの設定方法の参考となる、TCP エコーサーバ機能のみのシンプルなアプリケーションである。

(3) IPv6 UDP ECHO サーバ【usrv6、IPv6】
　ネットワークアプリケーションに必要な各ファイルの設定方法の参考となる、UDP エコーサーバ機能のみのシンプルなアプリケーションである。

(4) IPv4 UDP ECHO サーバ【usrv4、IPv4】
　ネットワークアプリケーションに必要な各ファイルの設定方法の参考となる、UDP エコーサーバ機能のみのシンプルなアプリケーションである。

(5) クライアントサーバ・セット【nserv、IPv6/IPv4】
　以下に示すサーバが提供されており、必要に応じて組み込むサーバを選択できる。

　[1] WWW サーバ
　[2] TCP エコーサーバ
　[3] UDP エコーサーバ
　[4] TCP ディスカードサーバ
　[5] 簡易コンソール

　また、クライアントとしては以下の機能が提供されており、必要に応じて組み込むクライアントを選択できる。

　[1] TCP エコークライアント
　[2] UDP エコークライアント
　[3] TCP ディスカードクライアント
　[4] UDP ディスカードクライアント
　[5] ping

(6) TOPPERS ASP/JSP サンプルプログラム sample1 のネットワーク対応プログラム【sample1n、IPv6/IPv4】
　TOPPERS/ASP と TOPPERS/JSP のサンプルプログラム sample1 のネットワーク対応プログラムである。telnet で接続すると、シリアルの入出力を引き継いで実行する。切断すると、元のシリアルに入出力を戻す。

(7) 最小構成サーバ【minsv、IPv4】
　WWW サーバ・タスクと TCP エコーサーバ・タスクのみからなる最小構成のサーバである。H8/3069F が内蔵している RAM（16K バイト）と ROM（512K バイト）に収まり、外部メモリは不要である。現在は、品川通信計装サービス製 NKEV-010H8（TOPPERS/JSP リリース 1.4.2 のみ）と秋月電子通商製 H8/3069F（TOPPERS/JSP リリース 1.4.1 以降と TOPPERS/ASP）のシステムに対応している。

9. 1 サンプルアプリケーションの構築

TINET サンプルアプリケーションの構築は、TOPPERS/ASP 環境と TOPPERS/JSP 環境におけるのサンプルアプリケーションの構築とほぼ同じである。

(1) TINET コンフィギュレーションスクリプトの実行
　TINET サンプルアプリケーションの構築用ディレクトリを作成し、TINET コンフィギュレーションスクリプトを実行する。

[1] TOPPERS/ASP 用 TINET コンフィギュレーションスクリプトの実行

```
$ mkdir NETOBJ
$ cd NETOBJ
$ perl ../tinet/tinet_asp_configure -T akih8_3069f_gcc -A echos6
                                    -i ether -v if_ed -n inet6 -s tcp
```

オプション（-T、-A、-a、-U、-L、-f、-D、-l、-t、-d、-r、-p、-g）は、TOPPERS/ASP コンフィギュレーションスクリプトと同じである。TOPPERS/ASP カーネルユーザーズマニュアル（user.txt）の「**５．コンフィギュレーションスクリプトの使い方**」を参照すること。その他のオプションについては、「**9.2 TINET コンフィギュレーションスクリプトのオプション**」を参照すること。

アプリケーションとカーネルを別々に構築する方法については、TOPPERS/ASP 環境におけるサンプルアプリケーションの構築と同じである。TOPPERS/ASP カーネルユーザーズマニュアル（user.txt）の「**3.5 アプリケーションとカーネルを別々に構築する方法**」を参照すること。

[2] TOPPERS/JSP 用 TINET コンフィギュレーションスクリプトの実行

```
$ mkdir NETOBJ
$ cd NETOBJ
$ perl ../tinet/tinet_jsp_configure -C h8 -S akih8_3069f -A echos6
                                    -i ether -v if_ed -n inet6 -s tcp
```

オプション（-C、-S、-T、-A、-U、-L、-D、-P、-p、-l）は、TOPPERS/JSP コンフィギュレーションスクリプトと同じである。TOPPERS/JSP カーネルユーザーズマニュアル（user.txt）の「**7.6 コンフィギュレーションスクリプトの使い方**」を参照すること。その他のオプションについては、「**TINET コンフィギュレーションスクリプトのオプション**」を参照すること。

アプリケーションとカーネルを別々に構築する方法については、TOPPERS/JSP 環境におけるサンプルアプリケーションの構築と同じである。TOPPERS/JSP カーネルユーザーズマニュアル（user.txt）の「**7.5 アプリケーションとカーネルを別々に構築する方法**」を参照すること。

上記の例で、TOPPERS/ASP 環境と TOPPERS/JSP 環境のいずれの場合も、ディレクトリ NETOBJ に以下のファイルが生成される。

Makefile	Makefile
echos6.c	サンプルプログラム本体
echos6.h	サンプルプログラムのヘッダファイル
echos6.cfg	サンプルプログラム用 ASP コンフィギュレーションファイル
tinet_echos6.cfg	サンプルプログラム用 TINET コンフィギュレーションファイル
route_cfg.c	サンプルプログラム用ルーティング表
tinet_app_config.h	サンプルプログラム用コンパイル時指定コンフィギュレーション

必要であれば、Makefile を修正する。修正については、TOPPERS/ASP 環境では「**7.3 アプリケーションの Makefile**」、TOPPERS/JSP 環境では「**8.3 アプリケーションの Makefile**」を参照すること。

以下のサンプルアプリケーションでは、構築上の注意がある。各章を参照すること。

[1] 「**9.3** クライアントサーバ・セット」
[2] 「**9.4** TOPPERS ASP/JSP サンプルプログラム sample1 のネットワーク対応プログラム」
[3] 「**9.5** 最小構成サーバ」

(2) tinet_app_config.h の設定

IPv4 の場合、IP アドレス、サブネットマスク、ディフォルトゲートウェイを指定する。

[1] IPV4_ADDR_LOCAL
自分の IP アドレスを指定する。ただし、PPP を使用するとき、相手に割当ててもらう場合は 0 を指定すること。なお、PPP は参考実装である。

[2] IPV4_ADDR_REMOTE
相手の IP アドレスを指定する。ただし、PPP を使用するとき、相手に割当ててもらう場合は 0 を指定すること。なお、PPP は参考実装である。

[3] IPV4_ADDR_LOCAL_MASK
サブネットマスクを指定する。ただし、ネットワークインタフェースがイーサネットのとき有効である。

[4] IPV4_ADDR_DEFAULT_GW
ディフォルトゲートウェイを指定する。ただし、ネットワークインタフェースがイーサネットのとき有効である。

(3) route_cfg.c の設定

ネットワークインタフェースがイーサネットの場合は、ルーティング表 routing_tbl を設定する。ただし、ディフォルトゲートウェイのみのシンプルなネットワークでは、変更する必要はない。

(4) サンプルプログラムのコンパイル・リンク

TINET サンプルアプリケーションの構築用ディレクトリで、サンプルプログラムをコンパイル・リンクする。コンパイル・リンクの方法を以下に示す。

```
$ make depend
$ make
```

9. 2 TOPPERS/APS 用 TINET コンフィギュレーションスクリプトのオプション

TOPPERS/ASP 環境では、オプション（-T、-A、-a、-U、-L、-f、-D、-l、-t、-d、-r、-p、-g）は、TOPPERS/ASP コンフィギュレーションスクリプトと同じである。TINET コンフィギュレーションスクリプト特有のオプションを以下に示す。

-e <tinetdir>	TINET のソースの置かれているディレクトリ
-i <net_if>	ネットワークインタフェース（必須） <net_if> には ether、ppp、loop の何れかを指定する。
-v <net_dev>	イーサネット・ディバイスドライバ （ネットワークインタフェースに ether を指定した場合は必須）
-n <net_proto>	ネットワーク層プロトコル（必須） <net_proto> には inet4、inet6、inet64m の何れかを指定する。ネットワーク層として IPv6 と IPv4 の両方を組み込む場合は、inet64m を指定する。
-s <trans_proto>	トランスポート層プロトコル（必須）

<trans_proto> には tcp、udp、tcp/udp の何れかを指定する。

9. 3 TOPPERS/JPS 用 TINET コンフィギュレーションスクリプトのオプション

TOPPERS/JSP 環境では、オプション（-C、-S、-T、-A、-U、-L、-D、-P、-p、-l）は、TOPPERS/JSP コンフィギュレーションスクリプトと同じである。また、-d <dir> オプションを除いた TOPPERS/JPS 用 TINET コンフィギュレーションスクリプト特有のオプションは、TOPPERS/APS 用 TINET コンフィギュレーションスクリプトのオプションと同じである。以下に、-d <dir> オプションを述べる。

-d <dir>	テンプレートディレクトリを指定する。 デフォルトは tinet/jsp_sample である。なお、TOPPERS/JSP の同じオプションのデフォルトは sample である。

9. 4 クライアントサーバ・セット【nserv、IPv6/IPv4】の構築

以下に示すサーバが提供されており、必要に応じて組み込むサーバを選択できる。（）内は、tinet/netapp 内にあるソースファイル名である。なお、ディフォルトで、ITRON TCP/IP API 仕様の TCP と UDP の拡張機能が組込まれているので、「**9.3.2 簡易コンソールのコマンド**」の wtw、wte、wue、wtd コマンドで、それぞれのサーバ・タスクを起動する必要がある。

[1] WWW サーバ・タスク（wwws.c）
ルートページの他に、ネットワーク統計情報を表示する 2 ページから構成され、タスク数は最大 2 である。
[2] TCP エコーサーバ・タスク
tcp_echo_srv1.c と tcp_echo_srv2.c のどちらかを選択する。tcp_echo_srv1.c を選択した場合、省コピー API を使用して、ノンブロッキングコールを使用しない時は、タスク数を 8 まで指定可能である。
[3] UDP エコーサーバ・タスク（udp_echo_srv.c）
[4] TCP ディスカードサーバ・タスク（tcp_discard_srv.c）
[5] 簡易コンソール・タスク（dbg_cons.c）
シリアルインタフェースだけでなく、telnet プロトコルを使用して、ネットワーク経由で利用することも可能である。telnet で接続すると、シリアルの入出力を引き継いで実行する。切断すると、元のシリアルに入出力を戻す。ただし、TCP のノンブロッキングコールを組込む必要がある。

また、クライアントとしては以下の機能が提供されており、必要に応じて組み込むクライアントを選択できる。

[1] TCP エコークライアント・タスク（tcp_echo_cli.c）
[2] UDP エコークライアント・タスク（udp_echo_cli.c）
[3] TCP ディスカードクライアント・タスク（tcp_discard_cli.c）
[4] UDP ディスカードクライアント・タスク（udp_discard_cli.c）
[5] ping（ping.c）

9. 4. 1 Makefile の設定

(1) 組込む機能の選択

[1] TCP の受信ウィンドバッファの省コピー機能

サンプルアプリケーションで、TCP の受信ウィンドバッファの省コピー機能を組込む場合は、

```
TCP_CFG_RWBUF_CSAVE_ONLY = true
```

または、

```
TCP_CFG_RWBUF_CSAVE = true
```

を選択する。TCP_CFG_RWBUF_CSAVE_ONLY を選択すると、TCP 通信端点を生成する静的 API である TCP_CRE_CEP で、受信ウィンドバッファ rbuf に、メモリアドレスを指定しても、プロトコルスタックは、この指定を無視して、TCP の受信ウィンドバッファの省コピー機能により処理する。また、TCP_CFG_RWBUF_CSAVE を選択すると、TCP_CRE_CEP で、受信ウィンドバッファ rbuf に、NADR（NULL）を指定したときのみ、プロトコルスタックは TCP の受信ウィンドバッファの省コピー機能により処理する。

[2] TCP の送信ウィンドバッファの省コピー機能

サンプルアプリケーションで、TCP の送信ウィンドバッファの省コピー機能を組込む場合は、

```
TCP_CFG_SWBUF_CSAVE_ONLY = true
```

または、

```
TCP_CFG_SWBUF_CSAVE = true
```

を選択する。TCP_CFG_SWBUF_CSAVE_ONLY を選択すると、TCP 通信端点を生成する静的 API である TCP_CRE_CEP で、送信ウィンドバッファ rbuf に、メモリアドレスを指定しても、プロトコルスタックは、この指定を無視して、TCP の送信ウィンドバッファの省コピー機能により処理する。また、TCP_CFG_SWBUF_CSAVE を選択すると、TCP_CRE_CEP で、送信ウィンドバッファ rbuf に、NADR（NULL）を指定したときのみ、プロトコルスタックは TCP の送信ウィンドバッファの省コピー機能により処理する。

[3] TCP のノンブロッキングコール

サンプルアプリケーションで、TCP のノンブロッキングコールを組込む場合は、

```
TCP_CFG_NON_BLOCKING = true
```

を選択する。

[4] UDP のノンブロッキングコール

サンプルアプリケーションで、UDP のノンブロッキングコールを組込む場合は、

```
UDP_CFG_NON_BLOCKING = true
```

を選択する。

[5] ITRON TCP/IP API 仕様の TCP の拡張機能

サンプルアプリケーションで、ITRON TCP/IP API 仕様の TCP の拡張機能を組込む場合は、

```
TCP_CFG_EXTENTIONS = true
```

を選択する。ディフォルトで true に設定されている。

[6] ITRON TCP/IP API 仕様の UDP の拡張機能

サンプルアプリケーションで、ITRON TCP/IP API 仕様の UDP の拡張機能を組込む場合は、

```
UDP_CFG_EXTENTIONS = true
```

を選択する。ディフォルトで true に設定されている。

(2) ノンブロッキングコール、ITRON TCP/IP API 仕様の拡張機能と省コピー API の選択

[1] TCP のノンブロッキングコール

サンプルアプリケーションで、TCP のノンブロッキングコールを使用する場合は、

```
USE_TCP_NON_BLOCKING = true
```

を選択する。

[2] ITRON TCP/IP API 仕様の TCP の拡張機能

サンプルアプリケーションで、ITRON TCP/IP API 仕様の TCP の拡張機能を使用する場合は、

```
USE_TCP_EXTENTIONS = true
```

を選択する。ディフォルトで true に設定されている。

[3] 省コピー API

サンプルアプリケーションで、省コピー API を使用する場合は、

```
USE_COPYSAVE_API = true
```

を選択する。

[4] UDP のノンブロッキングコール

サンプルアプリケーションで、UDP のノンブロッキングコールを使用する場合は、

```
USE_UDP_NON_BLOCKING = true
```

を選択する。ただし、コールバックとは同時に使用できない。

[5] ITRON TCP/IP API 仕様の UDP の拡張機能

サンプルアプリケーションで、ITRON TCP/IP API 仕様の UDP の拡張機能を使用する場合は、

```
USE_UDP_EXTENTIONS = true
```

を選択する。ディフォルトで true に設定されている。

[6] UDP のコールバック

サンプルアプリケーションで、UDP のコールバックを使用する場合は、

```
USE_UDP_CALL_BACK = true
```

を選択する。ただし、ノンブロッキングコールとは同時に使用できない。

(3) 共通サーバプログラムの選択

[1] WWW サーバプログラムを使用する場合は、以下の行を有効にする。

```
USE_WWW_SRV = ture
```

[2] UDP エコーサーバプログラムを使用する場合は、以下の行を有効にする。

```
USE_UDP_ECHO_SRV = ture
```

[3] TCP ディスカードサーバプログラムを使用する場合は、以下の行を有効にする。

```
USE_TCP_DISCARD_SRV = ture
```

[4] 送受信タスク同一型の TCP エコーサーバプログラムを使用する場合は、以下の行を有効にする。

```
TCP_ECHO_SRV = tcp_echo_srv1
```

[5] 送受信タスク分離型の TCP エコーサーバプログラムを使用する場合は、以下の行を有効にする。

```
TCP_ECHO_SRV = tcp_echo_srv2
```

[6] シリアル経由のみコンソール入出力を使用する場合は、以下の行を有効にする。

```
USE_DBG_CONS = true
```

[7] シリアルとネットワーク経由のコンソール入出力を使用する場合は、以下の行を有効にする。

```
USE_NET_CONS = true
```

ただし、ノンブロッキングコールを組込んだ時のみ有効である。

(4) 共通クライアントプログラムの選択

[1] TCP エコークライアントプログラムを使用する場合は、以下の行を有効にする。

```
USE_TCP_ECHO_CLI = ture
```

[2] UDP エコークライアントプログラムを使用する場合は、以下の行を有効にする。

```
USE_UDP_ECHO_CLI = ture
```

[3] TCP ディスカードクライアントプログラムを使用する場合は、以下の行を有効にする。

```
USE_TCP_DISCARD_CLI = ture
```

[4] UDP ディスカードクライアントプログラムを使用する場合は、以下の行を有効にする。

```
USE_UDP_DISCARD_CLI = ture
```

[5] PING クライアントプログラムを使用する場合は、以下の行を有効にする。

```
USE_PING = true
```

(5) 共通サーバタスク数の選択

[1] WWW サーバタスク数は以下の行で選択する。ただし最大 2 タスクである。

```
CDEFS := $(CDEFS) -DNUM_WWW_SRV_TASKS=2
```

[2] TCP ECHO サーバタスク数を選択する。ただし以下の条件のとき有効である。

・tcp_echo_srv1.c を選択した。
・省コピー API を使用する。
・ノンブロッキングコールを使用しない。

TCP ECHO サーバタスク数は以下の行で選択する。ただし最大 8 タスクである。

```
CDEFS := $(CDEFS) -DNUM_TCP_ECHO_SRV_TASKS=8
```

(6) 予約 ID 数の選択

[1] TCP/IPv4 受付口予約 ID 数は以下の行で選択する。ただし最大 2 である。

```
CDEFS := $(CDEFS) -DNUM_VRID_TCP_REPS=2
```

[2] TCP/IPv4 通信端点予約 ID 数は以下の行で選択する。ただし最大 4 である。

```
CDEFS := $(CDEFS) -DNUM_VRID_TCP_CEPS=4
```

[3] UDP/IPv4 通信端点予約 ID 数は以下の行で選択する。ただし最大 2 である。

```
CDEFS := $(CDEFS) -DNUM_VRID_UDP_CEPS=2
```

[4] TCP/IPv6 受付口予約 ID 数は以下の行で選択する。ただし最大 2 である。

```
CDEFS := $(CDEFS) -DNUM_VRID_TCP6_REPS=2
```

[5] TCP/IPv6 通信端点予約 ID 数は以下の行で選択する。ただし最大 4 である。

```
CDEFS := $(CDEFS) -DNUM_VRID_TCP6_CEPS=4
```

[6] UDP/IPv6 通信端点予約 ID 数は以下の行で選択する。ただし最大 2 である。

```
CDEFS := $(CDEFS) -DNUM_VRID_UDP6_CEPS=2
```

(7) その他

[1] TCP のセグメントサイズを MSS にする場合は、以下の行を有効にする。

```
CDEFS := $(CDEFS) -DUSE_TCP_MSS_SEG
```

[2] IPv6 MMTU サイズのネットワークバッファを組込む場合は、以下の行を有効にする。

```
CDEFS := $(CDEFS) -DUSE_IPV6_MMTU
```

9. 4. 2 簡易コンソールのコマンド

インターネットサーバ・セットに組み込まれている簡易コンソールのコマンドを以下に示す。

`cf` tinet_app_config.h 等で指定された、コンパイル時コンフィギュレーションを表示する。

`ct <cepid> [<fncd>]`
ペンディングしている TCP 通信端点 <cepid> の処理をキャンセルする。キャンセルする処理は <fncd> で指定する。<fncd> を省略した場合は、全ての処理をキャンセルする。

cu <cepid> [<fncd>]

ペンディングしている UDP 通信端点 <cepid> の処理をキャンセルする。キャンセルする処理は <fncd> で指定する。<fncd> を省略した場合は、全ての処理をキャンセルする。

dc

ネットワークインタフェースが PPP の時、または、シリアルとネットワーク経由のコンソール入出力を使用する時に有効であり、接続を切断する。なお、PPP は参考実装である。

dt <host> [<portno> [<repeat>]]

TCP ディスカードクライアント・タスクを起動し、ディスカードサーバ <host> にディスカードパターンを送信する。<portno> は、ディスカードサーバのポート番号で、省略時（- を指定する）は 9 である。<repeat> は、繰り返し回数で、省略時は 1 である。

dts

TCP ディスカードクライアント・タスクの繰り返し動作を停止する。

du <host> [<portno> [<repeat>]]

UDP ディスカードクライアント・タスクを起動し、ディスカードサーバ <host> にディスカードパターンを送信する。<portno> は、ディスカードサーバのポート番号で、省略時（- を指定する）は 9 である。<repeat> は、繰り返し回数で、省略時は 1 である。

dus

UDP ディスカードクライアント・タスクの繰り返し動作を停止する。

et <host> [<portno> [<repeat>]]

TCP エコークライアント・タスクを起動し、エコーサーバ <host> にエコーパターンを送信する。<portno> は、エコーサーバのポート番号で、省略時（- を指定する）は 7 である。<repeat> は、繰り返し回数で、省略時は 1 である。

ets

TCP エコークライアント・タスクの繰り返し動作を停止する。

eu <host> [<portno>] [<msg> | <repeat>]

UDP エコークライアント・タスクを起動し、エコーサーバ <host> にメッセージを送信する。<portno> は、エコーサーバのポート番号で、省略時（- を指定する）は 7 である。<repeat>（数字）を指定した場合は、定型のメッセージを <repeat> 回繰り返し送信する。<msg>（数字以外）を指定した場合は、メッセージ <msg> を送信する。

eus

UDP エコークライアント・タスクの繰り返し動作を停止する。

i

ネットワークインタフェースが PPP の時に有効である。直接接続の場合は、直ちに LCP を起動して、サーバに接続する。モデム接続の場合は、コンパイル時コンフィギュレーションの MODEM_CFG_PHONE_NUMBER パラメータで指定されているサーバに発呼する。なお、PPP は参考実装である。

if [<addr> <mask>]

ネットワークインタフェースが PPP の時は、IP アドレス、サブネットマスク、ブロードキャストアドレスを出力する。ネットワークインタフェースがイーサ

ネットで、ネットワーク層が IPv4 の時は、[<addr> <mask>] を指定できる。[<addr> <mask>] を指定しなければ、MAC アドレス、IP アドレス、サブネットマスク、ブロードキャストアドレスの出力のみ行う。[<addr> <mask>] を指定した時は、IP アドレスとサブネットマスクを変更した後、MAC アドレス、IP アドレス、サブネットマスク、ブロードキャストアドレスを出力する。<addr> は、IP アドレス、<mask> は、サブネットマスクである。なお、PPP は参考実装である。

na ネットワークインタフェースがイーサネットの時に有効である。IPv4 では ARP キャッシュ、IPv6 では近隣アドレスキャッシュの状態を出力する。

nb ネットワークバッファの統計情報を出力する。

nc ネットワーク統計情報を表示する。

nr [<index> <target> <mask> <gateway>]
ネットワークインタフェースがイーサネットのとき有効である。ネットワーク層が IPv4 の時は、[<index> <target> <mask> <gateway>] を指定できる。[<index> <target> <mask> <gateway>] を指定しなければ、ルーティング表の出力のみ行う。[<index> <target> <mask> <gateway>] を指定した時は、ルーティング表を変更した後、ルーティング表を出力する。<index> は、経路エントリのインデックス、<target> は、目標ネットワークの IP アドレス、<mask> は、目標ネットワークのサブネットマスク、<gateway> は、ゲートウェイの IP アドレスである。

nrl ネットワークインタフェースがイーサネットで、ネットワーク層が IPv6 の時に有効である。ディフォルトルータ・リストを出力する。

nrp ネットワークインタフェースがイーサネットで、ネットワーク層が IPv6 の時に有効である。プレフィックスリストを出力する。

nt TCP 通信端点と TCP 受付口の状態を表示する。

nu UDP 通信端点の状態を表示する。

p <host> [<tmo> [<size>]]
ホスト <host> に ICMP パケットを送信する。<tmo> はタイムアウト値（単位は秒）で、省略時（- を指定する）は 3 秒である。<size> はデータサイズで、指定しない場合は 64 オクテットである。

ps タスクの状態を表示する。

r <tskid> タスク <tskid> の待ち状態を強制的に解除する。

tt <repid> ITRON TCP/IP API の TCP の拡張機能を組込む必要がある。TCP 受付口 <repid> を削除し、対応するサーバを停止する。

tu <cepid> ITRON TCP/IP API の UDP の拡張機能を組込む必要がある。UDP 通信端点 <cepid> を削除し、対応するサーバを停止する。

w <tskid>	タスク <tskid> を起床する。
wtd	ITRON TCP/IP API の TCP の拡張機能を組込む必要がある。TCP ディスカードサーバ・タスクに TCP 受付口と TCP 通信端点を割当て、TCP ディスカードサーバ・タスクを起動する。
wte	ITRON TCP/IP API の TCP の拡張機能を組込む必要がある。TCP エコーサーバ・タスクに TCP 受付口と TCP 通信端点を割当て、TCP エコーサーバ・タスクを起動する。
wtw	ITRON TCP/IP API の TCP の拡張機能を組込む必要がある。WWW サーバ・タスクに TCP 受付口と TCP 通信端点を割当て、WWW サーバ・タスクを起動する。
wue	ITRON UDP/IP API の UDP の拡張機能を組込む必要がある。UDP エコーサーバ・タスクに UDP 通信端点を割当て、UDP エコーサーバ・タスクを起動する。

9. 5 sample1 のネットワーク対応プログラム【sample1n、IPv6/IPv4】の構築

TOPPERS/ASP と TOPPERS/JSP のサンプルプログラム sample1 のネットワーク対応プログラムである。telnet で接続すると、シリアルの入出力を引き継いで実行する。切断すると、元のシリアルに入出力を戻す。

以下に構築方法を述べる。

(1) ASP/JSP コンフィギュレーションスクリプトの実行

それぞれの環境におけるコンフィギュレーションスクリプトを実行する。以下は、TOPPERS/ASP 環境におけるコンフィギュレーションスクリプトの実行例である。

```
$ mkdir NETOBJ
$ cd NETOBJ
$ perl ../configure -T akih8_3069f_gcc
```

(2) TINET コンフィギュレーションスクリプトの実行

それぞれの環境における TINET コンフィギュレーションスクリプトを実行する。この時、アプリケーションプログラム名として sample1n を指定する。以下は、TOPPERS/ASP 環境における TINET コンフィギュレーションスクリプトの実行例である。

```
$ perl ../tinet/tinet_asp_configure -T akih8_3069f_gcc -A sample1n
                                    -i ether -v if_ed -n inet6 -s tcp
```

(3) Makefile の修正

アプリケーション本体（sample1n.c）と TOPPERS/ASP と TOPPERS/JSP の sample1.c を一緒にコンパイル・リンクするため、Makefile を修正する。

[1] TOPPERS/ASP 環境

Makefile の APPL_COBJS に sample1.o を追加する。

```
APPL_COBJS = $(APPLNAME).o sample1.o
```

[2] TOPPERS/JSP 環境

Makefile の UTASK_COBJS に sample1.o を追加する。

```
UTASK_COBJS = $(UNAME).o sample1.o
```

(4) sample1.c の修正

TOPPERS/ASP と TOPPERS/JSP の sample1.c のインクルードファイルの指定

```
#include "sample1.h"
```

の前に、以下のインクルードファイルを追加する。

```
#include "sample1n.h"
```

(5) tinet_app_config.h の設定

IPv4 の場合、IP アドレス、サブネットマスク、ディフォルトゲートウェイを指定する。

9. 6 最小構成サーバ【minsv、IPv4】の構築

WWW サーバ・タスクと TCP エコーサーバ・タスクのみからなる最小構成のサーバである。H8/3069F が内蔵している RAM（16K バイト）と ROM（512K バイト）に収まり、外部メモリは不要である。現在は、品川通信計装サービス製 NKEV-010H8（TOPPERS/JSP リリース 1.4.2 のみ）と秋月電子通商製 H8/3069F（TOPPERS/JSP リリース 1.4.1 以降と TOPPERS/ASP）のシステムに対応している。

各システム依存部の Makefile.config の「実行環境の定義」で、

```
# ROM化 外部RAM未使用
#DBGENV := INMEM_ONLY
```

を有効にして、コンパイル・リンクする。

10. 謝辞

本 TCP/IP プロトコルスタックは、次の組織の皆様の御支援により研究・開発を行いました。関係各位に感謝いたします。

(1) 財団法人道央産業技術振興機構様

[1] 事業名（実施年度）
高度技術開発委託事業（平成 12 年度）
[2] テーマ名
組込み型制御システム用 TCP/IP プロトコルスタックの開発

(2) 株式会社 NTT ドコモ北海道苫小牧支店様
(3) 経済産業省東北経済産業局（委託先管理法人: 財団法人みやぎ産業振興機構）様

[1] 事業名（実施年度）
地域新生コンソーシアム研究開発事業（平成 14、15 年度）
[2] テーマ名
組込みシステム・オープンプラットホームの構築とその実用化開発

(4) 宮城県産業技術総合センター様
(5) TOPPERS プロジェクト様
(6) 株式会社ヴィッツ様
(7) 財団法人電気・電子情報学術振興財団様

[1] 第 6 回 LSI IP デザイン・アワード IP 受賞（2004 年、平成 16 年 5 月 20 日）
オープンソースの組込みシステム用 TCP/IP プロトコルスタック : TINET
[2] 第 7 回 LSI IP デザイン・アワード IP 受賞（2005 年、平成 17 年 5 月 19 日）
組込みシステム用 IP バージョン 6 対応 TCP/IP プロトコルスタック : TINET-1.2

(8) 株式会社北斗電子様
(9) 有限会社品川通信計装サービス様
(10) 北海道立工業試験場様

[1] 事業名（実施年度）
重点領域特別研究（平成 17、18 年度）
[2] テーマ名
組込みシステム向けネットワーク接続ソフトウェア群の開発

11. ライセンス

TINET は FreeBSD を元に開発を行ったため、TINET を含むソフトウェアを、他のソフトウェア開発に使用できない形で再配布する場合（TOPPERS ライセンス (3) に規程されている形態）は、TOPPERS ライセンス (3) の (b) の報告だけでは不十分で、(a) による方法が必要である。

以下に示す TOPPERS、FreeBSD および FreeBSD へのソフトウェアの寄贈者のライセンス規定に従って、再配布に伴うドキュメント（利用者マニュアルなど）に、ライセンス表示を行うと。

(1)FreeBSD

```
/*
 * Copyright (c) 1980, 1986, 1993
 *     The Regents of the University of California.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *     This product includes software developed by the University of
 *     California, Berkeley and its contributors.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */
```

(2)KAME

```
/*
 * Copyright (C) 1995, 1996, 1997, and 1998 WIDE Project.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the project nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE PROJECT AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE PROJECT OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */
```

(3)イーサネット・ディバイスドライバ

```
/*
 * Copyright (c) 1995, David Greenman
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice unmodified, this list of conditions, and the following
 *    disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * $FreeBSD: src/sys/i386/isa/if_ed.c,v 1.148.2.4 1999/09/25 13:08:18 nyan Exp $
 */

/*
 * Device driver for National Semiconductor DS8390/WD83C690 based ethernet
 *   adapters. By David Greenman, 29-April-1993
 *
 * Currently supports the Western Digital/SMC 8003 and 8013 series,
 *   the SMC Elite Ultra (8216), the 3Com 3c503, the NE1000 and NE2000,
 *   and a variety of similar clones.
 *
 */
```

(4)/usr/sbin/ppp

```
/*
 *                      User Process PPP
 *
 *          Written by Toshiharu OHNO (tony-o@iij.ad.jp)
 *
 *   Copyright (C) 1993, Internet Initiative Japan, Inc. All rights reserverd.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that the above copyright notice and this paragraph are
 * duplicated in all such forms and that any documentation,
 * advertising materials, and other materials related to such
 * distribution and use acknowledge that the software was developed
 * by the Internet Initiative Japan, Inc.  The name of the
 * IIJ may not be used to endorse or promote products derived
 * from this software without specific prior written permission.
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
 * WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 */
```

(5)/usr/sbin/pppd

```
/*
 * main.c - Point-to-Point Protocol main module
 *
 * Copyright (c) 1989 Carnegie Mellon University.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that the above copyright notice and this paragraph are
 * duplicated in all such forms and that any documentation,
 * advertising materials, and other materials related to such
 * distribution and use acknowledge that the software was developed
 * by Carnegie Mellon University.  The name of the
 * University may not be used to endorse or promote products derived
 * from this software without specific prior written permission.
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
 * WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 */
```

(6)TINET と TOPPERS

```
/*
 *  TINET (TCP/IP Protocol Stack)
 *
 *  Copyright (C) 2001-2006 by Dep. of Computer Science and Engineering
 *                   Tomakomai National College of Technology, JAPAN
 *
 *  上記著作権者は，以下の (1)～(4) の条件か，Free Software Foundation
 *  によって公表されている GNU General Public License の Version 2 に記
 *  述されている条件を満たす場合に限り，本ソフトウェア（本ソフトウェア
 *  を改変したものを含む．以下同じ）を使用・複製・改変・再配布（以下，
 *  利用と呼ぶ）することを無償で許諾する．
 *  (1) 本ソフトウェアをソースコードの形で利用する場合には，上記の著作
 *      権表示，この利用条件および下記の無保証規定が，そのままの形でソー
 *      スコード中に含まれていること．
 *  (2) 本ソフトウェアを，ライブラリ形式など，他のソフトウェア開発に使
 *      用できる形で再配布する場合には，再配布に伴うドキュメント（利用
 *      者マニュアルなど）に，上記の著作権表示，この利用条件および下記
 *      の無保証規定を掲載すること．
 *  (3) 本ソフトウェアを，機器に組み込むなど，他のソフトウェア開発に使
 *      用できない形で再配布する場合には，次の条件を満たすこと．
 *    (a) 再配布に伴うドキュメント（利用者マニュアルなど）に，上記の著
 *        作権表示，この利用条件および下記の無保証規定を掲載すること．
 *  (4) 本ソフトウェアの利用により直接的または間接的に生じるいかなる損
 *      害からも，上記著作権者およびTOPPERSプロジェクトを免責すること．
 *
 *  本ソフトウェアは，無保証で提供されているものである．上記著作権者お
 *  よびTOPPERSプロジェクトは，本ソフトウェアに関して，その適用可能性も
 *  含めて，いかなる保証も行わない．また，本ソフトウェアの利用により直
 *  接的または間接的に生じたいかなる損害に関しても，その責任を負わない．
 *
 *  @(#) $Id: tinet.d,v 1.5 2009/12/24 05:41:48 abe Exp abe $
 */
```