M3S-T2-Tiny V.1.01
M3S-T2-30 V.1.01
M3S-T2-308 V.1.01

R8C/Tiny, M16C/Tiny, H8/Tiny シリーズ用　TCP/IP ライブラリ
M16C/62 グループ用　TCP/IP ライブラリ
M32C/83 グループ用　TCP/IP ライブラリ

ユーザーズマニュアル

Rev. 2.00
04.08.01
RJJ10J0629-0200Z

ルネサス テクノロジ
www.renesas.com

[目次]

1. はじめに

1.1 概要

Tiny な TCP/IP ライブラリ M3S-T2-xxx(以下、T2 と呼びます)は、以下のシリーズ、グループのマイクロコンピュータに対応したネットワークソフトウェアライブラリです。本マニュアルでは T2 を使用してアプリケーションプログラムを作成するための共通情報を提供します。機種に依存する情報(開発環境など)は、各リリースノートで提供します。

M3S-T2-Tiny ：R8C/Tiny シリーズ, M16C/Tiny シリーズ, H8/Tiny シリーズ
M3S-T2-30 ：M16C/62 グループ
M3S-T2-308 ：M32C/83 グループ
※ 本文中の xxx には、シリーズ名、機種名などが入ります。

T2 の位置付けを図 1 に示します。

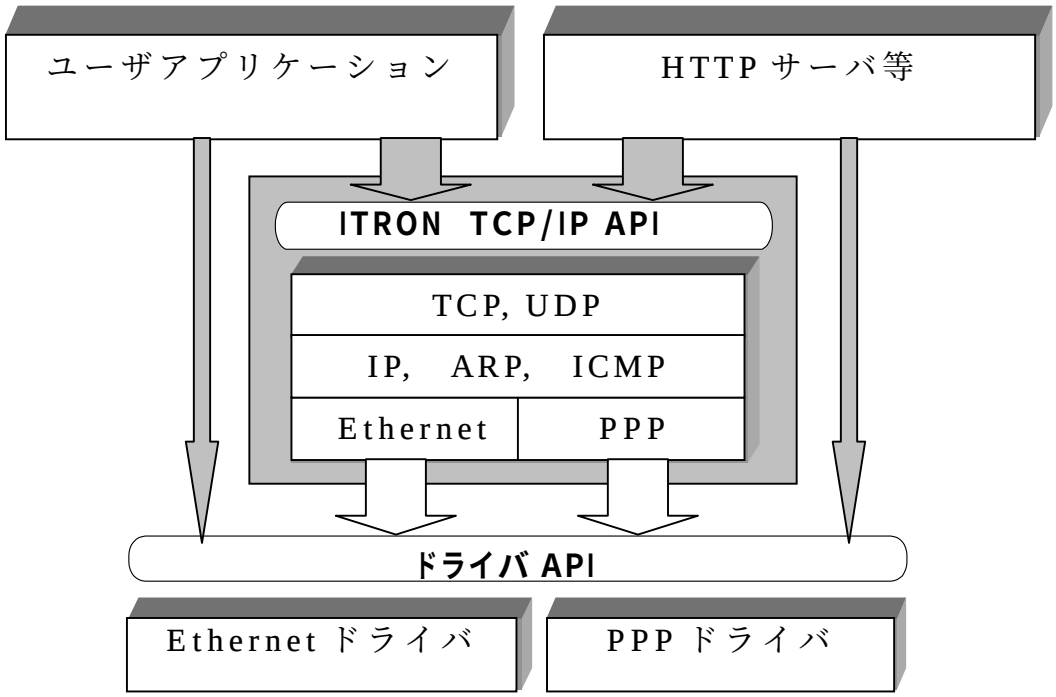

図 1 T2 の位置付け

- 図 1では、ネットワークミドルウェアに係わるソフトウェアの全体像を示し、その中の T2 の位置付けを表わしています。中央のグレーの背景で示した部分が T2 です。
  矢印は、関数呼び出しを表わしています。
- T2 は、Ethernet ドライバ、または PPP ドライバとアプリケーションプログラムの間でプロトコル処理を行い、ネットワークに対してデータの送受信を行います。
- T2 には、プロトコル処理として TCP、UDP、IP、ICMP、ARP、PPP が含まれています。
  ユーザアプリケーションや上位プロトコルのプログラムから TCP や UDP の通信を行う場合、T2 で規定する API（ITRON TCP/IP API に準拠）を利用してライブラリ内の関数を呼び出します。IP、ICMP、ARP、PPP については、TCP と UDP の下に位置するプロトコルであり、これらのプロトコル処理関数をユーザプログラムから直接、呼び出すことはありません。
- T2 では、データリンク層/物理層として Ethernet と PPP をサポートします。これらの内、

Ethernet コントローラやシリアル I/O 等、ご使用になるハードウェアに依存する部分については、ドライバとして T2 のライブラリから切り離しています。

ドライバ API の仕様を規定していますので、お使いになるハードウェアに合わせてドライバを作成してください。ドライバ API の仕様は、ドライバのユーザーズマニュアルに記載しています。

※ドライバのユーザーズマニュアルは、本マニュアルとは別に、本製品パッケージに付属しています。

なお T2 を使用してプログラムを作成する場合、ドライバ内の初期設定関数および終了関数を直接、呼び出していただく必要があります（図 1ではグレーの矢印で示しました)。これらの API 関数仕様については、ドライバのユーザーズマニュアルだけで無く、本 T2 のマニュアル中でも説明しています。

- 本製品パッケージには、ドライバのサンプルプログラムが付属しています。お客様独自のドライバを作成する際にご参照ください。
- T2 のライブラリでは、リアルタイム OS の機能を使用しておりません。

1.2 TCP/IP の基礎知識

本節では、T2 をご使用いただく上で必要となる TCP/IP の基礎知識について説明します。T2 では、組み込みシステムに適した API を採用しており、BSD や Linux のような一般の TCP/IP とは異なる部分があります。これらについては後節で説明しますが、その際、本節で説明する一般の TCP/IP と対比できるようにしました。

なお本節で説明する内容は、本マニュアルをお読みいただく上で必要となる最小限の内容に限定しています。TCP/IP についてより詳しい説明が必要な場合は、市販の書籍をご活用いただきますようお願いします。

1.2.1 コネクションについて

TCP では、仮想的な通信路を作って通信します。この仮想的な通信路のことを「コネクション」と呼びます。そして通信相手とコネクションを持つことを「コネクションを確立する」と言います。1 つのプログラム（通信するコンピュータ上で稼動するプログラムのこと）は、複数のコネクションを持つことができ、これにより複数のコンピュータプログラムと通信できます。ただし 1 つのコネクションで通信できる相手は 1 つだけです。

1 つのコネクションを確立するには、通信する相手との間（すなわち 2 者間）で交渉する必要があります。そして 1 つのコネクションが確立できると、双方向で通信できるようになります（送信用、受信用に異なる 2 つのコネクションを用意する必要はありません）。このコネクション確立のための手続きを「コネクションを要求する」と表現します。

これに対して、UDP では通信の際にコネクションを確立しません。相手の通信状態を確認せずに、一方的に通信を試みるため、コネクションの確立や、正常に通信できたか確認する際の交渉は行いません。

1.2.2 クライアント/サーバモデルについて

TCP では、前記のコネクションを確立する際の交渉において、クライアント/サーバモデルを採用しています。したがって通信する両者の内、どちらか一方が「クライアント」、もう片方が「サーバ」の役目を担います。コネクションを確立する際、**サーバ側は、通信相手を特定せずに、誰かが自分に対してコネクションを要求してくるのを待ちます**。これに対して、**クライアント側は、通信相手となるサーバのアドレスを指定し、通信相手を特定した上でコネクションを要求します**。

この両者の手続きによってコネクションを確立した後は、クライアント/サーバに係わらず、自由に送受信できるようになります。

ただし一般的には、TCP/IP の通信プログラム自体をクライアント/サーバモデルで実装することが多く、その場合、サーバ側はクライアント側からの要求を待って、クライアント側に情報を返信するように実装することになります。

1.2.3 IP アドレスとポート番号

IP アドレスとポート番号は、TCP/IP および UDP/IP の通信において、通信相手や自分を指し示すためのアドレス情報です。

・IP アドレスは、通信対象の通信ポートを示すユニークなアドレスです。
・ポート番号は、通信対象のアプリケーションプログラムを示す番号です。

IP アドレスは、インターネットに接続しないローカルなネットワークの場合を除き、全世界でユニークな番号である必要があります（IP アドレスを割り当てる専門機関があります）。

ポート番号は、同じコンピュータ上で稼動する全ての通信プログラムにおいて、ユニークに設定する必要があります。

1.2.4 ソケットについて

一般の TCP/IP では、通信においてソケットと呼ばれる API を利用します。TCP/IP および UDP/IP では、パケット通信において複雑な処理が行われていますが、アプリケーションプログラムを作成する際は、初期設定、コネクション要求、送信、受信、終了処理等の API が用意されており、複雑なパケット処理を意識することなく、C 言語のファイル入出力に似た感覚で通信処理をプログラミングできます。API では、socket と呼ばれる抽象的なデータ構造（構造体）を用います。

1.3 ITRON TCP/IP API 仕様の基礎知識

本節では、T2 において採用した、ITRON TCP/IP API 仕様（以下、ITRON 仕様と呼びます）の基礎知識について説明します。ただし本節で説明する内容は、本書をお読みいただく上で必要になる最小限の内容に限定しています。ITRON TCP/IP API 仕様についてより詳しい説明が必要な場合は、下記のホームページに掲載されている(社)トロン協会 ITRON 専門委員会 Embedded TCP/IP 技術委員会発行の仕様書をご活用いただきますようお願いします。

URL: http://www.assoc.tron.org/itron/home-j.html

1.3.1 コネクションの扱い

ITRON TCP では、前節のコネクションのことを日本語で「接続」と表現します。そしてコネクションを要求する手続きを「接続要求」「接続要求待ち」と言います。この両者の違いについては、後で説明します。

1.3.2 クライアント/サーバモデルの扱い

ITRON TCP でも、一般の TCP/IP と同様に、接続の手続きにおいてクライアント/サーバモデルを採用しています。しかし ITRON TCP の仕様書中では、クライアント/サーバ という用語は用いません。その代わりに、接続に関する API として、サーバ用に「接続要求待ち（受動オープン）」、クライアント用に「接続要求（能動オープン）」の 2 つを用意して区別しています。すなわち「接続を要求するのがクライアント」で、「その接続要求を待つのがサーバ」です。

1.3.3 受付口および通信端点について

ITRON TCP では、通信において socket のデータ構造は用いず、その代わりに「受付口」「通信端点」と呼ばれる構造体を用います。これは、組み込みシステムにおいて求められる以下の性質を考慮した結果です。

1） バッファのためのメモリ領域や、データのコピーの回数が最小限であること。
2） できる限りプロトコルスタック内部で動的なメモリ管理を使う必要がないこと。
3） 非同期インターフェイスないしはノンブロッキングコールをサポートすること。
4） API ごとのエラーの詳細がわかることが望ましい。

（(社)トロン協会 ITRON 専門委員会 Embedded TCP/IP 技術委員会発行の仕様書より抜粋）

1.4 用語の定義

1.4.1 受付口

受付口は、TCP の通信において API で使用する構造体です。具体的には（サーバ側が）通信相手（クライアント）からの接続要求を待つための API（関数 tcp_acp_cep）で使用します。受付口は、プログラム全体で複数持つことができ、プログラム全体で 1 つの構造体配列として定義します。

※ T2 では受付口をひとつだけ持つことが出来ます。

接続要求待ちの API では、使用する受付口をパラメータで指定しますが、その際、構造体のポインタを指定するのでは無く、1 から始まるユニークな ID 番号で指定します。この ID 番号はプログラマ自身で定義するのでは無く、受付口の構造体配列を定義する際に T2 が先頭から順に 1、2…の ID 番号を割り当てます。したがってプログラマは、構造体配列の先頭で定義した受付口を 1 番、2 番目の受付口を 2 番として、それぞれの ID 番号が何番であるか把握できます。

トロン協会が規定する仕様では、受付口はメンバとして属性、自局の IP アドレス、自局のポート番号を持っています。

※ T2 では受付口を指定する場合、1 を ID 番号として指定します。

1.4.2 通信端点

通信端点は、TCP と UDP の通信において各種 API で使用する構造体です。通信端点は（受付口と同じように）構造体配列として定義し、先頭の構造体を 1 番とする ID 番号で表わします。この通信端点 ID を使用して、通信相手との接続の確立、データの送受信、接続の切断を行います。

トロン協会が規定する仕様では、TCP の通信端点はメンバとして属性、送信ウィンドウの先頭アドレス、送信ウィンドウのサイズ、受信ウィンドウの先頭アドレス、受信ウィンドウのサイズ、コールバックルーチンのアドレスを持っています。

TCP では、接続が完全に切断されている状態の通信端点を未使用状態と呼びます。

トロン協会が規定する仕様では、UDP の通信端点はメンバとして属性、自局の IP アドレス、自局のポート番号、コールバックルーチンのアドレスを持っています。

※ T2 では、通信端点を指定する場合、1 を ID 番号として指定します。

1.4.3 タイムアウト

API をコールすると処理に時間がかかり、関数のなかで待ち状態になることがあります。この時、指定した時間（タイムアウト時間）を過ぎても処理が終了しない場合、処理をキャンセルして API からリターンすることができます。この現象をタイムアウトと呼びます。

1.4.4 ノンブロッキングコール、コールバック

API のなかで待ち状態となった場合、処理を継続したまま API からリターンすることをノンブロッキングコールと呼びます。そして API からリターンした後、継続された処理の終了をアプリケーションに通知する機能をコールバックと呼びます。

※ T2 の TCP では、これらの機能をサポートしておらず、ブロッキングコールのみサポートしています。ブロッキングコールでは、処理が終わるまで API のなかで待ち状態になることがあります。

※ T2 の UDP では、これらの機能をサポートしています。

1.4.5 ポーリング

タイムアウト時間を 0 に設定したタイムアウト処理と同じ動作をします。

※ T2 の TCP では、この機能をサポートしていません。本来ポーリングの機能をサポートする API にポーリングを指定した場合、動作は不明です。

※ T2 の UDP では、この機能をサポートしています。

1.4.6 ペンディング

API のなかで待ち状態になっていることをペンディグしていると呼びます。

1.4.7 MSS（Maximum Segment Size）

MSS は、TCP のプロトコル処理において、一回に送信できるセグメントの最大サイズのことです。システムで一意に決定します（※T2 でも MSS を定義する必要があります）。アプリケーションプログラムから API で渡された送信データが MSS より大きなサイズの場合、TCP のプロトコル処理部は MSS のサイズで分割して複数のセグメントで送信します。分割の対象となるのは、TCP ヘッダを除いた純粋なデータ部分です。

TCP では、通信の初期段階で通信相手と自局の MSS 情報を交換し、両者の内、小さい MSS で送信するようにします。このときの MSS は、TCP のヘッダのオプション部に置かれます。したがってこの機能を「ヘッダオプションの MSS」と表現します。

※ T2 はこの機能をサポートしています。

1.4.8 IP フラグメント

IP プロトコル処理のオプション機能の１つで、ルータが MTU（ネットワーク最大転送単位）の小さな通信路を使用する場合、MTU よりサイズが大きなデータを送信するために、データを MTU 以下のサイズに分割して送信し、最終的に受信した側で再構築する仕組みのことです。例えば Ethernet では、MTU が 1500bytes のため、これ以上のデータを送信する場合、分割が必要となります。

※ T2 では、この機能をサポートしていません。従って、MTU の小さな通信路を使って通信する可能性のある場合は、MSS を MTU より小さなサイズに設定する必要があります。

1.4.9 受信ウィンドウ

受信したデータを溜めておくためのバッファ領域のことです。TCP では、通常、セグメントを送信する度に相手からの確認応答を待ちますが、毎回、確認応答を待っていたのではオーバヘッドが生じます。したがって通信の際に TCP のヘッダを使って受信側から受信ウィンドウの残りサイズを送信側に通知し、送信側は受信ウィンドウが一杯になるまでは、確認応答を待たずにセグメントを連続して送信するようにします。

通信中に受信ウィンドウが一杯になると、その通知を受けた送信側は送信を止め、受信側から受信ウィンドウが空いたことを通知してくれるのを待ちます。

※ T2 では、受信側の確認応答を待ってから次の送信を再開します。そのため、オーバヘッドが生じます。

1.4.10 PPP（Point to Point Protocol）

シリアル回線上に IP プロトコルを載せるためのプロトコルです。付随したプロトコルとして回線接続後のユーザ認証手順があります。

1.4.11 PAP（Password Authentication Protocol）

PPP 接続時の認証プロトコルのひとつです。発信側が送信したパスワードの正誤を認証します。パスワードは、サーバ側では暗号化されていますが、通信回線上では暗号化されません。同様のプロトコルに CHAP がありますが、こちらはパスワードを暗号化するため、PAP に比べ信頼性が高いです。

※ T2 では CHAP をサポートしておりません。

1.4.12 ICMP（Internet Control Message Protocol）

IP パケットの配送に伴うエラーを通知したり、各ホストの IP の状態を調べたり通知したりするためのプロトコルで、IP と常に一緒に使われます。

1.4.13 ARP（Address Resolution Protocol）

IP アドレスから MAC アドレス（例えば、Ethernet アドレス）に変換するためのプロトコルです。

1.5　プログラム開発手順

T2 を使用したアプリケーションプログラムの開発フローを図 2に示します。

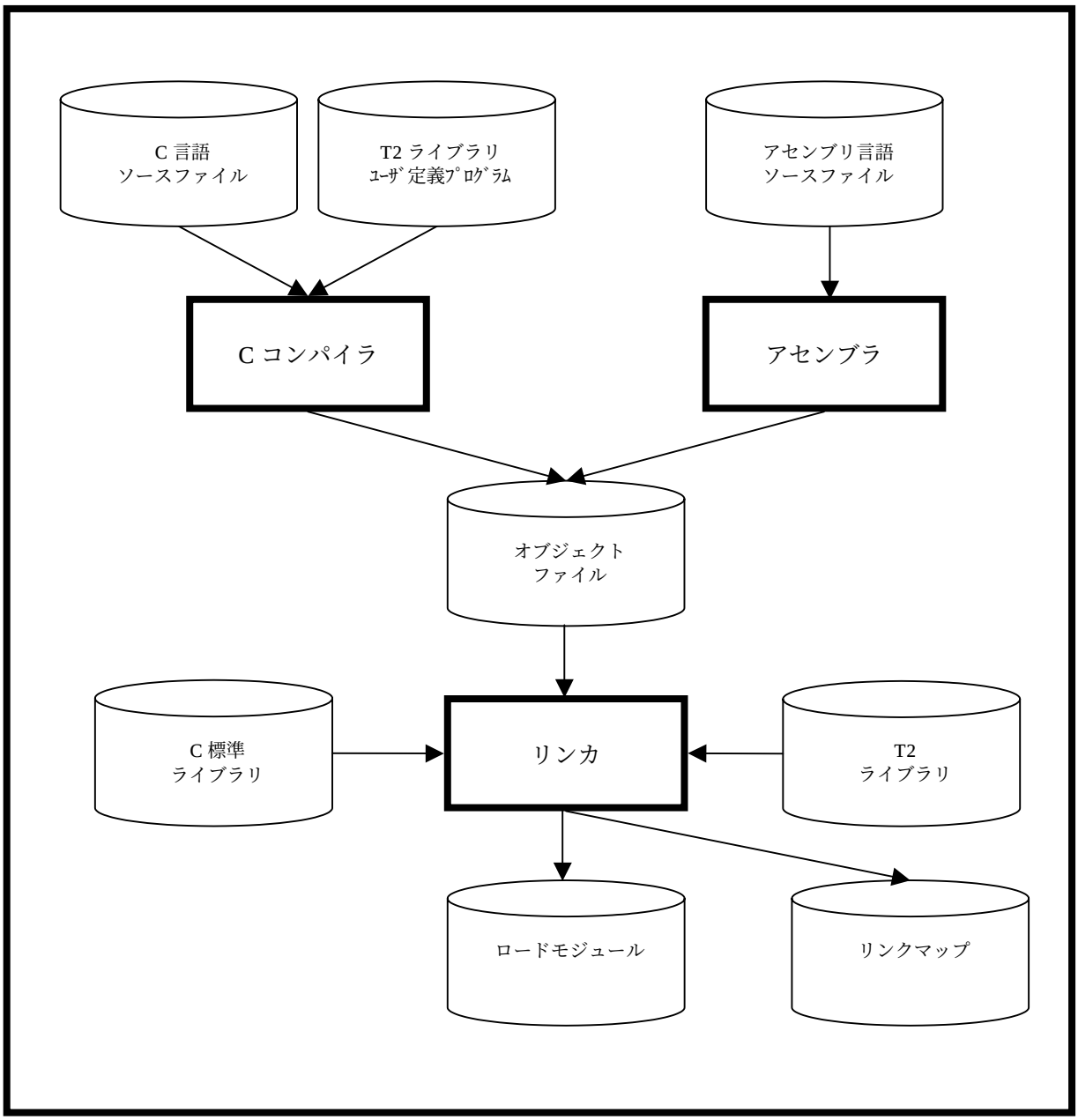

図 2　アプリケーションプログラムの開発フロー

2. T2 の概要

2.1 製品構成

本製品には、**表 1**のものが含まれています。

表 1 T2 製品構成

	内容
T2 ライブラリ	
T2_ether_xxx.lib	ライブラリファイル**(Ethernet** 対応**)**
T2_ppp_xxx.lib	ライブラリファイル**(PPP** 対応**)**
itcpip.h	**T2** ヘッダファイル
config_tcpudp.c.tpl	**T2** コンフィグレーションファイルのテンプレート
サンプルドライバ	
eth_drv.c	**Ethernet** ドライバソースファイル
eth_drv.h	**Ethernet** コントローラのレジスタマップの定義
ppp_drv.c	**PPP** ドライバソースファイル
ppp_drv.h	**PPP** ドライバ用ヘッダファイル
サンプルプログラム	
main.c	メインプログラム
config_tcpudp.c	**T2** コンフィグレーションファイル
intprg.c	**TCP/IP** 周期処理プログラム

2.1.1 T2 ライブラリ

● ライブラリファイル（バイナリファイル）

API 関数や各種プロトコル処理のプログラムを含みます。アプリケーションプログラムと共にリンクして使用します。

Ethernet をお使いになる場合は T2_ether_xxx.lib を、PPP の場合は T2_ppp_xxx.lib をお使いください。

● ヘッダファイル（**itcpip.h**）

T2 の API のプロトタイプ宣言やマクロ定義等が記述されています。T2 の API を呼び出すアプリケーションプログラムでは、このヘッダファイルをインクルードしてください。

● コンフィグレーションファイルのテンプレート（**config_tcpudp.c.tpl**）

T2 の通信端点、受付口等、通信に必要なデータを定義するファイルのテンプレートです。T2 では、このファイルをコンフィグレーションファイルと呼びますが、実態は前記のデータ定義を含む C ソースファイルのため、ユーザのアプリケーションプログラムと一緒にコンパイル・リンクしてください。

config_tcpudp.c.tpl は、必要なデータ定義を全て含んでいますので、各データの中身をユーザのアプリケーションに従って書き換えてください（※変更せずにご利用いただくファイルではありません）。

2.1.2 サンプルドライバ

T2 に対応した Ethernet と PPP のサンプルドライバです。

Ethernet ドライバは、イーサネットフレームの送受信処理等のデバイスに依存した処理を行います。本ドライバは、RealTek 社の Ethernet コントローラ RTL8019AS に対応しています。

PPP ドライバは、モデムの制御、ダイヤルアップ、シリアル I/O のデータ送受信等のデバイスに依存した処理を行います。

2.1.3 サンプルプログラム

アプリケーションプログラム（Telnet 使用）のサンプルソースを用意しています。アプリケーションプログラムの動作方法は、6章を参照ください。

2.2 ライブラリの概略仕様

T2 の概略仕様を表 2に示します。 本ライブラリは TCP、UDP、IP、ICMP、ARP、PPP のプロトコルをサポートしています。API(Application Program Interface)は、ITRON TCP/IP API 仕様に準拠しています。

表 2 プロトコルごとの概略仕様

プロトコル	項目	仕様
TCP	API	ITRON TCP/IP API 仕様に準拠
	ノンブロッキングコール	未サポート (ブロッキングコールのみサポート)
	コールバック機能	未サポート
	最大サイズ*	TCP データセグメント長(データ長) :1480(1460)Byte
	TCP ヘッダオプション	MSS のみサポート 受信：MSS 以外のヘッダオプションは無視 送信：MSS 以外のヘッダオプションは不可
UDP	API	ITRON TCP/IP API 仕様に準拠
	ノンブロッキングコール	サポート
	コールバック機能	サポート
	キューイング	未サポート
	最大サイズ*	UDP データグラム長(データ長): 1480(1472)Byte
IP	バージョン	IPv4 (バージョン 4)のみ
	フラグメント	未サポート 受信：フラグメント化されたデータグラムは破棄 送信：データグラムのフラグメント化は不可
	ヘッダオプション	未サポート 受信：ヘッダオプションが含まれるデータグラムは破棄 送信：ヘッダオプションは不可
	最大サイズ*	IP データグラム長(データ長) ：1500(1480)Byte
ICMP	メッセージタイプ	エコー応答のみサポート 受信：エコー要求のみ、他のメッセージは破棄 送信：エコー応答のみ
ARP	キャッシュエントリ	ユーザ定義による
	キャッシュ保持期間	約 10 分
PPP	サーバ/クライアント	クライアント機能のみサポート、サーバ機能は未サポート
	最大サイズ*	PPP フレーム長(データ長)：1504(1500)Byte
	認証方式	PAP
	圧縮オプション	圧縮関連オプション・未サポート プロトコルフィールド圧縮、 アドレスと制御フィールド圧縮、 TCP/IP ヘッダ圧縮の設定要求を拒否

※TCP と UDP と IP と PPP の最大サイズはドライバの送受信バッファサイズに依存します。

3. T2 のコンフィグレーションファイル

TCP 受付口、TCP 通信端点、UDP 通信端点、IP ヘッダ関連、TCP 関連オプション、自局の設定（Ethernet、PPP）を定義します。これらの定義については、T2 の初期化よりも前に設定するのであれば、アプリケーションプログラム中でも可能です。ただし、使用状況にかかわらず、全ての項目を設定する必要があります。

また、本章で説明するユーザ定義の項目には、RAM 領域に配置しなければならないものと、RAM/ROM 領域のどちらに配置しても構わないものがあります。次の３つの変数については、必ず RAM 領域に配置してください。

UW	_tcp_initial_seqno
TCPUDP_ENV	tcpudp_env
UB	_myethaddr[6]

本製品のテンプレートファイル config_tcpudp.c.tpl を用いて、次の各定義について説明します。各図中の赤色の文字は、テンプレートファイルには記述されていません。

3.1 TCP 受付口の定義

```
/*************************************************************************/
/**************************       TCP 関連の定義       **************************/
/*************************************************************************/

/***  TCP 受付口の定義 (ポート番号のみ設定要) ***/
const T_TCP_CREP tcp_crep[] = {
 /* { 受付口属性, {自局の IP アドレス, 自局のポート番号}}} */
    { 0x0000, { 0, 23 }},        /*TCP 受付口 ID:1、ポート番号:23 */
};
const H __tcprepn = sizeof(tcp_crep)/sizeof(T_TCP_CREP);  /* TCP 受付口の総数   */
```

図 3　TCP 受付口の定義(config_tcpudp.c.tpl より抜粋)

TCP 受付口は、T_TCP_CREP 構造体（第**4**章参照）を用いて静的に生成します。図 3に、TCP 受付口の定義の記述例を示します。

現在、これら３つのフィールドの内、自局のポート番号のみ設定が必要です。

属性は、未サポートのため設定値は無効となります。

自局の IP アドレスについては、T2 の初期化において、変数 tcpudp_env (図 8、図 9参照)に設定された自局の IP アドレスが設定されるため、TCP 受付口の定義で設定された自局の IP アドレスは無効となります。

設定できる TCP 受付口は１つです。

変数__tcprepn には、TCP 受付口の総数が自動的に設定されるため、変更の必要はありません。

本定義により､指定したポート番号での TCP を用いた接続要求待ちからの接続が可能となります。また、TCP を使用しない場合も定義してください。

3.2 TCP 通信端点の定義

```
/*************************************************************************/
/*************************       TCP 関連の定義       *********************/
/*************************************************************************/
                    :
                    :
/***  TCP 通信端点の定義 (受信ウィンドウサイズのみ設定要)***/
const T_TCP_CCEP tcp_ccep[] = {
 /*     {TCP 通信端点属性,
         送信用ウィンドウバッファの先頭, 送信用ウィンドウバッファのサイズ,
         受信用ウィンドウバッファの先頭, 受信用ウィンドウバッファのサイズ,
         コールバックルーチン}
 */
        { 0, 0, 0, 0, 64, 0 }           /* TCP 通信端点 ID:1 */
};
const H __tcpcepn = sizeof(tcp_ccep)/sizeof(T_TCP_CCEP);  /* TCP 通信端点の総数 */
```

図 4　TCP 通信端点の定義（config_tcpudp.c.tpl より抜粋）

TCP 通信端点は、T_TCP_CCEP 構造体（第**4**章参照）を用いて静的に生成します。図 4に、TCP 通信端点の定義の記述例を示します。

1 つの TCP 通信端点の定義は、以下で示すように 6 つのフィールドで構成されています。

{ 属性 , 送信ウィンドウの先頭アドレス , 送信ウィンドウのサイズ ,
 受信ウィンドウの先頭アドレス , 受信ウィンドウのサイズ ,
 コールバックルーチンアドレス }

現在、これら 6 つのフィールドの内、受信ウィンドウのサイズのみ設定が必要です。

属性、送信ウィンドウおよびそのサイズ、コールバックルーチンアドレスは未サポートのため、設定値は無効となります。

受信ウィンドウの先頭アドレスのフィールドについては、T2 の初期化で自動的に割り当てられるため、コンフィグレーションファイルでの設定値は無効です。

設定できる TCP 通信端点は、 1 つです。

変数__tcpcepn には、TCP 通信端点の総数が自動的に設定されるため、変更の必要はありません。

本定義により、TCP を用いた接続の確立、データの送受信、接続の切断が可能となります。また、TCP を使用しない場合も定義してください。

3.3 UDP 通信端点の定義

```
/*************************************************************************/
/*************************       UDP 関連の定義       *********************/
/*************************************************************************/
/***  UDP 通信端点の定義  ***/
const T_UDP_CCEP udp_ccep[] = {
    /* ポート番号のみ設定 */
    { 0x0000, { 0, 1365 }, 0x00000000 },/* ID:1、ポート番号:1365 */
};
const H __udpcepn = (sizeof(udp_ccep)/sizeof(T_UDP_CCEP));
```

図 5　UDP 関連の設定（config_tcpudp.c.tpl より抜粋）

UDP 通信端点は、T_UDP_CCEP 構造体を用いて静的に生成します。図 5に、UDP 関連の定義の記述例を示します。

1 つの UDP 通信端点の定義は、以下で示すように 4 つのフィールドで構成されています。

{ 属性 ,{ 自局の IP アドレス , 自局のポート番号 }, コールバックルーチンアドレス }

現在、これら 4 つのフィールドの内、自局のポート番号、コールバックルーチンアドレスの設定が必要です。属性は未サポートのため、設定値は無効となります。

自局の IP アドレスは、T2 の初期化において、変数 tcpudp_env に設定された自局の IP アドレス

が設定されるため、コンフィグレーションファイルでの設定値は無効となります。
コールバック機能を使用する場合には、コールバックルーチンアドレスにコールバックルーチンへのアドレスを設定して下さい。コールバック機能を使用しない場合には、NULL(=0)を設定して下さい。
設定できる UDP 通信端点は、1 つです。
変数__udpcepn には、UDP 通信端点の総数が自動的に設定されるため、変更の必要はありません。

本定義により、指定したポート番号における UDP を用いたデータの送受信が可能となります。また、UDP を使用しない場合も定義してください。

3.4 IP ヘッダ関連の定義

```
/***  マルチキャスト送信時の TTL  ***/
const UB __multi_TTL = 1;
```

図 6　IP ヘッダ関連の定義（config_tcpudp.c.tpl より抜粋）

変数__multi_TTL には、マルチキャスト宛に送信する IP データグラムの TTL を設定します。TTL は、IP ヘッダの要素であり、IP データグラムが通過できるルータの数をあらわします。
マルチキャスト宛以外の IP データグラムを送信する場合、IP ヘッダの TTL にはデフォルトの 80 が設定されます。
通常、マルチキャスト宛 IP データグラムのプロトコルは UDP のため、マルチキャスト宛の UDP データの送信時にこの設定値を使用します。ただし、T2 では基本的に API で指定されたパラメータの妥当性をチェックしません。そのため、TCP の接続要求 API で宛先 IP アドレスをマルチキャストアドレスに設定した場合、マルチキャスト宛の不正な TCP パケットが送信されます。TCP では宛先 IP アドレスをマルチキャストアドレスに設定しないで下さい。

3.5 TCP オプションの定義

```
/*************************************************************************/
/*************************      TCP 関連の定義      *************************/
/*************************************************************************/
                  :
                  :
/*** TCP の MSS ***/
const UH _tcp_mss = 64;                                  /* 最大：1460 バイト */

/*** シーケンス番号の初期値 (0 以外の値を設定) ***/
UW      _tcp_initial_seqno = 1;

/*** 2MSL 待ち時間 (単位：10ms) ***/
const UH  _tcp_2msl = (1*60*(1000/10));                  /* 1 分 */


/*** 再転送タイムアウトの最大値  (単位：10ms) ***/
const UH  _tcp_rt_tmo_rst = (10*60*(1000/10));           /* 10 分 */
```

図 7　TCP のオプション定義（config_tcpudp.c.tpl より抜粋）

図 7に TCP 関連のオプション設定の記述例を示します。
TCP 関連のオプション設定では、MSS(Maximum Segment Size)のデフォルト値、シーケンス番号の初期値、２MSL 待ち時間、再転送タイムアウトの最大値の設定が必要です。以下に、各々のオプションについて説明します。

＜ TCP の MSS ＞
変数_tcp_mss には、TCP による通信において、ひとつの TCP セグメントで送ることのできる最大データ数（バイト）を指定します。ヘッダのサイズは含みません。１～1460 までの値を設定

することが可能です。それ以外を設定した場合は、RFC によるデフォルトの 536 に設定されます。

この値は TCP 接続時に、TCP ヘッダオプションとして通信相手に送信されます。相手もこのオプションを送信してきた場合、両者のうち小さい方の値を以後この接続における MSS として用います。相手がこのオプションを送信してこなかった場合は、相手がデフォルトの 536 を送信してきたものとして処理します。

ただし、使用する通信端点の受信ウィンドウサイズが_tcp_mss よりも小さい場合、受信ウィンドウサイズの値を MSS として相手に送信します。

また、T2 では、IP フラグメントをサポートしていないため、MTU の小さな通信路を使って通信する可能性のある場合は、_tcp_mss を MTU より小さなサイズに設定する必要があります。

＜ シーケンス番号の初期値 ＞

送信する TCP セグメントのシーケンス番号の初期値を設定します。変数 _tcp_initial_seqno に、正の値（0 は含まない）を設定して下さい。

また、変数 _tcp_initial_seqno は、必ず RAM 領域に配置して下さい。

＜ 2MSL 待ち時間 ＞

TCP の状態遷移において、TIME-WAIT 状態に留まる時間を設定して下さい。

時間の単位は 10ms です。上記の設定では、

（1×60×（1000÷10））×10ms = 60000ms = 60sec = 1 分

となります。

＜ 再転送タイムアウトの最大値 ＞

TCP の再転送を行う時間を設定します。再転送を繰り返し、最初にセグメントを送信してからの経過時間がこの設定値に達すると、通信相手にリセットセグメントを送信してコネクションを破棄します。

時間の単位は 10ms です。上記の設定では、

（10×60×（1000÷10））×10ms = 600000ms = 600sec = 10 分

となります。

再転送までの時間は、指数バックオフのアルゴリズムにより、指数的に長くなります。この再転送までの時間の最大値は 60 秒です。再転送までの時間が 60 秒に達すると、それ以降は 60 秒間隔で再転送が繰り返されます。

3.6 自局の設定（Ethernet）

```
/*****************************************************************************/
/**************************      IP 関連の定義              ********************/
/*****************************************************************************/
#if defined(_ETHER)
const UH _ip_tblcnt = 3;
#define MY_IP_ADDR      192,168,0,3     /* 自局の IP アドレス  */
#define GATEWAY_ADDR    0,0,0,0         /* ゲートウェイアドレス(全て 0 の場合は無効)*/
#define SUBNET_MASK     255,255,255,0   /* サブネットマスク  */
                :
                :
TCPUDP_ENV tcpudp_env = {
        {MY_IP_ADDR},           /* IP アドレス */
        {SUBNET_MASK},          /* サブネットマスク */
        {GATEWAY_ADDR}          /* ゲートウェイアドレス */
};

/*****************************************************************************/
/**************************      ドライバ関連の定義          ********************/
/*****************************************************************************/
#if defined(_ETHER)
/*-------------------------------------------------------------------------*/
/*   Ethernet                                                              */
/*-------------------------------------------------------------------------*/
/* 自局の MAC アドレス(未指定時は全て 0 を設定) */
#define MY_MAC_ADDR     0x40,0x00,0x00,0x00,0x00,0x00

UB _myethaddr[6]={MY_MAC_ADDR}; /* MAC アドレス */
```

図 8　Ethernet 使用時の自局の設定（config_tcpudp.c.tpl より抜粋）

Ethernet を使用する場合の自局に関する情報として、IP アドレス、サブネットマスク、ゲートウェイアドレス、イーサネットアドレス（MAC アドレス）を定義します。図 8に記述例を示します。

自局の設定は、以下で示す 3 つのフィールドを含む TCPUDP_ENV 構造体（第4章参照）とイーサネットアドレス(MAC アドレス)で構成されています。

tcpudp_env = { IP アドレス , サブネットマスク , ゲートウェイアドレス }

_myethaddr = { MAC アドレス }

各情報は、図 8に示すように define 定義を用いることもできます。各情報の値は、コンマで区切って指定してください。

IP アドレス、サブネットマスクの設定は必須です。

ゲートウェイアドレスは、使用する場合にはそのアドレスを、そうでない場合には{0, 0, 0, 0}と、全て 0 を設定して下さい。ゲートウェイアドレスの設定は 1 つのみ可能です。

MAC アドレスには、プログラムにより Ethernet コントローラに対して MAC アドレスを指定する場合にはそのアドレスを、EEPROM 等により Ethernet コントローラの MAC アドレスを自動設定する場合には{0, 0, 0, 0, 0, 0}と、全て 0 を設定して下さい。

また、変数 tcpudp_env、変数_myethaddr は、必ず RAM 領域に配置して下さい。

変数_ip_tblcnt には、T2 で使用する ARP のキャッシュエントリ数を設定します。T2 では通信するホスト 1 台につき、 1 つのキャッシュエントリを使用します。

「同時に通信する可能性のあるホスト数＋１」以上の値を推奨します。「同時に通信する可能性のあるホスト数」よりも小さな値を設定した場合、動作は不定です。例えば、TCP、UDP を同時に使用し、それぞれ別ホストと通信する場合、 3 以上の値を設定するようにして下さい。

また、キャッシュエントリの保持期間は 10 分のため、その間に、頻繁に多数のホストと通信する場合、ホスト数以上の値を設定すると通信効率が上がります。例えば、4 つのホストと順に UDP で通信する場合、4 以下のエントリ数を設定すると毎回 ARP の解決が必要となります。

3.7 自局の設定（PPP）

```
/************************************************************************/
/**************************      IP 関連の定義              *****************/
/************************************************************************/
                  :
                  :
#elif defined(_PPP)
#define MY_IP_ADDR      0,0,0,0         /* 自局の IP アドレス(全て 0 を推奨) */
#define GATEWAY_ADDR    0,0,0,0         /* ゲートウェイアドレス (全て 0 を設定) */
#define SUBNET_MASK     0,0,0,0         /* サブネットマスク（全て 0 を設定） */
#endif

TCPUDP_ENV tcpudp_env = {
        {MY_IP_ADDR},           /* IP アドレス */
        {SUBNET_MASK},          /* サブネットマスク */
        {GATEWAY_ADDR}          /* ゲートウェイアドレス */
};

/************************************************************************/
/**************************      ドライバ関連の定義          *****************/
/************************************************************************/
                  :
                  :
/*----------------------------------------------------------------------*/
/*   PPP                                                                */
/*----------------------------------------------------------------------*/
/* PAP 関連の設定 */
const UH ppp_auth = AUTH_PAP;           /* PAP,認証無しを許可 */
UB user_name[]   = "abcde";             /* ユーザ名    */
UB user_passwd[] = "abc00";             /* パスワード */

/* ダイヤルアップ関連の設定 */
const UB peer_dial[]    = "0, 123";             /* 宛先の電話番号 */
const UB at_commands[] = "ATW2S0=2&C0&D0&S0M0S10=255X3"; /* モデムの初期化コマンド */
#endif
```

図 9　PPP 使用時の自局の設定（config_tcpudp.c.tpl より抜粋）

PPP を使用する場合の自局に関する情報として、IP アドレス、PAP 関連、ダイヤルアップ関連を定義します。図 9に記述例を示します。

IP アドレスの設定には、Ethernet の場合と同様に、以下に示す TCPUDP_ENV 構造体（第4章参照）を用います。サブネットマスクおよびゲートウェイアドレスには全て 0 を設定してください。

tcpudp_env = { IP アドレス , サブネットマスク , ゲートウェイアドレス }

図 9に示すように define 定義を用いることもできます。値は、コンマで区切って指定してください。

また、変数 tcpudp_env は、必ず RAM 領域に配置して下さい。

T2 で使用する PPP は、PPP クライアントであり、IP アドレスを PPP サーバで割り当てるように要求する場合、図 9のように自局の IP アドレスに{0, 0, 0, 0}と、全て 0 を設定します。

また、特定の IP アドレスを PPP サーバに対して要求する場合には、要求する IP アドレスを設定します。ただし、PPP サーバが PPP クライアントの要求した IP アドレスを許可しなかった場合、IP アドレスは PPP サーバが割り当てた IP アドレスとなります。

PPP サーバから IP アドレスが割り当てられる場合、変数 tcpudp_env の IP アドレスは PPP サーバが割り当てた IP アドレスに更新されます。

サブネットマスクは、本来 PPP において意味を持ちませんが、T2 ではサブネットマスクを用いて、自局と通信相手が同一ネットワークかどうかをチェックしています。そのため、サブネットマスクが全て 0 の場合は正常に通信できますが、それ以外の場合、API にエラーが返り、通信できないことがあります。

認証方式として PAP を使用する場合、認証方式、ユーザ名とパスワードの設定が必要です。変数 ppp_auth に AUTH_PAP、変数 user_name にユーザ名、変数 user_passwd にパスワードを設定してください。また、変数 user_name、user_passwd は near 領域に確保してください。認証を用いない場合には、変数 ppp_auth に AUTH_NON を設定して下さい。この場合、変数 user_name,

user_passwd への値の設定は不要です(設定値は無視されます)。
　ダイヤアップ関連の定義として、宛先の電話番号と、モデムに対する初期化コマンドの設定が必要です。宛先の電話番号は変数 peer_dial に、モデムの初期化コマンドは変数 at_commands に設定してください。また、モデムの初期化コマンドについては、モデムのマニュアルを参照下さい。

4. T2 の API 仕様

T2 でサポートする API の一覧を表 3に示します。TCP、UDP 関連の各 API は、ITRON TCP/IP API 仕様に準拠しています。

T2 では、同時に実行可能な API の数は、TCP と UDP で各１つです。

表 3　T2 の API 一覧

T2 の API		C 言語 API
TCP	TCP 接続要求待ち(受動オープン)	tcp_acp_cep()
	TCP 接続要求　　(能動オープン)	tcp_con_cep()
	TCP データ送信の終了	tcp_sht_cep()
	TCP 通信端点のクローズ	tcp_cls_cep()
	TCP データ送信	tcp_snd_dat()
	TCP データ受信	tcp_rcv_dat()
UDP	UDP データ送信	udp_snd_dat()
	UDP データ受信	udp_rcv_dat()
初期化	ワークメモリ使用サイズの取得	tcpudp_get_ramsize()
	T2 ライブラリ・オープン	tcpudp_open()
周期処理	TCP/IP プロトコル処理	_process_tcpip()
終了処理	T2 ライブラリ・クローズ	tcpudp_close()

※　上記 API のスタックサイズは、各リリースノートを参照ください。

以降では、API で使用するデータ構造とマクロ定義、各 API の詳細について説明します。各 API の詳細は、以下のフォーマットに沿っています。

＜　API 書式　＞
API の書式を示します。

＜　引数　＞
API に与える引数の意味、値の制限を示します。

＜　戻り値・エラーコード　＞
API からの戻り値やエラーコードの種類、発生する条件を示します。

＜　機能　＞
各 API の機能・動作、使用上の注意点を示します。

データ構造とマクロ定義

T2 の API で使用するデータ構造およびマクロ定義は、ヘッダファイル itcpip.h で定義されています。itcpip.h では、API のプロトタイプ宣言も行っています。

（1）データ構造

● データタイプ

```
typedef  char           B;
typedef  unsigned char  UB;
typedef  short          H;
typedef  unsigned short UH;
typedef  long           W;
typedef  unsigned long  UW;
typedef  void           (*FP)();
typedef  void far       *VP;
typedef  H              INT;
typedef  H              ID;
typedef  H              TMO;
typedef  H              ER;
typedef  H              ATR;
typedef  INT            FN;
```

● IP アドレスとポート番号

```
typedef  struct  t_ipv4ep {
     UW      ipaddr;                 /* IP アドレス */
     UH      portno;                 /* ポート番号 */
} T_IPV4EP;
```

● TCP 受付口

```
typedef  struct  t_tcp_crep {
     ATR     repatr;                 /* TCP 受付口の属性 */
     T_IPV4EP myaddr;                /* 自局の IP アドレスとポート番号 */
} T_TCP_CREP;
```

● TCP 通信端点

```
typedef  struct  t_tcp_ccep {
     ATR     cepatr;                 /* TCP 通信端点属性 */
     VP      sbuf;                   /* 送信用ウィンドウバッファの先頭アドレス */
     INT     sbufsz;                 /* 送信用ウィンドウバッファのサイズ */
     VP      rbuf;                   /* 受信用ウィンドウバッファの先頭アドレス */
     INT     rbufsz;                 /* 受信用ウィンドウバッファのサイズ */
     ER    (*callback)(ID cepid, FN fncd , VP p_parblk); /* コールバックルーチン */
} T_TCP_CCEP;
```

● UDP 通信端点

```
typedef struct t_udp_ccep {
     ATR     cepatr;                 /* UDP 通信端点属性 */
     T_IPV4EP myaddr;                /* 自局の IP アドレスとポート番号 */
     ER    (*callback)(ID cepid, FN fncd , VP p_parblk); /* コールバックルーチン */
} T_UDP_CCEP;
```

● T2 ライブラリ用環境変数

```
typedef struct        {
    UB          ipaddr[4];              /* 自局の IP アドレス */
    UB          maskaddr[4];            /* サブネットマスク */
    UB          gwaddr[4];              /* ゲートウェイアドレス */
} TCPUDP_ENV;
```

（2）マクロ定義

● API で使用するマクロ

表 4　API で使用する機能コードと事象コード

名前	値	意味
TFN_UDP_SND_DAT	-0x223	UDP データの送信
TFN_UDP_RCV_DAT	-0x224	UDP データの受信
TEV_UDP_RCV_DAT	0x221	UDP パケット受信
TMO_POL	0	ポーリング
TMO_FEVR	-1	永久待ち
TMO_NBLK	-2	ノンブロッキングコール

表 5　特殊な IP アドレスとポート番号

名前	値	意味
IPV4_ADDRANY	0	IP アドレスの指定省略
TCP_PORTANY	0	TCP ポート番号の指定省略
NADR	-1	無効アドレス

● エラーコード

表 6　API で使用するエラーコード

名前	値	意味
E_OK	0	正常終了
E_OBJ	-63	オブジェクト状態エラー
E_QOVR	-73	キューイングオーバーフロー
E_WBLK	-83	ノンブロッキングコール受付け
E_TMOUT	-85	タイムアウト
E_CLS	-87	接続の失敗
E_BOVR	-89	バッファオーバーフロー

TCP 接続要求待ち（受動オープン）　　tcp_acp_cep()

＜ API 書式 ＞

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

＜ 引数 ＞

ID	cepid	TCP 通信端点 ID 指定(1 のみ)
ID	repid	TCP 受付口 ID 指定(1 のみ)
T_IPV4EP	*p_dstaddr	相手の IP アドレスとポート番号取得 接続を要求してきた相手の IP アドレスとポート番号を取得
TMO	tmout	タイムアウト指定 正の値: 接続が完了するのを待つ時間 時間の単位は、10msec TMO_FEVR： 接続が完了するまで待ち状態(永久待ち)

＜ 戻り値・エラーコード ＞

E_OK	正常終了（接続が確立）
E_OBJ	オブジェクト状態エラー（使用中の通信端点 ID を指定）
E_TMOUT	タイムアウト（tmout に設定した時間を経過）

＜ 機能 ＞

TCP 受付口 **repid** に対する接続要求を待ちます。接続要求があった場合には、指定した TCP 通信端点 **cepid** を用いて接続を確立し、接続を要求してきた相手の IP アドレスとポート番号を引数 **p_dstaddr** が指す領域に格納して返します。

接続が確立するまでは待ち状態となりますが、この待ち時間に対してタイムアウト指定することができます。指定時間内に接続が確立されない場合、エラーコード **E_TMOUT** を返します。

接続が確立された場合、戻り値 **E_OK** が返されます。確立されない場合は、各原因により上記 **E_OK** 以外のエラーコードを返します。エラーコードに対応した原因を（）内に示します。

T2 では受付口、通信端点ともにひとつしか持たないので、ID 番号には 1 を指定します。1 以外を指定した場合はエラーを返さずに 1 とみなして処理を続けます。

TCP 接続要求（能動オープン）　tcp_con_cep()

＜ API 書式 ＞

```
ER  tcp_con_cep( ID cepid, T_IPV4EP *p_myaddr, T_IPV4EP *p_dstaddr, TMO tmout )
```

＜ 引数 ＞

ID	cepid	TCP 通信端点 ID 指定(1 のみ)
T_IPV4EP	*p_myaddr	自局の IP アドレスとポート番号指定
T_IPV4EP	*p_dstaddr	接続したい相手側の IP アドレスとポート番号指定
TMO	tmout	タイムアウト指定 正の値: 接続が完了するのを待つ時間 時間の単位は、10msec TMO_FEVR： 接続が完了するまで待ち状態(永久待ち)

＜ 戻り値・エラーコード ＞

E_OK	正常終了（接続が確立）
E_OBJ	オブジェクト状態エラー（使用中の通信端点 ID、または自局ポート番号を指定）
E_TMOUT	タイムアウト（tmout に設定した時間を経過）
E_CLS	接続の失敗（接続が拒否された）

＜ 機能 ＞

TCP 通信端点 cepid を用いて、接続したい相手の IP アドレス/ポート番号に対して接続を要求し、接続が完了するまで待ち状態となります。接続が確立するまでは待ち状態となりますが、この待ち時間に対してタイムアウト指定することができます。指定時間内に接続が確立されない場合、E_TMOUT を返します。

タイムアウトにより接続の要求がキャンセルされた場合、および相手がサポートしていないポート番号を指定するなど接続が拒否された場合には、通信端点 cepid は未使用状態に戻ります。

この API がコールされると先ず通信端点が使用中でないかをチェックし､以下の設定を行います。

自局の IP アドレスには、変数 tcpudp_env (3章参照)に設定された自局の IP アドレスが設定されます。

自局のポート番号に 0 以外の値を指定した場合、その値が設定されます。また、自局のポート番号に TCP_PORTANY（0）を指定した場合、T2 で 1024～5000 番の範囲の中から割り当てます。

自局の IP アドレス/ポート番号 p_myaddr に NADR(−1)を指定した場合には、自局の IP アドレスには変数 tcpudp_env に設定された IP アドレスを、自分側のポート番号には T2 で 1024～5000 番の範囲の中からポート番号を割り当てます。

接続が確立された場合、戻り値 E_OK が返されます。確立されない場合は、各原因により上記 E_OK 以外のエラーコードを返します。エラーコードに対応した原因を（）内に示します。

T2 では通信端点をひとつしか持たないので、ID 番号には 1 を指定します。1 以外を指定した場合はエラーを返さずに 1 とみなして処理を続けます。

TCP データ送信終了　　tcp_sht_cep()

＜ API 書式 ＞

ER tcp_sht_cep(ID cepid)

＜ 引数 ＞

ID	**cepid**	**TCP** 通信端点 **ID** 指定**(1** のみ**)**

＜ 戻り値・エラーコード ＞

E_OK	正常終了（データ送信が終了）
E_OBJ	オブジェクト状態エラー（指定した通信端点が未接続）

＜ 機能 ＞

TCP 通信端点 **cepid** に対する接続の切断処理の準備として、データ送信の終了手続きを行います。具体的には、送信したデータに対する **ACK** を受信した時点で、**FIN** を送信します。本 **API** は切断処理の手配を行うだけのため、待ち状態にはなりません。

本 **API** 発行後、指定した **TCP** 通信端点 **cepid** に対してデータを送信することはできず、送信しようとした場合、エラーコード **E_OBJ** を返します。データの受信は可能です。

データ送信の終了手続きを完了した場合、戻り値 **E_OK** が返されます。手続きに失敗した場合は、各原因により上記 **E_OK** 以外のエラーコードを返します。エラーコードに対応した原因を（）内に示します。

T2 では通信端点をひとつしか持たないので、ID 番号には 1 を指定します。1 以外を指定した場合はエラーを返さずに 1 とみなして処理を続けます。

TCP 通信端点のクローズ　　tcp_cls_cep()

＜ API 書式 ＞

```
ER  tcp_cls_cep( ID cepid, TMO tmout )
```

＜ 引数 ＞

ID	cepid	TCP 通信端点 ID 指定(1 のみ)
TMO	tmout	タイムアウト指定 正の値: 接続がクローズするのを待つ時間 時間の単位は、10msec TMO_FEVR： 接続がクローズするまで待ち状態(永久待ち)

＜ 戻り値・エラーコード ＞

E_OK	正常終了（接続が正常切断）
E_OBJ	オブジェクト状態エラー（指定した通信端点が未接続）
E_TMOUT	タイムアウト（tmout に設定した時間を経過。接続を強制切断）

＜ 機能 ＞

TCP 通信端点 cepid の接続を切断します。本 API を発行した後は、相手から送信されたデータを破棄します。

タイムアウトにより接続の切断処理がキャンセルされた場合、本 API で指定した TCP 通信端点から RST を送信し、強制的に接続を切断します。この場合、正常切断ではないため、エラーコード E_TMOUT を返します。

本 API は、正常切断、強制切断のどちらの場合も通信端点が未使用状態になるのを待って API からリターンするため、本 API からリターンした後は TCP 通信端点 cepid をすぐに利用することが可能です。TCP 通信端点は、接続が完全に終了するまで未使用状態となりません。TCP/IP の規格に従うと接続が完全に終了するまでに TIME_WAIT 状態に留まる場合があります。TIME_WAIT 状態の時間 2MSL は、T2 のコンフィグレーションファイルで設定できます（3章参照）。

接続が正常に切断された場合、戻り値 E_OK が返されます。タイムアウトにより強制的に切断された場合は、エラーコード E_TMOUT を返します。これらのコードが返された場合、接続は完全に終了しているため、指定した TCP 通信端点は未使用状態となります。また通信端点が未接続の場合には本 API は、E_OBJ を返します。

T2 では通信端点をひとつしか持たないので、ID 番号には 1 を指定します。1 以外を指定した場合はエラーを返さずに 1 とみなして処理を続けます。

【補足事項】

ITRON 仕様では、データの送信が終了していない場合は、送信が完了するのを待ってから FIN を送信し、接続の切断を行います。T2 では、同時に複数の API を実行できない為、本 API を発行するときには、すでにデータの送信は完了しているため、FIN の送信が待たされることはありません。

TCP データ送信　　tcp_snd_dat()

＜ API 書式 ＞

```
ER  tcp_snd_dat( ID cepid, VP data, INT len, TMO tmout )
```

＜ 引数 ＞

ID	cepid	TCP 通信端点 ID 指定(1 のみ)
VP	data	送信データの先頭アドレス指定 ユーザが送信するデータの先頭アドレス
INT	len	送信データの長さ指定 正の値
TMO	tmout	タイムアウト指定 正の値: 送信が完了するのを待つ時間 時間の単位は 10msec TMO_FEVR： 送信が完了するまで待ち状態(永久待ち)

＜ 戻り値・エラーコード ＞

正の値	正常終了（送信したデータのサイズ=第 3 引数 len の値。単位：バイト）
E_OBJ	オブジェクト状態エラー（未接続、送信終了）
E_TMOUT	タイムアウト（tmout に設定した時間を経過）
E_CLS	相手から接続を切断された（RST の送受信により異常切断した場合）

＜ 機能 ＞

TCP 通信端点 cepid からデータを送信します。正常に送信した場合、API は送信したデータサイズを返します。

T2 では、RAM 使用の効率化、送信速度の向上のため、ITRON TCP の仕様と異なる点があります。本ライブラリでは、送信データが格納されている領域（以下、ユーザ送信バッファと呼びます）が、ITRON TCP で定義されている送信ウィンドウを兼ねています。同様に、送信ウィンドウサイズは、送信データの長さとなり、本 API の使用状況によりサイズが異なります。

ITRON 仕様では、ユーザ送信バッファのデータを送信ウィンドウにコピーした時点で、API からリターンしますが、本 API ではデータを送信し、送信データに対する ACK を受信した後、API からリターンします。具体的には、MSS や相手の受信ウィンドウサイズにより TCP レベルの分割が発生した場合、１つ目の分割データの送信に対する ACK ではなく、全てのデータの送信に対する ACK を受信した時点で API からリターンします。

T2 ではデータが送信された場合、送信したデータのサイズ(第 3 引数 len の値)が返されます。送信に失敗した場合は、各原因により上記エラーコードを返します。エラーコードに対応した原因を（）内に示します。

T2 では通信端点をひとつしか持たないので、ID 番号には 1 を指定します。1 以外を指定した場合はエラーを返さずに 1 とみなして処理を続けます。

【補足事項】

ITRON 仕様では、送信ウィンドウの空き領域のサイズにより、送信データのサイズが決まります。このため、API の正常終了時の戻り値は必ず第 3 引数 len と一致するわけではありません。T2 以外の ITRON TCP/IP API 仕様準拠のプロトコルスタック上に移植される場合は、ご注意ください。

TCP データ受信　　tcp_rcv_dat()

＜ API 書式 ＞

```
ER  tcp_rcv_dat( ID cepid, VP data, INT len, TMO tmout )
```

＜ 引数 ＞

ID	cepid	TCP 通信端点 ID 指定（1 のみ）
VP	data	受信データを格納する領域の先頭アドレス指定 ユーザが確保した受信データを格納するバッファの先頭アドレス
INT	len	受信データを格納する最大サイズ 正の値
TMO	tmout	タイムアウト指定 正の値: 受信が完了するのを待つ時間 時間の単位は 10msec TMO_FEVR： 受信が完了するまで待ち状態(永久待ち)

＜ 戻り値・エラーコード ＞

正の値	正常終了（受信したデータのサイズ。単位：バイト）
0	データ終了（接続が正常切断。切断までのデータは全て受信）
E_OBJ	オブジェクト状態エラー（未接続）
E_TMOUT	タイムアウト（tmout に設定した時間を経過）
E_CLS	接続を切断され、受信ウィンドウが空（RST の受信により異常切断した場合）

＜ 機能 ＞

TCP 通信端点 cepid からデータを受信します。

通信相手から送信されたデータは、受信ウィンドウに格納されます。本 API は、受信ウィンドウから引数 data が指すユーザ領域にデータをコピー（以下、データの取り出しと呼びます）し、リターンします。受信ウィンドウが空の場合は、データを受信するまで本 API は待ち状態となります。

受信ウィンドウに入っているデータのサイズが、受信しようとしたデータのサイズ len よりも短い場合、受信ウィンドウが空になるまでデータを取り出し、取り出したデータのサイズを戻り値として返します。

相手から接続が正常に切断され、受信ウィンドウのデータを全て取り出し、データがなくなると、API から 0 が返ります。

データを受信した場合、受信したデータのサイズが返されます。受信に失敗した場合は、各原因により上記エラーコードを返します。エラーコードに対応した原因を（）内に示します。

T2 では通信端点をひとつしか持たないので、ID 番号には 1 を指定します。1 以外を指定した場合はエラーを返さずに 1 とみなして処理を続けます。

UDP データ送信　　udp_snd_dat()

＜ API 書式 ＞

```
ER  udp_snd_dat( ID cepid, T_IPV4EP *p_dstaddr, VP data, INT len, TMO tmout )
```

＜ 引数 ＞

ID	cepid	UDP 通信端点 ID 指定（1 のみ）
T_IPV4EP	*p_dstaddr	送信したい相手側の IP アドレスとポート番号
VP	data	送信データの先頭アドレス指定 ユーザが送信するデータの先頭アドレス
INT	len	送信データの長さ指定 0 以上 1472 以下
TMO	tmout	タイムアウト指定 正の値 ： 送信が完了するのを待つ時間 時間の単位は 10msec TMO_FEVR：送信が完了するまで待ち状態(永久待ち) TMO_NBLK: ノンブロッキング呼び出し

＜ 戻り値・エラーコード ＞

0, 正の値	正常終了（送信したデータのサイズ=第 4 引数 len の値。単位：バイト）
E_PAR	パラメータエラー（tmout が不正な値）
E_QOVR	キューイングオーバーフロー（キューイングできる送受信の総数を超えた）
E_TMOUT	タイムアウト（tmout に設定した時間を経過）
E_WBLK	ノンブロッキングコール受付

＜ 機能 ＞

UDP 通信端点 cepid から、相手側の IP アドレスとポート番号を指定して、UDP データグラムを送信します。UDP データグラムを送信バッファに入れた時点で本 API からリターンします。ここで指す送信バッファとは、T2 内部で確保したものではなく、Ethernet コントローラ内部の送信バッファ（Ethernet の場合）、シリアルドライバで確保した送信バッファ（PPP の場合）のことです。データが送信バッファに格納された場合、送信したデータのサイズ(第 4 引数 len の値)が返されます。送信に失敗した場合は、各原因により上記エラーコードを返します。エラーコードに対応した原因を（）内に示します。

本 API では、宛先アドレスにユニキャストアドレス、または、マルチキャストアドレスを指定してデータを送信することができます。宛先アドレスにブロードキャストアドレスを指定したものは送信できません。ブロードキャストアドレスを指定した場合、動作、戻り値は不定です。

T2 では、1 度に送信できるデータサイズの最大値は、送信バッファのサイズに依存します。ドライバ・インタフェース関数 lan_write, ppp_write により、最大サイズが M (バイト)の Ethernet フレーム、PPP フレームを送信(送信バッファに格納)できる場合、1 度に送信可能なデータの最大サイズ N(バイト)は、

N ＝ M － フレーム・ヘッダサイズ(F) － IP ヘッダの最小サイズ(I) － UDP ヘッダサイズ(U)

となります。ここで、

M <= 1514 (Ethernet の場合)、M <= 1504 (PPP の場合)
F = 14 (Ethernet の場合), F = 4 (PPP の場合)
I = 20
U = 8

です。T2 では、IP フラグメントに対応していないため、これより大きなサイズのデータを送信する場合には、データを分割し、N バイト以下にする必要があります。N バイト以上のデータサイズを指定した場合の動作、戻り値は不定です。

タイムアウト指定 tmout に TMO_FEVR を指定した場合は、送信バッファにデータが入るまで待ち状態になります。タイムアウト指定 tmout に正の値を指定した場合は、指定時間まで待ち状態になります。指定時間までに送信バッファにデータが入らなかった場合には、エラーコード E_TMOUT を返します。データが送信バッファに格納された場合、格納されたデータのサイズを返します。

タイムアウト指定 **tmout** に **TMO_NBLK** を指定した場合、ノンブロッキングコールとなり、本 **API** では待ち状態になりません。送信要求が受け付けられるとノンブロッキングコール受付 **E_WBLK** を返します。この場合、送信バッファにデータが格納された時点でコールバック関数が呼び出されます。コールバック関数には、引数として **UDP** 通信端点 **ID**、機能コード **TFN_UDP_SND_DAT**、エラーコードへのポインタが渡されます。エラーコードには、格納されたデータサイズが示されます。

本 **API** をブロッキングコールしてから **API** を抜けるまでの間、ノンブロッキングコールしてからコールバック関数**(**機能コード **TFN_UDP_SND_DAT)**が呼び出されるまでの間は、**UDP** 送信処理中とみなされます。そのため、処理中は送信データを書き換えないで下さい。

T2 では、事象コード **TEV_UDP_RCV_DAT** により呼び出されたコールバック関数の中でポーリング指定**(TMO_POL)**により **udp_rcv_dat()**を呼び出す場合を除き、同時に複数の **UDP** 関数（**udp_rcv_dat()**、および、**udp_snd_dat()**）を発行できません。**UDP** 関数の処理中に本 **API** を発行した場合、エラーコード **E_QOVR** を返します。

T2 では通信端点を 1 つしか持たないので、ID 番号には 1 を指定します。1 以外を指定した場合はエラーを返さずに 1 とみなして処理を続けます。

UDP データ受信　　udp_rcv_dat()

＜ API 書式 ＞

ER udp_rcv_dat(ID cepid, T_IPV4EP *p_dstaddr, VP data, INT len, TMO tmout)

＜ 引数 ＞

ID	**cepid**	**UDP** 通信端点 **ID** 指定（**1** のみ）
T_IPV4EP	***p_dstaddr**	相手の **IP** アドレスとポート番号取得 データを送信してきた相手の **IP** アドレスとポート番号を取得
VP	**data**	受信データを格納する領域の先頭アドレス指定 ユーザが確保した受信データを格納するバッファの先頭アドレス
INT	**len**	受信データを格納する最大サイズ **0** 以上 **1472** 以下
TMO	**tmout**	タイムアウト指定 正の値 **:** 受信が完了するのを待つ時間 時間の単位は **10msec** **TMO_FEVR**： 受信が完了するまで待ち状態**(**永久待ち**)** **TMO_NBLK:** ノンブロッキング呼び出し **TMO_POL :** ポーリング 事象コード **TEV_UDP_RCV_DAT** によるコールバック関数の中でのみ指定可能。

＜ 戻り値・エラーコード ＞

0,正の値	正常終了（受信したデータのサイズ。単位：バイト）
E_PAR	パラメータエラー（**tmout** が不正な値）
E_QOVR	キューイングオーバーフロー（キューイングできる送受信の総数を超えた）
E_TMOUT	タイムアウト（**tmout** に設定した時間を経過）
E_WBLK	ノンブロッキングコール受付
E_BOVR	バッファオーバーフロー（受信データを格納する領域以上のデータが届いた）

＜ 機能 ＞

UDP 通信端点 **cepid** から **UDP** データグラムを受信し、相手側の **IP** アドレスとポート番号を取得します。データを受信した場合、受信したデータのサイズが返されます。受信に失敗した場合は、各原因により上記エラーコードを返します。エラーコードに対応した原因を（）内に示します。

T2 では、**1** 度に受信できるデータサイズの最大値は、ドライバの受信バッファのサイズに依存します。ドライバ・インタフェース関数 **lan_read, ppp_read** により、最大サイズが **M (**バイト**)**の **Ethernet** フレーム、**PPP** フレームを受信**(**受信バッファに格納**)**できる場合、**1** 度に受信可能なデータの最大サイズ **N(**バイト**)**は、

N = M － フレーム・ヘッダサイズ(F) － IP ヘッダの最小サイズ**(I) － UDP** ヘッダサイズ**(U)**

となります。ここで、

M <= 1514 (Ethernet の場合**)**、**M <= 1504 (PPP** の場合**)**
F = 14 (Ethernet の場合**), F = 4 (PPP** の場合**)**
I = 20
U = 8

です。**T2** では、**IP** フラグメントに対応していないため、これより大きなサイズのデータを受信した場合には破棄されます。

タイムアウト指定 **tmout** に **TMO_FEVR** を指定した場合は、**UDP** データグラムを受信するまで待ち状態になります。タイムアウト指定 **tmout** に正の値を指定した場合は、指定時間まで待ち状態になります。指定時間までに **UDP** データグラムを受信できなかった場合、エラーコード **E_TMOUT** を返します。

タイムアウト指定 **tmout** に **TMO_NBLK** を指定した場合は、ノンブロッキングコールとなり、本 **API** では待ち状態になりません。要求が受け付けられるとノンブロッキングコール受付 **E_WBLK** が返ります。この場合、受信データをユーザの受信バッファに格納した時点でコールバック関数が呼び出さ

れます。コールバック関数には、引数として UDP 通信端点 ID、機能コード TFN_UDP_RCV_DAT、エラーコードへのポインタが渡されます。受信したデータサイズはエラーコードに示され、受信したデータはノンブロッキングコールを行った API で指定したユーザ領域 data に格納されています。

本 API をブロッキングコールしてから API を抜けるまでの間、ノンブロッキングコールしてからコールバック関数(機能コード TFN_UDP_RCV_DAT)が呼び出されるまでの間は、UDP データ受信待ち状態であり、UDP 受信処理中とみなされます。

UDP データ受信待ち状態で UDP データグラムを受信した場合、受信した UDP データグラムのデータ部の先頭から、受信データを格納する領域 data へコピーします。受信したデータのサイズが指定したサイズ len 以下の場合、受信したデータのサイズだけコピーします。受信したデータのサイズが指定したサイズ len よりも大きい場合は、指定したサイズ len までコピーし、残りを破棄します。エラーコードは、前者の場合には受信したデータのサイズ、後者の場合にはバッファオーバーフローE_BOVR となります。

UDP データ受信待ちではない状態で UDP データグラムを受信した場合、事象コード TEV_UDP_RCV_DAT のコールバック関数が呼び出されます。コールバック関数には、引数として UDP 通信端点 ID、事象コード TEV_UDP_RCV_DAT、エラーコード(受信したデータのサイズ)へのポインタが渡されます。コールバック関数の中で、本 API のタイムアウト指定 tmout にポーリング TMO_POL を指定して呼び出した場合のみ、データを読み出すことができます。ポーリング指定で読み出さない場合は、コールバック関数を抜けた後、ドライバの受信バッファを解放します。

T2 では、事象コード TEV_UDP_RCV_DAT により呼び出されたコールバック関数の中でポーリング指定(TMO_POL)により udp_rcv_dat()を呼び出す場合を除き、同時に複数の UDP 関数（udp_rcv_dat()、および、udp_snd_dat()）を発行できません。UDP 関数の処理中に本 API を発行した場合、エラーコード E_QOVR を返します。

本 API では、宛先 IP アドレスがユニキャストアドレス、または、マルチキャストアドレス（224.0.0.0 ～ 239.255.255.255）の UDP データグラムを受信することができます。ただし、T2 では IGMP をサポートしていませんので、ルータに対してマルチキャストへの参加を通知することはできません。宛先 IP アドレスにブロードキャストアドレスが設定されたものは受信できません。

T2 では通信端点を 1 つしか持たないので、ID 番号には 1 を指定します。1 以外を指定した場合はエラーを返さずに 1 とみなして処理を続けます。

【補足事項】

ノンブロッキングコールを使用する場合、コールのタイミングによっては、事象コード TEV_UDP_RCV_DAT のコールバック関数が先に呼ばれることがあります。その場合、次の UDP データの受信により機能コード TFN_UDP_RCV_DAT のコールバック関数が呼ばれます。

UDP コールバック関数　　callback()

＜ API 書式 ＞

```
ER  callback( ID cepid, FN fncd, VP p_parblk )
```

＜ 引数 ＞

ID	cepid	UDP 通信端点 ID 指定（1 のみ）
FN	fncd	事象の種類
VP	p_parblk	事象に固有なパラメータブロックのアドレス

＜ 戻り値・エラーコード ＞

ER 型の戻り値を返すが、ライブラリ側では参照しない

＜ 機能 ＞

UDP コールバック関数は、UDP 関数をノンブロッキングコールした場合の完了通知、UDP データ受信待ちではない状態での UDP データ受信通知を行います。

UDP コールバック関数の中身は各通知における処理であり、ユーザが作成します。引数 fncd が、通知の種類をあらわします。事象の種類とコールバック関数が呼ばれるタイミングを表 7に示します。

表 7　コールバック関数の呼ばれるタイミングと引数

タイミング	引数 fncd	引数 p_parblk	備考
udp_snd_dat(,,s_buf, ,TMO_NBLK) コール後、 UDP データ送信完了時	TFN_UDP_SND_DAT	送信データサイズ (ブロッキングコールの戻り値と同等)	ユーザ送信データ s_buf を送信後、コールされる。
udp_rcv_dat(,d_addr, r_buf,,TMO_NBLK) コール後、 UDP データ受信完了時	TFN_UDP_RCV_DAT	受信データサイズ, E_BOVR (ブロッキングコールの戻り値と同等)	ユーザのデータ受信領域 r_buf に受信データをコピーした時点で、コールされる。相手の IP アドレス、ポート番号は d_addr が示す領域に格納される。
UDP データ受信待ちでない状態で、UDP データを受信した場合	TEV_UDP_RCV_DAT	受信データサイズ	コールバック関数中で udp_rcv_dat(,,buf,,TMO_POL)をコールすると、buf に受信データがコピーされる。戻り値は、受信データサイズ、または、E_BOVR となる。

コールバック関数の各引数は、読出しのみ可能です。

コールバック関数は ER 型の戻り値を返す仕様となっていますが、T2 ライブラリ側では戻り値を参照しません。

UDP コールバック機能を使用する場合、T2 コンフィグレーションファイルの UDP 通信端点の定義において、作成したコールバック関数へのアドレスをコールバックルーチンアドレスに設定します（第3章参照）。

ワークメモリ使用サイズの取得　tcpudp_get_ramsize()

＜　API 書式　＞

```
W  tcpudp_get_ramsize( void )
```

＜　引数　＞

なし

＜　戻り値・エラーコード　＞

T2 が使用するワーク領域のサイズ（単位：バイト）

＜　機能　＞

T2 が使用するワーク領域のサイズ**(RAM** サイズ**)**をリターン値として返します。ワーク領域は、**TCP** の受信ウィンドウなどに使用するメモリ領域のことで、プログラム中で確保し、**API** の初期化関数 **tcpudp_open()**の引数で初期化する必要があります。ワーク領域の先頭アドレスは、**4** バイト境界に配置して下さい。

例： **tcpudp_get_ramsize()**の戻り値が **100** の場合、**T2** が使用するワーク領域 **work** は以下のように定義することができます。

```
UW      work[100/4];
```

本 **API** は、以下の目的で使用できます。

1) 静的な配列でワーク領域を確保する場合

本 **API** では、**T2** のコンフィグレーションファイルで設定した内容により、**T2** で使用するワーク領域のサイズを算出しますが、予めユーザ自身でこのサイズを算出するのは困難です。従って **T2** のコンフィグレーションファイルの内容が決定した時点で、プログラムをビルドしてデバッガで本 **API** を実行して戻り値を調べます。そして戻り値が示すサイズ分、ワーク領域のメモリ配列を確保するようにします。

なお **T2** のコンフィグレーションファイルを変更した場合には、必要なワーク領域のサイズが変わりますので、本 **API** を用いてワーク領域のサイズを再計算してください。またエラー判定処理として、初期設定の処理の中で必ず本 **API** を呼び出して、戻り値と（ユーザが最終的に決めた）ワーク領域のサイズを比較し、異なっていた場合はエラー処理に分岐させて、デバッグやテストの段階で間違いが発見できるようにしておいてください。

2) 動的メモリからワーク領域を確保する場合

アプリケーションプログラムの初期設定で本 **API** を呼び出してワーク領域のサイズを算出します。算出したサイズのワーク領域を動的メモリから確保し、初期化関数 **tcpudp_open()**に渡して初期化してください。

T2 ライブラリ・オープン　　tcpudp_open()

＜ API 書式 ＞

```
ER  tcpudp_open( UW *work )
```

＜ 引数 ＞

UW	*work	T2 が使用するワーク領域のアドレス

＜ 戻り値・エラーコード ＞

E_OK	正常終了
負の値	初期化失敗

＜ 機能 ＞

T2 ライブラリの初期化を行います。ライブラリの初期化処理では、内部管理領域のメモリ割り当てとその初期化およびライブラリが使用する TCP/IP 周期処理関数の起動を行います。

T2 が使用するワーク領域の先頭アドレスを、work に指定します。必要なワーク領域のサイズは、tcpudp_get_ramsize() のリターン値で得られます。

本 API は、Ethernet と組み合わせて使用する場合には lan_open()より前に、また PPP と組み合わせて使用する場合にも ppp_open()より前に呼び出して下さい。

TCP/IP 処理　_process_tcpip()

＜ API 書式 ＞

```
void  _process_tcpip( void )
```

＜ 引数 ＞

なし

＜ 戻り値・エラーコード ＞

なし

＜ 機能 ＞

T2 ライブラリの **TCP/IP** 部の処理を行います。
本 **API** を **10msec** 以下の間隔で起動して下さい（タイマ割り込みなどを使用）。
T2 の **TCP/IP** 部は、**10msec** 単位で時間を管理しているため、起動の間隔が **10msec** を越える場合、

- API で指定したタイムアウトが所定の時間に発生しない。
- 再転送の間隔が所定の間隔にならない。
- ゼロウィンドウプローブの間隔が所定の間隔にならない。
- コネクション切断後の 2MSL 待ち時間が所定の時間にならない。

等の問題が発生します。

T2 ライブラリ・クローズ　　tcpudp_close()

＜　API 書式　＞

ER　tcpudp_close(void)

＜　引数　＞

なし

＜　戻り値・エラーコード　＞

E_OK	正常終了
負の値	終了処理失敗

＜　機能　＞

T2 ライブラリの終了処理を行います。ライブラリの終了処理では、ライブラリが使用する **TCP/IP** 周期処理関数を停止します。

本 **API** を呼び出す前に、**TCP** 通信端点を切断・未使用状態にして下さい。

本 **API** を呼び出す前に、**Ethernet** と組み合わせて使用する場合には **lan_close()**を、**PPP** の場合には **ppp_close()**を呼び出して下さい。

本 **API** の実行後、ライブラリ・オープン関数 **tcpudp_open()**で指定したワーク領域は開放されますので、アプリケーションプログラムでワーク領域が使用可能となります。

T2 の制限・注意事項

T2 の制限・注意事項を以下の(1)～(17)に示します。

1. キャンセル処理関数 tcp_can_cep(), udp_can_cep()はサポートしていません。
2. 省コピーAPI を用いたデータの送受信関数 tcp_get_buf(), tcp_snd_buf(), tcp_rcv_buf(), tcp_rel_buf()はサポートしていません。
3. 緊急データの送受信関数 tcp_snd_oob(), tcp_rcv_oob()はサポートしていません。
4. オプションの設定/取得の関数 tcp_set_opt(), tcp_get_opt(), udp_set_opt(), udp_get_opt() はサポートしていません。
5. TCP の API では、宛先 IP アドレスにマルチキャストアドレス、ブロードキャストアドレス、ループバックアドレスを設定できません。
6. UDP の API では、宛先 IP アドレスにブロードキャストアドレス、ループバックアドレスを設定できません。
7. TCP では、コールバック機能はサポートしていません。TCP 通信端点の定義において、コールバックルーチンアドレスの指定は無効です。
8. TCP の API で可能なタイムアウト指定は永久待ち TMOP_FEVR と正の値のみです。タイムアウト指定に、ノンブロッキングコール(TMO_NBLK)、ポーリング(TMO_POL)を指定した場合、動作は不明です。
9. UDP 受信関数 udp_rcv_dat()のポーリング(TMO_POL)指定は、事象コード TEV_UDP_RCV_DAT により呼び出されたコールバック関数の中でのみ可能です。コールバック関数以外でポーリング(TMO_POL)指定した場合には、パラメータエラー(E_PAR)が返ります。
10. IGMP をサポートしていませんので、ルータに対して宛先 IP アドレスがマルチキャストアドレスの IP データグラムの転送を要求することはできません。
11. T2 で同時に実行可能な API の数は、TCP と UDP で各々1 つです。同時に複数の API を実行した場合、動作は不明です。
12. TCP では、ヘッダオプションは MSS のみサポートしています。TCP セグメントを受信した場合、MSS 以外のオプションについては無視します。
13. IP では、IP オプション、フラグメントはサポートしていません。受信した IP データグラムに IP オプションが含まれている、または、フラグメント化されている場合には、そのデータグラムを破棄します。
14. ICMP では、エコー要求の受信とそれに対するエコー応答の送信のみサポートしています。その他の ICMP メッセージを受信した場合、そのパケットを破棄します。
15. PPP では、圧縮関連のオプション(プロトコルフィールド圧縮、アドレスと制御フィールド圧縮、TCP/IP ヘッダ圧縮) はサポートしていません。圧縮関連のオプションの設定要求を受信した場合には、設定拒否の応答を送信します。
16. 割り込みハンドラからの API の呼び出しは、推奨しません。
17. 関数 tcpudp_close()をコールする場合、コール前に通信端点を切断・未使用の状態にして下さい。通信状態で tcpudp_close()をコールした場合、動作は不明です。

5. Ethernet/PPP ドライバ関連の API 仕様

T2 ドライバ関連の API の一覧を表 7、表８および表９に示します。これらの API は、ITRON TCP/IP API 仕様とは無関係の弊社オリジナルの API です。

表 8　Ethernet ドライバの API 一覧

Ethernet ドライバの API	C 言語 API	備考
Ethernet ドライバ・オープン	ER lan_open(void)	ユーザアプリケーションからコールする API
Ethernet ドライバ・クローズ	ER lan_close(void)	
データ受信	H lan_read(B **)	ライブラリ内部からコールする API
データ送信	H lan_write(B *, H, B *, H)	
Ethernet コントローラのリセット	void lan_reset(void)	

※　表７の API のプロトタイプ宣言は、itcpip.h ファイルに含まれています。

表 9　PPP ドライバの API 一覧

PPP ドライバの API	C 言語 API	備考
PPP ドライバ・開始処理	ER ppp_open(void)	ユーザアプリケーションからコールする API
PPP ドライバ・終了処理	ER ppp_close(void)	
シリアル I/O 開始	void sio_open(UB)	
シリアル I/O 終了	void sio_close (void)	
モデムの接続	ER modem_open(void)	
モデムの切断	ER modem_close(void)	
PPP 状態参照	UH ppp_status(void)	
PPP フレームの受信	H ppp_read(UB **)	ライブラリ内部からコールする API
PPP フレームの送信	H ppp_write(B *, H, B **, H *, H)	
モデムの応答コードの受信	H modem_read(UB **)	
モデムのコマンドの送信	H modem_write(void far *)	
PPP ドライバのステータス取得	UH ppp_drv_status(void)	
PPP 処理関数への要求発行	ER ppp_api_req(UH, void far *, H)	ドライバ API の作成を補助する API

※　表８の API のプロトタイプ宣言は、itcpip.h ファイルに含まれています。

表 10　Ethernet/PPP ドライバ共通の API 一覧

Ethernet/PPP ドライバの API	C 言語 API	備考
受信バッファ解放許可	H rcv_buff_release(void)	ライブラリ内部からコールする API
API 完了待ち (完了チェック待ち)	void api_slp(void)	
API 完了待ち解除	void api_wup(void)	
TCP 周期処理関数の起動制御	void tcpudp_act_cyc(UB)	
時刻の取得	UH tcpudp_get_time(void)	

※　表 9 の API のプロトタイプ宣言は、itcpip.h ファイルに含まれています。

表 7、表８および表９に示した API は、下記の３つに分類できます。

1. アプリケーションプログラムの中から呼び出す必要のある API
2. T2 ライブラリ内部から呼び出される API で、アプリケーションからは呼び出す必要のない API（表中ではグレーの背景で記載）
3. ドライバ API の作成を補助する API で、アプリケーションからは呼び出す必要のない API

本マニュアルでは、アプリケーションプログラムから呼び出す必要のある上記 1 に分類される API について、使用方法を説明します。
各 API の仕様については、別紙の「Ethernet ドライバ・インタフェース仕様書」「PPP ドライバ・インタフェース仕様書」を参照いただき、仕様に従ってドライバ関数を作成してください。

【補足事項 1】
表 9に示した API の内、関数 ppp_open、ppp_close、ppp_api_req は PPP ドライバを制御する関数ですが、実体は T2 ライブラリに組み込まれています（スタックサイズは各リリースノートを参照ください）。
これに対して、その他の API はすべて Ethernet ドライバまたは PPP ドライバとして T2 ライブラリから分離されており、各 API のサンプルソースはドライバのサンプルプログラムとして提供されます。

Ethernet ドライバ・オープン　　lan_open()

＜　API 書式　＞

```
ER  lan_open( void )
```

＜　引数　＞

なし

＜　戻り値・エラーコード　＞

E_OK	正常終了
負の値	初期化失敗

＜　機能　＞

Ethernet コントローラ、**Ethernet** ドライバの初期化を行います。**T2** のコンフィグレーションファイルで設定するグローバル変数**_myethaddr** が **0** の場合には、**ROM** に格納されている **Ethernet** アドレスを **Ethernet** コントローラに設定し、かつ、**_myethaddr** にコピーします。**_myethaddr** が **0** でない場合には、その値を **Ethernet** コントローラに設定します。グローバル変数**_myethaddr** の設定方法は、**3**章を参照ください。

本 **API** は、**T2** ライブラリの初期化関数 **tcpudp_open()**の後で呼び出して下さい。

Ethernet ドライバ・クローズ　　lan_close()

＜　API 書式　＞

```
ER  lan_close( void )
```

＜　引数　＞

なし

＜　戻り値・エラーコード　＞

E_OK	正常終了
負の値	終了処理失敗

＜　機能　＞

Ethernet コントローラの停止、**Ethernet** ドライバの終了処理を行います。
本 **API** は、**T2** ライブラリの終了関数 **tcpudp_close()**より前で呼び出して下さい。

PPP 開始処理　　ppp_open()

＜ API 書式 ＞

```
ER  ppp_open( void )
```

＜ 引数 ＞

なし

＜ 戻り値・エラーコード ＞

E_OK	正常終了
負の値	初期化失敗

＜ 機能 ＞

PPP サーバとのリンクを確立し、PAP による認証を行い、IPCP によりネットワークをオープン状態にします。PAP では、T2 のコンフィグレーションファイルで設定するグローバル変数 UB user_name[]、UB user_passwd[]に格納されたユーザ名とパスワードを用いて認証を行います。IPCP では、T2 のコンフィグレーションファイルで設定する変数 tcpudp_env の IP アドレスが 0 でない場合、PPP サーバに特定の IP アドレスの割り当てを、また、変数 tcpudp_env の IP アドレスが 0 の場合には、PPP サーバによる IP アドレスの自動割当を要求します。特定の IP アドレスを要求した場合でも、PPP サーバがその IP アドレスを許可するとは限りません。最終的に PPP サーバが許可した IP アドレスは、変数 tcpudp_env の IP アドレスに格納されます。

本 API は、T2 ライブラリの初期化関数 tcpudp_open()の後で呼び出して下さい。

PPP 終了処理　　ppp_close()

＜　API 書式　＞

```
ER  ppp_close( void )
```

＜　引数　＞

なし

＜　戻り値・エラーコード　＞

E_OK	正常終了
負の値	終了処理失敗

＜　機能　＞

PPP サーバとの接続の切断処理を行います。

本 API は、T2 ライブラリの終了関数 tcpudp_close()より前で呼び出して下さい。

PPP 状態参照　　ppp_status()

＜ API 書式 ＞

```
UH  ppp_status( void )
```

＜ 引数 ＞

なし

＜ 戻り値・エラーコード ＞

PPP の接続状態に応じて、以下の値を返します。

0x0001(PS_DEAD)	リンク切断状態
0x0002(PS_ESTABLISH)	LCP フェーズ
0x0004(PS_AUTHENTICATE)	認証フェーズ(PAP)
0x0008(PS_NETWORK)	NCP フェーズ(IPCP)
0x0010(PS_NETOPEN)	ネットワーク確立
0x0020(PS_TERMINATE)	リンク切断中

＜ 機能 ＞

PPP の接続状態を返します。

SIO 初期化処理　　sio_open()

＜ API 書式 ＞

```
void  sio_open( UB rate )
```

＜ 引数 ＞

UB	rate	シリアル I/O のボーレート設定
		0: (BR96)　9600bps
		1: (BR192)　19200bps
		2: (BR288)　28800bps
		3: (BR384)　38400bps
		4: (BR576)　57600bps
		5: (BR1152)　115200bps

＜ 戻り値・エラーコード ＞

なし

＜ 機能 ＞

使用するシリアル I/O の初期設定を行い、送受信できる状態にします。ボーレートは、パラメータ rate に各ボーレートを示す値（0～5）で渡されます。これらの値は、ヘッダファイル itcpip.h 内で BR96～BR1152 に定義されています。

SIO 終了処理　　sio_close()

＜　API 書式　＞

```
void  sio_close( void )
```

＜　引数　＞

なし

＜　戻り値・エラーコード　＞

なし

＜　機能　＞

使用するシリアル I/O の送受信割り込みを禁止し、送受信できない状態にします。

モデム初期化処理　　modem_open()

＜　API 書式　＞

```
ER  modem_open ( void )
```

＜　引数　＞

なし

＜　戻り値・エラーコード　＞

0	正常終了（モデム接続成功）
-1	異常終了（モデム接続失敗）

＜　機能　＞

モデムの初期化と、電話回線の接続を行い、モデム接続を確立します。モデムの初期化では、ユーザ指定の初期化 AT コマンド at_commands[]を用いてモデムの動作設定を行います。また、電話回線の接続では、モデムを用いて宛先電話番号 peer_dial[]に電話を掛けます。

モデム終了処理　　modem_close()

＜　API 書式　＞

```
ER  modem_close( void )
```

＜　引数　＞

なし

＜　戻り値・エラーコード　＞

0	正常終了（モデム切断成功）
-1	異常終了（モデム切断失敗）

＜　機能　＞

モデムをデータ通信モードからコマンドモードに遷移させて、電話回線の切断を行います。

6. サンプルアプリケーションプログラム

6.1 機能

T2 を用いたアプリケーションプログラムのサンプルです。本サンプルは、PC 上の TELNET コマンドから TCP 接続された後、受信した文字に対してエコーを返すサーバ機能のプログラムです。

図 10に本サンプルプログラムの処理フローを示します。ネットワークに、Ethernet を使用するサンプルと PPP を使用するサンプルがあり、フロー図には併記しています。

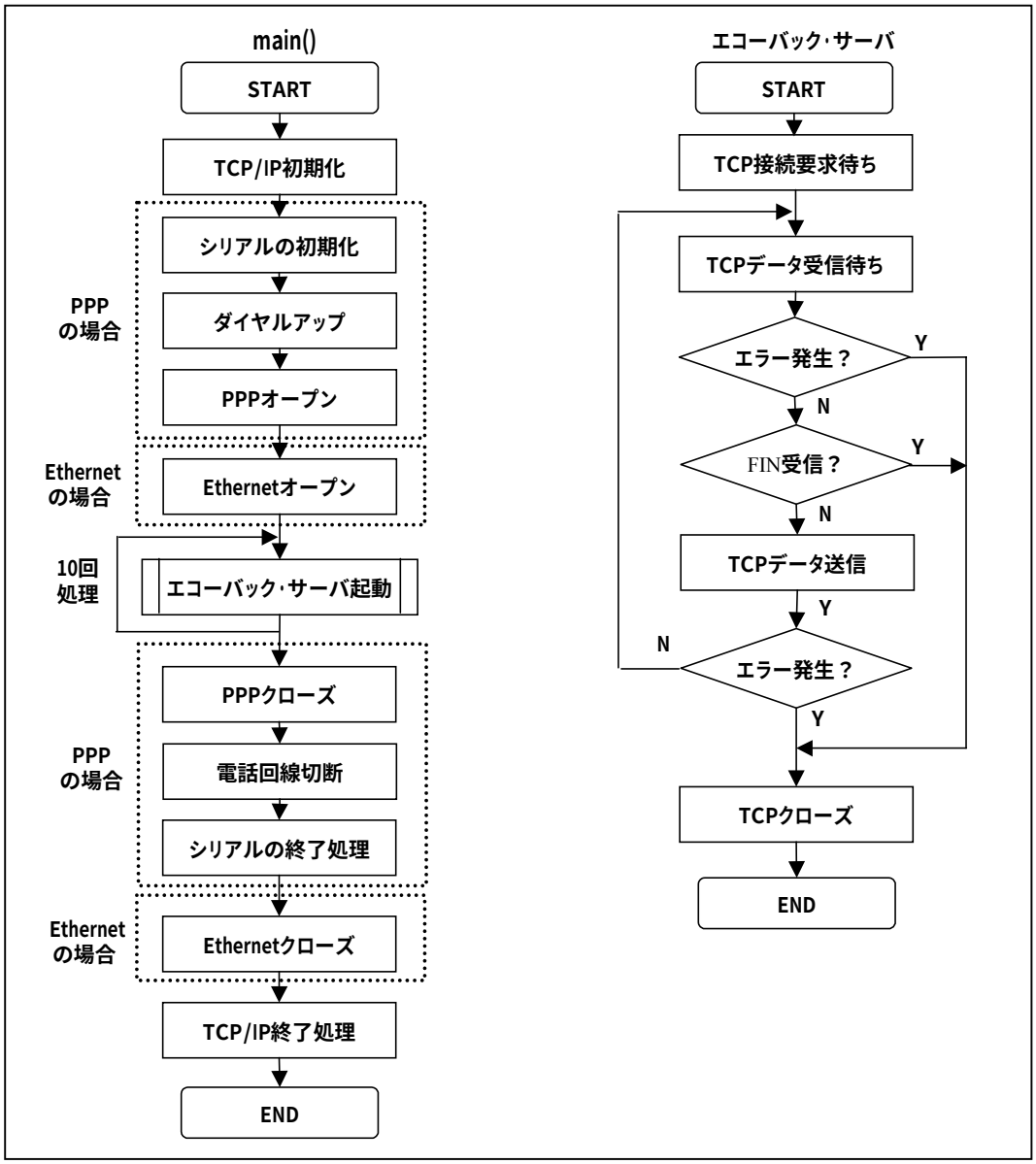

図 10　サンプルアプリケーションプログラムの処理フロー

6.2 実行環境

本サンプル・プログラムを実行する場合のハードウェアの接続方法を、Ethernet については図 11 に、PPP の場合については図 12示します。

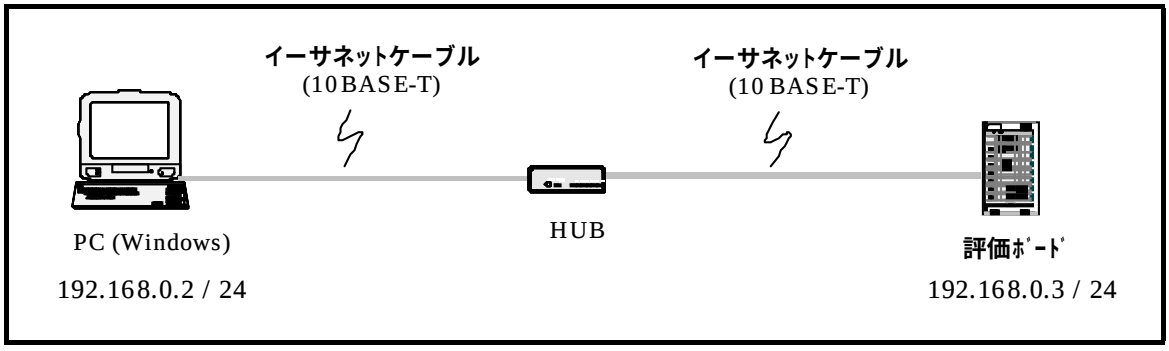

図 11　サンプルアプリケーションプログラムの実行環境 (Ethernet)

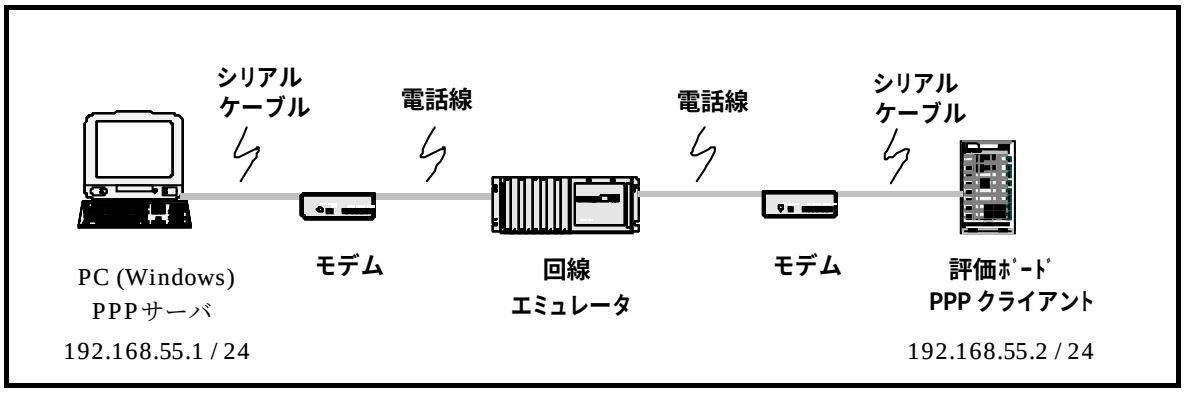

図 12　サンプルアプリケーションプログラムの実行環境 (PPP)

6.3 実行方法

以下の(1)～(7)に、サンプルプログラムの実行方法を示します。Ethernet の場合には(2), (4)～(7)、PPP の場合には(1)～(7)に従って実行して下さい。

(1) PC 上のダイヤルアップサーバの起動
予めダイヤルアップサーバの設定で、ユーザ名："abcde", パスワード："abc00"を登録しておきます。また、ダイヤルアップサーバの TCP/IP の設定で、クライアントに割り当てる IP アドレスが 192.168.55.2 になるように設定しておきます。

(2) サンプルプログラムのダウンロード
評価ボードにサンプルプログラムをダウンロードし、実行します。

(3) PPP サーバへの接続処理
評価ボードからダイヤルアップにより PC 上の PPP サーバへ接続し、PPP のネゴシエーションを実行します。

(4) コネクションの設定
PC 上の MS-DOS プロンプトで下記コマンドを実行し、コネクションを確立します。
telnet 192.168.0.3 1024 (Ethernet の場合)
telnet 192.168.55.2 1024 (PPP の場合)

(5) 文字の送信
TELNET の画面上でキーボードから文字を入力すると、その文字が評価ボードに送信されます。評価ボードでは受信した文字を返信するため、TELNET の画面上に信された文字が表示されます。

(6) コネクションの切断
PC の TELNET 画面のメニュー「接続」→「切断」を選択し、コネクションを切断します。

(7) プログラムの終了
コネクションの接続・終了回数が 10 回に達すると、本サンプルプログラムは終了します。

付録 A　TCP 用 API の戻り値

A1. 意図せず回線が切断された場合の API の動作

ケーブルが外れる等して回線が切断された場合、APIに通知は行いません。APIは、通信相手がハングアップ/リブート等して正常に通信できなくなった場合と同様の動作（戻り値）となります。引数でタイムアウトを設定するAPIについては、タイムアウトにより異常を検知できます。

回線が切断された場合にAPIへ戻り値が返るのは、基本的には、
(T1) APIの引数で指定したタイムアウトに達した時点
または、
(T2) TCPセグメントを送信中に再転送タイムアウトの最大値に達した時点
です。

APIへ戻り値が返るタイミングとしては、APIでタイムアウトにTMO_FEVRを指定した場合は(T2)に該当します。それ以外の場合は、(T1)または(T2)の時間が短い方に該当します。このため、(T2)の時間の方が短い場合、(T2)で指定した時間に達した時点でタイムアウトし、戻り値は(T2)のものとなります。

A2. 各 TCP 用 API の戻り値と通信端点の状態

上記A１．に示す(T1)の場合、(T2)の場合、相手からRSTを受信した場合(R)の各APIの戻り値と通信端点の状態について説明します。

tcp_acp_cep()

【戻り値】		【通信端点の状態】
T1：	E_TMOUT	未使用状態
T2：	戻り値は返さない	引き続き待ち状態(LISTEN)になる
R：	戻り値は返さない	引き続き待ち状態(LISTEN)になる

tcp_con_cep()

【戻り値】		【通信端点の状態】
T1：	E_TMOUT	未使用状態
T2：	E_CLS	未使用状態
R：	E_CLS	未使用状態

tcp_cls_cep()

【戻り値】		【通信端点の状態】
T1：	E_TMOUT	未使用状態
T2：	E_OK	未使用状態
R：	E_OK	未使用状態

tcp_snd_dat()　*1

【戻り値】		【通信端点の状態】
T1：	E_TMOUT	接続状態
T2：	E_CLS	切断状態
R：	E_CLS	切断状態

tcp_rcv_dat()

【戻り値】		【通信端点の状態】
T1：	E_TMOUT	接続状態
T2：	データ等を送信しないので発生しない	
R：	E_CLS	切断状態 *2

【*1】　tcp_snd_dat()の戻り値
　通信相手からFINを受信しても、自局からFINを送信するまではデータを送信できます。このため、相手からFINを受信しても、tcp_snd_dat()は戻り値E_CLSを返しません。
　相手からRSTを受信した場合は、戻り値E_CLSを返します。

【*2】　tcp_rcv_dat()のRST受信時の処理
　通信相手からRSTを受信した場合、受信ウィンドウのデータをすべて読み出してから戻り値E_CLSを返します。受信ウィンドウからデータを読み出している間は、戻り値は読み出したデータサイズとなります。

【補足１】　tcp_sht_cep()の戻り値
　ペンディング状態にならないため、回線が切断されていても通常と同じ戻り値が返ります。

M3S-T2-xxx V.1.01 ユーザーズマニュアル

Rev. 2.00
04.08.01
RJJ10J0629-0200Z

株式会社ルネサス テクノロジ
東京都千代田区大手町2-6-2 日本ビル 〒100-0004