ITRON TCP/IP API 仕様

Ver. 1.00.01

Embedded TCP/IP 技術委員会
(社) トロン協会 ITRON 専門委員会
（高田 広章 編）

まえがき

今から考えると，ITRON 仕様 OS 上の TCP/IP プロトコルスタックが非常に重要であるという認識は数年前からありましたが，ソフトウェアインタフェースの標準化を行う ITRON プロジェクトの視点からは，この分野では BSD UNIX のソケットインタフェースが事実上の標準として広く使われており，新しい API を定義する意義は低いと考えていました．

1996 年頃には，すでに複数の会社が ITRON 仕様 OS 上の TCP/IP プロトコルスタックを開発・販売している状況にありましたが，その中の 1 社の方から，ソケットインタフェースは組込みシステムには必ずしも適当でないため，独自の API を用意しているという話を聞きました．そこで各社の状況を調査したところ多くが同様の状況で，ソケットインタフェースに代わる組込みシステム向けの TCP/IP API 標準化の要求が高いことを認識することとなりました．

このことも 1 つの動機となり，1996 年半ば頃から ITRON 専門委員会でソフトウェア部品の標準化への取り組みを始める中で，分野毎のソフトウェア部品インタフェース標準化の最初のテーマとして TCP/IP プロトコルスタックの API の標準化活動に取り組むことになりました．

ところが，このテーマに取り組むためには，TCP/IP プロトコルスタックに対する深い理解が必要であり，半導体メーカ中心の ITRON 専門委員会メンバだけで進めるのは，能力的にも役割的にも難しい状況でした．そこで 1997 年 3 月には，この分野で技術力を持つ会社に参加を呼びかけて，Embedded TCP/IP 技術委員会を発足させることとなりました．それにあわせて，ITRON Club メイリングリストなどの場で活動を手伝って下さるボランティアを募集したり，インターネット分野の学識経験者に活動への参加を依頼しました．

本仕様書は，1997 年 3 月から 1998 年 4 月までの，Embedded TCP/IP 技術委員会の活動の結果まとめられた「組込みシステムのための標準 TCP/IP API」を，ITRON 専門委員会の審議を経て，1998 年 5 月に「ITRON TCP/IP API 仕様」と認定されたものです．この仕様を完成させることができたのは，ひとえに Embedded TCP/IP 技術委員会メンバ各位の熱心な検討と議論によるものであり，ここに深くお礼申し上げます．

Embedded TCP/IP 技術委員会のメンバの中には，すでに本仕様書に準拠した製品の実装を開始している会社もあり，数カ月以内には，本仕様に準拠した製品の販売が開始されるものと予想しています．

本 API 仕様に準拠した TCP/IP プロトコルスタック製品が複数社から提供されることは，TCP/IP プロトコルスタックのユーザ，開発メーカの両者に大きなメリットがあります．ユーザにとっては，製品選択の幅が広がることで，ソフトウェア部品を外部から導入する際の安心感が増し，結果的に進んだ技術をリーゾナブルなコストで導入できることになります．また開発メーカ間では，競争を通じた技術向上が期待できます．また，TCP/IP API を ITRON 仕様の 1 つと位置付けることにより，TCP/IP プロトコルスタックも組込みシステム向けのソフトウェア製品の 1 つとの位置付けを確かなものにするものと期待できます．

開発メーカの立場からは, 標準化によって厳しい価格競争により苦しくなるという声もないわけではありませんが, 厳しい競争を経験しなかった業界は業界全体として地盤沈下するのは必至です. Embedded TCP/IP 技術委員会の活動が, 組込みシステムのためのソフトウェア部品業界, さらには組込みシステム業界全体の発展につながることを願ってやみません.

最後になりましたが, Embedded TCP/IP 技術委員会の活動を支援して下さった ITRON 専門委員会メンバならびにトロン協会事務局の皆様に感謝の意を表したいと思います.

1998 年 5 月

高田 広章（豊橋技術科学大学 情報工学系）
ITRON 専門委員会 幹事・
Embedded TCP/IP 技術委員会 幹事

目次

1章 はじめに

1.1 標準化の背景と範囲

近年の組込みシステムの大規模化・複合化の流れの中で，アプリケーションソフトウェアの開発のベースを引き上げる必要性から，組込みシステムのためのソフトウェア部品の重要性が広く認識されつつある．ソフトウェア部品が広く使われるようになると，その標準化が重要な課題となる．このような背景から，ITRON プロジェクトにおいても，ニーズの高い分野を選んで，ソフトウェア部品のインタフェースの標準化に取り組んでいる．

TCP/IP プロトコルスタックは，組込みシステムのためのソフトウェア部品の中でも最も重要なものの一つである．現在，TCP/IP のアプリケーションプログラムインタフェース（API）としては，ソケットインタフェースが広く使われている．ところが，ソケットインタフェースには組込みシステム（特に小規模なシステム）には不都合な点があり，組込みシステムに適した TCP/IP API の標準化が求められている．

Embedded TCP/IP 技術委員会は，組込みシステムに適した TCP/IP API の標準化を行うために，ITRON 専門委員会の呼びかけにより，1997 年 3 月に結成された．本仕様書は，Embedded TCP/IP 技術委員会における 1997 年 3 月から 1998 年 4 月までの検討の結果作成されたもので，1998 年 5 月に ITRON 専門委員会によって ITRON 仕様に認定された．本仕様は，TRON プロジェクトの基本ポリシーに従い，誰もが自由にライセンス料なしで利用することができるオープンな仕様である．

本仕様は，IPv4 上の TCP と UDP の両プロトコルの C 言語からの API の標準仕様を定めるものである．現状で，IPv6 の仕様が完全にフィックスしていないことから，IPv6 への適用方法は別途検討する．

実際の TCP/IP プロトコルスタックには，本仕様で定める API 以外に，インタフェースの制御（IP アドレスやネットマスク，ブロードキャストアドレスの設定 / 参照など），ルーティングテーブルの設定 / 参照，ARP テーブルの設定 / 参照といった API を持たせる必要があるが，本仕様ではそのような API の標準は定めていない．また，ICMP の扱いについては実装依存とし，本仕様では定めていない．

1.2 組込みシステムにおける要求事項

一般に組込みシステムには，以下のような特徴がある．

(1) ハードウェア資源（プロセッサの能力，メモリ容量）に対する制約が厳しい．そのため，プログラムやデータのサイズを小さく押え，プロセッサの性能をフルに発揮させることが必要になる．

(2) 何らかのリアルタイム性が求められる．そのため，プロトコルスタックの振舞いに対しても，きめ細かい制御が必要になる．

(3) 高い信頼性が求められる．

具体的には, 組込みシステムのためのプロトコルスタックに求められる性質として，以下のものが挙げられる．

(a) 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．

(b) 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．

(c) 非同期インタフェー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．

(d) API 毎のエラーの詳細がわかることが望ましい．

ソケットインタフェースでは，これらの要求事項のいくつかを満たすことができない．例えば，ソケットインタフェースでは，ソケットの管理ブロックや，受信した UDP パケットをキューイングするためのメモリ領域を動的に管理する必要がある．また後者については，メモリ領域のサイズの上限をプロトコル上からは定めることができず，プロトコルスタック内でメモリが足りなくなる事態が避けられない．この様に，アプリケーションプログラムから見えないところでメモリ不足が起こるのは，一定の振舞いを求められる組込みシステムには好ましくないと考えられる．

またソケットインタフェースは，UNIX のプロセスモデルに整合するよう設計されているため，ITRON 仕様 OS などのリアルタイム OS におけるタスクモデルとは相性が悪い面もある．

1.3 API の設計方針

本仕様書で定義する TCP/IP API は，前述の要求事項を満たす TCP/IP プロトコルスタックを実現できるものとすることを目指している．それに加えて，以下の設計方針を設定した．

- ソケットインタフェースに慣れているプログラマが多いことや，ソケットインタフェースを用いて開発されたソフトウェアが多いことから，ソケットインタフェースをベースとしたものとする．また，定義した API の上にライブラリをかぶせ，ソケットインタフェースと互換の API を実現できるようにする．
- 小規模な組込みシステムでは，システム構築時に決定する静的な設定で十分なものがあることを考慮する．
- 主に ITRON 仕様 OS 上での利用を想定し, ITRON 仕様のコンベンションを踏襲してAPIを定めるが, 他のリアルタイムOSにも適用可能なように配慮する．

1.4 共通の概念

1.4.1 API のレベル分けと静的 API

- 各 API やその機能を, 必要な度合いに応じて標準機能と拡張機能に分類する．市販される TCP/IP プロトコルスタックには, 少なくとも標準機能を実装することを推奨する．

- オブジェクトを生成する各 API に対して，システム構成ファイル中に記述することで，プロトコルスタック初期化時にオブジェクトを自動的に生成する静的な設定方法（これを，静的 API と呼ぶ）を用意する．静的 API は，API 名を大文字で記述することで，通常の API（これを，動的 API と呼ぶ）と区別する．

1.4.2 API の返値とエラーコード

- 各 API の返値は，ITRON 仕様のコンベンションに従い，エラーが発生した場合には負の値のエラーコード，正常に実行された場合には 0 または正の値とする．正常実行された場合の返値の意味は API 毎に定義される．
- エラーコードは，メインエラーコードとサブエラーコードで構成される．メインエラーコード，サブエラーコード共に負の値とし，それらを組み合わせたエラーコードも負の値とする．
- メインエラーコードのニーモニックと値および意味は，ITRON カーネル仕様のエラーコードと同じになるように標準化する．ただし，ITRON カーネル仕様で足りないエラーコード（E_WBLK，E_CLS，E_BOVR）は追加定義する．μITRON3.0 仕様では，8 ビットで表現できる範囲（負の数なので，–128 〜 –1）でエラーコードの値が定められている．
- サブエラーコードについては実装依存とする．この仕様書では，各 API が返すエラーコードとして，メインエラーコードのみを定義する．
- 以下のエラーコードは，API 毎には記述しないが，すべての API が返す可能性がある．具体的にどの API がどのエラーを返す可能性があるかは実装依存である．

E_SYS	システムエラー（プロトコルスタックの内部エラー）
E_NOMEM	メモリ不足
E_NOSPT	未サポート機能
E_MACV	メモリアクセス違反

1.4.3 通信端点

- IP アドレスとポート番号で識別される各プロトコルの通信端点（end point）を，ITRON カーネル仕様のオブジェクトに相当するものと考える．通信端点は，プロトコルや機能毎に別種類のものとする．通信端点は，ソケットインタフェースのソケットに相当するものであるが，この意味でソケットよりも抽象度が低い．
- 通信端点は，その種類毎にシステム全体でユニークな ID 番号で識別する．
- 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に相当する抽象化は行わず，通信端点を直接 API から操作する．この違いが最も顕著にあらわれるのは TCP の切断手順である．ソケットインタフェースにおける close は，ファイルディスクリプタとソケットの対応関係を切り離すために，データの送信が終れば切断手順の完了を待たずに close からリターンし，リターンした直後から同じファイルディスクリプタを再利用できる．それに対して本仕様の tcp_cls_cep は，（タイムアウトなしの場合）同じ通信端点が再利用可能になるまで待ち状態となる．

1.4.4 タイムアウトとノンブロッキングコール

- 待ち状態に入る可能性のある API には，タイムアウトとノンブロッキングコールを用意する．
- タイムアウトは，一定時間経過しても処理が完了しない場合に，処理をキャンセルして API からリターンするものである（この時, API からは E_TMOUT エラーが返る）．そのため，タイムアウトが起った場合には，API を呼び出したことでオブジェクト状態は変化していないのが原則である．ただし，API の機能上，処理のキャンセル時に元の状態に戻せない場合は例外とする．
- ノンブロッキングコールは，API の中で待ち状態に入るべき状況になった場合に, 処理を継続したままAPIからリターンする（この時, APIからはE_WBLK エラーが返る）．そのため，API からリターンしても処理の実行は継続しており，処理が完了した時点（ないしは，処理がキャンセルされた時点）で，次節で説明するコールバックを用いてアプリケーションプログラムに処理完了を通知する．また，API からリターンした後も処理が継続しているため，データ領域へのポインタを渡す API をノンブロッキング指定で呼び出した場合，処理が完了するまで，それらの領域を別の目的に使ってはならない．
- ポーリングはタイムアウト時間を 0 としたタイムアウト処理である．ノンブロッキングコールとは，処理がキャンセルされるか継続されるかの違いがある．
- API の中で待ち状態になっている場合またはノンブロッキングコールによる処理が継続している場合，API による処理がペンディングしているという．
- 本仕様中の API の説明では，タイムアウトがない（永久待ちの）場合の振舞いを説明している．API の機能説明中で「待ち状態となる」とある場合でも，タイムアウト指定をした場合には，指定時間経過後に待ち状態が解除され，E_TMOUT を返値として API からリターンする．またノンブロッキングコール指定をした場合には，待ち状態に入らずに，E_WBLK を返値として API からリターンする．
- タイムアウト値は，ITRON カーネル仕様にあわせて，正の値でタイムアウト時間（単位はミリ秒を推奨），TMO_POL（= 0）でポーリング，TMO_FEVR（= –1）で永久待ちをあらわすこととする．また，TMO_NBLK（= –2）でノンブロッキングコールをあらわすこととする．

1.4.5 コールバック

- プロトコルスタックで起こった事象をアプリケーションプログラムに伝えるために，コールバックを用いる．コールバックを起こす事象は，大きく分けて，ノンブロッキングコールの処理完了通知とその他の事象に分類できる．
- コールバックルーチンは，アプリケーションプログラムが定義して，実装依存のコンテキストで実行される．そのためコールバックルーチンは，どのようなコンテキストでも実行できるように記述すべきである. また, コールバックルーチン内で待ち状態に入ってはならない.
- コールバックルーチンは, 各通信端点に対して 1 つづつ定義される（ただし, TCP 受付口はコールバックルーチンを持たない）．コールバックを起こした事象の種類は，コールバックルーチンへの引数として渡される．

- 同一の通信端点に対するコールバックルーチンはネストして呼ばれることはない．言い換えると，コールバックルーチンからリターンするまでは，同一の通信端点に対するコールバックルーチンが呼び出されることはない．

【仕様決定の理由】本仕様のコールバックは，ソフトウェアアーキテクチャ上は安全性が低いといった問題点があるが，ITRON 仕様 OS が主なターゲットとしている小規模な組込みシステムでは，実行時効率を重視すべきと考えた．

1.4.6 API とタスクの関係

- 本仕様の API は，パラメータが同じであれば，どのタスクから呼び出しても同じように機能する．すなわち，本仕様の API によってタスクに割り付けられる資源はない．よって，タスク終了時にプロトコルスタックが解放すべき資源はない．
- 本仕様の API の中で待ち状態に入っているタスクに対して rel_wai を発行した場合，API から E_RLWAI エラーが返る．また，同じ状況で ter_tsk を発行した場合の振舞いは実装依存である．

1.5 共通定義

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

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

【補足説明】ipaddr フィールドには，bcopy を使わず直接値を代入することができる．

(2) オブジェクト生成用のデータ構造

```
typedef struct t_tcp_crep {
        ATR             repatr;         /* TCP 受付口属性 */
        T_IPV4EP        myaddr;         /* 自分側の IP アドレスとポート番号 */
        （他に実装依存のフィールドがあってもよい）
} T_TCP_CREP;

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 */
        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 */
        INT             rbufsz;         /* 受信用ウィンドウバッファのサイズ */
        FP              callback;       /* コールバックルーチン */
        （他に実装依存のフィールドがあってもよい）
} T_TCP_CCEP;
```

```
typedef struct t_udp_ccep {
        ATR             cepatr;         /* UDP 通信端点属性 */
        T_IPV4EP        myaddr;         /* 自分側の IP アドレスとポート番号 */
        FP              callback;       /* コールバックルーチン */
        （他に実装依存のフィールドがあってもよい）
} T_UDP_CCEP;
```

【補足説明】各通信端点の属性を使って，通信端点オプションの初期値を決めるなど，細かな動作の違いを指定することができる．

1.5.2 ユーティリティ関数 / マクロ

(1) バイトオーダ変換関数 / マクロ

UW nl = htonl(UW hl)	ホストバイトオーダの 32 ビット値を, ネットワークバイトオーダに変換する.
UH ns = htons(UH hs)	ホストバイトオーダの 16 ビット値を, ネットワークバイトオーダに変換する.
UW hl = ntohl(UW nl)	ネットワークバイトオーダの 32 ビット値を, ホストバイトオーダに変換する.
UH hs = ntohs(UH ns)	ネットワークバイトオーダの 16 ビット値を, ホストバイトオーダに変換する.

(2) エラーコード取出し関数 / マクロ

ER mercd = mainercd(ER ercd)	エラーコードからメインエラーコードを取り出す.
ER sercd = subercd(ER ercd)	エラーコードからサブエラーコードを取り出す.

1.5.3 定数

(1) 一般

NADR	–1	無効アドレス

(2) API の機能コードと事象のコード

TFN_TCP_CRE_REP	–0x201 (0xfdff)
TFN_TCP_DEL_REP	–0x202 (0xfdfe)
TFN_TCP_CRE_CEP	–0x203 (0xfdfd)
TFN_TCP_DEL_CEP	–0x204 (0xfdfc)
TFN_TCP_ACP_CEP	–0x205 (0xfdfb)
TFN_TCP_CON_CEP	–0x206 (0xfdfa)
TFN_TCP_SHT_CEP	–0x207 (0xfdf9)
TFN_TCP_CLS_CEP	–0x208 (0xfdf8)
TFN_TCP_SND_DAT	–0x209 (0xfdf7)
TFN_TCP_RCV_DAT	–0x20a (0xfdf6)

TFN_TCP_GET_BUF	–0x20b (0xfdf5)
TFN_TCP_SND_BUF	–0x20c (0xfdf4)
TFN_TCP_RCV_BUF	–0x20d (0xfdf3)
TFN_TCP_REL_BUF	–0x20e (0xfdf2)
TFN_TCP_SND_OOB	–0x20f (0xfdf1)
TFN_TCP_RCV_OOB	–0x210 (0xfdf0)
TFN_TCP_CAN_CEP	–0x211 (0xfdef)
TFN_TCP_SET_OPT	–0x212 (0xfdee)
TFN_TCP_GET_OPT	–0x213 (0xfded)
TFN_TCP_ALL	0

TEV_TCP_RCV_OOB	0x201	TCP 緊急データの受信

（その他の事象は実装依存に定義する）

TFN_UDP_CRE_CEP	–0x221 (0xfddf)
TFN_UDP_DEL_CEP	–0x222 (0xfdde)
TFN_UDP_SND_DAT	–0x223 (0xfddd)
TFN_UDP_RCV_DAT	–0x224 (0xfddc)
TFN_UDP_CAN_CEP	–0x225 (0xfddb)
TFN_UDP_SET_OPT	–0x226 (0xfdda)
TFN_UDP_GET_OPT	–0x227 (0xfdd9)
TFN_UDP_ALL	0

TEV_UDP_RCV_DAT	0x221	UDP パケットの受信

（その他の事象は実装依存に定義する）

(3) メインエラーコード

E_OK	0	正常終了
E_SYS	–5	システムエラー
E_NOMEM	–10	メモリ不足
E_NOSPT	–17	未サポート機能
E_RSATR	–24	予約属性
E_PAR	–33	パラメータエラー
E_ID	–35	不正 ID 番号
E_NOEXS	–52	オブジェクト未生成
E_OBJ	–63	オブジェクト状態エラー
E_MACV	–65	メモリアクセス違反
E_QOVR	–73	キューイングオーバフロー
E_DLT	–81	待ちオブジェクトの削除
E_WBLK	–83	ノンブロッキングコール受付け
E_TMOUT	–85	ポーリング失敗またはタイムアウト
E_RLWAI	–86	処理のキャンセル, 待ち状態の強制解除
E_CLS	–87	接続の切断
E_BOVR	–89	バッファオーバフロー

【補足説明】ITRON カーネル仕様と共通のエラー（E_WBLK, E_CLS, E_BOVR 以外）には, ITRON カーネル仕様と同一のエラーコードを割り当てている.

E_WBLK は，ノンブロッキングコールが受け付けられ，処理が継続したままリターンしたことを示す．処理の完了（ないしはキャンセル）は，コールバックを用いて通知される．

E_CLS は，接続が異常切断されたことを示す．送受信などの処理の完了を待っている間に接続が異常切断されることもあるため，場合によっては待ち状態の解除を伴う．

E_BOVR は，受信データを入れる領域の長さが足りない場合で，領域いっぱいまでデータを取り出し，入りきらないデータを捨てたことを示す．他のエラーと異なり，このエラーコードが返った場合，API の処理が行われて，オブジェクトの状態が変化している．

【仕様決定の理由】接続の異常切断によるエラーを新設したのは，待ち状態の解除を伴うことから，E_OBJ とするのは適切でないと考えたためである．API の呼出し時点ですでに接続が切断されていた場合には E_OBJ エラーとし，待ち状態に入った後に接続が切断された場合に E_CLS エラーとする仕様も考えられるが，アプリケーションには扱いにくいと考えた．

(4) タイムアウト指定

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

(5) 特殊な IP アドレスとポート番号

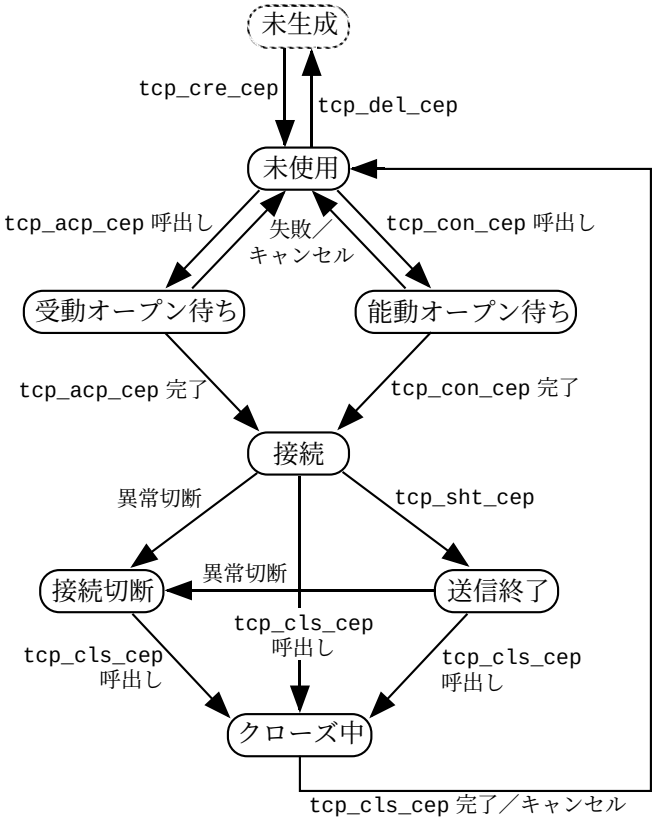
IPV4_ADDRANY	0	IP アドレスの指定省略
TCP_PORTANY	0	TCP ポート番号の指定省略
UDP_PORTANY	0	UDP ポート番号の指定省略

2 章 TCP の API

2.1 概要

- TCP に使う通信端点として，相手側からの接続要求を待ち受けるための TCP 受付口（TCP Reception Point，rep と略す）と，接続の端点として用いる TCP 通信端点（TCP Communication End Point，cep と略す）を用意する．
- TCP 通信端点は，API 上，「未生成」，「未使用」，「受動オープン待ち」，「能動オープン待ち」，「接続」，「送信終了」，「接続切断」，「クローズ中」の 8 つの状態を遷移する（図参照）．未生成以外の 7 つの状態のいずれかにある場合を「生成済み」，未生成および未使用以外の 6 つの状態のいずれかにある場合を「使用中」と呼ぶ．また，接続，送信終了，接続切断以外の 5 つの状態のいずれかにある場合を「未接続」と呼ぶ．
- ソケットインタフェースの read/write に相当する API に加えて，データのコピー回数を減らせる可能性のあるより効率の良い API も用意する．これを省コピー API と呼ぶ．

図 1. TCP 通信端点の状態遷移



【仕様決定の理由】本仕様で用意する省コピー API は，プロトコルスタックが管理するバッファにユーザが直接アクセスすることを許すものである．より効率を上げられる可能性のある方法として，アプリケーション側が管理するバッファをプロトコルスタックに直接使わせる方法が考えられる．この方法も有力ではあるが，アプリケーションプログラムが複雑化することや，データ長が短

い場合には効率が上がるとは限らないなどの理由から，本仕様では採用していない．

また，付録 D.1 に示す通り，省コピー API を用いて標準 API を実現することもできるが，プロトコルスタックの実現方法によってはそのような実現が効率が良いとは限らないため，両方の API を用意している．

2.2 TCP 受付口の生成 / 削除

TCP_CRE_REP TCP 受付口の生成 静的 API…【標準機能】
tcp_cre_rep 動的 API…【拡張機能】

【静的 API】
TCP_CRE_REP(ID repid, { ATR repatr, { UP myipaddr, UH myportno }});

【C 言語 API】
ER ercd = tcp_cre_rep(ID repid, T_TCP_CREP *pk_crep);

【パラメータ】

ID	repid	TCP 受付口 ID
T_TCP_CREP	*pk_crep	TCP 受付口生成情報

pk_crep の内容

ATR	repatr	TCP 受付口属性
T_IPV4EP	myaddr	自分側の IP アドレス（myipaddr）とポート番号（myportno）

（他に実装依存のパラメータがあってもよい）

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
E_ID	不正 ID 番号
E_RSATR	予約属性
E_PAR	パラメータエラー（pk_crep のアドレス，IP アドレスポート番号が不正）
E_OBJ	オブジェクト状態エラー（指定した ID の TCP 受付口が生成済み，ポート番号既使用，特権ポート違反）

【API の機能】指定した ID の TCP 受付口を生成し，指定した IP アドレスとポート番号で接続待ち状態にする．自分側の IP アドレスに IPV4_ADDRANY（= 0）を指定した場合，自分の持つすべての IP アドレスに対する接続要求を待ち受ける．IP アドレスを指定した場合には，指定した IP アドレス宛の接続要求のみを待ち受ける．

TCP 受付口属性の使い方，特権ポートの扱いは実装依存．

【仕様決定の理由】TCP 受付口毎に接続要求の最大キューイング数を指定する方法は標準的には用意していない．接続要求をキューイングすることは意味があるが，その場合にも，最大キューイング数をシステム全体で決める方法で十分と考えられるため，実装依存に対応することとする．

---

tcp_del_rep TCP 受付口の削除 動的 API…【拡張機能】

【C 言語 API】

ER ercd = tcp_del_rep(ID repid);

【パラメータ】

ID	repid	TCP 受付口 ID

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成

【API の機能】指定した TCP 受付口を削除し，接続待ち状態を解除する．削除された TCP 受付口で接続要求を待っている tcp_acp_cep には E_DLT エラーが返る．

2.3 TCP 通信端点の生成 / 削除

---

tcp_CRE_CEP TCP 通信端点の生成 静的 API…【標準機能】
tcp_cre_cep 動的 API…【拡張機能】

【静的 API】

TCP_CRE_CEP(ID cepid, { ATR cepatr, VP sbuf, INT sbufsz, VP rbuf,
INT rbufsz, FP callback });

【C 言語 API】

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

【パラメータ】

ID	cepid	TCP 通信端点 ID
T_TCP_CCEP	*pk_ccep	TCP 通信端点生成情報

pk_ccep の内容

ATR	cepatr	TCP 通信端点属性
VP	sbuf	送信用のウィンドウバッファの先頭アドレス
INT	sbufsz	送信用のウィンドウバッファのサイズ
VP	rbuf	受信用のウィンドウバッファの先頭アドレス

INT	rbufsz	受信用のウィンドウバッファのサイズ
FP	callback	コールバックルーチンのアドレス

（他に実装依存のパラメータがあってもよい）

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
E_ID	不正 ID 番号
E_RSATR	予約属性
E_PAR	パラメータエラー（pk_ccep のアドレス，sbuf，sbufsz，rbuf，rbufsz，callback が不正）
E_OBJ	オブジェクト状態エラー（指定した ID の TCP 通信端点が生成済み）

【API の機能】指定した ID の TCP 通信端点を生成する．生成した TCP 通信端点は，受動オープン，能動オープンの両方に使うことができる．

ウィンドウバッファをプロトコルスタック内部で確保する実装では，ウィンドウバッファの先頭アドレスとして NADR（= –1）を指定させるものとする．その場合にも，ウィンドウバッファのサイズの指定は有効である．また，NADR が指定された場合にはウィンドウバッファを内部で確保し，それ以外の場合には与えられたバッファを用いる実装も可能である．TCP 通信端点属性の使い方は実装依存．

tcp_del_cep　TCP 通信端点の削除　動的 API…【拡張機能】

【C 言語 API】

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

【パラメータ】

ID	cepid	TCP 通信端点 ID

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_OBJ	オブジェクト状態エラー（指定した TCP 通信端点が使用中）

【API の機能】指定した TCP 通信端点を削除する．使用中の TCP 通信端点を削除しようとした場合，E_OBJ エラーとなる．

2.4 接続 / 切断

tcp_acp_cep 接続要求待ち（受動オープン） 【標準機能】

【C 言語 API】

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

【パラメータ】

ID	cepid	TCP 通信端点 ID
ID	repid	TCP 受付口
TMO	tmout	タイムアウト指定

【リターンパラメータ】

T_IPV4EP	dstaddr	相手側の IP アドレスとポート番号
ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー（dstaddr のアドレス，tmout が不正）
E_OBJ	オブジェクト状態エラー（指定した TCP 通信端点が使用中）
E_DLT	接続要求を待っている TCP 受付口が削除された
E_WBLK	ノンブロッキングコール受付け
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	処理のキャンセル，待ち状態の強制解除

【API の機能】指定した TCP 受付口に対する接続要求を待つ．接続要求があった場合には，指定した TCP 通信端点を用いて接続を行い，相手側の IP アドレスとポート番号を返す．接続が完了するまで待ち状態となる．

同じ TCP 受付口に対して, 同時に複数の tcp_acp_cep を発行することができる．その場合，接続要求をどの TCP 通信端点に受け付けるかは実装依存である．

tcp_acp_cep を呼び出すことで，TCP 通信端点は受動オープン待ち状態となり，処理完了した時点で接続状態となる．タイムアウトや tcp_can_cep によって tcp_acp_cep の処理がキャンセルされた場合，TCP 通信端点は未使用状態に戻る．

【補足説明】同一の TCP 通信端点に対する tcp_acp_cep がペンディングしている間に tcp_acp_cep を発行した場合, TCP 通信端点が使用中であるために, E_OBJ エラーとなる．

tcp_con_cep 接続要求（能動オープン） 【標準機能】

【C 言語 API】

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

【パラメータ】

ID	cepid	TCP 通信端点 ID
T_IPV4EP	myaddr	自分側の IP アドレスとポート番号
T_IPV4EP	dstaddr	相手側の IP アドレスとポート番号
TMO	tmout	タイムアウト指定

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー（myaddr のアドレス，dstaddr のアドレス，IP アドレス，ポート番号，tmout が不正）
E_OBJ	オブジェクト状態エラー（指定した TCP 通信端点が使用中，ポート番号既使用，特権ポート違反）
E_WBLK	ノンブロッキングコール受付け
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	処理のキャンセル，待ち状態の強制解除
E_CLS	接続が要求が拒否された

【API の機能】指定した TCP 通信端点を用いて，指定した相手側の IP アドレス / ポート番号に対して接続を行う．接続が完了するまで待ち状態となる．

自分側の IP アドレスに IPV4_ADDRANY（= 0），ポート番号に TCP_PORTANY（= 0）を指定した場合，プロトコルスタックでそれらを決定する．また，myaddr に NADR（= –1）を指定した場合，IP アドレス，ポート番号ともにプロトコルスタックで決定する．特権ポートの扱いは実装依存．

tcp_con_cep を呼び出すことで，TCP 通信端点は能動オープン待ち状態となり，処理完了した時点で接続状態となる．タイムアウトや tcp_can_cep によって tcp_con_cep の処理がキャンセルされた場合や接続要求が拒否された場合，TCP 通信端点は未使用状態に戻る．

【補足説明】同一の TCP 通信端点に対する tcp_con_cep がペンディングしている間に tcp_con_cep を発行した場合，TCP 通信端点が使用中であるために，E_OBJ エラーとなる．

【仕様決定の理由】

接続が即時に完了することはないため，タイムアウト指定としてポーリングはほとんど意味がないが，積極的には排除しない方針とした．

tcp_sht_cep　データ送信の終了　【標準機能】

【C 言語 API】
ER ercd = tcp_sht_cep(ID cepid);

【パラメータ】

ID	cepid	TCP 通信端点 ID

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_OBJ	オブジェクト状態エラー（指定した TCP 通信端点が接続状態ではない）

【API の機能】指定した TCP 通信端点に対するデータ送信を終了する．具体的には，送信バッファ中のデータを送信し終わったら，FIN を送って接続の切断手順を行うよう手配する．tcp_sht_cep は，切断する手配を行うだけであるため，API 中で待ち状態になることはない．

tcp_sht_cep を呼んで以降は，TCP 通信端点は送信終了状態となり，それに対するデータの送信はできない．送信しようとすると E_OBJ エラーとなる．データの受信は可能である．

【補足説明】同一の TCP 通信端点に対して tcp_sht_cep を複数回呼んだ場合，2 回目以降の呼び出しは，TCP 通信端点が送信終了状態であるために E_OBJ エラーとなる．

tcp_cls_cep　通信端点のクローズ　【標準機能】

【C 言語 API】
ER ercd = tcp_cls_cep(ID cepid, TMO tmout);

【パラメータ】

ID	cepid	TCP 通信端点 ID
TMO	tmout	タイムアウト指定

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了

E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー（tmout が不正）
E_OBJ	オブジェクト状態エラー（指定した TCP 通信端点が未接続）
E_WBLK	ノンブロッキングコール受付け
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	処理のキャンセル，待ち状態の強制解除

【API の機能】指定した ID の TCP 通信端点がまだ送信終了していない場合は，送信バッファ中のデータを送信し終わるのを待って FIN を送り，接続の切断を行う．また，今後受信したデータは捨てる．TCP 通信端点が未使用状態になるのを待ってリターンするため，tcp_cls_cep からリターンした後は TCP 通信端点をすぐに再利用することができる．

タイムアウトや tcp_can_cep によって tcp_cls_cep の処理がキャンセルされた場合，指定した TCP 通信端点から RST を送信して接続を強制切断し，それぞれ E_TMOUT, E_RLWAI を返値としてリターンする．RST をすぐに送信できない場合には，送信する手配のみを行うか，それもできない場合には RST の送信を省略する．いずれの場合にも，tcp_cls_cep からリターンすると（ノンブロッキング指定をした場合には，完了を通知するコールバックが呼ばれると），TCP 通信端点は未使用状態になっている．

【補足説明】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ならない．TCP/IP プロトコルの規格に従うと，接続状態が完全に終了するまでに数分かかる場合がある．

同一の TCP 通信端点に対する tcp_cls_cep がペンディングしている間に tcp_cls_cep を発行した場合，TCP 通信端点が未接続であるために，E_OBJ エラーとなる．

tcp_cls_cep は，タイムアウトが起った場合には API を呼び出したことでオブジェクト状態は変化していないという原則の例外となっている．

2.5 データの送受信（標準 API）

tcp_snd_dat　　データの送信　　【標準機能】

【C 言語 API】

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

【パラメータ】

ID	cepid	TCP 通信端点 ID
VP	data	送信データの先頭アドレス
INT	len	送信したいデータの長さ

TMO	tmout	タイムアウト指定

【リターンパラメータ】

ER	ercd	送信バッファに入れたデータの長さ／エラーコード

【エラーコード】

正の値	正常終了（送信バッファに入れたデータの長さ）
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー（data，len，tmout が不正）
E_OBJ	オブジェクト状態エラー（指定した TCP 通信端点が未接続または送信終了，tcp_snd_dat または tcp_get_buf がペンディング中）
E_WBLK	ノンブロッキングコール受付け
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	処理のキャンセル，待ち状態の強制解除
E_CLS	TCP 接続が切断された

【API の機能】指定した TCP 通信端点からデータを送信する．データが送信バッファに入った時点でこの API からリターンする．空いている送信バッファ長が送信しようとしたデータ長よりも短い場合，送信バッファが一杯になるまで送信バッファにデータを入れ，送信バッファに入れたデータの長さを返す．送信バッファに空きがない場合には，空きが生じるまで待ち状態となる．

同一の TCP 通信端点に対する tcp_snd_dat か tcp_get_buf がペンディングしている間に tcp_snd_dat を発行すると E_OBJ エラーとなる．

tcp_rcv_dat　　データの受信　　【標準機能】

【C 言語 API】

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

【パラメータ】

ID	cepid	TCP 通信端点 ID
VP	data	受信データを入れる領域の先頭アドレス
INT	len	受信したいデータの長さ
TMO	tmout	タイムアウト指定

【リターンパラメータ】

ER	ercd	取り出したデータの長さ／エラーコード

【エラーコード】

正の値	正常終了（取り出したデータの長さ）
0	データ終結（接続が正常切断された）
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー（data，len，tmout が不正）

E_OBJ	オブジェクト状態エラー（指定した TCP 通信端点が未接続, tcp_rcv_dat または tcp_rcv_buf がペンディング中）
E_WBLK	ノンブロッキングコール受付け
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	処理のキャンセル，待ち状態の強制解除
E_CLS	TCP 接続が切断され，受信バッファが空

【API の機能】指定した TCP 通信端点からデータを受信する．受信バッファに入ったデータを取り出した時点でこの API からリターンする．受信バッファに入っているデータ長が受信しようとしたデータ長よりも短い場合，受信バッファが空になるまでデータを取り出し，取り出したデータの長さを返す．受信バッファが空の場合には，データを受信するまで待ち状態となる．相手側から接続が正常切断され，受信バッファにデータがなくなると，API から 0 が返る．

同一の TCP 通信端点に対する tcp_rcv_dat か tcp_rcv_buf がペンディングしている間に tcp_rcv_dat を発行すると E_OBJ エラーとなる．

【補足説明】TCP 接続が異常切断した場合にも，受信バッファ中にデータがある間は，tcp_rcv_dat で受信データを取り出すことができる．

2.6 データの送受信（省コピー API）

tcp_get_buf 送信用バッファの取得 【標準機能】

【C 言語 API】

```
ER ercd = tcp_get_buf(ID cepid, VP *p_buf, TMO tmout);
```

【パラメータ】

ID	cepid	TCP 通信端点 ID
TMO	tmout	タイムアウト指定

【リターンパラメータ】

VP	buf	空き領域の先頭アドレス
ER	ercd	空き領域の長さ／エラーコード

【エラーコード】

正の値	正常終了（空き領域の長さ）
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー（p_buf，tmout が不正）
E_OBJ	オブジェクト状態エラー（指定した TCP 通信端点が未接続または送信終了，tcp_snd_dat または tcp_get_buf がペンディング中）
E_WBLK	ノンブロッキングコール受付け
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	処理のキャンセル，待ち状態の強制解除

E_CLS	TCP 接続が切断された

【API の機能】送信バッファ中の，次に送信すべきデータを入れることができる空き領域の先頭アドレスと，連続した空き領域の長さを返す．送信バッファに空きがない場合には，空きが生じるまで待ち状態となる．

tcp_get_buf を呼び出したことにより，プロトコルスタックの内部状態は変化しない．そのため，tcp_get_buf を続けて呼び出すと同じ領域が返される（空き領域の長さは長くなるかもしれない）．逆に，tcp_snd_dat，tcp_snd_buf を呼び出すとプロトコルスタックの内部状態は変化する．これらの API を呼び出すと，それ以前に呼び出した tcp_get_buf が返した情報は無効となる．

同一の TCP 通信端点に対する tcp_snd_dat か tcp_get_buf がペンディングしている間に tcp_get_buf を発行すると E_OBJ エラーとなる．

tcp_snd_buf　バッファ内のデータの送信　【標準機能】

【C 言語 API】

```
ER ercd = tcp_snd_buf(ID cepid, INT len);
```

【パラメータ】

ID	cepid	TCP 通信端点 ID
INT	len	データの長さ

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー（len が不正）
E_OBJ	オブジェクト状態エラー（指定した TCP 通信端点が未接続または送信終了，len が長すぎる）
E_CLS	TCP 接続が切断された

【API の機能】tcp_get_buf で取り出したバッファに書き込んだ長さ len のデータを送信するよう手配する．tcp_snd_buf は，送信する手配を行うだけであるため，API 中で待ち状態になることはない．

【補足説明】len が，送信すべきデータを入れることができる連続した空き領域の長さ（tcp_snd_buf を呼ぶことで取り出される値）よりも大きい場合には E_OBJ エラー，どのような条件においても不正な値（例えば負の値）になる場合には E_PAR エラーとするのが原則である．

tcp_rcv_buf 受信したデータの入ったバッファの取得 【標準機能】

【C 言語 API】

ER ercd = tcp_rcv_buf(ID cepid, VP *p_buf, TMO tmout);

【パラメータ】

ID	cepid	TCP 通信端点 ID
TMO	tmout	タイムアウト指定

【リターンパラメータ】

VP	buf	受信データの先頭アドレス
ER	ercd	受信データの長さ／エラーコード

【エラーコード】

正の値	正常終了（受信データの長さ）
0	データ終結（接続が正常切断された）
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー（p_buf，tmout が不正）
E_OBJ	オブジェクト状態エラー（指定した TCP 通信端点が未接続, tcp_rcv_dat または tcp_rcv_buf がペンディング中）
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	処理のキャンセル，待ち状態の強制解除
E_CLS	TCP 接続が切断され，受信バッファが空

【API の機能】受信したデータが入っているバッファの先頭アドレスと，そこから連続して入っているデータの長さを返す．受信バッファが空の場合には，データを受信するまで待ち状態となる．相手側から接続が正常切断され，受信バッファにデータがなくなると，API から 0 が返る．

tcp_rcv_buf を呼び出したことにより，プロトコルスタックの内部状態は変化しない．そのため，tcp_rcv_buf を続けて呼び出すと同じ領域が返される（データの長さは長くなるかもしれない）．逆に，tcp_rcv_dat，tcp_rel_buf を呼び出すとプロトコルスタックの内部状態は変化する．これらの API を呼び出すと，それ以前に呼び出した tcp_rcv_buf が返した情報は無効となる．

同一の TCP 通信端点に対する tcp_rcv_dat か tcp_rcv_buf がペンディングしている間に tcp_rcv_buf を発行すると E_OBJ エラーとなる．

【補足説明】TCP 接続が異常切断した場合にも，受信バッファ中にデータがある間は，tcp_rcv_buf で受信データの先頭アドレスと長さを取り出すことができる．

tcp_rel_buf 受信用バッファの解放 【標準機能】

【C 言語 API】
ER ercd = tcp_rel_buf(ID cepid, INT len);

【パラメータ】

ID	cepid	TCP 通信端点 ID
INT	len	データの長さ

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー（len が不正）
E_OBJ	オブジェクト状態エラー（指定した TCP 通信端点が未接続，len が長すぎる）

【API の機能】tcp_rcv_bufで取り出したバッファ中の長さlenのデータを捨てる．この API 中で待ち状態になることはない．

【補足説明】len が，受信したデータが連続して入っているデータの長さ（tcp_rcv_buf で呼ぶことで取り出される値）よりも大きい場合には E_OBJ エラー，どのような条件においても不正な値（例えば負の値）になる場合には E_PAR エラーとするのが原則である．

2.7 緊急データの送受信

- 本仕様の API では，緊急データは out-of-band データとして扱うことを標準とするが，実装依存に，TCP 通信端点属性や TCP 通信端点オプションの設定によって，in-band データとして扱う方法を用意することもできる．

tcp_snd_oob 緊急データの送信 【拡張機能】

【C 言語 API】
ER ercd = tcp_snd_oob(ID cepid, VP data, INT len, TMO tmout);

【パラメータ】

ID	cepid	TCP 通信端点 ID
VP	data	送信データの先頭アドレス
INT	len	送信したいデータの長さ
TMO	tmout	タイムアウト指定

【リターンパラメータ】

ER	ercd	送信バッファに入れたデータの長さ／エラーコード

【エラーコード】

正の値	正常終了（送信バッファに入れたデータの長さ）
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー（data，len，tmout が不正）
E_OBJ	オブジェクト状態エラー（指定した TCP 通信端点が未接続または送信終了，tcp_snd_oob がペンディング中）
E_WBLK	ノンブロッキングコール受付け
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	処理のキャンセル，待ち状態の強制解除
E_CLS	TCP 接続が切断された

【API の機能】指定した TCP 通信端点から緊急データを送信する．データが送信バッファに入るまで待ち状態となる．

同一の TCP 通信端点に対して tcp_snd_oob の処理がペンディングしている間に tcp_snd_oob を発行すると E_OBJ エラーとなる．

tcp_rcv_oob 緊急データの受信 【拡張機能】

【C 言語 API】

```
ER ercd = tcp_rcv_oob(ID cepid, VP data, INT len);
```

【パラメータ】

ID	cepid	TCP 通信端点 ID
VP	data	受信データを入れる領域の先頭アドレス
INT	len	受信データを入れる領域の長さ

【リターンパラメータ】

ER	ercd	取り出したデータの長さ／エラーコード

【エラーコード】

正の値	正常終了（取り出したデータの長さ）
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー（data，len が不正）
E_OBJ	オブジェクト状態エラー（指定した TCP 通信端点が未接続，緊急データを受信していない）
E_BOVR	バッファオーバフロー

【API の機能】指定した TCP 通信端点に受信した緊急データを取り出す．取り出したデータ長が返される．受信データを入れる領域の長さが，受信した緊急データの長さよりも短い場合には，領域いっぱいまでデータを取り出し，入り

きらないデータは捨てて，E_BOVR を返す．緊急データを受信していない場合，E_OBJ エラーとなる．

【補足説明】緊急データ受信のコールバックルーチン内で呼び出すことを想定している．

2.8 その他の API

tcp_can_cep ペンディングしている処理のキャンセル 【標準機能】

【C 言語 API】

ER ercd = tcp_can_cep(ID cepid, FN fncd);

【パラメータ】

ID	cepid	TCP 通信端点 ID
FN	fncd	キャンセルする API の機能コード

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー（fncd が不正）
E_OBJ	オブジェクト状態エラー（指定した TCP 通信端点に fncd で指定した処理がペンディングしていない）

【API の機能】指定した TCP 通信端点にペンディングしている指定した処理の実行をキャンセルする．キャンセルされた API には，E_RLWAI エラーが返る．また，ノンブロッキングコールをキャンセルした場合には，処理の完了を通知するコールバックルーチンが呼ばれる．

キャンセル可能な処理の API 名と，それを指定する機能コードは次の通り．また，TFN_TCP_ALL（= 0）を指定すると，指定した TCP 通信端点にペンディングしているすべての処理をキャンセルすることができる．

API 名	機能コード
tcp_acp_cep	TFN_TCP_ACP_CEP（= –0x205）
tcp_con_cep	TFN_TCP_CON_CEP（= –0x206）
tcp_cls_cep	TFN_TCP_CLS_CEP（= –0x208）
tcp_snd_dat	TFN_TCP_SND_DAT（= –0x209）
tcp_rcv_dat	TFN_TCP_RCV_DAT（= –0x20a）
tcp_get_buf	TFN_TCP_GET_BUF（= –0x20b）
tcp_rcv_buf	TFN_TCP_RCV_BUF（= –0x20d）
tcp_snd_oob	TFN_TCP_SND_OOB（= –0x20f）
すべて	TFN_TCP_ALL（= 0）

tcp_set_opt TCP 通信端点オプションの設定 【拡張機能】

【C 言語 API】
ER ercd = tcp_set_opt(ID cepid, INT optname, VP optval, INT optlen);

【パラメータ】

ID	cepid	TCP 通信端点 ID
INT	optname	オプションの種類
VP	optval	オプション値を入れた領域のアドレス
INT	optlen	オプション値の長さ

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー（optname，optval，optlen が不正）
E_OBJ	オブジェクト状態エラー（指定したオプションが設定できない状態にある）

【API の機能】指定した TCP 通信端点のオプション値（ソケットオプションに相当）を設定する．optname に設定するオプションの種類，optval にオプション値を入れた領域のアドレス，optlen にオプション値の長さを渡す．設定可能な TCP 通信端点オプションの種類と機能は実装依存.

【補足説明】IP オプションを設定する機能を用意する場合には，この API を利用する.

【仕様決定の理由】TCP 受付口に対するオプションの設定 / 読み出し機能は，必要性が低いと考え用意していない. TCP 受付口生成時に静的にオプションを設定する必要がある場合には，TCP 受付口属性を使って対応可能である．例えば，自分側の IP アドレス / ポート番号の再利用を許すソケットオプション（SO_REUSEADDR）に相当する機能は，この方法で対応できる.

tcp_get_opt TCP 通信端点オプションの読出し 【拡張機能】

【C 言語 API】
ER ercd = tcp_get_opt(ID cepid, INT optname, VP optval, INT optlen);

【パラメータ】

ID	cepid	TCP 通信端点 ID
INT	optname	オプションの種類
VP	optval	オプション値を入れる領域のアドレス
INT	optlen	オプション値を入れる領域の長さ

【リターンパラメータ】

ER	ercd	読み出したオプション値の長さ／エラーコード

【エラーコード】

0 または正の値	正常終了（読み出したオプション値の長さ）
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー（optname，optval，optlen が不正，オプション値を入れる領域が短すぎる）
E_OBJ	オブジェクト状態エラー（指定したオプションが読み出せない状態にある）

【API の機能】指定した TCP 通信端点のオプション値（ソケットオプションに相当）を読み出す．optname に読み出すオプションの種類，optval にオプション値を入れる領域のアドレス，optlen にオプション値を入れる領域の長さを渡す．オプション値を入れる領域の長さが，読み出そうとしているオプション値の長さよりも短い場合，E_PAR エラーとなる．読出し可能な TCP 通信端点オプションの種類と機能は実装依存.

【補足説明】IP オプションを読み出す機能を用意する場合には，この API を利用する．また，デバッグ用に TCP 通信端点の状態を読み出す機能を用意する場合にも，この API を利用する．

2.9 コールバック

共通

【C 言語 API】

ER ercd = callback(ID cepid, FN fncd, VP p_parblk);

【共通パラメータ】

ID	cepid	TCP 通信端点 ID
FN	fncd	事象の種類

【固有パラメータ】

VP	p_parblk	事象に固有なパラメータブロックのアドレス

【返値】

事象の種類に依存

【解説】コールバックルーチンには，事象の発生した TCP 通信端点 ID，事象の種類，事象に固有なパラメータが渡される．事象に固有なパラメータブロックは，コールバックルーチンの中でのみ参照できる．コールバックルーチンの返値の使い方は，事象の種類に依存する．

以下では，標準化されたコールバックについて説明する．その他の事象に対するコールバックのサポートは実装依存.

ノンブロッキングコールの完了通知 【標準機能】

【事象の種類】
終了した API の機能コード

【固有パラメータ】
（以下はパラメータブロックに入れて渡される）
ER ercd API からの返値

【返値】
使わない

【コールバックの機能】ノンブロッキングコールの処理が完了した（ないしはキャンセルされた）場合に呼び出される．API からの返値が，パラメータブロックに入れて渡される．API からのその他のリターンパラメータは，API を呼び出した時に指定した領域に格納される．

【補足説明】例えば, ノンブロッキングの tcp_rcv_buf 処理が完了した場合, バッファに連続して入っている受信データの長さが ercd に渡され, バッファの先頭アドレスは tcp_rcv_buf を呼び出した時のパラメータ p_buf が指す領域に格納される．

緊急データの受信 【拡張機能】

【事象の種類】
TEV_TCP_RCV_OOB（= 0x201）

【固有パラメータ】
（以下はパラメータブロックに入れて渡される）
INT len 緊急データの長さ

【返値】
使わない

【コールバックの機能】緊急データを受信した場合に呼び出される．コールバックルーチンの中で, tcp_rcv_oob を使って緊急データを取り出さなければならない（取り出さずにコールバックルーチンからリターンすると，データは捨てられる）．

3 章 UDP の API

3.1 概要

- UDP に使う通信端点として，UDP 通信端点（UDP Communication End Point, cep と略す）を用意する．
- UDP 通信端点に対して, 相手側の IP アドレスとポート番号をあらかじめ設定する機能（ソケットインタフェースの接続された UDP ソケットに相当）は用意しない．
- あるインタフェースのブロードキャストアドレス宛に送信されたパケットは，そのインタフェースに対してブロードキャストされる．255.255.255.255 宛に送信されたパケットは，ブロードキャストをサポートしているすべてのインタフェースからブロードキャストされる．

【仕様決定の理由】UDP 通信端点に対して相手側の IP アドレスとポート番号をあらかじめ設定する機能は, 本仕様の API の上にライブラリをかぶせて実現することができるため，API をシンプルにするという観点から用意しないこととした．

3.2 UDP 通信端点の生成 / 削除

UDP_CRE_CEP UDP 通信端点の生成 静的 API…【標準機能】
udp_cre_cep 動的 API…【拡張機能】

【静的 API】

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

【C 言語 API】

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

【パラメータ】

ID	cepid	生成する UDP 通信端点 ID
T_UDP_CCEP *pk_ccep		UDP 通信端点生成情報

pk_ccep の内容

ATR	cepatr	UDP 通信端点属性
T_IPV4EP	myaddr	自分側の IP アドレス（myipaddr）とポート番号（myportno）
FP	callback	コールバックルーチンのアドレス

（他に実装依存のパラメータがあってもよい）

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
E_ID	不正 ID 番号
E_RSATR	予約属性
E_PAR	パラメータエラー（pk_ccep のアドレス，IP アドレス，ポート番号，callback が不正）
E_OBJ	オブジェクト状態エラー（指定した ID の UDP 通信端点が生成済み，ポート番号既使用，特権ポート違反）

【API の機能】指定した ID の UDP 通信端点を生成する. 自分側の IP アドレスに IPV4_ADDRANY（= 0）を指定した場合，自分の持つすべての IP アドレス宛とブロードキャストされた UDP パケットを受信し，送信パケットに入る自分側の IP アドレスはプロトコルスタックが決定する. ポート番号に UDP_PORTANY（= 0）を指定した場合，プロトコルスタックでポート番号を決定する．UDP 通信端点属性の使い方，特権ポートの扱いは実装依存.

udp_del_cep　UDP 通信端点の削除　動的 API…【拡張機能】

【C 言語 API】

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

【パラメータ】

ID	cepid	UDP 通信端点 ID

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成

【API の機能】指定した ID の UDP 通信端点を削除する．削除された UDP 通信端点でデータの送受信を待っている API には E_DLT エラーが返る.

3.3 データの送受信

udp_snd_dat　パケットの送信　【標準機能】

【C 言語 API】

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

【パラメータ】

ID	cepid	UDP 通信端点 ID

T_IPV4EP	dstaddr	相手側の IP アドレスとポート番号
VP	data	送信パケットの先頭アドレス
INT	len	送信パケットの長さ
TMO	tmout	タイムアウト指定

【リターンパラメータ】

ER	ercd	送信バッファに入れたデータの長さ／エラーコード

【エラーコード】

正の値	正常終了（送信バッファに入れたデータの長さ）
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー（dstaddr のアドレス，IP アドレス，ポート番号，data，len，tmout が不正）
E_OBJ	オブジェクト状態エラー（udp_snd_dat がペンディング中）
E_QOVR	キューイングオーバフロー
E_DLT	送信を待っている UDP 通信端点が削除された
E_WBLK	ノンブロッキングコール受付け
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	処理のキャンセル，待ち状態の強制解除

【API の機能】指定した UDP 通信端点から，相手側の IP アドレスとポート番号を指定してパケットを送信する．データが送信バッファに入るまで待ち状態となる．

同じ UDP 通信端点に対して，同時に複数の udp_snd_dat を発行することができる．ただし，実装によっては，キューイングできる udp_snd_dat の数に上限があってもよい．その場合，上限を越えて udp_snd_dat を発行すると E_QOVR エラーとなる．

【仕様決定の理由】複数の udp_snd_dat をキューイングするためには，キューを作るためのメモリエリアが必要になるが，キューイングできる数に上限がないと，必要なメモリエリアの上限が押さえられない．そこで，実装依存に，キューイングできる数に上限を設けることを許すしようとした．

udp_rcv_dat　パケットの受信　【標準機能】

【C 言語 API】

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

【パラメータ】

ID	cepid	UDP 通信端点 ID
VP	data	受信パケットを入れる領域の先頭アドレス
INT	len	受信パケットを入れる領域の長さ
TMO	tmout	タイムアウト指定

【リターンパラメータ】

T_IPV4EP	dstaddr	相手側の IP アドレスとポート番号
ER	ercd	取り出したデータの長さ／エラーコード

【エラーコード】

正の値	正常終了（取り出したデータの長さ）
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー（dstaddr のアドレス，data，len，tmout が不正）
E_OBJ	オブジェクト状態エラー（udp_rcv_dat がペンディング中）
E_QOVR	キューイングオーバフロー
E_DLT	受信を待っている UDP 通信端点が削除された
E_WBLK	ノンブロッキングコール受付け
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	処理のキャンセル，待ち状態の強制解除
E_BOVR	バッファオーバフロー

【API の機能】指定した UDP 通信端点からパケットを受信し，相手側の IP アドレスとポート番号を返す．パケットを未受信の場合は，パケットを受信するまでの間待ち状態となる．受信パケットを入れる領域の長さが，受信したパケットの長さよりも短い場合には，領域いっぱいまでデータを取り出し，入りきらないデータは捨てて，E_BOVR を返す．

同じ UDP 通信端点に対して，同時に複数の udp_rcv_dat を発行することができる．ただし，実装によっては，キューイングできる udp_rcv_dat の数に上限があってもよい．その場合，上限を越えて udp_rcv_dat を発行すると E_QOVR エラーとなる．

【仕様決定の理由】複数の udp_rcv_dat をキューイングするためには，キューを作るためのメモリエリアが必要になるが，キューイングできる数に上限がないと，必要なメモリエリアの上限が押さえられない．そこで，実装依存に，キューイングできる数に上限を設けることを許すしようとした．

3.4 その他の API

udp_can_cep　ペンディングしている処理のキャンセル　【標準機能】

【C 言語 API】

```
ER ercd = udp_can_cep(ID cepid, FN fncd);
```

【パラメータ】

ID	cepid	UDP 通信端点 ID
FN	fncd	キャンセルする API の機能コード

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー（fncd が不正）
E_OBJ	オブジェクト状態エラー（指定した UDP 通信端点に fncd で指定した処理がペンディングしていない）

【API の機能】指定した UDP 通信端点にペンディングしている指定した処理の実行をキャンセルする．キャンセルされた API には，E_RLWAI エラーが返る．また，ノンブロッキングコールをキャンセルした場合には，処理の完了を通知するコールバックルーチンが呼ばれる．

キャンセル可能な処理の API 名と，それを指定する機能コードは次の通り．また，TFN_UDP_ALL（= 0）を指定すると，指定した UDP 通信端点にペンディングしているすべての処理をキャンセルすることができる．

API 名	機能コード
udp_snd_dat	TFN_UDP_SND_DAT（= –0x223）
udp_rcv_dat	TFN_UDP_RCV_DAT（= –0x224）
すべて	TFN_UDP_ALL（= 0）

udp_set_opt　UDP 通信端点オプションの設定　【拡張機能】

【C 言語 API】

```
ER ercd = udp_set_opt(ID cepid, INT optname, VP optval, INT optlen);
```

【パラメータ】

ID	cepid	UDP 通信端点 ID
INT	optname	オプションの種類
VP	optval	オプション値を入れた領域のアドレス
INT	optlen	オプション値の長さ

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー（optname，optval，optlen が不正）
E_OBJ	オブジェクト状態エラー（指定したオプションが設定できない状態にある）

【API の機能】指定した UDP 通信端点のオプション値（ソケットオプションに相当）を設定する．optname に設定するオプションの種類，optval にオプション値を入れた領域のアドレス，optlen にオプション値の長さを渡す．設定可能な UDP 通信端点オプションの種類と機能は実装依存.

【補足説明】IP オプションを設定する機能を用意する場合には，この API を利用する．

udp_get_opt　UDP 通信端点オプションの読出し　【拡張機能】

【C 言語 API】

ER ercd = udp_get_opt(ID cepid, INT optname, VP optval, INT optlen);

【パラメータ】

ID	cepid	UDP 通信端点 ID
INT	optname	オプションの種類
VP	optval	オプション値を入れる領域のアドレス
INT	optlen	オプション値を入れる領域の長さ

【リターンパラメータ】

ER	ercd	読み出したオプション値の長さ／エラーコード

【エラーコード】

0 または正の値	正常終了（読み出したオプション値の長さ）
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー（optname，optval，optlen が不正，オプション値を入れる領域が短すぎる）
E_OBJ	オブジェクト状態エラー（指定したオプションが読み出せない状態にある）

【API の機能】指定した UDP 通信端点のオプション値（ソケットオプションに相当）を読み出す．optname に読み出すオプションの種類，optval にオプション値を入れる領域のアドレス，optlen にオプション値を入れる領域の長さを渡す．オプション値を入れる領域の長さが，読み出そうとしているオプション値の長さよりも短い場合，E_PAR エラーとなる．読出し可能な UDP 通信端点オプションの種類と機能は実装依存.

【補足説明】IP オプションを読み出す機能を用意する場合には，この API を利用する．また，デバッグ用に UDP 通信端点の状態を読み出す機能を用意する場合にも，この API を利用する．

3.5 コールバック

共通

【C 言語 API】
ER ercd = callback(ID cepid, FN fncd, VP p_parblk);

【共通パラメータ】

ID	cepid	UDP 通信端点 ID
FN	fncd	事象の種類

【固有パラメータ】

VP	p_parblk	事象に固有なパラメータブロックのアドレス

【返値】
事象の種類に依存

【解説】コールバックルーチンには，事象の発生した UDP 通信端点 ID，事象の種類，事象に固有なパラメータが渡される．事象に固有なパラメータブロックは，コールバックルーチンの中でのみ参照できる．コールバックルーチンの返値の使い方は，事象の種類に依存する．

以下では，標準化されたコールバックについて説明する．その他の事象に対するコールバックのサポートは実装依存.

ノンブロッキングコールの完了通知 【標準機能】

【事象の種類】
終了した API の機能コード

【固有パラメータ】
（以下はパラメータブロックに入れて渡される）

ER	ercd	API からの返値

【返値】
使わない

【コールバックの機能】ノンブロッキングコールの処理が完了（ないしはキャンセル）した場合に呼び出される．API からの返値が，パラメータブロックに入れて渡される．API からのその他のリターンパラメータは，API を呼び出した時に指定した領域に格納される．

【補足説明】例えば，ノンブロッキングの udp_rcv_dat 処理が完了した場合，取り出したデータの長さが ercd に渡され，相手側の IP アドレスとポート番号は udp_rcv_dat を呼び出した時のパラメータ p_dstaddr が指す領域に格納される．

UDP パケットの受信 【標準機能】

【事象の種類】
TEV_UDP_RCV_DAT（= 0x221）

【固有パラメータ】
（以下はパラメータブロックに入れて渡される）
INT len パケットの長さ

【返値】
使わない

【コールバックの機能】udp_rcv_dat がペンディングしていない状態で UDP パケットを受信した時に呼び出される．コールバックルーチンの中で，udp_rcv_dat を使って受信したパケットを取り出さなければならない（取り出さずにコールバックルーチンからリターンすると，パケットは捨てられる）．

4 章 仕様の取扱いなど

4.1 オープン仕様・無保証・仕様書の著作権

本仕様は，TRON プロジェクトの基本ポリシーに従って取り扱われるオープンな仕様である．誰でも自由に，ライセンス料なしで，本仕様書に定義する API に準拠した TCP/IP プロトコルスタックを実装・利用・販売することができる．

ただし，Embedded TCP/IP 技術委員会ならびに ITRON 専門委員会は，本仕様に関して，その正当性や妥当性も含めて，いかなる保証も行わない．また，本仕様を利用したことに関連して直接的・間接的に発生した被害・損害に対する一切の責任は，これを負わない．

本仕様書そのものの著作権は, Embedded TCP/IP 技術委員会ならびに ITRON 専門委員会が保有している. Embedded TCP/IP 技術委員会ならびに ITRON 専門委員会は，本仕様書を自由に印刷・複製すること，実費程度のコストで配布することを認めるが，本仕様書を許可なく改変することは認めていない．本仕様書を改変して製品マニュアルを作成する場合には，Embedded TCP/IP 技術委員会ないしは ITRON 専門委員会の許可が必要である．

4.2 仕様に関する問い合わせ先

ITRON 仕様の作成／保守は，社団法人トロン協会 ITRON 専門委員会が行っている．仕様に関する問い合わせ先は次の通りである．

〒 108 東京都港区三田 1–3–39 勝田ビル 5F
(社) トロン協会 ITRON 専門委員会
TEL: 03–3454–3191
FAX: 03–3454–3224

ITRON 専門委員会では，ITRON プロジェクトに関連する各種の情報や ITRON 専門委員会の活動を広く知らせるために，ITRON ホームページを開設している．ITRON ホームページの URL は次の通りである．

http://tron.um.u-tokyo.ac.jp/TRON/ITRON/

4.3 仕様のステータス

本仕様書に定義する API に準拠した TCP/IP プロトコルスタックの実装はまだ行われていないため，実装経験を反映して改良を行うべきであると考えられる．また, API やニーモニックなどのコンベンションは，現在検討が進んでいる新しい ITRON カーネル仕様に合致させるべきである．

以上の理由から，新しい ITRON カーネル仕様の検討スケジュールと，本仕様に準拠した TCP/IP プロトコルスタックの実装スケジュールを勘案し，仕様の最初のバージョンのフィックスから約半年後をメドに，Embedded TCP/IP 技術委員会において仕様の見直しを行う予定である．

付録 A Embedded TCP/IP 技術委員会 参加者リスト

足立 和雄 ((株) ジール)
天野 亘孝 ((社) トロン協会)
大槻 弥 ((株) エーアイコーポレーション)
筏島 雅之 ((株) アクセス)
加藤 博之 ((株) エーアイコーポレーション)
鎌田 富久 ((株) アクセス)
神田 拓哉 (富士通デバイス (株))
幹事 : 黒田 隆 (神鋼電機 (株) ／個人参加)
小島 巧 ((有) データグラム)
幹事 : 児玉 剛 (アルパイン情報システム (株) ／個人参加)
小早川 学 ((株) 日立製作所)
小林 康浩 (富士通 (株))
佐藤 直樹 (日本電気 (株))
澤田 勉 (エルグ (株))
砂原 秀樹 (奈良先端科学技術大学院大学)
多賀 直久 ((株) グレープシステム)
幹事 : 高田 広章 (豊橋技術科学大学)
田丸 喜一郎 ((株) 東芝)
綱島 真一 ((株) アクセス)
冨田 恭夫 ((株) エーアイコーポレーション)
内藤 隆一 (日立ビジネスソリューション (株) ／個人参加)
中本 幸一 (日本電気 (株))
長谷川 雄三 ((株) エーアイコーポレーション)
舟越 淳 ((株) グレープシステム)
水口 和美 ((株) ジール)
水谷 亨 (三菱電機マイコン機器ソフトウエア (株))
宮崎 久則 ((有) 宮崎システム設計事務所)
門田 浩 (日本電気 (株))
山田 真二郎 ((株) 日立製作所)
若林 綱一 (東芝情報システム (株))

(あいうえお順)

付録 B 仕様改訂履歴

1998 年 5 月 14 日	Ver. 1.00.00	最初のバージョンを公開
1998 年 5 月 19 日	Ver. 1.00.01	タイプミスの修正など

付録C ソケットインタフェースとの関連

C.1 ソケットインタフェースとの主な違い

本仕様の API とソケットインタフェースとの主な違いは次の通り．

- ソケットインタフェースでは，できる限りプロトコル独立に API が定められているのに対して，本仕様の API はプロトコル毎に定義されている．具体的には，本仕様では，TCP の API と UDP の API を別々に定義している．また，アドレスを指定する構造体は，IPv4 上の TCP と UDP に特化して定義している．その他のプロトコルはサポートしていない．
- ソケットに相当するアブストラクションとして，通信端点を用いている．通信端点は，プロトコルや機能毎に別種類のものと定義されている．具体的には，TCP 用の通信端点と UDP 用の通信端点を区別することに加えて，TCP 用の通信端点としても，TCP 受付口と TCP 通信端点を区別して扱っている．通信端点を生成/削除する API も，通信端点の種類毎に用意している．
- ソケットインタフェースのファイルディスクリプタに相当するアブストラクションは用意せず，通信端点を直接 API から操作する（1.4.3 節参照）．また，ソケットインタフェースの select に相当する機能はサポートしていない（類似の機能はコールバックを用いて実現できる）．
- 待ち状態に入る可能性のある各 API に，タイムアウトとノンブロッキングコールを用意している．ソケットインタフェースでもノンブロッキングは用意されているが，各 API の単位ではなく，ソケットの単位でノンブロッキングを指定する点が異なる．
- ペンディングしている処理をキャンセルする機能（tcp_can_cep, udp_can_cep）を用意している．
- ソケットインタフェースの listen の機能は，TCP 受付口を生成する API である tcp_cre_rep があわせ持っており，listen に対応する API となっている．ただし，tcp_cre_rep は，ソケットインタフェースの listen が持つ接続要求の最大キューイング数を指定する機能をサポートしていない．
- ソケットインタフェースの bind の機能を他の API（tcp_cre_rep，tcp_con_cep，udp_cre_cep）があわせ持っており，bind に直接対応する API は用意していない．
- ソケットインタフェースの accept は，接続要求を受け付けた時に内部で動的にソケットを生成するのに対して，本仕様の API の tcp_acp_cep は，接続要求があった時に接続に用いる TCP 通信端点もパラメータとして渡す．
- TCP 用に省コピー API を用意している．
- TCP 接続の受信のみを終了する API は用意していない．具体的には，ソケットインタフェースでは，shutdown に対するパラメータで送受信のいずれを終了するかを指定可能であるが，本仕様の tcp_sht_cep はそれに相当するパラメータを持たない．

- TCP の緊急データを送受信する API は，通常のデータ送受信 API とは別に定義している．また，受信した緊急データは，コールバックルーチン内で取り出さないと捨てられるものとし，プロトコルスタック内でキューイングしない．
- UDP パケットの受信時に呼ばれるコールバックを用意している．また，受信した UDP パケットは，コールバックルーチン内で取り出さないと捨てられるものとし，プロトコルスタック内でキューイングしない．
- 接続された UDP ソケットに相当する機能はサポートしていない．

付録 D プログラム例

D.1 TCP の省コピー API を用いた read/write の実装例

```
ER read(ID cepid, VP buf, INT len, TMO tmout)
{
        VP rbuf;
        INT rlen;
        INT tlen = 0;       /* 受信したデータの合計長 */

        if ((rlen = tcp_rcv_buf(cepid, &rbuf, tmout)) <= 0) {
                return(rlen);
        }
        while (len > 0 && rlen > 0) {
                if (rlen > len) {
                        rlen = len;
                }
                bcopy(rbuf, buf, rlen);
                buf += rlen;
                len –= rlen;
                tlen += rlen;
                if (tcp_rel_buf(cepid, rlen) < 0) {
                        /* エラーが発生するまでに受信したデータ長を返す */
                        return(tlen);
                }
                if ((rlen = tcp_rcv_buf(cepid, &rbuf, TMO_POL)) < 0) {
                        /* エラーが発生するまでに受信したデータ長を返す */
                        return(tlen);
                }
        }
        return(tlen);
}

ER write(ID cepid, VP buf, INT len, TMO tmout)
{
        VP sbuf;
        INT slen;
        INT tlen = 0;       /* 送信したデータの合計長 */

        if ((slen = tcp_get_buf(cepid, &sbuf, tmout)) <= 0) {
                return(slen);
        }
        while (len > 0 && slen > 0) {
                if (slen > len) {
                        slen = len;
                }
                bcopy(buf, sbuf, slen);
                buf += slen;
                len -= slen;
```

```
                tlen += slen;
                if (tcp_snd_buf(cepid, slen) < 0) {
                        /* エラーが発生するまでに送信したデータ長を返す */
                        return(tlen);
                }
                if ((slen = tcp_get_buf(cepid, &sbuf, TMO_POL)) < 0) {
                        /* エラーが発生するまでに送信したデータ長を返す */
                        return(tlen);
                }
        }
        return(tlen);
}
```

D.2 TCP の省コピー API を用いた getc/putc の実装例

※ 送受信対象を 1 つの TCP 通信端点に固定している.
※ API が返すエラーの処理は省いている.

```
extern ID cepid;

static unsigned char *rcvbuf;
static INT rcvbuflen = 0;
static INT rcvdatlen = 0;

int getc()
{
        if (rcvbuflen == 0) {
                if (rcvdatlen > 0) {
                        tcp_rel_buf(cepid, rcvdatlen);
                        rcvdatlen = 0;
                }
                rcvbuflen = tcp_rcv_buf(cepid, &rcvbuf, TMO_FEVR);
                if (rcvbuflen == 0) {
                        /* データ終結（EOF）*/
                        return(-1);
                }
        }
        rcvbuflen -= 1;
        rcvdatlen += 1;
        return(*rcvbuf++);
}

static unsigned char *sndbuf;
static INT sndbuflen = 0;
static INT snddatlen = 0;

void putc(char c)
{
        if (sndbuflen == 0) {
```

```
                        if (snddatlen > 0) {
                                tcp_snd_buf(cepid, snddatlen);
                                snddatlen = 0;
                        }
                        sndbuflen = tcp_get_buf(cepid, &sndbuf, TMO_FEVR);
                }
                sndbuflen -= 1;
                snddatlen += 1;
                *sndbuf++ = c;
}

void flush()
{
        if (snddatlen > 0) {
                tcp_snd_buf(cepid, snddatlen);
                snddatlen = 0;
                sndbuflen = 0;
        }
        if (rcvdatlen > 0) {
                tcp_rel_buf(cepid, rcvdatlen);
                rcvdatlen = 0;
                rcvbuflen = 0;
        }
}
```