ITRON TCP/IP API
仕様の概要

Embedded TCP/IP 技術委員会

なぜITRONでTCP/IPなのか

- インターネットにつながる組み込みシステム
  - ＰＤＡ / インターネット・テレビ / イントラネット用端末 / カーナビゲーションシステム / 携帯電話 …
- パソコン、ワークステーションとの接続
  - コピー機、ＦＡＸをブラウザから設定する、など
  - 工作機械のユーザーインターフェースをＰＣで持たせる（ブラウザなど）

組み込みシステムへの適応

- 組み込みシステムにおける要求項目
  - プログラムが小さい（プログラムサイズ・データサイズ）
  - ＣＰＵの性能をフルに活用
  - 信頼性の確保
  - 動的なメモリ管理をもたない
  - ファイルシステムをもたない

ソケットインターフェースの問題点

- ソケット
  - 動的なメモリ管理を前提としている
  - 設定などの情報をファイルシステムに保存することを前提としている
  - システムによって差異があり、標準と呼べるものが存在しない
- 互換性
  - 各社とも独自のインターフェースを持っている
  - １つのアプリケーションを複数のプラットフォーム上に実装する場合、実装モデルを変更する必要が生ずることもある

仕様作成にあたって

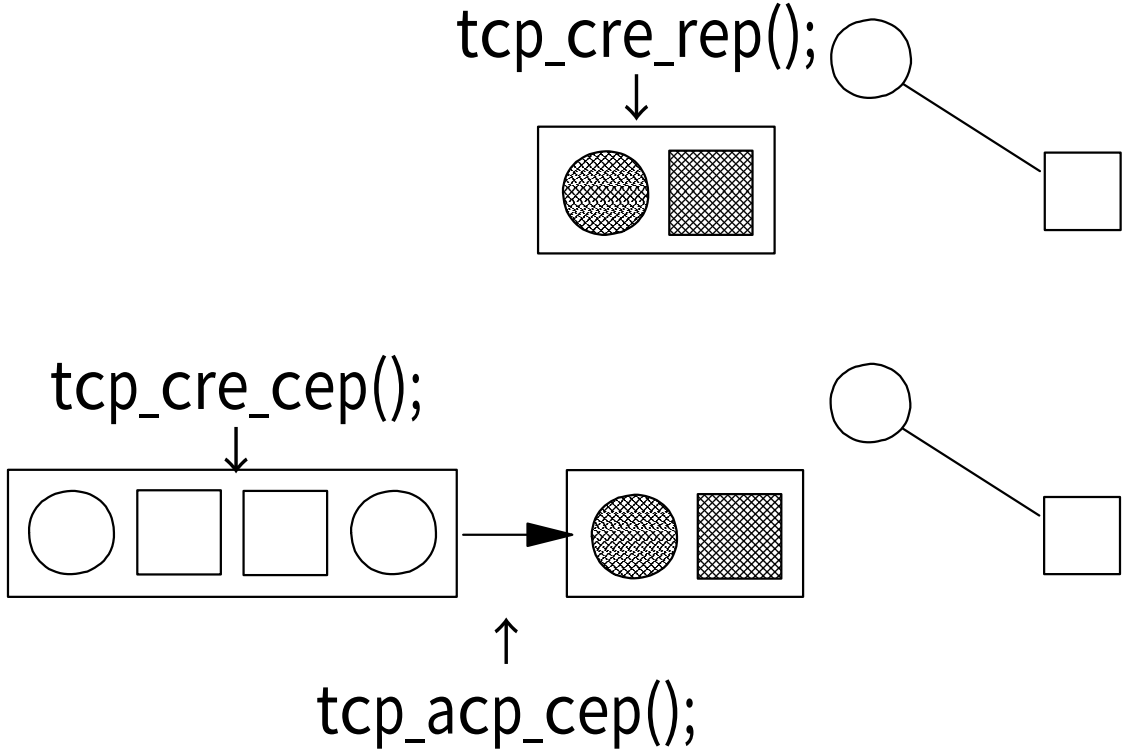
- Embedded TCP/IP 技術委員会
  - ITRON専門委員会の呼び掛けによって1997年3月に発足
  - 成果をITRON専門委員会が承認 → ITRON TCP/IP API 仕様
- 方針
  - ソケットインターフェースから大きくははずさない
  - 本APIにライブラリを被せることでソケットインターフェースを実現できるものにする
  - ITRON上での利用を想定するが、他のＲＴＯＳにも載るよう配慮する
  - 静的な設定を活用する

ITRON TCP/IP API の特徴

- TCPと、UDPのAPIを別々に定義
- 「ソケット」に代えて「端点(end point)」を採用
  - TCP接続要求を受け付ける端点と、TCP接続の端点とは別のオブジェクト。UDP通信端点も別として扱う
  - ITRON カーネル仕様と同様、明示的に生成・削除を行う
  - オブジェクト種類毎にシステム内で一意なID番号で識別
- コールバックによる非同期インターフェースを用意
  - UDPの受信はコールバック関数内で行う
- TCP用の省コピーAPIを用意

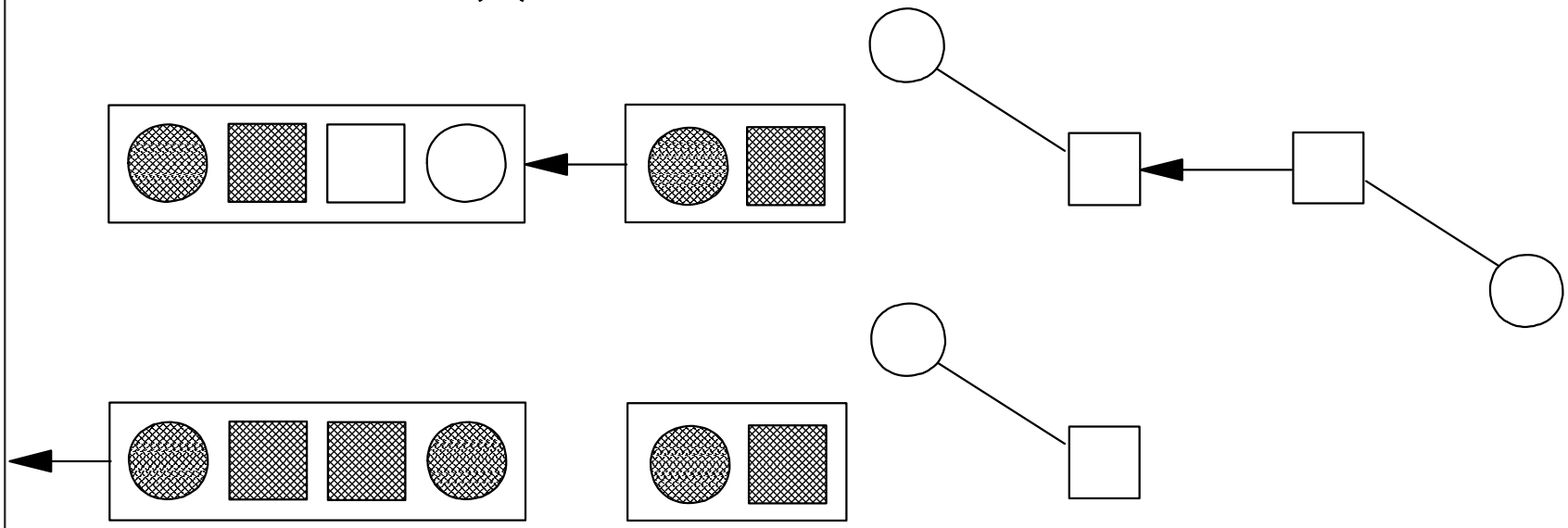
ITRON TCP/IP のモデル(１)

- 待ち受け手順(1)



ITRON TCP/IP のモデル(2)

- 待ち受け手順(2)



↓

tcp_del_rep();

ITRON TCP/IP のモデル(3)

- 待ち受け手順(3) ～ ソケットとの比較
  - ソケットの場合
    - 接続要求が到着する度に、ソケットが１つ新設される。
    - 待ち受け用ソケットは、接続が確立しない。
  - ITRON TCP/IP の場合
    - CEP, REPは、明示的に生成される。
    - 接続要求が到着しても、待ち受けに入っているCEPがなければ要求は捨てられる。

TCPのAPI(1)

- tcp_cre_cep：TCP通信端点の生成
  - ER tcp_cre_cep(ID cepid, T_TCP_CCEP *pk_ccep);
  - 指定したIDのTCP通信端点を生成する。
  - パラメータ:
    - 生成するTCP通信端点ID
    - 送信用・受信用、各ウィンドウバッファ
      - 送信・受信で個別に指定する
      - 実装によって、サイズのみの指定も許す
    - コールバックルーチンのアドレス
  - 補足
    - 静的生成が標準でサポートされている。

TCPのAPI(2)

- tcp_cre_rep：TCP受付口の生成
  - ER tcp_cre_rep(ID repid, T_TCP_CREP *pk_crep);
  - 指定したIDのTCP通信端点を生成する。
  - パラメータ:
    - 生成するTCP受付口ID
    - 自分側のIPアドレス
    - 自分側のポート番号
  - 補足
    - 一つのシステムで複数のIPアドレスを持つ場合を考慮に入れてある。
    - IPV4_ADDRANY(=0)で、すべてのアドレスを指定可能。

TCPのAPI(3)

- tcp_con_cep：接続要求（能動オープン）
  - ER tcp_con_cep(ID cepid,T_IPV4EP *p_myaddr, T_IPV4EP *p_dstaddr, TMO tmout);
  - 指定したIDのCEPを用いて、指定した相手IPアドレス/ポート番号に対して接続を行う。
  - パラメータ:
    - TCP通信端点 ID
    - 自分側のポート番号 (自動割付あり)
    - 相手側のIPアドレスとポート番号
    - タイムアウト指定 (TMO_POL, TMO_FEVR, TMO_NBLK)
  - 自分側のアドレスの自動割りつけは３通りで行う。

TCPのAPI(4)

- tcp_acp_cep：接続要求待ち(受動オープン)
  - ER tcp_acp_cep(ID cepid, ID repid, T_IPV4EP *p_dstaddr,

    TMO tmout);
  - 指定したTCP受付口に対する接続要求を待つ。
  - パラメータ:
    - 接続要求を待つTCP受付口ID
    - 接続に使うTCP通信端点ID
    - タイムアウト指定 (TMO_POL, TMO_FEVR, TMO_NBLK)
  - 補足
    - 一つのTCP受付口に対して、同時に複数のtcp_acp_cepを発行することが可能。

TCPのAPI(5)

- tcp_snd_dat：データの送信
  - ER tcp_snd_dat(ID cepid, VP data, INT len, TMO tmout);
- tcp_rcv_cep：データの受信
  - ER tcp_rcv_cep(ID cepid, VP data, INT len, TMO tmout);
- パラメータ:
  - コネクションポートID
  - バッファの先頭と長さ
  - タイムアウト指定 (TMO_POL, TMO_FEVR, TMO_NBLK

TCPのAPI(6)

- 省コピー送受信
  - ER tcp_get_buf(ID cepid, VP *p_buf, TMO tmout);
    - 次に送信するデータを入れるべきバッファを取り出す。
  - tcp_snd_buf(ID cepid, INT len);
    - tcp_get_buf で取り出したバッファに書き込んだデータの送信を手配する。
  - tcp_rcv_buf(ID cepid, VP *p_buf, TMO tmout);
    - 受信したデータが入っているエリアを取り出す。
  - tcp_rel_buf(ID cepid, INT len);
    - tcp_rcv_buf で取り出したバッファ中のデータを捨てる。

TCPのAPI(7)

- コールバック関数
  - ER callback(ID cepid, FN fncd, VP p_parblk);
    - パラメータ
      - TCP通信端点ID
      - 事象の種類
      - 事象の種類に固有なパラメータブロックへのアドレス
    - ノンブロッキングコールの完了通知
      - fncd = 完了した API の機能コード(例：TFN_TCP_SND_DAT)
      - p_parablk -> ercd
    - 緊急データの受信
      - fncd = TEV_TCP_RCV_OOB (= 0x0201)
      - p_parablk -> ercd

TCPのAPI(8)

- エラーコード
  - エラーは負の値。０または正の値で正常終了
  - メインエラーコードとサブエラーコードとで構成される。
    - ER mainercd(ercd);
    - ER subercd(ercd);
    - ITRONカーネルと同一のエラーコード
      - E_OK　　= 0
      - E_OBJ　　= -63
    - 追加したエラーコード
      - E_WBLK (= -83)　　ノンブロッキングコール受付
      - E_CLS (= -87)　　接続が異常切断された
      - E_BOVR (= -89)　　バッファオーバーフロー

使用例(1)

- クライアントの場合

```
void main(void)
{
    ercd = tcp_cre_cep(cepid,pk_ccep);
    while (stat) {
        ercd = tcp_con_cep(cepid,p_myaddr,p_dstaddr,tmoout);
        while (stat) {
            ercd = tcp_snd_cep(cepid, data, len, TMO_FEVR);
            ercd = tcp_rcv_cep(cepid, data, len, TMO_NBLK);
        }
        ercd = tcp_cls_cep(cepid, TMO_FEVR);
    }
    ercd = tcp_del_cep(cepid);
}
```

使用例(2)

- シングルタスク・サーバーの場合

```
void main(void)
{
    ercd = tcp_cre_cep(cepid,pk_ccep);
    ercd = tcp_cre_rep(repid,pk_crep);
    while (true) {
        ercd = tcp_acp_cep(cepid,repid,dstaddr,tmout);
        while (true) {
            ercd = tcp_snd_cep(cepid, data, len, TMO_FEVR);
            ercd = tcp_rcv_cep(cepid, data, len, TMO_NBLK);
        }
        ercd = tcp_cls_cep(cepid, TMO_FEVR);
    }
}
```

使用例(3)

- マルチタスク・サーバーの場合(1)
  - まず、サーバータスクを用意する。

```
void server(ID repid)
{
    ercd = tcp_cre_cep(cepid,pk_ccep);
    while (true) {
        ercd = tcp_acp_cep(cepid,repid,dstaddr,tmout);
        while (true) {
            ercd = tcp_snd_cep(cepid, data, len, TMO_FEVR);
            ercd = tcp_rcv_cep(cepid, data, len, TMO_NBLK);
        }
        ercd = tcp_cls_cep(cepid, TMO_FEVR);
    }
}
```

使用例(4)

- マルチタスク・サーバーの場合(2)
  - リッスンポートを用意し、サーバータスクを起動する。
  - 静的設定を利用する場合は生成不要。

```
void main(void)
{
    int i, start_code;

    ercd = tcp_cre_rep(repid,pk_crep);
    start_code = repid;
    for (i=0; i<SERVER_NUM; i++) {
        create & start_task(server, start_code);
    }
    sleep_task(SELF);
}
```

使用例(5)

- マルチタスク・サーバーの場合(3)