TOPPERS基礎3実装セミナー
（LPC2388：基本1）
プラットフォーム編：2日目

TOPPERSプロジェクト
教育ワーキング・グループ

TOPPERS

本ドキュメントに関して

1. 著作権に関しての表記

2. 本ドキュメントに関するご意見・ご提言・ご感想・ご質問等がありましたら、TOPPERSプロジェクト事務局までE-Mailにてご連絡ください。
3. 本ドキュメントの内容は、内容の改善や適正化の目的で予告無く改定することがあります。

TOPPERS

スケジュール

- 2日目

1.	ITRON-TCP/IP仕様	1.0時間
2.	TINETデバイスドライバの設計	0.5時間
3.	DHCP+ECHOサーバの実装確認	1.5時間
4.	プラットフォーム作成	0.5時間
5.	仮想端末アプリの構築	0.5時間
6.	仮想端末アプリの拡張	1.5時間
7.	宿題：プラットフォームOSの変更	
8.	まとめ	0.2時間

ITRON-TCP/IP仕様

1. TCP/IPの基礎
2. ITRON-TCP/IP仕様
3. TINET（TCP/IPプロトコルスタック）
4. TINETデバイスドライバ

ネットワークプロトコル

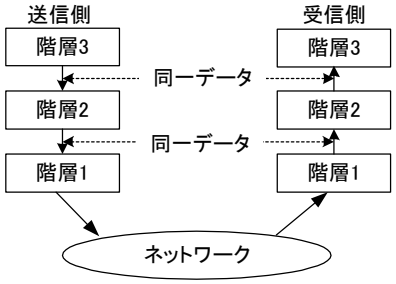
- プロトコル
  - 通信のための規約や手順
- TCP/IP
  - Internetの標準プロトコル群
  - IP
    - 信頼性のないコネクションレスパケット配送プロトコル
  - TCP
    - 信頼性のあるストリーム配送プロトコル
  - 4.2BSDに実装され、急速に普及
  - RFC(Request for Comments) 文書で規定
    - 実装して、実績のあるものだけ残す

階層化の必要性

- 巨大な一つのプロトコルを全て制御することは困難
  - ➡「層」の中で機能を限定して規定し、「層」を積み上げ、全体として通信が上手くいくように制御する

階層化原理

- 受信側の層*n*では、送信側の層*n*によって送られたものとまったく同じものを受け取る

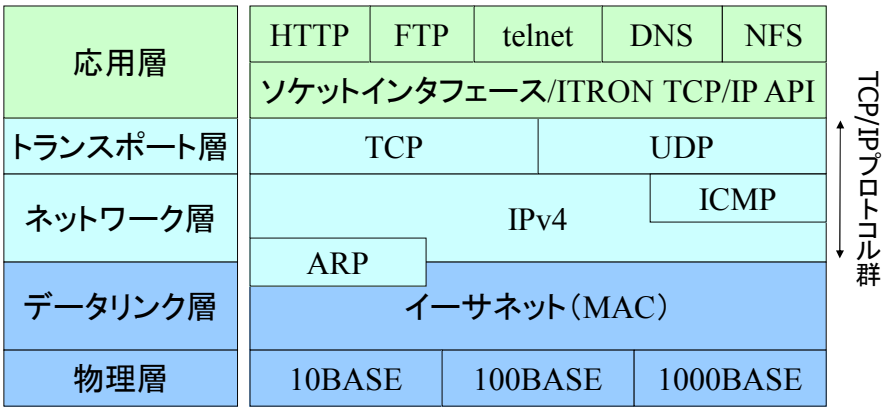


TCP/IP インターネットのプロトコル階層は5階層

- １～４層を重点的に説明を行う

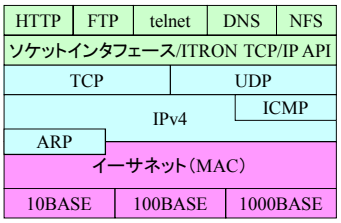
	名称	機能
5	アプリケーション（応用）層	各種のアプリケーションを制御する
4	トランスポート（TCP,UDP）層	通信主体のプロセス間での通信の制御および信頼性の確保
3	ネットワーク（IP)層	経路制御によるノード間（コンピュータ間）の通信の制御
2	データリンク層	隣接するノード間の通信を制御
1	ハードウェア層	ハードウェアの制御

階層図　（TCP/IPv4, イーサネット）



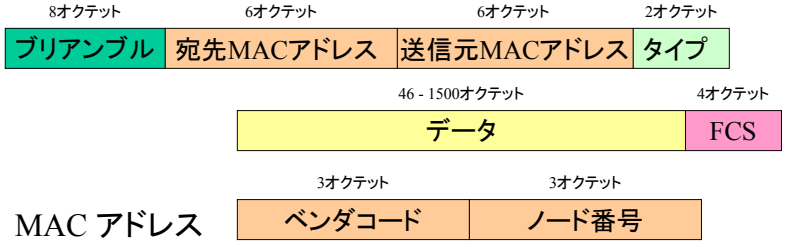
データリンク層・物理層：イーサネット

- パケット交換型
  - データをパケット単位で送信
- バス型からスター型
- 帯域共有HubからスイッチングHub
- 最善努力型（ベストエフォート）
  - 破棄されることもある
  - 帯域共有型では衝突もある
- 10Mbpsから1Gbps（10Gbps）
- インタフェースには48ビットのイーサネットアドレス（MACアドレス）が割り当てられている



イーサネットフレーム

- 可変長で64オクテット以上、1518オクテット以下
- フレームを受信すると、タイプを見て処理するプロトコルスタックを選択



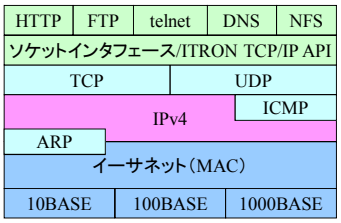
タイプ

IPv4	0x0800
IPv6	0x86dd
ARP	0x0806

オクテット（octet）
1オクテットは8ビットに相当
バイトは何ビットかという明確な定義はないため，明確に8ビットを表す言葉として用いられている．通信の分野で用いられる．

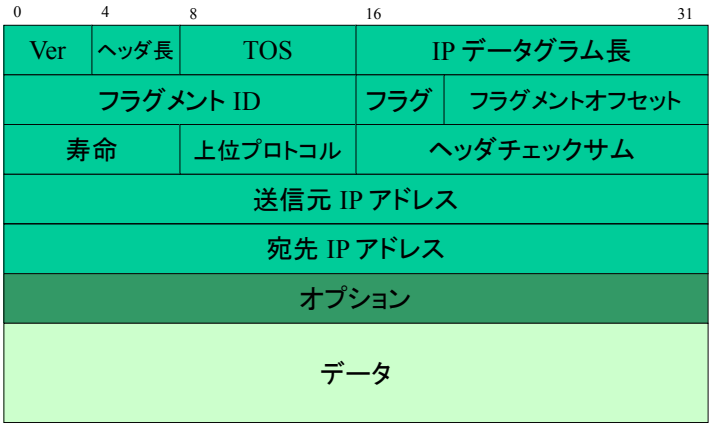
ネットワーク層：IP（Internet Protocol）

- 信頼性のないコネクションレスパケット配信サービス
- 経路選択と中継制御
  - 経路選択表
- データグラムの分割と再構成
- 最善努力型（ベストエフォート）
  - パケットは破棄されることもある
- アドレスは 32 ビット
  - アドレスの枯渇問題



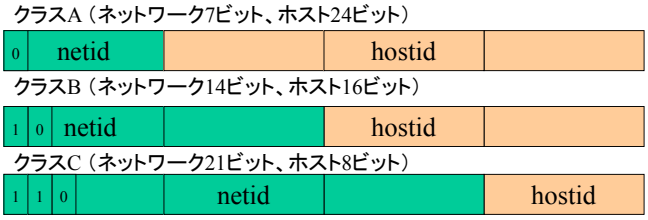
IPv4 データグラム

- データの基本転送単位
- ヘッダとデータ領域で構成されている



IPv4 のクラス型アドレス

- 各ホストには、32ビットのアドレスが割り当てられる



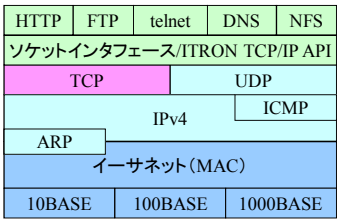
- クラス分けが固定的でクラスBが不足
- アドレスの使用効率の低下
  - クラスCでは足りないが、クラスBでは多い
- トポロジー（地理）と無関係
  - 経路情報が増える
- 最大でも約40億個まで

応用層プログラムの識別：ポート番号

- IPはコンピュータ間の通信内容を定める
- 接続相手の応用層のアプリケーションを識別する番号
  - プロセス番号は、プロセスを起動すると割当てられ、常に同一の番号とは限らない
  - プロセス番号とは別の番号が必要
- Well Known Port（サーバー用）
  - 0から1,023
  - よく使われるサーバーの応用プログラムのポート番号
    - HTTP（80）、FTP（20と21）、SMTP（25）、DNS（53）
  - 定義ファイル：c:¥windows¥system32¥drivers¥etc¥services
- 一般ユーザ用のWell Known Port
  - 1,024から49,151
- 自動設定及び自由に使える範囲
  - 49,152から65,535

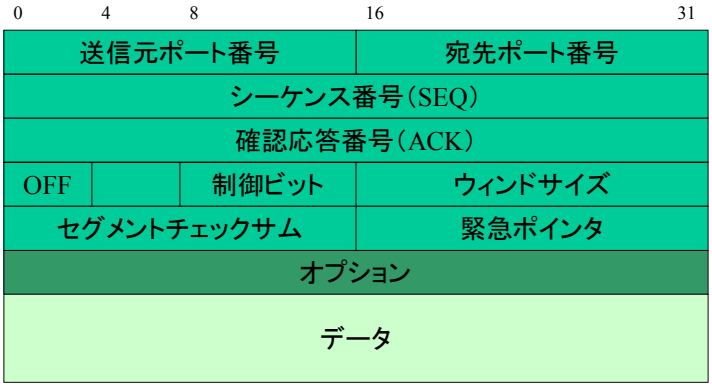
トランスポート層：TCP（Transmission Control Protocol）

- 信頼性のあるコネクション型プロトコル
- ストリーム指向
  - 送受信データをストリームとする
- 全二重通信
- 誤り制御（再送制御）
- 順序付け制御
- フロー制御と輻輳制御
- エンドポイント（通信相手の識別）
  - IPアドレスとポート番号の組
- コネクション（通信路）
  - エンドポイントの組で識別する



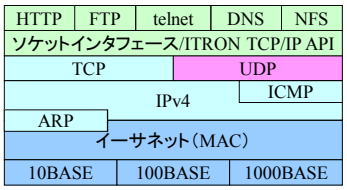
TCP セグメント

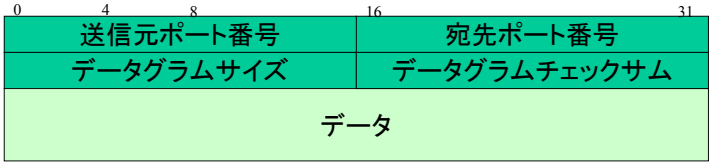
- シーケンス番号　：送信バイトストリーム中における位置
- 確認応答番号　：次に受信すると期待しているオクテットの番号



トランスポート層 : UDP（User Datagram Protocol）

- 信頼性のないコネクションレス型プロトコル
- IPとの違い
  - コンピュータ間ではなく、ポート番号を持つ応用プログラム間の通信
- オーバヘッドが小さく高速
- 応用層の例
  - DNS（512オクテットまでは UDP）
  - NFS（ファイルシステムが自分で信頼性確保）
  - VoIP 等のストリームデータ
- UDPデータグラム





TOPPERS

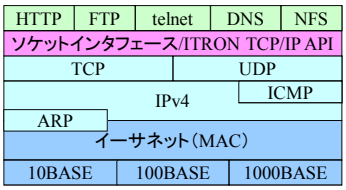
ネットワーク層 の拡張: IPv6

- IPv4のアドレス枯渇問題に対応
- IP アドレスは 128 ビット
  - 16E×16E 個または4G×4G×4G×4G個
- IP ヘッダの簡略化
  - 固定長（40オクテット）
  - ルータでの転送の高速化（ハードウェア処理）
  - 拡張ヘッダの導入
- アドレス割当ての自動化
- 階層化されたアドレス割当て
- 通信品質管理とセキュリティ

TOPPERS

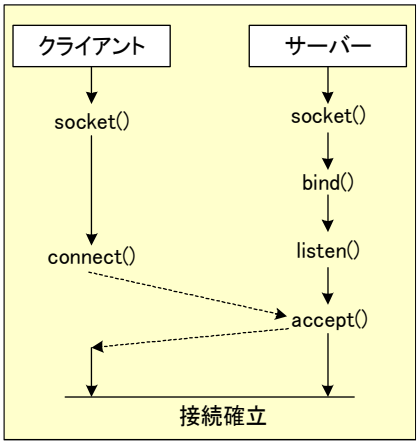
ソケットインタフェース

- BSD UNIXにおける応用アプリケーションプログラムとTCP/IPプロトコル間のインタフェース．多くのOSで採用されている。
  - WindowsではWinSock
- ソケット
  - 通信のための端点
  - 特定の終点アドレスにバインドすることなく生成可能



ソケットインターフェース : API

- ソケットの生成
  - socket(pf, type, protocol)
- アドレスのバインド
  - bind(socket, localaddr, addrlen)
- ソケット受動モードに
  - listen(socket, qlength)
- コネクションの受付
  - accept(socket, addr, addrlen)
- サーバーへの接続
  - connect(socket, destaddr, addrlen)
- データ受信
  - readv(descriptor, buffer, length)
- データ送信
  - write(socket, buffer, length)
- 終了
  - close(socket)



ITRON-TCP/IP仕様

1. TCP/IPの基礎
2. ITRON-TCP/IP仕様
3. TINET（TCP/IPプロトコルスタック）
4. TINETデバイスドライバ

組込みシステムとTCP/IP

- 組込みシステムのインターネット接続
  - PDA、プリンタ、DVDレコーダー、プロジェクター
- インターネットプロトコルの流用
  - ！インターネットプロトコルである必然性はなく、他に適切なプロコルがあればそれでもよい

➡ パソコン側で既存のソフトウェアが利用可能

例）Webブラウザ

万能ユーザーインタフェースツール

➡ パソコン側のユーザーインタフェースツールを作成する必要がない

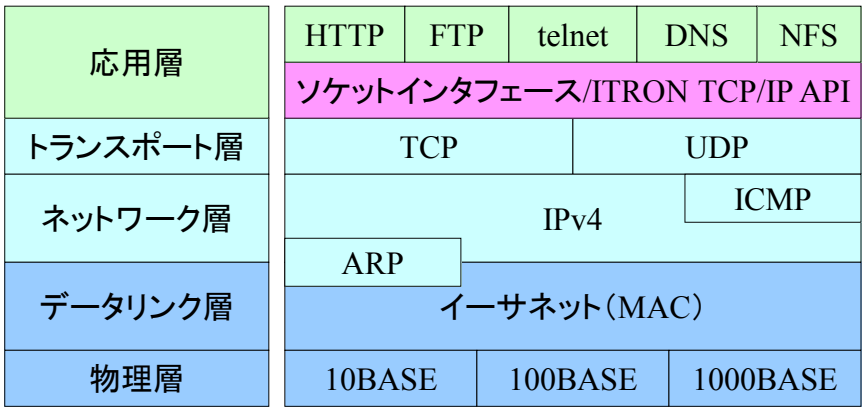
TCP/IPの重要性はさらに高まる傾向

組込みシステムの特性

- 多くの（特に小規模な）組込みシステムが持つ特性
  - 行うべき処理が決まっている
  - ハードウェア資源に対する制約が厳しい
  - リアルタイム性が求められる
  - 高い信頼性が求められる
- 動的メモリ管理 vs. 静的メモリ管理
  - 組込みシステムでは、メモリをなるべく静的に管理したい（管理できる場合が多い）
  - やむをえず動的に管理する場合は、メモリ管理のポリシーをアプリケーションに任せる形が望ましい

TCP/IPプロトコルスタックのAPI

- TCPとUDPのAPIが最も重要
  - ➡ 以下、これに絞って議論



ソケットインタフェースとその問題点

- 現在、TCPとUDPのAPIとしては、BSD UNIX用に設計されたソケットインタフェース（またはその変形）が広く使われている
- 組込みシステム（特に小規模なもの）には不向きとの指摘
  - プロトコル非依存の汎用インタフェース
    例）通信相手を指定する時のアドレス形式
  - プロトコルスタック内で動的なメモリ管理が必須
    例）UDPパケットの受信
  - データのコピー回数が多くなる
    read/write
  - RTOSのタスクモデルとUNIXのプロセスモデルの違い
    fork/select/シグナルハンドラ

Embedded TCP/IP技術委員会

- 各社が独自に、ソケットインタフェースに代わる独自のAPIを用意。アプリケーションの互換性に問題

標準化の必要性

Embedded TCP/IP技術委員会

- ITRONプロジェクトにおけるソフトウェア部品APIの標準化活動の第一弾
- 1998年に標準化の成果を ITRON TCP/IP API仕様（Ver 1.0）として公開

ITRON TCP/IP APIの設計仕様

(a) ソケットインタフェースをベースに

- ソケットに慣れているプログラマが多い
- ソケットで書かれたソフトウェア資産を活かしたい
- 上にライブラリを載せてソケット互換に

(b) APIの理解しやすさ、プログラムしやすさを重視

- 複雑な/使いにくいAPIはなるべく避ける
- サービスコールのエラーの詳細をわかるように

(c) ハードウェア資源(プロセッサの能力、メモリ容量)を有効に活用できことを重視

- メモリ不足時の振舞いをアプリケーションが制御できる
- プロトコルスタック内部での動的メモリ管理の必要性を最小限に
- データのコピー回数を減らせる省コピーAPIを用意する

ITRON TCP/IP APIの設計仕様

(d) プロトコル毎にそれに最適なAPIを定義

- TCPとUDPのAPIを別々に定義
- アプリケーション開発者が、必要な資源量を把握しやすくなる効果も

(e) リアルタイムシステムへの適用を考慮

- 待ち入るサービスコールには、一律タイムアウトとノンブロッキングコールを用意

(f) 静的な設定で十分なものを考慮

- 「静的API」
  システム構成ファイルを書くと静的に設定
  例)特定のポート番号でTCP待ちを行う

(g) ITRON仕様の作法を踏襲、他のRTOSにも適応可能

- サービスコール、パラメータ、エラーの名称等はITRON仕様の作法に準拠

ITRON TCP/IP の主概念

- 通信端点
  - ネットワークを介した通信サービスの端点（ソケットに対応）
  - プロトコルの種類ごとに定義
    - UDPの通信端点（UDP communication end point）
    - TCPの受付口（TCP reception point）
    - TCPの通信端点（TCP communication end point）

      ソケットインタフェースでは，TCP接続を待ち受けているソケットは、データの送受信には使われない
  - 種類ごとにシステム全体でユニークなID番号で識別

← (e) (g)の方針

TOPPERS

ITRON TCP/IP の主概念

- ノンブロッキングコール
  - サービスコールの中でブロックされる状況になった場合に、処理を継続したままサービスコールからリターン

    UNIXの非同期I/Oとの違い
  - 処理が完了した時点でコールバックを用いて通知
  - 処理を途中でキャンセルすることも可能
- コールバック
  - プロトコルスタックからのイベントをアプリケーションに通知する（OS非依存な）方法
    - 通信端点毎にアプリケーションで定義
    - プロトコルスタック側のコンテキストで実行
      - メモリ保護のないOSを想定

中でイベントフラグをセットするという使い方も可能

TOPPERS

ITRON TCP/IP の主概念

- サービスコールの返値とエラーコード
  - サービスコールの返値 ← (g)の方針
    - 正または0 ・・・ 正常終了
    - 負 ・・・ エラーコード
  - 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
    - メインエラーコード ・・・ 仕様で標準化
    - サブエラーコード ・・・ 実装依存、デバッグで使うことを想定 ← (d)の方針
- 静的API ← (f)の方針
  - システム構成ファイル中に記述することで、通信端点を初期化時に静的に生成するための記述方法

TCPのサービスコールの一覧

- 通信端点の生成/削除
  - TCP_CRE_REP　TCPの受付口の生成(静的API)　標準
  - tcp_cre_rep　TCP受付口の生成　拡張
  - tcp_del_rep　TCP受付口の削除　拡張
  - TCP_CRE_CEP　TCP通信端点の生成(静的API)　標準
  - tcp_cre_cep　TCP通信端点の生成　拡張
  - tcp_del_cep　TCP通信端点の削除　拡張
- 接続/切断
  - tcp_acp_cep　接続要求待ち(受動オープン)　標準
  - tcp_con_cep　接続要求(能動オープン)　標準
  - tcp_sht_cep　データ送信の終了　標準
  - tcp_cls_cep　TCP通信端点のクローズ　標準
- データの送受信
  - tcp_snd_dat　データの送信　標準
  - tcp_rcv_dat　データの受信　標準

TCPのサービスコールの一覧

- データ送受信（省コピーAPI）
  - tcp_get_buf　送信用バッファの取得　標準
  - tcp_snd_buf　バッファ内のデータの送信　標準
  - tcp_rcv_buf　受信バッファの取得　標準
  - tcp_rel_buf　受信用バッファの開放　標準
- 緊急データの送受信・その他のサービスコール
  - tcp_snd_oob　緊急データの送信　拡張
  - tcp_rcv_oob　緊急データの受信　拡張
  - tcp_can_cep　ペンディング処理のキャンセル　標準
  - tcp_set_opt　TCP通信端点オプションの設定　拡張
  - tcp_get_opt　TCP通信端点オプションの読出し　拡張
- コールバック
  - ノンブロッキングコールの終了　標準
  - 緊急データの受信　拡張

クライアント側の接続手順

- ソケットインターフェースとの対比

  ソケット：socket ⇒ (bind) ⇒ connect ⇒
  端点の生成 ⇒（自ノードの ⇒ 接続 ⇒
  アドレス設定）

  ITRON：tcp_cre_cep ⇒ tcp_con_cep ⇒
- プログラム例

```
/* TCP通信端点の生成（システム構成ファイルに記述する）*/
TCP_CRE_CEP(CEPID, {0, NADR, SBUFSZ, NADR, RBUFSZ,callback});
```

```
/* 接続を確立する */
dstaddr.ipaddr  = REMOTE_IPADDR;
dstaddr.portno  = REMOTE_PORTNO;
if (( ercd = tcp_con_cep(CEPID, NADR, &dstaddr, TMO_FEVR)) != E_OK){
                /* エラー処理 */
}
```

TCP_CRE_CEP　TCP通信端点の生成　【標準】
tcp_cre_cep　【拡張】

【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

内容 : TCP通信端点属性
送信用ウィンドウバッファの先頭番地とサイズ
受信用ウィンドウバッファの先頭番地とサイズ
コールバックルーチンのアドレス

tcp_con_cep　接続要求（能動オープン）　【標準】

【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
TMP	tmout	タイムアウト指定T_TCP_CCEP

【特記事項】

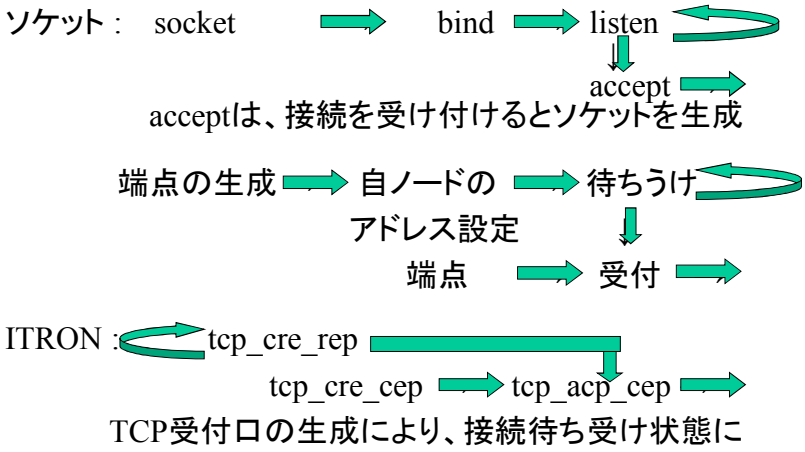
myaddrをNADRとした場合、自分側のIPアドレスとポート番号はプロトコルスタック内部で決定する（ソケットインタフェースでbindを呼ばない場合に相当）。片方だけ与えることも可能

【構造体】

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

サーバー側の接続手順

- ソケットインターフェースとの対比



TOPPERS

サーバー側の接続手順

プログラム例

```
/* TCP受付口とTCP通信端点の生成（システム構成ファイルに記述する）*/
TCP_CRE_REP(REPID, {0, {IPV4_ADDRANY, LOCAL_PRTNO}});
TCP_CRE_CEP(CEPID, {0, NADR, SBUFSZ, NADR, RBUFSZ, callback});
```

```
/* 接続を待ち受ける */
if ((ercd = tcp_acp_cep(CEPID, REPID, &dsaddr, TMO_FEVR) < 0){
      /* エラー処理 */
}
```

マルチタスクサーバー

- マルチタスクサーバーにする場合には、全く同一のプログラムを、通信端点ID（cepid）を変えて複数のタスクで動作させればよい
  - TCP受付口はすべてのタスクで共有している
  - TCP受付口に、接続待ち受けキューができる

TOPPERS

TCP_CRE_REP　TCP受付口の生成　【標準】
tcp_cre_rep　【拡張】

【静的API】

```
TCP_CRE_REP(ID repid, {ATR repatr,
                       {UW 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受付口情報

内容：TCP受付口属性
自分側のIPアドレスとポート番号

【特記事項】

- 自分側のIPアドレスの自動決定指定あり

tcp_acp_cep　接続要求（受動オープン）　【標準】

【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ID
TMP	tmout	タイムアウト指定T_TCP_CCEP

【リターンパラメータ】

T_IPV4EP	p_dstaddr	相手側のIPアドレスとポート番号

【特記事項】

- dstaddrには接続を要求した相手のIPアドレスとポート番号が返される
- 複数の tcp_acp_cep のどれに受け付けるかは実装依存

データの送受信

- 標準のAPI
  - ソケットインタフェースの write/read とほぼ同じ
    - 厳密には、非同期I/Oモードの write/read と同じ
- 省コピーAPI
  - データのコピー回数を減らせる可能性のある効率のよいAPI
  - プロトコルスタックが管理するバッファ（ウィンドバッファ）に直接アクセスする
  - 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は用意していない
    - アプリケーションプログラムの複雑化
    - データ長が短い場合は効率が上がらない

tcp_snd_dat データの送信 【標準】

【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

【特記事項】

- 送信バッファに1バイトでも入ればリターンする
- 戻り値が送信データの長さより短い場合は，送信されていないデータが存在する
- 複数の送信要求がペンディングする場合はエラー

tcp_rcv_dat　データの受信　【標準】

【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　受信バッファに入れたデータの長さ/エラーコード

【特記事項】

- 複数の受信要求がペンディングする場合はエラー
- TCP接続異常切断後も、バッファ内のデータは取り出せる
- 通信相手から切断されると戻り値が0となる

切断手順

ソケットインタフェースとの違い

- close
  - ファイルディスクリプタとソケットの対応を切り離す。ソケットは切断手順を開始する。切断が完了するまでソケットは残る。
- tcp_cls_cep
  - 通信端点の切断を行う。通信端点が未使用状態になるまでブロックする。
- 実装方法
  - 切断手順が完了するまでブロックするようにする
  - 送信が完了するまでブロックし、後は切断完了まで別扱いとし、通信端点は開放する。

tcp_cls_cep　通信端点のクローズ　【標準】

【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

【特記事項】

- 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持ってFINを送り、接続を切断する
- 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つ
- タイムアウトエラーになると、RSTを送って強制切断する。この場合でも、元の状態に戻らない（例外的なサービスコール）

tcp_sht_cep　データ送信の終了（ハーフクローズ）　【標準】

【C言語API】

ER ercd = tcp_sht_cep(ID cepid);

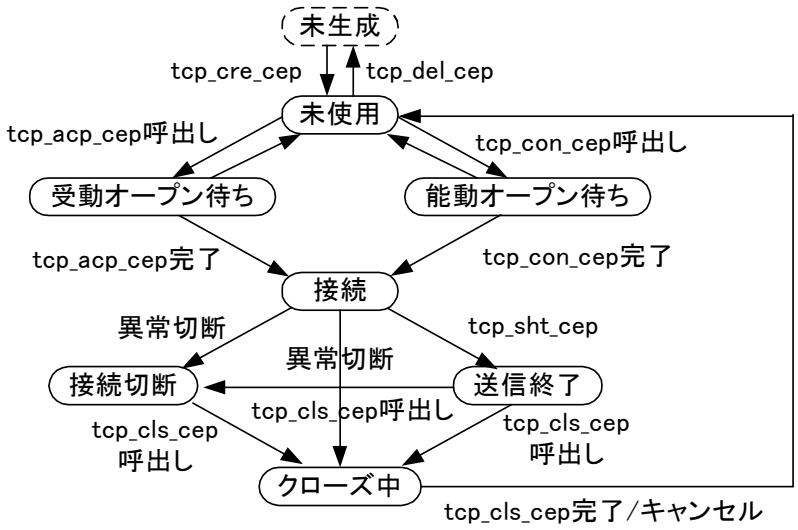
【パラメータ】

ID	cepid	TCP通信端点ID

【特記事項】

- 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、接続の切断手順を開始する。
- 実際には接続手順を解する手配をするだけなので、このサービスコール内でブロックすることはない
- TCP通信端点が未使用状態になるのを待つ
- tcp_sht_cep を呼び出した後はデータ送信ができない
- 受信側の shutdown は用意していない

TCP通信端点の状態遷移



TOPPERS

UDPのサービスコールの一覧

- 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
  - ucp_cre_cep　UDP通信端点の生成　拡張
  - ucp_del_cep　UDP通信端点の削除　拡張
- パケットの送受信
  - udp_snd_dat　UDPパケットの送信　標準
  - upd_rcv_dat　UDPパケットの受信　標準
- その他のサービスコール
  - udp_can_cep　ペンディング処理のキャンセル　標準
  - udp_set_opt　UDP通信端点オプション設定　拡張
  - ucp_get_opt　UDP通信端点オプションの読出し　拡張
- コールバック
  - ノンブロッキングコールの終了　標準
  - パケットの受信(UDP)のみ　標準

TOPPERS

その他のサービスコール

受付口・通信端点の削除

緊急データの送受信

- 緊急データは out-of-band のデータと扱う。実装依存に in-band のデータと扱うことも可能
- 受信（tcp_rcv_oob）はコールバック内で呼び出すことを想定

ペンディング中の処理のキャンセル

- ペンディング中の処理（サービスコール内でのブロック、ノンブロッキングコールで未完了）のキャンセル可能

通信端点オプションの設定/読出し

- 通信端点オプションを設定/読み出すサービスコール
- 通信端点オプションの使い方は実装依存

TCPのコールバック

コールバックルーチンの設定と呼出し

- TCP通信端点に対して1つのコールバックルーチンを設定できる（TCP受付口はコールバックを持たない）
- コールバックを起こした事象の種類を第1引数に、事象依存のパラメータを第2引数に渡す

コールバックを起こす事象

- ノンブロッキングコール完了通知
  - サービスコールから返値が渡される
  - 処理をキャンセルした場合も
- 緊急データの受信
  - コールバック中で緊急データを取り出さなければならない
  - 実装依存の事象

省コピーAPIによるデータの送受信

- ウィンドウバッファの領域が連続して取られている場合を想定（tcp_cre_cepでウィンドウバッファ領域を与えるとこうなる）。この場合、データ送受信時には、最低1回のコピーは避けられない
- ウィンドウバッファに直接読み書きするモデル
- 実際に何回コピーが必要かは、用いるLANコントローラやプロトコルスタックの実装に依存
  - 条件によってはゼロコピーになることも

tcp_get_buf　送信用バッファの取得（省コピーAPI）【標準】

【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	送信バッファに入れることができるデータの長さ/エラーコード

【特記事項】

- 送信バッファに1バイトでも空きがあればリターンする
- 連続している空き領域の長さを返す
- 続けて呼び出すと同じアドレスが返る（長さは変わる）
- 複数の送信要求がペンディングする場合はエラー

tcp_snd_buf　バッファ内のデータ送信(省コピーAPI)　【標準】

【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

【特記事項】

- tcp_get_bufで取得した送信バッファに入れたデータを送信する(よって、先頭番地を渡す必要はない)
- 実際には送信する手配をするだけなので、このサービスコール内でブロックすることはない

tcp_rcv_buf　受信バッファの取得(省コピーAPI)　【標準】

【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

【特記事項】

- バッファに1バイトでも受信データがあればリターンする
- 連続して置かれている受信データの長さを返す
- 続けて呼び出すと同じアドレスが返る(長さは変わる)
- 複数の受信要求がペンディングする場合はエラー
- 通信相手から切断されると戻り値が0となる

tcp_rel_buf　受信バッファの開放(省コピーAPI)　【標準】

【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

【特記事項】

- tcp_rcv_buf で取得した受信バッファに入っていたデータを捨てる(よって、先頭番地を渡す必要はない)
- このサービスコール内でブロックすることはない

省コピーAPI : プログラム例

```
/* 受信したデータ長とそれが入った番地を取り出す */
while (( ercd1 = tcp_rcv_buf(cepid, &rbuf, TMO_FEVR)) > 0){
    /* 送信用バッファを取り出す */
    ercd2 = tcp_get_buf(cepid, &sbuf, TMO_FEVR); /* エラー処理を省略 */
    len = min(ercd1, ercd2);
    for ( i = 0; i < len; i++){
        if (isupper(rbuf[i]))
            sbuf[i] = tolower(rbuf[i]);
        else
            sbuf[i] = rbuf[i];
    }
    /* データを送信する */
    ercd = tcp_snd_buf(cepid, len);          /* エラー処理を省略 */
    /* 受信用のバッファを開放する */
    ercd = tcp_rel_buf(cepid, len);          /* エラー処理を省略 */
}
```


ITRON-TCP/IP仕様

1. TCP/IPの基礎
2. ITRON-TCP/IP仕様
3. TINET（TCP/IPプロトコルスタック）
4. TINETデバイスドライバ

TINETの特徴

- 苫小牧高専情報工学科において開発されたITRON TCP/IP API仕様に準拠したコンパクトなTCP/IPプロトコルスタック
- FreeBSDをベースに
  - 枯れたソフトウェアで、他システムの手本
  - BSD ライセンスによる配布
- 組込みシステムのリソース制約に対応
- 対応 RTOS は TOPPERS/JSP及びASP カーネル
- API は ITRON TCP/IP API 仕様

実装上考慮した事項

- 組込みシステムのリソース制約（TINET のみ）
  - IPv4 は、RAM が約 8Kバイト、ROM が約 41Kバイト
  - IPv6 は、RAM が約 9Kバイト、ROM が約 59Kバイト
- ITRON TCP/IP API 仕様 の必要性能
  - 最小のコピー回数
  - 動的メモリ管理の排除
  - 非同期インターフェース
  - API毎のエラーの詳細化
- 実時間性の制約

TINET が提供する機能と実装ターゲット

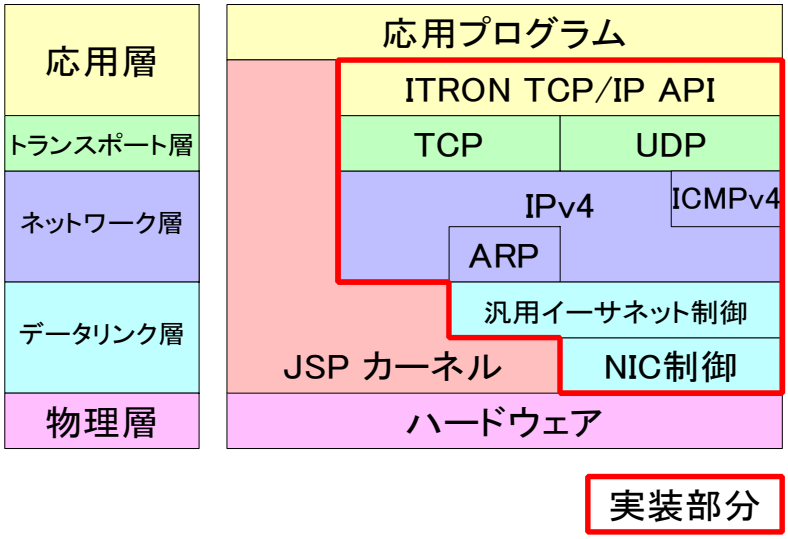
提供機能

- 幅広い応用層に対応
- TCP のオプションは MSS（Maximum Segment Size）のみ
- ネットワーク上の終端ノード
- 単一のネットワークインタフェース

実装ターゲット

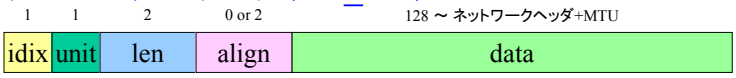
- 品川通信計装サービス製 NKEV-010H8
- 秋月電子通商製AKI-H8/3069F LAN ボード
  - NE2000 互換 NIC
  - 内部ループバック
  - シリアルインタフェースを利用した PPP
- 北斗電子製 HSB7727ST（SH3）

ネットワーク階層図（IPv4）



TOPPERS

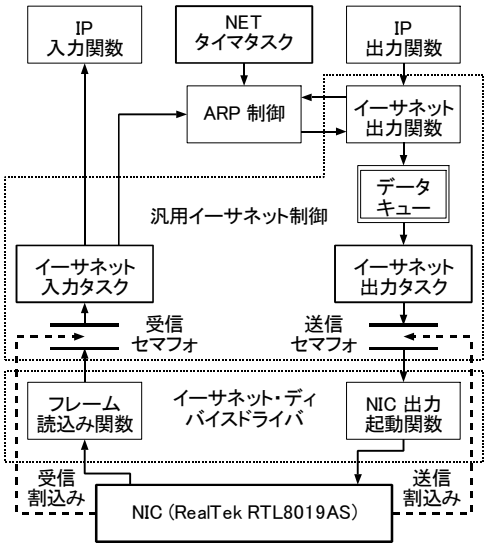
ネットワークバッファ（net_buf）



- TCP/UDP、IP、イーサネットヘッダとデータが入るプロコトルバッファ
- 出力時はTCP/UDPで確保
- 入力時はイーサネットデバイスドライバで確保
- BSD の mbuf に相当
- 固定長のメモリブロック
  - 下位層のヘッダ領域をあらかじめ確保。
  - 途中での動的なメモリ操作は行わない。
  - ヘッダ長は 4 オクテット、1,514 オクテットのイーサネットフレームでオーバヘッドは 0.4%。
  - MTU（Maximum Transmission Unit）
    - ネットワークの最大転送単位（Ethernetで1500）

TOPPERS

イーサネット制御とデータの流れ



TOPPERS

ITRON TCP/IP API の実装

- API
  - ITRON TCP/IP API仕様の標準機能
  - 暫定的な ITRON TCP/IP（バージョン6） API仕様の標準機能
- TCP
  - BSD の通信機能
  - 最大セグメントサイズ（MSS）オプション
  - 省コピー API
  - ノンブロッキングコール
- UDP
  - ノンブロッキングコール
- 静的API
  - TCP/UDP 通信端点の生成
  - TCP 受付口の生成

TOPPERS

TCP のAPI

API 名	機　　能	NBLK*		省コピー
		送信系	受信系	
TCP_CRE_REP	TCP受付口の生成			
TCP_CRE_CEP	TCP通信端点の生成			
tcp_acp_cep	接続要求待ち		○	
tcp_con_cep	接続要求	○		
tcp_sht_cep	データ通信終了			
tcp_cls_cep	通信端点のクローズ		○	
tcp_snd_dat	データの送信	○		
tcp_rcv_dat	データの受信		○	
tcp_get_buf	送信用バッファの取得	○		○
tcp_snd_buf	バッファ内データの送信			○
tcp_rcv_buf	受信バッファの取得		○	○
tcp_rel_buf	受信用バッファの解放			○
tcp_can_cep	処理のキャンセル			

NBLK*: ノンプロッキングコール

UDP のAPI

API 名	機　　能	NBLK	
		送信系	受信系
UDP_CRE_CEP	UDP通信端点の生成		
udp_snd_dat	パケットの送信	○	
udp_rcv_dat	パケットの受信		○
udp_can_cep	処理のキャンセル		
UDPパケットの受信	UDPパケットの受信		

TINET ディレクトリ構成

tinet/	: ルートディレクトリ
./cfg	: TINET コンフィギュレータ
./doc	: ドキュメント類
./net	: 汎用ネットワーク
./netapp	: サンプルのネットワークプログラム
./netdev	: ネットワークインタフェースのドライバ
./netdev/if_ed	: NE2000互換イーサネットデバイスドライバ
./netinet	: TINET の本体（主に IPv4）
./netinet6	: IPv6 の本体

TINET のドキュメント（./doc）

- tinet.txt
  - メインマニュアル．API，アプリケーション構築方法
- tinet_install.txt
  - インストールマニュアル．サンプルアプリの紹介
- tinet_config.txt
  - コンフィギュレーションマニュアル．マクロの解説等
- tinet_defs.txt
  - プロセッサ、システム依存の解説
- tinet_sample.txt
  - サンプルプログラムの説明
- tinet_chg.txt
  - 変更履歴

コンフィギュレーション/パラメータ定義ファイル

- config/$(CPU)/tinet_cpu_config.h
  - プロセッサ依存定義ファイル
- config/$(CPU)/$(SYS)/tinet_sys_config.h
  - システム依存定義ファイル
- tinet/netdev/$(NIC)/tinet_nic_config.h
  - ネットワークインタフェース依存定義ファイル
- config/$(CPU)/tinet_cpu_defs.h
  - プロセッサ依存定義ファイル
- tinet/netdev/$(NIC)/tinet_nic_defs.h
  - ネットワークインタフェース定義ファイル

応用プログラムファイル

$(APP_DIR)/tinet_$(UNAME).cfg
- TINET コンフィギュレーションファイル
- ITRON TCP/IP仕様の静的APIを記述する
- TINETコンフィギュレータにより処理される

$(APP_DIR)/tinet_app_config.h
- アプリケーションに依存する TINET コンフィギュレーション・パラメータを定義するファイル

$(APP_DIR)/route_cfg.c
- 静的経路定義ファイル

tinet_app_config.h の例

```
//IP アドレスの設定
#define IPV4_ADDR_LOCAL                 ¥
    MAKE_IPV4_ADDR(172,25,129,88)
#define IPV4_ADDR_LOCAL_MASK            ¥
    MAKE_IPV4_ADDR(255,255,255,0)
#define IPV4_ADDR_DEFAULT_GW            ¥
    MAKE_IPV4_ADDR(172,25,129,140)
```


静的経路定義ファイル route_cfg.c の例

```
T_IN4_RTENTRY
routing_tbl[NUM_ROUTE_ENTRY] = {
    /* default  gateway、間接配送 */
    { 0,  0,   IPV4_ADDR_DEFAULT_GW },
    /* 同一 LAN 内、直接配送  */
    { IPV4_ADDR_LOCAL & IPV4_ADDR_LOCAL_MASK,
      IPV4_ADDR_LOCAL_MASK,  0 },
    /* 同一 LAN 内へのブロードキャスト、直接配送 */
    { 0xffffffff,     0xffffffff,  0 },
};
```


TINETコンフィギュレータ生成ファイル

$(APP_DIR)/tinet_cfg.c
- TCP 受付口、TCP 通信端点、及び UDP 通信端点に対応する構造体が生成されるファイル。アプリケーションプログラム、TINET と共にコンパイルしてリンクする。

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

$(APP_DIR)/tinet_id.h
- TCP 受付口、TCP 通信端点、及び UDP 通信端点の ID 自動割付結果ファイル。

ファイル tinet_kern.cfg の例

```
CRE_SEM(SEM_TCP_CEP_LOCK1,{ TA_TPRI, 1, 1 });
CRE_SEM(SEM_TCP_CEP_SBUF_BUSY1,
        { TA_TPRI, 1, 1 });
CRE_SEM(SEM_TCP_CEP_RBUF_READY1,
        { TA_TPRI, 0, 1 });
CRE_FLG(FLG_TCP_CEP1,
        { TA_TFIFO|TA_WSGL,CEP_EVT_CLOSED });
```


ITRON-TCP/IP仕様

1. TCP/IPの基礎
2. ITRON-TCP/IP仕様
3. TINET（TCP/IPプロトコルスタック）
4. TINETデバイスドライバ

TINETのコンパイル変数定義

- TINETでは、プラットフォーム依存部として以下のコンパイル変数定義を使用する
  - $(CPU) プロセッサの総称名、本実装ではarmv4
  - $(SYS) システムの総称名、本実装ではlpc2388
  - $(NET_DEV) イーサネットデバイスドライバの総称名
    本実装ではif_lpcemac
- $(UNAME)はアプリケーションプログラム名を意味する
- これらのコンパイル変数定義はMakefileにて定義を行い、実装依存部のディレクトリ上の保管位置を示す

実装依存部ファイル

- TINET内部パラメータ調整用ファイル

ファイル名	ディレクトリ	定義内容
tinet_config.h	tinet	以下のファイルのインクルード
tinet_app_config.h	$(UNAME)	アプリプログラムの依存定義
tinet_cpu_config.h	config/$(CPU)	プロセッサ依存定義
tinet_sys_config.h	config/$(CPU)/$(SYS)	システム依存定義
tinet_nic_config.h	tinet/netdev/$(NET_DEV)	イーサネットデバドラ依存定義

- TINETに対するハードウェア依存性定義ファイル

ファイル名	ディレクトリ	定義内容
tinet_defs.h	tinet	以下のファイルのインクルード
tinet_cpu_defs.h	config/$(CPU)	プロセッサ依存定義
tinet_nic_defs.h	tinet/netdev/$(NET_DEV)	イーサネットデバドラ依存定義

イーサネットデバイスドライバマクロ

- TINETはネットワーク端末ノードを単一ネットワークのみ対象とする、そのためIP層からドライバを直接呼び出し可能
- IP層ではマクロ化したデバイスドライバ関数を呼び出す記述でインターフェイスの依存性を排除している

マクロ名	設定内容
IF_ETHER_NIC_START(i, o)	NIC出力起動関数
IF_ETHER_NIC_GET_SOFTC()	NICデバイスドライバ共通構造体取得
IF_ETHER_NIC_PROBE(i)	NIC検出関数
IF_ETHER_NIC_INIT(i)	NIC初期化関数
IF_ETHER_NIC_READ(i)	NICイーサネットフレーム読み込み関数
IF_ETHER_NIC_RESET(i)	NICリセット関数
IF_ETHER_NIC_WATCHDOG(i)	NICワッチドック関数
IF_ETHER_NIC_IN6_IFID(I,a)	NICインターフェイス識別子の設定
IF_ETHER_NIC_ADDMULTI(a)	NICマルチキャストアドレス追加関数
T_IF_ETHER_NIC_SOFTC	NIC固有構造体

デバイスドライバ用のデータ定義

- デバイスドライバのインクルードファイルに、デバイスドライバ管理用の構造体を定義する
- TINETは中身を参照せず、取り扱う

```
#define T_IF_ETHER_NIC_SOFTC   struct t_emac_softc
```

- デバイスドライバソースで定義する

```
/*
 *  ネットワークインタフェースに依存するソフトウェア情報
 */
typedef struct t_emac_softc {
    UW      nic_addr;               /* NIC のベースアドレス   */
    UH      phy_addr;               /* PHY ADDRESS番号 */
    UB      phy_speed;
    UB      phy_duplex;
    UW      tx_error;               /* 送信エラー割込み回数 */
    UW      rx_error;               /* 受信エラー割込み回数 */
} T_EMAC_SOFTC;
```


デバイスドライバの初期化

- ドライバの初期化は、ether_input_taskの初期化にて、３つのマクロを呼び出す手順で行う

```
void
ether_input_task(VP_INT exinf)
{
        T_IF_SOFTC                      *ic;
          ....

        /* ネットワークインタフェース管理を初期化する。*/
        ifinit();

        /* NIC を初期化する。*/
        ic = IF_ETHER_NIC_GET_SOFTC();
        IF_ETHER_NIC_PROBE(ic);
        IF_ETHER_NIC_INIT(ic);

        /* Ethernet 出力タスクを起動する */
        syscall(act_tsk(ETHER_OUTPUT_TASK));

        /* ネットワークタイマタスクを起動する */
        syscall(act_tsk(NET_TIMER_TASK));
```


送信処理

- 送信用タスクether_output_taskで送信要求があった場合、IF_ETHER_NIC_STARTマクロを呼び出す

```
void
ether_output_task(VP_INT exinf)
{
        T_IF_SOFTC                      *ic;
        ....
        get_tid(&tskid);
        syslog(LOG_NOTICE, "[ETHER OUTPUT:%d] started.", tskid);
        ic = IF_ETHER_NIC_GET_SOFTC();
        while (TRUE) {
                while (rcv_dtq(DTQ_ETHER_OUTPUT, (VP_INT*)(VP)&output) == E_OK) {
                        NET_COUNT_ETHER(net_count_ether.out_octets,  output->len);
                        NET_COUNT_MIB(if_stats.ifOutOctets, output->len + 8);
                        NET_COUNT_ETHER(net_count_ether.out_packets, 1);
                        .....
                        syscall(wai_sem(ic->semid_txb_ready));
                        IF_ETHER_NIC_START(ic, output);
                        .....
                }
```


受信処理

- 初期化と同じタスクで、受信割込みからのセマフォ通知の後、受信関数マクロIF_ETHER_NIC_READを呼び出す

```
void
ether_input_task(VP_INT exinf)
{
        T_IF_SOFTC                      *ic;
        ....
        /* ネットワークインタフェース管理を初期化する。*/
        ifinit();
        ....
        ether_ifnet.ic = ic;

        while (TRUE) {
                syscall(wai_sem(ic->semid_rxb_ready));
                if ((input = IF_ETHER_NIC_READ(ic)) != NULL) {
                        NET_COUNT_ETHER(net_count_ether.in_octets,  input->len);
                        NET_COUNT_MIB(if_stats.ifInOctets, input->len + 8);
                        NET_COUNT_ETHER(net_count_ether.in_packets, 1);
                        eth = GET_ETHER_HDR(input);
                        proto = ntohs(eth->type);
                        ....
```

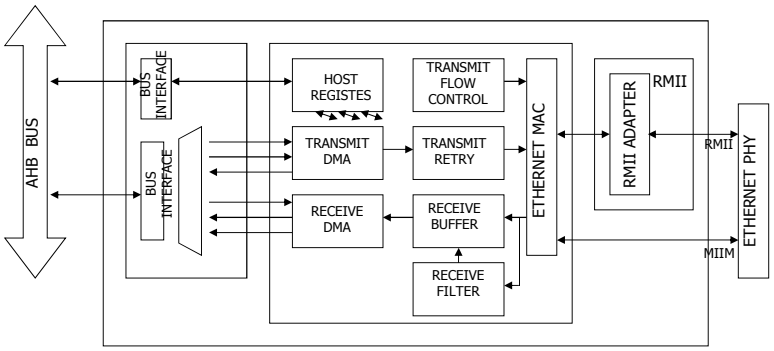

TINETデバイスドライバの設計

1. RTOS依存部
2. netdev/$(NET_DEV)

TOPPERS

LPC2388のETHERNETドライバ設定

- この章では、LPC2388に実装のEMAC用のドライバに対する実装説明を行う
- EMACドライバではRAM上の送受信データをDMAを使ってETHERNET上に送受信する設定となっている



TOPPERS

CPU依存部の設定

- config/armv4では以下の2つのインクルードファイルを必要とする
  - tinet_cpu_config.h　　プロセッサ依存config定義
  - tinet_cpu_defs.h　　プロセッサ依存define定義
- tinet_cpu_config.hは通信デバイス依存の設定を行う、デフォルトのif_edと同設定でも問題ない
- tinet_cpu_defs.hはアライン設定で、デフォルトのif_edと同設定で問題ない

ディスクリプタ管理

- EMACの送受信データはEthernet RAM(0x7EF00000～0x7FE03FFF)の16KBのメモリ上でディスクリプタを介して管理する
- ディスクリプタは送信用、受信用がある
- Control Registerで管理を行う

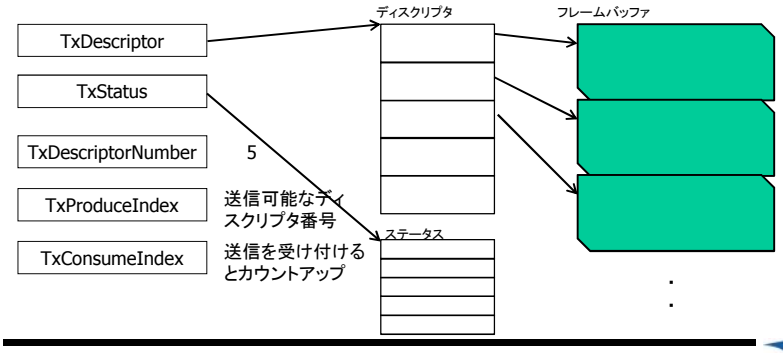
Control Registers

name	address	access	width	discpition
Command	0xFFE00100	R/W	2	コントロールレジスタ
Status	0xFFE00104	RO	4	ステータスレジスタ
RxDisciptor	0xFFE00108	R/W	8	受信ディスクリプタベースアドレス
RxStatus	0xFFE0010C	R/W	8	受信ステータスベースアドレス
RxDescriptorNumber	0xFFE00110	R/W	6	受信ディスクリプタの数
RxProduceIndex	0xFFE00114	RO	6	受信produce index
RxCousumeIndex	0xFFE00118	R/W	5	受信consume index
TxDescriptor	0xFFE0011C	R/W	5	送信ディスクリプタベースアドレス
TxStatus	0xFFE00120	R/W	3	送信ステータスベースアドレス
TxDescritorNumber	0xFFE00124	R/W	9	送信ディスクリプタの数
TxProduceIndex	0xFFE00128	R/W	4	送信produce index
TxConsumeIndex	0xFFE0012C	RO	12	受信produce index

ディスクリプタはフレーム領域へのポインタとコントロールフィールドの2ワード（8バイト）で構成される
ステータスは、送信用は1ワード（4バイト）、受信用は2ワード（8バイト）のステータスで構成される

実際のEthernet RAMの構成

- 16KBのRAMに送受信用のディスクリプタ、ステータス、フレームバッファを配置しなければならない
- 1つのディスクリプタは送信用1548バイト受信用1552バイトであるため、Ethernet RAM上に5個ずつ配置する
- 送信用の配置を以下に示す



TOPPERS

システム依存設定

- config/armv4/lpc2388/tinet_sys_config.hではデバイスドライバに依存した定義を行う

```
/*   NIC (EMAC 互換) に関する定義 */
#define TMO_IF_ED_GET_NET_BUF   1    /* [ms]、受信用 net_buf 獲得タイムアウト */
#define TMO_IF_ED_XMIT          (2*IF_TIMER_HZ)
/*#define IF_ED_CFG_ACCEPT_ALL  マルチキャスト、エラーフレームも受信するときはコメントを外す。*/
/*  EMACに関する定義 */
#define INHNO_ETHER             INTNO_ETHER          /* IRQ5 */
#define LPC_ETHER_ADDR_LEN      6       /* Ethernet (MAC) Addressの長さ*/
/* MAC用のクロック設定値 */
#define PCONP_EMAC_CLOCK        0x40000000
/* EMAC MODULE ID                    */
#define PHILIPS_EMAC_MODULE_ID  ((0x3902 << 16) | 0x2000)
/*   PHYに関する定義 */
#define PHY_ADDRNO              1       /* PHYアドレス番号 */
#define USE_RMII                        /* PHYとの通信にRMIIを使用 */
#ifndef _MACRO_ONLY
#define ed_ena_inter(ipm)       chg_ipm(ipm)
/*   データ */
extern UB lp23xx_macaddr[LPC_ETHER_ADDR_LEN];
/*   関数 */
void tinet_sys_initialize(void);
#endif /* of #ifndef _MACRO_ONLY */
```


TOPPERS

システム依存定義

- config/armv4/lpc2388/tinet_sys_config.cではイーサネットクロックやPHY設定を行う関数を記述

```
void
tinet_sys_initialize(void)
{
        unsigned long tmp;
        tmp  = sil_rew_mem((VP)(TADR_SCB_BASE+TOFF_SCB_PCONP));
        tmp |= PCONP_EMAC_CLOCK;
        sil_wrw_mem((VP)(TADR_SCB_BASE+TOFF_SCB_PCONP), tmp);
        sil_dly_nse(20000);
#ifdef USE_RMII
        tmp = sil_rew_mem((VP)(TADR_MAC_BASE+TOFF_MAC_MODULEID));
        if ( tmp == PHILIPS_EMAC_MODULE_ID ){
                sil_wrw_mem((VP)(TADR_PINSEL_BASE+TOFF_PINSEL2), 0x50151105);
        }
        else{
                sil_wrw_mem((VP)(TADR_PINSEL_BASE+TOFF_PINSEL2), 0x50150105);
        }
#else                                           /* else RMII, then it's MII mode */
        sil_wrw_mem((VP)(TADR_PINSEL_BASE+TOFF_PINSEL2), 0x55555555);   /* selects P1[15:0] */
#endif
        sil_wrw_mem((VP)(TADR_PINSEL_BASE+TOFF_PINSEL3), 0x00000005);   /* selects P1[17:16] */
}
```


TINETデバイスドライバの設計

1. RTOS依存部
2. netdev/$(NET_DEV)

LPC用EMACドライバの実装

- EMACドライバの保管場所
  - RTOSディレクトリ/tinet/netdev/if_lpcemac
- 実装ファイル
  - Makefile.tinet　デバイスドライバ用メイクファイル
  - tinet_nic_defs.h　ネットワークインターフェイス定義ファイル
  - tinet_nic_config.h ネットワークインターフェイス依存定義ファイル
  - nic_rename.h　デバイスドライバ実装名リネームファイル
  - nic.cfg　デバイスドライバ用コンフィギュレーションファイル
  - if_lpcemac.h　LPC-EMAC用インクルードファイル
  - if_lpcemac.cfg　LPC-EMAC用コンフィギュレーションファイル
  - if_lpcemac.c　LPC-EMAC用デバイスドライバソースファイル

ドライバ用コンフィギュレーションファイル

- nic.cfg
  - NIC用カーネルオブジェクトの指定を行う、LPC-EMACではif_lpcemac.cfgを指定（インクルードする）
- if_lpcemac.cfg
  - デバイスドライバで使用する割込みハンドラと汎用イーサネット制御用との同期送受信セマフォの定義を行う

```
INCLUDE("¥"if_lpcemac.h¥"");

/* 割込みハンドラ */
DEF_INH(INHNO_ETHER, {TA_HLNG, if_ether_handler });

/* 入出力同期用セマフォ */
CRE_SEM(SEM_IF_EMAC_SBUF_READY, {TA_TPRI, NUM_IF_EMAC_TXBUF,
                                          NUM_IF_EMAC_TXBUF});
CRE_SEM(SEM_IF_EMAC_RBUF_READY, {TA_TPRI, 0,
                                          NUM_IF_EMAC_RXBUF});
```

ドライバ用メイクファイル：Makefile.tinet

- デバイスドライバコンパイル用のコンパイルオプション、依存性検索パス及びオブジェクトの指定を行う
- RTOSディレクトリ/tinet/Makefile.tinetよりイーサネットドライバを使用する場合、インクルードされる

```
#
# コンパイルオプション
#
INCLUDES      := $(INCLUDES) –I$(TINET_ROOT)/netdev/$(NET_DEV)
TINET_DIR     := $(TINET_DIR):$(TINET_ROOT)/netdev/$(NET_DEV)
TINET_COBJS := $(TINET_COBJS)  if_lpcemac.o
TINET_KERNEL_CFG := $(TINET_KERNEL_CFG)
              $(TINET_ROOT)/netdev/$(NET_DEV)/if_lpcemac.cfg
```


ドライバ定義用インクルードファイル：tinet_nic_defs.h

- TINETに対するNICのハードウェア依存性定義ファイル
- 本実装ではイーサネットヘッダのアライン調整量のみを指定している

```
/*
 * T_ETHER_HDRで、アラインを調整する場合、調整量を指定する。
 * 調整しない場合は、 0を指定する。
 */
#define IF_ETHER_NIC_HDR_ALGIN     0
```


ドライバ用インクルードファイル

- tinet_nic_config.h
  - tinetで定義されたドライバインクルードファイル
  - 実際は以下の2つのインクルードファイルをインクルードする
- nic_rename.h
  - ドライバ関数の実装名を定義する
- if_lpcemac.h(実装依存)
  - 実装依存のインクルードファイル
  - ドライバの個別定義、マクロ名の変換を行う

if_lpcemac.h

- ドライバの設定定義を行う
- EMACやPHYの設定定義を行う
- マクロ名のドライバ関数を実関数に割り当てる定義を行う

マクロ名	対応ドライバ関数	実装名
IF_ETHER_NIC_START(i, o)	lpcemac_start(i, o)	_tinet_emac_start
IF_ETHER_NIC_GET_SOFTC()	lpcemac_get_softc()	_tinet_emac_get_softc
IF_ETHER_NIC_PROBE(i)	lpcemac_probe(i)	_tinet_emac_probe
IF_ETHER_NIC_INIT(i)	lpcemac_init(i)	_tinet_emac_init
IF_ETHER_NIC_READ(i)	lpcemac_read(i)	_tinet_emac_read
IF_ETHER_NIC_RESET(i)	lpcemac_reset(i)	_tinet_emac_reset
IF_ETHER_NIC_WATCHDOG(i)	lpcemac_watchdog(i)	_tinet_emac_watchdog
IF_ETHER_NIC_IN6_IFID(I,a)	get_mac6_ifid(i, a);	ドライバでは未定義
IF_ETHER_NIC_ADDMULTI(a)	lpcemac_addmulti(s)	_tinet_emac_addmulti
T_IF_ETHER_NIC_SOFTC	struct t_emac_softc	

LPC用EMACドライバ：if_lpcemac.c

- EMAC用の制御とPHYの制御をおこなう
- EMAC用のイーサネットデバイスドライバ制御用にローカルな構造体（T_EMAC_SOFTC）を定義する

T_EMAC_SOFTC構造体の定義

項目	型	内容	初期値
nic_addr	UW	LPC-EMACのレジスタベースアドレス	TADR_EMC_BASE (0xFFE08000)
phy_addr	UH	PHY番号	PHY_ADDRNO(1)
phy_speed	UB	通信速度	
phy_duplex	UB	DUPLEX設定	
tx_error	UW	送信エラー回数	
rx_error	UW	受信エラー回数	

lpcemac_get_softc

- プロトコルスタックからとNICの情報を指定する場合、T_IF_SOFTC型のデータ交換を行う。この型はnet/ethernet.hで定義されデバイスドライバでインスタンス化する
- lpcemac_get_softc関数はデバイスドライバで指定するT_IF_SOFTC型のデータのポインタを取り出す関数

T_IF_SOFTC型の定義

項目	型	内容	初期値
ifaddr	T_IF_ADDR	netI/F address	
timer	UH	送信timeout	0
sc	T_IF_ETHER_NIC_SOFTC*	依存部	T_EMAC_SOFTC
semid_txb_ready	ID	送信セマフォ	SEM_IF_EMAC_SBUF_READY
semid_rxb_ready	ID	受信セマフォ	SEM_IF_EMAC_RBUF_READY

lpcemac_probe

- NIC検出関数
- もともと、NICの有無を検出用の関数だが、LPCではLPC-EMACが確定しているため、初期化とMACアドレスの取得を行う
- NICの初期化はtinet_sys_initialize関数の呼び出しで実行する
  - クロック設定、PHYのインターフェイスの初期化を行う
- MACアドレスはアプリケーションごとに、lpcemac_probe関数実行前に、lp23xx_macaddrにMACアドレスを設定することを前提としている
  - アプリケーションごとにmacaddr_init関数をATT_INI静的APIで実行することにより実行している

lpcemac_init

- NIC初期化関数
- PHYの初期化、EMACの初期化、送受信ディスクリプタの設定、MACの送受信を許可する

```
void
lpcemac_init(I_IF_SOFTC *ic)
{
   int result;
   unsigned long tmp;
   syslog_1(LOG_NOTICE, “start lpcemac_init(%08x)”, ic);
   if(!(result = lpcemac_phy_initilaize(ic))){
      syslog_1(LOG_ERROR, “lpcemac_phy_initailze result(%d)”, result);
      slp_tsk();
   }
   /* MAC TX/RX enable */
   tmp  = sil_rew_mem((VP)(TADR_MAC_BASE+TOFF_MAC_COMMAND);
   tmp |= MAC_TX_ENABLE | MAC_RXENABLE;
   sil_wrw_mem((VP)(TADR_MAC_BASE+TOFF_MAC_COMMAND, tmp);
   tmp  = sil_rew_mem((VP)(TADR_MAC_BASE+TOFF_MAC_MAC1);
   tmp |= MAC_RECEIVEENABLE;
   sil_wrw_mem((VP)(TADR_MAC_BASE+TOFF_MAC_MAC1);
}
```


lpcemac_read

- 受信フレームの取り込みを行う。戻り値は取得したバッファのポインタ。
- RxProduceIndexとRxConsumeIndexに差がある場合、RxConsumeIndexに対応した受信ディスクリプタから受信フレームを取り出しバッファにセットする
- そのあと、RxConsumeIndexをインクリメントする
- RxProduceIndexとRxConsumeIndexに、まだ差がある場合は、まだ読み取っていないイーサネットフレームがあるので受信セマフォ（ic->semid_rxb_ready）をセットして、イーサネット入力タスクと同期をとる
- 受信用のバッファはtget_net_buf関数で取得する。取得できなかった場合は、フレームは破棄され、この関数は戻り値NULLが返される

lpcemac_readソースコード

- 以下のソースは簡略化しています

```
T_NET_BUF *
lpcemac_read (T_IF_SOFTC *ic)
{
    T_EMAC_SOFTC  *sc = ic->sc;
    T_NET_BUF     *input = NULL;
      ・・・
    if(RxProduceIndex != RxConsumeIndex) {
          ・・・
        frame_ptr    = (char *)*rx_desc_addr;
        len = (*rx_status_addr & DESC_SIZE_MASK) - 1;
        if (len <= EMAC_BLOCK_SIZE) {
            input = lpcemac_get_frame(sc, frame_ptr, len);
            syslog_5(LOG_INFO, "ETH[RECV][%08x](%d) type[%04x] src[%08x] dst[%08x]",
                    input->buf, input->len,
                   (input->buf[12]<<8)|input->buf[13],
                   (input->buf[26]<<24)|(input->buf[27]<<16)|(input->buf[28]<<8)|input->buf[29],
                   (input->buf[30]<<24)|(input->buf[31]<<16)|(input->buf[32]<<8)|input->buf[33]);
        }
          ・・・
    }
    if (RxProduceIndex != RxConsumeIndex)
        sig_sem(ic->semid_rxb_ready);
    return input;
}
```

受信したパケットを
LOG_INFOで表示します

lpcemac_start

- NIC出力起動関数、送信開始要求を行う
- TxProduceIndexとTxConsumeIndexを取り出し、差がある場合は未処理
- TxProduceIndexに対応した送信ディスクリプタに送信データをコピーし、ディスクリプタの属性に送信要求とデータ長を書き込む
- TxProduceIndexをインクリメントする

lpcemac_startソース

- 以下のソースは簡略化しています

```
void
lpcemac_start (T_IF_SOFTC *ic, T_NET_BUF *output)
{
    long *tx_desc_addr;
    int TxProduceIndex,TxConsumeIndex,i, templen;

    TxProduceIndex = sil_rew_mem((VP)(TADR_MAC_BA
    TxConsumeIndex = sil_rew_mem((VP)(TADR_MAC_B
    syslog_2(LOG_DEBUG, "ETH[SEND] p(%d) c(%d)", TxPro      dex, TxConsumeIndex);
    if(TxProduceIndex == NUM_IF_EMAC_TXBUF)
        sil_wrw_mem((VP)(TADR_MAC_BASE+TOFF_MAC_T      RODUCEINDEX), 0);
    tx_desc_addr = (long *)(TX_DESCRIPTOR_ADDR + TxPr      uceIndex * 8);
    memcpy((char *)(*tx_desc_addr), output->buf, output->len);
    syslog_5(LOG_INFO, "ETH[SEND][%08x](%d) type[%04x] src[%08x] dst[%08x]",
            output->buf, output->len,
           (output->buf[12]<<8)|output->buf[13],
           (output->buf[26]<<24)|(output->buf[27]<<16)|(output->buf[28]<<8)|output->buf[29],
           (output->buf[30]<<24)|(output->buf[31]<<16)|(output->buf[32]<<8)|output->buf[33]);
    *(tx_desc_addr+1) = (long)(EMAC_TX_DESC_INT | ・・・ | (output->len -1));
    TxProduceIndex++;                       /* transmit now */
    if(TxProduceIndex == NUM_IF_EMAC_TXBUF )
        TxProduceIndex = 0;
    sil_wrw_mem((VP)(TADR_MAC_BASE+ ・・・), TxProduceIndex);
}
```

送信したパケットを
LOG_INFOで表示します

if_ether_handler

- LPC-EMACからの割込み関数
- 割込み属性EMAC_INT_RXDONEが発生した場合、受信セマフォ(ic->semid_rxb_ready)をセットして、イーサネット入力タスクと同期をとる
- 割込み属性EMAC_INT_TXDONEが発生した場合、送信セマフォ(ic->semid_txb_ready)をセットして、イーサネット出力タスクと同期をとる
- 上記以外は、基本的に通信エラーケースなので、履歴の設定を行う

lpcemac_reset

- NICリセット関数

```
/*
 *  lpcemac_reset – emac ネットワークインターフェイスをリセットする
 */
void
lpcemac_reset(T_IF_SOFTC *ic)
{
    int result;

    disint();
    result = lpcemac_phy_initilaize(ic);
    enaint();
    NET_COUNT_ETHER_NIC(net_count_ether_nic[NC_ETHER_RESETS], 1);
    syslog_2(LOG_NOTICE, “lpcemac_reset(%08x);(%d)”, ic, result);
}
```


lpcemac_watchdog

- NICウォッチドック関数、送信用のタイムアウトが発生した場合、この関数が呼び出される
- エラーカウンタアップとイーサネットドライバのリセットを行う

```
/*
 *  lpcemac_watchdog – emac ネットワークインターフェイスのウォッチドッグタイムアウト
 */
void
lpcemac_watchdog(T_IF_SOFTC *ic)
{
    NET_COUNT_ETHER_NIC(net_count_ether_nic[NC_ETHER_OUT_ERR_PACKETS], 1);
    NET_COUNT_ETHER_NIC(net_count_ether_nic[NC_ETHER_TIMEOUTS], 1);
    lpcemac_reset(ic);
}
```


PHY用の関数

- PHY用の関数として以下の3つの関数を用意する
- INT init_PHY(T_EMAC_SOFTC *sc)
  - PHYの初期化を行う、現状接続はオートネゴシエーションとなっている。他の設定を行いたい場合はプログラムの書き換えが必要
- void write_PHY(UH phyaddr, INT reg, INT data)
  - PHYレジスタにデータを書き込む
- INT read_PHY(UH phyaddr, INT reg)
  - PHYレジスタの内容を読み込む

DHCP+ECHOサーバの実装確認

1. ECHOサーバ
2. DHCP

エコーサーバーを作成

- エコーサーバーはTCPのサーバーとして起動し、クライアントから受信した文字を一つ進めて返す
- 実行すると、TCPを用いてポート番号“0007”でクライアントからの接続を待つ
- クライアントと接続した後は、クライアントから送られてきたデータをクライアントに送り返す

```
$ cd ECHO4
$ ls
Makefile     echos4.cfg    route_cfg.c   tinet_echos.cfg
echos4.c     echos4.h      tinet_app_config.h
```


TINETを使用したエコーサーバーの作成

- TINETの機能を使用してエコーサーバーアプリを作成する場合、tinet_jsp_configureコマンドによりプログラムを自動生成できます
- ver 1.5.2では、IPv4とIPv6のエコーサーバープログラムをサポートしています、ここではIPv4用のエコーサーバーを使用します
- 開発環境のECHOS4は、これにmacaddrの初期化とタスクモニタを加えています

```
$ cd ECHO4
$ ../../../tinet/tinet_jsp_configure –C armv4 –S lpc2388
   -A echos4 –i ether –v if_lpcemac –n inet4 –s tcp
$ ls
Makefile      echos4.cfg    route_cfg.c   tinet_echos.cfg
echos4.c      echos4.h      tinet_app_config.h
```

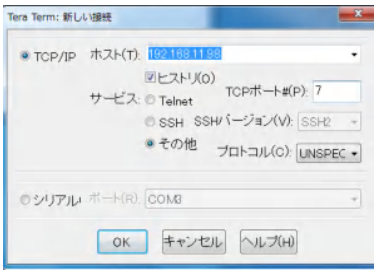

エコーサーバー：接続

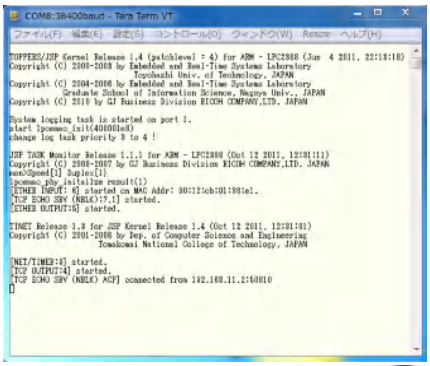
- パソコンに固定アドレス（192.168.11.2）を設定し、telnetまたは、TeraTermを用いる
- XPの場合、telnetはスタートメニューの”ファイル名を指定して実行”から実行する
  - telnet 192.168.11.98 7
- Windows7では、telnetをサポートしていないため、TeraTermより接続を行う
  - 最新のTeraTermでは行単位送信がデフォルトとなっているので、文字単位では送信を行わない
  - これを変更するにはTERATERM.INIの以下を変更

    EnableLineMode=on ⇒EnableLineMode=off

エコーサーバーの接続：TeraTerm

- TeraTermを起動し、新しい接続からTCP/IP:サービス・その他：TCPポート#・7を選択する
- 接続後、TerTermの192.168.11.98:7-Tera Term VTから文字を入力するとエコーバック表示される





TOPPERS

エコーサーバーアプリ・ファイル構成

- Makefile : メイクファイル
- route_cfg.c : 静的ルーティング設定ファイル
- tinet_app_config.h : コンフィギュレーション定義ファイル
- tinet_echoes4.cfg : TCP/IPコンフィギュレーション
- echos4.h : ユーザープログラムヘッダ
- echos4.c : ユーザープログラムソース
- echos4.cfg : jspコンフィギュレーションファイル

TOPPERS

エコーサーバーのモード設定

- ITRON-TCP/IP仕様では、省メモリや高速化のために4つの実装モードを設定可能です
- エコーサーバではコンパイルスイッチによりモード設定が可能です。この設定はechos4.hで定義している
  - USE_TCP_NON_BLOCKING
  - USE_COPYSAVE_API
- デフォルトの設定ではノンブロッキングモードの非コピーセーブとしている

```
/*
 * 使用するAPIの選択
 */
#define USE_TCP_NON_BLOCKING
//#define USE_COPYSAVE_API
```


エコーサーバー・メイクファイル（Makefile）

- JSPの標準的なMakefileとほぼ同じ
- 各種設定を定義してから、TINETのMakefile.configをインクルードする

```
# ネットワークインタフェース
NET_IF = ether

# イーサネット・ディバイスドライバの選択
NET_DEV = if_lpcmac

# ネットワーク層の選択
SUPPORT_INET4 = true

# トランスポート層の選択
SUPPORT_TCP = true

#
# TINET の Makefile.config のインクルード
#
include $(SRCDIR)/monitor/Makefile.config
include $(SRCDIR)/tinet/Makefile.config
```


エコーサーバー・静的ルーティング設定ファイル

- 静的ルーティング情報として、tinet/netinet/in_var.hで定義されているT_IN4_RTENTRY型の構造体の配列を作成
- 詳細はTINETのマニュアル（tinet/doc/tinet.txt）の6章を参照
- デフォルトゲートウェイのみのシンプルなネットワークでは、サンプルアプリケーションのroute_cfg.cをそのまま流用可能
- 静的経路表の書式

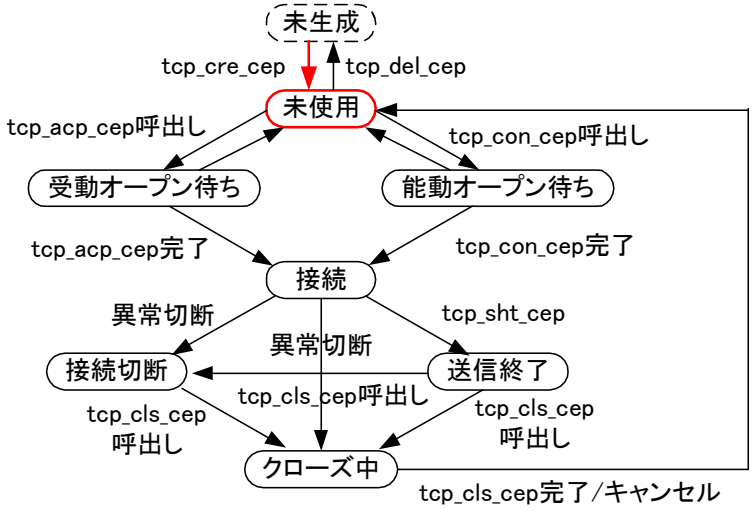
```
T_IN4_RTENTRY routing_tbl[NUM_ROUTE_ENTRY] = {
        { 0             0,              <gateway> },    /* ディフォルト       */
        { <net>,        <mask>,         0         },    /* 直接配送           */
        { 0xffffffff,   0xffffffff,     0         },    /* ブロードキャスト */
};
```


エコーサーバー・コンフィギュレーション定義ファイル

- tinet_app_config.h
- アプリケーションプログラムに依存するパラメータを定義する（tinet/doc/tinet_config.txtを参照）

```
#ifdef SUPPORT_ETHER
#ifdef DHCP_CFG
#define IPV4_ADDR_LOCAL             MAKE_IPV4_ADDR(0,0,0,0)
#define IPV4_ADDR_LOCAL_MASK        MAKE_IPV4_ADDR(0,0,0,0)
#else  /* of  #ifdef DHCP_CFG */
#define IPV4_ADDR_LOCAL             MAKE_IPV4_ADDR(192.168.11.98)
#define IPV4_ADDR_LOCAL_MASK        MAKE_IPV4_ADDR(255.255.255.0)
#define IPV4_ADDR_DEFAULT_GW        MAKE_IPV4_ADDR(192.168.11.1)
#endif /* of #ifdef DHCP_CFG */
/* ルーティング表の静的ルーティングエントリ数 */
#define NUM_STATIC_ROUTE_ENTRY          3
/* 向け直し(ICMP)によるルーティング数。0を指定すると向け直しを無視 */
#define NUM_REDIRECT_ROUTE_ENTRY    0
```


エコーサーバー・TCP通信端点の生成



エコーサーバー・TINETコンフィギュレーション

- tinet_echos4.cfg
- TCP受付口
  - サーバーアプリケーションにのみ必要
  - クライアントからの接続要求を待つ
  - IPアドレスとポート番号を定義
  - 複数のタスクにより共用可能
- TCP受付口生成静的API

```
TCP_CRE_REP(TCP受付口ID,
                {受付口属性, {IPアドレス, ポート番号}})
```

- エコーサーバーでは受付口を一個作成

```
TCP_CRE_REP (TCP_SRV_REPID, {
                0, { IPV4_ADDRANY, REP_PORT } } )
```

自分の持つ全てのIPアドレスに対する接続要求を待ち受ける

7

エコーサーバー・TINETコンフィギュレーション

- tinet_echos4.cfg
- TCP通信端点の生成のための静的API

```
TCP_CRE_CEP(TCP通信端点ID, {
        通信端点属性,
        送信バッファ, 送信バッファサイズ,
        受信バッファ, 受信バッファサイズ,
        コールバック関数
})
```

- エコーサーバーではTCP通信端点を一個作成

```
TCP_CRE_CEP (TCP_ECHO_SRV_CEPID, {
        0,
        tcp_echo_srv_cep_sbuf, TCP_ECHO_SRV_SWBUF_SIZE,
        tcp_echo_srv_cep_rbuf, TCP_ECHO_SRV_RWBUF_SIZE,
        NULL
} )
```

NON_BLOCKINGの場合、コールバック関数を設定

TOPPERS

エコーサーバー・TINETコンフィギュレーション

- 送受信バッファ（TCPウィンドバッファ）
  - アプリケーション側で確保
- echos4.h

  TCP_MSS:IPv4のTCP最大セグメントサイズ

  tinet/netinet/tcp.hで定義

```
#define TCP_SRV_CEP_SBUF_SIZE       (TCP_MSS)
#define TCP_SRV_CEP_RBUF_SIZE       (TCP_MSS)
```

- echos4.c

```
UB tcp_echo_srv_swbuf[TCP_SRV_CEP_SBUF_SIZE];
UB tcp_echo_srv_rwbuf[TCP_SRV_CEP_RBUF_SIZE];
```


TOPPERS

エコーサーバー・ユーザープログラム

- コンフィギュレーションファイル
  - TINETコンフィギュレーションファイル

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

  - タスクを一個生成

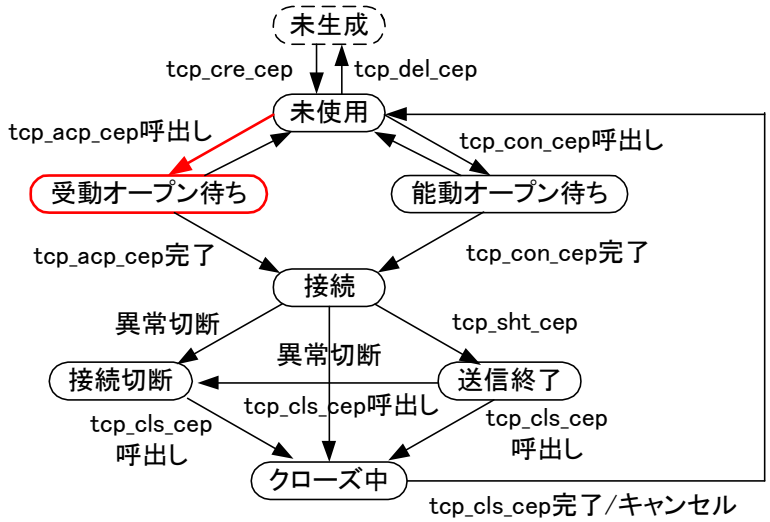
```
CRE_TSK(TCP_ECHO_SRV_TASK,  {
      TA_HLNG|TA_ACT, TCP_ECHO_SRV_CEPID,
      tcp_echo_srv_task,
      TCP_ECHO_SRV_MAIN_PRIORITY,
      TCP_ECHO_SRV_STACK_SIZE, NULL});
```

- TINET標準インクルードファイル

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


TOPPERS

エコーサーバー・受動オープン



TOPPERS

エコーサーバー・受動オープン

- 受動オープンAPI

```
tcp_acp_cep(
        TCP通信端点ID, TCP受付口ID,
        IPアドレス/ポートアドレス構造体,
        タイムアウト)
```

- エコーサーバー

```
T_IPV4EP dst;

tcp_acp_cep(
        (INT)exinf, TCP_ECHO_SRV_REPID,
        &dst,
        TMO_FEVR)
```

NON_BLOCKINGモードの場合はTMO_NBLK

エコーサーバーの受動オープン

- tcp_acp_cepで受動オープンを行う

```
UW      total;
UH      rlen, slen, soff, count;
UB      *rbuf, *sbuf, head, tail;
ID tskid;
ER_UINT         rblen, sblen;
ERerror = E_OK;

get_tid(&tskid);
syslog(LOG_NOTICE, "[TCP ECHO SRV:%d,%d] (copy save API) started.“,
        tskid, (INT)exinf);
while (TRUE) {
    if (tcp_acp_cep((INT)exinf, TCP_ECHO_SRV_REPID, &dst,
            TMO_FEVR) != E_OK) {
        syslog(LOG_NOTICE, "[TCP ECHO SRV ACP] error: %s",
                    itron_strerror(error));
        continue;
    }
    total = count = 0;
    syslog(LOG_NOTICE, "[TCP ECHO SRV ACP] connected from %s:%d",
        ip2str(NULL, &dst.ipaddr), dst.portno);
```

NON_BLOCKINGモードの場合はTMO_NBLK

エコーサーバー・データの受信

- データ受信API

```
tcp_rcv_buf(TCP通信端点ID,
            バッファ, タイムアウト)
```

- 戻り値
  - 受信文字数
  - 0の場合は、通信相手がコネクションを切断
  - 負の場合はエラー
- 受信文字数をerrorに、受信データをrbufに格納

```
ER error = E_OK;
error = tcp_rcv_buf((INT)exinf, (VP *)&rbuf,
                    TMO_FEVR);
```

NON_BLOCKINGモードの場合TMO_NBLK

エコーサーバーの受信処理

- 受信用バッファの解放

```
tcp_rel_buf(TCP通信端点ID,
            データの長さ)
```

- 戻り値
  - E_OKで正常終了
  - 負の値はエラー
- 受信データを引き取り後、バッファを解放する

```
ER error = E_OK;
error = tcp_rcv_buf((INT)exinf, rblen);
```


エコーサーバーの受信

- tcp_rcv_bufで受信バッファを取り出し、データを取り出し後、tcp_rel_bufでバッファを解放する

```
        total = count = 0;
        syslog(LOG_NOTICE, "[TCP ECHO SRV ACP] connected form %s:%d”..);
        while (TRUE) {
            if ((rblen = tcp_rcv_buf((INT)exinf, (VP *)&rbuf, TMO_FEVR)) <= 0) {
                if (rblen != E_OK)
                    syslog(LOG_NOTICE, "[TCP ECHO SRV RCV] error: %s”, ….);
                break;
            }
            rlen   = (UH)rblen;
            total += (UW)rblen;
            head = *rbuf;
            tail = *(rbuf + rlen - 1);
            count ++;
....
            if ((error = tcp_rel_buf((INT)exinf, rblen)) < 0) {
              syslog(LOG_NOTICE, "[TCP ECHO SRV REL] error: %s", …);
              break;

        }
```


エコーサーバー・データの送信

- 送信バッファ取得API

```
tcp_get_buf(TCP通信端点ID,
        バッファ, タイムアウト)
```

- 戻り値
  - バッファ文字数
  - 負の場合はエラー
- バッファ送信API

```
tcp_snd_buf(TCP通信端点ID, 送信文字数)
```

- 戻り値
  - 送信結果、E_OK以外はエラー

エコーサーバー・データの送信

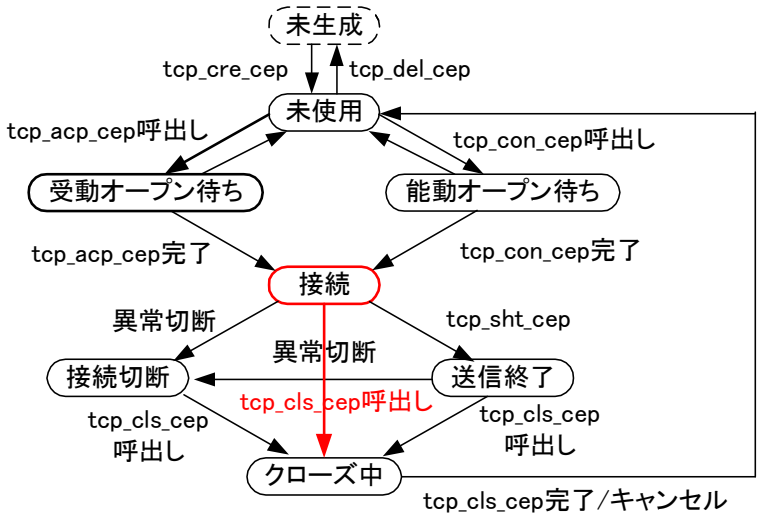
- sbufにバッファを取得、バッファに送信データを詰めて送信要求を行う

```
    UH rlen, slen, soff, count;
    UB *sbuf;
    ER  error = E_OK;
....
    soff = 0;
    while(rlen > 0){
        if ((slen = tcp_get_buf((INT)exinf, (VP*)&sbuf, TMO_FEVR)) < 0) {
            syslog(LOG_NOTICE, "[TCP ECHO SRV GET] error:%s",  .....);
            goto err_fin;
        }
        slen = rlen < (UH)sblen ? rlen : (UH)sblen;
        memcpy(sbuf, rbuf + soff, slen);
        if((error = tcp_snd_buf((INT)exinf, slen) != E_OK) {
            syslog(LOG_NOTICE, "[TCP ECHO SRV SND] error:%s", …);
            goto err_fin;
        }
        rlen -= slen;
        soff += slen;
    }
```

NON_BLOCKINGモードの場合、TMO_NBLK

TOPPERS

エコーサーバー・コネクションの切断



TOPPERS

エコーサーバー：tcp_sht_cepとtcp_cls_cepの違い

- tcp_cls_cep
  - 通信端点をクローズする。これ以降、通信端点を使用できない
- tcp_sht_cep
  - データ送信を終了する。これ以降、データ送信は不可能だがデータ受信は可能。相手に終了したことを通知するため
- 通信の終了のためのクライアント・サーバー間の協調
  - サーバーはクライアントの要求が終了したか判定できないのでコネクションを切れない
  - クライアントは要求が終了したことは分かるが、サーバーからのデータがすべて到着したか分からない
  - クライアントがサーバーへの要求の終了を通知するためにtcp_sht_cepを使用する

エコーサーバー・コネクションの切断

- データ送信の終了API

```
tcp_sht_cep(TCP通信端点ID)
```

- コネクションの切断

```
tcp_cls_cep(TCP通信端点ID, タイムアウト)
```

- エコーサーバー

```
tcp_sht_cep((INT)exinf);
tcp_cls_cep((INT)exinf, TMO_NBLK);
```

NON_BLOCKINGモードの場合、TMO_NBLKで終了後バッファの解放待ちを行う

エコーサーバー・コネクションの切断

- 切断確認後、tcp_sht_cep、tcp_cls_cepを行い切断する

```
    ER error = E_OK;
....
    err_fin:

        if ((error = tcp_sht_cep((INT)exinf)) != E_OK)
            syslog(LOG_NOTICE, "[TCP ECHO SRV SHT] error: %s",
                        itron_strerror(error));

        if ((error = tcp_cls_cep((INT)exinf, TMO_FEVR)) != E_OK)
            syslog(LOG_NOTICE, "[TCP ECHO SRV CLS] error: %s",
                        itron_strerror(error));

        syslog(LOG_NOTICE, "[TCP ECHO SRV FIN] finished, total cnt:
                %d, len: %d", count, total);
    }
}
```


DHCP+ECHOサーバの実装確認

1. ECHOサーバ
2. DHCP

エコーサーバーの改造

- エコーサーバーを改造して、固定IPをDHCPを使用してIPアドレスを自動取得するように改造します
- エコーサーバータスクの起動時、UPDポート番号“68”のDHCP手順で、DHCPサーバーからIPアドレスを取り出す手順を追加する
- 処理的には、追加のソースファイルdhcp.c中のdhcp_open関数を実行して、正常終了すればIPv4のIPアドレスが取得でき、そのIPアドレスを用いてエコーサーバーのIPアドレスとして使用する
- MACアドレスをボードごとに変更する必要がある
  - MACアドレスはユーザープログラムで設定している

DHCPプロトコル

- DHCPは以下の手順でIPアドレスを取得します
  - DHCPサーバーに対してマルチキャストでDISCOVERコマンドを送信
  - DHCPサーバーからOFFER応答がある
  - OFFER応答のあったDHCPサーバーに対してREQUEST送信を行う
  - DHCPサーバーから確認のPACK応答があればIPアドレスが確定、ダメな場合はPNAK応答がある
- PACKまでが完了すれば、OFFER応答中のユーザIPアドレスが、IPアドレスとして確定する

DHCP対応・メイクファイル

- メイクファイルに対して以下の変更が必要
  - DHCP_CFGを有効にプロトコルスタックをDHCPモードに
  - dhcp.cを取り込む
  - SUPPORT_UDPを有効にする

```
#  共通コンパイルオプションの定義
#
COPTS := $(COPTS)  –DDHCP_CFG
CDFES := $(CDEFS)
....
ifdef USE_CXX
        UTASK_CXXOBJS = $(UNAME).o
        UTASK_COBJS = dhcp.o
else
        UTASK_COBJS = $(UAME).o dhcp.o
endif
....
SUPPORT_TCP = true
SUPPORT_UDP = true
```


DHCP対応・TINETコンフィギュレーション

- tinet_echos4.cfgの修正
- UDP通信端点の定義

```
UDP_CRE_CEP(UDP通信端点ID, {
      UDP通信端点属性,
      {自分のIPアドレス, ポート番号},
      コールバック関数
})
```

- UDPポート番号”68”の通信端点を追加

```
UDP_CRE_CEP(UDP_CLIENT_CEPID,  {
      {IPV4_ADDRANY,
      68},
      NULL
});
```


エコーサーバーアプリケーション

- echos.cの修正
  - dhcp.hをインクルード
  - dhcp_open()を実行

```
#include “dhcp.h”
#include “echos.h”
……
 syslog(LOG_NOTICE, “[TCP ECHO SRV (NBLK):%d,%d]….);
 /*
  * 自分のMACアドレス読み込み
  */
 ic = IF_ETHER_NIC_GET_SOFTC();
 error = dhcp_open(ic);
 if(error != E_OK){
       syslog(LOG_ERROR, “DHCP ERROR: %s !”, itron_strerror(error));
       slp_tsk();
 }
 while(TRUE) {
….
```


エコーサーバーのMACアドレスの変更

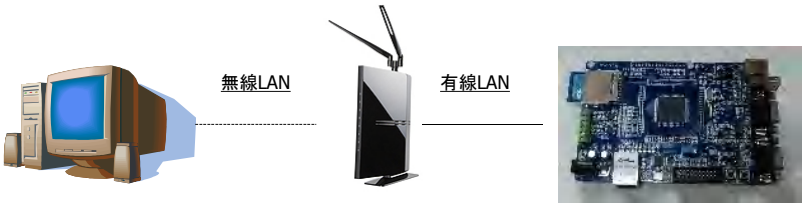
- ボードごとにMACアドレスを修正する必要がある
- MACアドレス設定部の6バイト目の設定を座席番号に変更してください

6バイト目に座席番号に設定すれば、受講者のボード毎に個別MACアドレスとなる

```
……
/*
 * MACアドレスの設定
 */
void
macaddr_init(VP_INT exinf)
{
   static const UB macaddr[LPC_ETHER_ADDR_LEN] = {
                0x00, 0x00, 0x12, 0xCB, 0x88, 0xXX};
   memcpy(lpc23xx_macaddr, macaddr, LPC_ETHER_ADDR_LEN);
}
…….
```

実習：送受信パケットの確認を行う

- DHCPを実行します
- DHCPが成功した後、log modeコマンドにてLOG INFO表示を行い、パソコンからPINGを行って、送受信パケットを確認してください
- ARP⇒PINGのパケットを確認
- ECHOサーバ通信パケットを確認

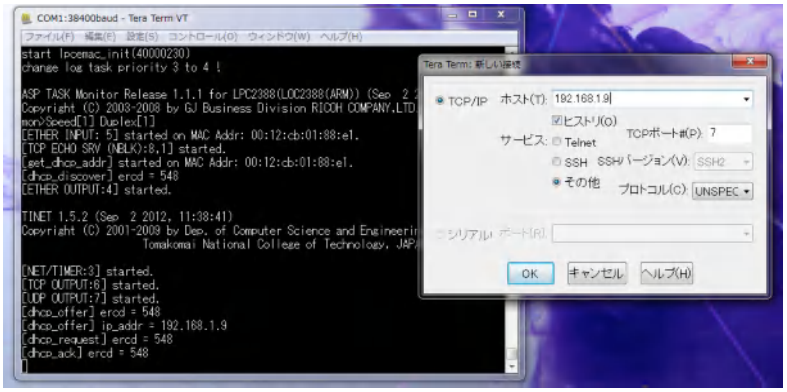


実習：送受信パケットを確認する1

- DHCPを実行する
  - DHCPをビルドして、Flash Magicにてボードに設定
  - TeraTermを立ち上げ、ボードを起動する
  - TeraTerm上にIPアドレスの確定と表示を確認
  - TeraTermにてlog mode 1 6としてLOG_INFOを表示
- パソコンからボードにpingを行いパケットを確認
  - Cygwin[DOS窓]は管理者モードで起動
  - ipconfigコマンドにてパソコンのLAN状態を確認
  - ping xx.xx..xx.xx(ボードのIPアドレス)にて通信確認
  - 通信パケットを確認する

実習：送受信パケットを表示する2

- TeraTerm⇒新しい接続、その他（ポート7）、ホストのボードのIPアドレスをセットします
- 開いたターミナルに入力、パケット表示を確認してください



TOPPERS

プラットフォーム作成

1. APIの解説
2. プラットフォーム構造図
3. ワークベースの作成

TOPPERS

概要

- ここまでの作業で作成したデバイスドライバ、ミドルウェアをまとめて解説します
- プラットフォームとしての構造図を示します
- APIを明確にします
- ワークベースを作成しプラットフォームを実装します

タイマーAPI

- 日時の取得設定用にRTC用のデバイスドライバを用意し、APIとして以下の4つのPOSIX互換関数を用意する
  - mktime　　tm構造体からtime_tへの変換
  - gmtime_r　　timer_tからtm構造体へ構造体を指定して変換
  - gmtime　　timer_tからtm構造体(static)へ変換
  - time　　現在時刻をtime_t型で取り出す
- mktimeはRTCで作成したclock.c中にある
- その他の関数はtime.cに用意する

timer_tは秒数を表すサインなしの整数で、1970年1月1日からの経過時間で日時を表します。UNIXで使用されます。

ファイルシステムAPI

- ファイルシステム関数としてC89、C99、POSIXに準拠し、よく使われる関数を用意した
  - fopen,fclose,fread,fwrite,fputc,fputs,putc,fgetc,fgets,fflush
  - rename,remove
  - open,close,read,write,stat,fstat,lstat
  - access,mkdir,rmdir,chmod,opendir,closedir,readdir,statfs
- ファイルシステム関数は標準入出力に対応可能な構造とした
  - telnetのリダイレクション機能で使用する

ネットワークAPI

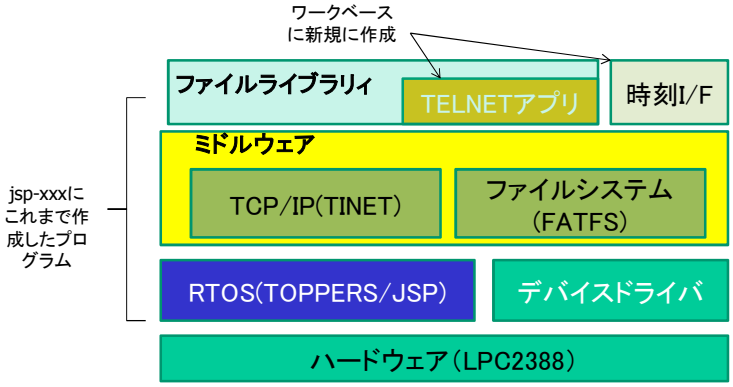
- 今回作成するtelnetアプリではネットワーク用のコマンドはサポートしない
  - ネットワーク用のコマンドを作成するにはBSDソケットの方が、簡易的にアプリケーション構築が可能
- ターゲットボードにメモリ制約があり（非機能要件）、ネットワークはtelnetサーバーのみ対応である
- メモリ効率が良く、ポーティングしやすいTINET（ITRON TCP/IP仕様）を選択した

プラットフォーム作成

1. APIの解説
2. プラットフォーム構造図
3. ワークベースの作成

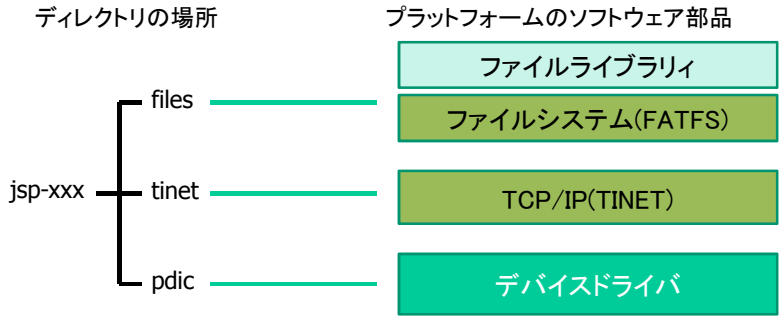
プラットフォームの構造図

- プラットフォームの概要を明確にするために構造図を作成します
- アプリケーションからはファイルライブラリィや時刻I/FがAPIとなり、APIに従ってアプリケーションを作成します



JSPディレクトリ上の実装関係

- JSPディレクトリとプラットフォームで使用するソフトウェア部品の関係を以下に示します
- TINETアプリと時刻I/F以外はjsp-xxxディレクトリにすでに構築済みとなります



TELNETアプリとファイルライブラリィの関係

- TELNETアプリはポート番号23のネットワークアプリケーションでTELNET用の特殊なコマンドを持つ以外はエコーサーバーのようにパソコンとのデータ交換が主な役目です
- ファイルライブラリィは標準入出力関数を用いてファイルアクセスを行いますが、これらをTELNET用のデータ交換に接続すれば、標準入出力関数を用いてパソコンとの通信を行うことができます

printf文	stdout設定をTELNETの送信にすればパソコンにデータ送信を行える
fgetc文	stdin設定をTELNETの受信にすればパソコンからのデータ受信を行える

ファイルライブラリィの標準入出力設定

- TOPPERS/JSPファイルライブラリィの標準入出力設定は、標準ではシリアルポート(CONSOLE_PORT)となっており、シリアルデバイスからのデータ入出力を行う
- 通常はタスクモニタのコマンドの入出力に使用している
- これらの設定(標準入出力設定)を行っているのは、以下のファイルです
  - monitor/stdio.h
  - monitor/stdio/stddev.c
- この設定を変更すれば、ネットワーク通信やファイルのアクセスを標準入出力に設定することができます

FILE構造体

- ファイルアクセスに使用するFILE構造体はstdio.hで定義を行っている
- 標準入出力に用いる3つのFILE構造体のポインタもstdio.hで定義を行っている
  - stdin　標準入力FILE構造体ポインタ
  - stdout　標準出力FILE構造体ポインタ
  - stderr　標準エラー出力FILE構造体ポインタ
- FILE構造体自体はfopen関数等で生成されるものと同等であるため、この構造体をネットワーク通信用に作成すれば、TELNETのデータ交換を標準入出力関数で行うことができる

FILE構造体の内容

- FILE構造体はデバイスやファイルを指定する属性と、それをアクセスする関数ポインタで構成されている
- デバイスの指定やアクセス関数を変更すれば、別のデバイスに対する入出力を行うことができる

```
typedef struct _msFILE {
    int    _flags;                  /* flags, below; this FILE is free if 0 */
    int    _file;                   /* fileno, if Unix desciptor, else -1 */
    int    (*_func_in)(void*);      /* pointer to a character input function */
                                    /* pointer string input function */
    int    (*_func_ins)(void *, unsignded int, char *);
    int    (*_func_out)(void *, int); /* pointer to a charcater output function */
                                    /* pointer to string output function */
    int    (*_func_outs)(void *, unsigned int, char *);
                                    /* pointer to flush output function */
    int    (*_func_flush)(struct __msFILE *);
    void  *_dev;                    /* pointer to local device structure */
};
```


TELNET用標準入出力関数

- TELNETアプリの入出関数をFILE構造体で使用する入出関数に準拠し、stdin、stdout、stderrに割り付ければ標準入出力を用いてTELNETのデータ交換を行える
- TELNET通信用の特別なAPIを作成する必要はない

File関数	Telnet関数	引数	機能
_func_in	int netlocal_getc	FILE * st	1文字入力
_func_ins	int netlocal_gets	FILE *,st unsigned int len, char *s	文字列入力
_func_out	int netlocal_putc	FILE *, INT ch	1文字出力
_func_outs	int netlocal_puts	FILE *st, unsigned int len, char *s	文字列出力
_func_flush	int netlocal_flush	FILE *st	出力フラッシュ

プラットフォーム作成

1. APIの解説
2. プラットフォーム構造図
3. ワークベースの作成

ワークベースの作成

- アプリケーションとプラットフォームを作成するワークベースを作成する
  - ワークベースはRTOSと並列化するためjsp-xxxと同列のディレクトリ上に作成する
- ワークベース上に不足の以下のプラットフォーム部品を作成する
  - TELNETアプリ
  - 時刻I/F
- TELNETアプリはDHCP対応のECHOサーバーを用いて作成する（ノンブロッキングモードのみサポート）
- 時刻I/Fは以下のソースファイルに関数として用意する
  - time.c

TELNETアプリを作成

- jsp-xxx/OBJ/LPC2388/DHCP4をjsp-xxxと同列のディレクトリにplatform_netの名称でコピーします
- これを改造して、TELNETサーバを作成します
- ECHOサーバーの定数、変数、関数名がtcp_echo_xxとなっているため、まず、これらをtcp_telnet_xxxに修正します。また、ファイル名をehcos4からtelnetに修正します
- 名称の修正が終わったら、メインタスクをtelnet用のタスクと分離するためconfig.cfg/config.h/config.cを追加します
- DHCPの実行やMACアドレスの初期化もメインルーチン（config.c）に移動します（他のデバイスの取り込みをメインルーチンで行うため）

PLATFORM_NETのファイル構成

- DHCPを改造前と後の比較を行う

DHCPディレクトリ

ソースファイル
dhcp.c
dhcp.h
Makefile
route_cfg.c
echos4.c
echos4.cfg
echos4.h
tinet_app_config.h
tinet_echos4.cfg

platform_netディレクトリ

ソースファイル	内容	対応
dhcp.c	dhcpプロトコル	変更なし
dhcp.h	dhcpインクルード	変更なし
Makefile	メイクファイル	修正
route_cfg.c	ルート経路図定義	変更なし
telnet.c	telnetサーバー	修正
telnet.cfg	telnetRTOS静的API	修正
telnet.h	telnetサーバ定義	修正
tinet_app_config.h	tinetアプリ定義	変更なし
tinet_config.cfg	tinet用の静的API設定	修正
config.c	メインタスク定義	新規作成
config.cfg	全体の静的API定義	新規作成
config.h	メインのインクルード	新規作成

名称の変更

- 機能上は問題ありませんが、エコーサーバープログラムは関数や定数名がECHOまたはECHOSとなっています
- これをTELNETとなるようにソースファイルを修正しました

元の名称	変更した名称	対象のファイル
_ECHOS_H_	_TELNET_H_	telnet.h
TCP_ECHO_SRV_TASK	TCP_TELNET_SRV_TASK	telnet.cfg
TCP_ECHO_SRV_CEPID	TCP_TELNET_SRV_CEPID	telnet.cfg,tinet_telnet.cfg
TCP_ECHO_SRV_MAIN_PRIORITY	TCP_TELNET_SRV_MAIN_PRIORITY	telnet.h,telnet.cfg
TCP_ECHO_SRV_STACK_SIZE	TCP_TELNET_SRV_STACK_SIZE	telnet.h,telnet.cfg
SEM_TCP_ECHO_SRV_NBLK_READY	SEM_TCP_TELNET_SRV_NBLK_READY	telnet.cfg,telnet.c
TCP_ECHO_SRV_SWBUF_SIZE	TCP_TELNET_SRV_SWBUF_SIZE	telnet.c,telnet.h,tinet_telnet.cfg
TCP_ECHO_SRV_RWBUF_SIZE	TCP_TELNET_SRV_RWBUF_SIZE	telnet.c,telnet.h,tinet_telnet.cfg
tcp_echo_srv_swbuf	tcp_telnet_srv_swbuf	telnet.c,telnet.h,tinet_telnet.cfg
tcp_echo_srv_rwbuf	tcp_telnet_srv_rwbuf	telnet.c,telnet.h,tinet_telnet.cfg
tcp_echo_srv_task	tcp_telnet_srv_task	telnet.cfg,telnet.h,telnet.c
callback_nblk_tcp_echo_srv	callback_nblk_tcp_telnet_srv	telnet.c,telnet.h,tinet_telnet.cfg
TCP_ECHO_SRV_REPID	TCP_TELNET_SRV_REPID	telnet.c,tinet_telnet.cfg

サーバー部の変更

- サーバー部に以下の改造を加えました
  - USE_COPYSAVE_APIを有効に
    - こちらの方が標準入出力の対応が容易なため
  - コールバック関数に送受信専用セマフォを追加
    - エコーサーバーではSEM_ECHO_SRV_NBLK_READYですべてのイベント通信を行っていたが、送信用にSEM_TELNET_SRV_NSND_READYを受信用にSEM_TELNET_SRV_NRCV_READYセマフォを追加
  - コネクションの状態が他のタスクから参照できるように修正
  - TELNETオプションに対応
    - TELNETで使用するオプションキャラクタを受信時解析して、ネグレクトするように改造した

TELNETオプション

- TELNETには0xF0から0xFFのオプションキャラクタがあり、ここに機能を持つ

ラベル	値	意味
TELNET_OPT_SE	0xF0	二次交渉パラメータの終了
TELNET_OPT_NOP	0XF1	オペレーションなし。受信した場合これを無視する
TELNET_OPT_DM	0xF2	データ削除。リセット
TELNET_OPT_BRK	0xF3	ブレーク
TELNET_OPT_IP	0xF4	操作の一時中断・割込み・停止
TELNET_OPT_AO	0xF5	出力を抑制する
TELNET_OPT_AYT	0xF6	相手が動作してるかどうかを確認する
TELNET_OPT_EC	0xF7	最後の文字を消去する
TELNET_OPT_EL	0xF8	最後の行をすべて削除する
TELNET_OPT_GA	0xF9	送信するように受信側に促す
TELNET_OPT_SB	0xFA	二次交渉の開始
TELNET_OPT_WILL	0xFB	オプション希望
TELNET_OPT_WONT	0xFC	オプション拒絶
TELNET_OPT_DO	0xFD	オプション実行要求
TELNET_OPT_DONT	0xFE	オプション使用中止
TELNET_OPT_IAC	0xFF	telnetエスケープシーケンス

メインタスクを新たに作成

- エコーサーバーではtcp_echo_srv_taskにて、総ての設定を行っていたが、メインのタスク（main_task）を用意してそちらで全体の設定を行い、tcp_echo_srv_taskはTELNET通信専用になるように改造した
- この改造用に以下をメインソースとして追加した
  - config.cfg　全体のコンフィギュレーション
telnet.cfgをインクルードする
  - config.h　メインのインクルードファイル
  - config.c　main_taskを含むメインのソース

PLATFORM_NETからPLATFORMへの改造

- PLATFORM_NETはTELNETが動作するアプリケーションとして作成しました
- これをプラットフォームとするために以下の改造が必要です
  - TELNETアプリの標準入出力対応
  - ファイルや時刻のミドルウェアやデバイスドライバの取り込み
- これを行うために、PLATFORM_NETをさらに改造し、PLATFORMをワークベースに作成します
- platform_netと同列にワークベースとしてplatformディレクトリを作成します

PLATFORMへ改造

- PLATFORM_NETをベースに他のデバイスを追加

platform_netディレクトリ

ソースファイル
dhcp.c
dhcp.h
Makefile
route_cfg.c
telnet.c
telnet.cfg
telnet.h
tinet_app_config.h
tinet_config.cfg
config.c
config.cfg
config.h

platformディレクトリ

ソースファイル	内容	対応
dhcp.c	dhcpプロトコル	変更なし
dhcp.h	dhcpインクルード	変更なし
Makefile	メイクファイル	修正
route_cfg.c	ルート経路図定義	変更なし
telnet.c	telnetサーバー	修正
telnet.cfg	telnetRTOS静的API	修正
telnet.h	telnetサーバ定義	削除
tinet_app_config.h	tinetアプリ定義	変更なし
tinet_config.cfg	tinet用の静的API設定	変更なし
config.c	メインタスク定義	修正
config.cfg	全体の静的API定義	修正
config.h	メインのインクルード	修正
clock.c	時刻表示pipeコマンド	追加
time.c	時刻関数	追加

TELNET入出力関数

- TELNETサーバーを標準入出力に対応するように修正します
- TELNET標準入出力関数を作成し、この関数の設定がtcp_telnet_srv_taskと通信するように設定します
- TELNETサーバーにコネクションした場合、標準入出力関数がTELNET標準入出力関数に入れ替えられ、ディスコネクトした場合、元の標準入出力関数に戻されるように修正します
- この修正は以下のソースファイルを修正しました
  - telnet.cfg
  - telnet.h　削除し、config.hに含めた
  - telnet.c

デバイスの追加

- ETHERNET以外の以下のデバイスを追加
  - RTC
  - MCI/DMA
  - SDカードファイルシステム
- この拡張を以下のファイルに対して行いました
  - Makefile
  - config.cfg
  - config.h
  - config.c
- 不足のシステム関数を追加しました
  - time.c　gmtime_r関数、time関数を追加
  - clock.c　時刻の設定参照(RTCの章を参照)

アプリケーションリンク部の設定

- コールバック関数applicationを用意し、設定されている場合、コネクト後、この関数を呼び出すように設定する

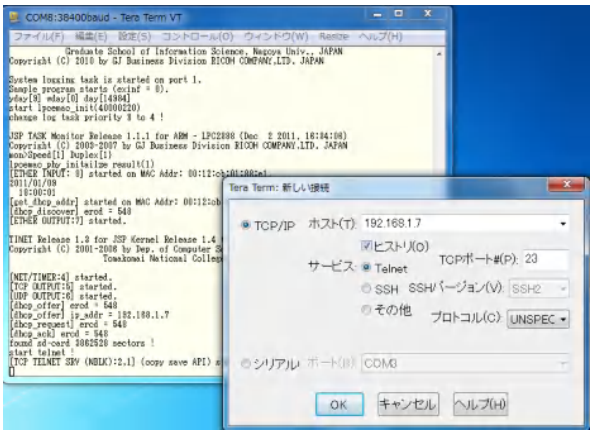
```
int  (*application)(INT mode);

....
void main_task(VP exinf)
{
    INT ch;
    ....
    tcp_telnet_srv_init(callback_telnet);
    while(ch != EOF){
        if(tcp_telnet_connect()){    /* connect */
            if(application != NULL)
                ch = application(0);
            else{
                putchar(ch = getchar());
                if(ch == CHAR_CR)
                    putchar(CHAR_LF);
            }
        }
        else                         /* disconnect */
            dly_tsk(100);
    }
}
```

applicationが設定されていない場合、受信データをエコーするように設定

TOPPERS

PLATFORMの確認実習

- PLATFORMをビルドして、TELNETで接続してECHOサーバーのように動作することを確認しましょう



TOPPERS

仮想端末アプリの構築

1. システム構想図
2. TELNET標準入出力
3. TELNETシェル
4. コマンドアプリ

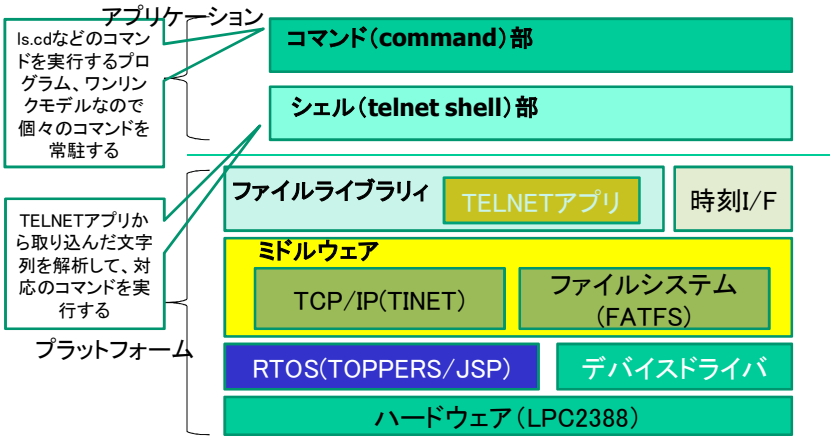
仮想アプリ端末アプリの機能

- 想定のアプリ：仮想端末を用いUNIXライクなコマンドを使ってSDカードの管理を行う
- ファイル管理用に必要なコマンドを列記する

コマンド	コマンドの処理内容
date	日時の設定、表示を行う
cd	カレントディレクトリを変更する
ls	ディレクトリ上のファイルリストを表示する
cat	ファイルをテキスト表示する
echo	以下の入力データをエコー表示する
mkdir	ディレクトリを作成する
rmdir	ディレクトリを削除する
rm	ファイルを削除する

システムブロック図

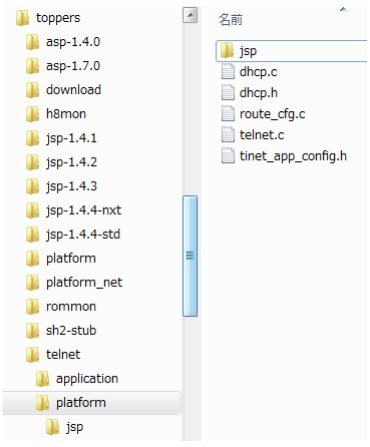
- 作成する組込みシステムのブロック図を作成する
- プラットフォーム上にアプリケーションを作成する



TOPPERS

ワークスペースの構成

- アプリケーションを含んだtelnetワークスペースを作成します
- ワークスペースはプラットフォーム部(platform)とアプリケーション部(application)を分離できるように別のディレクトリで作成します
- platformには、プラットフォーム共通ソースを直下にプログラムを配置し、RTOS依存部はサブディレクトリ(jsp)以下に配置します



TOPPERS

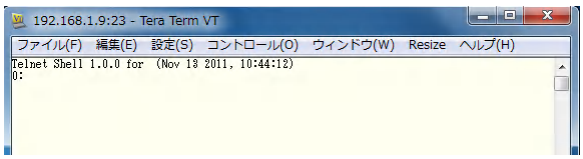
アプリケーション・ディレクトリの構成

- アプリケーションはAPIに従って作成し、APIに従う限りはプラットフォーム部に変更があっても、変更なく使用できるように設計します
- プラットフォーム部の情報はconfig.hを参照します

ファイル	内容
shell.h	シェル部のインクルードファイル
shell.c	シェル部のソースファイル
command.c	コマンドプログラム
clock.c	RTCの設定コマンドプログラム
time.c	時刻関数プログラム
devtest.c	ファイル、RTCのテストプログラム

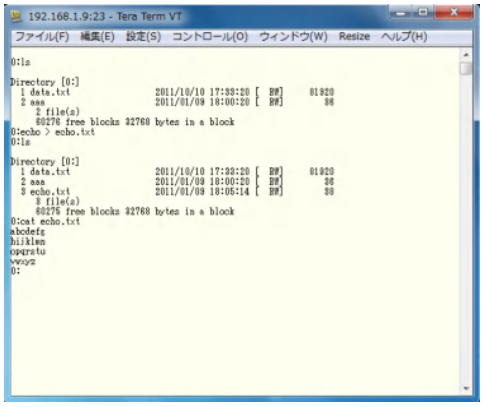
TELNET SHELL

- TELNET SHELLを以下の機能を持つように作成する
  - 最初の起動時バナー表示とプロンプトを表示する
  - 標準入力から文字列を読み取り、コマンド名と引数を解析し、対応のコマンドを引数付きで実行する
  - リダイレクション設定(>、>>)があった場合、標準出力をリダイレクション先のファイルに割り当てる
  - リダイレクション時のコマンド実行終了でファイルをCLOSEし、標準出力をもとの設定に戻す
  - コマンドの実行後、プロンプトを表示する



リダイレクション機能

- リダイレクション機能は、表示データをファイルに書き込む機能です
- echo > echo.txt(return)にて、エコー入力したデータをecho.txtに書き込みます
- echoコマンドの終了はCntl-Zです
- 作成されたecho.txtをcatコマンドで表示させると入力した文字列が表示されます



TOPPERS

コマンド実装部

- このシステムはワンリンクモデルなので、総てのコマンドがROM上に常駐します
- 個々のコマンドは関数として実行します
- コマンドの文字列と関数ポインタを対にしたテーブルをTELNET SHELLに渡すことによりコマンドパースします

```
const command_type command_table[] = {
        {“date”,        cmd_date},
        {“cd”,          cmd_cd},
        {“ls”,          cmd_ls},
        {“cat”,         cmd_cat},
        {“echo”,        cmd_echo},
        {“mkdir”,       cmd_mkdir},
        {“rmdir”,       cmd_rmdir},
        {“rm”,          cmd_rm},
        {“mon”,         cmd_mon},
        {“help”,        cmd_help},
        {“cmd”,         cmd},
        {0,             0}
};
```

コマンド名

対応する関数名

テーブルの終了を表す

TOPPERS

仮想端末アプリの構築

1. システム構想図
2. TELNET標準入出力
3. TELNETシェル
4. コマンドアプリ

標準入出力

- 標準入出力では、_iob[]の3つのFILE構造体のインスタンスが、標準入力、標準出力、エラー出力のデバイスとなります
- 通常は_iob[]には、タスクモニタ出力用にシリアルデバイス関数が割り当てられています
- TELNETアプリで通信するデータを入出力する関数を_iob[]に入れ替えれば、標準入出力を用いてTELNETのコマンド通信が可能となります
- 標準出力をファイル関数に入れ替えればファイル書き込みが可能となります⇒リダイレクション機能

```
extern FILE _iob[];
#define  stdin   (&_iob[0])          /* FILE input */
#define  stdout  (&_iob[1])          /* FILE output */
#define  stderr  (&_iob[2])          /* FILE error input output */
```


通常の標準入出力設定

- 通常の設定では、RTOSで使用するシリアルデバイスの入出力が設定されている

```
/*
 *  標準入出力ストリーム初期化関数
 */
int _set_stdio(int no)
{
        if(no >= 0 && no < 3){
                _iob[no]._file      = -1;
                _iob[no]._func_in   = local_getc;
                _iob[no]._func_ins  = local_gets;
                _iob[no]._func_out  = local_putc;
                _iob[no]._func_outs = local_puts;
                _iob[no]._func_flush = local_flush;
                _iob[no]._dev       = 0;
                return no;
        }
        else
                return -1;
}
```

_iob[]に入出力関数を設定する関数(_set_stdio)

```
/*
 *  モニタの文字列出力文
 */
static int
local_puts(FILE *st, unsigned int len, char *s)
{
        return serial_wri_dat(*stdport, s, len);
}
```

文字列出力はシリアルデバイス出力関数

TOPPERS

TELNETアプリのデータ管理

- TELNETアプリでは、送信キューバッファ（sbuffer）、受信キューバッファ（rbuffer）をキュー構造体により管理し、受信データはrbufferに転送し、sbufferにデータがある場合送信する構造となっている
- 入力関数はrbufferからデータを読み取り、出力関数sbufferにデータを書き込むように設計し、これを_iob[]に登録すれば、TELNET通信を標準入出力に割り当てられる

```
typedef struct netqueue {
        UH   head;                  /* netqueue head */
        UH   tail;                  /* netqueue tail */
        UH   cnt;
        UH   size;                  /* バッファサイズ */
        UB   *pbuffer;              /* バッファエリア */
} NETQUEUE;
typedef struct netdev {
        NETQUEUE *sndq;
        NETQUEUE *rcvq;
} NETDEV;
```


TOPPERS

TELNET入出力関数の設定

- TELNETアプリの初期化時、tcp_telnet_srv_init関数にて送受信用のキューバッファ(sndqueue、rcvqueue)を作成しtelnetdevに登録します
- TELNETのコネクションが確立した時点でtcp_telnet_srv_file関数にて_iobへのポインタを引数に設定し入出力関数とtelnetdevへのポインタを各_iobに設定します

```
static NETQUEUE sndqueue;
static NETQUEUE rcvqueue;
static NETDEV   telnetdev;

void
tcp_telnet_srv_init(void (*func)(INT mode))
{
        NETQUEUE *sndq = &sndqueue;
        NETQUEUE *rcvq = &rcvqueue;

        callback_telnet = func;
          …
        telnetdev.sndq = sndq;
        telnetdev.rcvq = rcvq;

        act_tsk(TCP_TELNET_SRV_TASK);
}

/*
 *  標準入出力設定
 */
INT
tcp_telnet_srv_set_file(void *pt)
{
        FILE *st = pt;

        st->_file = -1;
        st->_func_in    = netlocal_getc;
        st->_func_ins   = netlocal_gets;
        st->_func_out   = netlocal_putc;
        st->_func_outs  = netlocal_puts;
        st->_func_flush = netlocal_flush;
        st->_dev        = (void *)&telnetdev;
        return 1;
}
```


TELNETアプリでの文字出力の例

- 文字列出力の例としてnetlocal_puts関数を示します
- netlocal_puts関数は_dev⇒telnetdev⇒sndqueueを用いてsbufferに送信データを書き込みます
- sbufferのデータはTELNETアプリがホストに送信を行います

```
static INT
setqueue(NETQUEUE *que, UB *buf, INT len)
{
    INT slen;

    syscall(wai_sem(SEM_TCP_TELNET_QUEUE_LOCK));
    for(slen = 0 ; len > 0 && que->cnt < que->size ; len--, slen++){
        que->pbuffer[que->tail++] = *buf++;
        que->cnt++;
        if(que->tail >= que->size)
            que->tail = 0;
    }
    syscall(sig_sem(SEM_TCP_TELNET_QUEUE_LOCK));
    return slen;
}

/*
 *  TELNETサーブの文字列出力文
 */
static int
netlocal_puts(FILE *st, unsigned int len, char *s)
{
    NETQUEUE *que = ((NETDEV*)st->_dev)->sndq;
    int cnt, slen;

    for(cnt = 0 ; len > 0 ;){
        slen = setqueue(que, s+cnt, len);
        len -= slen;
        cnt += slen;
        if(len > 0)
            dly_tsk(10);
    }
    return cnt;
}
```

仮想端末アプリの構築

1. システム構想図
2. TELNET標準入出力
3. TELNETシェル
4. コマンドアプリ

ディスパッチャー

- TELNETの通信が確立するとアプリケーションとしてdispatcherを呼び出します。最初の呼び出しではfirst_callがTRUEなのでバナー表示を行います
- comlenはコマンドの文字数を表し、最初の呼び出しでは-1となっています
- 標準入力から文字を取得しcmdbufに蓄積します。comlenをインクリメントします
- LFを受信した時点で文字列を引数にしてtshを呼び出します

```
/*
 *  COMMAND DISPATHER
 */
INT dispatcher(INT mode)
{
    INT ch = 0;
    INT len, result;

    if(first_call){
        printf(banner,
        (TSHELL_PRVER >> 12) & 0x0f,
        (TSHELL_PRVER >> 4) & 0xff,
        TSHELL_PRVER & 0x0f);
        prompt();
        first_call = FALSE;
    }
    ch = getchar();
    if(ch == '¥b' || ch == 127){
        if(comlen > 0){
            comlen--;
            printf("¥b ¥b");
        }
    }
    else if(ch == CHAR_CR){
        putchar(CHAR_LF);
        combuf[comlen] = 0;
        len = conlen;
        comlen = 0;
        result = tsh(combuf, len, mode);
    }
    else if(ch > 0 && ch != CHAR_LF){
        combuf[comlen++] = ch;
        putchar(ch);
    }
    return ch;
}
```


TELNETシェルの処理：引数リストの作成

- tshでは文字列を引数列に分解します
- 引数列はC言語のmain関数で渡されるargc（引数の数）とargv（各引数文字列へのポインタ：ここではargb）の形です
- 第一引数がコマンド名文字列へのポインタとなります

```
/*
 *  TELNET SHELL
 */
INT tsh(char *buf, INT len, mode)
{
    INT  argc, no;
    char *argb[16];
    FILE *fd;
    INT  cno = 0;
    INT  result = 0;
    FILE save_file;
    char rediectpath[256];

    /*
     *  文字列のチェック
     */
    if(len == 0 || buf[0] <= ' '){
        if(mode == 0) /* normal dispatch */
            prompt();
        return 1;
    }
    /*
     *  引数の設定
     */
    for(no = argc = 0 ; argc < 16 && no < len ; no++){
        if(test_next(buf[no])){
            buf[no] = 0;
            cno = 0;
        }
        else if(no < len){
            if(cno == 0){
                argb[argc++] = &buf[no];
                if(buf[no] == '"')
                    comin = TRUE;
            }
            cno++;
        }
    }
    buf[no] = 0;
```


TELNETシェルの処理：リダイレクション

- コマンド名以外の引数列を検索し、リダイレクションマークの有無を確認します
- リダイレクションマークがある場合、ファイル名を探し、書き込みモードでOPENします
- 既存標準出力をセーブした後、OPENしたFILE構造体を標準出力に設定します

```
/*
 *  リダイレクションの設定
 */
for(no = 1 ; no < argc ; no++){
    if(*(argb[no]) == '>'){
        if(argc <= (no+1)){
            fwrite(noredirect, sizeof(noredirect), 1, stderr);
            goto tsh_exit;
        }
        mkpath(argb[no+1], rediectpath);
        fd = fopen(rediectpath, "wb");
        if(fd == NULL){
            fwrite(errredirect, sizeof(errredirect), 1, stderr);
            goto tsh_exit;
        }
        if(*(argb[no]+1) == '>'){
            fseek(fd, 0, SEEK_END);
        }
        copy_file_func(stdout, &save_file);
        copy_file_func(fd, stdout);
        argc = no;
        redirect_mode = REDIRECT_OUT;
        break;
    }
}
```


TELNETシェルの処理:コマンド実行

- command_tableコマンド名と一致するコマンド関数を探し実行する
- リダイレクションモードの場合、ファイルをCLOSEして、標準出力を元の設定に戻す
- プロンプトを表示して終了

```
    /*
     * コマンドの実行
     */
    for(cno = 0 ; command_table[cno].cmdstr != 0 ; cno++){
        if(strcmp(command_table[cno].cmdstr, argb[0]) == 0){
            result = command_table[cno].func(argc, argb);
            break;
        }
    }
    /*
     * リダイレクション後処理
     */
    if(redirect_mode == REDIRECT_OUT){
        fclose(fd);
        copy_file_func(&save_file, stdout);
    }
tsh_exit:
    redirect_mode = NO_REDIRECTION;
    prompt();
    return result;
}
```


仮想端末アプリの構築

1. telnetプロトコル
2. TELNET標準入出力
3. TELNETシェル
4. コマンドアプリ

コマンドのディスパッチ

- TELNETシェルはcommand_typeのcommand_tableから第一引数の文字列にあった関数呼び出します
- catコマンドはcmd_cat関数を呼び出します

```
typedef struct _command_table{
    char *cmdstr;
    int    (func)(int argc, char **argv);
} command_type;

const command_type command_table[] = {
    {“date”,      cmd_date},
    {“cd”,        cmd_cd},
    {“ls”,        cmd_ls);
    {“cat”,       cmd_cat},
          …..
    {“cmd”,       cmd},
    {0,            0}
};
```


CATコマンド

- CATコマンドを例にコマンドの実行を解説します
- 第二引数がない場合はエラーとする
- 第二引数をファイル名として読み取りモードでOPENします
- ファイルを読み取り、LFコードがあるところまでをprintf文で表示する
- 最後まで読み取りと表示を繰り返しファイルをCLOSEして終了

```
/*
 * ファイルの表示
 */
static int cmd_cat(int argc, char **argv)
{
    FILE *fid;
    int  c, i = 0;
    char buf[68];

    if(argc < 2){
        fwrite(noargment, sizeof(noargment), 1, stderr);
        return -1;
    }
    mkpath(argv[1], fname);
    fid = fopen(fname, "rb");
    if(fid == NULL){
        fwrite(nofile, sizeof(nofile), 1, stderr);
        return -1;
    }
    while((c = fgetc(fid)) >= 0){
        buf[i] = c;
        i++;
        if(c == CHAR_LF || i >= 64){
            buf[i] = 0;
            printf("%s", &buf[0]);
            i = 0;
        }
    }
    if(i > 0){
            buf[i+1] = 0;
            printf("%s", &buf[0]);
    }
    fclose(fid);
    return 0;
}
```

エラーは標準エラーに出力する

仮想端末アプリの拡張

1. dumpコマンドを拡張
2. Shell コマンドを作る

実習：dumpコマンドを拡張

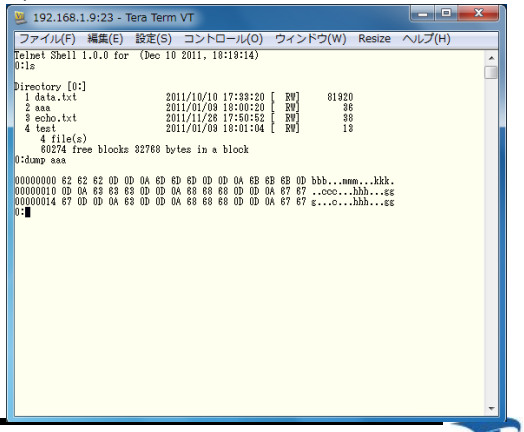
- リダイレクションは'>'、または、'>>'設定にて出力データをファイルにリダイレクションする機能です
- echoコマンドでaaa.txtファイルを作成し、catコマンドでaaa.txtを表示させます
- aaa.txtをHEX(16進数)でdumpするコマンドを拡張しましょう

```
0:echo > aaa.txt
 abcdef
 ghijkl
 mnopqrst
Ctrl-Z
0:cat  aaa.txt
 abcdef
 ghijkl
 mnopqrst
0:
```

この文字は実際は表示されない

実習：dumpコマンドの拡張

- dump ファイル名にて、ファイルの内容を16進数でダンプします
- command.cを拡張します



TOPPERS

実習：dumpコマンドの追加

- command.c中のcommand_tableにdumpコマンド関数（cmd_dump）を拡張します

```
static cmd_cd(int argc, char **argv);
static cmd_cat(int argc, char **argv);
static cmd_dump(int argc, char **argv);
    …..

const command_type command_table[] = {
    {“date”,      cmd_date},
    {“cd”,        cmd_cd},
    {“ls”,        cmd_ls);
    {“cat”,       cmd_cat},
    {“dump”,    cmd_dump},
          …..
    {0,             0}
};
```


TOPPERS

実習：dumpコマンドの追加

- cmd_cat関数を利用して表示の部分を16進数表示するように修正します
- 一行16バイトの表示を行うとして68バイトの文字列に16進数を埋めていきます
- 16進数の変換はmake_hex関数を使用します

この部分を16進表示するように修正

```
/*
 * ファイルのDUMP
 */
static int cmd_dump(int argc, char **argv)
{
    FILE *fid;
    int  c, i = 0;
    char buf[68];

        .....
    }
    while((c = fgetc(fid)) >= 0){
        buf[65] = 0;
        buf[i*3] = make_hex((c>>4) % 0xf);
        buf[i*3+1] = make_hex(c & 0xf);
        buf[i*3+2] = ' ';
        if(c >= ' ' && c < 0x7f)
            buf[i+49] = c;
        else
            buf[i+49] = '.';
        no++;
        i++;
        if(i >=16) {
            printf("\n%08x %s", no - 16, buf);
            i = 0;
        }
    }
    if(i > 0)
        printf("\n%08x %s", no - i, buf);
    fclose(fid);
    return 0;
}
```


TOPPERS

仮想端末アプリの拡張

1. dumpコマンドを拡張
2. Shell コマンドを作る

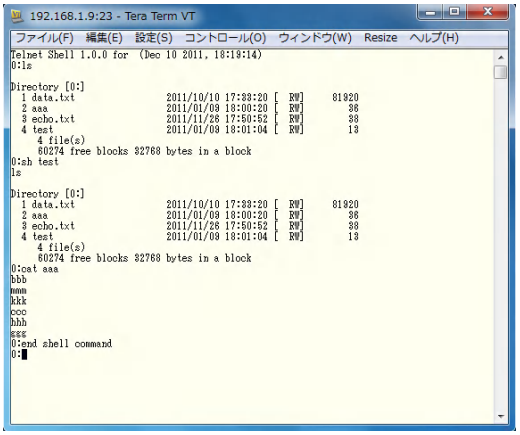
TOPPERS

実習：Shellコマンドを作る

- コマンドをファイルに登録し、ファイルの内容を実行するコマンド（Shellコマンド）を作る

testファイルには
以下の文字列を設定している

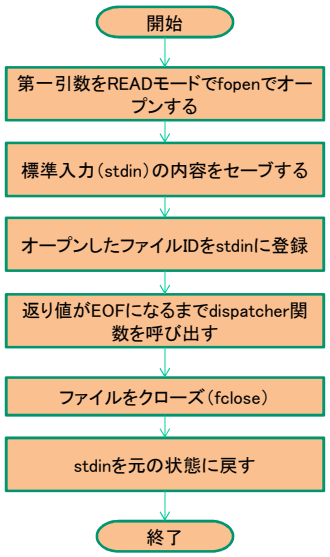
ls⏎
cat aaa⏎



TOPPERS

実習：Shellコマンドを作る

- Shellコマンドはファイルの内容を標準入力に設定し、dispacherを呼び出す
- ファイルの読み出し終了時は、EOF（-1）を返すので、dispacterの返り値がEOFの場合、コマンドを終了する



TOPPERS

実習：Shellコマンドを作る

- 標準入力のファイル構造体の内容を入れ替えるには copy_file_func関数を使用します
- 以下の例はstdinの内容をsave_fileに保存します

```
/*
 * ファイル構造体をコピーする
 */
void copy_file_func(FILE *sst, FILE *dst)
{
    dst->_file          = sst->_file;
    dst->_func_in       = sst->_func_in;
    dst->_func_ins      = sst->_func_ins;
    dst->_func_out      = sst->_func_out;
    dst->_func_outs     = sst->_func_outs;
    dst->_func_flush    = sst->_func_flush;
    dst->_dev           = sst->_dev;
}
```

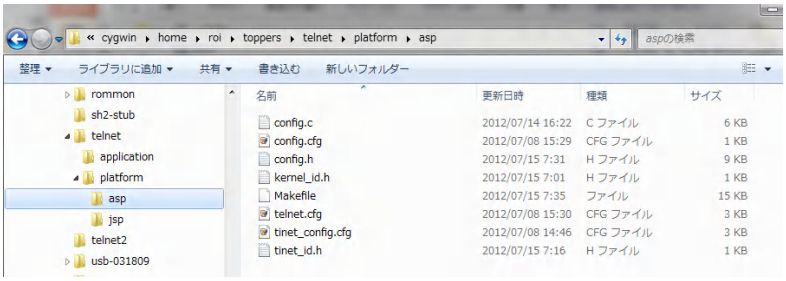
```
static FILR save_save;
  …..
 copy_file_func(stdin, &save_file);
```


宿題：プラットフォームOSの変更

1. RTOSの変更
2. TOPPERS/ASPのセットアップ
3. ワークスペースにASPを追加

RTOSをTOPPERS/ASPに変更

- APIに準拠したアプリケーションは、プラットフォーム部が改変されても、変更なく(あるいは、小変更で)、そのまま再利用が可能です
- この章では、アプリケーションを再利用可能のまま、RTOSをTOPPERS/JSPからTOPPERS/ASPに入れ替え作業を行います



TOPPERS/ASPプラットフォームの構築手順

- TOPPERS/ASP用のプラットフォーム構築手順
  - LPC2388用のTOPPERS/ASPポーティング
    - 既にポーティング済みの手順を説明
  - デバイスドライバーをASP用に修正
    - APIを変更しないように注意
  - RTOSの差異部分を実装で吸収
    - ASPを使用する場合はμITRON4.0互換ヘッダー(itron.h)を使用
    - 実装ヘッダーの差異はプラットフォーム側の実装で吸収
  - telnet/platform/aspにASP関連ソースを集約

宿題：プラットフォームOSの変更

1. RTOSの変更
2. TOPPERS/ASPのセットアップ
3. ワークスペースにASPを追加

TOPPERS

TOPPERS/ASPのダウンロード

- TOPPERSプロジェクトのWebからasp-1.7.0の個別パッケージをダウンロードします
- 同様にARMアーキテクチャ・GCC依存部パッケージ asp_arch_arm_gcc-1.7.0をダウンロードします

ASP Kernel | Documents | Community | Report | Contacts

TOPPERS/ASPカーネル 個別パッケージのダウンロード

TOPPERS/ASPカーネルの最新リリースの個別パッケージを記載しています。各ターゲットに対応するカーネル非依存部パッケージ、依存部パッケージ(アーキテクチャ依存部、ターゲット依存部、シリアルドライバ依存部(PDIC))パッケージを個別にダウンロードできます。

ターゲットごとに必要なソースコードを一つにまとめた簡易パッケージは、こちらからダウンロードできます。

一般公開以前のバージョン(Release 1.3.0以前)に対応した個別パッケージは、会員向け早期リリースとしての扱いですが、こちらからダウンロードできます。

ターゲット非依存部

TOPPERS/ASPカーネル ターゲット非依存部パッケージ(担当：名古屋大学)

パッケージ	リリース日
asp-1.7.0.tar.gz	2011-05-09
asp-1.6.0.tar.gz	2010-08-01

ARMアーキテクチャ・GCC依存部パッケージ(担当：名古屋大学)

パッケージ	ディレクトリ	対応する非依存部のバージョン	リリース日
asp_arch_arm_gcc-1.7.0.tar.gz	arch/arm_gcc/common target/at91skyeye_gcc	1.7.*	2011-05-19
asp_arch_arm_gcc-1.4.0.tar.gz	arch/arm_gcc/common arch/arm_gcc/at91sam7s target/at91skyeye_gcc target/btc090_gcc	1.4.*	2009-05-18
asp_arch_arm_gcc-1.3.2.tar.gz	arch/arm_gcc/common arch/arm_gcc/at91sam7s target/at91skyeye_gcc target/btc090_gcc	1.3.*	2008-08-27
	arch/arm_gcc/common		

TOPPERS

TOPPERS/ASPの解凍

- jsp-1.4.4-stdと同列のディレクトリにダウンロードした圧縮ファイルを置きます
- 添付の開発環境asp-1.7.0-081912.tar.gzも同様のフォルダに置きます
- asp-1.7.0.tar.gzとasp_arch_arm_gcc-1.7.0.tar.gzの解凍後、解凍されたaspディレクトリ名をasp-1.7.0に変更します
- その後、asp-1.7.0-081912.tar.gzを解凍します

```
$ ls
  asp_arch_arm_gcc-1.7.0.tar.gz asp-1.7.0.tar.gz asp-1.7.0-070812.tar.gz
$ tar zxvf asp-1.7.0.tar.gz
$ tar zxvf asp_arch_arm_gcc-1.7.0.tar.gz
$ mv asp asp-1.7.0
$ tar zxvf asp-1.7.0-081912.tar.gz
```


TOPPERS

コンフィギュレータの構築

- Webより、コンフィギュレータをダウンロードしasp-1.7.0フォルダの下に置きます
- LINUXやCygwinでは、cfgの下のMakefileを用いてコンフィギュレータのビルドを行う
- コンフィギュレータの構築はCygwinのバージョンにも関係するため、構築済みの実行形式を利用してもよい

```
$ tar zxvf cfg-1.8.0.tar.gz
$ cd cfg
$ ./configure
$ make
```


TOPPERS新世代カーネル用コンフィギュレータ

TOPPERS新世代カーネル用コンフィギュレータの最新リリースを配布しています。コンフィギュレータ自体についてはこちらを、構築方法等についてはアーカイブに含まれる README.txt をご覧ください。

バージョンごとに3種類を配布しています。ソースファイルについて tar.gz (EUC,LF) と lzh(SJIS,CR/LF) の内容は同一です。

最新のリリース

リリース名	タイプ	サイズ	リリース日
コンフィギュレータ Release 1.8.0	tar.gz (EUC,LF)	91KB	2012-05-09
コンフィギュレータ Release 1.8.0(Windows用バイナリ)	zip	1.0MB	2012-05-09
コンフィギュレータ Release 1.8.0(Cygwin用バイナリ)	tar.gz	628KB	2012-05-09

Release 1.7.0との違い

TOPPERS

TOPPERS/ASP用のワークスペースの作成

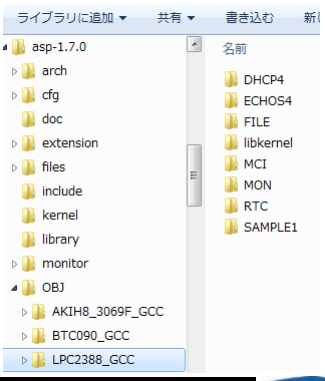
- OBJディレクトリ以下にJSPと同等のワークスペースを用意しました
- ASPカーネル用のカーネルライブラリィを作成します
- SAPMLE1ディレクトリに移動し、SAMPLE1を作成します
- 作成されたasp.hexをLPC2388に書き込めば実行できます

```
$ cd ../OBJ/LPC2388_GCC
$ ls
  DHCP4 ECHOS4 FILE MCI MON RTC SAMPLE1
$ mkdir libkernel
$ cd libkernel
$ ../../../configure –T lpc2388_gcc -f
$ make depend
$ make libkernel.a
$ ../SAMPLE1
$ make depend
$ make
```


ドライバテストの実行と確認

- TOPPERS/ASPでも、JSPのドライバーテストプログラムと同様にドライバテストプログラムを用意しました
- １日目のRTC、ファイルシステム、２日目のDHCPの実習手順を参考にデバイスドライバーのテストを行ってください

ディレクトリ	内容
DHCP4	DHCPアプリ評価プログラム
ECHOS4	エコーサーバー評価プログラム
FILE	ファイルシステム評価プログラム
libkernel	カーネルライブラリィ
MCI	MCIデバイスドライバー評価プログラム
MON	sample1+タスクモニタ
RTC	RTCデバイスドライバー評価プログラム
SAMPLE1	JSP評価プログラム



デバイスドライバーの確認

- TOPPERS/JSPではデバイスドライバをpdic/lpc23xx以下の３つのディレクトリに分散して実装しました
  - dma mci rtc
- TOPPERS/ASPではpdic/lpc23xxにそのまま置きました
- これらのソースファイルは、まったく、同じ機能を実行しますが、RTOSの差異により、実装が異なります
- Cygwinのdiffコマンドを用いて、実装の違いを確認しましょう

```
$ cd ../../../..
$ diff asp-1.7.0/pdic/lpc23xx jsp-1.4.4-std/pdic/lpc23xx/dma
$ diff asp-1.7.0/pdic/lpc23xx jsp-1.4.4-std/pdic/lpc23xx/mci
$ diff asp-1.7.0/pdic/lpc23xx jsp-1.4.4-std/pdic/lpc23xx/rtc
```


宿題：プラットフォームOSの変更

1. RTOSの変更
2. TOPPERS/ASPのセットアップ
3. ワークスペースにASPを追加

platform/aspの作成

- platformの下のディレクトリにTOPPERS/ASPアプリ生成用のディレクトリaspを作成する
- JSP用のAPIをasp中のファイルで吸収するように各ファイルを作成する

platform/asp	platform/jsp	記載の内容
config.cfg	config.cfg	asp/jspでは記載が異なる
config.h	config.h	アプリはrtosの設定を、このインクルードに集中している
Makefile	Makefile	asp/jspでは記載が異なる
telnet.cfg	telnet.cfg	asp/jspでは記載が異なる
tinet_config.cfg	tinet_config.cfg	asp/jspでは記載が異なる
kernel_id.h		aspではkernel_cfg.hを生成するため、これをインクルード
telnet_id.h		aspではtelnet_cfg.hを生成するため、これをインクルード

まとめ

基礎3講座2日目のまとめ

- ITRON-TCP/IP仕様について勉強しました
- TINETのETHERNETドライバの作成方法について学びました。他のボードにポーティングするケースの参考にしてください
- ECHOサーバーの内容について学びました。ECHOサーバーをベースにTELNETサーバーを作成し、他のデバイスドライバを追加してプラットフォームを作成しました
- プラットフォーム上に種々のコマンドを実行するTELNETアプリを作成し、改造を行いました

基礎3講座2日目のまとめ

- RTOS上にデバイスドライバやミドルウェアを単純に足しこんでTELNETアプリを作成するケースと比べて、APIをきちんと設定しプラットフォーム上にアプリケーションを構築した方が、簡単に改造や拡張が可能なことが確認できたと思います
- 特に、きちんとプラットフォームを作成した場合、アプリケーションの作成にデバイスドライバの内容や不具合を気にせず、APIのみをベースにアプリケーションの構築ができたはずです
- 組込みプラットフォームを導入すると、複雑なメカニズム開発作業とアプリケーション開発作業を分離して行うことができます

参考文献

- LPC23XX User manual(UM10211)
- LPC2388 Data Book
- ITRON TCP/IP API仕様（Ver. 1.00.01）
  - （社）トロン協会　ITRON専門委員会
- TOPPERS/JSP用 TCP/IPプロトコルスタック（TINET）
  - ユーザズマニュアル
  - TINET-1.3におけるイーサネットの実装

謝辞

DHCPクライアントを実行するプログラムdhcp.cは長野工業技術総合センターの浜 淳さんが作成したプログラムを使用しています

著者リスト

- ITRON-TCP/IP仕様
  - 高田 広章(名古屋大学), 本田 晋也(名古屋大学)
  - 山本 雅基(名古屋大学)
- TINETデバイスドライバの設計
- DHCP+ECHOサーバーの実装確認
- プラットフォームの作成
- 仮想端末アプリの構築
- 仮想端末アプリの拡張
  - 竹内 良輔((株)リコー)