TK-850/JH3U-SP

ネットワークプロトコル

ユーザーズ・マニュアル(応用編)

（第 1.0 版）

テセラ･テクノロジー株式会社

目次

〔簡単アプリ作成編〕

1. 準備

アプリケーションを作成するには、まず、開発環境及び評価キットのソフトウェアのインストールが必要です。

1.1 用意して頂くもの

開発および評価で使用する PC に必要な条件は以下のとおりです。

CPU	Pentium3 800MHz 以上、または同等以上の CPU
メモリ	128MB 以上（256MB 以上推奨）
OS	Windows2000 Professional WindowsXP Professional
Web ブラウザ	InternetExplorer6.0 以上
RS-232C	1 ポート
Ethernet	100BASE-TX/10BASE-T ×1 （アプリケーションで Ethernet を使用する場合）

1.1.1 使用する開発ツール

- デバイス・ファイル DF70F3769_V1.00

  デバイス固有の情報は、デバイス・ファイルに入っているため、開発ツールを使用するにはデバイス・ファイルが必要となります。
- 統合開発環境 PM+ V6.30

  Windows 上での統合開発環境プラットホームです。

  編集ウィンドウとしてアイデアプロセッサ機能付きエディタを搭載し、コンパイラ、ディバッガなどの開発ツールと連携して効率的な開発を行なうことができます。
- C コンパイラ・パッケージ CA850 W3.30（サイズ限定版）

  C ソース・プログラムやアセンブラ・ソース・プログラムから V850 シリーズで実行することができる実行コードを生成します。

  本製品のCA850 W3.30 にはオブジェクト・サイズが 128KByteまでの制限事項を設けております。

- 統合ディバッガ ID850-QB V3.50

  ホスト PC 上で動作する Windows ベースのソフトウェアです。

  C ソース・レベルでのディバグを実現する統合ディバッガです。

  変数の参照・変更やソース行単位でのステップ実行など、ソース・ディバグを簡単かつ効率的に行うことができます。

- マイコン内蔵フラッシュメモリ書き込みプログラム WriteEZ1

  マイコンの内蔵フラッシュメモリにプログラムを書き込む Windows ベースのソフトウェアです。

  TK-850/JH3U-SP を添付の USB ケーブルでパソコンと接続することによって V850ES/JH3-U の内蔵フラッシュメモリに対する書き込み／消去を行うことが出来ます。

1.1.2 ソフトウェアのインストールと設定

(1) アプリケーションを作成するためには、上記開発ツールのインストールが必要です。
インストール方法については、TK-850/JH3U-SP ユーザーズ・マニュアルをご覧ください。

1.1.3 評価キットソフトウェアのインストール

評価キットソフトウェアのプロジェクトを、パッケージより評価用 PC にインストール(コピー)します。

(1) TK-850/JH3U-SP ユーザーズ・マニュアルの「サンプル・プログラム本体のインストール」を参照し、サンプルソフトをインストールします。

(2) プロジェクトは以下のようなフォルダ構成で、標準で添付するサンプルアプリケーションと同一の物件を作成するためのプロジェクト構成となっています。

プロジェクト

¥cnet ……..ユーザのプログラム作成用フォルダ

¥include……..include ファイルを置きます。

¥lib……オブジェクトで提供する評価キットソフトウェアが格納されています。
(ファイルの変更不可)

¥src……ユーザが作成するプログラムのソースを置きます。

main.c…….プログラムを起動するファイル

userap.c……API 関数等を使用したユーザプログラムを記載するファイル

httpdfunc.c…….HTTP サーバに実行させたいユーザの関数を記載するファイル

¥bin……実行ファイルが格納されています。

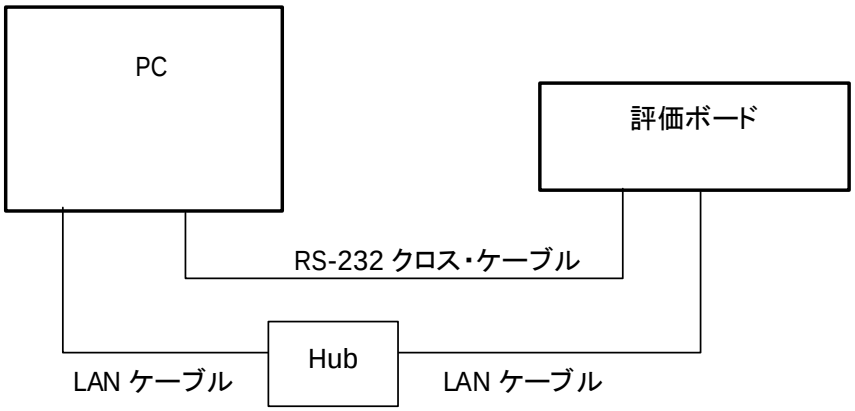
GMakefsdata. exe.. Web ファイル登録ツール （後述）

¥fs……ユーザが作成した HTML ファイルを置きます。 (後述)

¥cgi..... ユーザが作成した CGI ファイルを置きます。 (後述)

上記¥lib フォルダ内のファイルおよび¥include フォルダ内のファイルに不要な変更を行うと、動作不正が発生する場合がありますので注意してください。

1.2 機器接続



ホスト PC と評価キットボードを接続します。

(1) 評価ボードのディップスイッチを以下の状態に設定します。

SW1	1～4	1:OFF 2:ON 3:ON 4:ON	ID850QB を使用する場合
		1:OFF 2:OFF 3:OFF 4:ON	ID850QB を使用せずに実行する場合
		1:OFF 2:ON 3:ON 4:ON	WriteEZ1 を使用する場合
	5～8	ON／OFF 任意 （ユーザ開放）	
SW2		押下:リセット （リセット・スイッチ）	
SW3		押下:機能なし （未使用）	
SW4		押下:機能なし （未使用）	

2. 自分でアプリケーションを作成してみる

2.1 概説

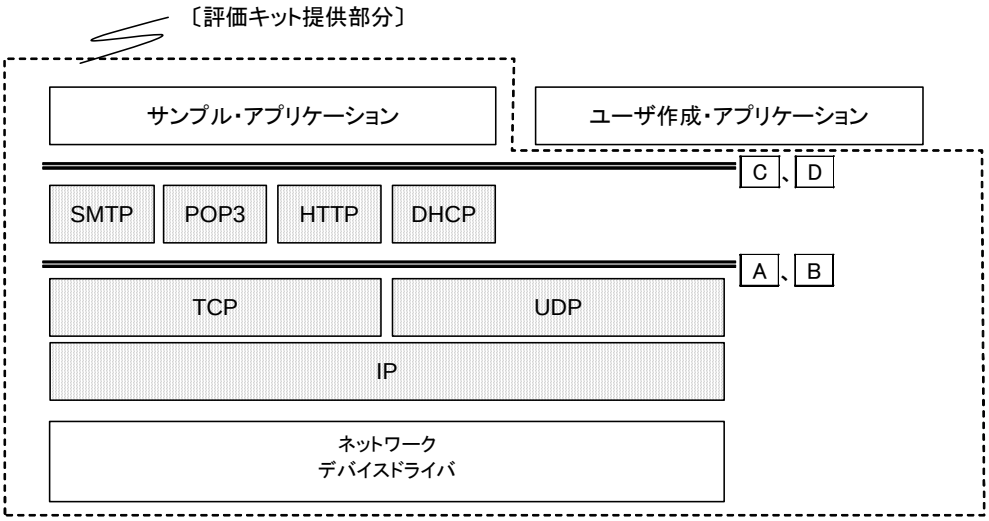
評価キットでは、自分でアプリケーションを作成し、プログラムを実行したり、Web アプリケーションを作成したりするための各種 API インタフェース関数を提供します。

A. ネットワーク機能（TCP） API

B. ネットワーク機能（UDP） API

C. ネットワーク機能　独自 API

D. アプリケーション layerAPI



2.1.1 ネットワーク機能（TCP） ＡＰＩ

TCP で通信を行うプログラムを作成するための API 関数を提供します。

2.1.2 ネットワーク機能（UDP） ＡＰＩ

UDP で通信を行うプログラムを作成するための API 関数を提供します。

2.1.3 ネットワーク機能 独自ＡＰＩ

ネットワークの初期設定および各種設定取得を行う API 関数をサポートしています。

2.1.4 アプリケーション layerAPI

アプリケーションプロトコルとして DHCP、HTTP、POP3、SMTP の API を提供しています。

- DHCP

インターネットに一時的に接続するコンピュータに、IP アドレスなど必要な情報を自動的に割り当てるプロトコルです。本キットは、DHCP クライアントとして動作するための API をサポートしています。DHCP サーバ機能はサポートしておりません。

- HTTP

Web サーバとクライアント(Web ブラウザなど)がデータを送受信するのに使われるプロトコルです。HTTP サーバを構築し、ユーザが HTML 文書や、CGI を作成し、実行するための API をサポートしています。

- POP3

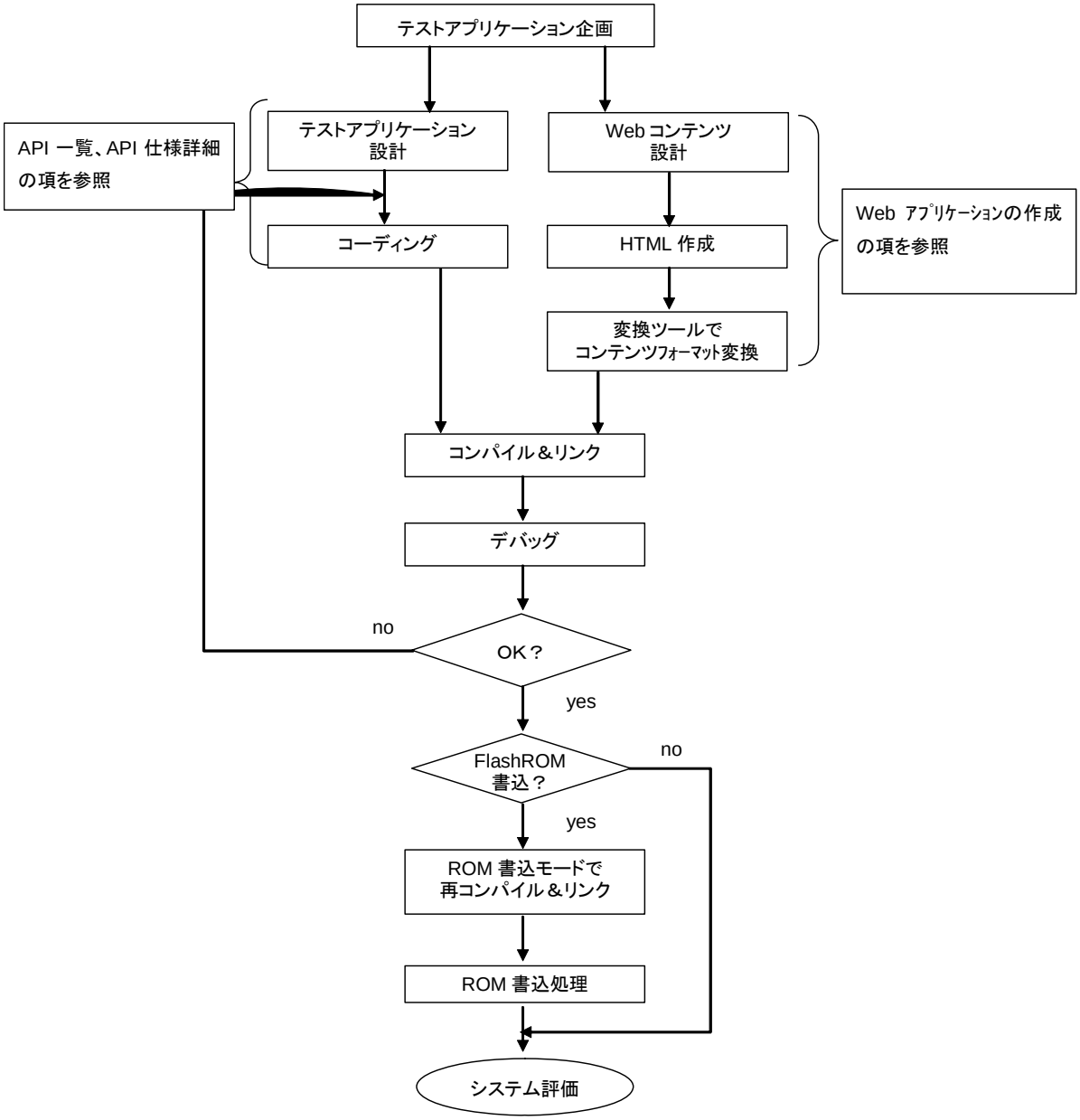
インターネットやイントラネット上で、電子メールを保存しているサーバからメールを受信するためのプロトコルです。メールを受信するための各種 API 関数をサポートしています。

- SMTP

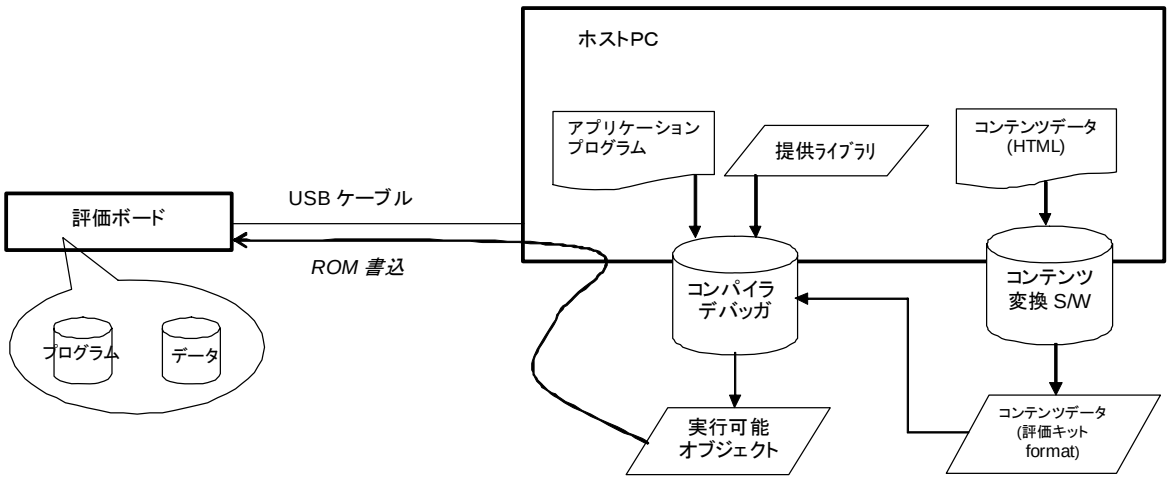
インターネットやイントラネットで電子メールを送信するためのプロトコルです。メールを送信するための各種 API 関数をサポートしています。

2.2 アプリケーションの作成方法

評価キットを用いてユーザアプリケーションを作成する際の流れの例を以下に示します。



また、評価キットを用いてユーザアプリケーションを作成する際の開発ツールなどの構成を以下に示します。



2.2.1 添付しているアプリケーションの説明

プログラム起動時に動作するプログラムとして添付しているものは以下のとおりです。

- メール操作サンプルプログラム (¥cnet¥src¥userap.c)

Web 用コンテンツとして、添付しているプログラムは、以下のとおりです。

- シリアル操作 (¥cnet¥fs¥serial.html)
- DIP-SW 状態取得 (¥cnet¥fs¥cgi¥dipsw.cgi)

各プログラムの仕様の詳細については、「**TK-850/JH3U-SP ネットワークプロトコル ユーザーズ・マニュアル(基礎編)** 2.4 サンプルアプリケーション」の項を参照して下さい。

2.2.2 メモリマッピング

(1) メモリ配置

評価キットでのメモリ配置は、基本的に以下のようになります。

<table>
<tr><td>0x0000000

0x007F000
0x007FFFF</td><td>内蔵 ROM
（512KB）</td><td>評価用 TCP/IP
ユーザアプリ

ネットワーク設定情報</td></tr>
<tr><td></td><td>外部メモリ領域※
※今回は使用していません</td><td></td></tr>
<tr><td>0x3FF3000

0x3FFEFFF
:</td><td>内蔵 RAM
（48KB）</td><td>ワーク領域
FlashROM 関連関数</td></tr>
</table>

例として、サンプルアプリケーションを含む添付プロジェクトのメモリ配置を示します。

```
開始アドレス      終了アドレス      コメント
--------------------------------------------------------------------------------
0x00000000        0x000004C4        割り込みベクタ
0x000004C8        0x00004981        HTML データを含む固定値のデータ
0x00004988        0x0001004F        プログラムコード部（TCP/IP、ユーザプログラム）
0x0007F000        0x0007F1EF        ネットワーク設定情報

0x03ff3100        0x03ff7A9F        データ領域
0x03ff7AA0        0x03ff836B        FlashROM 関連関数
```

(2)　　セクション名一覧

セクション名	内容
flash.const	FlashROM 関連セクション
SelfLib_ToRamUsrInt.text	FLASH セルフライブラリ関連セクション
SelfLib_Rom.text	
SelfLib_ToRam.text	
SelfLib_RomOrRam.text	
SelfLib_ToRamUsr.text	

予約されているセクション名は使用できません。

上記のセクション名はすでに使用されていますので、ユーザ作成のアプリケーションでは、同じセクション名をつけないようにして下さい。

2.2.3 API 一覧

A. ネットワーク機能（TCP） API

API 名	機能概要
tcp_new	TCP PCB の作成
tcp_close	TCP のコネクション切断と TCP PCB の削除
tcp_abort	RST の送信による TCP のコネクション中断と TCP PCB の削除
tcp_bind	ローカルのポート番号と IP アドレスの設定
tcp_listen	接続要求を受け入れることができる TCP PCB の作成
tcp_connect	接続要求の送信
tcp_accept	接続要求受け入れ時に呼ばれるコールバック関数の登録
tcp_recv	データ受信時に呼ばれるコールバック関数の登録
tcp_sent	データ送信成功時に呼ばれるコールバック関数の登録
tcp_poll	定期的に呼ばれるコールバック関数の登録
tcp_err	通信エラー発生時に呼ばれるコールバック関数の登録
tcp_arg	コールバック関数へ渡すデータの登録
tcp_write	送信データをキューへ格納
tcp_output	未送信データの送信
tcp_recved	受信完了の通知
tcp_setprio	プライオリティの設定
tcp_sndbuf	使用可能な送信バッファ領域（バイト数）の取得

B. ネットワーク機能（UDP） API

API 名	機能概要
udp_new	UDP PCB の作成
udp_remove	UDP PCB の削除
udp_bind	ローカルのポート番号と IP アドレスの設定
udp_connect	リモートのポート番号と IP アドレスの設定
udp_disconnect	リモートのポート番号と IP アドレスの設定解除
udp_recv	データ受信時に呼ばれるコールバック関数の登録
udp_send	データの送信

C. ネットワーク機能 独自API

API 名	機能概要
cnet_init	ネットワークの初期化
netif_add	ネットワークインタフェースの作成
netif_remove	ネットワークインタフェースの削除
netif_set_default	デフォルトのネットワークインタフェースの登録
netif_set_addr	IP アドレスの設定
cnet_single_routine	ネットワーク処理
cnet_getipaddr	IP アドレスの取得
cnet_getnetmask	サブネットマスクの取得
cnet_getgwaddr	ゲートウェイアドレスの取得
inet_addr	ドッド形式の IP アドレスをバイトオーダ形式へ変換
inet_ntoa	バイトオーダ形式の IP アドレスをドット形式へ変換
htonl	ホストバイトオーダーの 32 ビット値をネットワークバイトオーダーへ変換
htons	ホストバイトオーダーの 16 ビット値をネットワークバイトオーダーへ変換
ntohl	ネットワークバイトオーダーの 32 ビット値をホストバイトオーダーへ変換
ntohs	ネットワークバイトオーダーの 16 ビット値をホストバイトオーダーへ変換

D. アプリケーションlayerAPI (SMTP、POP3、DHCP、HTTP)

API 名	機能概要
SMTP	
cnet_smtpinit	SMTP 機能の開始
cnet_smtpend	SMTP 機能の終了
cnet_smtpdatafirst	メールデータの送信開始
cnet_smtpdatanext	メールデータの続きの送信
cnet_smtpdataend	メールデータの送信終了
POP3	
cnet_pop3init	POP3 機能の開始
cnet_pop3end	POP3 機能の終了
cnet_pop3auth	ユーザ名、パスワードによる認証

cnet_pop3stat	メール情報の取得
cnet_pop3getsize	メッセージサイズの取得
cnet_pop3getmsgfirst	メッセージの取得開始
cnet_pop3getmsgnext	メッセージの続きの取得
cnet_pop3getmsgend	メッセージの取得終了
cnet_pop3delete	メッセージの削除マークの設定
cnet_pop3reset	全メッセージの削除マークの設定取り消し
cnet_pop3getuid	メッセージの unique-id の取得
DHCP	
cnet_dhcpstart	DHCP クライアント処理の開始
cnet_dhcpstop	DHCP クライアント処理の終了
HTTP	
cnet_httpdinit	HTTP サーバの開始
cnet_httpdsend	HTTP サーバ専用のデータ送信
cnet_httpdsendcont	データ送信の継続処理関数の登録、登録解除

2.3 ユーザアプリケーションのプログラミング例

2.3.1 ネットワーク機能（TCP）API を使用する

TCP を使用したアプリケーションである TCP エコーサーバプログラムを例に TCP API の使用方法を説明します。

```
#include "lwip/tcp.h"
#include "arch/lib.h"

struct tcpecho_state {
   struct pbuf *p;
};
/*-----------------------------------------------------------------------------------*/
static void
close_conn(struct tcp_pcb *pcb, struct tcpecho_state *es)
{
   tcp_arg(pcb, NULL);
   tcp_recv(pcb, NULL);
   tcp_sent(pcb, NULL);
   tcp_err(pcb, NULL);
   tcp_poll(pcb, NULL, 0xff);
   if (es != NULL) {
      pbuf_free(es->p);
      mem_free(es);
   }
   tcp_close(pcb);
}
/*-----------------------------------------------------------------------------------*/
static void
send_buf(struct tcp_pcb *pcb, struct tcpecho_state *es)
{
   struct pbuf *q;
   u16_t len;

   while (es->p != NULL) {
      q = es->p;
      len = tcp_sndbuf(pcb);
      if (len == 0) {
         break;
      }
      if (len > q->len) {
         len = q->len;
      }
      if (tcp_write(pcb, q->payload, len, 1) != ERR_OK) {
         break;
      }
      if (len < q->len) {
         pbuf_header(q, -len);
      }
```

```
      else {
        es->p = pbuf_topfree(q);
        tcp_recved(pcb, len);
      }
    }
}
/*-----------------------------------------------------------------------------*/
static void
tcpecho_err(void *arg, err_t err)
{
  struct tcpecho_state *es = arg;

  if (arg != NULL) {
    pbuf_free(es->p);
    mem_free(arg);
  }
}
/*-----------------------------------------------------------------------------*/
static err_t
tcpecho_poll(void *arg, struct tcp_pcb *pcb)
{
  struct tcpecho_state *es = arg;

  if (es == NULL) {
    tcp_abort(pcb);
    return ERR_ABRT;
  }
  if (es->p != NULL) {
    send_buf(pcb, es);
  }
  return ERR_OK;
}
/*-----------------------------------------------------------------------------*/
static err_t
tcpecho_sent(void *arg, struct tcp_pcb *pcb, u16_t len)
{
  struct tcpecho_state *es = arg;

  if (es != NULL && es->p != NULL) {
    send_buf(pcb, es);
  }
  return ERR_OK;
}
/*-----------------------------------------------------------------------------*/
static err_t
tcpecho_recv(void *arg, struct tcp_pcb *pcb, struct pbuf *p, err_t err)
{
  struct tcpecho_state *es = arg;

  if (p == NULL) {
    close_conn(pcb, es);
    return ERR_OK;
  }
  if (es->p != NULL) {
```

```
      pbuf_cat(es->p, p);
    }
    else {
      es->p = p;
    }
    send_buf(pcb, es);
    return ERR_OK;
}
/*-----------------------------------------------------------------------------*/
static err_t
tcpecho_accept(void *arg, struct tcp_pcb *pcb, err_t err)
{
    struct tcpecho_state *es;

    tcp_setprio(pcb, TCP_PRIO_MIN);                          ・・・(5)

    /* Allocate memory for the structure that holds the state of the
       connection. */
    es = mem_malloc(sizeof(struct tcpecho_state));
    if (es == NULL) {
      return ERR_MEM;
    }
    /* Initialize the structure. */
    es->p = NULL;

    tcp_arg(pcb, es);                                        ・・・(6)
    tcp_recv(pcb, tcpecho_recv);                             ・・・(7)
    tcp_sent(pcb, tcpecho_sent);                             ・・・(8)
    tcp_err(pcb, tcpecho_err);                               ・・・(9)
    tcp_poll(pcb, tcpecho_poll, 2);                          ・・・(10)
    return ERR_OK;
}
/*-----------------------------------------------------------------------------*/
struct tcp_pcb*
tcpecho_init(void)
{
    struct tcp_pcb *pcb;

    pcb = tcp_new();                                         ・・・(1)
    tcp_bind(pcb, IP_ADDR_ANY, 7);                           ・・・(2)
    pcb = tcp_listen(pcb);                                   ・・・(3)
    tcp_accept(pcb, tcpecho_accept);                         ・・・(4)
    return pcb;
}
/*-----------------------------------------------------------------------------*/
void
tcpecho_end(struct tcp_pcb *pcb)
{
    tcp_accept(pcb, NULL);
    tcp_close(pcb);                                          ・・・(11)
}
/*-----------------------------------------------------------------------------*/
```

TCP エコーサーバの処理は、tcpecho_init 関数から始まります。

（1）最初に tcp_new 関数で、TCP PCB を作成します。

（2）tcp_bind 関数で、ローカルアドレスを設定します。

TCP エコーサーバでは、IP アドレスに IP_ADDR_ANY、ポート番号に7を設定しています。

（3）tcp_listen 関数で、TCP PCB が接続要求を受け入れられるようにします。

（4）tcp_accept 関数で、接続要求受け入れ後の処理を行うコールバック関数を登録します。

TCP エコーサーバでは、tcpecho_accept 関数を登録しています。

tcpecho_accept 関数で、接続要求受け入れ後の処理を行っています。

（5）tcp_setprio 関数で、プライオリティを設定します。

tcp_setprio 関数の実行は、必須ではありません。tcp_setprio 関数を実行しない場合のプライオリティは、標準値（64）になります。

TCP エコーサーバでは、プライオリティを最低にしています。

（6）tcp_arg 関数で、各コールバック関数へ渡したいデータを設定します。

TCP エコーサーバでは、tcpecho_state 構造体を登録しています。

（7）tcp_recv 関数で、データ受信時に呼ばれるコールバック関数を登録します。

TCP エコーサーバでは、tcpecho_recv 関数を登録しています。

tcpecho_recv 関数では、受信したデータをそのまま送信し返しています。

（8）tcp_sent 関数で、データ送信成功時に呼ばれるコールバック関数を登録します。

TCP エコーサーバでは、tcpecho_sent 関数を登録しています。

tcpecho_sent 関数では、未処理の送信データがある場合、送信処理を行います。

（9）tcp_err 関数で、通信エラー時に呼ばれるコールバック関数を登録します。

TCP エコーサーバでは、tcpecho_err 関数を登録しています。

tcpecho_err 関数では、受信データと tcpecho_state 構造体の領域を解放しています。

（10）tcp_poll 関数で、定期的に呼ばれるコールバック関数を登録します。

TCP エコーサーバでは、tcpecho_poll 関数を 1 秒毎に呼ぶようにしています。

tcpecho_poll 関数では、未処理の送信データがある場合、送信処理を行います。

tcpecho_end 関数を実行すると、TCP エコーサーバが終了します。

（11）tcp_close 関数で、TCP PCB を削除します。

2.3.2 ネットワーク機能（UDP）API を使用する

UDP を使用したアプリケーションである UDP エコーサーバプログラムを例に UDP API の使用方法を説明します。

```
#include "lwip/udp.h"

/*-----------------------------------------------------------------------------------*/
static void
udpecho_recv(void *arg, struct udp_pcb *upcb,
                    struct pbuf *p, struct ip_addr *addr, u16_t port)
{
   if (p == NULL) {
      return;
   }

   udp_connect(upcb, addr, port);                                    ・・・（4）
   udp_send(upcb, p);                                                ・・・（5）
   pbuf_free(p);
   udp_disconnect(upcb);                                             ・・・（6）
}
/*-----------------------------------------------------------------------------------*/
struct udp_pcb *
udpecho_init(void)
{
   struct udp_pcb *upcb;

   upcb = udp_new();                                                 ・・・（1）
   udp_bind(upcb, IP_ADDR_ANY, 7);                                   ・・・（2）
   udp_recv(upcb, udpecho_recv, NULL);                               ・・・（3）
   return upcb;
}
/*-----------------------------------------------------------------------------------*/
void
udpecho_end(struct udp_pcb *upcb)
{
   if (upcb == NULL) {
      return;
   }

   udp_recv(upcb, NULL, NULL);
   udp_remove(upcb);                                                 ・・・（7）
}
/*-----------------------------------------------------------------------------------*/
```

UDP エコーサーバの処理は、udpecho_init 関数から始まります。

（1）最初に udp_new 関数で、UDP PCB を作成します。

（2）udp_bind 関数で、ローカルアドレスを設定します。

UDP エコーサーバでは、IP アドレスに IP_ADDR_ANY、ポート番号に7を設定しています。

（3）udp_recv 関数で、受信したデータの処理を行うコールバック関数を登録します。

UDP エコーサーバでは、udpecho_recv 関数を登録しています。

udpecho_recv 関数で、受信したデータの処理を行っています。

（4）udp_connect 関数で、通信相手のポート番号と IP アドレスを設定します。

UDP エコーサーバでは、データ送信元のポート番号と IP アドレスを設定しています。

（5）udp_send 関数で、データを送信します。

UDP エコーサーバでは、受信したデータをそのまま送信し返しています。

（6）udp_disconnect 関数で、通信相手のポート番号と IP アドレスの設定を解除します。

udpecho_end 関数を実行すると、UDP エコーサーバが終了します。

（7）udp_remove 関数で、UDP PCB を削除します。

2.3.3 ネットワーク初期化関数を使用する

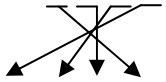
ネットワーク初期化関数(cnet_init)を使用する前には、以下のファイルを設定する必要があります。

- ファイル名：¥cnet¥include¥config.h
  使用するネットワーク関連情報を設定します。

```
=== config.h ============================================================
        #define   CONFIG_DHCP              0               /* 0: OFF, 1: ON */
        #define   CONFIG_MY_IPADDRESS      0x0500a8c0      /* for no DHCP */
        #define   CONFIG_NETMASK           0x00ffffff      /* for no DHCP */
        #define   CONFIG_GATEWAY           0xfe00a8c0      /* for no DHCP */
        #define   CONFIG_DNS_SERVER1       0x0100a8c0      /* for no DHCP */
        #define   CONFIG_DNS_SERVER2       0x00000000      /* for no DHCP */
        #define   CONFIG_DOMAIN            "localdomain"   /* for no DHCP, MAX 64 characters */
        #define   CONFIG_HOSTNAME          " tk850_jh3u-sp "   /* MAX 64 characters */
==========================================================================
```

IP アドレス、ネットマスク、ゲートウェイ等の設定は、16 進で逆から設定してください。

例：CONFIG_MY_IPADDRESS： 0x0500a8c0



192.168.0.05

上記のファイルを修正後、プログラムビルドを行います。
ビルドの方法については、「2.5 実行モジュールの作成」の項を参考して下さい。

2.3.4 メール送受信を行う

メール送受信 API 関数の使い方については、userap.c 内にある、メール送受信関数使用例を参考にしてください。

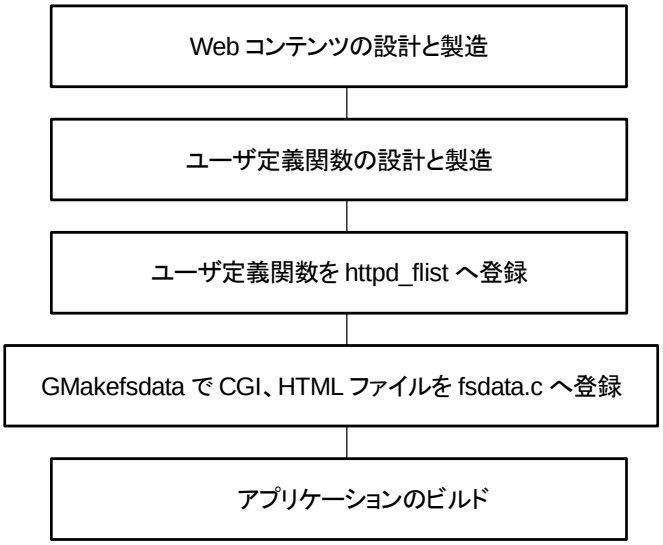
2.3.5 ユーザアプリケーション作成時の注意点

¥cnet¥lib フォルダ内のファイルおよび¥cnet¥include フォルダ内のファイルに不要な変更を行わないでください。システムファイルやシステム変数等が不正となり、動作しない場合があります。

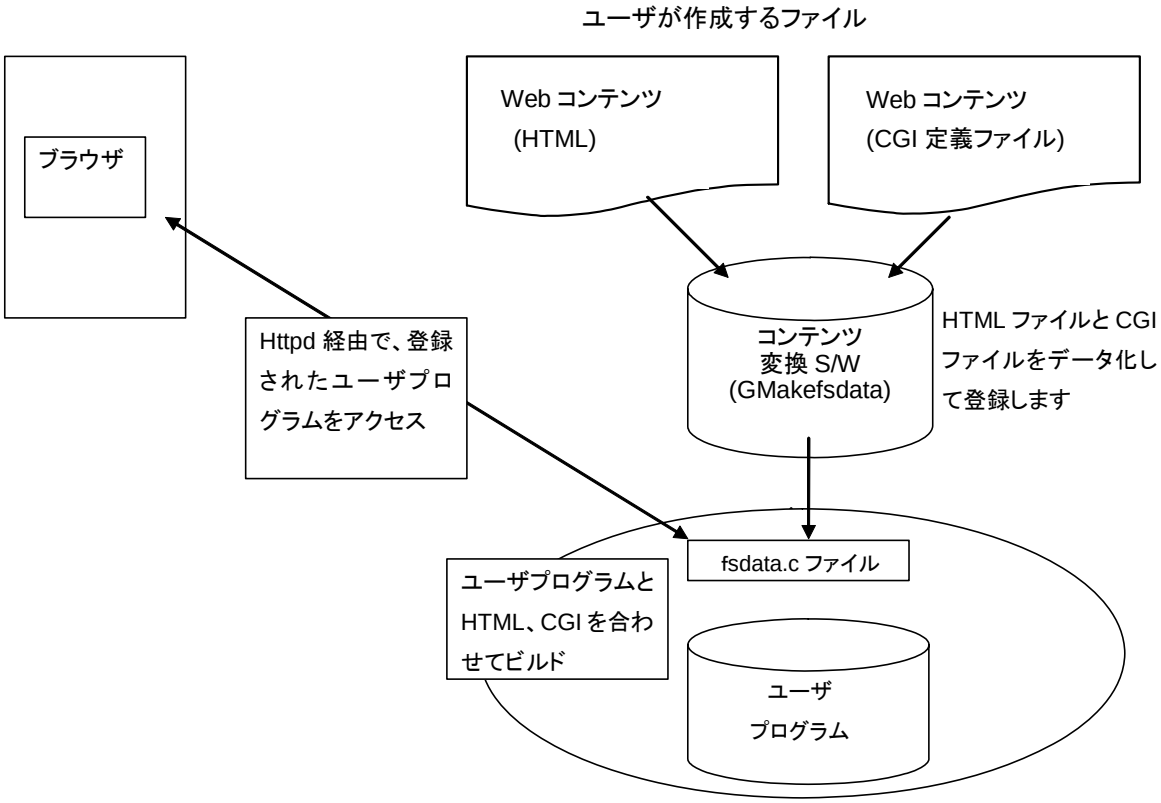
2.4 Web アプリケーションの作成方法

2.4.1 概要

Web アプリケーションを作成する際の流れの例を以下に示します。



また、Web のアプリケーションを作成するための構成を以下に示します。



2.4.2 CGI 関連の仕様

CGI の記述形式

CGI の記述形式は以下のとおりです。

- 1 処理単位を1行で表します。
- 各行の先頭文字が処理内容を表します。

i “i”に続けて指定されたファイルの内容を出力します

t “t”に続けて指定されたデータを出力します

c “c”に続けて指定された関数識別名、パラメータにより機能（関数）を実行します

＊ 204(No Contents) を送信します

コメント。処理を行いません

.（ﾋﾟﾘｵﾄﾞ）

HTML の記述を終了します（</BODY></HTML>¥r¥n）

その他 処理を行いません

2.4.3 各種メソッドの定義

(1) GET メソッド

【機能】

HTML や CGI、画像データの表示を要求します。

【規則】

“/”のみ指定された場合は、index.html を表示します。

指定された HTML または CGI がない場合は、404.html を表示します。

/cgi/配下のファイルが指定された場合は、CGI と見なします。

【指定例】

GET /index.html HTTP/1.0

GET /test.html?func1=param1 HTTP/1.0

(2) POST メソッド

【機能】

CGI へパラメータを引き渡します。

【規則】

/cgi/配下のファイルを指定しなかった場合は、エラー405 を返却します。

指定された CGI がない場合は、404.html を表示します。

2.4.4 ユーザ定義関数の定義方法

CGI 等で処理を行う関数を作成する場合は、httpd_flist 構造体に関数識別名と関数の対応を記述します。この関数識別名を CGI や POST メソッドで渡すデータ等で指定することによって対応する関数が呼び出されます。

(1) ユーザ定義関数

【機能】

GET メソッドのパラメータ、POST メソッドで渡すデータ、CGI の"c"（関数実行)コマンドのいずれかで関数識別名を指定することで、ユーザ定義関数の呼び出しが行われます。

このユーザ定義関数は、httpdfunc.c または任意のファイルに実装してください。

(2) ユーザ定義関数の形式

ユーザ定義関数の引数は3つで数、内容ともに固定です。

第 1 引数	int 型	HTTP サーバが情報を管理するための ID を渡します。
第 2 引数	const char*型	関数識別名を渡します。
第 3 引数	const char*型	CGI では指定されたパラメータ、 GET、POST メソッドでは受け取ったデータを渡します。

cnet_httpd_datacmn()の形式

cnet_httpd_datacmn()は、GET、POST メソッドで受け取ったデータの処理後に必ず呼ばれる関数で、引数を 1 つ持ちます。ただし、GET メソッドでパラメータを指定しない場合は呼ばれません。

第 1 引数	int 型	HTTP サーバが情報を管理するための ID を渡します。

ユーザ定義関数の実装例については、サンプルアプリケーションとして添付する httpdfunc.c を参考にしてください。

(3)　ユーザ定義関数の登録

httpdfunc.c 内にユーザ定義関数を登録します。

```
/* cnet_httpd.h に記述している構造体 cnet_finfo の宣言*/
    typedef struct {
        const char      *fn;                                                    /*  関数識別名  */
        void            (*fi)(int id, const char *name, const char *value);     /*  関数ポインタ  */
    } cnet_finfo;
=== httpdfunc.c ============================================================
/* httpdfunc.c に記述するユーザ定義関数の定義と httpd_flist の例*/
    static void
    a(int id, const char *name, const char *value)
    {
        ～
    }

    static void
    b(int id, const char *name, const char *value)
    {
        ～
    }

    static void
    c(int id, const char *name, const char *value)
    {
        ～
    }

    static void
    func1(int id, const char *name, const char *value)
    {
        ～
    }

    const cnet_finfo    httpd_flist[] = {
        {"a", a},
        {"b", b},
        {"c", c},
        {"func1", func1},
        {NULL, NULL}
    };
=============================================================================
```

httpd_flist 構造体にユーザ定義関数識別名と関数ポインタを設定します。関数識別名と関数ポインタに NULL の組み合わせを指定することにより、登録の終わりを判断します。ですので、NULL、NULL の組み合わせの記述を最後に必ず記載してください。

【指定例】

- POST メソッドでの指定例

```
POST /cgi/test.cgi HTTP/1.0
Content-Length: 12
func1=param1
```

・ CGI での指定例

c func1 param1

・ 関数の呼び出し

func1(id, “func1”, “param1”);

CGI の各メソッドの使い方とユーザ関数との呼び出し方については、Web アプリケーションの添付サンプルプログラムを参考にしてください。

2.4.5 Web ファイル登録ツールの使用方法(GMakefsdata)

HTTP サーバが扱うファイルについては、必ず以下の GMakefsdata.exe を用いて、fsdata.c に登録しておかなければなりません。

- ツール動作環境条件

OS:Windows2000 Professional、WindowsXP Professional

ファイルの登録は、以下の手順で行います。

① 任意の場所にフォルダ fs を作成します。

→添付サンプルの動作環境には、すでに¥fs、¥cgi が作成してあります。

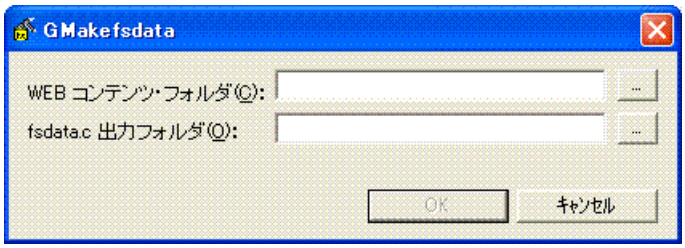
```
¥cnet
        ¥bin
            GMakefsdata.exe          .....ファイル登録ツール
        ¥fs
            (HTML ファイル)
            ¥cgi
                (CGI ファイル)
            ¥gif
                (GIF ファイル)
        ¥src
            ¥fs
                fsdata.c                 .....生成されるｺﾝﾃﾝﾂﾃﾞｰﾀﾌｧｲﾙ
```

② 作成した fs 配下に Web ページを構成する HTML ファイル、CGI ファイル、画像ファイル等を作成します。

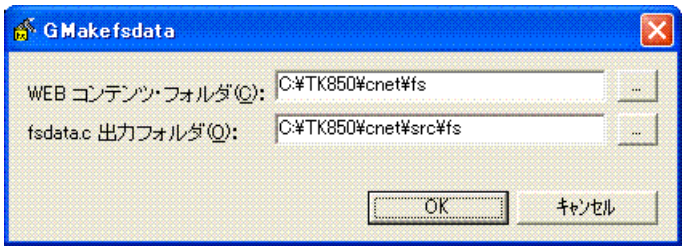
フォルダの階層は任意に作成できますが、CGI ファイルだけは fs¥cgi 配下に作成しなければ

なりません。

③　フォルダ fs 配下のファイルをすべて作成し終わったら、GMakefsdata.exe を実行します。
GMakefsdata.exe を実行すると、以下の画面が表示されます。



④　作成したフォルダ fs の場所と fsdata.c の出力先を指定して OK ボタンを押下します。
OK ボタンを押下すると、コンテンツ・ファイルの登録を開始します。



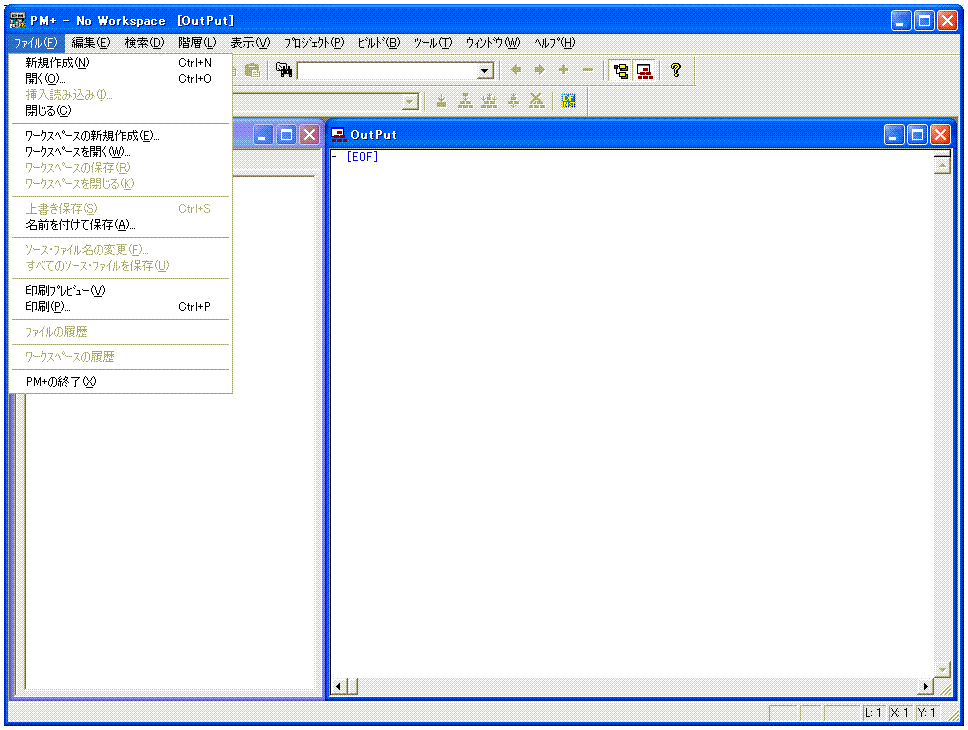
⑤　コンテンツ・ファイルの登録が終わると、指定した出力先に fsdata.c が作成され、以下の画面が表示されます。OK ボタンを押下すると、GMakefsdata を終了します。

⑥　作業が完了したら、再ビルドを行います。
ビルド方法については、「2.5 実行モジュールの作成」の項を参照して下さい。

2.5　実行モジュールの作成

2.5.1　プログラムのビルド方法

(1)　　PM+を起動します。



(2)　　以下の構成に対するプロジェクトの例で説明します。

各プロジェクトが以下のフォルダにあるとします。

C:¥TK850¥JH3U_graphic¥cnet ……..ユーザのプログラム作成・格納用フォルダ

(「1.1.3 評価キットソフトウェアのインストール**エラー! 参照元が見つかりません。**」の項を参照して下さい。)

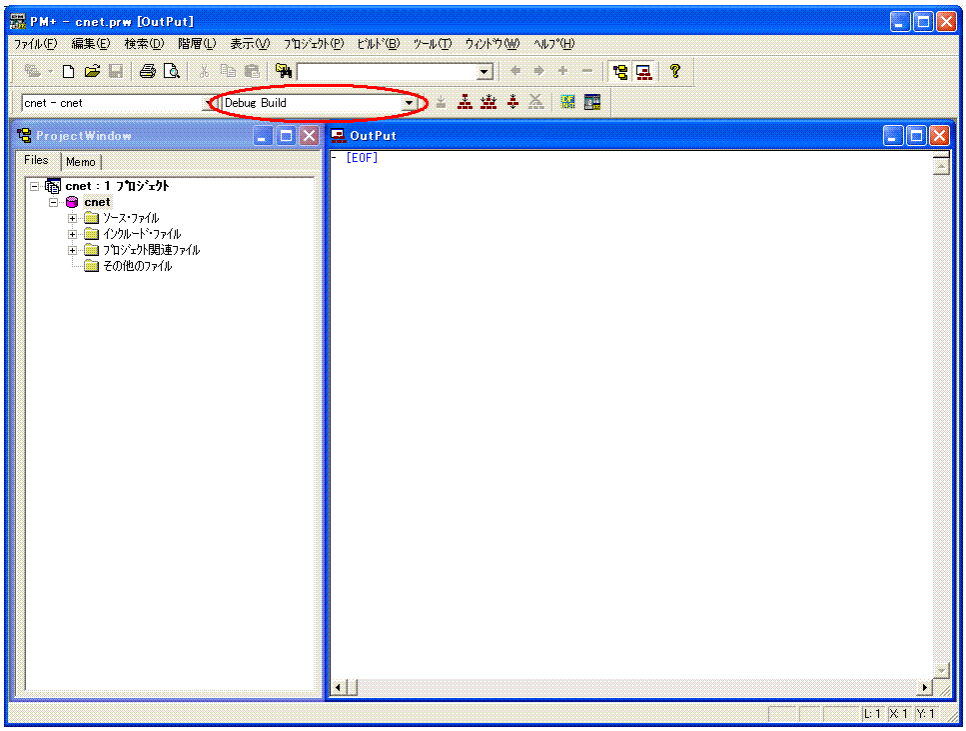
(3)　　ユーザプログラム関連オブジェクトをビルドします。

「ファイル」->「ワークスペースを開く」を選択し、ワークスペースファイルに、cnet フォルダにある、ether.prw ファイルを選択します。

cnet ワークスペースが開かれますので、ビルドを実行します。

ビルドには、ディバッグをおこなうことが出来る DebugBuild モードと、リリース形式のロードモジュールを作成する ReleaseBuild モードがあります。DebugBuild モード／ReleaseBuild モード の切り

替えは、PM+の画面で、以下の赤丸〇部分のリストから選択します。



(4)　　ビルドを実行すると、ユーザプログラム関連オブジェクトが作成されます。

2.5.2　生成される実行モジュール

ビルドが正常に終了すると、ロードモジュールファイルが生成されます。
（前述の例では c:¥TK850¥JH3U_graphic¥cnet¥romp.out）

2.6　ディバッグビルド物件の実行手順

DebugBuild モードにより生成したオブジェクトを、PM+を使ってディバッグ実行します。

評価ボードのSW1 の 1 をOFF、2～4 をONに設定[注]後、「ビルド」->「ディバグ」を選択し、ディバッガを起動します。ビルド後生成されたromp.outファイルをディバッガで開いてください。DebugBuildモードで作成されたロードモジュールがダウンロードされ、ディバグ実行が可能となります。必要に応じてBreakPointを設定し動作確認を行います。

詳細については、**「TK-850/JH3U-SP　仕様」**を参照して下さい。

注：ディップスイッチ切り替え後は、リセットするか電源を入れ直して下さい。

2.7　リリースビルド物件の FlashROM へのインストール

ReleaseBuild モードにより作成した ROM イメージを、評価ボードの FlashROM に書き込みます。

ReleaseBuildモード指定でプログラムをビルドし、フラッシュメモリ書き込みプログラムWriteEZ1 を使用してFlashROMにromp.hexを書き込みます。評価ボードのSW1の1～3をOFF、4をONに設定[注]すると、評価ボード単体で起動しても、書き込んだプログラムが実行されます。

詳細については、**「TK-850/JH3U-SP 仕様」**を参照して下さい。

注：ディップスイッチ切り替え後は、リセットするか電源を入れ直して下さい。

〔API 仕様編〕

3. API 仕様詳細

3.1 ネットワーク機能（TCP） API

3.1.1 tcp_new 関数

【書式】

struct tcp_pcb *tcp_new(void);

【機能】

TCP PCB の作成

【引数】

なし

【返却値】

成功時は、作成した TCP PCB を返却します。

エラーの場合は、NULL を返却します。

【発行有効範囲】

cnet_init 関数を実行してネットワークの初期化を行った後。

【解説】

TCP の PCB（protocol control block）を新たに作成します。

PCB は、ネットワークアプリケーションのプロトコル制御に使用します。

3.1.2 tcp_close 関数

【書式】

err_t tcp_close(struct tcp_pcb *pcb);

【機能】

TCP のコネクション切断と TCP PCB の削除

【引数】

struct tcp_pcb *pcb(i)　　TCP PCB

【返却値】

ERR_OK　　成功

ERR_MEM　　メモリエラー

ERR_BUF　　バッファエラー

【発行有効範囲】

tcp_new 関数で TCP PCB を作成した後。

【解説】

コネクションが確立している場合は、コネクションの切断が終了してから TCP PCB を削除します。コネクションが確立していない場合は、直ちに TCP PCB を削除します。

3.1.3 tcp_abort 関数

【書式】

void tcp_abort(struct tcp_pcb *pcb);

【機能】

RST の送信による TCP のコネクション中断と TCP PCB の削除

【引数】

struct tcp_pcb *pcb(i)	TCP PCB

【返却値】

なし

【発行有効範囲】

tcp_new 関数で TCP PCB を作成した後。

【解説】

コネクションを強制的に切断し、TCP PCB を削除します。

TCP アプリケーションのエラー終了時に使用します。

3.1.4 tcp_bind 関数

【書式】

err_t tcp_bind(struct tcp_pcb *pcb, struct ip_addr *ipaddr, u16_t port);

【機能】

ローカルのポート番号と IP アドレスの設定

【引数】

struct tcp_pcb *pcb(i)	TCP PCB
struct ip_addr *ipaddr(i)	IP アドレス
u16_t port(i)	ポート番号

【返却値】

ERR_OK	成功
ERR_USE	指定されたポート番号と IP アドレスは使用中

【発行有効範囲】

tcp_new 関数で TCP PCB を作成した後。

【解説】

ローカルのポート番号と IP アドレスを TCP PCB へ割り当てます。

ポート番号に 0 を指定した場合は、4,096 から 32,767 の間で空いている番号を使用します。

3.1.5 tcp_listen 関数

【書式】

struct tcp_pcb *tcp_listen(struct tcp_pcb *pcb);

【機能】

接続要求を受け入れることができる TCP PCB の作成

【引数】

struct tcp_pcb *pcb(i)　　TCP PCB

【返却値】

成功時は、接続要求を受け入れることができる TCP PCB を返却します。

エラーの場合は、NULL を返却します。

【発行有効範囲】

tcp_new 関数で TCP PCB を作成した後。

【解説】

接続要求を受け入れることができる TCP PCB を新しく作成し、指定された TCP PCB を削除します。指定された TCP PCB が接続要求を受け入れることができる場合は、指定された TCP PCB をそのまま返却します。

3.1.6 tcp_connect 関数

【書式】

err_t tcp_connect(struct tcp_pcb *pcb, struct ip_addr *ipaddr, u16_t port,
err_t (* connected)(void *arg, struct tcp_pcb *tpcb, err_t err));

【機能】

接続要求の送信

【引数】

struct tcp_pcb *pcb(i)　　TCP PCB

struct ip_addr *ipaddr(i)　　接続先の IP アドレス

u16_t port(i)　　接続先のポート番号

err_t (* connected)(void *arg, struct tcp_pcb *tpcb, err_t err)(i)
接続されたときに呼ばれるコールバック関数。

【返却値】

ERR_OK　　成功

ERR_VAL　　引数の値が不正

ERR_MEM　　メモリエラー

【発行有効範囲】

tcp_new 関数で TCP PCB を作成した後。

【解説】

tcp_connect 関数は、接続要求を送信したら終了するので、関数の終了時点では、コネクションが確立していません。コネクション確立後の処理は、引数 connected に指定するコールバック関数で定義します。コネクション確立が失敗した場合は、tcp_err 関数で登録したコールバック関数が呼ばれます。

※コールバック関数の引数の説明

void *arg(i)　　tcp_arg 関数で登録されたデータ

struct tcp_pcb *tpcb(i)　TCP PCB

err_t err(i)　　エラーコード

3.1.7　tcp_accept 関数

【書式】

void tcp_accept(struct tcp_pcb *pcb,

err_t (* accept)(void *arg, struct tcp_pcb *newpcb, err_t err));

【機能】

接続要求受け入れ時に呼ばれるコールバック関数の登録

【引数】

struct tcp_pcb *pcb(i)　　接続要求を受け入れることができる TCP PCB

err_t (* accept)(void *arg, struct tcp_pcb *newpcb, err_t err)(i)

接続要求を受け入れたときに呼ばれるコールバック関数

【返却値】

なし

【発行有効範囲】

tcp_listen()関数で接続要求を受け入れることができる TCP PCB を作成した後。

【解説】

接続要求を受け入れた時に呼ばれるコールバック関数を登録します。

接続要求を受け入れてコネクションが確立したあとの処理をこのコールバック関数で定義します。

※コールバック関数の引数の説明

void *arg(i)	tcp_arg 関数で登録されたデータ
struct tcp_pcb *newpcb(i)	コネクションが確立された状態の TCP PCB
err_t err(i)	エラーコード

3.1.8 tcp_recv 関数

【書式】

void tcp_recv(struct tcp_pcb *pcb,
err_t (* recv)(void *arg, struct tcp_pcb *tpcb, struct pbuf *p, err_t err));

【機能】

データ受信時に呼ばれるコールバック関数の登録

【引数】

struct tcp_pcb *pcb(i)	TCP PCB
err_t (* recv)(void *arg, struct tcp_pcb *tpcb, struct pbuf *p, err_t err)(i)	データ受信時に呼ばれるコールバック関数

【返却値】

なし

【発行有効範囲】

tcp_new 関数で TCP PCB を作成した後。

【解説】

データを受信した時に呼ばれるコールバック関数を登録します。
受信したデータの処理をこのコールバック関数で定義します。
※コールバック関数の引数の説明

void *arg(i)	tcp_arg 関数で登録されたデータ
struct tcp_pcb *tpcb(i)	TCP PCB
struct pbuf *p(i)	受信したデータ
err_t err(i)	エラーコード

3.1.9 tcp_sent 関数

【書式】

void tcp_sent(struct tcp_pcb *pcb,
err_t (* sent)(void *arg, struct tcp_pcb *tpcb, u16_t len));

【機能】

データ送信成功時に呼ばれるコールバック関数の登録

【引数】

struct tcp_pcb *pcb(i)	TCP PCB
err_t (* sent)(void *arg, struct tcp_pcb *tpcb, u16_t len)(i)	データ送信成功時に呼ばれるコールバック関数

【返却値】

なし

【発行有効範囲】

tcp_new 関数で TCP PCB を作成した後。

【解説】

データ送信が成功した時に呼ばれるコールバック関数を登録します。

データの送信処理の継続や、送信完了の確認を行いたい場合に使用します。

※コールバック関数の引数の説明

void *arg(i)	tcp_arg 関数で登録されたデータ
struct tcp_pcb *tpcb(i)	TCP PCB
u16_t len(i)	送信したバイト数

3.1.10 tcp_poll 関数

【書式】

void tcp_poll(struct tcp_pcb *pcb,
err_t (* poll)(void *arg, struct tcp_pcb *tpcb), u8_t interval);

【機能】

定期的に呼ばれるコールバック関数の登録

【引数】

struct tcp_pcb *pcb(i)	TCP PCB
err_t (* poll)(void *arg, struct tcp_pcb *tpcb)(i)	定期的に呼ばれるコールバック関数
u8_t interval(i)	コールバック関数が呼ばれる間隔（500 ミリ秒単位）

【返却値】

なし

【発行有効範囲】

tcp_new 関数で TCP PCB を作成した後。

【解説】

定期的に呼ばれるコールバック関数を登録します。

一定の間隔で行いたい処理がある場合に使用します。

※コールバック関数の引数の説明

void *arg(i)	tcp_arg 関数で登録されたデータ
struct tcp_pcb *tpcb(i)	TCP PCB

3.1.11 tcp_err 関数

【書式】

void tcp_err(struct tcp_pcb *pcb, void (* errf)(void *arg, err_t err));

【機能】

通信エラー発生時に呼ばれるコールバック関数の登録

【引数】

struct tcp_pcb *pcb(i)	TCP PCB
void (* errf)(void *arg, err_t err)(i)	通信エラー発生時に呼ばれるコールバック関数

【返却値】

なし

【発行有効範囲】

tcp_new 関数で TCP PCB を作成した後。

【解説】

コネクションの確立、コネクションの切断、データ送信でエラーになった場合に呼ばれるコールバック関数を登録します。

※コールバック関数の引数の説明

void *arg(i)	tcp_arg 関数で登録されたデータ
err_t err(i)	発生したエラーコード

3.1.12 tcp_arg 関数

【書式】

void tcp_arg(struct tcp_pcb *pcb, void *arg);

【機能】

コールバック関数へ渡すデータの登録

【引数】

struct tcp_pcb *pcb(i)	TCP PCB
void *arg(i)	コールバック関数へ渡すデータ

【返却値】

なし

【発行有効範囲】

tcp_new 関数で TCP PCB を作成した後。

【解説】

登録したデータは、tcp_connect、tcp_accept、tcp_recv、tcp_sent、tcp_poll、tcp_err の各関数で登録されたコールバック関数が呼ばれるときに、第1引数 arg に渡されます。

3.1.13 tcp_write 関数

【書式】

err_t tcp_write(struct tcp_pcb *pcb, const void *arg, u16_t len, u8_t copy);

【機能】

送信データをキューへ格納

【引数】

struct tcp_pcb *pcb(i)	TCP PCB	
const void *arg(i)	送信するデータ	
u16_t len(i)	送信するデータのバイト数	
u8_t copy(i)	非 0	…送信データのコピーを作成します。
	0	…送信データのコピーを作成しません。

【返却値】

ERR_OK	成功
ERR_CONN	コネクションが確立していない
ERR_MEM	メモリエラー

【発行有効範囲】

コネクションが確立した後。

【解説】

引数 arg で指定されたデータを送信キューへ格納します。送信キューへ格納されたデータは、適宜送信が行われます。引数 copy に 0 以外が指定された場合は、新たに確保した領域に送信データをコピーし、その領域を送信キューへ格納します。新たに領域を確保できなかった場合は、ERR_MEM を返却します。引数 copy に 0 が指定された場合は、コピーを作成せずに引数 arg で指定された領域を参照します。

3.1.14 tcp_output 関数

【書式】

err_t tcp_output(struct tcp_pcb *pcb);

【機能】

未送信データの送信

【引数】

struct tcp_pcb *pcb(i)　　TCP PCB

【返却値】

ERR_OK　　成功

ERR_BUF　　バッファエラー

【発行有効範囲】

コネクションが確立した後。

【解説】

送信キューに存在する未送信のデータを送信します。

通常は、tcp_write 関数で送信キューへ格納すればデータの送信が行われます。送信キューのデータを強制的に送信したい場合にこの関数を使用します。

3.1.15 tcp_recved 関数

【書式】

void tcp_recved(struct tcp_pcb *pcb, u16_t len);

【機能】

受信完了の通知

【引数】

struct tcp_pcb *pcb(i)　　TCP PCB

u16_t len(i)　　受信完了したデータのサイズ（バイト数）

【返却値】

なし

【発行有効範囲】

tcp_new 関数で TCP PCB を作成した後。

【解説】

tcp_recv 関数で登録した受信コールバック関数で、引数 p で受け取った受信データを解放するときに、解放する受信データのサイズを通知します。

この関数は、受信データを解放したときに必ず呼び出す必要があります。

3.1.16 tcp_setprio 関数

【書式】

void tcp_setprio(struct tcp_pcb *pcb, u8_t prio);

【機能】

プライオリティの設定

【引数】

struct tcp_pcb *pcb(i)　　TCP PCB

u8_t prio(i)　　プライオリティ

【返却値】

なし

【発行有効範囲】

tcp_new 関数で TCP PCB を作成した後。

【解説】

TCP PCB のリソースが足りなくなったときに、プライオリティが高い TCP PCB が優先的に保持されます。プライオリティは、1～127 の値を指定でき、値が大きいほどプライオリティが高くなります。プライオリティの標準値は 64 です。

3.1.17 tcp_sndbuf 関数

【書式】

u16_t tcp_sndbuf(struct tcp_pcb *pcb);

【機能】

使用可能な送信バッファ領域（バイト数）の取得

【引数】

struct tcp_pcb *pcb(i)　　TCP PCB

【返却値】

使用可能な送信バッファ領域（バイト数）

【発行有効範囲】

tcp_new 関数で TCP PCB を作成した後。

【解説】

tcp_write 関数は、使用可能な送信バッファ領域より大きいサイズのデータを指定するとエラーになります。tcp_sndbuf 関数で使用可能な送信バッファ領域を調べて、tcp_write 関数でそれより大きいデータを指定しないようにすればエラーにならずに済みます。

3.2 ネットワーク機能（UDP） API

3.2.1 udp_new 関数

【書式】

struct udp_pcb *udp_new(void);

【機能】

UDP PCB の作成

【引数】

なし

【返却値】

成功時は、作成した UDP PCB を返却します。

エラーの場合は、NULL を返却します。

【発行有効範囲】

cnet_init 関数を実行してネットワークの初期化を行った後。

【解説】

UDP の PCB（protocol control block）を新たに作成します。

PCB は、ネットワークアプリケーションのプロトコル制御に使用します。

3.2.2 udp_remove 関数

【書式】

void udp_remove(struct udp_pcb *pcb);

【機能】

UDP PCB の削除

【引数】

struct udp_pcb *pcb(i)　　UDP PCB

【返却値】

なし

【発行有効範囲】

udp_new 関数で UDP PCB を作成した後。

【解説】

指定された UDP PCB を削除します。

3.2.3 udp_bind 関数

【書式】

err_t udp_bind(struct udp_pcb *pcb, struct ip_addr *ipaddr, u16_t port);

【機能】

ローカルのポート番号と IP アドレスの設定

【引数】

struct udp_pcb *pcb(i)　　UDP PCB

struct ip_addr *ipaddr(i)　IP アドレス

u16_t port(i)　ポート番号

【返却値】

ERR_OK　成功

ERR_USE　指定されたポート番号と IP アドレスは使用中

【発行有効範囲】

udp_new 関数で UDP PCB を作成した後。

【解説】

ローカルのポート番号と IP アドレスを UDP PCB へ割り当てます。

ポート番号に 0 を指定した場合は、4,096 から 32,767 の間で空いている番号を使用します。

3.2.4 udp_connect 関数

【書式】

err_t udp_connect(struct udp_pcb *pcb, struct ip_addr *ipaddr, u16_t port);

【機能】

リモートのポート番号と IP アドレスの設定

【引数】

struct udp_pcb *pcb(i)　UDP PCB

struct ip_addr *ipaddr(i)　リモートの IP アドレス

u16_t port(i)　リモートのポート番号

【返却値】

ERR_OK　成功

【発行有効範囲】

udp_new 関数で UDP PCB を作成した後。

【解説】

リモートのポート番号と IP アドレスを UDP PCB へ設定します。

3.2.5 udp_disconnect 関数

【書式】

void udp_disconnect(struct udp_pcb *pcb);

【機能】

リモートのポート番号と IP アドレスの設定解除

【引数】

struct udp_pcb *pcb(i)　UDP PCB

【返却値】

なし。

【発行有効範囲】

udp_connect 関数でリモートアドレスの設定を行った後。

【解説】

udp_connect 関数で行ったリモートのポート番号と IP アドレスの設定を解除します。

3.2.6 udp_recv 関数

【書式】

```
void udp_recv(struct udp_pcb *pcb,
              void (* recv)(void *arg, struct udp_pcb *upcb,
                            struct pbuf *p, struct ip_addr *addr, u16_t port),
              void *recv_arg);
```

【機能】

データ受信時に呼ばれるコールバック関数の登録

【引数】

struct udp_pcb *pcb(i)	UDP PCB
void (* recv)(void *arg, struct udp_pcb *upcb, struct pbuf *p, struct ip_addr *addr, u16_t port)(i)	データ受信時に呼ばれるコールバック関数
void *recv_arg(i)	コールバック関数の引数 arg へ渡すデータ

【返却値】

なし

【発行有効範囲】

udp_new 関数で UDP PCB を作成した後。

【解説】

データを受信した時に呼ばれるコールバック関数を登録します。

受信したデータの処理をこのコールバック関数で定義します。

※コールバック関数の引数の説明

void *arg(i)	recv_arg に指定したデータ
struct udp_pcb *upcb(i)	UDP PCB
struct pbuf *p(i)	受信データ
struct ip_addr *addr(i)	データ送信元の IP アドレス
u16_t port(i)	データ送信元のポート番号

3.2.7 udp_send 関数

【書式】

err_t udp_send(struct udp_pcb *pcb, struct pbuf *p);

【機能】

データの送信

【引数】

struct udp_pcb *pcb(i)	UDP PCB
struct pbuf *p(i)	送信するデータ。

【返却値】

ERR_OK	成功
ERR_MEM	メモリエラー
ERR_BUF	バッファエラー
ERR_RTE	ルーティング失敗
ERR_CONN	通信エラー

【発行有効範囲】

udp_connect 関数でリモートアドレスの設定を行った後。

【解説】

指定されたデータを送信します。送信先は、udp_connect 関数で設定します。

3.3 ネットワーク機能 独自 API

3.3.1 cnet_init 関数

【書式】

void cnet_init (void);

【機能】

ネットワークの初期化

【引数】

なし

【返却値】

なし

【発行有効範囲】

ネットワークを使用する場合に、最初に呼び出します。

【解説】

TCP/IP ネットワーク用のネットワークデータバッファなどの初期化を行います。

3.3.2 netif_add 関数

【書式】

struct netif * netif_add(struct ip_addr *ip_addr,struct ip_addr *netmask,
struct *ip_addr *gw ,void *state,
err_t (*init)(struct netif *netif),
err_t (*input)(struct pbuf,struct netif *netif));

【機能】

ネットワーク・インタフェースの作成

【引数】

struct ip_addr *ip_addr(i)	IP アドレス
struct ip_addr *netmask(i)	サブネット・マスク
struct *ip_addr *gw(i)	ゲートウェイ・アドレス
void *state(i)	（NULL を指定）
err_t (*init)(struct netif *netif) (i)	ネットワークドライバ初期化関数
err_t (*input)(struct pbuf,struct netif *netif) (i)	TCP/IP 受信用関数(ip_input)

【返却値】

成功時は、作成したネットワーク･インタフェースを返却します。
エラーの場合は、NULL を返却します。

【発行有効範囲】

ネットワークの初期化後。

【解説】

ネットワーク・インタフェースを作成し、ネットワーク情報を登録後、引数 init で指定されたネットワーク・ドライバ初期化関数を実行します。引数 state には NULL を指定します。TCP/IP 受信用関数は、ip_input 関数を指定します。

3.3.3 netif_remove 関数

【書式】

void netif_remove(struct netif *netif);

【機能】

ネットワークインタフェースの削除

【引数】

struct netif *netif (i)	ネットワーク・インタフェース

【返却値】

なし

【発行有効範囲】

ネットワーク・インタフェースの作成後。

【解説】

指定されたネットワーク・インタフェースを削除します。

3.3.4 netif_set_default 関数

【書式】

void netif_set_default(struct netif *netif);

【機能】

デフォルトのネットワーク・インタフェースの登録

【引数】

struct netif *netif (i)　　ネットワーク・インタフェース

【返却値】

なし

【発行有効範囲】

ネットワーク・インタフェースの作成後。

【解説】

指定されたネットワーク・インタフェースをデフォルトのネットワーク・インタフェースとして登録します。

3.3.5 netif_set_addr 関数

【書式】

void netif_set_addr(struct netif *netif, struct ip_addr *ipaddr,
struct ip_addr *netmask, struct ip_addr *gw);

【機能】

IP アドレスの設定

【引数】

struct netif *netif(i)　　ネットワーク・インタフェース
struct ip_addr *ipaddr(i)　IP アドレス
struct ip_addr *netmask(i)　サブネット・マスク
struct ip_addr *gw(i)　　ゲートウェイ・アドレス

【返却値】

なし

【発行有効範囲】

ネットワーク・インタフェースの作成後。

【解説】

指定されたネットワーク・インタフェースへ IP アドレス、サブネット・マスク、ゲートウェイ・アドレスを設定します。

3.3.6 cnet_single_routine 関数

【書式】

void cnet_single_routine();

【機能】

ネットワーク処理

【引数】

なし

【返却値】

なし

【発行有効範囲】

ネットワーク設定後。

【解説】

ネットワークの受信処理を行い、受信結果に応じてネットワーク制御処理、TCP、UDP の登録済みコールバック関数の呼び出しを行います。TCP、DHCP のタイマー処理も行います。

3.3.7 cnet_getipaddr 関数

【書式】

int cnet_getipaddr(struct netif *netif);

【機能】

IP アドレスの取得

【引数】

struct netif *netif(i)　　ネットワーク・インタフェース

【返却値】

IP アドレス（バイトオーダ形式）

【発行有効範囲】

ネットワーク・インタフェースの作成後。

【解説】

指定されたネットワーク・インタフェースの IP アドレス（バイトオーダ形式）を返却します。

3.3.8 cnet_getnetmask 関数

【書式】

int cnet_getnetmask(struct netif *netif);

【機能】

サブネット・マスクの取得

【引数】

struct netif *netif(i)　　ネットワーク・インタフェース

【返却値】

サブネット・マスク（バイトオーダ形式）。

【発行有効範囲】

ネットワーク・インタフェースの作成後。

【解説】

指定されたネットワーク・インタフェースのサブネット・マスク（バイトオーダ形式）を返却します。

3.3.9 cnet_getgwaddr 関数

【書式】

int cnet_getgwaddr(struct netif *netif);

【機能】

ゲートウェイ・アドレスの取得

【引数】

struct netif *netif(i)　　ネットワーク・インタフェース

【返却値】

ゲートウェイ・アドレス（バイトオーダ形式）。

【発行有効範囲】

ネットワーク・インタフェースの作成後。

【解説】

指定されたネットワーク・インタフェースのゲートウェイ・アドレス（バイトオーダ形式）を返却します。

3.3.10 inet_addr 関数

【書式】

u32_t　inet_addr(const char *cp);

【機能】

ドット形式の IP アドレスをバイトオーダ形式へ変換

【引数】

const char *cp(i)　ドット形式の IP アドレス

【返却値】

バイトオーダ形式の IP アドレス

【発行有効範囲】

特になし。

【解説】

ドット形式の IP アドレスをバイトオーダ形式へ変換します。

3.3.11 inet_ntoa 関数

【書式】

u8_t　*inet_ntoa(u32_t addr);

【機能】

バイトオーダ形式の IP アドレスをドット形式へ変換

【引数】

u32_t addr(i)　　バイトオーダ形式の IP アドレス

【返却値】

ドット形式の IP アドレス

【発行有効範囲】

特になし。

【解説】

バイトオーダ形式の IP アドレスをドット形式へ変換します。

3.3.12 htonl 関数

【書式】

u32_t　htonl(u32_t n);

【機能】

ホストバイトオーダーの 32 ビット値をネットワークバイトオーダーへ変換

【引数】

u32_t n(i)　　ホストバイトオーダーの 32 ビット値

【返却値】

ネットワークバイトオーダーの 32 ビット値。

【発行有効範囲】

特になし。

【解説】

ホストバイトオーダーの 32 ビット値をネットワークバイトオーダーへ変換します。

3.3.13 htons 関数

【書式】

u16_t htons(u16_t n);

【機能】

ホストバイトオーダーの 16 ビット値をネットワークバイトオーダーへ変換

【引数】

u16_t n(i) ホストバイトオーダーの 16 ビット値

【返却値】

ネットワークバイトオーダーの 16 ビット値。

【発行有効範囲】

特になし。

【解説】

ホストバイトオーダーの 16 ビット値をネットワークバイトオーダーへ変換します。

3.3.14 ntohl 関数

【書式】

u32_t ntohl(u32_t n);

【機能】

ネットワークバイトオーダーの 32 ビット値をホストバイトオーダーへ変換

【引数】

u32_t n(i) ネットワークバイトオーダーの 32 ビット値

【返却値】

ホストバイトオーダーの 32 ビット値。

【発行有効範囲】

特になし。

【解説】

ネットワークバイトオーダーの 32 ビット値をホストバイトオーダーへ変換します。

3.3.15 ntohs 関数

【書式】

u16_t ntohs(u16_t n);

【機能】

ネットワークバイトオーダーの 16 ビット値をホストバイトオーダーへ変換

【引数】

u16_t n(i) ネットワークバイトオーダーの 16 ビット値

【返却値】

ホストバイトオーダーの 16 ビット値。

【発行有効範囲】

特になし。

【解説】

ネットワークバイトオーダーの 16 ビット値をホストバイトオーダーへ変換します。

3.4 アプリケーション layerAPI (SMTP、POP3、DHCP、HTTP)

3.4.1 cnet_smtpinit 関数

【書式】

int cnet_smtpinit(unsigned int addr, unsigned short port, const char* domain);

【機能】

SMTP 機能の開始

【引数】

unsigned int addr(i)	受信用メールサーバの IP アドレス。
unsigned short port(i)	受信用メールサーバのポート番号。 0 が指定された場合は、25 を使用します。
const char* domain(i)	ドメイン名。

【返却値】

正常時 :

SMTP_ERROR_NORMAL	0x00000000

エラー時:

SMTP_ERROR_PARAMETER	0xFFFFFFFF
SMTP_ERROR_STATE	0xFFFFFFFE
SMTP_ERROR_CONNECT	0xFFFFFFF8

SMTP_ERROR_NOBUF 0xFFFFFF80

【発行有効範囲】

SMTP 機能を使用する場合に、最初に呼び出します。

【解説】

領域初期化、セッション開始、HELO コマンド実行を行います。

3.4.2 cnet_smtpend 関数

【書式】

int cnet_smtpend(unsigned char mode);

【機能】

SMTP 機能の終了

【引数】

unsigned char mode(i) 終了処理モード。
0: 正規の手続きにより終了します。
1: 強制終了します。

【返却値】

正常時 :

SMTP_ERROR_NORMAL 0x00000000

エラー時:

SMTP_ERROR_STATE 0xFFFFFFFE
SMTP_ERROR_REPLY 0xFFFFFFFC
SMTP_ERROR_DISCONNECT 0xFFFFFFF0
SMTP_ERROR_READ 0xFFFFFFE0
SMTP_ERROR_WRITE 0xFFFFFFC0

【発行有効範囲】

SMTP 機能を終了するときに呼び出します。

【解説】

正規の手続きでは、QUIT コマンドを実行して応答を受け取ってからセッションを終了します。
強制終了の場合は、QUIT コマンドを実行せずに強制的にセッションを終了します。

3.4.3 cnet_smtpdatafirst 関数

【書式】

int cnet_smtpdatafirst(const char* sendaddr, const char* recvaddr,
const char* databuf, unsigned short datalen);

【機能】

メールデータの送信開始

【引数】

const char* sendaddr(i)	送信元のメールアドレス。
const char* recvaddr(i)	送信先のメールアドレス。
const char* databuf(i)	送信するメールデータ。
unsigned short datalen(i)	送信するメールデータのサイズ（バイト数）。

【返却値】

正常時 :

SMTP_ERROR_NORMAL	0x00000000

エラー時:

SMTP_ERROR_PARAMETER	0xFFFFFFFF
SMTP_ERROR_STATE	0xFFFFFFFE
SMTP_ERROR_REPLY	0xFFFFFFFC
SMTP_ERROR_DISCONNECT	0xFFFFFFF0
SMTP_ERROR_READ	0xFFFFFFE0
SMTP_ERROR_WRITE	0xFFFFFFC0

【発行有効範囲】

メールを送信する場合に、cnet_smtpinit 関数の後に呼び出します。

【解説】

MAIL、RCPT、DATA の各コマンドの実行と、指定されたメールデータの送信を行います。
メールデータ中のピリオドについては、終了文字シーケンスと区別するための処理を行います。
すなわち、メールデータ中の行の先頭文字をチェックし、ピリオドの場合は、もうひとつピリオドを追加して送信します。

3.4.4 cnet_smtpdatanext 関数

【書式】

int cnet_smtpdatanext(const char* databuf, unsigned short datalen);

【機能】

メールデータの続きの送信

【引数】

const char* databuf(i)	送信するメールデータ。
unsigned short datalen(i)	送信するメールデータのサイズ(バイト数)。

【返却値】

正常時 :

SMTP_ERROR_NORMAL	0x00000000

エラー時:

SMTP_ERROR_PARAMETER	0xFFFFFFFF
SMTP_ERROR_STATE	0xFFFFFFFE
SMTP_ERROR_DISCONNECT	0xFFFFFFF0
SMTP_ERROR_READ	0xFFFFFFE0
SMTP_ERROR_WRITE	0xFFFFFFC0

【発行有効範囲】

メールを送信する場合に、cnet_smtdatafirst 関数の後に続けて呼び出します。

cnet_smtdatafirst 関数は、メール送信のうち必要なコマンドの発行まで行うため、さらにメールデータを送る場合には、cnet_smtpdatanext 関数を呼び出さなければなりません。

【解説】

指定されたメールデータの送信を行います。

メールデータ中のピリオドについては、終了文字シーケンスと区別するための処理を行います。すなわち、メールデータ中の行の先頭文字をチェックし、ピリオドの場合は、もうひとつピリオドを追加して送信します。

cnet_smtpdatafirst()に続いて cnet_smtpdatanext()を任意回数実行することにより、メールデータを分割して送信することができます。

3.4.5 cnet_smtpdataend 関数

【書式】

int cnet_smtpdataend();

【機能】

メールデータの送信終了

【引数】

なし。

【返却値】

正常時 :

SMTP_ERROR_NORMAL	0x00000000

エラー時:

SMTP_ERROR_STATE	0xFFFFFFFE
SMTP_ERROR_REPLY	0xFFFFFFFC
SMTP_ERROR_DISCONNECT	0xFFFFFFF0
SMTP_ERROR_READ	0xFFFFFFE0
SMTP_ERROR_WRITE	0xFFFFFFC0

【発行有効範囲】

メールデータの送信を終了する際に呼び出します。

【解説】

[<CRLF>].<CRLF>を送信します。

cnet_smtpdatafirst()、cnet_smtpdatanext()によるメールデータの送信が終了したときは、cnet_smtpdataend()を実行する必要があります。

3.4.6 cnet_pop3init 関数

【書式】

int cnet_pop3init(unsigned int addr, unsigned short port);

【機能】

POP3 機能の開始

【引数】

unsigned int addr(i)	メールサーバの IP アドレス。
unsigned short port(i)	メールサーバのポート番号。 0 が指定された場合は、110 を使用します。

【返却値】

正常時 :

POP3_ERROR_NORMAL	0x00000000

エラー時:

POP3_ERROR_PARAMETER	0xFFFFFFFF
POP3_ERROR_STATE	0xFFFFFFFE
POP3_ERROR_CONNECT	0xFFFFFFF8
POP3_ERROR_NOBUF	0xFFFFFF80

【発行有効範囲】

POP3 機能を使用していない状態。

【解説】

領域初期化、セッション開始を行います。

3.4.7 cnet_pop3end 関数

【書式】

int cnet_pop3end(unsigned char mode);

【機能】

POP3 機能の終了

【引数】

unsigned char mode(i)　終了処理モード。
0: 正規の手続きにより終了します。
1: 強制終了します。

【返却値】

正常時 :

POP3_ERROR_NORMAL	0x00000000

エラー時:

POP3_ERROR_STATE	0xFFFFFFFE
POP3_ERROR_REPLY	0xFFFFFFFC
POP3_ERROR_DISCONNECT	0xFFFFFFF0
POP3_ERROR_READ	0xFFFFFFE0
POP3_ERROR_WRITE	0xFFFFFFC0

【発行有効範囲】

POP3機能の開始（cnet_pop3init）後で、POP3機能を終了(cnet_pop3end)するまでの間、かつメッセージの取得中ではない場合。

【解説】

正規の手続きでは、QUIT コマンドを実行して応答を受け取ってからセッションを終了します。
強制終了の場合は、QUIT コマンドを実行せずに強制的にセッションを終了します。

3.4.8 cnet_pop3auth 関数

【書式】

int cnet_pop3auth(const char* user, const char* pass);

【機能】

ユーザ名、パスワードによる認証

【引数】

const char* user(i)　ユーザ名。
const char* pass(i)　パスワード。

【返却値】

正常時 :

POP3_ERROR_NORMAL	0x00000000

エラー時:

POP3_ERROR_PARAMETER	0xFFFFFFFF
POP3_ERROR_STATE	0xFFFFFFFE
POP3_ERROR_REPLY	0xFFFFFFFC

POP3_ERROR_DISCONNECT	0xFFFFFFF0
POP3_ERROR_READ	0xFFFFFFE0
POP3_ERROR_WRITE	0xFFFFFFC0

【発行有効範囲】

POP3 機能の開始（cnet_pop3init）後で、POP3 機能を終了(cnet_pop3end)するまでの間。

【解説】

USER、PASS コマンドを実行します。

開始、終了以外の POP3 機能を実行する場合は、cnet_pop3auth 関数で認証を行わなければなりません。

3.4.9 cnet_pop3stat 関数

【書式】

int cnet_pop3stat(unsigned int* number, unsigned int* size);

【機能】

メール情報の取得

【引数】

unsigned int* number(o)	メール総数。
unsigned int* size(o)	メール合計サイズ。

【返却値】

正常時 :

POP3_ERROR_NORMAL	0x00000000

エラー時:

POP3_ERROR_PARAMETER	0xFFFFFFFF
POP3_ERROR_STATE	0xFFFFFFFE
POP3_ERROR_REPLY	0xFFFFFFFC
POP3_ERROR_DISCONNECT	0xFFFFFFF0
POP3_ERROR_READ	0xFFFFFFE0
POP3_ERROR_WRITE	0xFFFFFFC0

【発行有効範囲】

ユーザ認証（cnet_pop3auth）後で、POP3 機能を終了(cnet_pop3end)するまでの間、かつメッセージの取得中ではない場合。

【解説】

STAT コマンドを実行します。

3.4.10 cnet_pop3getsize 関数

【書式】

int cnet_pop3getsize(unsigned int msgno, unsigned int* size);

【機能】

メッセージサイズの取得

【引数】

unsigned int msgno(i)　　サイズを取得するメッセージの番号。

unsigned int* size(o)　　メッセージサイズ。

【返却値】

正常時 :

POP3_ERROR_NORMAL	0x00000000

エラー時:

POP3_ERROR_PARAMETER	0xFFFFFFFF
POP3_ERROR_STATE	0xFFFFFFFE
POP3_ERROR_REPLY	0xFFFFFFFC
POP3_ERROR_DISCONNECT	0xFFFFFFF0
POP3_ERROR_READ	0xFFFFFFE0
POP3_ERROR_WRITE	0xFFFFFFC0

【発行有効範囲】

ユーザ認証（cnet_pop3auth）後で、POP3 機能を終了(cnet_pop3end)するまでの間、かつメッセージの取得中ではない場合。

【解説】

LIST コマンドを実行します。

3.4.11 cnet_pop3getmsgfirst 関数

【書式】

int cnet_pop3getmsgfirst(unsigned int msgno, unsigned int maxline,
char* msgbuf, unsigned short msglen,
unsigned char rmode);

【機能】

メッセージの取得開始

【引数】

unsigned int msgno(i)　　取得するメッセージの番号。

unsigned int maxline(i)　　取得するメッセージの本体部分の処理最大行数。
0 が指定された場合は、全ての行を処理対象にします。

char* msgbuf(o)　　取得するメッセージの格納領域。取得サイズ+1 必要。
unsigned short msglen(i)　取得するメッセージのサイズ（バイト数）。
unsigned char rmode(i)　メッセージの取得処理モード。

以下の3つのいずれかを指定します。

POP3_MODE_TEXT:

引数 msglen で指定されたサイズ分のメッセージを取得します。

改行<CRLF>がある場合は、そこまでで取得を終了します。

POP3_MODE_TRUNC:

引数 msglen で指定されたサイズ分のメッセージを行単位で取得します。

取得行が指定されたサイズより大きい場合は、残りを切り捨てます。

POP3_MODE_BINARY:

引数 msglen で指定されたサイズ分のメッセージを取得します。

改行<CRLF>があっても取得を終了しません。

【返却値】

正常時　:

POP3_ERROR_NORMAL	0x00000000

メッセージ終了時:

POP3_ERROR_GETEND	0x00000001

エラー時:

POP3_ERROR_PARAMETER	0xFFFFFFFF
POP3_ERROR_STATE	0xFFFFFFFE
POP3_ERROR_REPLY	0xFFFFFFFC
POP3_ERROR_DISCONNECT	0xFFFFFFF0
POP3_ERROR_READ	0xFFFFFFE0
POP3_ERROR_WRITE	0xFFFFFFC0

【発行有効範囲】

ユーザ認証（cnet_pop3auth）後で、POP3 機能を終了(cnet_pop3end)するまでの間、かつメッセージの取得中ではない場合。

【解説】

引数 maxline の値が 0 の場合は RETR コマンド、0 以外の場合は TOP コマンドを実行します。メッセージの取得が正常に行われた結果、メッセージの最後まで取得していない場合は 0、メッセージの最後まで取得した場合は 1 を返却します。

3.4.12 cnet_pop3getmsgnext 関数

【書式】

int cnet_pop3getmsgnext(char* msgbuf, unsigned short msglen,
unsigned char rmode);

【機能】

メッセージの続きの取得

【引数】

char* msgbuf(o)　　取得するメッセージの格納領域。取得サイズ+1 必要。

unsigned short msglen(i)　取得するメッセージのサイズ（バイト数）。

unsigned char rmode(i)　メッセージの取得処理モード。

以下の3つのいずれかを指定します。

POP3_MODE_TEXT:

引数 msglen で指定されたサイズ分のメッセージを取得します。

改行<CRLF>がある場合は、そこまでで取得を終了します。

POP3_MODE_TRUNC:

引数 msglen で指定されたサイズ分のメッセージを行単位で取得します。

取得行が指定されたサイズより大きい場合は、残りを切り捨てます。

POP3_MODE_BINARY:

引数 msglen で指定されたサイズ分のメッセージを取得します。

改行<CRLF>があっても取得を終了しません。

【返却値】

正常時　:

POP3_ERROR_NORMAL	0x00000000

メッセージ終了時:

POP3_ERROR_GETEND	0x00000001

エラー時:

POP3_ERROR_PARAMETER	0xFFFFFFFF
POP3_ERROR_STATE	0xFFFFFFFE
POP3_ERROR_DISCONNECT	0xFFFFFFF0
POP3_ERROR_READ	0xFFFFFFE0
POP3_ERROR_WRITE	0xFFFFFFC0

【発行有効範囲】

cnet_pop3getmsgfirst 関数でメッセージの取得を開始後、cnet_pop3getmsgend 関数でメッセージの取得を終了するまでの間。

【解説】

メッセージの取得を開始する場合は cnet_pop3getmsgfirst 関数を使用します。

cnet_pop3getmsgfirst 関数は、メッセージ受信のためのコマンドを発行するため、メッセージの続きを取得する場合は、cnet_pop3getmsgfirst 関数ではなくcnet_pop3getmsgnext 関数を使

用します。

メッセージの取得が正常に行われた結果、メッセージの最後まで取得していない場合は 0、メッセージの最後まで取得した場合は 1 を返却します。

3.4.13 cnet_pop3getmsgend 関数

【書式】

int cnet_pop3getmsgend();

【機能】

メッセージの取得終了

【引数】

なし。

【返却値】

正常時 :

POP3_ERROR_NORMAL	0x00000000

エラー時:

POP3_ERROR_STATE	0xFFFFFFFE
POP3_ERROR_DISCONNECT	0xFFFFFFF0
POP3_ERROR_READ	0xFFFFFFE0
POP3_ERROR_WRITE	0xFFFFFFC0

【発行有効範囲】

cnet_pop3getmsgfirst 関数でメッセージの取得を開始後。

【解説】

cnet_pop3getmsgfirst 関数、cnet_pop3getmsgnext 関数で取得中のメッセージの終わりまでを読み込み、メッセージの取得を終了します。

メッセージの取得を終了するときは、cnet_pop3getmsgend 関数を実行しなければなりません。

3.4.14 cnet_pop3delete 関数

【書式】

int cnet_pop3delete(unsigned int msgno);

【機能】

メッセージの削除マークの設定

【引数】

unsigned int msgno(i)　　削除マークを付けるメッセージの番号。

【返却値】

正常時 :

POP3_ERROR_NORMAL	0x00000000

エラー時:

POP3_ERROR_PARAMETER	0xFFFFFFFF
POP3_ERROR_STATE	0xFFFFFFFE
POP3_ERROR_REPLY	0xFFFFFFFC
POP3_ERROR_DISCONNECT	0xFFFFFFF0
POP3_ERROR_READ	0xFFFFFFE0
POP3_ERROR_WRITE	0xFFFFFFC0

【発行有効範囲】

ユーザ認証（cnet_pop3auth）後で、POP3 機能を終了(cnet_pop3end)するまでの間、かつメッセージの取得中ではない場合。

【解説】

DELE コマンドを実行します。

削除マークを付けられたメッセージは、POP3 機能の終了時（cnet_pop3end 関数の実行時）に実際に削除されます。

3.4.15 cnet_pop3reset 関数

【書式】

int cnet_pop3reset();

【機能】

全メッセージの削除マークの設定取り消し

【引数】

なし

【返却値】

正常時 :

POP3_ERROR_NORMAL	0x00000000

エラー時:

POP3_ERROR_STATE	0xFFFFFFFE
POP3_ERROR_REPLY	0xFFFFFFFC
POP3_ERROR_DISCONNECT	0xFFFFFFF0
POP3_ERROR_READ	0xFFFFFFE0
POP3_ERROR_WRITE	0xFFFFFFC0

【発行有効範囲】

ユーザ認証（cnet_pop3auth）後で、POP3 機能を終了(cnet_pop3end)するまでの間、かつメッセージの取得中ではない場合。

【解説】

RSET コマンドを実行します。

cnet_pop3delete 関数で付けられたメッセージへの削除マークを全て取り消します。

3.4.16 cnet_pop3getuid 関数

【書式】

int cnet_pop3getuid(unsigned int msgno, char* uidbuf, unsigned short uidbuflen);

【機能】

メッセージの unique-id の取得

【引数】

unsigned int msgno(i)	unique-id を取得するメッセージの番号。
char* uidbuf(o)	取得した unique-id を格納する領域。
unsigned short uidbuflen(i)	uidbuf の領域サイズ（バイト数）。

【返却値】

正常時 :

POP3_ERROR_NORMAL	0x00000000

エラー時:

POP3_ERROR_PARAMETER	0xFFFFFFFF
POP3_ERROR_STATE	0xFFFFFFFE
POP3_ERROR_REPLY	0xFFFFFFFC
POP3_ERROR_DISCONNECT	0xFFFFFFF0
POP3_ERROR_READ	0xFFFFFFE0
POP3_ERROR_WRITE	0xFFFFFFC0
POP3_ERROR_NOBUF	0xFFFFFF80

【発行有効範囲】

ユーザ認証（cnet_pop3auth）後で、POP3 機能を終了(cnet_pop3end)するまでの間、かつメッセージの取得中ではない場合。

【解説】

UIDL コマンドを実行します。

3.4.17 cnet_dhcpstart 関数

【書式】

int cnet_dhcpstart(struct netif *netif);

【機能】

DHCP クライアント処理の開始

【引数】

struct netif *netif(i)　　ネットワーク・インタフェース

【返却値】

正常時 :

0

エラー時:

-1　作業領域不足

-8　指定されたパラメータが正しくない

-9　DHCP サーバへのアクセスに失敗した

【発行有効範囲】

ネットワークインタフェースの作成後。

【解説】

DHCP サーバを検出し、DHCP サーバより IP アドレス、ネットマスク値、ゲートウェイ・アドレスを取得します。取得した値は、ネットワーク･インタフェースに設定されます。

3.4.18 cnet_dhcpstop 関数

【書式】

void cnet_dhcpstop(struct netif *netif);

【機能】

DHCP クライアント処理の終了

【引数】

struct netif *netif(i)　　ネットワーク・インタフェース

【返却値】

なし

【発行有効範囲】

DHCP クライアントの処理開始後。

【解説】

DHCP クライアントの処理を終了します。

3.4.19 cnet_httpdinit 関数

【書式】

void cnet_httpdinit();

【機能】

HTTP サーバの開始

【引数】

なし

【返却値】

なし

【発行有効範囲】

ネットワークの初期化後。

【解説】

HTTP サーバを開始します。

3.4.20 cnet_httpdsend 関数

【書式】

int cnet_httpdsend(int id, const char *msg, int msglen, unsigned char copy);

【機能】

HTTP サーバ専用のデータ送信

【引数】

int id(i)	CGI 処理関数が受け取る ID。
const char *msg(i)	送信するデータ。
int msglen(i)	送信するデータのサイズ(バイト数)。
unsigned char copy(i)	送信データのコピー有無。

【返却値】

正常時 :

0

エラー時:

-1 作業領域不足

-8 指定されたパラメータが正しくない

【発行有効範囲】

HTTP サーバの CGI 処理関数内。

【解説】

HTTP サーバの CGI 処理関数内でデータを送信するために使用します。

copy には、送信データに指定した領域の内容が変わる可能性がある場合は 1、内容が変わらない定数等の場合は 0 を指定します。

3.4.21 cnet_htpdsendcont 関数

【書式】

void cnet_httpdsendcont(int id, void (*send)(int id));

【機能】

データ送信の継続処理関数の登録、登録解除

【引数】

int id(i)	情報管理用 ID 番号
void (*send)(int id)(i)	データ送信継続関数

【返却値】

なし

【発行有効範囲】

HTTP サーバの CGI 処理関数内。

【解説】

前回の続きのデータを送信する関数を、繰り返し呼ばれる関数として登録します。これによりデータの送信処理を複数回に分けて行うことができます。

登録された関数は、登録を解除しない限り繰り返し呼ばれ続けるため、データの送信が終わったら関数の登録を解除する必要があります。登録を解除するには、send に NULL を指定して cnet_httpdsendcont 関数を実行します。

4. トラブルシューティング

状況	考えられる原因	対処方法
・ネットワークがつながらない ・評価用 PC のブラウザで Web 画面が表示されない。	・IP アドレスが誤っている ・ネットマスクが誤っている	PC の IP アドレス、ネットマスクを正しく指定してください
	LAN ケーブルのクロス／ストレート種別が誤っている	PC とボードを直結する場合はクロス結線、Hub を介して接続する場合はストレート結線のケーブルを使用してください
	ブラウザの設定が誤っている	PC とボードをローカル接続している場合は、ダイヤルアップ、プロキシサーバ経由でアクセスを行わない設定にしてください。

5. 用語集

スループット

ネットワーク分野では、ネットワーク接続した端末間の通信速度を指す。例えば 2 台の端末間のファイル転送性能などです。スループットは、ビット／秒などで表すことができる。

RFC

インターネットに関する技術の標準を定める団体である IETF が正式に発行する文書で、IP(RFC 791)、TCP(RFC 793)などインターネットで利用されるプロトコルや、技術の仕様・要件。

パケット

データ通信において、送信先のアドレスなどの制御情報を付加されたデータの小さなまとまりのこと。

ポート番号

インターネット上の通信において、複数の相手と同時に接続を行うために IP アドレスの下に設けられた補助アドレス。

TCP や UDP の機能で、ネットワークから受信したパケットを受け渡すアプリケーションなどをポート番号で特定する。定められたポート番号には、例えば、Web サーバは 80、FTP は 21、などがある。

ボーレート

baud rate

アナログ通信回線における変復調速度の単位で、通信機器が 1 秒間に何回変復調を行えるかを表す。

Ethernet

IEEE 802.3 委員会によって標準化された LAN の規格。

TCP

インターネットで利用される標準プロトコルで、OSI 参照モデルのトランスポート層にあたり、ネットワーク層の IP と、セッション層以上のプロトコル(HTTP、FTP、SMTP、POP3など)の橋渡しをする。

IP

インターネットで利用される標準プロトコルで、OSI 基本参照モデルのネットワーク層に位置し、ネットワークに参加している機器のアドレッシングや、相互に接続された複数のネットワーク内での通信経路の選定をするための方式を定義している。

UDP

インターネットで利用される標準プロトコルで、OSI 参照モデルのトランスポート層にあたり、ネットワーク層の IP と、セション層以上のプロトコルの橋渡しをする。同じトランスポート層に位置する TCP と比較して転送速度は高いが、通信の信頼性が低い。

HTTP

HyperText Transfer Protocol の略

Web サーバとクライアント(Web ブラウザなど)がデータを送受信するのに使われるプロトコル。

SMTP

Simple Mail Transfer Protocol の略

インターネットやイントラネットで電子メールを送信するためのプロトコル。

POP/POP3

Post Office Protocol の略

インターネットやイントラネット上で、電子メールを保存しているサーバからメールを受信するためのプロトコル。

MAC アドレス

Media Access Control address の略。

各メーカー毎にユニークに割りあてられた数値と各メーカーがユニークに割りあてる数値を組み合わせた 48bit 長のアドレス。各 Ethernet カード(インタフェースモジュール)固有の ID 番号であり、全世界の Ethernet カードには 1 枚 1 枚固有の番号が割り当てられている。

ゲートウェイアドレス

評価キットボードをネットワークに接続する際に、他のネットワークへ通信を転送するためのローカルルータのIPアドレスを指定する。

ネットマスク

IP アドレスのうち、何ビットをネットワークを識別するためのネットワークアドレスに使用するかを定義する 32 ビットの数値。

10baseT/100baseTX

Ethernet 規格のうち、より対線(ツイストペアケーブル)を伝送媒体に使う規格群。最高通信速度は 10BaseT が 10Mbps、100BaseTX が 100Mbps である。100BASE-T 用の機器は 10BASE-T と互換性のあるものが多く、1 つのネットワークに混在させることができる。

プロバイダ

Internet Services Provider（インターネット接続業者）の略。

コンパイラ

ソースコードを、コンピュータが実行できる形式(オブジェクトコード)に変換するソフトウェア。

ディバッガ

debugger

プログラムの誤りの発見や修正を支援するソフトウェアで、プログラムの実行を特定の行で中断するブレークポイント機能やメモリや変数の内容を見るトレース機能が備えられている。デバッガともいう。

ICE

In-Circuit Emulator（インサーキットエミュレータ）の略

マイコンシステムを開発する際に用いるHWエミュレータの一種。

内蔵 RAM／内蔵 ROM

1チップの CPU デバイスに内蔵されたメモリ。外部メモリよりも早いアクセスが可能となる。