User's Manual

RENESAS

R-IN32M4-CL3 シリーズ

ユーザーズ・マニュアル　TCP/IP スタック編

・R-IN32M4-CL3

本資料に記載の全ての情報は本資料発行時点のものであり、ルネサス エレクトロニクスは、
予告なしに、本資料に記載した製品または仕様を変更することがあります。
ルネサス エレクトロニクスのホームページなどにより公開される最新情報をご確認ください。

資料番号：R18UZ0079JJ0100
発行年月：2021.8.31

ルネサス エレクトロニクス
www.renesas.com

arm

ご注意書き

1．本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

高品質水準：輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等） を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等

8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有·使用を規制するRoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注2. 本資料において使用されている「当社製品」とは、注1 において定義された当社の開発、製造製品をいいます。

製品ご使用上の注意事項

ここでは、CMOS デバイスの一般的注意事項について説明します。個別の使用上の注意事項については、本文を参照してください。なお、本マニュアルの本文と異なる記載がある場合は、本文の記載が優先するものとします。

1. 未使用端子の処理
   【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。
   CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】電源投入時は、製品の状態は不定です。
   電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。
   外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。
   同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. リザーブアドレスのアクセス禁止
   【注意】リザーブアドレスのアクセスを禁止します。
   アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレスがあります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
4. クロックについて
   【注意】リセット時は、クロックが安定した後、リセットを解除してください。
   プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。
   リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。

〇Arm® およびCortex® は、Arm Limited（またはその子会社）のEUまたはその他の国における登録商標です。 All rights reserved.
〇Ethernetおよびイーサネットは、 富士ゼロックス株式会社の登録商標です。
〇IEEEは、the Institute of Electrical and Electronics Engineers, Inc. の登録商標です。
〇TRONは”The Real-time Operation system Nucleus”の略称です。
〇ITRONは”Industrial TRON”の略称です。
〇μITRONは”Micro Industrial TRON”の略称です。
〇TRON、ITRON、およびμITRONは、特定の商品ないし商品群を指す名称ではありません。
〇EtherCAT® ,およびTwinCAT®は、ドイツBeckhoff Automation GmbHによりライセンスされた特許取得済み技術であり登録商標です。
〇CC-Link IE Field 及び CC-Link IE TSN は、三菱電機株式会社の登録商標です。
〇その他、本資料中の製品名やサービス名は全てそれぞれの所有者に属する商標または登録商標です。

このマニュアルの使い方

1.　目的と対象者

このマニュアルはイーサネット通信 LSI「R-IN32M シリーズ」の機能を理解し、それを用いた応用設計をするユーザを対象とします。このマニュアルを使用するには、電気回路、論理回路マイクロコンピュータに関する基本的な知識が必要です。

本製品は、注意事項を十分確認の上、使用してください。注意事項は、各章の本文中、各章の最後、注意事項の章に記載しています。

改訂記録は旧版の記載内容に対して訂正または追加した主な箇所をまとめたものです。改訂内容すべてを記録したものではありません。詳細は、このマニュアルの本文でご確認ください。
本文中の★印は、本版で改訂された主な箇所を示しています。この"★"を PDF 上でコピーして「検索する文字列」に指定することによって、改版箇所を容易に検索できます

関連資料　関連資料は暫定版の場合がありますが、この資料では「暫定」の表示をしておりません。あらかじめご了承ください。また各コアの開発・企画段階で資料を作成しているため、関連資料は個別のお客様向け資料の場合があります。下記資料番号の末尾****部分は版数です。当社ホームページより最新版をダウンロードして参照ください。

R-IN32M4-CL3に関する資料

資料名	資料番号
R-IN32M4-CL3 ユーザーズ・マニュアル ハードウェア編	R18UZ0073JJ****
R-IN32M4-CL3 ユーザーズ・マニュアル Gigabit Ethernet PHY 編	R18UZ0075JJ****
R-IN32M4-CL3 ユーザーズ・マニュアル ボード設計編	R18UZ0074JJ****
R-IN32M4-CL3 プログラミング・マニュアル ドライバ編	R18UZ0076JJ****
R-IN32M4-CL3 プログラミング・マニュアル OS 編	R18UZ0072JJ****
R-IN32M4-CL3 ユーザーズ・マニュアル CC-Link IE Field 編	R18UZ0071JJ****
R-IN32M4-CL3 ユーザーズ・マニュアル TCP/IP スタック編	本マニュアル

2.　数や記号の表記

データ表記の重み：左が上位桁、右が下位桁
アクティブ・ローの表記：
　　xxxZ（端子、信号名称のあとにZ ）
　　またはxxx_N　（端子、信号名称のあとに_N ）
　　またはxxnx　（端子、信号名称にnを含む）
注：
　　本文中につけた注の説明
注意：
　　気をつけて読んでいただきたい内容
備考：
　　本文の補足説明
数の表記：
　　2 進数 … xxxx , xxxxBまたはn'bxxxx（nビット）
　　10 進数 … xxxx
　　16 進数 … xxxxHまたはn'hxxxx（nビット）

2のべき数を示す接頭語（アドレス空間、メモリ容量）：
　　K （キロ）… $2^{10} = 1024$
　　M （メガ）… $2^{20} = 1024^2$
　　G （ギガ）… $2^{30} = 1024^3$
データ・タイプ：
　　ワード … 32 ビット
　　ハーフワード … 16 ビット
　　バイト … 8 ビット

目次

1. 概説 1

1.1 特長 1

1.2 主な機能 1

1.3 開発環境 2

1.3.1 開発ツール 2

1.3.2 評価ボード 2

1.3.3 開発手順 3

2. TCP/IP スタックの基本概念 4

2.1 用語の意味 4

2.1.1 プロトコル 4

2.1.2 プロトコルスタック 4

2.1.3 IP（Internet Protocol）アドレス 5

2.1.4 MAC（Media Access Control）アドレス 5

2.1.5 ポート番号 5

2.1.6 ビックエンディアンとリトルエンディアン 5

2.1.7 パケット 5

2.1.8 ホストとノード 6

2.1.9 Address Resolution Protocol（ARP） 6

2.1.10 Internet Protocol（IP） 6

2.1.11 Internet Control Message Protocol（ICMP） 6

2.1.12 Internet Group Management Protocol（IGMP） 6

2.1.13 User Datagram Protocol（UDP） 6

2.1.14 Transmission Control Protocol（TCP） 6

2.1.15 Dynamic Host Configuration Protocol（DHCP） 7

2.1.16 Hyper Text Transfer Protocol（HTTP） 7

2.1.17 File Transfer Protocol（FTP） 7

2.1.18 Domain Name System（DNS） 7

2.1.19 ソケット 7

2.1.20 ブロッキングとノンブロッキング 7

2.1.21 コールバック関数 8

2.1.22 タスクコンテキスト 8

2.1.23 リソース 8

2.1.24 MTU 8

2.1.25 MSS 8

2.1.26 IP リアセンブリ・フラグメント 8
2.2 ネットワークシステムのアーキテクチャ 9
2.2.1 ネットワークシステム構成図 9
2.3 ディレクトリとファイル構成 11

3. TCP/IP スタックの機能概要 14

3.1 プロトコルスタック 14
3.1.1 IP モジュール 14
3.1.2 ARP モジュール 16
3.1.3 UDP モジュール 16
3.1.4 TCP モジュール 19
3.2 ネットワークデバイスドライバ 24
3.2.1 デバイス構造体 24
3.2.2 インタフェース 26
3.2.3 パケットのルーティング 31
3.2.4 ループバックインタフェース 32
3.2.5 T_NET_DEV 情報の登録例 32
3.3 メモリ管理 33
3.3.1 ネットワークバッファ 34
3.3.2 ネットワークバッファ API 36
3.4 メモリI/O処理 37
3.4.1 メモリ I/O API 37
3.5 Ethernetデバイスドライバ 39
3.5.1 Ethernet デバイスドライバ構成 39
3.5.2 Ethernet デバイスドライバ API 40
3.5.3 コンフィグレーション 52
3.5.4 Ethernet デバイスドライバの注意事項 53
3.6 PHYドライバ 54
3.6.1 PHY ドライバ API 55
3.6.2 リンクイベントの通知 59

4. ネットワークコンフィグレーション 60

4.1 TCP/IPスタックのコンフィグレーション 60
4.1.1 コンフィグレーション一覧 60
4.1.2 IP アドレス 62
4.1.3 デバイスドライバ 62
4.1.4 プロトコルスタック情報テーブル 62
4.1.5 ネットワーク情報管理リソース 63

5. アプリケーションプログラミングインタフェースの説明 64

5.1 プロトコルスタックの初期化 64
5.2 ネットワーク・インタフェースAPI 65
5.3 ネットワークデバイス制御API 71
5.4 ソケットAPI (uNet3互換) 74
5.5 ソケットAPI (BSD互換) 89
5.5.1 モジュール概要 89
5.5.2 モジュール構成 90
5.5.3 API 一覧 91
5.5.4 API 詳細 92
5.5.5 ソケットオプション 120
5.5.6 サポート機能 121
5.5.7 BSD アプリの実装 123
5.6 その他API 127

6. ネットワークアプリケーション 133

6.1 DHCPクライアント 133
6.1.1 DHCP クライアント API 134
6.2 FTPサーバー 136
6.2.1 FTP サーバーAPI 140
6.2.2 制限事項 141
6.3 HTTPサーバー 142
6.3.1 HTTP サーバーAPI 146
6.3.2 HTTP サーバーサンプル 160
6.4 DNSクライアント 161
6.4.1 DNS クライアント API 162
6.5 DHCPクライアント拡張版 165
6.5.1 DHCP クライアント拡張版 API 167
6.5.2 DHCP クライアント拡張使用例 172
6.6 Pingクライアント 174
6.6.1 Ping クライアント API 175
6.7 SNTPクライアント 176
6.7.1 SNTP クライアント API 177
6.8 Stringライブラリ 178

7. サンプルを使ったチュートリアル 185

7.1 サンプルの説明 185
7.2 ハードウェア接続 185
7.3 ボードIPアドレス設定 186

7.3.1 固定 IP アドレス使用時 186
7.3.2 DHCP 機能使用時 188
7.4 動作確認 189
7.4.1 Web サーバー 189
7.4.2 MAC コントローラ制御 190
7.4.3 BSD ソケット 193
7.4.4 ノンブロッキング通信 197

8. 付録 198

8.1 パケット形式 198
8.2 定数とマクロ 205
8.3 エラーコード一覧 207
8.4 API一覧 208
8.5 リソース一覧 210
8.5.1 カーネルオブジェクト 210
8.5.2 Hardware ISR 211

R-IN32M4-CL3 シリーズ
ユーザーズ・マニュアル TCP/IP スタック編

R18UZ0079JJ0100
Rev.1.00
2021.8.31

1. 概説

本書は、R-IN32M4-CL3 用 TCP/IP および UDP/IP プロトコルスタックに関する資料です。

ルネサス製 R-IN32M4-CL3 用 TCP/IP プロトコル正式版スタック（以下、TCP/IP スタック）に関する、機能概要、アプリケーションプログラミングインタフェース（API）、アプリケーションサンプル、を記載しています。

1.1 特長

R-IN32M4-CL3 用 TCP/IP スタックは、R-IN32M4-CL3 に最適化されたコンパクトな TCP/IP プロトコルスタックです。

1.2 主な機能

TCP/IP スタックの主な機能は以下の通りです。

- IPv4、ARP、ICMP、IGMPv2、UDP、TCP プロトコルをサポート
- DHCP クライアント、DNS クライアント、FTP サーバー、HTTP サーバー機能が利用可能
- コンフィグレーションファイルによる TCP/IP の設定が可能
- TCP 高速再送/高速復帰アルゴリズムサポート
- IP 再構築とフラグメンテーションサポート
- 複数のネットワーク・インタフェースをサポート
- ネットワークアプリケーションのソースコード提供
- ネットワークデバイスドライバのソースコード提供
- ライブラリによるプロトコルスタックの提供

1.3 開発環境

ここでは、TCP/IP プロトコルスタックが対応している開発環境と、開発の手順について説明します。

1.3.1 開発ツール

サンプルソフトは下記のツールチェーンで動作を確認しています。

また、サンプルソフトでは Arm® Cortex®マイクロコントローラ・ソフトウェア・インタフェース規格（CMSIS）を採用しています。詳細は CMSIS のドキュメントを参照してください。

表 1.1 開発ツール対応表

LSI	ツールチェーン	IDE	コンパイラ	デバッガ	ICE
R-IN32M4-CL3	IAR	IAR Embedded Workbench for Arm V8.42.1 ～ 最新版 (最新版をお使いください) (IAR Systems)			I-Jet I-jet Trace for Arm Cortex-M (IAR Systems)

表 1.2 CMSIS Version

	R-IN32M4-CL3
Version	V4.5.0

1.3.2 評価ボード

TCP/IP スタックのサンプルアプリケーションはシマフジ電機株式会社製「R-IN32M4-CL3 評価ボード」、テセラ・テクノロジー株式会社製「TS-R-IN32M4-CL3 評価ボード」上で動作確認が可能です。

詳しくは、弊社またはシマフジ電機株式会社の WEB サイトをご覧ください。

1.3.3 開発手順

主な開発手順は以下の通りです。

1. TCP/IPスタックプログラムと、ドライバ／ミドルウェア一式、を統合
2. ネットワークのコンフィグレーション(IPアドレス、ソケット定義)、TCP/IPスタック初期化ルーチン呼出しなどのコンフィグレーションを、net_cfg.cに記述
3. アプリケーションプログラムを記述後、ビルド（コンパイル&リンク）して実行ファイルを作成

ファイル相関図を図 1.1 に示します。

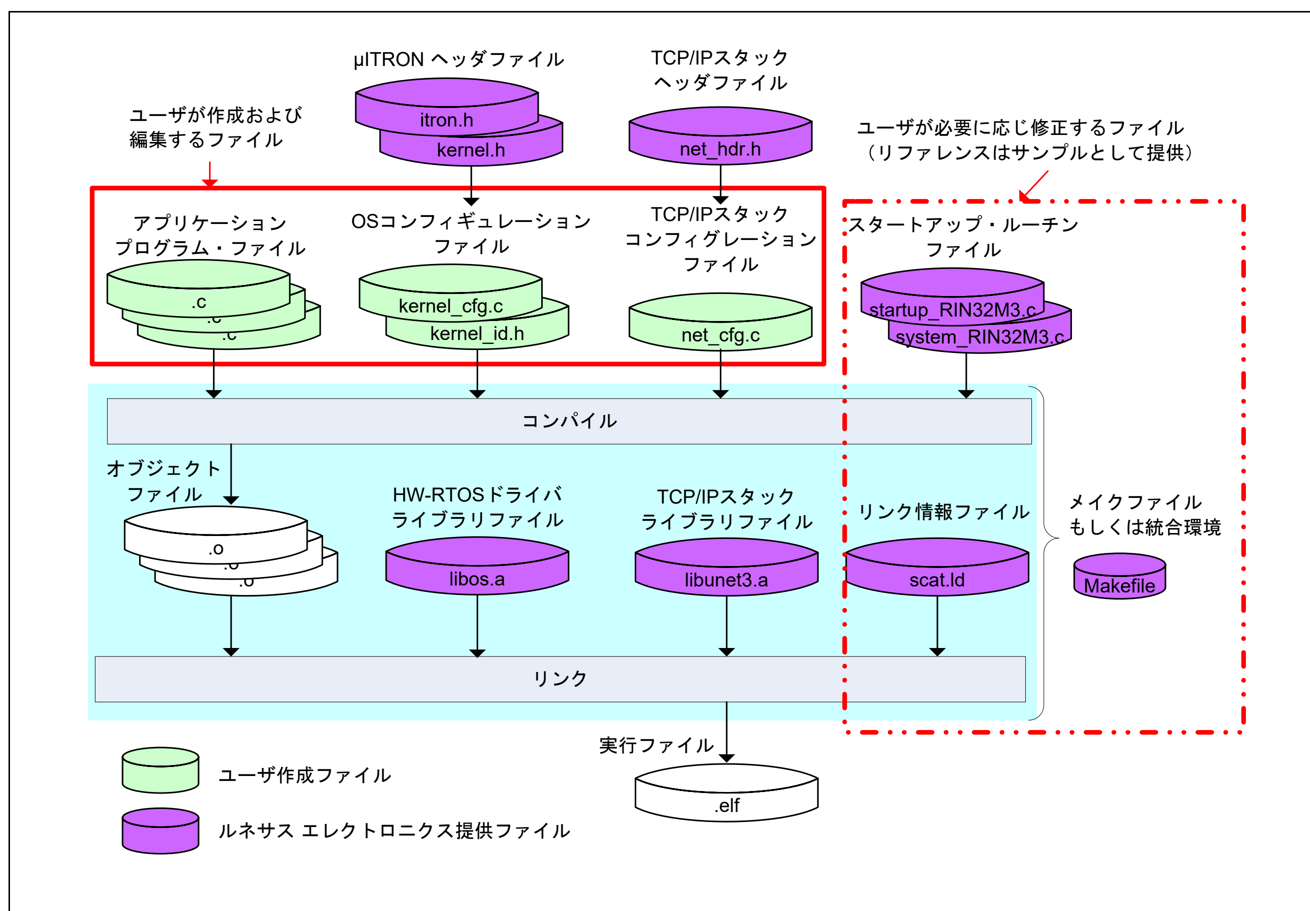


図 1.1 ファイル相関図

2. TCP/IP スタックの基本概念

2.1 用語の意味

2.1.1 プロトコル

ネットワーク間でデータを伝達する手順、方法等を定めたものを「プロトコル」と呼びます。TCP/IP スタックはこの「プロトコル」（=通信規則）を利用しています。これらの規則は“Request For Comments（通称：RFC）”と呼ばれるもので仕様が公開されています。

2.1.2 プロトコルスタック

ネットワーク上である機能を実現するために必要なプロトコルを選び、階層状に積み上げたソフトウェア群を「プロトコルスタック」と呼びます。TCP/IP スタックでは次のような階層となっています。

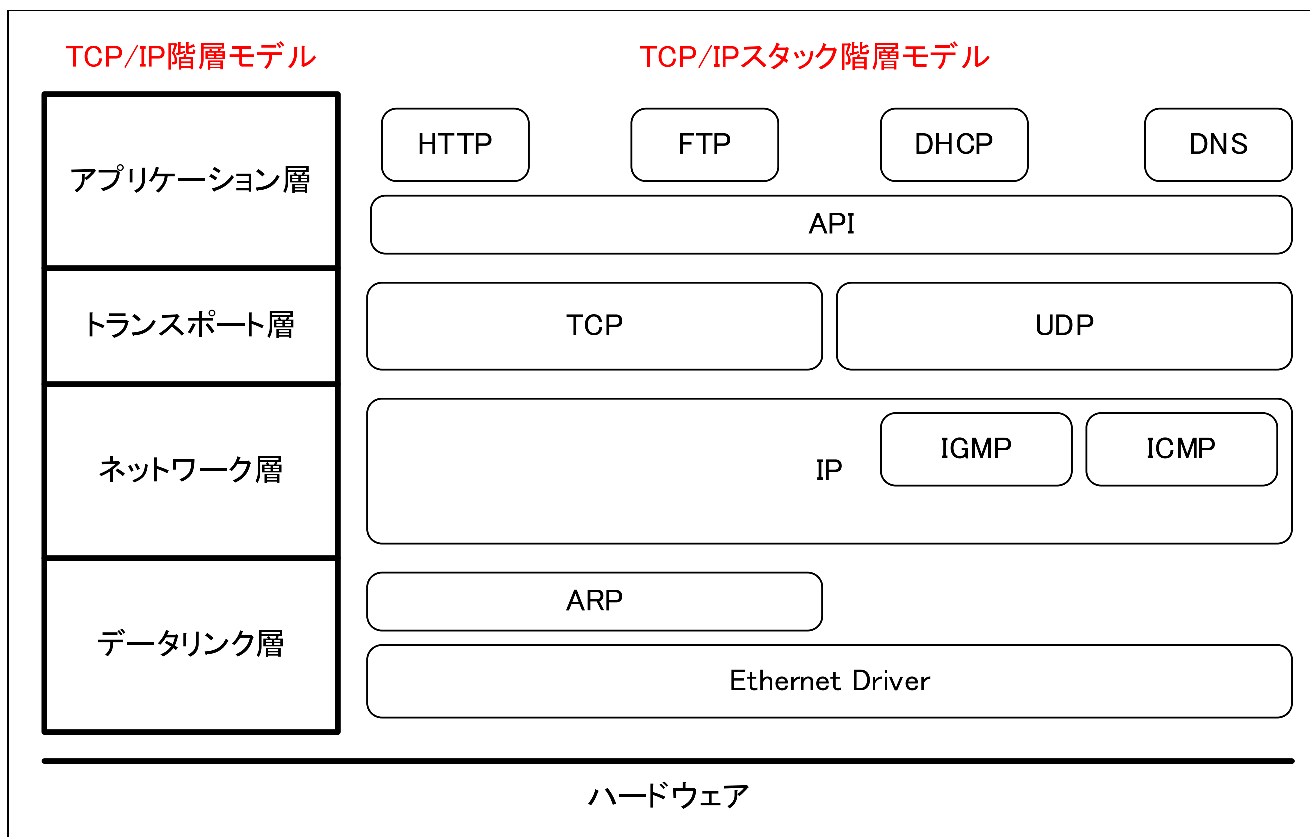


図 2.1　TCP/IP モデルと TCP/IP スタックの階層モデル図

2.1.3 IP（Internet Protocol）アドレス

ネットワーク上で各ノードを特定するための論理的な番号を「IP アドレス」と呼びます。「IP アドレス」は 32 ビットのアドレス空間を持ち、192.168.1.32 のように表記します。

(1) ブロードキャストアドレス

ブロードキャストとは一つのネットワークに属する全てのノードに対して、同時に同じデータを送る動作（=同報通信）のことを指します。そのブロードキャストのために特殊に割り当てられているアドレスのことを「ブロードキャストアドレス」と呼びます。通常、「ブロードキャストアドレス」にはすべてのビットが“１”の IP アドレス“255.255.255.255”を使用します。

(2) マルチキャストアドレス

ブロードキャストが全てのノードにデータを送信するのに対し、特定のグループに対してのみ、データを送信する専用のアドレスのことを「マルチキャストアドレス」と呼びます。

2.1.4 MAC（Media Access Control）アドレス

論理アドレスである「IP アドレス」に対し、LAN カードなどのネットワーク機器を識別するために設定されているハードウェア固有の物理アドレスを「MAC アドレス」と呼びます。「MAC アドレス」は 48 ビットのアドレス空間を持ち、12-34-56-78-9A-BC　や 12:34:56:78:9A:BC　のように表記します。

2.1.5 ポート番号

ネットワーク通信で通信相手のプログラムを特定する番号のことを「ポート番号」と呼びます。TCP/IP で通信を行なうノードはネットワーク内での住所にあたる IP アドレスを持っていますが、複数のノードと同時に通信するために、補助アドレスとして 0 から 65535 のポート番号を用います。

2.1.6 ビックエンディアンとリトルエンディアン

複数バイトで構成されている数値データを、メモリに格納するときの方式のことを「エンディアン」と呼び、最上位バイトから順に格納する方式のことを「ビッグエンディアン」と呼びます。最下位バイトから順に格納する方式のことを「リトルエンディアン」と呼びます。

TCP/IP ではヘッダ情報は「ビッグエンディアン」で送信することが定められています。

2.1.7 パケット

データの送受信の単位を「パケット」と呼びます。パケットには二つの情報が含まれており、ひとつは実際のデータが格納された部分（データ領域）と、もうひとつは、そのデータの宛先や送信元情報、エラーチェック情報といった管理用の情報が格納される部分（ヘッダー領域）です。

2.1.8 ホストとノード

ネットワークで通信するコンピュータのことを「ホスト」と呼びます。サーバー、クライアント、ハブ、ルーター、アクセスポイント等、ネットワーク節点のことを「ノード」と呼びます。

2.1.9 Address Resolution Protocol（ARP）

論理アドレス（TCP/IP の場合は IP アドレス）から物理アドレス（MAC アドレス）を導き出すためのプロトコルを「ARP」と呼びます。

2.1.10 Internet Protocol（IP）

ノード間またはノードとゲートウェイ間の通信を実現するプロトコルを「IP（IP プロトコル）」と呼びます。「IP（IP プロトコル）」は上位層の基礎となる重要なプロトコルです。「IP」の役割は IP アドレスを元に、ルーターなどを経由して宛先にデータを届けることですが、確実に届けるという保証はなく、データ信頼性の確保は上位層の役割となっています。

前述の「IP アドレス」はこの「IP プロトコル」のヘッダに置かれます。

2.1.11 Internet Control Message Protocol（ICMP）

「IP」ネットワーク通信で発生したエラーを通知したり、ネットワークの状態を確認する為の機能を提供するプロトコルを「ICMP」と呼びます。よく知られているものに Ping と言われるエコー要求、エコー応答メッセージがあります。

2.1.12 Internet Group Management Protocol（IGMP）

IP マルチキャストを実現する為のプロトコルを「IGMP」と呼びます。同一のデータを複数のホストに効率よく配送することができます。

2.1.13 User Datagram Protocol（UDP）

コネクションレス型のデータグラム通信サービスを提供するプロトコルを「UDP」と呼びます。「IP」はアプリケーションとのインタフェースを持っていません。「UDP」はその機能をアプリケーションから使えるようにしたプロトコルです。故に、パケットが相手に届いたことを知らせる手段がないことや、パケットの届く順番が入れ替る可能性があり、データの信頼性は保証されません。

2.1.14 Transmission Control Protocol（TCP）

コネクション型のストリーム通信サービスを提供するプロトコルを「TCP」と呼びます。「TCP」は IP プロトコルの上位層として、順序制御と誤り訂正や再送・フロー制御といった信頼性のある通信を提供します。

2.1.15 Dynamic Host Configuration Protocol（DHCP）

ネットワークに接続する際に、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を「DHCP」と呼びます。「DHCP」を使うためには DHCP サーバーを用意し、サーバー側で、あらかじめ DHCP クライアント用に IP アドレスをいくつか用意しておく必要があります（アドレスプール）。

2.1.16 Hyper Text Transfer Protocol（HTTP）

ホームページやウェブサイトの HTML ファイルなどのコンテンツの転送を行うためのプロトコルを「HTTP」と呼びます。「HTTP」は HTML ファイルだけの転送のみならず、WEB ブラウザで表示できる、JPEG、GIF、PNG、ZIP などのバイナリデータの転送も可能です。

2.1.17 File Transfer Protocol（FTP）

ホスト間でファイル転送を行うためのプロトコルを「FTP」と呼びます。

2.1.18 Domain Name System（DNS）

IP アドレスをホスト（ドメイン）名に、ホスト名を IP アドレスに変換する名前解決メカニズムのことを「DNS」と呼びます。「DNS」を利用すると IP アドレスをもとにホスト名を求めたり、ホスト名から IP アドレスを求めたりすることが可能になります。

2.1.19 ソケット

アプリケーションが TCP/IP 通信するための通信窓口のことを「ソケット」と呼びます。「ソケット」は IP アドレスとポート番号等で構成されています。アプリケーションは「ソケット」を指定して回線を開くだけで、通信手順の詳細を気にすることなくデータの送受信を行なうことができます。通信側で使用しているプロトコルによりソケットの種類が存在します。TCP ソケットは TCP プロトコルを使用してデータ通信を実施し、UDP ソケットは UDP プロトコルを使用してデータ通信を実施します。TCP/IP スタックでは操作対象となる「ソケット」を識別するのに ID 番号を使用します。アプリケーションでは ID 番号を用いてソケット API を呼び出します。

2.1.20 ブロッキングとノンブロッキング

何らかの関数を呼び出したとき、そのアクションが完了するまで戻らないことを「ブロッキングモード」と呼び、完了を待たずに即座に戻ることを「ノンブロッキングモード」と呼びます。

例えばソケット API において、「ブロッキング モード」で、rcv_soc 関数を呼び出したタスクはそのアクションが完了する（データが受信できる）まで待ち状態に置かれることになります。「ノンブロッキングモード」では rcv_soc 関数の呼出しは、エラーコード E_WBLK とともに即座に戻り、そのアクションの完了（EV_RCV_SOC）はコールバック関数に通知されます。

TCP/IP スタックのソケットのデフォルト動作は「ブロッキングモード」となっており、「ノンブロッキングモード」に変更するには cfg_soc 関数を使用してコールバック関数の登録とコールバックイベントフラグを設定します。

2.1.21 コールバック関数

プロトコルスタックの状態を非同期にアプリケーションに通知する為の関数を「コールバック関数」と呼びます。

2.1.22 タスクコンテキスト

TCP/IP スタックの全ての API・関数は、タスクコンテキストから呼び出さなければならない。

ネットワークコールバック関数から slp_tsk 等のタスクを待ち状態にするシステムコールを呼び出さないでください。また、ネットワークコールバック関数から TCP/IP スタックの全ての API・関数を呼び出さないでください。

2.1.23 リソース

プログラムで使用する資源のことを「リソース」と呼びます。タスク、セマフォといった「カーネルオブジェクト」、メモリなどが該当します。

2.1.24 MTU

MTU（Maximum Transfer Unit）は、通信ネットワークにおいて、1 回の転送で送信できるデータの最大値を示す値である。そして、MTU は、データリンク層のフレームの最大データサイズを示す。なお、MTU で指定のできる最小の値は 68 バイトとなります。

最大データサイズの指定は、データリンク層で使用するプロトコルに依存し、Ethernet インタフェースでは一般的に 1500 バイトが使用されています。

2.1.25 MSS

MSS （Maximum Segment Size）は TCP パケットの最大データサイズを示す。そのため、MSS の値は、次式で計算することができます。

MSS = MTU - (IP ヘッダーサイズ + TCP ヘッダーサイズ (通常は 40 バイト))

Ethernet インタフェースでは一般的に MSS の値は、1460 バイトとなります。

2.1.26 IP リアセンブリ・フラグメント

IP パケットの最大サイズは 64K バイトとなります。しかし、通信インタフェースの MTU はこれよりも小さい値となっているため、IP モジュールが、IP パケットをいくつかに分割して送信する必要があります。この処理を「IP フラグメンテーション」と呼び、分割された IP パケットのことを「IP フラグメント」と呼びます。

そして、受信側の IP モジュールでは、分割された「IP フラグメント」を連結する必要があり、この処理を「IP リアセンブリ」と呼びます。

2.2 ネットワークシステムのアーキテクチャ

2.2.1 ネットワークシステム構成図

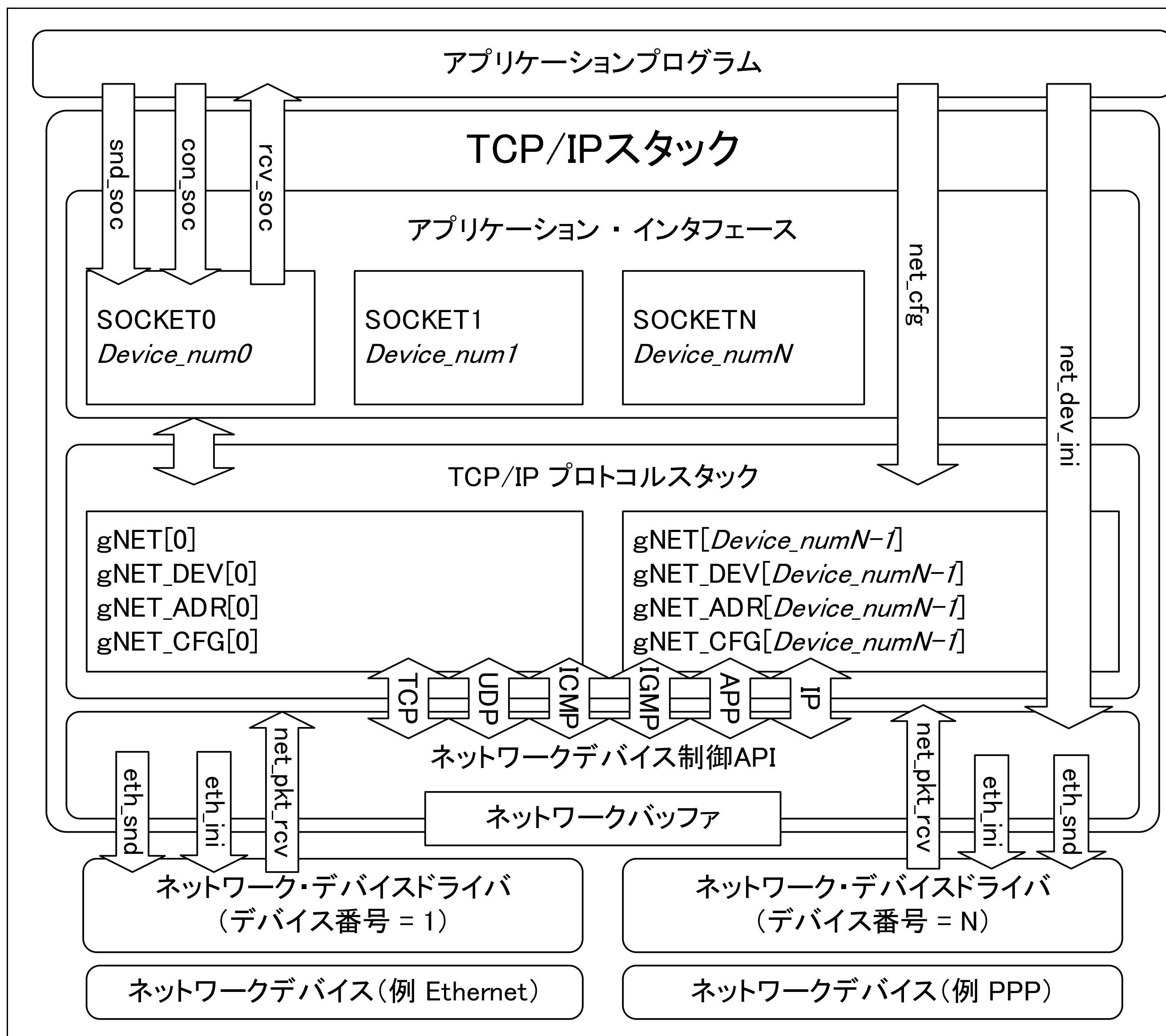


図 2.2　ネットワークシステムの構築図

- アプリケーションプログラム
  ネットワーク通信するためのユーザアプリケーションプログラムです。DHCP、FTP、Telnet、HTTP などのアプリケーションプロトコルも含まれます。

- アプリケーション・インタフェース
  リモートホストとの接続の確立、データの送信・受信等といった様々なネットワークサービスを利用するためのインタフェース（API）を提供します。
  通常アプリケーションではソケット ID やデバイス番号を指定してアプリケーション・インタフェースを使用します。

- TCP/IP プロトコルスタック
  このプログラムは TCP、UDP、ICMP、IGMP、IP、ARP といったネットワークプロトコルを処理します。

- ネットワークデバイス制御 API
  ネットワークシステムには様々はネットワークデバイスが存在している可能性があり、デバイス毎にデバイスドライバが必要になります。ネットワークデバイス制御 API は、これらのデバイスの違いを吸収し、統一的にアクセスするためのインタフェースを提供します。アプリケーションプログラムからデバイス番号を使用して各デバイスにアクセスします。

- ネットワークデバイスドライバ
  ネットワークデバイスを制御するプログラムです。この実装の中身はデバイスごとに異なります。
  TCP/IP スタックでは標準で Ethernet のデバイスドライバを提供しています。

- ネットワークデバイス
  実際のネットワークデータの送信、受信を行うハードウェアです。Ethernet、PPP（RS- 232）、WLAN などがこれに該当します。

2.3 ディレクトリとファイル構成

TCP/IP スタックに含まれるファイルは次の通りです。

(1) ヘッダファイル

/Source/Middleware/uNet3/Inc/

net_sup.h	TCP/IPプロトコルスタックのデフォルトコンフィグレーションマクロ
net_def.h	TCP/IPプロトコルスタックの定義（内部制御用）
net_sts.h	ネットワーク情報管理の定義（内部制御用）
net_sts_id.h	ネットワーク情報管理IDの定義
net_hdr.h	TCP/IPプロトコルスタックを使用するための必要な情報が定義 ※このヘッダファイルはアプリケーションのソースファイルに必ず含めてください

(2) ライブラリファイル

このフォルダには、ビルド済みの TCP/IP プロトコルスタックが、ツールチェーン毎にライブラリ化されて格納されています。

/Library/IAR/

libunet3.a	TCP/IPプロトコルスタックのライブラリファイル
libunet3bsd.a	TCP/IPプロトコルスタックのライブラリファイル(BSD互換)
libunet3snmp.a	TCP/IPプロトコルスタックのライブラリファイル(SNMP互換)

(3) Ethernet デバイスドライバファイル

/Source/Driver/ether_uNet3/

DDR_ETH.c	Etherドライバ本体
DDR_PHY0.c	LAN1のPHYドライバ本体
DDR_PHY1.c	LAN2のPHYドライバ本体

/Source/Driver/ethsw/

ethsw.c	Ethernet Switchドライバ

/Include/ ether_uNet3/

DDR_ETH.h	Etherドライバヘッダ
DDR_PHY.h	PHYドライバ共通ヘッダ
COMMONDEF.h	Ethernet ドライバ共通定義ヘッダ

/Include/ ethsw/

ethsw.h	Ethernet Switchドライバヘッダ

(4) アプリケーションプロトコルソースファイル

/Source/Middleware/uNet3/NetApp/

dhcp_client.h	DHCPクライアント マクロ、プロトタイプ、定義等
dhcp_client.c	DHCP クライアント ソースコード
ftp_server.h	FTPサーバー マクロ、プロトタイプ、定義等
ftp_server.c	FTP サーバー ソースコード
http_server.h	HTTPサーバー マクロ、プロトタイプ、定義等
http_server.c	HTTP サーバー ソースコード
dns_client.h	DNSクライアント マクロ、プロトタイプ、定義等
dns_client.c	DNSクライアント ソースコード
ping_client.h	ICMP エコー要求 マクロ、プロトタイプ、定義など
ping_client.c	ICMP エコー要求（ping） ソースコード
sntp_client.h	SNTPクライアント マクロ、プロトタイプ、定義等
sntp_client.c	SNTPクライアント ソースコード
net_strlib.h	TCP/IPスタック提供String系ライブラリ関数定義
net_strlib.c	TCP/IPスタック提供String系ライブラリ関数ソースコード

/Source/Middleware/uNet3/NetApp/ext/

dhcp_client.h	DHCPクライアント、プロトタイプ、定義等
dhcp_client.c	拡張版DHCP クライアント ソースコード

/Source/Middleware/uNet3/NetApp/cfg/

ftp_server_cfg.c	FTPサーバーコンフィグレーション
ftp_server_cfg.h	FTPサーバーコンフィグレーションヘッダ
http_server_cfg.h	HTTP サーバーコンフィグレーションヘッダ

/Source/Middleware/uNet3/snmp/

inc/snmp.h	SNMPヘッダ
inc/snmp_ber.h	SNMP BER
inc/snmp_def.h	SNMP定数、プロトタイプ、定義等
inc/snmp_lib.h	SNMPライブラリコンフィグレーション
inc/snmp_mac.h	SNMPマクロ定義
inc/snmp_mib.h	SNMP MIB定義
inc/snmp_net.h	SNMP TCP/IP用定義
src/snmp_mib_dat.c	SNMP MIB定義本体

(5) サンプルソースファイル

/Source/Project/uNet3_sample/

cgi_sample.c	CGIを使ったアプリケーションのサンプル
DDR_ETH_CFG.h	Etherドライバコンフィグレーションヘッダ
html.h	HTMLデータ
kernel_cfg.c	OS資源のコンフィグファイル
kernel_id.h	OS資源のID定義ファイル
main.c	メイン関数
net_cfg.c	TCP/IPスタックのコンフィグレーションファイル
net_sample.c	ネットワークのサンプルアプリケーション

/Source/Project/uNet3_nonblock/

DDR_ETH_CFG.h	Etherドライバコンフィグレーションヘッダ
nonblock_sample.c	ノンブロッキングで動作するEchoサーバーサンプル
kernel_cfg.c	OS資源のコンフィグファイル
kernel_id.h	OS資源のID定義ファイル
main.c	メイン関数
net_cfg.c	TCP/IPスタックのコンフィグレーションファイル
net_sample.c	ネットワークのサンプルアプリケーション

/Source/Project/uNet3_bsd/

DDR_ETH_CFG.h	Etherドライバコンフィグレーションヘッダ
socket_command.c	シリアルコンソールを使った動作確認プログラム
kernel_cfg.c	OS資源のコンフィグファイル
kernel_id.h	OS資源のID定義ファイル
main.c	メイン関数
net_cfg.c	TCP/IPスタックのコンフィグレーションファイル
net_sample.c	ネットワークのサンプルアプリケーション

/Source/Project/uNet3_mac/

console.c	シリアルコンソールを使ったテストプログラム
DDR_ETH_CFG.h	Etherドライバコンフィグレーションヘッダ
kernel_cfg.c	OS資源のコンフィグファイル
kernel_id.h	OS資源のID定義ファイル
main.c	メイン関数
net_cfg.c	TCP/IPスタックのコンフィグレーションファイル
net_sample.c	ネットワークのサンプルアプリケーション

/Source/Project/uNet3_snmp

cgi_sample.c	CGIを使ったアプリケーションのサンプル
DDR_ETH_CFG.h	Etherドライバコンフィグレーションヘッダ
html.h	HTMLデータ
kernel_cfg.c	OS資源のコンフィグファイル
kernel_id.h	OS資源のID定義ファイル
main.c	メイン関数
net_cfg.c	TCP/IPスタックのコンフィグレーションファイル
net_sample.c	ネットワークのサンプルアプリケーション
snmp_cfg.c	SNMPのコンフィグレーションファイル
snmp_mib_cfg.c	MIB定義ファイル

3. TCP/IP スタックの機能概要

3.1 プロトコルスタック

3.1.1 IP モジュール

IP モジュールでは送られてくるパケットの宛先 IP アドレスが、自ホストの IP アドレスと一致するときだけパケットを受信し処理します。それ以外のパケットは処理しません。

(1) IP オプション

TCP/IP スタックは IP オプションの内 IGMP ルーター警告オプションのみサポートしています。サポートしていない IP オプションは無視されます。

(2) TTL（Time to Live）

TCP/IP スタックで TTL のデフォルト値は CFG_IP4_TTL(64)に設定されています。この値は net_cfg()を使って変更することができます。net_cfg()を使って TTL 値を変更した場合、すべてのソケットの TTL 値が変更されます。個々のソケットの TTL 値を変更したい場合は cfg_soc()を使用してください。

(3) TOS（Type Of Service）

TCP/IP スタックで TOS は CFG_IP4_TOS(0)に設定されています。

(4) ブロードキャスト

ブロードキャストの受信可否は net_cfg()を使って変更することができます。初期値は受信不可に設定されています。ブロードキャストの送信は常に可能です。ブロードキャストの設定はすべてのソケットに対して有効で、ソケット単位でのブロードキャストの受信可否設定はできません。

ブロードキャストの送受信には UDP ソケットを使用してください。

(5) マルチキャスト

マルチキャスト受信を許可するには net_cfg()を使って、参加するマルチキャストグループアドレスを登録します。マルチキャストグループアドレスは CFG_NET_MGR_MAX(8)まで登録することができます。マルチキャストの送信は常に可能です。マルチキャストの設定はすべてのソケットに対して有効で、ソケット単位でのマルチキャストの受信可否設定はできません。

マルチキャスト送信用 TTL は CFG_IP4_MCAST_TTL(1)に設定されています。この値も net_cfg()を使って変更することができます。

マルチキャストのループバックはサポートしていません。

マルチキャストの送受信には UDP ソケットを使用してください。

(6) MTU

TCP/IP スタックでは MTU のデフォルト値として CFG_PATH_MTU（1500 バイト）を設定しています。この値は、コンフィグレーションファイルで設定することができます。

(7) IP リアセンブリ・フラグメント

TCP/IP スタックでは、IP パケットとして、最大サイズはデフォルトとして、1500 バイトとなっています（この値は、ネットワークバッファの値と関連しています）。IP パケットの最大サイズを大きくするためには、ネットワークバッファを大きくする必要があります。例えば、2048byte の UDP データを送受信する場合は、ネットワークバッファの値を、「コントロールヘッダサイズ（100 bytes）+ IP ヘッダーサイズ（20bytes）+ UDP ヘッダーサイズ（8bytes）+ 2048」の計算値よりも大きくする必要があります。

デフォルトの IP リアセンブリ・プロセス・タイムアウト値は、CFG_IP4_IPR_TMO（10 秒）となっています。もし、リアセンブリ・プロセスがこのタイムアウト内に完了しない場合、リアセンブリ処理は取り消され、ICMP エラーメッセージ（タイプ 11：時間超過によるパケット廃棄）がリモートホストに送られます。

デフォルトの IP リアセンブリ・プロセス回数は CFG_NET_IPR_MAX(2)として設定しています。CFG_NET_IPR_MAX の値は、ホストが同時に IP リアセンブリ処理を実施することができる値を示しています。

(8) IGMP

TCP/IP スタックでは（ルーターからの）「クエリ（グループ問い合わせ）」対する「レポート（応答）」メッセージの送信までのタイムアウトは CFG_IGMP_REP_TMO（10 秒）に設定されています。

TCP/IP スタックは IGMPv2 をサポートしていますが、IGMPv1 互換機能もサポートしています。

IGMPv1 の「クエリ」を受け取った場合、IGMPv1 モードに切り替えて処理を行います。その後、一定時間、IGMPv1 メッセージが無ければ IGMPv2 モードに戻ります。この IGMPv 1 から IGMPv 2 に戻るまでのタイムアウトは CFG_IGMP_V1_TMO（400 秒）に設定されています。

(9) ICMP

TCP/IP スタックは「エコー応答」、「エコー要求」、「時間超過」メッセージをサポートしています。

3.1.2 ARP モジュール

(1) アドレス解決

TCP/IP スタックではホストの IP アドレスと物理アドレス（MAC アドレス）の対応付けを管理しています。この対応付けの管理表（変換表）を ARP キャッシュと呼びます。ARP キャッシュサイズはCFG_NET_ARP_MAX(8)に設定されています。

IP パケットをネットワークに送信する際、ARP キャッシュを参照し該当する IP アドレスが存在した場合は、そこに記録されている物理アドレスを宛先としてパケットを送信します。IP アドレスが存在しない場合は、IP パケットは一旦キューに保存し、ARP 要求パケットをブロードキャスト送信します。リモートホストから ARP 応答パケットを受信したら、新たに ARP キャッシュに受信した物理アドレスを記録します。その後、キューから IP パケットを取り出し、新たに取得した物理アドレス宛てにパケットを送信します。

また、ARP エントリ情報は最長 ARP_CLR_TMO（20 分間）キャッシュに保持されます。

(2) IP アドレス競合検出

RFC5227 に従って、同一リンク内の他のホストと IP アドレスが重複していないかをチェックします。このチェックは LAN インタフェースの起動時や、リンクアップ状態に移行した際にアプリケーションの指示により実行します。またインタフェースに IP アドレスが設定されたのち、他のホストが同じ IP アドレスを使用していた場合には、競合を検出してアプリケーションに通知します。

IP アドレスの競合検出には ARP メッセージを使用します。これから使用する IP アドレスが、既に使用されていないかを探知する ARP メッセージを「ARP Probe」と言います。ARP Probe メッセージに対して他のホストが ARP 応答しなかった（競合する IP アドレスが無い）場合には、「ARP Announce」と言うメッセージを送信して IP アドレスを使用することを通知します。

3.1.3 UDP モジュール

UDP はリモートホストと接続する事なしにデータの送受信を行います。

(1) データの送信

データの送信前には必ず con_soc を使って、送信先（IP アドレス、ポート番号）とソケットの関連付けを行います。その後、snd_soc()を使ってデータを送信します。snd_soc()の処理フローは下図のようになります。

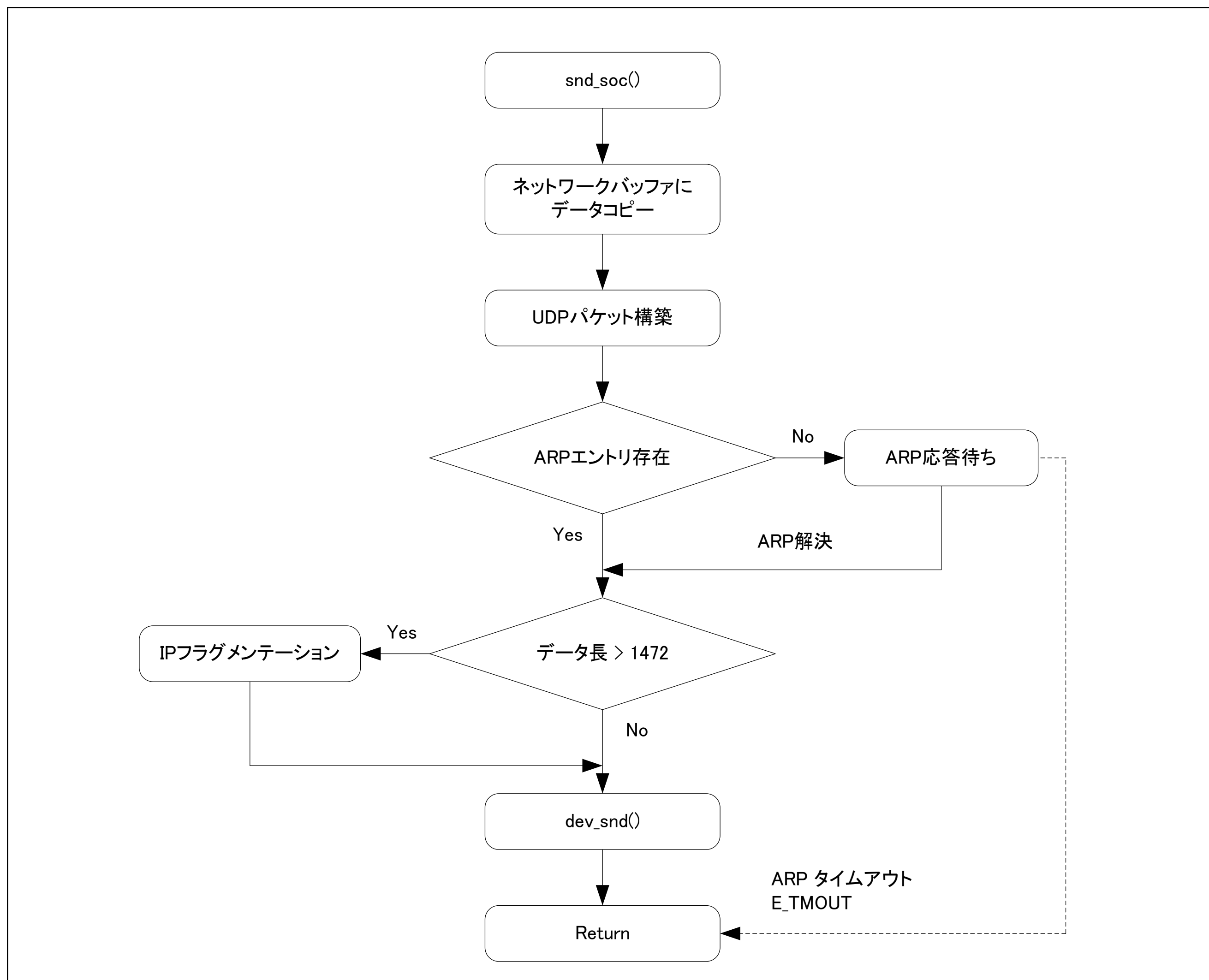


図 3.1　　UDP ソケット snd_soc の処理フロー

1. アプリケーションデータはネットワークバッファにコピーされ、リモートホストのIPアドレス、ポート番号などUDPヘッダを付加し、UDPパケットを構築します。
2. ARPプロトコルでリモートホストのMACアドレスが解決出来ないときは、E_TMOUTエラーを返します。
3. デフォルトでは、送信データの最大サイズが1472バイト（CFG_PATH_MTU（1500バイト）-IPヘッダーサイズ -UDPヘッダーサイズ）に設定されています。これ以上のサイズを送信する場合は、ネットワークバッファサイズの設定が必要です。詳細は、IPリアセンブリ・フラグメントの項目を参照してください。

(2) データの受信

データの受信は rcv_soc()を使います。rcv_soc()の処理フローは下図のようになります。

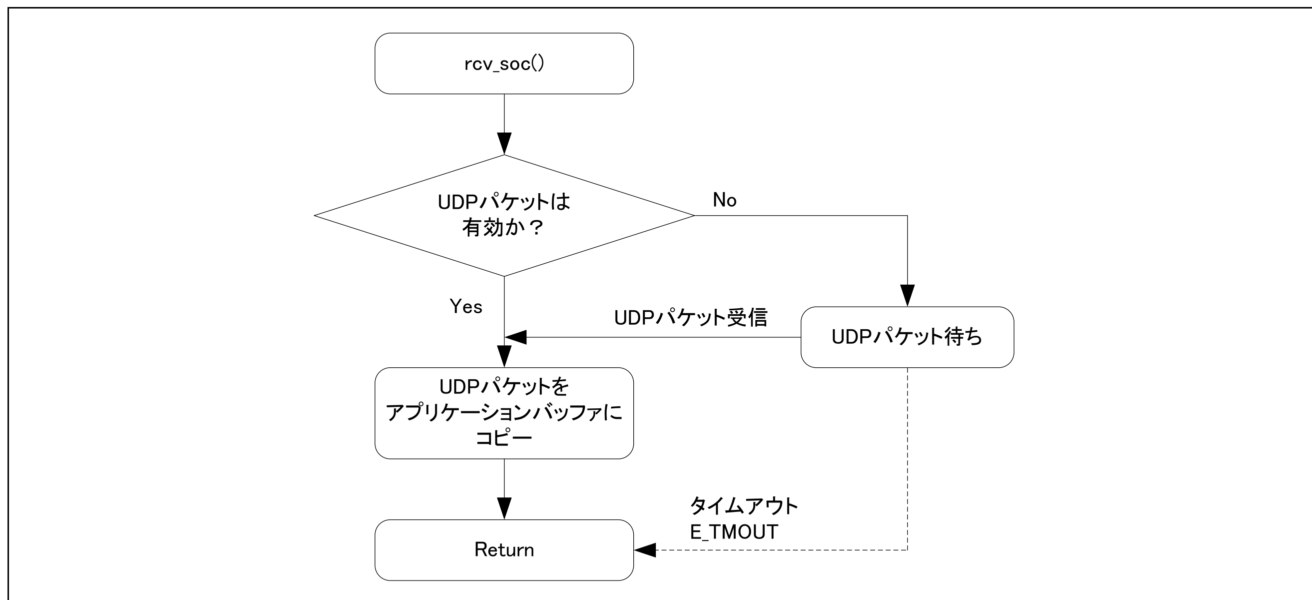


図 3.2 UDP ソケット rcv_soc の処理フロー

1. UDPパケットが未受信なら、UPDパケットの受信待ちになります。この時、ソケットの受信タイムアウトを過ぎたらE_TMOUTを返します。
2. 受信したパケットサイズが要求されたデータサイズよりも小さければ、アプリケーションのバッファにコピーします。受信したパケットサイズが要求されたデータサイズよりも大きいときは、要求サイズ分だけアプリケーションのバッファにコピーします。残ったデータは捨てられます。
3. デフォルトでは、受信データの最大サイズが1472バイト（CFG_PATH_MTU（1500バイト）-IPヘッダーサイズ-UDPヘッダーサイズ）に設定されています。これ以上のサイズを受信する場合は、ネットワークバッファサイズの設定が必要です。詳細は、IPリアセンブリ・フラグメントの項目を参照してください。

3.1.4 TCP モジュール

TCP は UDP と異なり、コネクション型ですので送信相手と通信路を確保しデータの送受信を行います。TCP のシーケンスは下図のようになります。

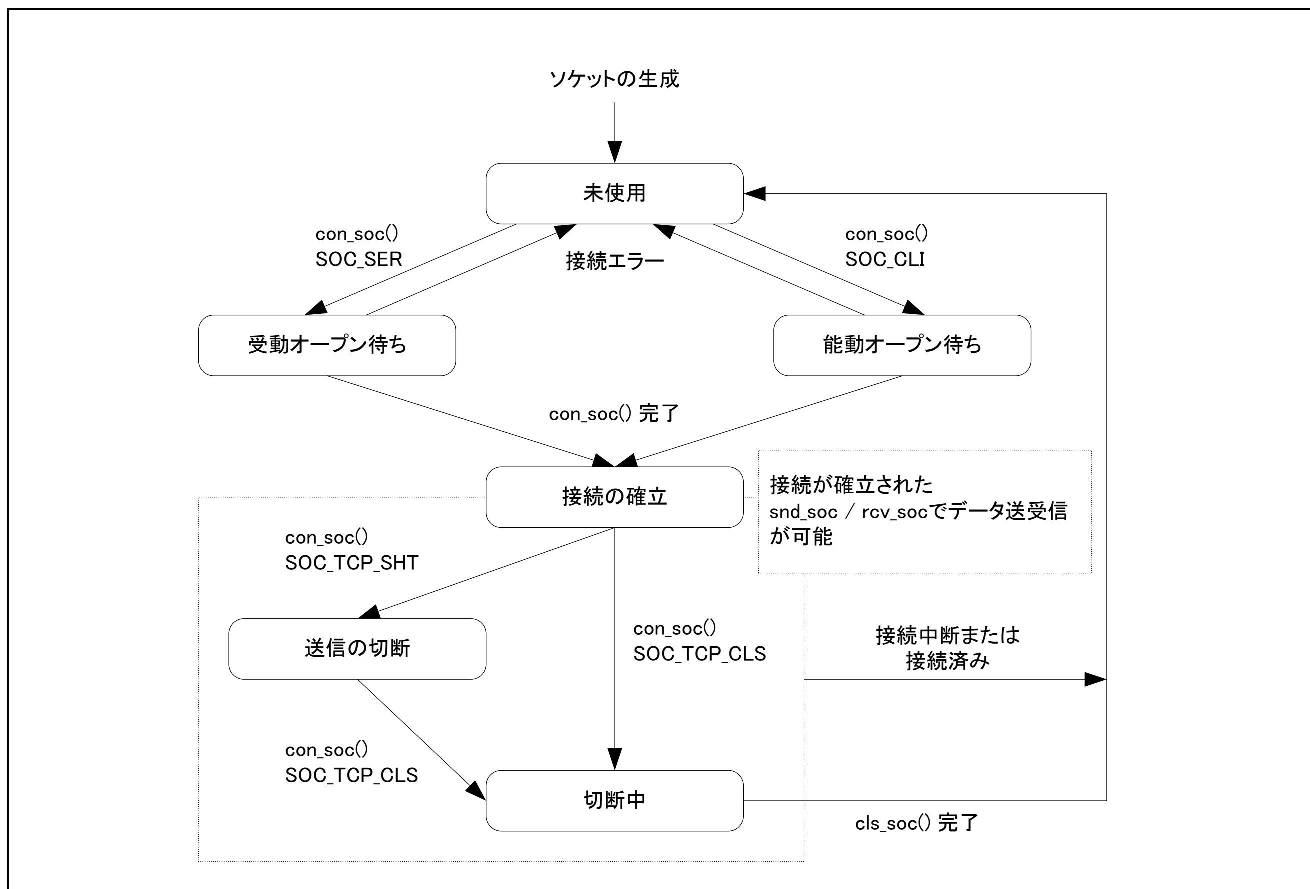


図 3.3 TCP のシーケンス

(1) 接続の確立

TCP 接続には能動接続と、受動接続の二つのモードがあります。能動接続はリモートホストに自ら接続要求します。対して受動接続はリモートホストからの接続を待ちうけます。

接続には con_soc()を使用し、SOC_CLI で能動接続、SOC_SER で受動接続を指定します。

(2) 接続の終了

接続を切断するには　cls_soc()を使用します。完全に接続を切断するには SOC_TCP_CLS を送信のみ切断するには SOC_TCP_SHT を指定します。

(3) データの送信

snd_soc()を使ってデータを送信します。snd_soc()の処理フローは下図のようになります。

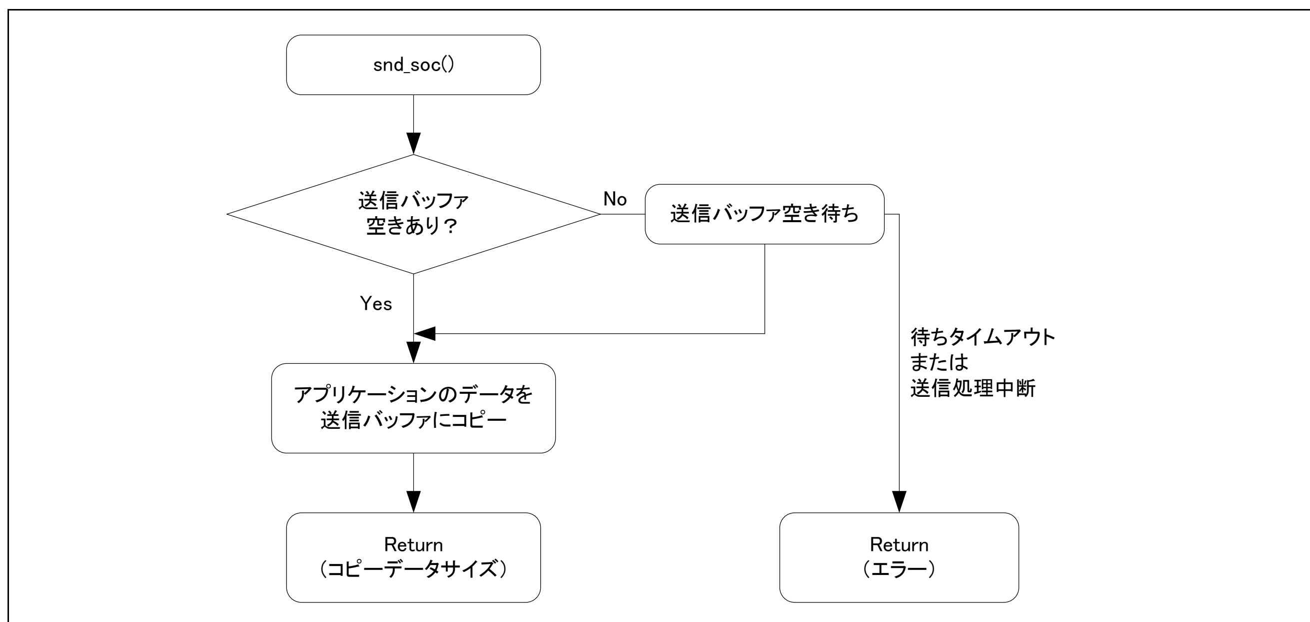


図 3.4 TCP ソケット snd_soc の処理フロー

4. アプリケーションのデータをTCP送信バッファにコピーします。コピーが成功したらTCPプロトコルがデータを送信します。リモートホストがデータを受信したらTCP送信バッファにあるデータはクリアされます。

- TCP 送信バッファ

送信バッファサイズは TCP ソケット作成時に指定する必要があります。バッファサイズは 4 バイトから 32 キロバイトの範囲で、2 の累乗(1024, 2048, 4096 など)で指定します。

(4) データの受信

rcv_soc()を使ってデータを送信します。受信した TCP パケットは、まず TCP 受信バッファに登録されます。rcv_soc()が呼ばれたら TCP 受信バッファからアプリケーションのバッファにコピーされます。

- TCP 受信バッファ（ウィンドウバッファ）

受信バッファサイズは TCP ソケット作成時に指定する必要があります。バッファサイズは 4 バイトから 32 キロバイトの範囲で、2 の累乗(1024, 2048, 4096 など)で指定します。

(5) 再送タイムアウト

再送タイマのシーケンスは下図のようになります。

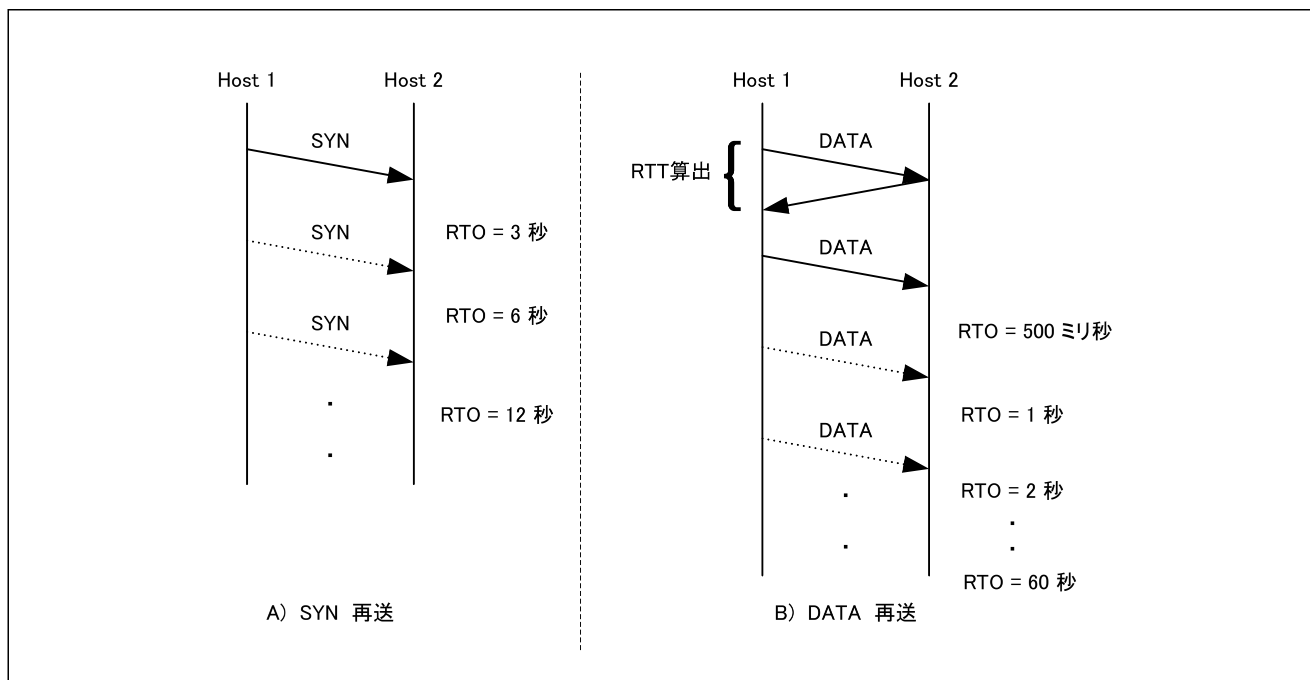


図 3.5　再送タイマ例

TCP では何らかの原因で一定時間の間 ACK パケットの応答が無い場合、応答が無かったセグメントを再送します。この再送するまでの待ち時間のことを「RTO」（Retransmission Time Out　再送タイムアウト）と呼びます。RTO の初期値は「RTT」（Round Trip Time）と呼ばれる「パケットが相手まで往復する時間」の「４倍＋α」となっています。RTO の値は再送を行うたびに２倍に増やされていきます。

上図 **A** の SYN 再送時、RTT 値は設定されていないので、CFG_TCP_RTO_INI（3 秒）を使用します。上図 **B** のデータ再送では前回の送信成功を元に計算された RTT 値、500 ミリ秒を使用しています。

RTO の範囲は CFG_TCP_RTO_MIN（500 ミリ秒）から CFG_TCP_RTO_MAX（60 秒）に設定されています。

(6)　接続タイムアウト

接続タイマシーケンスは下図のようになります。

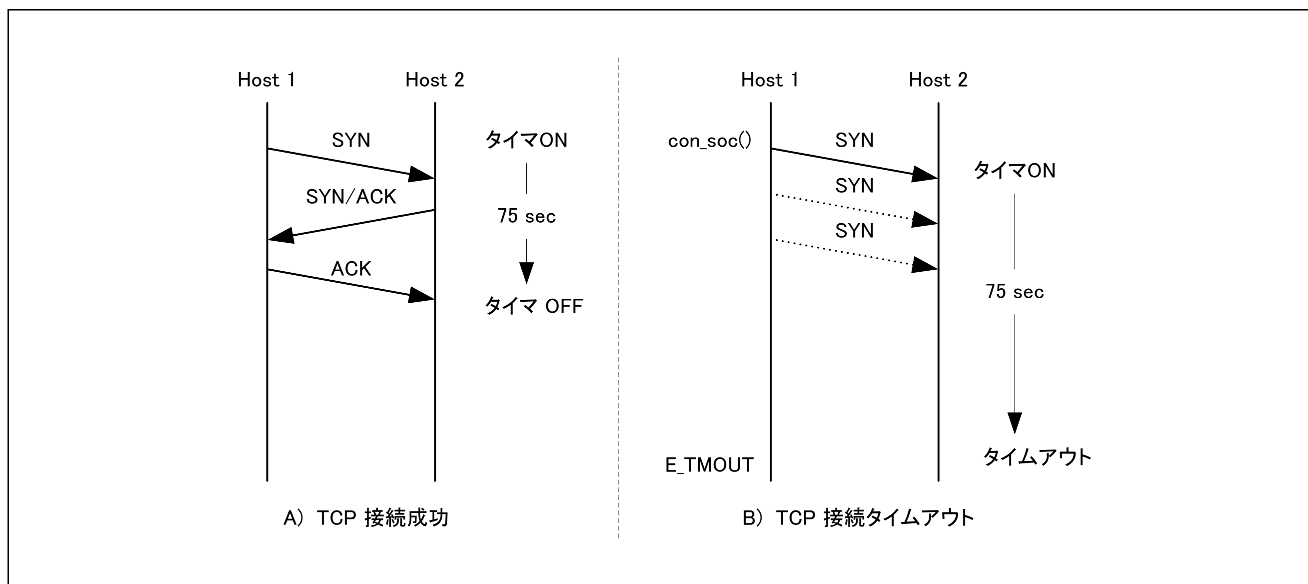


図 3.6　　接続タイムアウト例

con_soc()呼出し時に、このタイマは起動し 3 ウェイハンドシェイクがタイムアウトまでに完了すれば、E_OK を返します（A）。タイムアウトしたら E_TMOUT を返します（B）。

接続処理（3 ウェイハンドシェイク）のタイムアウト値は CFG_TCP_CON_TMO（75 秒）に設定されています。

※TCP ソケットは作成時、接続用ブロッキングタイムアウトを指定することができます。この値がタイムアウトした場合、接続処理は直ちに中断され con_soc()は E_TMOUT を返します。

(7)　送信タイムアウト

送信タイムアウトは CFG_TCP_SND_TMO（64 秒）に設定されています。データ通信中、CFG_TCP_SND_TMO を経っても相手から応答がない場合は接続を切断します。

(8)　切断タイムアウト

切断処理のタイムアウトは CFG_TCP_CLS_TMO（64 秒）に設定されています。cls_soc()が CFG_TCP_CLS_TMO までに完了しなければ、接続は強制切断され cls_soc()は E_TMOUT を返します。

※TCP ソケットは作成時、切断用ブロッキングタイムアウトを指定することができます。この値がタイムアウトした場合、切断処理は直ちに中断され cls_soc()は E_TMOUT を返します。

(9) 遅延 ACK タイムアウト

遅延 ACK タイムアウトは CFG_TCP_ACK_TMO（200 ミリ秒）に設定されています。

(10) TCP 輻輳制御

TCP/IP スタックは高速再送/高速復帰をサポートしています。重複 ACK 数は CFG_TCP_DUP_CNT(4)に設定されています。

(11) Maximum Segment Size（MSS）

MSS は CFG_TCP_MSS（1460 バイト）に設定されています。

(12) Keep Alive 機能

TCP/IP スタックは KeepAlive 機能をサポートしています。

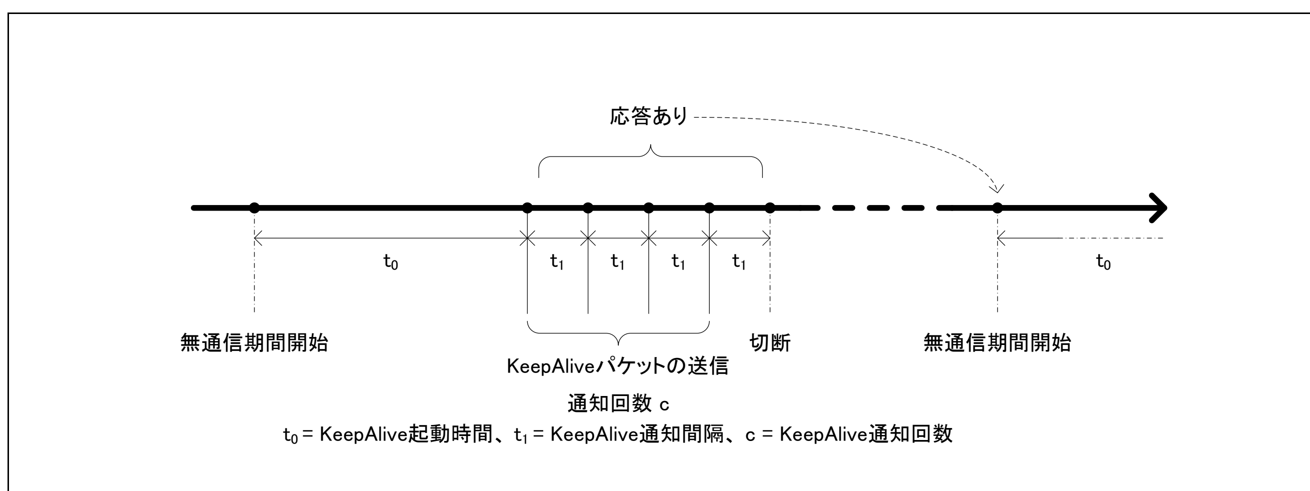


図 3.7 KeepAlive パケット動作

KeepAlive 機能が有効な場合（c＞0）、無通信状態で t_0 時間を経過すると KeepAlive パケットを接続先ホストに送信します。その後接続先から応答を得るか、c 回の再送を繰り返すまで t_1 時間間隔で KeepAlive パケットの送信を続けます。

KeepAlive パケットを c 回再送して応答が無い場合、接続先ホストとの TCP コネクションを切断します。接続先ホストから応答が有った場合、TCP コネクションの接続を維持します。（上図右の無通信期間開始に遷移）

KeepAlive 機能が無効な場合（c ＝0）、TCP コネクションは自動的に切断することはありません。

3.2 ネットワークデバイスドライバ

プロトコルスタックは **T_NET_DEV 構造体**を通じてデバイスドライバにアクセスしますので、予め **T_NET_DEV 構造体**にデバイス名、デバイス番号、デバイスドライバの関数といった情報を登録しておきます。プロトコルスタックはデバイス番号により **T_NET_DEV** に登録されたデバイスを特定しアクセスします。

3.2.1 デバイス構造体

```
typedef struct t_net_dev {
        UB        name[8];     /*デバイス名*/
        UH        num;         /*デバイス番号*/
        UH        type;        /*デバイスタイプ*/
        UH        sts;         /*予約*/
        UH        flg;         /*予約*/
        FP        ini;         /* dev_ini 関数へのポインタ*/
        FP        cls;         /* dev_cls関数へのポインタ*/
        FP        ctl;         /* dev_ctl関数へのポインタ*/
        FP        ref;         /* dev_ref関数へのポインタ*/
        FP        out;         /* dev_snd関数へのポインタ*/
        FP        cbk;         /* dev_cbk関数へのポインタ*/
        UW        *tag;        /*予約*/
        union     cfg;         /* MACアドレス*/
        UH        hhdrsz;      /*デバイスヘッダーサイズ*/
        UH        hhdrofs;     /*ネットワークバッファ書き込み位置*/
        VP        opt;         /* ドライバ拡張領域*/
}T_NET_DEV
```

(1) デバイス番号

デバイスを特定するためにユニークな番号をセットします。プロトコルスタックはこの番号を使ってデバイスにアクセスします。デバイス番号は必ず 1 から連番でつける必要があります。

(2) デバイス名

デバイスを特定するために名前をセットします。デバイス名の長さは 8 バイト以内です。

例）eth0、eth1 など。

(3) デバイスタイプ

ネットワークデバイスのタイプをセットします。以下のようなものがあります。

デバイスタイプ	意味
NET_DEV_TYPE_ETH	Ethernetデバイス
NET_DEV_TYPE_PPP	PPPデバイス

(4) デバイスドライバ関数

デバイスドライバは以下に示す関数をサポートする必要があります。これらの関数はプロトコルスタックから適宜呼び出されます。

プロトタイプ	説明	必須
ER dev_ini(UH dev_num)	デバイスの初期化	必須
ER dev_cls(UH dev_num)	デバイスの解放	必須ではない
ER dev_snd(UH dev_num, T_NET_BUF *pkt)	パケットをネットワークに送信	必須
ER dev_ctl(UH dev_num, UH opt, VP val)	デバイスの制御	必須ではない
ER dev_ref(UH dev_num, UH opt, VP val)	デバイス状態取得	必須ではない
void dev_cbk(UH dev_num, UH opt, VP val)	デバイスからのイベント通知 （コールバック関数）	必須ではない

(5) MAC アドレス

ハードウェアを特定するためのユニークな値をセットします。

```
union{
    struct{
            UB      mac[6];   /*MAC アドレス*/
    }eth;
}cfg;
```

(6) デバイスヘッダーサイズ

ネットワークデバイスのヘッダーサイズをセットします。

(7) ネットワークバッファ書き込み位置

ネットワークバッファのデータのオフセットをセットします。

(8) ドライバ拡張領域

デバイスドライバ固有のパラメータです。未使用の場合は NULL(0)をセットします。

3.2.2 インタフェース

dev_ini	デバイスの初期化

【書式】

ER ercd = dev_ini(UH dev_num);

【パラメータ】

UH	dev_num	デバイス番号

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	デバイス番号が不正
E_OBJ	既に初期化済み
E_PAR	T_NET_DEVに不正な値が設定された
<0	その他エラー（実装依存）

【解説】

デバイスの初期化を行います。この関数はプロトコルスタックからデバイスを初期化するために呼ばれます。この関数が呼ばれる前に T_NET_DEV にデバイス情報が登録されている必要があります。

dev_cls	デバイスの解放

【書式】

ER ercd = dev_cls(UH dev_num);

【パラメータ】

UH	dev_num	デバイス番号

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	デバイス番号が不正
E_OBJ	既に解放済み

【解説】

デバイスを解放します。

dev_ctl　デバイスの制御

【書式】

```
ER ercd = dev_ctl(UH dev_num, UH opt, VP val);
```

【パラメータ】

UH	dev_num	デバイス番号
UH	opt	制御コード
VP	val	設定する値

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	デバイス番号が不正
E_PAR	不正なパラメータ
E_OBJ	既に解放済み

【解説】

この関数の動作は実装依存になります。

dev_ref　デバイスの状態取得

【書式】

```
ER ercd = dev_ref(UH dev_num, UH opt, VP val);
```

【パラメータ】

UH	dev_num	デバイス番号
UH	opt	状態コード
VP	val	取得する値

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	デバイス番号が不正
E_PAR	不正なパラメータ
E_OBJ	既に解放済み

【解説】

この関数の動作は実装依存になります。

dev_snd	デバイスの送信

【書式】

```
ER ercd = dev_snd(UH dev_num, T_NET_BUF *pkt);
```

【パラメータ】

UH	dev_num	デバイス番号
T_NET_BUF	*pkt	ネットワークパッファへのポインタ

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_WBLK	パケットはキューへ登録された（エラーではない）
E_ID	デバイス番号が不正
E_PAR	不正なパラメータ
E_TMOUT	パケットの送信がタイムアウト
E_OBJ	既にデバイス状態が不正

【解説】

この関数はパケットを Ethernet に送信します。

実装例

```
ER dev_snd(UH dev_num, T_NET_BUF *pkt)
{
  /* Ethernet フレーム（IP/TCP/UDP）をコピー */
  memcpy(txframe,  pkt->hdr,  pkt->hdr_len);
  /* ネットワークへ送信 */
  xmit_frame(txframe);
  return E_OK;
}
```

上の例では、プロトコルスタックの処理がデバイスドライバによってブロッキングされてしまいます。次の例ではキューを使いブロッキングしない例を示します。

ノンブロッキング例

```
ER dev_snd(UH dev_num, T_NET_BUF *pkt)
{
  queue_tx(pkt);      /* パケットをキューに登録 */
  return E_WBLK;  /* ノンブロッキング */
}

void queue_tx_task(void)
{
  dequeue_tx(pkt); /* キューからパケット取り出し */
  /* Ethernet フレーム（IP/TCP/UDP）をコピー */
  memcpy(txframe,  pkt->hdr,  pkt->hdr_len);
  xmit_frame(txframe);  /* ネットワークへ送信 */
  if (transmission timeout) {
    pkt->ercd = E_TMOUT;   /* タイムアウトセット */
  }

  return E_OK;
}
```

dev_snd では送信処理は行わず、パケットはキューに登録して E_WBLK を返します。実際のパケット送信処理は別タスクで行い、ネットワークバッファの解放もそこで行うようにします。

dev_cbk	デバイスのイベント通知

【書式】

```
void dev_cbk(UH dev_num, UH opt, VP val);
```

【パラメータ】

UH	dev_num	デバイス番号
UH	opt	イベントコード
UH	val	イベント値

【戻り値】

なし

【エラーコード】

なし

【解説】

デバイスドライバからアプリケーションにイベントを通知するための関数です。この関数は実装依存です。

3.2.3 パケットのルーティング

デバイスドライバから上位プロトコルスタックへパケットを転送するには、次の API を使用します。
※この API はアプリケーションからは使用できません。

net_pkt_rcv	プロトコルスタックへパケットの転送

【書式】

void net_pkt_rcv(T_NET_BUF *pkt);

【パラメータ】

T_NET_BUF	*pkt	ネットワークバッファへのポインタ

【戻り値】

なし

【エラーコード】

なし

【解説】

この関数は上位プロトコルへのパケットを転送します。次の例ではデバイスドライバから上位プロトコルスタックへパケットを転送する例を示します。

例

```
/* ネットワークバッファの確保 */
T_NET_BUF *pkt;
net_buf_get(&pkt, len, TMO);

/* 受信したEthernetヘッダをネットワークバッファへセット */
pkt->hdr = pkt->buf + 2;
pkt->hdr_len = ETH_HDR_SZ;
memcpy(pkt->hdr, rx_frame, pkt->hdr_len);

/* 受信したIPペイロードをネットワークバッファへセット */
pkt->dat= pkt->hdr + pkt->hdr_len;
pkt->dat_len = rx_frame_len – pkt->hdr_len;
memcpy(pkt->dat, rx_frame + pkt->hdr_len, pkt->dat_len);

/* デバイス情報のセット*/
pkt->dev = dev;

/* プロトコルスタックへネットワークバッファの送信 */
net_pkt_rcv(pkt);
```

ネットワークバッファの解放は net_pkt_rcv()内で行われます。net_pkt_rcv()はタスクコンテキストから呼ばれなければなりません。

3.2.4 ループバックインタフェース

TCP/IP スタックではパケットをネットワーク・デバイスドライバで折り返すループバックインタフェースを提供します。DDR_LOOPBACK_NET.c を使用することで、そのインタフェースから送信するパケットはTCP/IP スタックに通知されます。

一般の(127.0.0.1 で表される) ループバックインタフェースとは異なり、TCP/IP スタックでは静的な IP アドレスと MAC アドレスを設定します。TCP/IP スタックでループバックインタフェースを使用するには、コンフィグレータのインタフェースの登録時にデバイスタイプに「Loopback」を選択します。

ループバックインタフェースからパケットを受信するには、そのパケットの宛先が割り当てた IP アドレスでなければなりません。またループバックインタフェースを使用する場合でも ARP を使ったアドレス解決が行われます。

3.2.5 T_NET_DEV 情報の登録例

T_NET_DEV 情報の登録例を以下に示します。

T_NET_DEV 情報の登録例

```
T_NET_DEV gNET_DEV[] = {
    {
        "lan0",                 /* Device Name      */
        1,                      /* Device Number    */
        NET_DEV_TYPE_ETH,       /* Device Type      */
        0,                      /* Status           */
        0,                      /* Flags            */
        eth_ini,                /* Device Init      */
        eth_cls,                /* Device Close     */
        eth_ctl,                /* Device Configure */
        eth_sts,                /* Device Status    */
        eth_snd,                /* Device Transmit  */
        eth_cbk,                /* Device Callback  */
        0,
        {{{ 0x12, 0x34, 0x56, 0x78, 0x9A, 0xBC }}},    /* MAC Address */
        ETH_HDR_SZ,                                     /* Link Header Size */
        CFG_NET_BUF_OFFSET                              /* Network buffer data Offset */
    }
};
```

3.3 メモリ管理

プロトコルスタックではメモリ管理にネットワークバッファを使用しています。ネットワークバッファを使うことにより動的にメモリの空きブロックを確保することが可能になります。下図にメモリ確保の例を示します。まず始めにハードウェアからデータを受信したデバイスドライバはネットワークバッファ API を使用して、メモリを確保（net_buf_get）します。次に確保したメモリに必要情報をセットし、上位層のプロトコルスタックへパケットを送信（net_pkt_rcv）します。

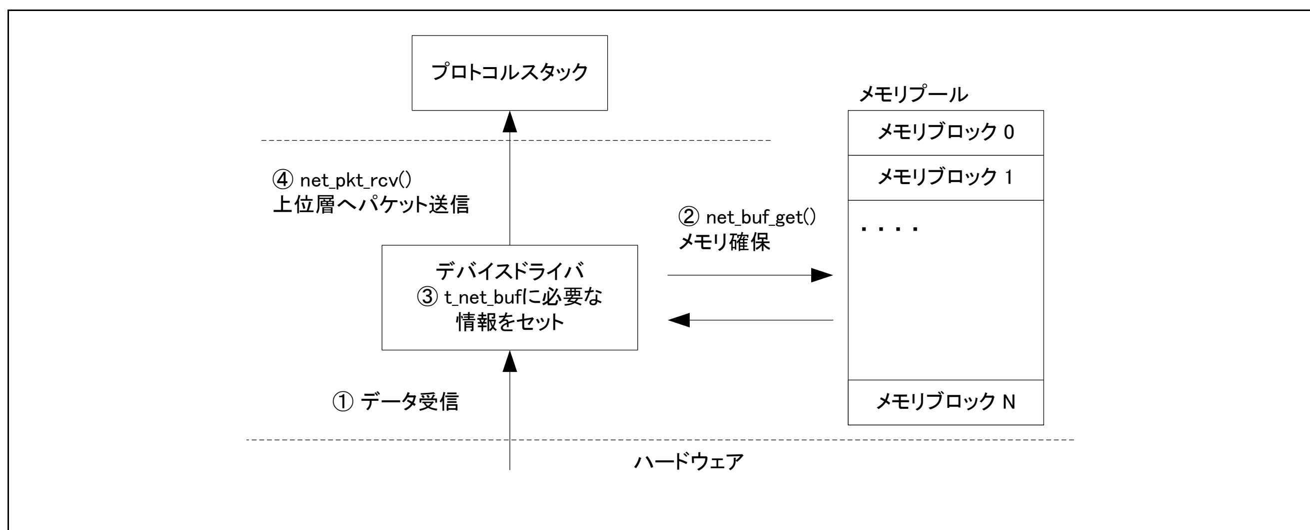


図 3.8 メモリ確保例の図

3.3.1 ネットワークバッファ

TCP/IP スタックではパケットの送受信にネットワークバッファを使用します。ネットワークバッファは OS が提供するメモリアロケータを使用して実装されます。ネットワークバッファサイズとネットワークバッファ数は通信量や MTU サイズに応じて各アプリケーションで定義が可能です。

TCP/IP スタックはネットワークバッファの初期化、取得、解放、終了の各手続に、net_memini(), net_memget(), net_memret(), net_memext() API をコールします。また取得可能なネットワークバッファの最大サイズをあらかじめ CFG_NET_BUF_SZ に設定する必要があります。

ネットワークバッファの構造（T_NET_BUF）

```
typedef struct t_net_buf {
        UW              *next;          /* 予約 */
        ID              mpfid;          /* メモリプールID */
        T_NET           *net;           /* ネットワーク・インタフェース */
        T_NET_DEV       *dev;           /* ネットワークデバイス */
        T_NET_SOC       *soc;           /* ソケット */
        ER              ercd;           /* エラーコード */
        UH              flg;            /* プロトコルスタック制御用フラグ */
        UH              seq;            /* フラグメントシーケンス */
        UH              dat_len;        /* パケットのデータサイズ */
        UH              hdr_len;        /* パケットのヘッダーサイズ */
        UB              *dat;           /* パケット(buf)内のデータ位置を指す */
        UB              *hdr;           /* パケット(buf)内のヘッダ位置を指す */
        UB              buf[];          /* 実際のパケット */
} T_NET_BUF ;
```

各プロトコル間、プロトコルとデバイスドライバ間のパケットの送受信には、T_NET_BUF を使用します。
TCP/IP で実際のパケットデータは‘buf’に格納されており、‘*dat’、‘*hdr’、‘hdr_len’ ‘dat_len’は、それにアクセスするために使用します。

(1) ネットワークバッファを用いたプロトコルスタックとデバイスドライバ間のアクセス

メンバ	送信(プロトコルスタック->ドライバ)	受信(ドライバ->プロトコルスタック)
dev	&gNET_DEV[dev_num-1]を設定します。 dev_num は、アプリケーションで指定したものが設定されます。	&gNET_DEV[eth_dev_num-1]を設定します。 eth_dev_num は、初期化時に通知された値を使用します。
ercd	ドライバの送信処理で失敗した場合、失敗要因が設定されます。 成功した場合は、プロトコルスタックが設定した送信サイズのままとします。	使用しません。
flg	ハードウェアチェックサムの有効／無効を制御します。 - HW_CS_TX_IPH4(0x0040) (IP ヘッダのチェックサム) - HW_CS_TX_DATA(0x0080) (ペイロードデータのチェックサム)	ハードウェアチェックサムの有効／無効を制御します。 - HW_CS_TX_IPH4(0x0040) (IP ヘッダのチェックサム) - HW_CS_RX_DATA(0x0020) (ペイロードデータのチェックサム) 上記のチェックサムの結果がエラーだった場合、次のビットをセットしてください。 - HW_CS_IPH4_ERR(0x0100) (IP ヘッダのチェックサムエラー) - HW_CS_DATA_ERR(0x0200) (ペイロードデータのチェックサムエラー)
hdr/hdr_len	プロトコルスタックが送信するフレームデータの先頭アドレスとサイズが設定されます。ドライバは、hdr 位置から hdr_len 分のデータを送信します。	ドライバにより受信したフレームデータの先頭アドレスとヘッダのサイズが設定されます。 (Ethernet フレームの場合、ヘッダサイズは 14Byte です)
dat/dat_len	使用しません。	ドライバにより受信したフレームデータのヘッダに続くデータ部の先頭アドレスとデータサイズが設定されます。 (Ethernet フレームの場合、hdr 位置から hdr_len 分ずらした値が dat 位置になります)
buf[]	実際のパケットデータが格納されます。 buf[0]と buf[1]は 4Byte アライメント制御用のバッファです。 送受信するデータは、buf[2]から書き込み可能です。	

3.3.2 ネットワークバッファ API

※このネットワークバッファ **API** はアプリケーションから使用することはできません。

net_buf_get	ネットワークバッファの確保

【書式】

```
ER ercd = net_buf_get(T_NET_BUF **buf, UH len, TMO tmo);
```

【パラメータ】

T_NET_BUF	**buf	メモリ確保するバッファのアドレス
UH	len	確保するバイト数
UH	tmo	タイムアウト指定

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータ値が設定された
E_NOMEM	メモリ確保できない
E_TMOUT	タイムアウト

【解説】

メモリプールよりメモリを確保します。確保したメモリのアドレスは buf に返します。

net_buf_ret	ネットワークバッファの解放

【書式】

```
void net_buf_ret(T_NET_BUF *buf);
```

【パラメータ】

T_NET_BUF	*buf	メモリ解放するバッファのアドレス

【戻り値】

なし

【エラーコード】

なし

【解説】

メモリプールにメモリを返却します。ネットワークバッファとソケットが関連づけられている場合は、ソケットにメモリ解放イベントを通知します。

3.4 メモリ I/O 処理

TCP/IP スタックは動作するデバイスやコンパイル環境に依存しないように、プロトコル処理で発生する連続したメモリへの書き込みや、比較処理はユーザ側で定義します。たとえば DMA 機能を備えたデバイスの場合、標準ライブラリ関数である memcpy()は使用せず、DMA 転送でメモリコピーを実行することができます。

3.4.1 メモリ I/O API

※このメモリ I /O API は必ずアプリケーションで定義する必要があります。

net_memset	メモリの値設定

【書式】

```
void* net_memset(void* d, int c, SIZE n);
```

【パラメータ】

void*	d	設定するメモリの先頭アドレス
int	c	設定する値
SIZE	n	設定バイト数

【戻り値】

void*	d	設定するメモリの先頭アドレス

【解説】

メモリの設定が正常に終了した場合は、引数で指定されるメモリの先頭アドレスを返却して下さい。

net_memcpy	メモリのコピー

【書式】

```
void* net_memcpy(void* d, const void* s, SIZE n);
```

【パラメータ】

void*	d	コピー先アドレス
const void*	s	コピー元アドレス
SIZE	n	コピーバイト数

【戻り値】

void*	d	コピー先アドレス

【解説】

メモリのコピーが正常に終了した場合は、引数で指定されるコピー先アドレスを返却して下さい。

net_memcmp	メモリの比較

【書式】

```
int net_memcmp(const void* d, const void* s, SIZE n);
```

【パラメータ】

const void*	d	比較メモリアドレス1
const void*	s	比較メモリアドレス2
SIZE	n	比較バイト数

【戻り値】

int	比較結果

【解説】

両メモリから指定されたバイト数分、同じ値の場合は 0 を返却して下さい。そうでない場合は、非 0 を返却して下さい。

3.5 Ethernet デバイスドライバ

R-IN32 に内蔵されている Ethernet MAC および Ethernet Switch のデバイスドライバです。ネットワークデバイスドライバから呼び出されて使用されます。

本ドライバで実装している主な機能は以下の通りです。
・PHY モード設定・参照
・PHY スピード設定・参照
・受信フレームフィルタリング設定
・マルチキャストアドレスフィルタの動的設定
・Raw データ送受信 API
・Direct MAC モード・Ether Switch モード
・ブロッキング送信・ノンブロッキング送信
・VLAN

3.5.1 Ethernet デバイスドライバ構成

Ethernet デバイスドライバは、MDIO インタフェースを制御する PHY ドライバと組み合わせて使用します。ドライバの構成を以下に示します。

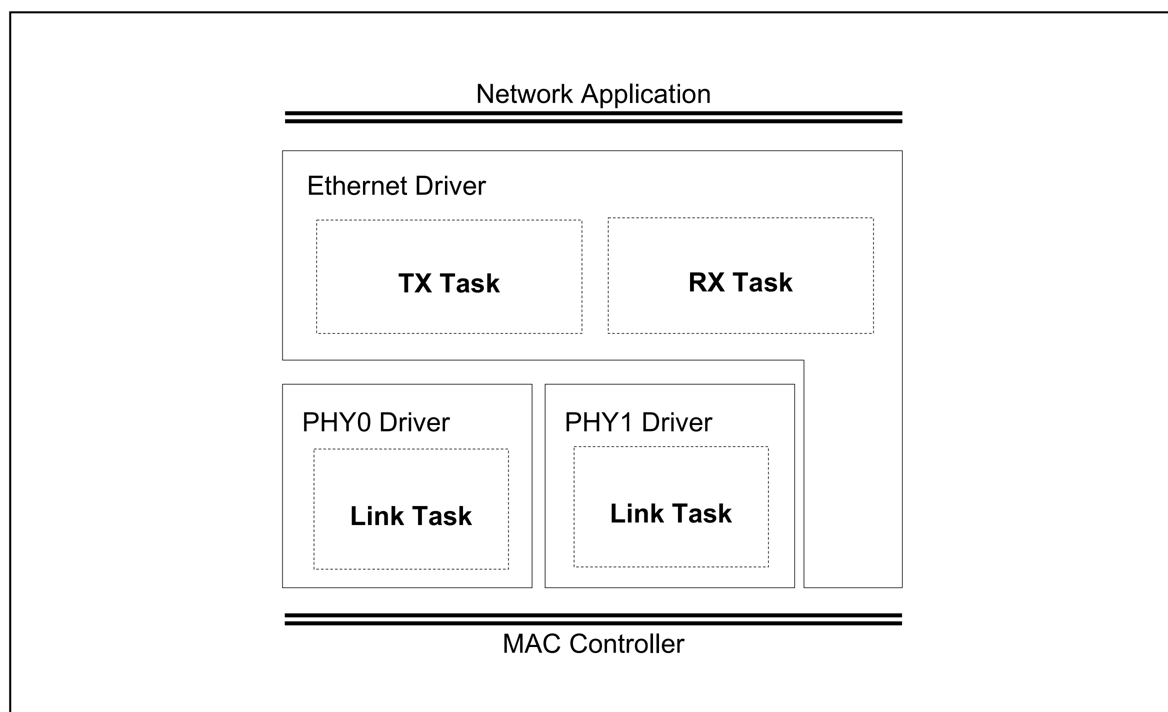


図 3.9 Ethernet デバイスドライバ構成

3.5.2 Ethernet デバイスドライバ API

3.5.2.1 Ether ドライバ初期化 eth_ini()

【プロトタイプ】

ER eth_ini(UH dev_num)

【処理】

Etherドライバの初期化

【引数】

UH	dev_num	デバイス番号(1)

【戻り値】

ER	E_OK	初期化成功
	E_ID	デバイス番号未指定
	E_PAR	デバイス番号不正
	その他	タスク起動エラー、PHYドライバ初期化エラー

【備考】

PHY, MACコントローラを初期化します。またEtherドライバを制御するタスクを起動します。Etherドライバを使用するには、本関数を必ず事前に呼ぶ必要があります。

3.5.2.2 Ether ドライバ終了　eth_cls()

【プロトタイプ】

ER eth_cls(UH dev_num)

【処理】

Etherドライバの終了

【引数】

UH	dev_num	デバイス番号(1)

【戻り値】

ER	E_OK	終了

【備考】

PHY, MACコントローラを終了します。

3.5.2.3 Ether フレーム送信 eth_snd()

【プロトタイプ】

ER eth_snd(UH dev_num, T_NET_BUF *pkt)

【処理】

Etherフレームの送信

【引数】

UH	dev_num	デバイス番号(1)
UH	pkt->hdr	送信データアドレス
UB*	pkt->hdr_len	送信データ長

【戻り値】

ER	E_OK	送信処理成功
	E_TMOUT	リンクダウン
	E_WBLK	ノンブロッキング受付 (ノンブロッキング送信時のみ)
	E_NOMEM	メモリ不足
	E_SYS	MACコントローラエラー

【備考】

ブロッキング送信の場合、関数は送信完了割り込みを待って戻ります。ノンブロッキング送信の場合で送信処理に成功すると関数はE_WBLKを返却します。E_WBLKが返却された場合、EtherドライバはEtherフレームの送信完了時に、フレーム送信完了通知 eth_raw_snddone()をコールバックします。

3.5.2.4 フレーム送信完了通知 eth_raw_snddone()

【プロトタイプ】

void eth_raw_snddone(T_NET_BUF *pkt)

【処理】

Etherフレーム送信完了時のコールバック関数

【引数】

T_NET_BUF*	pkt	eth_snd()に指定したpkt
ER	pkt->ercd	送信結果

【戻り値】

void

【備考】

ノンブロッキング送信が完了した時点で、Etherドライバから呼ばれます。本関数はアプリケーションが事前に登録する必要があります。関数の登録は次のようにします。

```
/* 送信完了通知関数本体 */
void eth_raw_snddone(T_NET_BUF *pkt)
{
}

/* 送信完了通知関数の登録 */
eth_ctl(1, ETH_OPT_RAW_SNDDONE, (VP)eth_raw_snddone);
```

本関数はEtherドライバのTX Task上で動作するためタスクを停止しないで下さい。

3.5.2.5 Ether フレーム受信通知 eth_raw_rcv()

【プロトタイプ】

void eth_raw_rcv(VP fram, UH len)

【処理】

Etherフレーム受信時の通知関数

【引数】

VP	fram	受信データアドレス
UH	len	受信データ長

【戻り値】

void

【備考】

Ehter ドライバがフレームを受信すると本関数を呼び出します。本関数はアプリケーションが事前に登録する必要があります。関数の登録は次のようにします。

```
/* 受信通知関数本体 */
void eth_raw_rcv(VP p, UH len)
{
}

/* 受信通知関数の登録 */
eth_ctl(1, ETH_OPT_RAW_RXFNC, (VP)eth_raw_rcv);
```

本関数は Ether ドライバの RX Task 上で動作するためタスクを停止しないで下さい。Ether ドライバを TCP/IP スタックと共有している場合、先に本関数が呼ばれます。そのため受信フレームの内容は変更できません。

3.5.2.6 PHY 能力設定 set_phy_mode()

【プロトタイプ】

ER set_phy_mode(PHY_MODE *mode)

【処理】

PHY能力の設定

【引数】

PHY_MODE*	mode	PHY能力
UW	mode->mode	速度・デュプレックス
UB	mode->nego	オートネゴシエーション有(TRUE)・無(FALSE)
UB	mode->ch	設定対象PHY

【戻り値】

ER	E_OK	設定成功
	E_PAR	入力値不正
	その他	PHYドライバ能力設定エラー

【備考】

アプリケーションは次のようにして、任意のタイミングでPHY能力を設定することができます。速度・デュプレックスに設定する値は、表3.1から選択します。設定対象PHYにはETH_INT_MII_PHY0、ETH_INT_MII_PHY1のいずれを設定します。

```
struct PHY_MODE pmod;

pmod.mode = LAN_AUTO_ABILITY;
pmod.nego = TRUE;
pmod.ch = ETH_INT_MII_PHY0;

set_phy_mode(&pmod);
```

表 3.1 PHY 能力設定

設定値	設定内容
LAN_10T_HD	10M/半 2 重
LAN_10T_FD	10M/全 2 重
LAN_100TX_HD	100M/半 2 重
LAN_100TX_FD	100M/全 2 重
LAN_1000T_HD	1G/半 2 重
LAN_1000T_FD	1G /全 2 重
LAN_AUTO_ABILITY	自動選択

3.5.2.7 PHY 能力取得 get_phy_mode()

【プロトタイプ】

ER get_phy_mode(PHY_MODE *mode)

【処理】

PHY能力の取得

【引数】

PHY_MODE*	mode	PHY能力

【戻り値】

ER	E_OK	設定成功
	E_PAR	入力値不正
	その他	PHYドライバ能力取得エラー

【備考】

PHY能力(速度、デュプレックス、オートネゴシエーション有無)、それにリンク状態を取得します。アプリケーションは次のようにして、任意のタイミングでPHY能力を取得することができます。

```
struct PHY_MODE pmod = {0};

/* 対象 PHY を ETH_INT_MII_PHY0 もしくは ETH_INT_MII_PHY1 で設定します*/
pmod.ch = ETH_INT_MII_PHY0;
get_phy_mode(&pmod);
```

pmod.modeにはPHY能力(表3.1参照)が設定されます。
pmod.negoにはオートネゴシエーションの有効(TRUE)、無効(FALSE)が設定されます。
pmod.linkにはリンクアップ(TRUE)、リンクダウン(FALSE)が設定されます。

リンクダウン時、PHY能力は正常な値が取得されませんのでご注意ください。

3.5.2.8 マルチキャストアドレスフィルタモード設定 set_mcast_filter_mode()

【プロトタイプ】

ER set_mcast_filter_mode(UINT mode)

【処理】

マルチキャストアドレスフィルタモードの設定

【引数】

UINT	mode	フィルタモード

【戻り値】

ER	E_OK	設定成功
	E_PAR	入力値不正

【備考】

フィルタモードには次の３つのモードを設定します。

設定値	モード
MCRX_MODE_ALLOW	マルチキャストアドレス全受信モード
MCRX_MODE_DENY	マルチキャストアドレス受信拒否モード
MCRX_MODE_FILTER	指定マルチキャストアドレス受信モード

注意1 個別のアドレスを設定する場合は、MCRX_MODE_FILTERを選択します。

注意2 MCRX_MODE_DENYを選択した場合、すでに登録されているマルチキャストアドレスはすべて削除されます。

3.5.2.9 マルチキャストアドレスフィルタ追加 add_mcast_filter()

【プロトタイプ】

ER add_mcast_filter(MAC_FILTER *adr)

【処理】

受信するマルチキャストアドレスの追加

【引数】

MAC_FILTER*	adr	受信するマルチキャストアドレス
UB	adr->mac[6]	マルチキャストアドレス
UB	adr->bitnum	有効ビット数(0, 40-48)

【戻り値】

ER	E_OK	設定成功
	E_PAR	入力値不正

【備考】

追加するマルチキャストアドレスを受信するには、あらかじめフィルタモードがMCRX_MODE_FILTERに設定されている必要があります。

マルチキャストMACアドレスはクラスD(第1オクテッドが0xE0～0xEF(224～239))のIPアドレスの下位23ビットと、固定の上位25ビット(0x01.0x00.0x5e.0x00/25)を組み合わせて表します。アプリケーションは参加するマルチキャストIPアドレスから、受信を希望するMACアドレスを導き出す必要があります。有効ビット数には40ビット(5オクテッド)からビット単位で48ビットまで指定可能です。0を指定した場合は有効ビットなし(48と等価)と見なします。

MACアドレス01:00:5e:00:01:* (*は任意)をすべて受信するには次のようにします。

```
MAC_FILTER adr;
UB macadr[] = {0x01, 0x00, 0x5e, 0x00, 0x01, 0x00};

memcpy(&adr.mac[0], macadr, 6);
adr.bitnum = 40;
add_mcast_filter(&adr);
```

※ 全ノードを示すマルチキャストアドレス(224.0.0.1)も追加しなければ受信することはできませんのでご注意下さい。

3.5.2.10 Ether ドライバイベントの通知

【プロトタイプ】

void eth_cbk(UH dev_num, UH opt, VP val)

【処理】

Etherドライバからアプリケーショ

【引数】

UH	dev_num	デバイス番号
UH	opt	イベント種別
VP	val	イベント内容

【戻り値】

void

【備考】

本関数はネットワークアプリケーション側で定義する関数です。ネットワークアプリケーションはデバイスコンフィグレーション(T_NET_DEV gNET_DEV[])の cbk メンバにコールバック関数を登録することで、非同期に発生するイベントに対する制御やアプリケーション固有の処理が可能になります。

イベント種別やイベント内容には次のものがあります。

イベント種別	イベント内容	発生契機
EV_CBK_DEV_INIT	常に 0	Ether ドライバ初期化時
EV_CBK_DEV_LINK	リンクダウン 0，アップ 1	リンク変化時

注意 現状ではリンクイベントの通知は行いません。

3.5.2.11 Ether ドライバオプション設定 eth_ctl()

【プロトタイプ】

ER eth_ctl(UH dev_num, UH opt, VP val)

【処理】

Etherドライバのオプション設定

【引数】

UH	dev_num	デバイス番号
UH	opt	オプション種別
VP	val	設定値

【戻り値】

ER	E_OK	設定成功
	E_PAR	入力値不正

【備考】

EtherドライバやPHYドライバの設定関数を呼び出します。現状設定可能なオプションは以下の通りです。

オプション種別	オプション内容	設定可能値
ETH_OPT_PHY_MODE	PHY 能力設定	set_phy_mode()同等
ETH_OPT_MCRX_MODE	マルチキャストアドレスフィルタモード設定	set_mcast_filter_mode()同等
ETH_OPT_MCRX_ADR	マルチキャストアドレスフィルタ追加	add_mcast_filter()同等
ETH_OPT_RAW_RXFNC	Ether フレーム受信通知	eth_raw_rcv()同等
ETH_OPT_RAW_SNDDONE	フレーム送信完了通知	eth_raw_snddone()同等

3.5.2.12 Ether ドライバオプション取得 eth_sts()

【プロトタイプ】

ER eth_sts(UH dev_num, UH opt, VP val)

【処理】

Etherドライバのオプション取得

【引数】

UH	dev_num	デバイス番号
UH	opt	オプション種別
VP	val	取得内容

【戻り値】

ER	E_OK	取得成功
	E_PAR	入力値不正

【備考】

EtherドライバやPHYドライバの取得関数を呼び出します。現状取得可能なオプションは以下の通りです。

オプション種別	オプション内容	設定可能値
ETH_OPT_PHY_MODE	PHY 能力取得	get_phy_mode()同等

3.5.3 コンフィグレーション

以下のコンフィグレーションを、用途に合わせて DDR_ETH_CFG.h に記述します。

#define PHY_ADR0
R-IN32M4-CL3 使用 LAN1 用 PHY アドレス

#define PHY_ADR1
R-IN32M4-CL3 使用 LAN2 用 PHY アドレス

#define PROMISCUOUS_FILTER_MODE
プロミスキャスフレームフィルタモード。(0:すべてのフレームを受信する, 1:自ステーションのみ受信する)

#define ETH_EARLY_TX_ENA
Early 転送モード。(0:無効, 1:有効)
Early 転送モードが有効な場合、送信データが MAC コントローラの送信 FIFO に転送された時点で送信完了とみなします。

#define ETH_TX_ASYNC
ノンブロッキング転送モード。(0:ブロッキングモード, 1:ノンブロッキングモード)
ノンブロッキング転送モードが有効な場合、送信処理は Ether ドライバの送信タスクによって実行されます。

#define USE_ETHSW
Ether スイッチの使用有無。(0:使用しない, 1:使用する)

#define USE_ETHSW_MGTAG
Ether スイッチマネージメントタグの使用有無。(0:使用しない, 1:使用する)

3.5.4 Ethernet デバイスドライバの注意事項

3.5.4.1 TCP/UDP のハードウェアチェックサム機能

Ethernet デバイスドライバでは、R-IN32 のハードウェアチェックサム機能を使用しています。そのため、プロトコルスタック内でのチェックサム処理は行っておりません。

プロトコルスタックでチェックサム処理を行うか否かは、T_NET_BUF 構造体のメンバ flg と、グローバル変数 gNet[デバイス番号-1].flag で制御しています。

ハードウェアチェックサム機能を有効にしている箇所を、以下に示します。

受信ハードウェアチェックサムの有効化

```
T_NET_BUF *pkt;

pkt->flg    |= (HW_CS_RX_IPH4 | HW_CS_RX_DATA);
```

送信ハードウェアチェックサムの有効化

```
T_NET      *net;

net = &gNET[eth_devnum-1];

/* set hardware checksum flag */
net->flag |= (HW_CS_TX_IPH4 | HW_CS_TX_DATA);
```

3.6 PHY ドライバ

Ether ドライバと PHY ドライバの I/F は広域変数 PHY_IO gPHY_IO[]に定義します。gPHY_IO[]は複数のPHY を定義することができる配列です。

この変数に登録された PHY ドライバの API は、適宜 Ether ドライバからコールされます。最後の要素には PHY_IO のメンバをすべて 0 に設定した終端を設定する必要があります。PHY_IO のメンバは以下の通りです。

```
typedef struct phy_io {
    UW phy_id;
    UW phy_adr;
    ER (*phy_ini)(ID flg, UW id, UW adr);
    ER (*phy_ext)(void);
    ER (*phy_set_mode)(UW mode, UB nego);
    ER (*phy_get_mode)(UW *mode, UB *nego, UB *link);
}PHY_IO;
```

メンバ	説明
phy_id	PHY ID(LAN1～LAN4 の該当する ID。LAN1 の場合は 1)を表します
phy_adr	PHY アドレス(回路依存)を表します
phy_ini	初期化関数
phy_ext	終了関数
phy_set_mode	PHY 能力設定関数
phy_get_mode	PHY 能力取得関数

3.6.1 PHY ドライバ API

3.6.1.1 PHY ドライバ起動 phy_ini()

【プロトタイプ】

ER phy_ini(ID flg, UW id, UW adr)

【処理】

PHYドライバの起動

【引数】

ID	flg	イベントフラグID
UW	id	PHY ID
UW	adr	PHYアドレス

【戻り値】

ER	E_OK	起動成功
	その他	タスク起動エラー

【備考】

本関数はeth_ini()関数実行時にEtherドライバから呼ばれます。イベントフラグIDはEtherドライバにリンクイベントを通知する場合に使用します。PHY IDはリンクイベント通知時のイベントパターンに設定します。

3.6.1.2 PHY ドライバ終了 phy_ext()

【プロトタイプ】

ER phy_ext(void)

【処理】

PHY ドライバの終了

【引数】

void

【戻り値】

ER	E_OK	起動成功
	その他	タスク終了エラー

【備考】

本関数はeth_cls()関数実行時にEtherドライバから呼ばれます。PHYドライバではリンクタスクを終了します。

3.6.1.3 PHY 能力設定 phy_set_mode()

【プロトタイプ】

ER phy_set_mode(UW mode, UB nego)

【処理】

PHY能力設定

【引数】

UW	mode	速度・デュプレックス
UB	nego	オートネゴシエーション有(TRUE)・無(FALSE)

【戻り値】

ER	E_OK	設定終了

【備考】

本関数はset_phy_mode()関数実行時にEtherドライバから呼ばれます。速度・デュプレックスには表3.2の値が指定されます。この値はEtherドライバAPIのset_phy_mode()に指定する速度・デュプレックスの値とは異なります。

PHYドライバ起動時の設定値は、PHY_AUTO_ABILITYです。

表 3.2 PHY 能力設定

設定値	設定内容
PHY_10T_HD	10M/半 2 重
PHY_10T_FD	10M/全 2 重
PHY_100TX_HD	100M/半 2 重
PHY_100TX_FD	100M/全 2 重
PHY_1000T_HD	1G/半 2 重
PHY_1000T_FD	1G /全 2 重
PHY_AUTO_ABILITY	自動選択

3.6.1.4 PHY 能力取得 phy_get_mode()

【プロトタイプ】

```
ER phy_get_mode(UW *mode, UB *nego, UB *link)
```

【処理】

PHY能力取得

【引数】

UW*	mode	速度・デュプレックス
UB*	nego	オートネゴシエーション有無
UB*	link	リンク有無

【戻り値】

ER	E_OK	設定終了

【備考】

本関数はget_phy_mode()関数実行時にEtherドライバから呼ばれます。速度・デュプレックスには表3.2の値が設定されます。この値はEtherドライバAPIのset_phy_mode()に指定する速度・デュプレックスの値とは異なります。

3.6.2 リンクイベントの通知

Ether ドライバの RX Task ではフレームの受信とリンク状態の変化をイベントフラグで待っています。リンク状態の変化を表すイベントは、PHY ドライバの Link Task もしくは割り込みハンドラによってセット(set_flg())します。

set_flg(id, ptn)のイベントフラグ ID(第 1 引数)には、PHY ドライバ起動時に指定されたイベントフラグ ID を用います。またイベントを表すビットパターン(第 2 引数)には、リンクイベントを表す定数「PHY_LINK_EVT」と、PHY ドライバ起動時に指定された PHY ID の論理和を設定します。

4. ネットワークコンフィグレーション

本章では、TCP/IP プロトコルスタックのコンフィグレーションについて説明します。

4.1 TCP/IP スタックのコンフィグレーション

コンフィグレーションファイル（net_cfg.c）を編集することで、ソケットや TCP/IP の通信に関連したコンフィグレーションを設定することができます。

4.1.1 コンフィグレーション一覧

コンフィグレーション可能な項目を以下のリストに示します。

アプリケーションは、#define CFG_XXX で定義されるマクロの値を編集します。それ以外のマクロや変数をアプリケーションが直接設定することはできません。また一部項目を除いて、各初期値は DEF_XXX マクロによって#define 定義されています。

(1/2)

コンフィグレーション [単位]	定義名	初期値	最小値	最大値
データリンクのデバイス数	CFG_NET_DEV_MAX	1	1	2
全プロトコルのソケットの最大数	CFG_NET_SOC_MAX	10	1	1000
TCP ソケットの最大数(※1)、(※3)	CFG_NET_TCP_MAX	5	0	1000
ARP キャッシュ数	CFG_NET_ARP_MAX	8	1	32
マルチキャストエントリ数	CFG_NET_MGR_MAX	8	1	100
IP フラグメントパケット並列キュー数	CFG_NET_IPR_MAX	2	1	16
ネットワークバッファ最大サイズ [Byte] (※2)	CFG_NET_BUF_SZ	8192	2048	8192
ネットワークバッファ数	CFG_NET_BUF_CNT	8	2	31
ネットワークバッファデータ書き込み位置(※2)	CFG_NET_BUF_OFFSET	2	42	42
MTU サイズ(※2)、(※4)	CFG_PATH_MTU	1500	576	1500
ARP リトライ回数(※4)	CFG_ARP_RET_CNT	3	0	10
ARP リトライ間隔 [ミリ秒] (※4)	CFG_ARP_RET_TMO	1000	500	10000
ARP キャッシュ生存時間 [ミリ秒] (※4)	CFG_ARP_CLR_TMO	1200000	1000	3600000
IP ヘッダ TTL 値 (※4)	CFG_IP4_TTL	64	1	255
IP ヘッダ TOS 値 (※4)	CFG_IP4_TOS	0	0	255
IP フラグメントパケット待ち時間 [ミリ秒] (※4)	CFG_IP4_IPR_TMO	10000	100	60000
マルチキャスト IP ヘッダ TTL 値(※4)	CFG_IP4_MCAST_TTL	1	1	255
IGMPv1 タイムアウト値 [ミリ秒] (※4)	CFG_IGMP_V1_TMO	40000	40000	120000
IGMP レポートタイムアウト値 [ミリ秒] (※4)	CFG_IGMP_REP_TMO	10000	10000	30000
MSS(TCP/IPv4) (MTU-IP ヘッダ-TCP ヘッダ) (※4)	CFG_TCP_MSS	1460	536	1460
TCP/RTO リトライタイムアウト初期値 [ミリ秒] (※4)	CFG_TCP_RTO_INI	3000	2000	3000
TCP/RTO リトライタイムアウト最小値 [ミリ秒] (※4)	CFG_TCP_RTO_MIN	500	200	500
TCP/RTO リトライタイムアウト最大値 [ミリ秒] (※4)	CFG_TCP_RTO_MAX	60000	30000	60000

(2/2)

コンフィグレーション [単位]	定義名	初期値	最小値	最大値
TCP 送信バッファサイズ [Byte] (※3)、(※4)、(※5)	CFG_TCP_SND_WND	1024	1024	8192
TCP 受信バッファサイズ [Byte] (※4)、(※5)	CFG_TCP_RCV_WND	1024	1024	8192
TCP 重複 ACK による再送閾値 [受信回数] (※4)	CFG_TCP_DUP_CNT	4	1	10
TCP/SYN 送信タイムアウト値 [ミリ秒] (※4)	CFG_TCP_CON_TMO	75000	10000	75000
TCP/データ送信タイムアウト値 [ミリ秒] (※4)	CFG_TCP_SND_TMO	64000	10000	64000
TCP/FIN 送信タイムアウト値 [ミリ秒] (※4)	CFG_TCP_CLS_TMO	75000	10000	75000
TCP/2MSL タイムアウト値 [ミリ秒] (※4) (未使用)	CFG_TCP_CLW_TMO	20000	0	20000
TCP/遅延 ACK 送信間隔 [ミリ秒] (※4)	CFG_TCP_ACK_TMO	200	200	1000
TCP/Keep-Alive 通知回数(0 の場合は Keep-Alive 無効)	CFG_TCP_KPA_CNT	0	0	100
TCP/Keep-Alive 通知間隔 [ミリ秒]	CFG_TCP_KPA_INT	1000	1000	60000
TCP/Keep-Alive 起動条件(無通信時間) [ミリ秒]	CFG_TCP_KPA_TMO	7200000	10000	14400000
受信パケットキューイング数(※4)	CFG_PKT_RCV_QUE	1	1	10
受信シーケンス保障キューイング数	CFG_TCP_RCV_OSQ_MAX	6	-	-
受信パケットチェックサム検証無効フラグ	CFG_PKT_CTL_FLG	0	-	-
ARP PROBE パケット送信待ち時間 [ミリ秒]	CFG_ARP_PRB_WAI	1000	1000	3000
ARP PROBE パケット送信回数	CFG_ARP_PRB_NUM	3	1	6
ARP PROBE パケット送信間隔(最小値) [ミリ秒]	CFG_ARP_PRB_MIN	1000	100	1000
ARP PROBE パケット送信間隔(最大値) [ミリ秒]	CFG_ARP_PRB_MAX	2000	200	2000
ARP ANNOUNCE パケット待ち時間 [ミリ秒]	CFG_ARP_ANC_WAI	2000	200	2000
ARP ANNOUNCE パケット送信回数	CFG_ARP_ANC_NUM	2	1	4
ARP ANNOUNCE パケット送信間隔 [ミリ秒]	CFG_ARP_ANC_INT	2000	200	2000

単位が[ミリ秒]の設定項目では、使用するタイマの精度によって近似値で反映されます。

※1 CFG_NET_SOC_MAX 以下である必要があります。また CFG_NET_SOC_MAX との差分が非 TCP ソケットの上限数になります。

※2 ネットワークバッファのサイズは、ネットワークバッファ管理構造体サイズ(44Byte)に、MTU、データリンクヘッダサイズ(Ethernet の場合は 14Byte)、ネットワークバッファデータ書き込み位置（デフォルト 2Byte）を合わせたサイズより大きい必要があります。

※3 TCP の送信バッファは使用するデバイスに関わらず、共通でグローバル変数 UB gTCP_SND_BUF[ ]を使用します。gTCP_SND_BUF[ ]は CFG_TCP_SND_WND×CFG_NET_TCP_MAX でこのサイズを決定します。もし TCP 送信バッファサイズの異なるデバイスを複数使用する場合は、その最大値に合わせて gTCP_SND_BUF[ ]を設定する必要がありあます。

※4 使用するデバイス単位に設定することが可能です。デバイス番号-1 をインデックスとして gNET_CFG[ ] に設定します。

※5 バッファサイズは 4 バイトから 32 キロバイトの範囲で、2 の累乗(1024, 2048, 4096 など)で指定します。

4.1.2 IP アドレス

IP アドレスを設定します。ネットワークデバイス毎に IP アドレスが必要になりますので、IP アドレスは CFG_NET_DEV_MAX 分登録してください。

IP アドレス:192.168.1.10、ゲートウェイ：192.168.1.1、サブネットマスク:255.255.255.0 の設定例は下記の通りとなります。

```
T_NET_ADR gNET_ADR[] = {
  {  /*   for Device 1   */
     0x0,              /*  必ず 0 を指定  */
     0x0,              /*  必ず 0 を指定  */
     0xC0A8000A,/* IP アドレス 192.168.1.10 を設定     */
     0xC0A80001,/*  ゲートウェイ 192.168.1.1       */
     0xFFFFFF00,/*  サブネットマスク 255.255.255.0 */
  }
};
```

4.1.3 デバイスドライバ

デバイスドライバを設定します。デバイスドライバは CFG_NET_DEV_MAX 分登録してください。

```
T_NET_DEV gNET_DEV[]   =   {
   { ..}
}
```

詳細は「3.2 ネットワークデバイスドライバ」を参照してください。

4.1.4 プロトコルスタック情報テーブル

次のようにプロトコルスタック広域変数を設定します。

```
const VP net_inftbl[] = {
0,                          /*  必ず 0 を指定  */
(VP)gNET_SOC,        /*  ソケットを使用しない場合は NULL を指定  */
(VP)gNET_TCP,        /*  ソケットを使用しない場合は NULL を指定  */
(VP)gNET_IPR,        /* IP 再構築機能を使用しない場合は NULL を指定  */
(VP)gNET_MGR,        /* IGMP を使用しない場合は NULL を指定  */
(VP)gTCP_SND_BUF,   /* TCP ソケットを使用しない場合は NULL を指定  */
};
```

4.1.5 ネットワーク情報管理リソース

TCP/IP スタックではネットワークの情報を管理するためにデバイスやソケットの数に応じた情報管理領域を以下のように広域変数で用意する必要があります。

T_NET_STS_DEV	net_cfg_sts_dev[CFG_NET_DEV_MAX]	デバイス情報
T_NET_STS_IFS	net_cfg_sts_ifs[CFG_NET_DEV_MAX]	TCP/IP 情報
T_NET_STS_IFS	net_cfg_sts_ifs_tmp[CFG_NET_DEV_MAX]	TCP/IP 情報(テンポラリ)
T_NET_STS_ARP	net_cfg_sts_arp[CFG_NET_ARP_MAX]	ARP 情報
T_NET_STS_SOC	net_cfg_sts_soc[CFG_NET_SOC_MAX]	ソケット情報
VP	net_cfg_sts_ptr[4 + CFG_NET_DEV_MAX]	ステータスポインタ
VP	net_cfg_sts_ptr_tmp[4 + CFG_NET_DEV_MAX]	ステータスポインタ

T_NET_STS_CFG	T_NET_STS_CFG gNET_STS_CFG = {
	net_cfg_sts_dev,
	net_cfg_sts_ifs,
	net_cfg_sts_ifs_tmp,
	net_cfg_sts_arp,
	net_cfg_sts_soc,
	net_cfg_sts_ptr,
	net_cfg_sts_ptr_tmp,
	0
	};
		ネットワーク情報管理テーブル

5. アプリケーションプログラミングインタフェースの説明

5.1 プロトコルスタックの初期化

TCP/IP プロトコルスタックを使用するにはプロトコルスタックの初期化とネットワークデバイスの初期化が必要になります。基本的には次のように初期化します。

初期化コード例）

```
/*  プロトコルスタックの初期化  */
        ercd     =     net_ini();
        if (ercd != E_OK) {
           return ercd;
        }
/*  ネットワークデバイス(デバイス番号 N)の初期化  */
        ercd     =     net_dev_ini(N);
        If (ercd != E_OK) {
           return ercd;
}
```

5.2 ネットワーク・インタフェース API

net_ini	TCP/IP プロトコルスタックの初期化

【書式】

ER ercd = net_ini(void);

【パラメータ】

なし

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

<0	初期化失敗

【解説】

プロトコルスタックで使用するリソースを初期化します。プロトコルスタックで使用するカーネルオブジェクト（タスク、メモリプール、セマフォ）も同時に生成初期化されます。また、プロトコルスタックで使用する広域変数には初期値がセットされます。

プロトコルスタックを使用する場合にはどの API よりも先に、この API を発行する必要があります。

net_cfg　　ネットワーク・インタフェースのパラメータ設定

【書式】

```
ER ercd = net_cfg(UH num, UH opt, VP val);
```

【パラメータ】

UH	num	デバイス番号
UH	opt	パラメータコード
VP	val	取得する値

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_NOSPT	不正なパラメータコード
E_ID	不正なデバイス番号
E_NOMEM	マルチキャストテーブルがいっぱい

【解説】

IP アドレスやサブネットマスク、ブロードキャストアドレス、マルチキャスト、その他基本的な設定を行います。

設定例
net_cfg(1, NET_BCAST_RCV, (VP)1);　/* ブロードキャストの受信を有効 */

パラメータコード	データタイプ	意味
NET_IP4_CFG	T_NET_ADR	IP アドレス、サブネットマスク、ゲートウェイを設定します。 val には T_NET_ADR のポインタを渡してください。
NET_IP4_TTL	UB	TTL（Time to Live）を設定します。 デフォルトは 64 が設定されています。
NET_BCAST_RCV	UB	ブロードキャストの受信の可否を設定します。“1”を設定した場合は受信可能となり、“0”を設定した場合は不可となります。
NET_MCAST_JOIN	UW	参加するマルチキャストグループの IP アドレスを登録します。
NET_MCAST_DROP	UW	脱退するマルチキャストグループの IP アドレスを設定します。
NET_MCAST_TTL	UB	マルチキャスト送信で使用する TTL を設定します。
NET_ACD_CBK	コールバック関数ポインタ	運用中に IP アドレス競合を検出したことをコールバック通知する関数を設定します。 この設定により競合検出の通知機能が有効になります。

net_ref　ネットワーク・インタフェースのパラメータ参照

【書式】

```
ER ercd = net_ref(UH num, UH opt, VP val);
```

【パラメータ】

UH	num	デバイス番号
UH	opt	パラメータコード
VP	val	取得する値のバッファへのポインタ

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_NOSPT	不正なパラメータコード
E_ID	不正なデバイス番号

【解説】

IP アドレスやサブネットマスク、ブロードキャストアドレス、そのほかの基本的な設定の確認を行います。

設定例

```
UB bcast;
net_ref(1, NET_BCAST_RCV, (VP)&bcast);   /* ブロードキャストの受信状態 */
```

パラメータコード	データタイプ	意味
NET_IP4_CFG	T_NET_ADR	IP アドレス、サブネットマスク、ゲートウェイを取得します。 val には T_NET_ADR のポインタを渡してください。
NET_IP4_TTL	UB	TTL（Time to Live）を取得します。
NET_BCAST_RCV	UB	ブロードキャストの受信の状態を取得します。
NET_MCAST_TTL	UB	マルチキャスト送信 TTL を取得します。

net_acd	IPアドレスの競合探知

【書式】

```
ER ercd = net_acd(UH dev_num, T_NET_ACD *acd);
```

【パラメータ】

UH	num	デバイス番号
T_NET_ACD	*acd	アドレス競合情報

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正なデバイス番号
E_PAR	パラメータ不正
E_OBJ	重複呼び出し、ホストIP未設定時の呼び出し
E_TMOUT	ARP送信タイムアウト
E_SYS	IPアドレス競合検出
E_OK	IPアドレス競合非検出

【解説】

dev_num で指定されるデバイスの、IP アドレスの競合検出を行います。

IP アドレスの競合を検出した場合、引数の競合情報には相手側の MAC アドレスが格納されます。

この API とは別に非同期で IP アドレスの競合を検出したい場合は、net_cfg()API でコールバック関数を登録する必要があります。

※本関数は最大で約 10 秒、競合アドレスの検出を試みるため専用タスクで呼び出すことをお勧めします。

acd_cbk	IPアドレスの競合検出時のコールバック関数

【書式】

```
ER acd_cbk(T_NET_ACD* acd);
```

【パラメータ】

T_NET_ACD	*acd	アドレス競合情報

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【解説】

この関数は運用中に IP アドレスの競合を検出した場合に呼び出されます。引数の競合情報には、競合したホストの MAC アドレスが格納されます。

IP アドレスの競合に対して、自身のホストでその IP アドレスを使用し続ける場合は E_OK を返却して下さい。それ以外の場合は E_SYS を返却して下さい。

コールバック関数は ARP パケットを受信したタスク（Ethernet ドライバの受信タスク）上で呼ばれます。そのためコールバック関数は即座に終了して下さい。また IP アドレス探知中（net_acd()実行中）はコールバック関数が呼ばれることはありません。

使用例

```
#define    ID_DEVNUM_ETHER   1    /* デバイス番号 */
/* アドレス競合検出時のコールバック関数 */
ER acd_detect(T_NET_ACD * acd)
{
    return E_OK;
}

/* ネットワーク初期化関数 */
ER net_setup(void)
{
    ER ercd;
    T_NET_ACD acd;

    ercd = net_ini();
    if (ercd != E_OK) {
        return ercd;
    }
    ercd = net_dev_ini(ID_DEVNUM_ETHER);
    if (ercd != E_OK) {
        return ercd;
    }

    /* IP アドレス競合探知 */
    ercd = net_acd(ID_DEVNUM_ETHER, &acd);
    if (ercd == E_OK) {
        /* IP アドレスの競合無し */
        /* IP アドレス競合検出時のコールバック関数設定 */
        net_cfg(ID_DEVNUM_ETHER, NET_ACD_CBK, (VP)acd_detect);
    }
    else if (ercd == E_SYS) {
        /* MAC アドレスが acd.mac のホストと IP が競合 */
    } else {
        /* IP アドレスの競合探知に失敗 */
    }
    return ercd;
}
```

5.3 ネットワークデバイス制御 API

ネットワークデバイス制御 API はアプリケーションからデバイスドライバに統一的にアクセスするためのインタフェースを提供します。各デバイスには、本 API に'デバイス番号'を指定してアクセスします。デバイス番号とは、デバイスを識別するための固有の番号です。

net_dev_ini	ネットワークデバイスの初期化

【書式】

```
ER ercd = net_dev_ini(UH dev_num);
```

【パラメータ】

UH	dev_num	デバイス番号

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

<0	初期化失敗

【解説】

dev_num を使って特定のデバイスを初期化します。net_dev_ini は、実際にはデバイスドライバの dev_ini を使ってデバイスを初期化します。

正常終了すると、そのネットワークデバイスを通じてパケットの処理が可能となります。

net_dev_cls	ネットワークデバイスの解放

【書式】

ER ercd = net_dev_cls(UH dev_num);

【パラメータ】

UH	dev_num	デバイス番号

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

<0	解放失敗

【解説】

dev_num を使って特定のデバイスを解放します。net_dev_cls は、実際にはデバイスドライバの dev_cls を使ってデバイスを解放します。

net_dev_ctl	ネットワークデバイスの制御

【書式】

ER ercd = net_dev_ctl(UH dev_num, UH opt, VP val);

【パラメータ】

UH	dev_num	デバイス番号
UH	opt	制御コード
VP	val	設定する値

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

<0	解放失敗

【解説】

dev_num を使って特定のデバイスを制御します。net_dev_ctl はデバイスドライバの dev_ctl を呼び出しているだけですので、実際の動作はデバイスドライバの実装に依存します。

net_dev_sts	ネットワークデバイスの状態取得

【書式】

ER ercd = net_dev_sts(UH dev_num, UH opt, VP val);

【パラメータ】

UH	dev_num	デバイス番号
UH	opt	状態コード
VP	val	取得する値

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

<0	解放失敗

【解説】

dev_num を使って特定のデバイスの状態を取得します。net_dev_sts はデバイスドライバの dev_ref を呼び出しているだけですので、具体的な動作はデバイスドライバの実装に依存します。

5.4 ソケット API (uNet3 互換)

アプリケーションは、ソケット API を使用してリモートホストとの TCP/UDP データのやり取りを行います。

ソケットは生成もしくは接続時にデバイス番号を使って、接続するネットワークデバイスを指定する必要があります。デバイス番号に 0 を指定した場合は「デバイスを特定しない」という意味を持ち、送信と受信でソケット／ネットワークデバイス間のインタフェース選択動作が異なります。またソケット生成時に 0 以外のデバイス番号を指定した場合には、接続時にデバイス番号を指定する必要はありません。

例として N 個のネットワークデバイス（N は 2 以上）で構成されたシステム上で、ソケット API を使用した場合、以下の表の通りデバイスを使用します。

	生成時のデバイス番号（※1）	接続時のデバイス番号（※2）	使用するデバイス
ソケット送信動作 snd_soc()や TCP クライアントの con_soc()（SYN 送信）	0	0	デバイス番号 1（先頭）
	0	N	デバイス番号 N
	N	ANY	デバイス番号 N
ソケット受信動作 rcv_soc()や TCP サーバーの con_soc()（SYN 受信）	0	0	通知したデバイス（※3）
	0	N	デバイス番号 N
	N	ANY	デバイス番号 N

※1　con_soc() API の引数 host->num で指定します。

※2　con_soc() API の引数 host->num で指定します。UDP ソケットで受信する場合は con_soc() API を呼び出す必要はありません。

※3　ソケット生成時も接続時にもデバイス番号を指定していないソケットは、ポート番号とプロトコルが一致すればどのデバイスからでもパケットを受信することが可能です。この場合ソケットはパケットを通知したデバイスを以降の動作で使用します。

cre_soc　ソケットの生成

【書式】

ER ercd = cre_soc(UB proto, T_NODE *host);

【パラメータ】

UH	proto	プロトコル種別
T_NODE	*host	ローカルホスト情報

【戻り値】

ER	ercd	生成されたソケットのID（>0）またはエラーコード

【エラーコード】

E_ID	ソケットを作ることができない（ソケット最大数を超えている）
E_PAR	‘host’が不正
E_NOSPT	‘proto’が不正

【T_NODE】

使用するローカルポート番号とデバイスインタフェースを指定します。

UH	port	ポート番号	ローカルホストのポート番号。1 から65535の値またはPORT_ANYを指定する。PORT_ANYが指定された場合、ポート番号はプロトコルスタックで決定する。
UH	ver	IPバージョン	0を指定（IP_VER4を使う）
UB	num	デバイス番号	使用したいデバイスのデバイス番号を指定
UW	ipa	IPアドレス	0を指定（ローカルIPアドレスを使う）

【proto】

生成するソケットのプロトコル種別

IP_PROTO_TCP	TCPソケット
IP_PROTO_UDP	UDPソケット

【解説】

この API は指定したプロトコルのソケットを作ります。

TCP ソケット生成例

```
T_NODE　host;
host.num = 1;
host.port = 7;
host.ver = IP_VER4;
host.ipa = INADDR_ANY;
cre_soc(IP_PROTO_TCP, &host);
```

del_soc	ソケットの削除

【書式】

```
ER ercd = del_soc(SID sid);
```

【パラメータ】

SID	sid	ソケットを識別するID

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正なID番号
E_NOEXS	ソケットが存在しない（ソケットが作られていない）
E_OBJ	ソケットの状態が不正

【解説】

この API は指定した ID のソケットを削除します。TCP ソケットを削除する時、事前に cls_soc()を呼び出してソケットを閉じてください。

con_soc　ソケットの接続

【書式】

```
ER ercd = con_soc(SID sid, T_NODE *host, UB con_flg) ;
```

【パラメータ】

SID	sid	ソケットを識別するID
T_NODE	*host	リモートホスト情報
UB	con_fig	接続モード

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正なID番号
E_NOEXS	ソケットが存在しない（ソケットが作られていない）
E_PAR	hostまたはcon_flgが不正
E_OBJ	ソケットの状態が不正（既に接続済みのソケットに対してこのAPIを呼んだ時など）
E_TMOUT	接続処理がタイムアウトした
E_WBLK	ノンブロッキングモードで処理
E_CLS	リモートホストから接続拒否
E_RLWAI	接続処理が中止された
E_QOVR	既にcon_soc()実行中

【T_NODE】

リモートホストと使用するデバイスインタフェースを指定します。

UH	port	ポート番号	リモートホストのポート番号 （1 から65535）
UH	ver	IPバージョン	0を指定
UB	num	デバイス番号	使用したいデバイスのデバイス番号
UW	ipa	IPアドレス	リモートホストのIPアドレス

【con_fig】

接続を待ち受ける(サーバー)、能動的（クライアント）に接続するかを指定します。

UDPソケットの場合は常に0を指定してください。

SOC_CLI	リモートホストに接続する（能動接続）
SOC_SER	接続を待ち受ける（受動接続）

【解説】

この API は使用するプロトコルによって振る舞いが異なります。

TCP の時は、リモートホストとの接続の確立を行い、UDP の時は、データ送信先とソケットとの関連付けを行います。

TCP サーバーソケットの接続例
T_NODE　remote = {0};　　　　　　/* 0 でクリア */
con_soc(SID, &remote, SOC_SER);

TCP クライアントソケットの接続例
T_NODE　remote;
remote.port = 100;　　　　　　　/* リモートホストのポート番号 */
remote.ver = IP_VER4;
remote.num = 1;　　　　　　　　/* 使用するデバイス番号を指定 */
remote.ipa = ip_aton(“192.168.11.1”); /*　リモートホストの IP アドレス */
con_soc(SID, &remote, SOC_CLI);

cls_soc　　ソケットの切断

【書式】

```
ER ercd = cls_soc(SID sid, UB cls_flg);
```

【パラメータ】

SID	sid	ソケットを識別するID
UB	cls_fig	切断モード

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正なID番号
E_NOEXS	ソケットが存在しない（ソケットが作られていない）
E_PAR	cls_flgが不正
E_OBJ	ソケットの状態が不正（未接続状態でこのAPIを呼んだ時など）
E_TMOUT	切断処理がタイムアウトした
E_WBLK	ノンブロッキングモードで処理
E_CLS	リモートホストから接続の強制終了
E_RLWAI	接続処理が中止された
E_QOVR	既にcls_soc()実行中

【cls_flg】

このパラメータはTCPソケットのみに有効です。

SOC_TCP_CLS	ソケットを切断する。（接続を終了する）
SOC_TCP_SHT	送信処理のみを無効にする。受信は可能。（SOC_TCP_SHTを使用して、送信処理をのみを停止した後、完全に接続を終了したい場合、SOC_TCP_CLSを使用して完全に接続を終了する必要があります）

【解説】

この API は使用するプロトコルによって振る舞いが異なります。

TCP の時は、リモートホストとの接続を切断し、UDP の時は、ソケットに関連付けられたデータの送受信先の情報をクリアします。（この後、UDP データの送信を行うことはできません。）

cfg_soc　　ソケットのパラメータ設定

【書式】

```
ER ercd = cfg_soc(SID sid, UB code, VP val) ;
```

【パラメータ】

SID	sid	ソケットを識別するID
UB	code	パラメータコード
VP	val	設定する値

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正なID番号
E_NOEXS	ソケットが存在しない（ソケットが作られていない）
E_NOSPT	不正なパラメータコード
E_PAR	不正なパラメータ値
E_OBJ	ソケットの状態が不正

【解説】

次に示すパラメータの設定が可能です。設定する値は VP 型へキャストして渡してください。

設定例

```
UB ttl = 32;
cfg_soc(SID, SOC_IP_TTL, (VP)ttl);
```

パラメータコード	データタイプ	意味
SOC_TMO_CON	TMO	con_soc の呼出しタイムアウト
SOC_TMO_CLS	TMO	cls_soc の呼出しタイムアウト
SOC_TMO_SND	TMO	snd_soc の呼出しタイムアウト
SOC_TMO_RCV	TMO	rcv_soc 呼出しタイムアウト
SOC_IP_TTL	UB	IP ヘッダの TTL（Time to Live）を設定
SOC_IP_TOS	UB	IP ヘッダの TOS（Type of Server）を設定
SOC_CBK_HND	関数へのポインタ	コールバック関数の登録
SOC_CBK_FLG	UH	コールバックイベントフラグのビットパターンを設定 （設定する値は、以下を参照）
SOC_PRT_LOCAL	UH	ローカルポート番号の変更
SOC_UDP_RQSZ	UB	UDP ソケットの受信キューサイズの変更(*)

(*) UDP ソケットの受信キューサイズは通常 1 です。この値を変更した場合、すでに受信中のパケットを失うことがあります。受信キューサイズに 0 を設定するとパケットを受信しません。

コールバックイベントフラグビット	意味
EV_SOC_CON	con_soc()をノンブロッキングモードに設定 （TCP ソケットのみ）
EV_SOC_CLS	cls_soc()をノンブロッキングモードに設定 （TCP ソケットのみ）
EV_SOC_SND	snd_soc()をノンブロッキングモードに設定
EV_SOC_RCV	rcv_soc()をノンブロッキングモードに設定

コールバックイベントフラグビットについては、複数ビットの設定が可能です。複数設定する場合 OR で設定してください。以下に設定例を示します。

例) ercd = cfg_soc(ソケット ID, SOC_CBK_FLG, (VP)(EV_SOC_CON|EV_SOC_SND|EV_SOC_RCV|EV_SOC_CLS));

ノンブロッキングに設定したソケットイベントはそのイベントのソケットタイムアウトが無効になります。

コールバックイベントフラグビットを有効にするときは、SOC_CBK_HND でコールバック関数の登録が必要となります。コールバック関数については、次項を参照してください。

soc_cbt	コールバック関数

【書式】

```
UW soc_cbt(SID sid, UH event, ER ercd);
```

【パラメータ】

SID	sid	ソケットを識別するID
UH	event	コールバックイベントフラグビット
ER	ercd	エラーコード

このコールバック関数は、TCP/IP スタックから呼び出されます。なお、ノンブロッキングモードのソケット API を実行した場合、API 処理が待ち状態になる必要がある時、待ち状態とはならずに、E_WBLK の値が返ります。このとき、TCP/IP スタックからは、コールバック関数で処理が終わったことを通知します。

コールバック イベントフラグビット (event)	エラーコード (ercd)	意味
EV_SOC_CON	E_OK	con_soc()処理が正常終了
	< 0	con_soc()処理がエラーで終了。この時のエラー内容については con_soc()のエラーコードを参照してください。
EV_SOC_CLS	E_OK	cls_soc()処理が正常終了
	< 0	cls_soc()処理がエラーで終了。この時のエラー内容ついては cls_soc()のエラーコードを参照してください。
EV_SOC_SND	> 0	UDP ソケット： snd_soc()処理が正常終了 TCP ソケット： TCP 送信バッファに空きがある場合、空きのサイズを'ercd'値で表す。再び snd_soc()を呼び出して、送信データを TCP 送信バッファにコピーすることが出来る。
	<= 0	snd_soc()処理がエラーで終了。この時のエラー内容については snd_soc()のエラーコードを参照してください。
EV_SOC_RCV	> 0	UDP ソケット： UDP ソケットには受信データが存在する。受信データのサイズを'ercd'値で表す。再び rcv_soc()を呼び出してデータを受信することが出来る。 TCP ソケット： TCP ソケットには受信データが存在する。受信データのサイズを'ercd'値で表す。再び rcv_soc()読んでデータを受信することが出来る。
	<= 0	rcv_soc()処理がエラーで終了。この時のエラー内容については rcv_soc()のエラーコードを参照してください。

※コールバック関数から TCP/IP スタックの全ての API・関数を呼び出すことはできません。（コールバック関数は割り込みハンドラと同じように考えて使用してください）

ref_soc	ソケットのパラメータ参照

【書式】

```
ER ercd = ref_soc(SID sid, UB code, VP val) ;
```

【パラメータ】

SID	sid	ソケットを識別するID
UB	code	パラメータコード
VP	val	取得する値のバッファへのポインタ

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正なID番号
E_NOEXS	ソケットが存在しない（ソケットが作られていない）
E_NOSPT	不正なパラメータコード
E_PAR	不正なパラメータ値（valがNULLの場合）
E_OBJ	ソケットの状態が不正（ソケットを参照することができない）

【解説】

次に示すパラメータの参照が可能です。取得する値は VP 型へキャストして渡してください。

リモートホスト情報取得例
T_NODE remote; ref_soc(SID, SOC_IP_REMOTE, (VP)&remote);

パラメータコード	データタイプ	意味
SOC_TMO_CON	TMO	con_soc の呼出しタイムアウト
SOC_TMO_CLS	TMO	cls_soc の呼出しタイムアウト
SOC_TMO_SND	TMO	snd_soc の呼出しタイムアウト
SOC_TMO_RCV	TMO	rcv_soc 呼出しタイムアウト
SOC_IP_LOCAL	T_NODE	ローカルホストの IP アドレスとポート番号を取得
SOC_IP_REMOTE	T_NODE	リモートホストの IP アドレスとポート番号を取得
SOC_IP_TTL	UB	TTL（Time to Live）を取得
SOC_IP_TOS	UB	TOS(Type Of Service)を取得
SOC_RCV_PKT_INF	T_RCV_PKT_INF	UDP ソケットが受信した最新のパケット情報を取得。 TCP ソケットの場合は接続成立時の接続先情報を取得。
SOC_PRT_LOCAL	UH	ローカルポート番号の参照

マルチキャストアドレスとユニキャストアドレスを持つソケットで、直前のパケットを受信した IP アドレスを知るには次のように参照して下さい。

受信 IP アドレス取得例

```
T_RCV_PKT_INF rcv_pkt_inf;
ref_soc(SID, SOC_RCV_PKT_INF, (VP)&rcv_pkt_inf);
if (rcv_pkt_inf.dst_ipa == MULTICASTADDRESS) {
    /* マルチキャストアドレスで受信 */
}
```

TCP ソケットの場合は src_ipa と src_port のメンバ変数に接続先情報が設定され、dst_ipa と dst_port のメンバ変数に自ノードの IP アドレスおよびポート番号の情報が設定されます。

これはサーバ接続、クライアント接続に関係なくリモートホストの情報として TCP の接続が確立した時点のものが反映されます。

abt_soc　　ソケット処理の中止

【書式】

ER ercd = abt_soc(SID sid, UB code);

【パラメータ】

SID	sid	ソケットを識別するID
UB	code	制御コード

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正なID番号
E_NOEXS	ソケットが存在しない（ソケットが作られていない）

【解説】

この API は、con_soc、cls_soc、snd_soc、rcv_soc の待ち状態をキャンセルすることができます。 キャンセルされた API は、E_RLWAI を返します。

制御コード	意味
SOC_ABT_CON	con_soc()処理の中止
SOC_ABT_CLS	cls_soc()処理の中止
SOC_ABT_SND	snd_soc()処理の中止
SOC_ABT_RCV	rcv_soc()処理の中止
SOC_ABT_ALL	すべてのソケットの処理の中止

snd_soc	データの送信

【書式】

```
ER ercd = snd_soc(SID sid, VP data, UH len);
```

【パラメータ】

SID	sid	ソケットを識別するID
VP	data	送信データのポインタ
UH	len	送信データサイズ

【戻り値】

ER	ercd	実際に送信されたデータサイズ（>0）またはエラーコード

【エラーコード】

E_ID	不正なID番号
E_NOEXS	ソケットが存在しない（ソケットが作られていない）
E_PAR	不正な送信データか、送信データサイズが指定されていない
E_OBJ	ソケットの状態が不正
E_TMOUT	送信処理がタイムアウト
E_WBLK	ノンブロッキンモードで処理
E_CLS	TCPソケットが切断された
E_RLWAI	送信処理が中止された
E_NOMEM	メモリ不足
E_QOVR	既にsnd_soc()実行中
EV_ADDR	送信先デフォルトG/Wが不明

【解説】

この API はリモートホストにデータを送信します。処理が成功した時は、実際に送信されたデータサイズを返します。それ以外の時はエラーコードを返します。

TCP ソケットの場合、この API はデータをプロトコルスタック内部にコピーし、そのコピーしたサイズを返します。（返されるデータサイズは引数で指定された len 以下です）。詳細は、「3.1.4　TCP モジュール」を参照してください。

UDP ソケットの場合、データはネットワークに送信されその送信サイズを返します。詳細は、「3.1.3　UDP モジュール」を参照してください。

rcv_soc	データの受信

【書式】

ER ercd = rcv_soc(SID sid, VP data, UH len);

【パラメータ】

SID	sid	ソケットを識別するID
VP	data	受信データへのポインタ
UH	len	受信データサイズ

【戻り値】

ER	ercd	実際に受信したデータサイズ（>0）またはエラーコード

【エラーコード】

E_ID	不正なID番号
E_NOEXS	ソケットが存在しない（ソケットが作られていない）
E_PAR	不正な受信データか、受信データサイズが指定されていない
E_OBJ	ソケットの状態が不正
E_TMOUT	受信処理がタイムアウト
E_WBLK	ノンブロッキンモードで処理
E_CLS	TCPソケットが切断された
E_RLWAI	受信処理が中止された
E_QOVR	既にrcv_soc()実行中
0	TCPソケットのデータ終端まで受信した

【解説】

この API はリモートホストから送信されたデータを受信します。

TCP の場合、受信可能な最大サイズはコンフィグレーションファイルで指定した“受信バッファサイズ”です。詳細は、「3.1.4　TCP モジュール」を参照してください。

UDP の場合、受信可能な最大サイズは 1472 bytes（デフォルト MTU – IP ヘッダーサイズ – UDP ヘッダーサイズ)になります。詳細は、「3.1.3　UDP モジュール」を参照してください。

soc_ext	ソケット処理の一斉停止

【書式】

ER ercd = soc_ext(UH dev_num);

【パラメータ】

UH	dev_num	対象のデバイスID

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	無効なデバイスID

【解説】

API 実行中や通信中のソケットに対してその動作を停止しソケットを初期状態に戻します。

引数 dev_num には停止対象となるネットワークデバイスを指定しますが、DEV_ANY を指定した場合は、すべてのソケットが対象となります。

本 API を実行すると実行中のソケット API は E_RLWAI を返却します。

TCP ソケットの場合、セッション中ならそのセッションはリセットされます。また TCP、UDP に限らずソケットが保持している送受信パケットは即座に解放されます。

本 API 終了後ソケットは初期状態(ソケット生成後の状態)に戻りますがコールバック、ノンブロッキング、タイムアウトなどのアプリケーションが設定している各種ソケットオプションは保たれます。

5.5 ソケット API (BSD 互換)

TCP/IP スタックは、BSD インタフェースのソケット API も提供しています。これにより、Linux 等の BSD ソケット API を使用した、既存のネットワークアプリケーションを容易に流用することが可能です。

5.5.1 モジュール概要

5.5.1.1 POSIX 仕様での位置づけ

TCP/IP スタックは、4.4BSD-Lite 相当のソケット API を提供します。サポートしている API 一覧は「5.5.3 API 一覧」を参照して下さい。

5.5.1.2 μNet3 ソケット API との違い

POSIX 準拠のソケット API に加え、μNet3 ソケットでは兼ね備えていなかった機能も提供します。

- ソケット API の多重呼出し
- select()関数
- ループバックアドレス
- ソケット単位のマルチキャストグループ
- TCP ソケットの Listen キュー
- ソケットエラー

5.5.1.3 シンボル名の互換性に関して

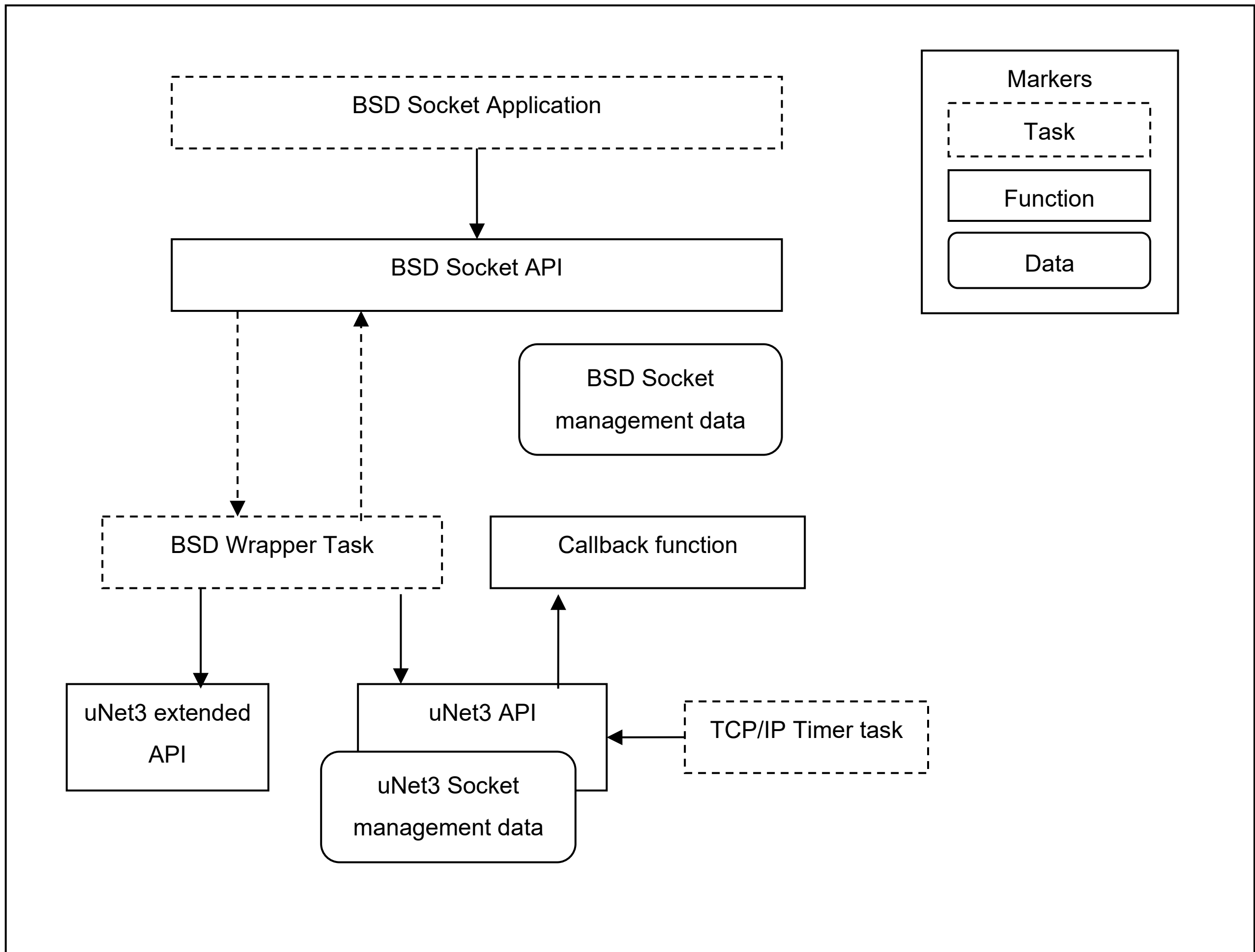
コンパイラ環境によるシンボル衝突を避けるため、API、構造体、マクロには独自接頭辞”unet3_”が冠してあります。

sys/socket.h をインクルードすることにより、アプリケーション内で使用している POSIX 標準のシンボル名は、これら独自接頭辞付きのシンボルへと置き換わります。したがって、BSD ソケットを使用するアプリケーションは、そのままのソースファイルで動作させることが可能です。

可読性を考慮し、本書での表記は POSIX 標準シンボルを用いています。

5.5.2 モジュール構成

図 5.1 BSD ソケットのモジュール構成



5.5.3 API 一覧

表 5.1 API 一覧

API	機能	インクルードヘッダ
unet3_bsd_init	TCP/IP スタック/BSD を初期化する	“sys/socket.h”
get_errno	タスク毎の errno を取得する	“sys/errno.h”
socket	通信のための端点(endpoint)を作成する	“sys/socket.h”
bind	ソケットに名前をつける	“sys/socket.h”
listen	ソケット(socket)上の接続を待つ	“sys/socket.h”
accept	ソケットへの接続を受ける	“sys/socket.h”
connect	ソケットの接続を行う	“sys/socket.h”
send	ソケットへメッセージを送る	“sys/socket.h”
sendto	ソケットへメッセージを送る	“sys/socket.h”
recv	ソケットからメッセージを受け取る	“sys/socket.h”
recvfrom	ソケットからメッセージを受け取る	“sys/socket.h”
shutdown	全二重接続の一部を閉じる	“sys/socket.h”
close	ディスクリプタ(ソケット)をクローズする	“sys/ unistd.h”
select	同期 I/O の多重化	“sys/ select.h”
getsockname	ソケットの名前を取得する	“sys/socket.h”
getpeername	接続している相手ソケットの名前を取得する	“sys/socket.h”
getsockopt	ソケットのオプションの取得を行なう	“sys/socket.h”
setsockopt	ソケットのオプションの設定を行なう	“sys/socket.h”
ioctl	デバイス(ソケット)を制御する	“sys/ioctl.h”
inet_addr	インターネットアドレス操作ルーチン	“arpa/inet.h”
inet_aton	インターネットアドレス操作ルーチン	“arpa/inet.h”
inet_ntoa	インターネットアドレス操作ルーチン	“arpa/inet.h”
if_nametoindex	ネットワーク・インタフェースの名前とインデックスのマッピングを行う	“net/if.h”
if_indextoname	ネットワーク・インタフェースの名前とインデックスのマッピングを行う	“net/if.h”
rresvport	ポートにバインドされたソケットを取得	“sys/unistd.h”
getifaddrs	インタフェースのアドレス取得	“sys/types.h”
freeifaddrs	インタフェース情報の解放	“sys/types.h”
inet_pton	IPv4/IPv6 アドレスをテキスト形式からバイナリ形式に変換	“arpa/inet.h”
inet_ntop	IPv4/IPv6 アドレスをバイナリ形式からテキスト形式に変換	“arpa/inet.h”

5.5.4 API 詳細

5.5.4.1 unet3_bsd_init

「5.5.7.7 初期化」を参照してください。

5.5.4.2 get_errno

「5.5.6.3 エラー処理」を参照してください。

5.5.4.3 socket (通信のための端点を作成する)

【書式】

```
#include "sys/socket.h"
int socket(int domain, int type, int protocol);
```

【パラメータ】

int	domain	ドメイン
int	type	通信方式
int	protocol	プロトコル

【戻り値】

int	生成されたソケットFD。エラーの場合は-1

【errno】

ENOMEM	生成可能なソケット数を超えている
	メッセージバッファが枯渇している
EINVAL	パラメータ不正
EINTR	待ち状態が強制解除された

- ドメインには AF_INET、AF_INET6 のみ指定可能です。
- 通信方式は SOCK_STREAM, SOCK_DGRAM のみ指定可能です。
- プロトコルは使用しないため値は不問です。
- 同時に生成可能なソケット数(TCP と UDP の合計)は#define CFG_NET_SOC_MAX の値になります。
- 同時に生成可能な TCP ソケット数は#define CFG_NET_TCP_MAX の値になります。
- ソケットのローカルポートに 0 を設定することはできません。
  従って生成直後のソケットは一時的な値のローカルポート番号が割り当てられます。

5.5.4.4 bind (ソケットに名前をつける)

【書式】

```
#include “sys/socket.h”
int bind(int sockfd, const struct sockaddr *addr, unsigned int addrlen);
```

【パラメータ】

int	sockfd	ソケットFD
const struct sockaddr *	addr	ローカルアドレス
unsigned int	addrlen	ローカルアドレス長

【戻り値】

int	処理の成否。成功の場合は0、エラーの場合は-1

【errno】

EINVAL	パラメータ不正
ENOMEM	メッセージバッファが枯渇している
EBADF	bindできないソケットFD
EPIPE	ソケットFDが不正な状態
EAFNOSUPPORT	非サポートのアドレスファミリ
EADDRINUSE	既に使用されているアドレス
EADDRNOTAVAIL	使用できないアドレス
EINTR	待ち状態が強制解除された

- 受信動作を開始するには、対象のソケットを指定して bind()を予め実行する必要があります。受信動作とは TCP の待ち受け接続(listen())や、UDP パケットの受信(recv()、recvfrom())を言います。
- wellknown ポート(1-1023)への bind()も可能です。

【アドレスファミリが AF_INET の場合】

- ローカルアドレスは struct sockaddr_in 型で設定します。
- ローカルアドレスの IP アドレス(IPv4)には、デバイスに設定されているアドレス、もしくは INADDR_ANY(不特定)のみ指定可能です。
- ローカルアドレスのポート番号に PORT_ANY(0)を設定した場合は、プロトコルスタックでポート番号を割り当てます。
- ローカルアドレス長は sizeof(struct sockaddr_in) (=16)のみ指定可能です。
- struct sockaddr_in 型のメンバ「sin_len」は使用しないため値は不問です。

【アドレスファミリが AF_INET6 の場合】

- ローカルアドレスは struct sockaddr_in6 型で設定します。
- ローカルアドレスの IP アドレス(IPv6)には、デバイスに設定されているアドレス、もしくは IN6ADDR_ANY_INIT(不特定)のみ指定可能です。
- ローカルアドレスの IP アドレス(IPv6)にリンクローカルアドレスを指定した場合、スコープ ID には、0 以外のネットワークインタフェース番号を指定してください。
- ローカルアドレスのポート番号に PORT_ANY(0)を設定した場合は、プロトコルスタックでポート番号を割り当てます。
- ローカルアドレス長は sizeof(struct sockaddr_in6) (=28)のみ指定可能です。
- struct sockaddr_in6 型のメンバ「sin6_len」は使用しないため値は不問です。
- POSIX とは異なり bind で特定したデバイスが複数の IPv6 アドレスを持つ場合、ソケット通信時のアドレススコープによっては bind で指定したアドレスが選択されない場合があります。

5.5.4.5 listen (ソケット上の接続を待つ)

【書式】

```
#include "sys/socket.h"
int listen(int sockfd, int backlog);
```

【パラメータ】

int	sockfd	ソケットFD
int	backlog	バックログ

【戻り値】

int	処理の成否。成功の場合は0、エラーの場合は-1

【errno】

EINVAL	パラメータ不正、接続済みのTCPソケットFD
ENOMEM	メッセージバッファが枯渇している
EBADF	listenできないソケットFD
EPROTONOSUPPORT	サポートしていなプロトコル(TCPではない)
EINTR	待ち状態が強制解除された

- TCP ソケットを接続待ち状態にします。
- ソケット FD には TCP のソケット FD のみ指定可能です。
- バックログに指定できる最大数は、#define CFG_NET_TCP_MAX －1 です。

5.5.4.6 accept (ソケットへの接続を受ける)

【書式】

```
#include "sys/socket.h"
int accept(int sockfd, struct sockaddr *addr, unsigned int *addrlen);
```

【パラメータ】

int	sockfd	ソケットFD
struct sockaddr *	addr	リモートアドレス(出力)
unsigned int *	addrlen	リモートアドレス長(入出力)

【戻り値】

int	接続されたソケットFD。エラーの場合は-1

【errno】

EINVAL	パラメータ不正
ENOMEM	メッセージバッファが枯渇している
EBADF	listenされていないソケット
EAGAIN	接続されていない(非同期実行時)
ETIMEDOUT	タイムアウト(タイムアウト設定時)
EINTR	待ち状態が強制解除された

- ソケット FD には listen()が成功した TCP のソケット FD のみ指定可能です。
- リモートアドレスは、アドレスファミリが AF_INET の場合は struct sockaddr_in*型、AF_INET6 の場合は struct sockaddr_in6*型で格納します。
- 指定されたリモートアドレス長が小さ過ぎた場合、addr には指定されたアドレス長分のデータのみコピーし、addrlen にはアドレスファミリが AF_INET の場合は sockaddr_in 型のサイズ(16 バイト)、AF_INET6 の場合は sockaddr_in6 型のサイズ(28 バイト)を格納します。
- 接続されたソケットが無かった場合には、accept()はリモートから接続されるまで処理をブロックします。
- TCP ソケットに対する errno については 5.5.7.6 TCP ソケット errno ユースケースを参照して下さい。

5.5.4.7 connect (ソケットの接続を行う)

【書式】

```
#include “sys/socket.h”
int connect(int sockfd, const struct sockaddr *addr, unsigned int addrlen);
```

【パラメータ】

int	sockfd	ソケットFD
const struct sockaddr *	addr	リモートアドレス
unsigned int	addrlen	リモートアドレス長

【戻り値】

int	処理の成否。成功の場合は0、エラーの場合は-1

【errno】

EINVAL	パラメータ不正
ENOMEM	メッセージバッファが枯渇している
EBADF	connectできないソケット
EHOSTUNREACH	アクセス不可能なノードに対して接続しようとした
ECONNREFUSED	接続が拒否された
EAFNOSUPPORT	非サポートのアドレスファミリ
EISCONN	既に接続済み
	listenしているソケット
EALREADY	既に接続中
EAGAIN	接続中(非同期実行時)
ETIMEDOUT	タイムアウト(タイムアウト設定時)
EINTR	待ち状態が強制解除された

- connect()はソケット FD のプロトコルや、送受信動作によって作用する動作や振る舞いが異なります。
- TCP ソケットの接続動作の場合、リモートアドレスに対して SYN を送信して接続を試みます。これは接続中や接続待ち以外の TCP ソケットにのみ動作します。
- UDP ソケットの送信動作の場合、リモートアドレスは宛先アドレスとして設定されます。ただし sendto()でこれと異なる宛先アドレスが指定された場合には sendto()の宛先アドレスが使用されます。
- リモートアドレスのアドレスファミリ(sa_family)に AF_UNSPEC を指定すると、これらの設定はクリアされます。
- POSIX 仕様とは異なり、UDP ソケットの受信動作に作用するリモートアドレスによるフィルタリング機能はありません。
- POSIX 仕様とは異なり、非同期設定した TCP ソケットに対する connect()では、接続完了後に再度 connect を発行することはできません。例えば connect()に対する EAGAIN を確認後、select()で書き込み可能を保証された場合、その時点で TCP のセッションは確立しているため、データの送受信が可能になります。
- 非同期設定した TCP ソケットに対する connect()を発行して、その後 connect()処理が終了した場合、次に connect()を発行した際の挙動として接続成功時は戻り値=0、errno は前回設定されたエラー番号となります。逆に接続失敗時は戻り値=-1、errno はエラー要因のエラー番号となります。また、これらの結果を返した connect()処理では接続処理（ハンドシェーク）を行わない為、改めて接続処理をしたい場合は再度 connect()を発行する必要があります。
- connect()にタイムアウトを設定するにはソケットオプション「SO_SNDTIMEO」を使用します。

5.5.4.8 send (ソケットへメッセージを送る)

【書式】

```
#include "sys/socket.h"
int send(int sockfd, const void *buf, unsigned int len, int flags);
```

【パラメータ】

int	sockfd	ソケットFD
const void *	buf	送信データのアドレス
unsigned int	len	送信データ長
int	flags	フラグ

【戻り値】

int	送信されたバイト数。エラーの場合は-1

【errno】

EINVAL	パラメータ不正 (bufにアドレスが設定されていない)
ENOMEM	メッセージバッファが枯渇している len相当のネットワークバッファが取得できない、もしくはlenの値が0
EBADF	sendできないソケット (TCP未接続・CLOSED状態)
EPIPE	ソケットFDが不正な状態
EDESTADDRREQ	宛先未設定(UDPソケット)
EACCESS	ブロードキャスト送信が未許可の為、送信拒否
EAGAIN	送信中(非同期実行時)
ETIMEDOUT	タイムアウト(タイムアウト設定時)
EINTR	待ち状態が強制解除された

- 送信データ長には 1～65535 を指定します。
- POSIX 仕様とは異なり、UDP の 0 バイトの送信はできません。
- UDP ソケットの送信においてフラグに MSG_DONTWAIT を指定した場合、送信データが下位レイヤのキューに積まれた時点で送信成功とします(*)
- フラグに USE_APLBUF ビットを指定した場合、ネットワークバッファの設定サイズに関係なく送信が可能になります。
- フラグに USE_APLBUF ビットを指定した場合、len サイズの送信が完了するか、エラーが発生するまで API は送信を試みますが、その間アプリケーションは buf が示す領域の一貫性を保証する必要があります。
- TCP ソケットに対する errno については 5.5.7.6 TCP ソケット errno ユースケースを参照して下さい。

(*) 下位レイヤのキューとは IP レイヤのアドレス解決処理、もしくはリンクレイヤの非同期送信処理を指します。

5.5.4.9 sendto (ソケットへメッセージを送る)

【書式】

```
#include "sys/socket.h"
int sendto(int sockfd, const void *buf, unsigned int len, int flags, const struct sockaddr *dest_addr, unsigned int addrlen);
```

【パラメータ】

int	sockfd	ソケットFD
const void *	buf	送信データのアドレス
unsigned int	len	送信データ長
int	flags	フラグ
const struct sockaddr *	dest_addr	送信先アドレス
unsigned int	addrlen	送信先アドレス長

【戻り値】

int	送信されたバイト数。エラーの場合は-1

【errno】

EINVAL	パラメータ不正(bufにアドレスが設定されていない)
ENOMEM	メッセージバッファが枯渇している
	len相当のネットワークバッファが取得できない、もしくはlenの値が0
EBADF	sendtoできないソケット (TCP未接続・CLOSED状態)
EPIPE	ソケットFDが不正な状態
EDESTADDRREQ	宛先未設定(UDPソケット)
EACCESS	ブロードキャスト送信が未許可の為、送信拒否
EAGAIN	送信中(非同期実行時)
ETIMEDOUT	タイムアウト(タイムアウト設定時)
EINTR	待ち状態が強制解除された

- 送信データ長には 1～65535 を指定します。
- TCP ソケットの場合、送信先アドレス、送信先アドレス長は使用しません。
- POSIX 仕様とは異なり、UDP の 0 バイトの送信はできません。
- UDP ソケットの送信においてフラグに MSG_DONTWAIT を指定した場合、送信データが下位レイヤのキューに積まれた時点で送信成功とします(*)
- フラグに USE_APLBUF ビットを指定した場合、ネットワークバッファの設定サイズに関係なく送信が可能になります。
- フラグに USE_APLBUF ビットを指定した場合、len サイズの送信が完了するか、エラーが発生するまで API は送信を試みますが、その間アプリケーションは buf が示す領域の一貫性を保証する必要があります。

(*) 下位レイヤのキューとは IP レイヤのアドレス解決処理、もしくはリンクレイヤの非同期送信処理を指します。

5.5.4.10 recv (ソケットからメッセージを受け取る)

【書式】

```
#include "sys/socket.h"
int recv(int sockfd, void *buf, unsigned int len, int flags);
```

【パラメータ】

int	sockfd	ソケットFD
void *	buf	受信バッファのアドレス
unsigned int	len	受信バッファ長
int	flags	フラグ

【戻り値】

int	受信したバイト数(0も含む)。エラーの場合は-1

【errno】

EINVAL	パラメータ不正 (bufにアドレスが設定されていない)
ENOMEM	メッセージバッファが枯渇している len相当のネットワークバッファが取得できない、もしくはlenの値が0
EBADF	recvできないソケット(TCP未接続・CLOSED状態)
EPIPE	ソケットFDが不正な状態
EAGAIN	パケット未受信(非同期実行時)
ETIMEDOUT	タイムアウト(タイムアウト設定時)
EINTR	待ち状態が強制解除された

- フラグに MSG_PEEK を指定した場合、ソケットの受信キューから削除されずにパケットを取得します。従って、次に受信コールを呼び出すと、同じパケットを取得できます。
- フラグに USE_APLBUF ビットを指定した場合、ネットワークバッファの設定サイズに関係なく受信が可能になりますが、パケットを受信するまでの間、アプリケーションは buf が示す領域の一貫性を保証する必要があります。
- 受信データ長には 1～65535 を指定します。
- 受信したパケットが無かった場合には、recv()はパケットを受信するまで処理をブロックします。
- TCP ソケットでまだリモートと接続していない場合は、recv()はエラーになります。
- TCP ソケットでリモートから切断された状態では、recv()は 0 を返却します。
- TCP ソケットに対する errno については 5.5.7.6 TCP ソケット errno ユースケースを参照して下さい。

5.5.4.11 recvfrom (ソケットからメッセージを受け取る)

【書式】

```
#include "sys/socket.h"
int recvfrom(int sockfd, void *buf, unsigned int len, int flags, struct sockaddr *src_addr, unsigned int *addrlen);
```

【パラメータ】

int	sockfd	ソケットFD
void *	buf	受信バッファのアドレス
unsigned int	len	受信バッファ長
int	flags	フラグ
struct sockaddr *	src_addr	送信元アドレス
unsigned int *	addrlen	送信元アドレス長

【戻り値】

int	受信したバイト数(0も含む)。エラーの場合は-1

【errno】

EINVAL	パラメータ不正 (bufにアドレスが設定されていない)
ENOMEM	メッセージバッファが枯渇している len相当のネットワークバッファが取得できない、もしくはlenの値が0
EBADF	recvfromできないソケット (TCP未接続・CLOSED状態)
EPIPE	ソケットFDが不正な状態
EAGAIN	パケット未受信(非同期実行時)
ETIMEDOUT	タイムアウト(タイムアウト設定時)
EINTR	待ち状態が強制解除された

- フラグに MSG_PEEK を指定した場合、ソケットの受信キューから削除されずにパケットを取得します。従って、次に受信コールを呼び出すと、同じパケットを取得できます。
- 受信データ長には 1～65535 を指定します。
- フラグに USE_APLBUF ビットを指定した場合、ネットワークバッファの設定サイズに関係なく受信が可能になりますが、パケットを受信するまでの間、アプリケーションは buf が示す領域の一貫性を保証する必要があります。
- 受信したパケットが無かった場合には、recvfrom()はパケットを受信するまで処理をブロックします。
- TCP ソケットでまだリモートと接続していない場合は、recvfrom ()はエラーになります。
- TCP ソケットでリモートから切断された状態では、recvfrom ()は 0 を返却します。
- 送信元アドレスは、アドレスファミリが AF_INET の場合は struct sockaddr_in*型、AF_INET6 の場合は struct sockaddr_in6*型で格納します。
- 指定された送信元アドレス長が小さ過ぎた場合、src_addr には指定されたアドレス長分のデータのみコピーし、addrlen にはアドレスファミリが AF_INET の場合は sockaddr_in 型のサイズ(16 バイト)、AF_INET6 の場合は sockaddr_in6 型のサイズ(28 バイト)を格納します。
- TCP ソケットの場合、送信元アドレス、アドレス長は使用しません。

5.5.4.12 shutdown (全二重接続の一部を閉じる)

【書式】

```
#include "sys/socket.h"
int shutdown(int sockfd, int how);
```

【パラメータ】

int	sockfd	ソケットFD
int	how	切断方向

【戻り値】

int	処理の成否。成功の場合は0、エラーの場合は-1

【errno】

EINVAL	パラメータ不正
ENOMEM	メッセージバッファが枯渇している
EBADF	shutdownできないソケット
EPIPE	接続されていない(TCPソケット)
EINTR	待ち状態が強制解除された

- 切断方向には、SHUT_WR か SHUT_RDWR のみ指定可能です。

5.5.4.13 close (ソケットをクローズする)

【書式】

```
#include "sys/unistd.h"
int close(int fd);
```

【パラメータ】

int	fd	ソケットFD

【戻り値】

int	処理の成否。成功の場合は0、エラーの場合は-1

【errno】

EINVAL	パラメータ不正
ENOMEM	メッセージバッファが枯渇している
EBADF	closeできないソケット
EINTR	待ち状態が強制解除された

- 切断されていない TCP ソケットの場合、TCP セッションを切断後、ソケットをクローズします。
- クローズしたソケット FD は再び生成するまでは使用できません。

5.5.4.14 select (同期 I/O の多重化)

【書式】

```
#include "sys/select.h"
int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);
```

【パラメータ】

int	nfds	readfdsとwritefdsに含まれるソケットFDの中で最大値に1足した値
fd_set *	readfds	読み込み可能かを監視するソケットFDセット
fd_set *	writefds	書き込み可能かを監視するソケットFDセット
fd_set *	exceptfds	例外を監視するソケットFDセット(未サポート)
struct timeval *	timeout	監視タイムアウト

【戻り値】

int	読み書き可能になったソケットFDの数。タイムアウトの場合は0、エラーの場合は-1

【errno】

EINVAL	パラメータ不正
ENOMEM	メッセージバッファが枯渇している
EBADF	selectできないソケットFDがセットされている

- exceptfds は使用しません。
- POSIX 仕様とは異なり、生成直後のソケット FD に対する select()の実行では、UDP ソケットの場合は書き込み可能、読み込み不可です。(ただし既にパケットを受信していれば読み込みも可能)。TCP ソケットの場合は読み込み可能、書き込み不可となります。
- select()は監視対象のソケットの状態(Read/Write のみ)が有効になった時にだけ、引数の fdset を 3 つとも更新してセットされたソケット FD の数を返却します。タイムアウトやエラーが発生した場合はこれらの引数は更新されないため、アプリケーションは初めに戻り値を検証して下さい。

5.5.4.15 getsockname (ソケットの名前を取得する)

【書式】

```
#include "sys/socket.h"
int getsockname(int sockfd, struct sockaddr *addr, unsigned int *addrlen);
```

【パラメータ】

int	sockfd	ソケットFD
struct sockaddr *	addr	ソケットアドレス格納バッファ
unsigned int *	addrlen	ソケットアドレス格納バッファサイズ

【戻り値】

int	処理の成否。成功の場合は0、エラーの場合は-1

【errno】

EINVAL	パラメータ不正
ENOMEM	メッセージバッファが枯渇している
EBADF	getsocknameできないソケット
EINTR	待ち状態が強制解除された

- *addrlen には、アドレスファミリが AF_INET の場合は sockaddr_in のサイズ(16 バイト以上)、AF_INET6 の場合は sockaddr_in6 のサイズ(28 バイト以上)を設定する必要があります。
- ソケットアドレスは次の API を発行した時点で決定します。
  bind()
  connect()
  accept()
  send/sendto()
  recv/recvfrom()
  尚、これらの API が失敗した場合のソケットアドレスについては不定値となります。

5.5.4.16 getpeername (接続している相手ソケットの名前を取得する)

【書式】

```
#include "sys/socket.h"
int getpeername(int sockfd, struct sockaddr *addr, unsigned int *addrlen);
```

【パラメータ】

int	sockfd	ソケットFD
struct sockaddr *	addr	リモートアドレス格納バッファ
unsigned int *	addrlen	リモートアドレス格納バッファサイズ

【戻り値】

int	処理の成否。成功の場合は0、エラーの場合は-1

【errno】

EINVAL	パラメータ不正
ENOMEM	メッセージバッファが枯渇している
EBADF	getpeernameできないソケット
ENOTCONN	宛先が設定されていない
EINTR	待ち状態が強制解除された

- *addrlen には、アドレスファミリが AF_INET の場合は sockaddr_in のサイズ(16 バイト以上)、AF_INET6 の場合は sockaddr_in6 のサイズ(28 バイト以上)を設定する必要があります。
- TCP の場合、接続されたソケットのみリモートアドレスが取得できます。
- UDP の場合、connect もしくは sendto で宛先が設定されたソケット、またはパケットを受信したソケットでのみリモートアドレスが取得できます。

5.5.4.17 getsockopt (ソケットのオプションの取得を行なう)

【書式】

```
#include "sys/socket.h"
int getsockopt(int sockfd, int level, int optname, void *optval, unsigned int *optlen);
```

【パラメータ】

int	sockfd	ソケットFD
int	level	オプションレベル
int	optname	オプション名
void *	optval	取得値格納用バッファ
unsigned int *	optlen	取得値格納用バッファサイズ

【戻り値】

int	処理の成否。成功の場合は0、エラーの場合は-1

【errno】

EINVAL	パラメータ不正
ENOMEM	メッセージバッファが枯渇している
EBADF	getsockoptできないソケット
EPROTONOSUPPORT	サポートしていないオプション
EINTR	待ち状態が強制解除された

- オプションレベルには SOL_SOCKET、IPPROTO_IP、IPPROTO_TCP、IPPROTO_IPV6 の指定が可能です。
- 各オプションレベルで取得可能なオプション名は、「5.5.5 ソケットオプション」を参照してください。

5.5.4.18 setsockopt (ソケットのオプションの設定を行なう)

【書式】

```
#include "sys/socket.h"
int setsockopt(int sockfd, int level, int optname, const void *optval, unsigned int optlen);
```

【パラメータ】

int	sockfd	ソケットFD
int	level	オプションレベル
int	optname	オプション名
const void *	optval	設定値バッファ
unsigned int	optlen	設定値バッファサイズ

【戻り値】

int	処理の成否。成功の場合は0、エラーの場合は-1

【errno】

EINVAL	パラメータ不正
ENOMEM	メッセージバッファが枯渇している
EBADF	setsockoptできないソケット
EPIPE	ソケットFDが不正な状態
EPROTONOSUPPORT	サポートしていないオプション
EINTR	待ち状態が強制解除された

- オプションレベルには SOL_SOCKET、IPPROTO_IP、IPPROTO_TCP、IPPROTO_IPV6 の指定が可能です。
- 各オプションレベルで設定可能なオプション名は、「5.5.5 ソケットオプション」を参照してください。

5.5.4.19 ioctl (ソケットを制御する)

【書式】

```
#include "sys/ioctl.h"
int ioctl(int d, int request, ...);
```

【パラメータ】

int	d	ソケットFD
int	request	リクエスト
...		リクエストパラメータ

【戻り値】

int	処理の成否。成功の場合は0、エラーの場合は-1

【errno】

EINVAL	パラメータ不正
EBADF	指定されたソケットディスクリプタが不正
ENOMEM	メッセージバッファが枯渇している
EFAULT	リクエストパラメータを展開できない
EINTR	待ち状態が強制解除された

- リクエストには次に示すパラメータの設定が可能です。

リクエストコード	意味	パラメータ
FIONBIO	ノンブロッキング通信設定	1(設定)、0(解除)
FIONREAD	ソケットが保持している受信パケットのバイト数を取得	(unsigned int *)&nread

- ノンブロッキング通信設定に関しては「5.5.6.1 ノンブロッキング設定」を参照して下さい。
- FIONREAD オプションで取得出来る値について、TCP ソケットの場合は受信ウィンドウバッファに保持されている受信パケットサイズ（全体サイズ）が設定されます。UDP ソケットの場合は次に受信する受信パケットブロックのサイズ（先頭のみ）が設定されます。

5.5.4.20 inet_addr (IP アドレス操作ルーチン)

【書式】

```
#include "arpa/inet.h"
unsigned int inet_addr(const char *cp);
```

【パラメータ】

const char *	cp	ドット表記IPアドレス

【戻り値】

unsigned int	変換後IPアドレスバイナリ値(ネットワークバイトオーダー)

【errno】

設定されません

- 変換に失敗した場合は戻り値として 0 を返却します。

5.5.4.21 inet_aton (IP アドレス操作ルーチン)

【書式】

```
#include “arpa/inet.h”
int inet_aton(const char *cp, struct in_addr *inp);
```

【パラメータ】

const char *	cp	ドット表記IPアドレス
struct in_addr *	inp	変換後IPアドレスバイナリ値(ネットワークバイトオーダー)格納バッファ

【戻り値】

int	処理の成否。成功の場合は0、エラーの場合は-1

【errno】

設定されません

- 変換に失敗した場合は戻り値として-1 を返却します。

5.5.4.22 inet_ntoa (IP アドレス操作ルーチン)

【書式】

```
#include “arpa/inet.h”
char *inet_ntoa(struct in_addr in);
```

【パラメータ】

struct in_addr	in	IPアドレスバイナリ値(ネットワークバイトオーダー)

【戻り値】

char *	変換後ドット表記IPアドレス

【errno】

設定されません

- 変換後ドット表記 IP アドレスを格納する領域は静的に割当てられたバッファに格納されて返されるので、 この後でこの関数を再度呼び出すと文字列は上書きされます。

5.5.4.23 if_nametoindex (インタフェース名とインデックスのマッピング)

【書式】

```
#include "net/if.h"
unsigned int if_nametoindex(const char *ifname)
```

【パラメータ】

const char *	ifname	インタフェース名

【戻り値】

unsigned int	インタフェースindex。エラーの場合は0

【errno】

ENXIO	存在しないインタフェース名

- インタフェース名は TCP/IP プロトコルスタックで使用されるデバイス名(gNET_DEV[index-1].name[8])によって設定されます。

5.5.4.24 if_indextoname (インタフェース名とインデックスのマッピング)

【書式】

```
#include "net/if.h"
char *if_indextoname(unsigned int ifindex, char *ifname)
```

【パラメータ】

unsigned int	ifindex	インタフェースindex
char *	ifname	インタフェース名格納用バッファ

【戻り値】

char*	処理の成否。成功の場合はifname、エラーの場合はNULL

【エラーコード】

ENXIO	存在しないインタフェースindex

- インタフェース名は TCP/IP プロトコルスタックで使用されるデバイス名(gNET_DEV[index-1].name[8])によって設定されます。

5.5.4.25 rresvport (ポートにバインドされたソケットを取得)

【書式】

```
#include “sys/unistd.h”
int rresvport(int *port)
```

【パラメータ】

int *	port	ポート番号格納用バッファ

【戻り値】

int	ポートにbindされているソケットFD。ソケットが無い場合は-1

【errno】

設定されません

5.5.4.26 getifaddrs (インタフェースのアドレスを取得)

【書式】

```
#include "sys/types.h"
int getifaddrs(struct ifaddrs **ifap)
```

【パラメータ】

struct ifaddrs**	ifap	インタフェース情報リスト先頭アドレス

【戻り値】

int	処理の成否。成功の場合は 0、エラーの場合は-1

【errno】

ENOMEM	インタフェース情報格納領域の取得エラー

- アプリケーションで設定しているデバイス数分(CFG_DEV_MAX)のインタフェース情報をチェーンで取得します。
- 正常に終了した場合 ifap には次の値が格納されます。
  (*ifap)->ifa_next にはインタフェース情報の次の構造体のポインタが格納されます。最後の要素の場合このフィールドは NULL です。
  (*ifap)->name にはインタフェース名を指すポインタが格納されます。
  (*ifap)-> ifa_flags にはデバイス番号が格納されます。
  (*ifap)-> ifa_addr にはデバイスの IP アドレスが sockaddr 型ポインタで格納されます。
  (*ifap)-> ifa_netmask にはデバイスのサブネットマスクが sockaddr 型ポインタで格納されます。
  (*ifap)->ifa_ifu.ifu_broadaddr にはデバイスのブロードキャストアドレスが sockaddr 型ポインタで格納されます。(*ifap)->ifa_addr の IP アドレスが AF_INET6 の場合は、(*ifap)->ifa_ifu.ifu_broadaddr は NULL となります。
  (*ifap)->ifa_ifu.ifu_dstaddr と(*ifap)->ifa_data フィールドは使用しません。
- インタフェース情報リストは動的に確保されるため、正常終了後は、freeifaddrs()関数でインタフェース情報リストを解放する必要があります。

5.5.4.27 freeifaddrs (インタフェース情報リストの解放)

【書式】

```
#include "sys/unistd.h"
void freeifaddrs(struct ifaddrs *ifa)
```

【パラメータ】

struct ifaddrs*	ifap	インタフェース情報リスト先頭アドレス

【戻り値】

void

【errno】

設定されません

- getifaddrs()で取得したインタフェース情報リストを解放します。

5.5.4.28 inet_pton (IPv4/IPv6 アドレスをテキスト形式からバイナリ形式に変換)

【書式】

```
#include “arpa/inet.h”
int inet_pton(int af, const char *src, void *dst);
```

【パラメータ】

int	af	アドレスファミリ
const char *	src	ネットワークアドレス文字列
void *	dst	ネットワークアドレス構造体の格納バッファ

【戻り値】

int	処理の成否。成功の場合は1、エラーの場合は-1

【errno】

EINVAL	パラメータ不正
EAFNOSUPPORT	非サポートのアドレスファミリ

- アドレスファミリは AF_INET、AF_INET6 のみ指定可能です。AF_INET、AF_INET6 以外を指定した場合は、戻り値として-1 を返却します。
- src に指定された文字列が、アドレスファミリに対する正しいネットワークアドレス表記ではない場合は、dst にはネットワークアドレス構造体の各メンバが全て 0 で格納され、戻り値は 1 となります。

5.5.4.29 inet_ntop (IPv4/IPv6 アドレスをバイナリ形式からテキスト形式に変換)

【書式】

```
#include “arpa/inet.h”
const char *inet_ntop(int af, const void *src, char *dst, unsigned int size);
```

【パラメータ】

int	af	アドレスファミリ
const void *	src	ネットワークアドレス構造体
char *	dst	変換結果文字列の格納バッファ
unsigned int	size	変換結果文字列の格納バッファ長

【戻り値】

const char *	dstへのポインタ。エラーの場合はNULL

【errno】

EINVAL	パラメータ不正
EAFNOSUPPORT	非サポートのアドレスファミリ
ENOSPC	格納バッファサイズ不足

- アドレスファミリは AF_INET、AF_INET6 のみ指定可能です。
- 変換に失敗した場合は戻り値として NULL を返却します。

5.5.5 ソケットオプション

setsockopt()/getsockopt()API で取得・設定が可能なオプション一覧を以下に示します。
記載されているオプション以外を指定した場合、setsockopt()/getsockopt()は-1 を返却します。

表 5.2 ソケットオプション

オプション名	値の型	用途
SOL_SOCKET レベル		
SO_ACCEPTCONN	int	TCP ソケットの LISTEN 状態の取得。GET のみ。
SO_BROADCAST	int	UDP ブロードキャスト送信動作のみに作用。GET/SET。
SO_DOMAIN	int	ソケットドメインの取得。GET のみ。
SO_ERROR	int	ソケットエラーの取得。GET のみ。
SO_KEEPALIVE (*1)	int	TCP ソケットの Keep Alive 機能の有効化。SET のみ。
SO_RCVBUF	int	受信バッファの設定。TCP の場合、受信 Window の Byte 数。UDP の場合は受信パケット数(キューサイズ)として扱う。GET/SET。
SO_RCVBUFFORCE	int	SO_RCVBUF と同じ。
SO_RCVTIMEO	timeval	ソケットの受信タイムアウト設定。GET/SET。
SO_SNDTIMEO	timeval	ソケットの送信、接続タイムアウト設定。GET/SET。
SO_TYPE	int	ソケットタイプの取得。GET のみ。
SO_REUSEADDR	int	UDP ソケットのローカルポート重複 bind を許可。GET/SET。
IPPROTO_IP レベル		
IP_ADD_MEMBERSHIP	ip_mreqn	マルチキャストグループへ参加。UDP ソケットのみ有効。SET のみ。
IP_DROP_MEMBERSHIP	ip_mreqn	マルチキャストグループの離脱。SET のみ。
IP_MTU	int	パス MTU の取得。GET のみ。
IP_MULTICAST_TTL	int	マルチキャスト送信パケットの TTL 設定。GET/SET。
IP_MULTICAST_IF	ip_mreqn	マルチキャスト送信パケットのデバイス設定。GET/SET
IP_MULTICAST_LOOP	int	マルチキャスト送信パケットのループバック設定。GET/SET。
IP_TOS	int	IP 送信パケットの TOS 設定。GET/SET。
IP_TTL	int	IP 送信パケットの TTL 設定。GET/SET。
IPPROTO_TCP レベル		
TCP_KEEPCNT (*1)	int	TCP Keep-Alive プローブ回数の設定。SET のみ。
TCP_KEEPIDLE (*1)	int	TCP Keep-Alive 起動無通信間隔の設定。SET のみ。
TCP_KEEPINTVL (*1)	int	TCP Keep-Alive プローブ送信間隔の設定。SET のみ。
TCP_MAXSEG	int	TCP ソケットの MSS 値の設定。GET/SET。
IPPROTO_IPV6 レベル		
IPV6_ADD_MEMBERSHIP	ipv6_mreq	マルチキャストグループへ参加。UDP ソケットのみ有効。SET のみ。
IPV6_DROP_MEMBERSHIP	ipv6_mreq	マルチキャストグループの離脱。SET のみ。

(*1) TCP Keep Alive の有効/無効化(SO_KEEPALIVE)は、TCP を接続する前に設定する必要があります。
また TCP Keep Alive の有効/無効化およびその他の Keep Alive の設定は、全てのソケットで共有されます。

5.5.6 サポート機能

5.5.6.1 ノンブロッキング設定

ioctl()を使用してソケットの API 呼出しをノンブロッキング(ブロッキング)に設定にすることができます。初期状態ではすべての API がブロッキング設定です。ノンブロッキングを設定した場合 API はエラー番号に EAGAIN を設定し、-1 を返却することがあります。表 5.3ノンブロッキング API にノンブロッキング設定が有効となる API と、エラー番号が EAGAIN になる条件、そしてアプリケーションが振る舞うべき動作を記します。

尚、ノンブロッキング動作を行う API には、ソケットオプションによるタイムアウトの設定は作用しません。また TCP/IP スタック/BSD の API はタスク間通信を介する仕様上ノンブロッキング設定下でも API 実行時には呼び出し元のタスクが起床待ち状態になることがあります。

表 5.3　ノンブロッキング API

API	条件	アプリケーション動作
connect	TCP ソケットの場合は必ず戻り値は-1, エラー番号が EAGAIN になります。	TCP ソケットは-1 を返却した後もリモートからの SYN/ACK を待ち続け規定時間 SYN を再送します。SYN/ACK を受信した時点で select により書き込み可能になるため、当該 TCP ソケットを writefds で監視します。書き込み可能になった後に再度 connect を実行する必要はありません。
accept	listen ソケットに接続してきたソケットが無い場合に戻り値は-1, エラー番号が EAGAIN になります。	リモートから SYN を受信した時点で select により読み込み可能になるため、当該 TCP ソケットを readfds で監視します。読み込み可能になった後に再度 accept を実行します。
send sendto	TCP ソケットの場合で送信バッファに空きが無い場合は EAGAIN になります。 UDP ソケットの場合、既にソケットが送信中の場合は EAGAIN になります。	send/sendto で EAGAIN が示された場合、ソケットの状態によってパケットが送信できなかったことを示します。(その後もパケットは送出されません)
recv recvfrom	パケット未受信の場合、EAGAIN になります。	リモートからパケットを受信した時点で select により読み込み可能になるため当該ソケットを readfds で監視します。読み込み可能になった後に再度 recv を実行します。

5.5.6.2 ループバック

宛先アドレスとしてローカルループバックアドレス(127.0.0.1～127.255.255.254)を指定した場合、送信したパケットはそのネットワーク I/F に通知されます。

TCP/IP スタック/BSD では、ループバックアドレスは特定のデバイス I/F を持っておらず、送信専用のアドレスとして扱われます。そのためループバックアドレスに対して bind()を実行することはできません。

5.5.6.3 エラー処理

シンボル errno は唯一の広域変数です。この値は、API 実行中に発生したエラーに伴い更新されます。

もし複数のタスクから API を実行する場合は、errno の整合性を保障するために get_errno()関数を使って、タスク内で発生した最後の errno を取得することを推奨します。

【書式】

```
#include "sys/errno.h"
int get_errno(void)
```

【パラメータ】

void

【戻り値】

int　　本APIを呼び出したタスクで発生した最後のerrno

【errno】

設定されません

- タスク毎の errno は、アプリケーションが用意する広域変数 UW tsk_errno[]に保存されます。この配列要素にはあらかじめ最大タスク数を設定する必要があります。

表 5.4　errno 一覧

errno	値	説明
EINTR	4	API の待ち状態が強制的に解除された
ENXIO	6	インタフェースが存在しない
EBADF	9	ソケット FD が無効
ENOMEM	12	メモリ不足
EACCESS	13	要求された処理に対するアクセス拒否
EFAULT	14	パラメータ異常
ENODEV	19	システム内で致命的(もしくは不明)な異常
EINVAL	22	パラメータ誤り
EPIPE	32	ソケットオブジェクトが不正
EAGAIN	35	ブロッキング処理を実行した
EALREADY	37	既に実行中の処理
EDESTADDRREQ	39	宛先設定が必要
EPROTONOSUPPORT	43	機能が未サポート
EAFNOSUPPORT	47	アドレスファミリが未サポート
EADDRINUSE	48	アドレスが既に使用中
EADDRNOTAVAIL	49	アドレスが使用できない
EISCONN	56	ソケットが既に接続されている
ENOTCONN	57	ソケットが接続されていない
ETIMEDOUT	60	タイムアウト
ECONNREFUSED	61	接続が拒否された
EHOSTUNREACH	65	アクセス不可能なノードに対して接続しようとした

5.5.7 BSD アプリの実装

5.5.7.1 ソースコード

BSD ソケットを使用するアプリケーションは、Middleware/uNet3/bsd/直下の 4 つのソースコード (unet3_iodev.c, unet3_option.c, unet3_socket.c, unet3_wrap.c)をプロジェクトに取り込む必要があります。

```
Middleware
  |
 +--uNet3
     +--bsd
        +--unet3_Iodev.c
        +--unet3_option.c
        +--unet3_socket.c
        +--unet3_wrap.c
```

加えて、BSD 向けのライブラリ(libunet3bsd.a)をリンクする必要があります。

```
Library
 |
 └────ARM or GCC or IAR
        +--libune3bsd.a       /* BSDソケット用ライブラリ */
```

5.5.7.2 インクルードパス

BSD ソケットを使用するアプリケーションは、Middleware/uNet3/bsd/以下のディレクトリ unet3_posix と inc を、インクルードパス設定に追加する必要があります。

```
Middleware
  |
 +--uNet3
     +--bsd
        |
        +--unet3_posix   /* BSDソケットのインクルードベースフォルダ*/
        |   |   :
        |
        +--inc           /*μNet3ソケットのインクルードベースフォルダ*/
```

5.5.7.3 コンフィギュレーション

TCP/IP スタック/BSD では、アプリケーションで使用する最大ソケット数とアプリケーションのタスク数を、予め unet3_cfg.h にマクロ定義する必要があります。

<u>最大ソケット数</u>

```
#define     BSD_SOCKET_MAX
```

最大ソケット数はプロトコルに関係なく、アプリケーションが同時に生成するソケット数を表します(listen のバックログも含みます)。このマクロ定義は後述する BSD ソケット管理テーブル数や fd_set 型の定義に使用されます。この値はμNet3 ソケット最大数(CFG_NET_SOC_MAX)と同じ値でなければなりません。

<u>アプリケーションタスク数</u>

```
#define     NUM_OF_TASK_ERRNO
```

アプリケーションタスク数は、カーネルで生成可能なタスク数を表します。このマクロ定義は、後述するエラー番号管理テーブル数に使用されます。この値は、BSD の使用有無に関係なく、生成可能なタスク数を設定して下さい。

5.5.7.4 リソース定義

BSD を使用するアプリケーションは、情報を管理するためのテーブルを用意する必要があります。

<u>BSD ソケット管理テーブル</u>

```
T_UNET3_BSD_SOC     gNET_BSD_SOC[BSD_SOCKET_MAX];
```

BSD ソケット管理テーブルは、要素数 BSD_SOCKET_MAX の T_UNET3_BSD_SOC 型配列として広域変数を定義します。

<u>エラー番号管理テーブル</u>

```
UW tsk_errno[NUM_OF_TASK_ERRNO];
```

エラー番号管理テーブルは、要素数 NUM_OF_TASK_ERRNO の UW 型配列として広域変数を定義します。

5.5.7.5　カーネルオブジェクト

TCP/IP スタック/BSD が使用するカーネルオブジェクトは、以下の通りです。

リソース名	用途	ID
タスク	BSD Wrapper タスク	ID TSK_BSD_API
	ループバックデバイスタスク	ID_LO_IF_TSK
メールボックス	BSD Wrapper タスク間通信	ID MBX_BSD_REQ
	ループバックデバイスタスク間通信	ID_LO_IF_MBX
メモリプール	メッセージバッファ	ID MPF_BSD_MSG

5.5.7.6　TCP ソケット errno ユースケース

TCP ソケットに対する send(), recv(), および accept()実行中に、他の API の実行や通信イベントが発生し、待ち状態だった元の API が終了する場合、この時の戻り値と errno を下表に示します。

send(), recv(), および accept()の実行中とはそれぞれ、送信バッファの空きを待っている状態、パケットの受信を待っている状態、それと TCP パッシブ接続待ちの状態です。

イベント	send		recv		accept	
	戻り値	errno	戻り値	errno	戻り値	errno
close()実行	待ち継続	—	-1	EBADF	-1	EINTR
shutdown(WR)実行	待ち継続	—	待ち継続	—	待ち継続	—
shutdown(RDWR)実行	待ち継続	—	0	—	待ち継続	—
FIN 受信	待ち継続	—	0	—		
RST 受信	-1	EBADF	-1	EBADF		
ソケットオプション タイマ満了	-1	ETIMEDOUT	-1	ETIMEDOUT	-1	ETIMEDOUT
TCP タイマ満了	-1	ETIMEDOUT	-1	ETIMEDOUT		

5.5.7.7 初期化

アプリケーションは BSD ソケット API を使用する前に、unet3_bsd_init()関数を呼び出して BSD モジュールを初期化する必要があります。また、この BSD モジュールを初期化するには、μNet3 の初期化とデバイスドライバの初期化が正常に終了している必要があります。

【書式】

```
#include "sys/socket.h"
ER unet3_bsd_init(void)
```

【パラメータ】

void

【戻り値】

ER	処理の成否。成功の場合はE_OK、失敗の場合はエラーコード

【エラーコード】

E_SYS	カーネルオブジェクトの初期化処理に失敗

使用例

```
ER net_sample(void)
{
    /* Initialize TCP/IP Stack */
    ER ercd;

    ercd = net_ini();
    if (ercd != E_OK) {
        return ercd;
    }

    /* Initialize Ethernet Driver */
    ercd = net_dev_ini(1);
    if (ercd != E_OK) {
        return ercd;
    }

    /* BSD wrapper */
    ercd = unet3_bsd_init();

    return ercd;
}
```

5.6 その他 API

htons	16ビット値をネットワークバイトオーダーへ変換

【書式】

UH htons(UH val);

【パラメータ】

UH	val	ホストバイトオーダーの16ビット値

【戻り値】

UH		ネットワークバイトオーダーの16ビット値

htonl	32ビット値をネットワークバイトオーダーへ変換

【書式】

UW htonl(UW val);

【パラメータ】

UW	val	ホストバイトオーダーの32ビット値

【戻り値】

UW		ネットワークバイトオーダーの32ビット値

ntohs	16ビット値をホストバイトオーダーへ変換

【書式】

UH ntohs(UH val);

【パラメータ】

UH	val	ネットワークバイトオーダーの16ビット値

【戻り値】

UH		ホストバイトオーダーの16ビット値

ntohl	32ビット値をホストバイトオーダーへ変換

【書式】

UW ntohl(UW val);

【パラメータ】

UW	val	ネットワークバイトオーダーの32ビット値

【戻り値】

UW		ホストバイトオーダーの32ビット値

ip_aton　　ドット表記のIPv4アドレス文字列を32ビット値に変換

【書式】

```
UW ip_aton(const char *str);
```

【パラメータ】

char	str	ドット表記のIPv4アドレス文字列へのポインタ

【戻り値】

UW	>0	正常終了（変換後32ビット値）

【エラーコード】

0	不正なIPアドレスが指定された

ip_ntoa　　32ビット値のIPv4アドレスをドット表記のIPv4アドレス文字列に変換

【書式】

```
void ip_ntoa(const char *str, UW ipaddr);
```

【パラメータ】

char	str	変換後、IPアドレス文字列を受け取るポインタ
UW	ipaddr	IPアドレスの32ビット値

【戻り値】

なし

【解説】

正常終了した時は、str に文字列がセットされる。エラーの時、str は NULL となる。

ip_byte2n	IPv4アドレスの配列を32ビット値に変換

【書式】

UW ip_byte2n(char *ip_array);

【パラメータ】

char	ip_array	IPアドレスのバイト値配列へのポインタ

【戻り値】

UW	>0	正常終了（変換後32ビット値）

【エラーコード】

0	不正なIPアドレスが指定された

ip_n2byte	IPv4アドレスの32ビット値を配列に変換

【書式】

void ip_n2byte(char *ip_arry, UW ip);

【パラメータ】

char	ip_array	IPアドレスのバイト値配列へのポインタ
UW	ip	IPアドレスの32ビット値

【戻り値】

なし

【解説】

正常終了した時は、ip_array に値がセットされる。エラーの時、ip_array は NULL となる。

arp_set	静的ARPエントリの設定

【書式】

```
ER ercd = arp_set(UH dev_num, UW ip, UB *mac);
```

【パラメータ】

UH	dev_num	デバイス番号
UW	ip	IPv4アドレス
UB	*mac	MACアドレス

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	デバイス番号不正
E_PAR	IPv4アドレス、MACアドレス不正
E_NOMEM	ARPキャッシュに空きがない

【解説】

ARP キャッシュに静的エントリを登録します。dev_num には該当アドレスが存在するネットワークに繋がるデバイス番号を指定します。

ip と mac には該当ホストの IP アドレスと MAC アドレスをそれぞれ設定します。MAC アドレスは UB 型の 6byte 配列にビッグエンディアンで設定します。既に動的エントリや他の静的エントリによって ARP キャッシュに空きがない場合は E_NOMEM が返却されます。その場合アプリケーションは通信前にエントリを設定するか ARP キャッシュのサイズを増やして下さい。

arp_ref	ARPキャッシュの参照

【書式】

```
ER ercd = arp_ref(UH dev_num, UW ip, UB *mac);
```

【パラメータ】

UH	dev_num	デバイス番号
UW	ip	検索対象IPv4アドレス
UB	*mac	検索結果MACアドレス

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	デバイス番号不正
E_NOEXS	該当アドレスがなし

【解説】

ARP キャッシュ中の MAC アドレスを参照します。dev_num には該当アドレスが存在するネットワークに繋がるデバイス番号を指定します。

ip には参照したホストの IP アドレスを指定します。該当する IP アドレスが見つかった場合、6byte の MAC アドレスを引数 mac へコピーします。

arp_req　　ARPリクエストの送信

【書式】

```
ER ercd = arp_req(UH dev_num, UW ip);
```

【パラメータ】

UH	dev_num	デバイス番号
UW	ip	検索対象IPv4アドレス

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	デバイス番号不正
E_OBJ	該当デバイス自身がアドレスを持っていない(起動していない)
E_NOMEM	ネットワークバッファが確保できない
E_PAR	該当デバイスでアドレス解決できない(ループバックやPPPなど)

【解説】

アプリケーションから任意のタイミングで ARP リクエストを送信します。この処理は ARP レスポンスを待ちません。

正常な ARP レスポンスを得た場合、その応答内容は ARP キャッシュに保存されます。アプリケーションは arp_ref( )を使用してこれを参照することが可能です。

arp_clr　　ARPキャッシュのクリア

【書式】

```
ER ercd = arp_clr(void);
```

【パラメータ】

なし

【戻り値】

ER	ercd	正常終了（E_OK）

【エラーコード】

なし

【解説】

ARP キャッシュのエントリをすべて削除します。アプリケーションは物理的にネットワークが変わった場合や(リンクダウン／リンクアップ検出後)、IP アドレスが変わった場合のタイミングで ARP キャッシュをクリアすることができます。

arp_del	ARPエントリの削除

【書式】

```
ER ercd = arp_del(UH dev_num, UW ip);
```

【パラメータ】

UH	dev_num	デバイス番号
UW	ip	削除対象IPv4アドレス

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	デバイス番号不正

【解説】

静的に設定した ARP エントリをキャッシュから削除します。削除対象となるのは、指定した IP アドレスで尚且つ、アプリケーションが静的に登録したエントリです。動的に取得したエントリは削除されません。

6. ネットワークアプリケーション

6.1 DHCP クライアント

DHCP クライアントは、DHCP サーバーからネットワークで利用できる IP アドレス情報を取得します。取得した IP アドレスはホストに割り当てられます。

DHCP 拡張版を使用すると、RENEW, RELEASE, DECLINE, INFORM 機能が使えます。DHCP 拡張版については、「6.5　DHCP クライアント拡張版」を参照して下さい。

(1) ホストアドレス情報

```
typedef struct t_host_addr {
        UW      ipaddr;         /* IP アドレス */
        UW      subnet;         /* サブネットマスク */
        UW      gateway;        /* ゲートウェイ */
        UW      dhcp;           /* DHCP サーバーアドレス */
        UW      dns[2];         /* DNS アドレス */
        UW      lease;          /* DHCP アドレスのリース期間 */
        UW      t1;             /* DHCP アドレスのリニューアル期間*/
        UW      t2;             /* DHCP アドレスのリバインド期間 */
        UB      mac[6];         /* MAC アドレス */
        UH      dev_num;        /* デバイス番号 */
        UB      state;          /* DHCP クライアント状態 */
        SID     socid;          /* UDP ソケット ID */
} T_HOST_ADDR ;
```

この構造体は DHCP クライアント API の引数として使用されます。**デバイス番号**と **UDP ソケット ID** はユーザアプリケーションでセットする必要があります。残りのパラメータは DHCP サーバーからの応答によりセットされます。

- UDP ソケット ID

DHCP クライアントでは UDP ソケットを使用します。UDP ソケットは次のようパラメータで作成する必要があります。（DHCP クライアントアプリケーション内で生成されます）

プロトコル	ID	ポート	送信タイムアウト	受信タイムアウト
UDP	ID_SOC_DHCP	68	3 秒	3 秒

- デバイス番号

デバイス番号には DHCP クライアントで使用するネットワークデバイスを指定します。’0’を指定した場合はデフォルトのネットワークデバイスが使用されます。

6.1.1 DHCP クライアント API

dhcp_client	DHCPクライアントの開始

【書式】

```
ER ercd = dhcp_client(T_HOST_ADDR *addr);
```

【パラメータ】

T_HOST_ADDR	*addr	ホストアドレス情報

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	*addrがNULLまたはsocidが指定されていない。
E_NOMEM	メモリ不足（ネットワークバッファの枯渇）
E_OBJ	ソケットの状態が不正（ソケットが未作成）
E_TMOUT	DHCPサーバーからの応答が遅延。またはDHCPサーバーが存在しない。

【解説】

この API は DHCP サーバーから IP アドレス、サブネットマスク、ゲートウェイアドレスを取得してホストに割り当てます。使用しているネットワークの構成によっては E_TMOUT エラーが発生することがあります。その時は正常終了するまでリトライしてみることを推奨します。

またこの API は新規に DHCP セッションを開始します。つまり API を呼び出すと、必ず DISCOVER の送信を開始して、OFFER の受信、REQUEST の送信、ACK の受信を期待する動作になります。

DHCP サーバーから取得した IP アドレスの有効期限は’lease（リース期間）’で指定されています。DHCP クライアントは有効期限が切れる前に新たにリースを行うには次のようにします。

使用例

```
void dhcp_tsk(VP_INT exinf)
{
    ER ercd;
    T_HOST_ADDR dhcp_addr = {0};
    UB status = DHCP_STS_INIT;

    dhcp_addr.socid = ID_SOC_DHCP;
    dhcp_addr.dev_num = ID_DEVNUM_ETHER;

    for (;;) {
        ercd = dhcp_client(&dhcp_addr);
        if (ercd == E_OK) {
            /* BOUND 期間 */
            dly_tsk(dhcp_addr.t1*1000);
            /* RENEWING 期間 */
            status = DHCP_STS_RENEWING;
            continue;
        }
        if (status == DHCP_STS_RENEWING) {
            /* REBINDING 期間 */
            dly_tsk((dhcp_addr.t2-dhcp_addr.t1)*1000);
            status = DHCP_STS_INIT;
            continue;
        }
        /* INIT 期間 */
        dly_tsk(1000);
    }
}
```

なお、REQUEST メッセージでリース期間を延長する場合、DHCP クライアント拡張版をご使用下さい。

6.2 FTP サーバー

FTP サーバーは、リモートホストに対してファイルのアップロードとダウンロードを可能にします。モードはアクティブモードとパッシブモードが使用できます。

(1) FTP サーバー制御情報

```
typedef struct t_ftp_server {
    ・      ・
    UW      sec;                /* セキュリティポリシー */
    ・      ・
    ・      ・
    UH      dev_num;            /* デバイス番号*/
    SID     ctl_sid;            /* 制御用ソケット ID */
    SID     dat_sid;            /* データ用ソケット ID */
    ・      ・
    ・      ・
    ER (*auth_cbk)(UH, const char*, const char*);   /* 認証コールバック */
    VB*     syst_name;          /* SYST コマンド応答文字列 */
} T_FTP_SERVER ;
```

この構造体に必要な情報をセットして FTP サーバーAPI の引数として渡します。

- **セキュリティポリシー**

セキュリティポリシーにはセキュリティ設定を行います。「0」を設定した場合はセキュリティ機能を使用しません。「ENA_DENY_PORTCOMMAND」を設定した場合は PORT コマンドの拒否機能が適用されます。「ENA_NOTCON_WELL_KNOWNPORT」を設定した場合は Well-known ポートからの接続拒否が適用されます。

なお、PORT コマンドの拒否機能を適用した場合、必然的に Well-known ポートでの接続は行われません。従って、「ENA_DENY_PORTCOMMAND」を設定した場合は「ENA_NOTCON_WELL_KNOWNPORT」の機能も自動的に有効となります。また、SYST コマンドに応答させる場合、「ENA_ALLOW_SYSTCOMMAND」を設定します。

ENA_DENY_PORTCOMMAND
PORT コマンド機能を拒否

ENA_NOTCON_WELL_KNOWNPORT
Well-known ポート接続を拒否

ENA_ALLOW_SYSTCOMMAND
SYST コマンドの有効化。T_FTP_SERVER::syst_name の値を問い合わせ元に返却します。値未設定時は「UNIX Type: L8」を問い合わせ元に返却します。

- **デバイス番号**

デバイス番号には FTP サーバーで使用するネットワークデバイスを指定します。’0’（DEV_ANY）を指定した場合はデフォルトのネットワークデバイスが使用されます。（通常は 0 をセットしてください）

- **TCP ソケット**

FTP サーバーでは制御用とデータ用の二つの TCP ソケットが必要になります。TCP ソケットは次のようパラメータで作成する必要があります。（FTP サーバーアプリケーション内で生成されます）

制御用ソケット：

ID	プロトコル	ポート	タイムアウト				バッファサイズ	
			送信	受信	接続	切断	送信	受信
ID_SOC_FTP_CTL	TCP	21	5 秒	15 秒	-1	5 秒	1024	1024

データ用ソケット：

ID	プロトコル	ポート	タイムアウト				バッファサイズ	
			送信	受信	接続	切断	送信	受信
ID_SOC_FTP_DATA	TCP	20	5 秒	15 秒	5 秒	5 秒	1024	1024

- **認証コールバック**

任意のユーザ認証処理を行う場合、本変数にコールバック関数を指定してください。ユーザ名・パスワード入力後の認証時に呼ばれます。通常は 0 指定で問題ありません。

- コンフィグレーション定義（ftp_server_cfg.h）

設定例

```
#include "ffsys.h"       /* File system */

/* Configuration */
#define CFG_FTPS_DRV_NAME         'C'               /* Drive name */
#define CFG_FTPS_PATH_MAX         PATH_MAX          /* Maximum length of file path */
#define CFG_FTPS_CMD_TMO          5000              /* 5 sec */
#define CFG_FTPS_DAT_TMO          5000              /* 5 sec */
#define CFG_FTPS_IDLE_TMO         (5 * 60 * 1000)   /* 5 minute */
#define CFG_FTPS_SES_NUM          1                 /* Number of sessionss */
```

CFG_FTPS_DRV_NAME
ファイルシステムで使用するドライブ名（’A’や’C’など）を指定します。FTP サーバーはファイルを開く場合、本マクロのドライブ名を使用します。

CFG_FTPS_PATH_MAX
ファイルシステムのファイルパスの最大長を指定します。ここでのパス長はルートのディレクトリ名＋ファイル名の文字列の長さを表します。

CFG_FTPS_CMD_TMO
制御用ソケットの送信と切断のタイムアウト（msec）の値を指定します。

CFG_FTPS_DAT_TMO
データ用ソケットの接続、送信、受信、切断のタイムアウト（msec）の値を指定します。

CFG_FTPS_IDLE_TMO
制御用ソケットの受信のタイムアウト（msec）の値を指定します。

CFG_FTPS_SES_NUM
FTP サーバのタスク数（セッション数）を指定します。

- アカウント設定（ftp_server_cfg.c）

設定例

```
#include "kernel.h"
#include "net_hdr.h"
#include "ftp_server.h"

/* Login user table (Max. 256 users) (DEV_ANY: All device is allowed) */
const T_FTP_USR_TBL ftp_usr_tbl[] = {
    {DEV_ANY, "", ""},                  /* Anyone can login (No user name,password) */
    {DEV_ANY, "User", "Password"},

    {0x00, 0x00, 0x00}                  /* Terminate mark (Do not change) */
};
```

ファイルの ftp_server_cfg.c にはアカウントを設定する構造体の配列変数を宣言してください。アカウントを設定する構造体は次のようになります。

```
typedef struct t_ftp_usr_tbl {
    UH      dev_num;        /* デバイス番号*/
    VB*     usr;            /* ユーザー名 */
    VB*     pwd;            /* パスワード */
} T_FTP_USR_TBL ;
```

構造体の dev_num はアカウントの認証時に使用するネットワークデバイスの番号を指定します。0（DEV_ANY）を指定した場合は全てのネットワークデバイスで認証を行います。一般に dev_num には 0 を代入してください。usr はユーザー名を文字列で指定します。pwd はパスワードを文字列で指定します。この構造体の配列変数を ftp_usr_tbl の変数名で宣言してください。配列の最後の要素は終端用のデータです。終端用のデータはすべての変数に 0 を代入してください。ユーザー名とパスワードは最大で 256 個まで登録が可能です。なお、上記の設定例の「{DEV_ANY, "", ""}」はユーザー名とパスワードの入力を省いても FTP サーバーにログインできる設定です。よって、テストの目的以外では登録しないでください。

6.2.1 FTP サーバーAPI

ftp_server	FTPサーバーの開始

【書式】

ER ercd = ftp_server(T_FTP_SERVER *ftp);

【パラメータ】

T_FTP_SERVER	*ftp	FTPサーバー制御情報

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。（*ftp が NULL。 ctl_sidかdat_sidが指定されていない。）
E_RLWAI	FTP サーバーの停止要求が実行された。

【解説】

この API は FTP サーバーを初期化し FTP クライアントからの要求を受け付け処理します。この API はブロッキング呼出しになっていますので、専用タスクを用意しそこから呼びだすようしてください。

ftp_server_stop	FTPサーバーの停止

【書式】

ER ercd = ftp_server_stop(UW retry);

【パラメータ】

UW	retry	サーバータスク停止処理のリトライ回数

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_TMOUT	サーバータスク停止処理のリトライオーバーが発生した。

【解説】

この API は起動している FTP サーバーを停止します。この API が実行された場合、FTP サーバーの全タスクの停止処理を実行します。

FTP サーバーのタスク情報は FTP サーバー起動 API（ftp_server）実行時に自動で内部メモリに保持されます。

なお、この API は実行時に指定されたリトライ回数（retry）のリトライオーバーが発生した場合、戻り値として E_TMOUT が返却されます。

【推奨】

この API で指定するリトライ回数（retry）の値は「5」以上である事を推奨します。例えばリトライ回数に「0」を指定して実行した場合、この API が失敗する確率がより高くなります。

6.2.2 制限事項

- IPv6、SSL 上での動作は対応していません。（IPv4 のみ）
- ファイルを扱うにはファイルシステムが必要です。

6.3 HTTP サーバー

HTTP サーバーは、 HTTP クライアント（インターネットブラウザ）に静的または動的なコンテンツを送信します。

(1) HTTP コンテンツ情報

```
typedef struct t_http_file {
        const char      *path;                      /* URL */
        const char      *ctype;                     /* コンテンツタイプ */
        const char      *file;                      /* コンテンツ */
        Int             len;                        /* コンテンツサイズ*/
        void(*cbk)(T_HTTP_SERVER *http);            /* HTTP コールバック関数 or CGI ハンドラ */
        UB              ext;                        /* 拡張動作フラグ */
} T_HTTP_FILE ;
```

この構造体に HTTP サーバーで使用するコンテンツを登録します。

- URL

コンテンツの URL を表しています。例えば、クライアントからその URL に対して要求があった場合、対応するコンテンツがクライアントに送信されます。

URL に NULL を指定することはできません。また、URL は常に’/’から開始します。

- コンテンツタイプ

text/html 等の Content-Type を指定します。動的コンテンツの場合は NULL を指定します。

- コンテンツ

実際のコンテンツを指定します。動的コンテンツの場合は NULL を指定します。

- コンテンツサイズ

コンテンツのサイズを指定します。動的コンテンツの場合は 0 を指定します。

- コールバック関数または CGI ハンドラ

動的コンテンツの時に HTTP サーバーから呼び出される関数のポインタを指定します。静的コンテンツの場合には NULL を指定します。

- 拡張動作フラグ

コンテンツに対して拡張動作を指定します。（HTTP 認証を使用する、任意の HTTP ヘッダを付与するなど）拡張動作を使用しない場合 0 を指定します。

```
#define HTTPD_EXT_AUTH          0x01        /* HTTP 認証使用 */
#define HTTPD_EXT_UHDR          0x02        /* カスタムヘッダ—使用 */
```

T_HTTP_FILE::extに入る定義値

(2) HTTP サーバー制御情報

```
typedef struct t_http_server {
        UW                  sbufsz;             /* 送信バッファサイズ */
        UW                  rbufsz;             /* 受信バッファサイズ */
        UW                  txlen;              /* 内部データ */
        UW                  rxlen;              /* 内部データ */
        UW                  rdlen;              /* 内部データ */
        UW                  len;                /* 内部データ */
        UB                  *rbuf;              /* 送信バッファ*/
        UB                  *sbuf;              /* 受信バッファ*/
        UB                  *req;               /* 内部データ*/
        UH                  Port;               /* リスニングポート番号*/
        SID                 SocketID;           /* ソケット ID */
        T_HTTP_HEADER       hdr;                /* HTTP クライアントリクエスト */
        UB                  NetChannel;         /* デバイス番号 */
        UB                  ver;                /* IP バージョン*/
        UB                  server_tsk_stat     /* HTTP サーバータスク起動状態*/
        ID                  server_tsk_id       /* HTTP サーバータスク ID */
        struct t_http_server *next              /* HTTP サーバーオブジェクト */
        UH                  kpa_max             /* HTTP KeepAlive max 値（制御用） */
} T_HTTP_SERVER;
```

この構造体は HTTP サーバーAPI の引数として使用されます。ソケット ID はユーザアプリケーションでセットする必要があります。

- **デバイス番号**

デバイス番号には HTTP サーバーで使用するネットワークデバイスを指定します。’0’を指定した場合はデフォルトのネットワークデバイスが使用されます。（通常は 0 をセットしてください）

- **ソケット ID**

HTTP サーバーでは TCP ソケットを使用します。TCP ソケットは次のようパラメータで作成する必要があります。（HTTP サーバーアプリケーション内で生成されます）

ID	プロトコル	ポート	タイムアウト				バッファサイズ	
			送信	受信	接続	切断	送信	受信
ID_SOC_HTTP	TCP	80	25 秒	25 秒	25 秒	25 秒	1024	1024

- **受信バッファ・送信バッファ**

HTTP サーバーではパケットの送受信ごとにプロトコルスタックのネットワークバッファを使用します。

コンテンツサイズなどの理由（例えばネットワークバッファより大きなコンテンツを送受信したいなど）で、これにアプリケーション独自のバッファを利用したい場合は、受信（送信）バッファおよび、受信（送信）バッファサイズに独自バッファの値を設定します。その場合 HTTP サーバーでネットワークバッファを取得することはありません。
また独自設定した領域は他の HTTP サーバープロセスと共有することはできません。

(3) HTTP ヘッダ情報

```
typedef struct t_http_header {
        char        *method;        /* メソッド */
        char        *url;           /* パス名 */
        char        *url_q;         /* URL クエリ*/
        char        *ver;           /* バージョン */
        char        *host;          /* ホスト名 */
        char        *ctype;         /* コンテンツタイプ */
        char        *Content;       /* コンテンツ */
        char        ContentLen;     /* コンテンツ長 */
        char        kpa;            /* HTTP Keep Alive 用フラグ（制御用） */
        char        *auth;          /* Authorization ヘッダ */
        char        *cookie;        /* Cookie ヘッダ */
}T_HTTP_HEADER;
```

この構造体は T_HTTP_SERVER::rbuf のバッファを HTTP のリクエストメッセージの各要素として表します。この構造体は HTTP コールバック関数からのみ参照してください。

- メソッド [method]

メソッドを表します。”GET”、”HEAD”、”POST”のいずれかが入ります。

- パス名 [URL]

URL のパス名を表します。

- URL クエリ

URL のクエリパラメータを表します。

- バージョン [version]

HTTP バージョンを表します。”HTTP/1.1”、”HTTP/1.0”のいずれかが入ります。

- コンフィグレーション定義（http_server_cfg.h）

設定例
#define ENA_KEEP_ALIVE /* Enable HTTP Keep-Alive */
#define HTTP_KPA_MAX 100 /* Keep-Alive max value */
#define ENA_USER_EXT /* Enable User Extension */

ENA_KEEP_ALIVE
定義ありで HTTP Keep-Alive 機能を有効にします。定義なしで無効にします。

HTTP_KPA_MAX
HTTP Keep-Alive ヘッダの max 項目の初期値を指定します。

ENA_USER_EXT
定義ありで HTTP サーバアプリのユーザ拡張動作機能を有効にします。定義なしで無効にします。

6.3.1 HTTP サーバーAPI

http_server	HTTPサーバーの開始

【書式】

ER ercd = http_server(T_HTTP_SERVER *http);

【パラメータ】

T_HTTP_SERVER	*http	HTTPサーバー制御情報

【戻り値】

ER	ercd	次のエラーコード

【エラーコード】

E_RLWAI	http_server_stop()がコールされ停止した。
E_PAR	不正なパラメータが指定された。 （*http が NULL。SocketID が指定されていない。）
E_NOMEM	メモリ不足（ネットワークバッファの枯渇）
その他	ソケットの con_soc のエラーコード

【解説】

この API は HTTP セッションを初期化し、HTTP クライアントからの要求を受け付け処理します。クライアントから要求された URL がコンテンツテーブル（T_HTTP_FILE）に存在する時は、そのコンテンツをクライアントに送信し、存在しない時は HTTP エラーメッセージ”404 File not found”を送信します。コンテンツが動的の場合（cbk が NULL でない）は、そのコールバック関数を呼び出します。

この API はブロッキング呼出しになっていますので、専用タスクを用意しそこから呼びだすようしてください。

引数の制御情報の受信バッファが NULL の場合、HTTP サーバーはネットワークバッファを使用します。

引数の制御情報の送信バッファが NULL の場合、HTTP サーバーはネットワークバッファを使用します。

http_server_stop	HTTPサーバーの停止

【書式】

ER ercd = http_server_stop(UW retry)

【パラメータ】

UW	retry	サーバータスク停止処理のリトライ回数

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_TMOUT	サーバータスク停止処理のリトライオーバーが発生した。

【解説】

この API は起動している HTTP サーバーを停止します。この API が実行された場合、HTTP サーバーの全タスクの停止処理を実行します。

HTTP サーバーのタスク情報は HTTP サーバー起動 API（http_server）実行時に自動で内部メモリに保持されます。

なお、この API は実行時に指定されたリトライ回数（retry）のリトライオーバーが発生した場合、戻り値として E_TMOUT が返却されます。

【推奨】

この API で指定するリトライ回数（retry）の値は「5」以上である事を推奨します。例えばリトライ回数に「0」を指定して実行した場合、この API が失敗する確率がより高くなります。

CgiGetParam	CGI引数の解析

【書式】

```
void CgiGetParam(char *msg, int clen, char *cgi_var[], char *cgi_val[], int *cgi_cnt);
```

【パラメータ】

char	*msg	CGI引数
int	clen	CGI 引数サイズ
char	*cgi_var[]	解析したCGI引数
char	*cgi_val[]	解析したCGI引数の値
int	*cgi_cnt	解析したCGI引数の個数

【戻り値】

なし

【エラーコード】

なし

【解説】

この API は‘フィールドー値’の組で構成されるクエリ文字列を解析します。例えばクエリ文字列が“name1=value1&name2=value2”と与えられ場合の解析結果は以下のようになります。

```
cgi_cnt = 2;

cgi_var[0] = "name1";
cgi_var[1] = "name2";

cgi_val[0] = "value1";
cgi_val[1] = "value2";
```

【補足】

この API は CgiGetParamN 関数にとって代わられました。本 API は引数 msg のクエリ文字列を解析して得られた要素の数が cgi_var と cgi_val の配列数を超えた場合、バッファオーバーランを引き起こす為、特別な理由がない限り CgiGetParamN 関数を使用する様にしてください。

CgiGetParamN 関数は今後廃止される予定です。

CgiGetParamN	CGI引数の解析

【書式】

```
void CgiGetParamN(char *msg, int clen, char *cgi_var[], char *cgi_val[], int *cgi_cnt);
```

【パラメータ】

char	*msg	CGI 引数
int	clen	CGI 引数サイズ
char	*cgi_var[]	解析したCGI引数
char	*cgi_val[]	解析したCGI引数の値
int	*cgi_cnt	IN：cgi_var 配列の要素数 OUT：解析したCGI引数の個数

【戻り値】

なし

【エラーコード】

なし

【解説】

この API は‘フィールドー値’の組で構成されるクエリ文字列を解析します。例えばクエリ文字列が“name1=value1&name2=value2”と与えられ場合の解析結果は以下のようになります。

```
char msg[] = "name1=value1&name2=value2";
char *cgi_var[10];
char *cgi_val[10];
int cgi_cnt;
cgi_cnt = sizeof(cgi_var) / sizeof(char *);
CgiGetParamN( msg, strlen(msg), cgi_var, cgi_val, &cgi_cnt );

cgi_cnt = 2;
cgi_var[0] = "name1";
cgi_var[1] = "name2";
cgi_val[0] = "value1";
cgi_val[1] = "value2";
```

【補足】

類似 API である CgiGetParam 関数は互換性の為に残されています。CgiGetParam 関数の使用は不具合の原因となり得るので、特別な理由がない限り CgiGetParamN 関数を使用してください。

CookieGetItem	Cookieヘッダの解析

【書式】

UB CookieGetItem(char **cookie, char **name, char **value)

【パラメータ】

char	**cookie	Cookieヘッダの値
char	**name	取得したCookie名称
char	**value	取得したCookie名称の値

【戻り値】

UB	Cookieペアが取得出来た場合1、それ以外は0

【エラーコード】

なし

【解説】

この API は Cookie ヘッダの値‘name1=value1; name2=value2; ..’形式から Cookie 名称と値を取得します。Cookie ペアが取得できると名称は name、値は value に格納され戻り値に 1 が返ります。それ以外の場合、戻り値に 0 が返ります。

この API をコールすると、**cookie の指すポインタ位置が次の Cookie ペアの位置になり、且つバッファ内容が変更されます。**cookie の指すバッファ内容が必要な場合はこの API をコール前に予め別のバッファなどに退避させてください。

使用例

```
void Http_Callback(T_HTTP_SERVER *http)
{
    VB *cname, *cval;
    VB buf[128];

    net_strcpy(buf, "<html><body>");

    /* Cookie ヘッダの内容を HTML に出力*/
    net_strcat(buf, "¥r¥n<pre>¥r¥n");
    while (CookieGetItem(&http->hdr.cookie, &cname, &cval)) {
        net_strcat(buf, cname);   /* Cookie 名称*/
        net_strcat(buf, "=");
        net_strcat(buf, cval);    /* Cookie 名称の値*/
        net_strcat(buf, "¥r¥n");
    }
    net_strcat(buf, "¥r¥n</pre>¥r¥n");
    net_strcat(buf, "</body></html>");
    HttpSendText(http, buf, net_strlen(buf));
}
```

HttpSendText	テキストコンテンツの送信

【書式】

ER ercd = HttpSendText(T_HTTP_SERVER *http, const char *str, UW len)

【パラメータ】

T_HTTP_SERVER	*http	HTTP サーバー制御情報
const char	*str	送信する文字列
UW	len	送信する文字列長

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。（*httpがNULL）

【解説】

この API は動的コンテンツを送信します。この API は CGI コールバック関数からのみ呼び出してください。

例

```
char page1[] = "<html><body> Welcome to this web server </body></html>";

void Http_Callback(T_HTTP_SERVER *http)
{
   HttpSendText(http, page1, sizeof(page1));
}
```

HttpSendFile　　ファイルの添付送信

【書式】

```
ER ercd = HttpSendFile(T_HTTP_SERVER *http, const char *str,
                       UW len, const char *name, const char *type)
```

【パラメータ】

T_HTTP_SERVER	*http	HTTP サーバー制御情報
const char	*str	送信するファイルの実体
UW	len	送信するファイルのバイトサイズ
const char	*name	送信するファイル名
const char	*type	HTTPヘッダーのContent-Typeの値(文字列)

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。 （*httpがNULL）

【解説】

この API は動的コンテンツを送信します。この API は HTTP コールバック関数からのみ呼び出してください。

この API ではファイルは添付送信（Content-Disposition: attachment）で送信されます。

例

```
char file[1024];

void Http_Callback(T_HTTP_SERVER *http)
{
int len;
                    :
/* ファイルの内容を file に，サイズを len に出力する処理 */
                    :
   HttpSendFile(http, file, len, “FILE NAME”, “text/plain”);
}
```

HttpSendResponse	指定コンテンツの送信

【書式】

```
ER ercd = HttpSendResponse(T_HTTP_SERVER *http, const char *str,
                           UW len, const char *type)
```

【パラメータ】

T_HTTP_SERVER	*http	HTTP サーバー制御情報
const char	*str	送信するコンテンツバイト列
UW	len	送信するコンテンツバイト列長
const char	*type	コンテンツタイプ名称

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。（*httpがNULL）

【解説】

この API は動的コンテンツを送信します。この API は CGI コールバック関数からのみ呼び出してください。

http_server_extcbk 拡張動作用コールバック関数

【書式】

ER ercd = http_server_extcbk(T_HTTP_SERVER *http, const T_HTTP_FILE *fp, UB evt)

【パラメータ】

T_HTTP_SERVER	*http	HTTP サーバー制御情報
T_HTTP_FILE	*fp	アクセスされたコンテンツ情報
UB	evt	発生イベントの判定フラグ

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【解説】

このコールバック関数（拡張動作用コールバック）は、HTTP サーバタスクから呼び出されます。拡張動作フラグが設定されているコンテンツに対して、Web ブラウザからアクセスを行うと拡張動作用コールバック関数が呼ばれます。アクセスされた該当コンテンツ情報は fp に入ります。また、判定フラグ evt には以下の値が入ります。

- HTTPD_EXT_AUTH（HTTP 認証の使用）

該当コンテンツに HTTP 認証が適用されます。コールバック関数はクライアント要求にある Authorization ヘッダの値が http->hdr.auth に入った状態、もしくは http->hdr.auth が NULL の状態で呼ばれます。

http->hdr.auth に値が入っている場合は、関数内で値のユーザ名・パスワード（or ダイジェスト値）が正しいかのチェックを行います。結果、成功すれば E_OK を、失敗では E_OK 以外を返すように設計してください。

http->hdr.が NULL の場合は、関数内で認証失敗の応答に表記する WWW-
Authenticate ヘッダ情報の作成（realm や nonce の指定など）を行ってください。

- HTTPD_EXT_UHDR（カスタムヘッダの指定）

Web ブラウザにコンテンツ情報を応答する前にコールバック関数が呼ばれ、関数内で任意の HTTP ヘッダを追記できます。CGI には適用されません。

（このフラグを CGI に適用してもカスタムヘッダは付与されません）

コールバック関数内で HttpSetContent()以外の HTTP サーバ API、待ち要因がある API の呼び出しは禁止です。コールバック関数は HTTP サーバタスクのスタックを使用します。

必要に応じて HTTP サーバタスクのスタックサイズを増やしてください。サーバタスクが複数ある場合はコールバック関数内の処理をリエントラントな作りにする必要があります。

以下に使用例、および解説を記します。

使用例

【net_cfg.c - コンテンツ登録部】

```
/*******************************
      HTTP Content List
 *******************************/
T_HTTP_FILE const content_list[] =
{
      {"/", "text/html", index_html1, sizeof(index_html1), NULL, 0},
      {"/manage/", "text/html", manage_html2, sizeof(manage_html2), NULL, HTTPD_EXT_AUTH | HTTPD_EXT_UHDR},
      {"/sample.cgi", "", NULL, 0, sample_fnc, HTTPD_EXT_AUTH},
      {"", NULL, NULL, 0, NULL}
};
```

【net_cfg.c － net_setup() HTTP サーバタスク起動部】

```
      {前略}

      gHTTP_EXT_CBK = httpd_evt_callback;      /* 拡張動作コールバック登録 */
      gHTTP_FILE = (T_HTTP_FILE*)content_list;

      /* Start HTTPd Task */
      sta_tsk(ID_HTTPD_TSK1, 0);

      {後略}
```

【ユーザソース】

```
ER httpd_evt_callback(T_HTTP_SERVER *http, const T_HTTP_FILE *fp, UB evt)
{
      T_HTTP_HEADER *hdr = &http->hdr;
      ER ercd;
      ercd = E_OK;

      switch (evt) {
      case HTTPD_EXT_AUTH:      /* HTTP 認証(ベーシック) */
            if (hdr->auth) {      /* 認証情報あり */
                  {省略}      /* 値は Base64 のため、デコード処理を行う */
                  {省略}      /* デコード後、ユーザ名・パスワードを取得する */
                  ercd = (OK == 妥当性チェック) ? E_OK : E_OBJ;
            }
            else {                  /* 認証失敗 */
                  /* 応答メッセージ用の WWW-Authenticate ヘッダ作成 */
                  HttpSetContent(http, "WWW-Authenticate: Basic ");
                  HttpSetContent(http, "realm=¥"Secret Zone¥"¥r¥n");
            }
            break;
      case HTTPD_EXT_UHDR:      /* カスタムヘッダ指定 */
            HttpSetContent(http, "X-Frame-Options: DENY¥r¥n");
            break;
      }
      return ercd;
}
```

【解説】

拡張動作用コールバック関数は次の条件を満たした場合に呼ばれます。

・http_server_set_extcbk()で処理コールバック関数を登録している。
・コンテンツリストの ext メンバに値を設定している。
・上記該当コンテンツに Web ブラウザからアクセスする。

例では、http://xxx.xxx.xxx.xxx/ にアクセスした場合にはコールバック関数は呼ばれず、それ以外の URL にアクセスした場合はコールバック関数が呼ばれます。

次にコールバック関数内の処理を見ていきます。はじめに、コールバック関数ではフラグ evt の値を見てどの拡張動作（認証処理、カスタムヘッダ処理）なのかを判定します。

フラグ evt が HTTP 認証処理（HTTPD_EXT_AUTH）の場合、hdr->auth の値が NULL 以外で認証チェックの処理を行います。hdr->auth にブラウザから取得した Authorization ヘッダ値が入ります。値をデコード・解析してユーザ名・パスワード、ダイジェスト値の妥当性チェックを行います。（BASE64、MD5 処理はユーザで対応が必要です）成功時は E_OK、それ以外は E_OK 以外を戻り値に指定するようにします。
認証が失敗時は hdr->auth の値が NULL でコールバックが呼ばれます。この場合は、WWW-Authenticate ヘッダ情報を指定するようにします。（～改行コードまで）

フラグ evt がカスタムヘッダ処理（HTTPD_EXT_UHDR）の場合、HTTP サーバタスクが用意する HTTP ヘッダ末尾に任意のヘッダ情報を付与できます。改行コードで終了するヘッダ情報を記述してください。

ファイル名やコンテンツタイプの判定処理が必要な場合、fp を参照してください。

HttpSetContent	HTTPサーバ制御情報の送信バッファ追記

【書式】

ER ercd = HttpSetContent(T_HTTP_SERVER *http, const char *str);

【パラメータ】

T_HTTP_SERVER	*http	HTTP サーバー制御情報
const char	*str	追記する文字列、NULL指定でバッファクリア

【戻り値】

ER	ercd	追記バイト数（≧0）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。 （*httpがNULL）

【解説】

この API は HTTP サーバ制御情報の送信バッファに文字列を追記します。この API は拡張動作用コールバック関数・CGI 用コールバック関数からのみ呼び出してください。

拡張動作用コールバック関数内で使用する場合、str の引数に NULL は指定しないでください。HTTP サーバーアプリが途中作成した HTTP ヘッダの内容がクリアされ、動作が不定になります。

HttpSetContentKpa	送信バッファ にHTTP Keep-Aliveヘッダ付与

【書式】

ER HttpSetContentKpa(T_HTTP_SERVER *http);

【パラメータ】

T_HTTP_SERVER	*http	HTTP サーバー制御情報

【戻り値】

ER	ercd	追記バイト数（≧0）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。 （*httpがNULL）

【解説】

この API は HTTP サーバ制御情報の送信バッファに HTTP Keep-Alive のヘッダ情報を付与します。この API は CGI 用コールバック関数からのみ呼び出してください。

HttpSetContentCookie 送信バッファ にSet-Cookieヘッダ付与

【書式】

ER HttpSetContentCookie(T_HTTP_SERVER *http,
const char *name, const char *val, const char *opt)

【パラメータ】

T_HTTP_SERVER	*http	HTTP サーバー制御情報
const char	*name	Cookie名称
const char	*val	Cookie名称の値
const char	*opt	属性指定用バッファ（オプション）

【戻り値】

ER	ercd	追記バイト数（≧0）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。 （*http, *name, *val がNULL）

【解説】

この API は HTTP サーバ制御情報の送信バッファに Set-Cookie ヘッダ情報を付与します。この API は拡張動作用コールバック関数・CGI 用コールバック関数からのみ呼び出してください。

Cookie の属性指定が必要な場合、引数の opt に属性を文字列形式で指定してください。opt に指定する文字列は {属性名 1}={値 1} の形式で指定してください、複数の属性を指定する場合はセミコロン（;）で属性ペアを結合した文字列にします。

HttpSendBuffer	指定バッファ内容の送信

【書式】

ER ercd = HttpSendBuffer(T_HTTP_SERVER *http, const char *str, UW len);

【パラメータ】

T_HTTP_SERVER	*http	HTTP サーバー制御情報
const char	*str	送信するバッファ
UW	len	送信するバッファ長

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。 （*httpがNULL）

【解説】

この API は HttpSetContent()で作成した HTTP サーバ制御情報の送信バッファの送信、および指定バッファの送信をします。引数 str に NULL を指定すると HTTP サーバの送信バッファに残ったデータのみ送信します。この API は CGI 用コールバック関数からのみ呼び出してください。以下に使用例を記します。

使用例

```
ER HttpSendResponse(T_HTTP_SERVER *http, char *str, int len, char *type)
{
    /* メッセージヘッダ部の作成 */
    HttpSetContent(http, 0);
    HttpSetContent(http, "HTTP/1.1 200 OK¥r¥n");
    HttpSetContent(http, "Content-Type: ");
    HttpSetContent(http, type);
    HttpSetContent(http, "¥r¥n");
    HttpSetContentLen(http, "Content-Length: ", len);
    HttpSetContentKpa(http);
    HttpSetContent(http, "¥r¥n");

    /* メッセージヘッダ部と、指定バッファ（ボディ部）の送信 */
    HttpSendBuffer(http, str, len);

    return E_OK;
}

/* HTTP サーバ CGI 用コールバック関数（サンプル） */
void user_cgi_callback (T_HTTP_SERVER *http)
{
    VB contents[128];
    net_strcpy(contents, "<html><body>¥r¥n");
    net_strcat(contents, "<center>uNet CGI demo</center>¥r¥n");
    net_strcat(contents, "</body></html>");
    HttpSendResponse(http, contents, net_strlen(contents), "text/html");
}
```

6.3.2 HTTP サーバーサンプル

```
/* コンテンツの定義 */
const char index_html[] =
"<html>¥
<title> R-IN32 HTTP Server </title>¥
<body>¥
<h1>Hello World!</h1>¥
</body>¥
</html>";

/* コンテンツリストの初期化 */
T_HTTP_FILE const content_list[] =
{
    {"/", "text/html", index_html, sizeof(index_html), NULL},
    {"", NULL, NULL, 0, NULL} /* 終端 */
};

/* HTTP セッションの開始 */
static T_HTTP_SERVER http_server1;
void httpd_tsk1(VP_INT exinf)
{
/* Initialize the content list global pointer */
gHTTP_FILE = (T_HTTP_FILE*)content_list;

    memset((char* )&http_server1, 0, sizeof(http_server1));
    http_server1.SocketID = ID_SOC_HTTP1;

    http_server(&http_server1);
}
```

6.4 DNS クライアント

DNS クライアントでは UDP ソケットを使用します。UDP ソケットは次のようパラメータで作成する必要があります。

ID	プロトコル	ポート	タイムアウト	
			送信	受信
ID_SOC_DNS	UDP	0	5 秒	5 秒

(1) DNS クライアント情報

```
typedef struct t_dns_client {
        UW          ipa;            /* DNS サーバ IP アドレス */
        char        *name;          /* ホスト名(設定・参照用) */
        UW          *ipaddr;        /* IP アドレス(設定・参照用) */
        SID         sid;            /* DNS ソケット ID(UDP) */
        UH          code:           /* 要求 RR タイプ */
        UB          dev_num;        /* デバイス番号 */
        UB          retry_cnt;      /* リトライ回数 */
} T_DNS_CLIENT ;
```

この構造体は dns_query_ext() API の引数として使用します。ipa,name,ipaddr,sid は必ず指定して下さい。未指定の場合、E_PAR を返却します。code で要求タイプを指定できます。現時点では下記の定義値のみ指定出来ます。それ以外は E_NOSPT を返却します。

デバイス番号(dev_num)、リトライ回数(retry_cnt)は必要に応じて指定出来ます。リトライ回数は未指定時(0)にはリトライしません、リトライは、受信タイムアウト発生時に再送処理を行います。

- DNS クライアント RR 定義値

```
#define RR_TYPE_A       1U      IP アドレスを取得する
#define RR_TYPE_PTR     12U     ホスト名を取得する
#define RR_TYPE_AAAA    28U     IPv6 アドレスを取得する
                                （別途、eForce 社の TCP/IP スタック μNet3/IPv6 が必要となります）
```

T_DNS_CLIENT::code に入る定義値

6.4.1 DNS クライアント API

dns_get_ipaddr	IPアドレスの取得

【書式】

ER ercd = dns_get_ipaddr(SID socid, UW dns_server, char *name, UW *ipaddr);

【パラメータ】

SID	socid	UDPソケットID
UW	dns_server	DNSサーバーのIPアドレス
char	*name	ホスト名
UW	*ipaddr	取得するIPアドレス

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。
E_TMOUT	DNSサーバーから応答なし
E_NOMEM	メモリ不足（ネットワークバッファの枯渇）
E_OBJ	ホスト名からIPアドレス解決できない

使用例

```
UW ip;
ER ercd;
UW dns_server = ip_aton(“192.168.11.1”);

dns_get_ipaddr(ID_SOC_DNS, dns_server, “TEST Server URL”, &ip);
```

dns_get_name ホスト名の取得

【書式】

ER ercd = dns_get_name(SID socid, UW dns_server, char *name, UW *ipaddr);

【パラメータ】

SID	socid	UDPソケットID
UW	dns_server	DNSサーバーのIPアドレス
char	*name	取得するホスト名
UW	*ipaddr	IPアドレス

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。
E_TMOUT	DNSサーバーから応答なし
E_NOMEM	メモリ不足（ネットワークバッファの枯渇）
E_OBJ	IPアドレスからホスト名を取得できない

使用例

```
UW ip = ip_aton(“192.168.11.30”);
ER ercd;
char host_name[256];
UW dns_server = ip_aton(“192.168.11.1”);

dns_get_name(ID_SOC_DNS, dns_server, host_name, &ip);
```

dns_query_ext	DNSクエリの発行

【書式】

ER ercd = dns_query_ext(T_DNS_CLIENT *dc);

【パラメータ】

T_DNS_CLIENT	*dc	DNSクライアント情報

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。
E_TMOUT	DNSサーバーから応答なし
E_NOMEM	メモリ不足（ネットワークバッファの枯渇）
E_OBJ	IPアドレスからホスト名を取得できない。
E_NOSPT	サポート外のRRが指定された。

【解説】

dns_get_ipaddr()、dns_get_name()の機能拡張版 API です。左記 API に比べて、新たに送信するデバイス番号(dc->dev_num)の指定、および失敗時のリトライ(dc->retry_cnt)が可能となりました。

使用例

```
ER ercd;
T_DNS_CLIENT dc = {0};
char host_name[256];
UW dns_server = ip_aton("192.168.11.1");

dc.code      = RR_TYPE_PTR;
dc.name      = name;
dc.ipaddr    = ip_aton("192.168.11.30");
dc.ipa       = dns_server;
dc.sid       = ID_SOC_DNS;
//dc.dev_num   = 0;
//dc.retry_cnt = 1;

ercd = dns_query_ext(&dc);
```

6.5 DHCP クライアント拡張版

DHCP クライアント拡張版は、既存の DHCP クライアントに対し、IP などのリソースに関するリース状態を保持して、これらの延長（RENEW）、解放（RELEASE）、拒否（DECLINE）、再起動（REBOOT）、また情報取得（INFORM）機能を提供できるように拡張されています。

(1) DHCP クライアント情報

```
typedef struct t_dhcp_client {
        T_DHCP_CTL      ctl             /* 制御情報 */
        UW              ipaddr;         /* IP アドレス */
        UW              subnet;         /* サブネットマスク */
        UW              gateway;        /* ゲートウェイ */
        UW              dhcp;           /* DHCP サーバーアドレス */
        UW              dns[2];         /* DNS アドレス */
        UW              lease;          /* DHCP アドレスのリース期間 */
        UW              t1;             /* DHCP アドレスのリニューアル期間*/
        UW              t2;             /* DHCP アドレスのリバインド期間 */
        UB              mac[6];         /* MAC アドレス */
        UH              dev_num;        /* デバイス番号 */
        UB              state           /* DHCP クライアント状態 */
        SID             socid;          /* UDP ソケット ID */
        UB              arpchk;         /* IP 重複チェック有無 */
        T_DHCP_UOPT     *uopt;          /* DHCP オプション取得パラメータ */
        UB              uopt_len;       /* DHCP オプション取得パラメータの数 */
        UB              retry_cnt;      /* リトライ回数 */
} T_DHCP_CLIENT ;
```

この構造体は DHCP クライアント API の引数として使用するもので、ホストアドレス情報構造体を拡張したものです。通常版同様、デバイス番号と UDP ソケット ID はユーザアプリケーションでセットする必要があります。設定する値は DHCP クライアントを参照して下さい。

IP 重複チェック有無に「ARP_CHECK_ON」を設定した場合、DHCP サーバーからリースされた IP に対し ACD 機能を使用した重複チェックを行います。

DHCP サーバーから任意のオプション情報(※)を取得したい場合、DHCP オプション取得パラメータを指定します。値を指定すると、DHCP DISCOVER/REQUEST/INFORM メッセージの Parameter Request List に指定オプションコードが追加され、応答メッセージから該当オプションの取得を試みます。特に不要な場合は 0 を指定してください。

※取得可能な値形式は バイナリ(1,2,4 バイト)、文字列、アドレス のいずれかです。

応答がなかった場合に要求のリトライを行うことが可能です。リトライ回数(retry_cnt)が未指定時(0)にはリトライは 3 回行います。必要に応じて指定してください。

この構造体は IP アドレス取得時に設定したものを IP アドレス更新時でも使用します。そのためアプリケーションで制御情報や DHCP クライアント状態を変更することはできません。

- DHCP クライアントステータス定義値

```
#define DHCP_STS_INIT            0
#define DHCP_STS_INITREBOOT      1
#define DHCP_STS_REBOOTING       2
#define DHCP_STS_REQUESTING      3
#define DHCP_STS_BOUND           4
#define DHCP_STS_SELECTING       5
#define DHCP_STS_REBINDING       6
#define DHCP_STS_RENEWING        7
```

T_DHCP_CLIENT::state に入る定義値

(2) DHCP 取得オプション情報

```
typedef struct t_dhcp_uopt {
        UB      code;       /* DHCP オプションコード */
        UB      len;        /* オプション要素のサイズ／個 */
        UB      ary;        /* オプション要素の数 */
        UB      flag;       /* 要素判別・状態フラグ */
        VP      val;        /* 要素格納先ポインタ */
} T_DHCP_UOPT;
```

この構造体は DHCP サーバーから任意のオプション情報を取得するために使用します。code に取得したいオプションのコード番号、len に対象オプションの値 1 つあたりのサイズ、ary に対象オプションの要素数、flag に後述するフラグ値の設定、val に値を実際に格納する先の変数・配列をそれぞれ指定します。以下にオプションの設定例を示します。

	Time Offset(2)	Host Name(12)	Log Server(7)
code;	2	12	7
len;	4	1	4
ary;	1	バッファのサイズ	1 … n (複数取得時)
flag;	DHCP_UOPT_BIN	DHCP_UOPT_STR	DHCP_UOPT_IPA
val;	INT 型変数のポインタ	バッファのポインタ	UW 型変数・配列のポインタ

T_DHCP_CLIENT 構造体の DHCP オプション取得パラメータにこの構造体の値を指定した後、dhcp_bind(), dhcp_inform() のいずれかをコールします。応答メッセージに対象オプションが含まれている場合に val のポインタ先に取得値が入ります。値が取得できた場合は flag に DHCP_UOPT_STS_SET ビット値が立ちます。

次に T_DHCP_UOPT 構造体の実際の使用方法について記述する。

6.5.1 DHCP クライアント拡張版 API

dhcp_bind	DHCPリース情報の取得

【書式】

```
ER ercd = dhcp_bind(T_DHCP_CLIENT *dhcp);
```

【パラメータ】

T_DHCP_CLIENT	*dhcp	DHCPクライアント情報

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	* dhcpがNULLまたはsocidが指定されていない
E_NOMEM	メモリ不足（ネットワークバッファの枯渇）
E_OBJ	ソケットの状態が不正（ソケットが未作成）
E_SYS	割り当てられたIPアドレスが他のホストと競合。
E_TMOUT	DHCPサーバーからの応答が遅延。またはDHCPサーバーが存在しない。

【解説】

この API は従来の dhcp_client( ) API と同等の機能を提供します。

取得した IP アドレスが他のホストと重複していないかを検証するには、引数の DHCP クライアント情報の IP 重複チェック有無に ARP_CHECK_ON を設定します。このとき IP アドレスの重複が検出された場合、DHCP サーバーに DHCP_DECLINE メッセージを送信し、API は E_SYS を返却します。

dhcp_renew　　DHCPリース情報の有効期間延長

【書式】

```
ER ercd = dhcp_renew(T_DHCP_CLIENT *dhcp);
```

【パラメータ】

T_DHCP_CLIENT	*dhcp	DHCPクライアント情報

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	* dhcpがNULLまたはsocidが指定されていない
E_NOMEM	メモリ不足（ネットワークバッファの枯渇）
E_OBJ	不正なDHCPクライアント情報。またはDHCPサーバーが要求を拒否した。
E_SYS	割り当てられたIPアドレスが他のホストと競合。
E_TMOUT	DHCPサーバーからの応答が遅延。またはDHCPサーバーが存在しない。

【解説】

この API は DHCP サーバーから取得した IP アドレスの有効期間を延長します。引数には dhcp_bind()で取得した DHCP クライアント情報を指定します。

通常この API は T1 時間経過時に呼び出して下さい。(T1 時間経過前でも問題はありません) 有効期間はタイマやタスク制御を使用してアプリケーションで計測します。

延長した IP アドレスが他のホストと重複していないかを検証するには、引数の DHCP クライアント情報の IP 重複チェック有無に ARP_CHECK_ON を設定します。このとき IP アドレスの重複が検出された場合、DHCP サーバーに DHCP_DECLINE メッセージを送信し、API は E_SYS を返却します。

RENEW 機能のみ行われます。R-IN32M3 TCP/IP スタックと動作を同じにしたい場合、本 API のエラーを確認した後、dhcp_rebind()をコールするようにしてください。

dhcp_rebind　　リース情報の再割り当て申請

【書式】

```
ER ercd = dhcp_rebind(T_DHCP_CLIENT *dhcp);
```

【パラメータ】

T_DHCP_CLIENT	*dhcp	DHCPクライアント情報

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	* dhcp が NULL または socid が指定されていない
E_NOMEM	メモリ不足（ネットワークバッファの枯渇）
E_OBJ	不正なDHCPクライアント情報。またはDHCPサーバーが要求を拒否した。
E_SYS	割り当てられたIPアドレスが他のホストと競合。
E_TMOUT	DHCPサーバーからの応答が遅延。またはDHCPサーバーが存在しない。

【解説】

この API は DHCP サーバーから取得した IP アドレスの再割り当てを申請します。引数には dhcp_bind()で取得した DHCP クライアント情報を指定します。

通常この API は T2 時間経過後に呼び出して下さい。有効期間はタイマやタスク制御を使用してアプリケーションで計測します。

延長した IP アドレスが他のホストと重複していないかを検証するには、引数の DHCP クライアント情報の IP 重複チェック有無に ARP_CHECK_ON を設定します。このとき IP アドレスの重複が検出された場合、DHCP サーバーに DHCP_DECLINE メッセージを送信し、API は E_SYS を返却します。

dhcp_reboot	DHCPクライアントの再起動

【書式】

ER ercd = dhcp_reboot(T_DHCP_CLIENT *dhcp);

【パラメータ】

T_DHCP_CLIENT	*dhcp	DHCPクライアント情報

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	* dhcp が NULL または socid が指定されていない または再利用するアドレスが無い。
E_NOMEM	メモリ不足（ネットワークバッファの枯渇）
E_OBJ	不正なDHCPクライアント情報。またはDHCPサーバーが要求を拒否した。
E_SYS	割り当てられたIPアドレスが他のホストと競合。
E_TMOUT	DHCPサーバーからの応答が遅延。またはDHCPサーバーが存在しない。

【解説】

この API はクライアントが以前使用していた IP リソースを再び使用する場合、その正当性を DHCP サーバーに確認するために使用します。たとえば休止中の LAN インタフェースが再び活性化した場合や LAN ケーブルを抜き挿しした時など、その前後で同じネットワークに参加している保障が無い場合に、それまで使用していた IP リソースを DHCP サーバーに告知します。

引数には dhcp_bind()で取得した DHCP クライアント情報を指定します。

この API は REQUEST メッセージ送信後 ACK を受信できなかった場合、もしくは DHCPNAK を受信した場合はエラーとします。

告知した IP アドレスが他のホストと重複していないかを検証するには、引数の DHCP クライアント情報の IP 重複チェック有無に ARP_CHECK_ON を設定します。このとき IP アドレスの重複が検出された場合、DHCP サーバーに DHCP_DECLINE メッセージを送信し、API は E_SYS を返却します。

dhcp_release　　DHCPリース情報の解放

【書式】

ER ercd = dhcp_release(T_DHCP_CLIENT *dhcp);

【パラメータ】

T_DHCP_CLIENT	*dhcp	DHCPクライアント情報

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	* dhcp が NULL または socid が指定されていない
E_NOMEM	メモリ不足（ネットワークバッファの枯渇）
E_OBJ	不正なDHCPクライアント情報。
E_TMOUT	DHCPRELEASEメッセージ送信タイムアウト。

【解説】

この API は DHCP サーバーから取得した IP アドレスを使用しなくなった場合に、DHCP サーバーにリソースの解放を通知します。

引数には dhcp_bind()で取得した DHCP 情報を指定します。

dhcp_inform　　DHCPオプションの取得

【書式】

ER ercd = dhcp_inform(T_DHCP_CLIENT *dhcp);

【パラメータ】

T_DHCP_CLIENT	*dhcp	DHCPクライアント情報

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	* dhcp が NULL または socid が指定されていない または再利用するアドレスが無い。
E_NOMEM	メモリ不足（ネットワークバッファの枯渇）
E_OBJ	ホストにIPアドレスが設定されていない。
E_TMOUT	DHCPサーバーからの応答が遅延。またはDHCPサーバーが存在しない。

【解説】

この API は DHCP サーバーから IP アドレス以外の情報を取得します。たとえば静的に IP アドレスを設定し、DNS サーバーのアドレスは DHCP サーバーから取得したい場合などに使用します。

引数の DHCP クライアント情報にはインタフェースのデバイス番号とソケット ID のみを設定します。

6.5.2 DHCP クライアント拡張使用例

使用例

```
ER ercd;
T_DHCP_CLIENT dhcp;
T_DHCP_UOPT dhcp_uopt[3];
INT     uo_i4;
UINT    uo_u4;
VB      uo_buf[64];

/* T_DHCP_UOPT 構造体の設定 */
net_memset(&dhcp_uopt[0], 0, sizeof(dhcp_uopt));

/* Time offset(2), Host Name(12), Log Server(7) */
SET_DHCP_UOPT_BIN4(dhcp_uopt[0], 2, &uo_i4);
SET_DHCP_UOPT_STR (dhcp_uopt[1], 12, uo_buf, sizeof(uo_buf));
SET_DHCP_UOPTS_IPA(dhcp_uopt[2],   7, uo_u4,   sizeof(uo_u4));
/* {T_DHCP_CLIENT 構造体の各種設定} */
dhcp.uopt = dhcp_uopt;
dhcp.uopt_len = 3;

/* {... T_DHCP_CLIENT 構造体の設定 ...} */
ercd = dhcp_bind(&dhcp);
if (E_OK == ercd) {
    for (ercd = 0; ercd < sizeof(dhcp_uopt); ++ercd) {
        if (dhcp_uopt[ercd].flag & DHCP_UOPT_STS_SET) {
            /* オプション取得時処理を行う */
            /* dhcp_uopt[ercd].val のポインタに値が入る */
        }
    }
}
```

ソース解説

上記例では TimeOffset(2)、HostName(12)、LogServer(7)のオプションを取得するように設定を行っています。処理の簡単化のためマクロを使用しています。T_DHCP_UOPT 構造体の変数とは別に、対象オプションの値を格納する変数が必要になることに注意してください。情報取得用の DHCP クライアント API（ここでは dhcp_bind()）を実行後に取得したオプションの値が上記変数に格納されます。

- T_DHCP_UOPT 要素判別・状態フラグ定義値

```
/* ユーザー設定値 */
#define DHCP_UOPT_STR        0x80    オプションは文字列形式
#define DHCP_UOPT_IPA        0x40    オプションはアドレス形式
#define DHCP_UOPT_BIN        0x20    オプションはバイナリ形式
/* ユーザー参照値 */
#define DHCP_UOPT_STS_SET    0x01    オプションの値が設定された
```

T_DHCP_UOPT::flag に入る定義値

- T_DHCP_UOPT 設定用マクロ

```
/* オプションの取得値が単数の場合用 */
SET_DHCP_UOPT_BIN1(_uopt_,_code_,_pval_)          1 バイトのバイナリ用
SET_DHCP_UOPT_BIN2(_uopt_,_code_,_pval_)          2 バイトのバイナリ用
SET_DHCP_UOPT_BIN4(_uopt_,_code_,_pval_)          4 バイトのバイナリ用
SET_DHCP_UOPT_IPA(_uopt_,_code_,_pval_)           アドレス用
SET_DHCP_UOPT_STR(_uopt_,_code_,_pval_,_len_)     文字列用

/* オプションの取得値が複数の場合用 */
SET_DHCP_UOPTS_BIN1(_uopt_,_code_,_pval_,_len_)   1 バイトのバイナリ用
SET_DHCP_UOPTS_BIN2(_uopt_,_code_,_pval_,_len_)   2 バイトのバイナリ用
SET_DHCP_UOPTS_BIN4(_uopt_,_code_,_pval_,_len_)   4 バイトのバイナリ用
SET_DHCP_UOPTS_IPA(_uopt_,_code_,_pval_,_len_)    アドレス用
```

T_DHCP_UOPT 構造体の値を設定するための補助マクロ。マクロの引数にはそれぞれ
uopt：T_DHCP_UOPT 構造体の実体、_code_：オプションコード、
pval：オプション値の格納先ポインタ、_len_：取得可能な要素数（_pval_の要素数）
を指定します。

6.6 Ping クライアント

Ping クライアントは任意の宛先に対して、ICMP エコー要求を送信します。相手先からエコー応答があれば IP アドレスでの通信が可能であることがわかります。

(1) PING クライアント制御情報

```
typedef struct t_ping_client {
        SID         sid;        /* ICMP ソケット ID */
        UW          ipa;        /* 宛先 IP アドレス */
        TMO         tmo;        /* 応答待ちタイムアウト（ms） */
        UH          devnum;     /* デバイス番号 */
        UH          len;        /* パケットサイズ（バイト） */
} T_PING_CLIENT ;
```

この構造体に必要な情報をセットして PING クライアント API の引数として渡します。

尚、Ping クライアントは ICMP ソケットを使用して送受信を行います。
次のように cre_soc でソケットを作成してください。ポート番号は 0 にします。作成したソケットの ID を構造体の T_PING_CLIENT の sid に代入してください。

ICMP ソケット作成

```
T_NODE node;

node .num   = 1;        /* ネットワークのデバイス番号 */
node .ipa   = INADDR_ANY;
node .port = 0;         /* Port should 0 for ICMP */
node .ver   = IP_VER4;
sid = soc_cre(IP_PROTO_ICMP, & node);
if (sid <= 0) {
      return E_NOMEM;
}
```

6.6.1 Ping クライアント API

ping_client	送信

【書式】

ER ping_client(T_PING_CLIENT *ping_client);

【パラメータ】

T_PING_CLIENT	*ping_client	ping送信情報

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。
E_TMOUT	アドレス解決失敗、宛先からの応答無し。
E_NOMEM	メモリ不足（ネットワークバッファの枯渇）
E_OBJ	その他のエラー

【解説】

宛先の IP アドレスへ ping を送信します。本関数は送信先から応答が返るまで待ち続けます。また、送信先から応答が無い場合はタイムアウト（E_TMOUT）のエラーを返します。

6.7 SNTP クライアント

SNTP クライアントは NTP パケットを利用してネットワーク上の時刻サーバー（NTP サーバー）から NTP 時刻（1900/1/1 を起点とした秒数）を取得します。

(1) SNTP クライアント制御情報

```
typedef struct t_sntp_client {
        SID         sid;        /* ソケット ID */
        UW          ipa;        /* SNTP サーバーIP アドレス */
        TMO         tmo;        /* タイムアウト設定 */
        UH          devnum;     /* デバイス番号 */
        UH          port;       /* ポート番号 */
        UB          ipv;        /* IP バージョン */
        UB          stt;        /* 受信データ Stratum フィールド値 */
} T_SNTP_CLIENT ;
```

この構造体に必要な情報をセットして SNTP クライアント API の引数として渡します。

- **デバイス番号**

デバイス番号には SNTP クライアントで使用するネットワークデバイスを指定します。’0’を指定した場合はデフォルトのネットワークデバイスが使用されます。（通常は 0 をセットしてください。）

- **SNTP サーバーIP アドレス**

接続先 SNTP サーバーの IP アドレスを指定してください。

- **タイムアウト設定**

SNTP サーバーからの応答パケットを受信するタイムアウトの時間を指定してください。

- **ポート番号**

SNTP サーバーの接続先ポート番号を変更する際に指定します。通常は 0 を入れてください。

- **IP バージョン**

IP バージョンを指定します。

- **受信データ Stratum フィールド値**

sntp_client()が正常終了した場合、受信データの Stratum フィールドの値が入ります。

Stratum フィールドは通常 0～15 の値となり、0 は時刻非同期 or 時刻ソース不明、1～は時計の階層レベルとなります。(1 がルート階層) 正しい時刻応答まで Stratum フィールドの値が 0 となるサーバーがある場合、この値で判定してください。

6.7.1 SNTP クライアント API

sntp_client	NTP時刻の取得

【書式】

```
ER ercd = sntp_client(T_SNTP_CLIENT *sntp_client, UW *sec, UW *fra);
```

【パラメータ】

T_SNTP_CLIENT	*sntp_client	SNTPクライアント情報
UW	*sec	NTP時刻（秒数）
UW	*fra	NTP時刻（秒端数）

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。
E_TMOUT	アドレス解決失敗、宛先からの応答無し。
E_NOMEM	メモリ不足（ネットワークバッファの枯渇）
E_OBJ	不正なSNTPクライアント情報。

【解説】

この API は引数で設定した SNTP サーバーから NTP 時刻を取得します。SNTP サーバーの設定には、IPv4 アドレスとポート番号を指定して下さい。SNTP クライアントでは UDP ソケットを使用します。使用可能なソケット ID を引数に指定して下さい。

正常に NTP 時刻を取得できた場合は E_OK を返却します。このとき引数の sec と fra には NTP 時刻が入ります。NTP 時刻は 1900/1/1 を起点としているため、UTC(JST)や Unix 時刻への変換は呼出し元で計算する必要があります。

使用例

```
T_SNTP_CLIENT sc = {0};
UW sec, fra;
ER ercd;

sc.sid = ID_SOC_SNTPC;
ercd = sntp_client(&sc, &sec, &fra);
if (ercd == E_OK) {
    /* UnixTime 変換 */
    sec -= 2208988800;

    /* ミリ秒精度の整数表現 */
    fra = ((fra >> 16) * 1000) >> 16;
}
```

6.8 String ライブラリ

TCP/IP スタックではコンパイラに依存しないよう String 系の標準ライブラリを提供します。ネットワークアプリケーションではこれらの提供関数を使用できます。

net_atoi	文字列をint型の数値変換

【書式】

int net_atoi(const char *str);

【パラメータ】

const char	*str	対象文字列

【戻り値】

int	変換結果

【解説】

str によって指示される文字列のはじめの部分を int 型整数に変換します。変換不能な場合は 0 を返します。

net_atol	文字列をlong型の数値変換

【書式】

long net_atol(const char *str);

【パラメータ】

const char	*str	対象文字列

【戻り値】

Long	変換結果

【解説】

str によって指示される文字列のはじめの部分を long 型整数に変換します。変換不能な場合は 0 を返します。

net_itoa　　int型の数値を文字列変換

【書式】

```
char* net_itoa(int num, char *str, int base);
```

【パラメータ】

int	num	対象数値
char	*str	変換結果文字列
int	base	変換基数

【戻り値】

char *	変換結果文字列

【解説】

非標準 C ライブラリです。num によって指示される数値を base の基数で文字列に変換し、結果を str および戻り値に返します。

net_strncasecmp　　文字列の比較(英大文字・小文字同一視)

【書式】

```
int net_strncasecmp(const char *str1, const char *str2, SIZE len);
```

【パラメータ】

const char	*str1	比較文字列1
const char	*str2	比較文字列2
SIZE	len	比較文字数

【戻り値】

int	比較結果

【解説】

文字コードで比較した結果、str1 = str2 なら 0 を返します。str1 > str2 なら正の値、str1 < str2 なら負の値を返します。比較文字数分またはいずれかの文字列の終端に辿り着くまでが比較対象になります。

本関数では英字の大文字・小文字を同一視します。

net_strcmp　　文字列の比較

【書式】

```
int net_strcmp(const char *str1, const char *str2);
```

【パラメータ】

const char	*str1	比較文字列1
const char	*str2	比較文字列2

【戻り値】

int	比較結果

【解説】

文字コードで比較した結果、str1 = str2 なら 0 を返します。str1 > str2 なら正の値、str1 < str2 なら負の値を返します。

いずれかの文字列の終端に辿り着くまでが比較対象になります。

net_strncmp　　文字列の比較

【書式】

```
int net_strncmp(const char *str1, const char *str2, SIZE len);
```

【パラメータ】

const char	*str1	比較文字列1
const char	*str2	比較文字列2
SIZE	len	比較文字数

【戻り値】

int	比較結果

【解説】

文字コードで比較した結果、str1==str2 なら 0 を返します。str1>str2 なら正の値、str1<str2 なら負の値を返します。比較文字数分またはいずれかの文字列の終端に辿り着くまでが比較対象になります。

net_strcpy	文字列のコピー

【書式】

```
char* net_strcpy(char *str1, const char *str2);
```

【パラメータ】

char	*str1	コピー先文字列のアドレス
const char	*str2	コピー元文字列のアドレス

【戻り値】

char *	コピー先文字列のアドレス

【解説】

コピー元文字列 str2 の終端（NULL）までを str1 にコピーします。

net_strlen	文字列長の取得

【書式】

```
SIZE net_strlen(const char *str);
```

【パラメータ】

char	*str	文字列

【戻り値】

SIZE	文字列長

【解説】

str の終端（NULL）までの文字列を取得します。（NULL は含まない）

net_strncat	文字列の連結

【書式】

```
char* net_strncat(char *str1, const char *str2, SIZE len);
```

【パラメータ】

char	*str1	連結先文字列のアドレス
const char	*str2	連結元文字列のアドレス
SIZE	len	連結文字数

【戻り値】

char *	連結先文字列のアドレス

【解説】

連結先文字列 str1 の終端（NULL）を開始位置として str2 の連結文字数分または終端までをコピーします。

net_strcat	文字列の連結

【書式】

```
char* net_strcat(char *str1, const char *str2);
```

【パラメータ】

char	*str1	連結先文字列のアドレス
const char	*str2	連結元文字列のアドレス

【戻り値】

char *	連結先文字列のアドレス

【解説】

連結先文字列 str1 の終端(NULL)を開始位置として str2 の終端までをコピーします。

net_strchr 文字の検索

【書式】

```
char* net_strchr(const char *str, int ch);
```

【パラメータ】

const char	*str	検索対象文字列
int	ch	検索文字

【戻り値】

char *	検索対象文字列の検索文字が現れるアドレス

【解説】

検索対象文字列 str の先頭から終端（NULL）に検索文字 ch が存在するか検索します。検索文字が存在する場合はその先頭アドレスを、存在しない場合は NULL を戻り値に返します。

net_strstr 文字列の検索

【書式】

```
char* net_strstr(const char *str1, const char *str2);
```

【パラメータ】

const char	*str1	検索対象文字列
const char	*str2	検索文字列

【戻り値】

char *	検索対象文字列の検索文字列が現れるアドレス

【解説】

検索対象文字列 str1 の先頭から終端（NULL）に検索文字列 str2 が存在するか検索します。検索文字列が存在する場合はその先頭アドレスを、存在しない場合は NULL を戻り値に返します。

net_strcasestr 文字列の検索 英大文字・小文字同一視

【書式】

```
char* net_strcasestr(const char *str1, const char *str2);
```

【パラメータ】

const char	*str1	検索対象文字列
const char	*str2	検索文字列

【戻り値】

char *	検索対象文字列の検索文字列が現れるアドレス

【解説】

検索対象文字列 str1 の先頭から終端（NULL）に検索文字列 str2 が存在するか検索します。検索文字列が存在する場合はその先頭アドレスを、存在しない場合は NULL を戻り値に返します。

本関数では英字の大文字・小文字を同一視します。

net_strncpy 文字列のn文字コピー

【書式】

```
char* net_strncpy(char *str1, const char *str2, SIZE len);
```

【パラメータ】

char	*str1	コピー先文字列のアドレス
const char	*str2	コピー元文字列のアドレス
SIZE	len	コピー文字数

【戻り値】

char *	コピー先文字列のアドレス

【解説】

コピー元文字列 str2 の終端（NULL）もしくは len(コピー文字数)まで str1 にコピーします。

len が str2 の長さより小さい場合、len 文字しかコピーされないため str1 の len+1 文字目に終端文字は付加されません。

len が str2 の長さより大きい場合、str2 の終端以降の文字列は len の長さまで NULL で埋められます。

7. サンプルを使ったチュートリアル

本章では、TCP/IP スタックに同梱しているアプリケーションサンプルを使用して、プログラムの動作確認を行います。

7.1 サンプルの説明

Project ディレクトリ以下の uNet3_sample を使用します。このサンプルを使って、HTTP サーバーを使用したプログラムの動作確認を行います。

- Web サーバー
  ウェブブラウザから LED の点滅間隔を 100msec 単位で変更させます。

サンプルに含まれるファイルの一覧は「2.3 ディレクトリとファイル構成」を参照してください。TCP/IP スタックは、ドライバ／ミドルウェア一式、を統合して使用してください。

7.2 ハードウェア接続

R-IN32M4-CL3 評価ボードのイーサネットポートは、下図のように**ポート 0** に接続してください（DDR_ETH_CFG.h の USE_ETHSW 定義が“0”のとき）。

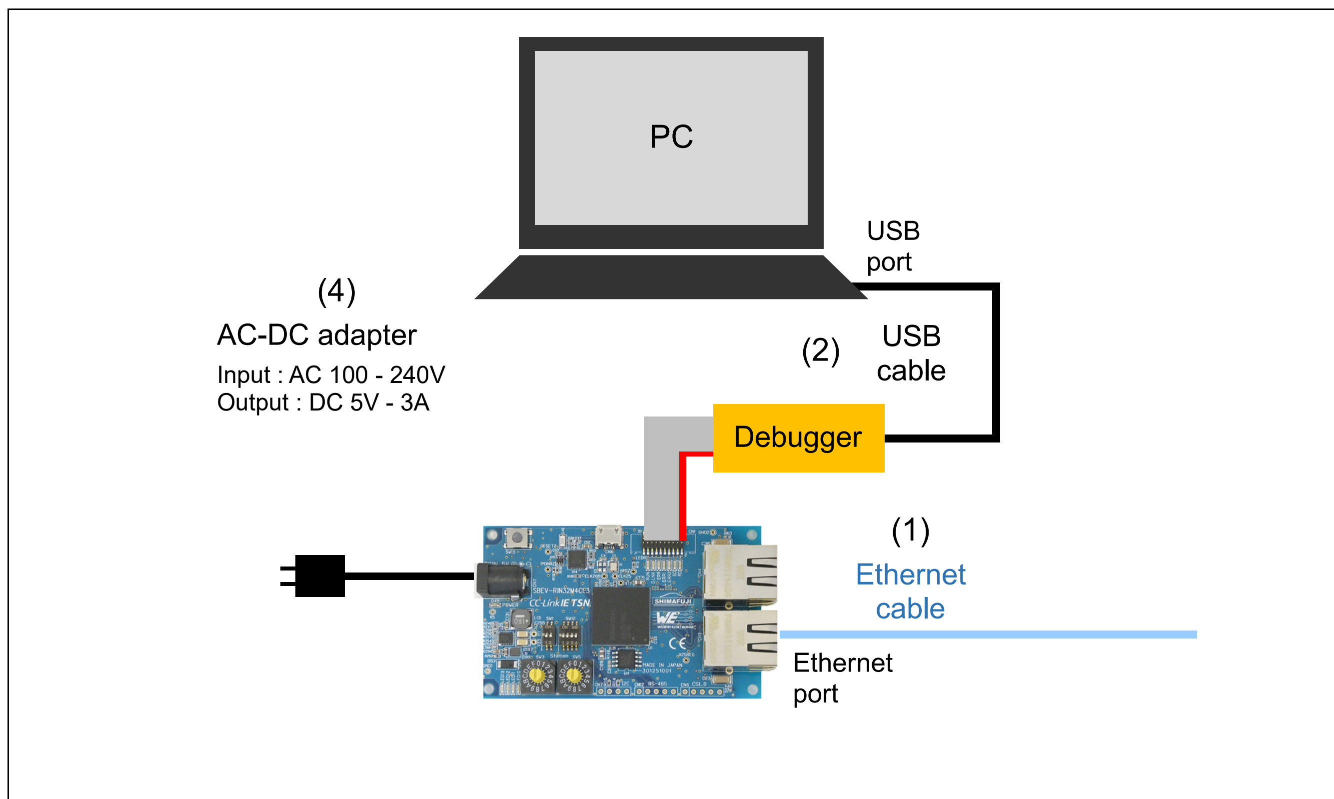


図 7.1 サンプルプログラム実行時のイーサネットポート接続例

7.3 ボード IP アドレス設定

TCP/IP プロトコルスタックの IP アドレス設定には、固定 IP アドレスを使う場合と DHCP コントローラにセットさせる場合の 2 つのオプションがあります。

7.3.1 固定 IP アドレス使用時

以下の手順で設定してください。

(1) net_sample.c内の“DHCP_ENA”定義を“0”に設定。

(2) net_cfg.c内のサーバーネットワークアドレス設定を、図7.2 のように設定。

```
/********************************************
    Define Local IP Address
*********************************************/
T_NET_ADR gNET_ADR[] = {
    {
        0x0,            /* Reserved */
        0x0,            /* Reserved */
        0xC0A80164,     /* IP address   (192.168.   1.100) */
        0xC0A80101,     /* Gateway      (192.168.   1.   1) */
        0xFFFFFF00,     /* Subnet mask (255.255.255.   0) */
    }
};
```

図 7.2 IP アドレスの設定例（IP アドレスが 192.168.1.100 の場合）

(3) Host PCのアドレスをR-IN32ボードと同じドメインに設定してください。（設定方法の詳細については次頁も参照ください。）

設定例：

サブネットマスク　：　255.255.255.0

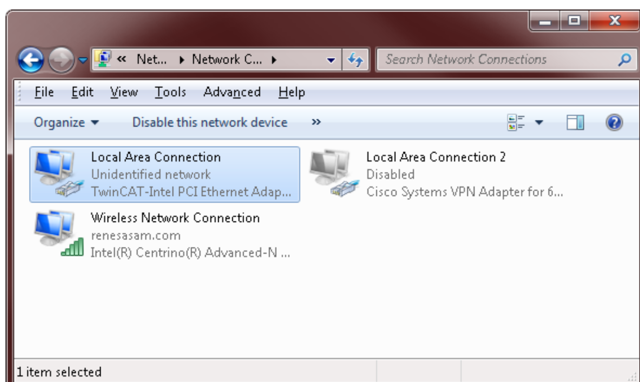
PC IPアドレス　：　192.168.1.101

(4) 次の節(7.3.2 DHCP機能使用時) をスキップ。

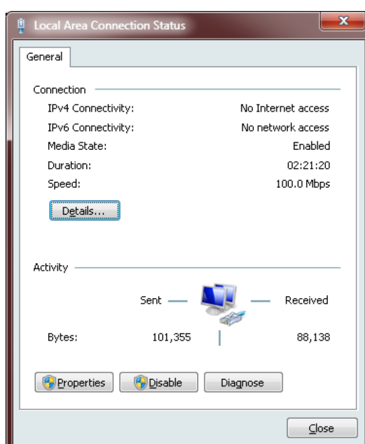
参考：Host PCのIPアドレス設定方法

- “ネットワーク設定”を開く。

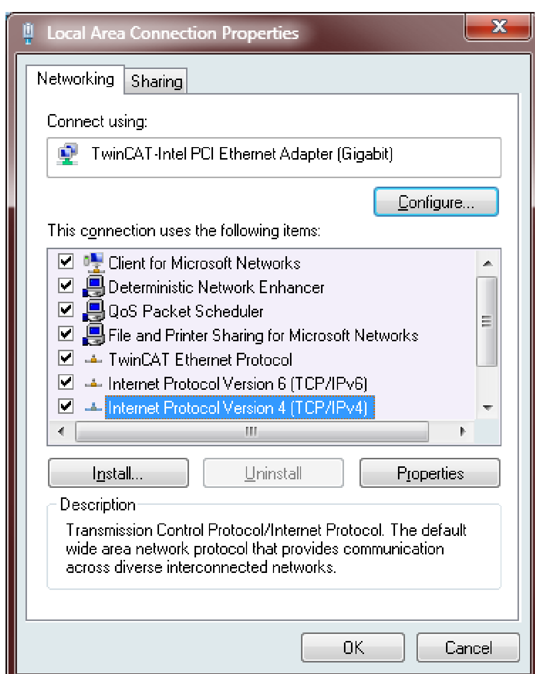
  Windows7 では、Control panel->Network and Sharing Center->Change adapter settings。



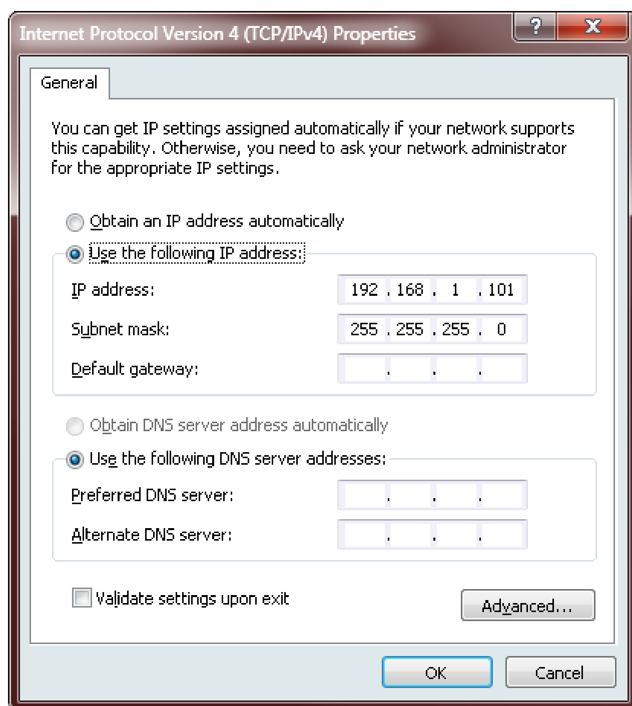
- ローカルエリア接続をダブルクリック（もしくは右クリック）しプロパティを選択。



- TCP/IPv4 を選択し、プロパティボタンを押下。



・ IP アドレスを 192.168.1.101 に設定、サブネットマスクを 255.255.255.0 に設定。



以上です。

7.3.2 DHCP 機能使用時

DHCP クライアントを有効にした場合は、DHCP を用いて自動的に IP アドレスが取得されます。また、この場合には DHCP 用の UDP ソケットが自動的に追加されます。

以下の手順で設定してください。

(1) net_sample.c内の“DHCP_ENA”定義を“1”に設定。

(2) LANケーブルをイーサネットポート1に接続。

(3) LANケーブルをPCに接続

注 ご使用の EWARM が評価版（32KB サイズ制限）の場合，サンプルプロジェクトがサイズ制限によりコンパイルできない可能性があります。

7.4 動作確認

7.4.1 Web サーバー

この例では、サンプルアプリケーション“uNet3_sample”を使用します。
以下の手順を実施してください。

1. プログラムをコンパイル、ダウンロードし、アプリケーションを実行
2. PC（クライアント）でインターネットエクスプローラを起動
3. URLフィールドに、http://192.168.1.100 （もしくは http://192.168.1.100:80 を）設定

上記にてプロジェクトを実行することで、図 7.3 のように R-IN32 から送信された Web ページが確認できます。もし動作に問題があるようであれば、IAR でのプログラムのリスタートの実施や、ブレイクポイントによるタイムアウトなどが無いかの確認を行ってください。

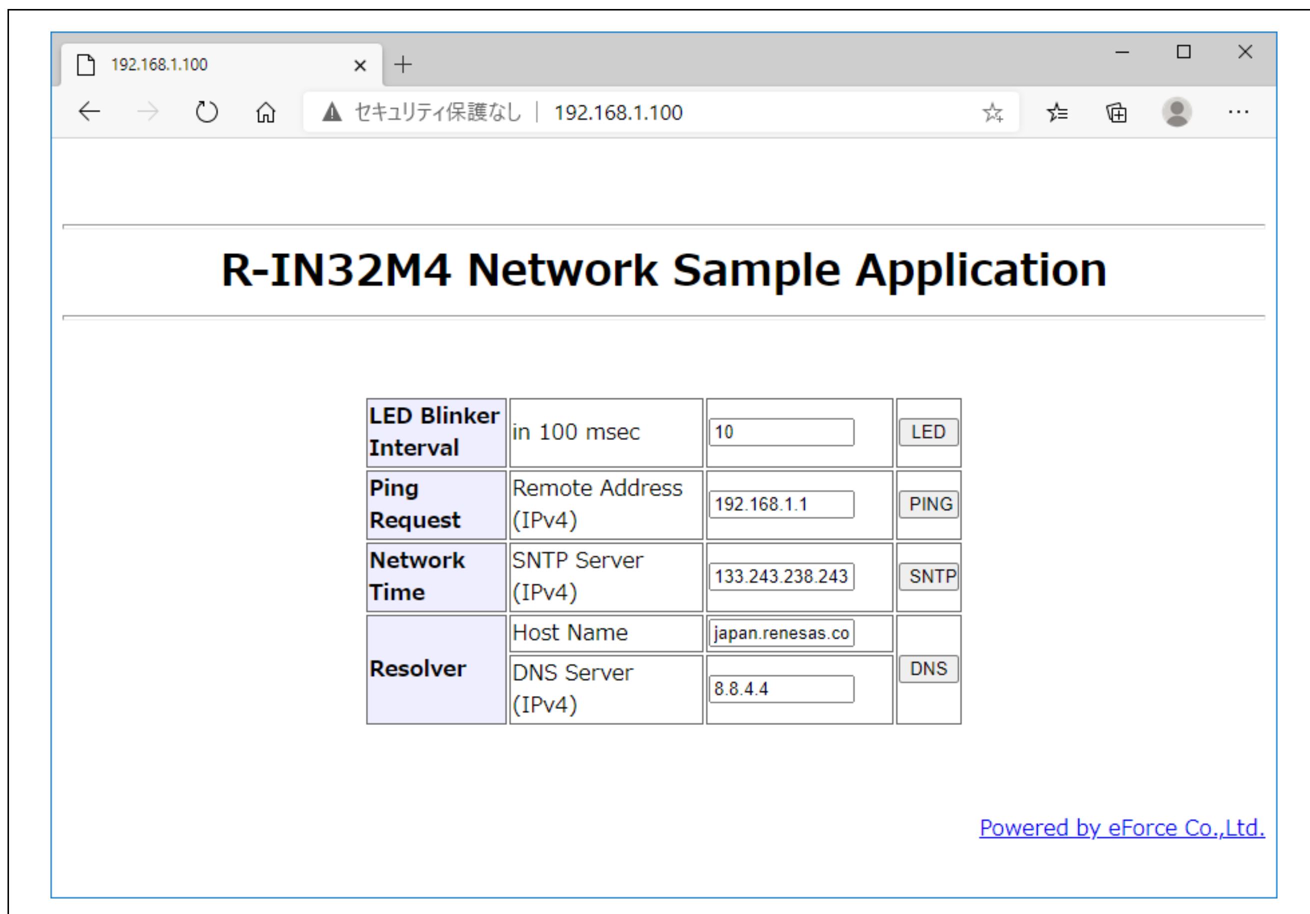


図 7.3 サンプルアプリケーションの実行結果（HTTP サーバー機能）

7.4.2 MAC コントローラ制御

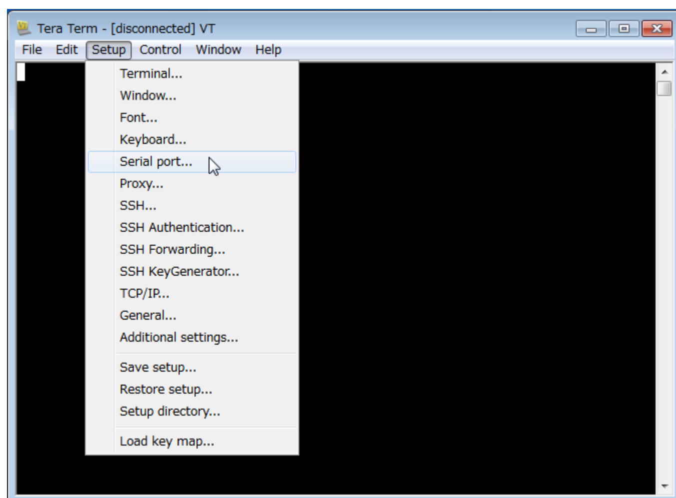
この例では、サンプルアプリケーション“uNet3_mac”を使用します。
以下の手順を実施してください。

1. プログラムをコンパイル、ダウンロードし、アプリケーションを実行。
2. シリアルコンソールからコマンドを入力して MAC コントローラを動作させる。
   コマンドの入力は USB シリアルポートを接続し、ターミナルソフトを起動。
3. コマンド実行にて動作確認。

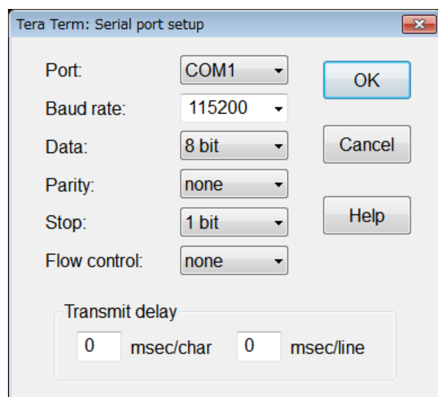
上記にてプロジェクトを実行することで、ターミナルソフトよりホスト IP アドレスの確認、PHY のモードの変更等が行えます。

ターミナルソフトの設定

- 設定タブをクリックしシリアルポートを選択。



- 下記の通り設定。（ポート設定は、お使いの PC に依存します。）



- 接続が成功すると以下のプロンプトが出力されます。

使用可能なコマンドの説明

ip　：ホストの IP アドレスと MAC アドレスを表示。
phy　: PHY のモード(スピード、デュプレックス、オートネゴシエーション)を変更。
view : PHY のモード(スピード、デュプレックス、リンク状態、オートネゴシエーション)を表示。
mac　: MAC アドレスの受信フィルタリングを動的に設定。
rx　：受信する Ethernet フレームをダンプ出力するように設定・解除。
tx　：任意の Ethernet フレームを送信。
?　：コマンドの一覧を表示。

```
R-IN32>ip
 IP Address  : 192.168.1.100
 Subnet Mask : 255.255.255.0
 Gateway     : 192.168.1.1
```

図 7.4　　コントローラの実行結果例（ip）

```
R-IN32>phy
Duplex? HALF(0)/FULL(1)  : 1
Speed?  10M(0)/100M(1)/1000M(2)   : 1
Auto nego?  NO(0)/YES(1) : 1
PHY channel? 1/2         : 2
R-IN32>view
PHY channel? 1/2 : 2
 Speed  : 100
 Duplex : Full
 Link   : up
 Nego   : on
```

図 7.5　　コントローラの実行結果例（phy,view）

・Duplex? HALF(0)/FULL(1)　→　通信方式についての変更。
　FULL「全二重通信」同時に双方向に通信ができる方式
　HALF「半二重通信」同時に単方向にしか通信ができず、切り替えて交互に通信を行う方式
・Speed?　→　通信速度についての変更。(mbps 単位)
・Auto nego?　NO(0)/YES(1)　→　オートネゴシエーション機能の切り替え。
　(接続時に最適な設定で通信をするよう自動的に設定する機能)
・PHY channel? 1/2　→　モードの変更を行う対象ポートの切り替え。
　PHY channel 1 = PHY port0
　PHY channne 2 = PHY port1

```
R-IN32>rx
Enable raw frame reception?  NO(0)/YES(1) : 1
R-IN32>
 FF FF FF FF FF FF CC E1 D5 0D 15 91 08 06 00 01 08 00 06 04 00 01 CC E1
5 91 C0 A8 01 65 00 00 00 00 00 00 C0 A8 01 64 00 00 00 00 00 00 00 00 0
00 00 00 00 00 00 00
 12 34 56 78 9A BC CC E1 D5 0D 15 91 08 00 45 00 00 3C 00 D9 00 00 80 01
0 A8 01 65 C0 A8 01 64 08 00 4D 56 00 01 00 05 61 62 63 64 65 66 67 68 6
6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69
 12 34 56 78 9A BC CC E1 D5 0D 15 91 08 00 45 00 00 3C 00 DA 00 00 80 01
0 A8 01 65 C0 A8 01 64 08 00 4D 55 00 01 00 06 61 62 63 64 65 66 67 68 6
6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69
 12 34 56 78 9A BC CC E1 D5 0D 15 91 08 00 45 00 00 3C 00 DB 00 00 80 01
0 A8 01 65 C0 A8 01 64 08 00 4D 54 00 01 00 07 61 62 63 64 65 66 67 68 6
6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69
 12 34 56 78 9A BC CC E1 D5 0D 15 91 08 00 45 00 00 3C 00 DC 00 00 80 01
```

図 7.6コントローラの実行結果例（rx）※R-IN32 に対し ping を実行した例。

・Enable raw frame reception?　NO(0)/YES(1)　→　R-IN32 にて受信したデータ表示の切り替え。

```
R-IN32>tx
Input destination MAC address[0] in hex : 01
Input destination MAC address[1] in hex : 02
Input destination MAC address[2] in hex : 03
Input destination MAC address[3] in hex : 04
Input destination MAC address[4] in hex : 05
Input destination MAC address[5] in hex : 06
Destination MAC address : 01.02.03.04.05.06
Input destination IP address : 1.1.1.1
Input UDP port(Local/Remote)  : 100
```

No.	Time	Source	Destination	Protocol	Length
1	0.000000	192.168.1.100	1.1.1.1	UDP	1514

```
▷ Frame 1: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on
▷ Ethernet II, Src: 12:34:56:78:9a:bc (12:34:56:78:9a:bc), Dst: Woonsang_04:05:
▷ Internet Protocol Version 4, Src: 192.168.1.100, Dst: 1.1.1.1
▷ User Datagram Protocol, Src Port: 100 (100), Dst Port: 100 (100)
▷ Data (1464 bytes)
```

図 7.7　　コントローラの実行結果例（tx）上：コントローラ、下：パケットキャプチャ

※データ確認にはパケットキャプチャを行うソフトを使用してください。

・Input destination MAC address[0~5] in hex :　→　送信先 MAC アドレスの設定

・Input destination IP address :　→　送信先 IP アドレスの設定

・Input UDP port(Local/Remote)　:　→　データ送受信のポート設定

7.4.3 BSD ソケット

この例では、サンプルアプリケーション“uNet3_bsd”を使用します。
以下の手順を実施してください。

1. プログラムをコンパイル、ダウンロードし、アプリケーションを実行。
2. コマンドの入力は USB シリアルポートを接続し、ターミナルソフトを起動。
3. コマンド実行にて動作確認。

上記にてプロジェクトを実行することで、デバッグコンソールから BSD ソケットの各 API を実行・動作確認が行えます。
※動作確認には R-IN32 にて生成したソケットに合わせて PC(クライアント)側のソケットが別途必要となります。

下記はサポートするコマンドとなります。

[BSD ソケット用コマンド]

socket	ソケットの作成
bind	ソケットに名前をつける
connect	ソケットの接続を行う
listen	ソケット上の接続を待つ
accept	ソケットへの接続を受ける
send	ソケットへメッセージを送る
sendto	ソケットへメッセージを送る(送信先指定)
recv	ソケットからメッセージを受け取る
recvfrom	ソケットからメッセージを受け取る(送信元指定)
select	1 つ以上のファイルディスクリプタの変更を監視する
shutdown	全二重接続の一部を閉じる
close	ソケットを閉じる
getsockopt	ソケットのオプションの取得を行う
setsockopt	ソケットのオプションの設定を行う
getsockname	ソケットの名前を取得する
getpeername	接続している相手ソケットの名前を取得する
ioctl	デバイスを制御する

[その他コマンド]

netstat	ソケット一覧の表示
?	コマンド一覧の表示

※BSD ソケットの概要については「2.1.19 ソケット」を参照してください。
※ターミナルソフトの設定方法については「7.4.2 MAC コントローラ制御」を参照してください。

7.4.3.1 UDP ソケットによるパケット送受信

```
uNet3/BSD>socket
- type? (6:SOCK_STREAM, 17:SOCK_DGRAM, 0:SOCK_RAW) : 17
# func_success ->6
uNet3/BSD>bind
sockfd : 6
- local-addr.
   ip (ex: 192.168.1.1) : 192.168.1.100
   port                 : 100
# func_success ->0
uNet3/BSD>recv
sockfd : 6
- buffer size(int, MAX=1024) : 100
# func_success ->4
[output]

uNet3/BSD>sendto
sockfd : 6
- remote-addr.
   ip (ex: 192.168.1.1) : 192.168.1.101
   port                 : 100
- buffer size(int, MAX=1024) : 100
# func_success ->100
```

	No.	Ti	Source	Destination	Protocol	Length
※1	1	…	192.168.1.101	192.168.1.100	UDP	46
※2	2	…	192.168.1.100	192.168.1.101	UDP	142

```
Frame 1: 46 bytes on wire (368 bits), 46 bytes captured (368 bits
Ethernet II, Src: BuffaloI_0d:15:91 (cc:e1:d5:0d:15:91), Dst: 12:
Internet Protocol Version 4, Src: 192.168.1.101, Dst: 192.168.1.1
User Datagram Protocol, Src Port: 100 (100), Dst Port: 100 (100)
```

図 7.8　　UDP での通信例　上：コントローラ、下：パケットキャプチャ

UDP 受信

1. ソケットを作成。

   「Socket」ソケット作成。　※UDPソケットなのでSOCK_DGRAMを指定。

   ・type? (6:SOCK_STREAM, 17:SOCK_DGRAM, 0:SOCK_RAW)　→　通信方式。

   17:SOCK_DGRAM　コネクションレス型　UDPを利用した通信

   ※# func_success ->の戻り値が作成されたソケット番号となります。

2. Bind する IP アドレスとポートを設定。

   「bind」ソケットに名前をつける。

   ・sockfd　→　対象のソケットを入力。

   ・local-addr.

   ip　→　IPアドレスの設定

   port　→　ポート設定

3. 「recv」データ受信。※1

   ・sockfd　→　対象のソケットを入力。

   ・buffer size(int, MAX=1024)　→　受信バッファサイズを入力。

UDP 送信

1. 宛先を指定して送信。

   「sendto」送信先IPアドレス、ポートを設定、指定バッファサイズ分データ送信。※2

7.4.3.2 TCP ソケットによる TCP サーバー作成

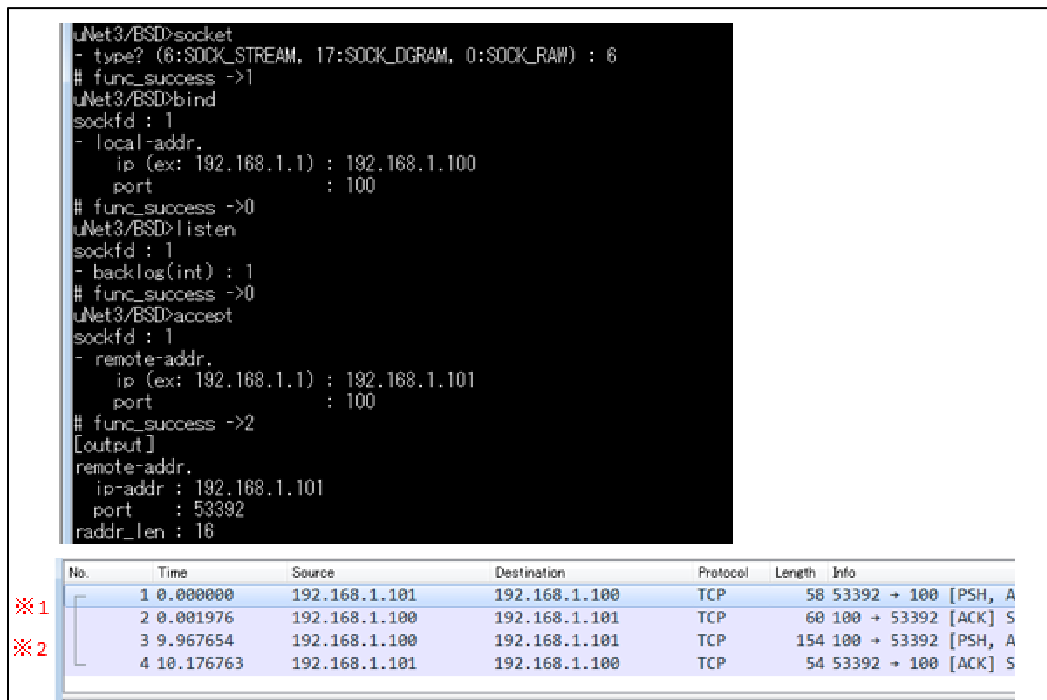


図 7.9 TCP での通信例（サーバー側）上：コントローラ、下：パケットキャプチャ

1. ソケットを作成。

   「Socket」ソケット作成。※TCPソケットなのでSOCK_ STREAMを指定。

   ・type? (6:SOCK_STREAM, 17:SOCK_DGRAM, 0:SOCK_RAW) → 通信方式。

   6:SOCK_STREAM コネクション型 TCPを利用した通信

   ※# func_success ->の戻り値が作成されたソケット番号となります。

2. 接続待ちをする IP アドレスとポートを設定。

   「bind」ソケットに名前をつける。

   ・sockfd → 対象のソケットを入力。

   ・local-addr.

   Ip → IPアドレスの設定

   port → ポート設定

3. 接続待ちをする。

   「listen」接続を待つ。

   ・sockfd → 接続を待ち受けるソケットを入力。

   「accept」接続を受ける。

   ・sockfd → 接続を待ち受けるソケットを入力。

   ・remote-addr.

   Ip → IPアドレスの設定

   port → ポート設定

4. 通信を行う。

   「recv」データ受信。※1

   「send」データ送信。※2

7.4.3.3 TCP ソケットによる TCP クライアント作成

```
uNet3/BSD>socket
- type? (6:SOCK_STREAM, 17:SOCK_DGRAM, 0:SOCK_RAW) : 6
# func_success ->1
uNet3/BSD>connect
sockfd : 1
- remote-addr.
    ip (ex: 192.168.1.1) : 192.168.1.101
    port                 : 100
# func_success ->0
uNet3/BSD>netstat
sockfd  proto  local-addr            remote-addr          listen
1       TCP    192.168.1.100:49154   192.168.1.101:100    x
# func_success ->0
uNet3/BSD>send
sockfd : 1
- buffer size(int, MAX=1024) : 100
# func_success ->100
uNet3/BSD>recv
sockfd : 1
- buffer size(int, MAX=1024) : 100
# func_success ->4
[output]
```

No.	Ti	Source	Destination	Protocol	Length
1	…	192.168.1.100	192.168.1.101	TCP	154
2	…	192.168.1.101	192.168.1.100	TCP	54
3	…	192.168.1.101	192.168.1.100	TCP	58
4	…	192.168.1.100	192.168.1.101	TCP	60

※1（No.1–2）　※2（No.3–4）

```
▷ Frame 1: 154 bytes on wire (1232 bits), 154 bytes captured (1232
▷ Ethernet II, Src: 12:34:56:78:9a:bc (12:34:56:78:9a:bc), Dst: Buf
▷ Internet Protocol Version 4, Src: 192.168.1.100, Dst: 192.168.1.1
▷ Transmission Control Protocol, Src Port: 49154 (49154), Dst Port:
```

図 7.10 TCP での通信例（クライアント側）上：コントローラ、下：パケットキャプチャ

1. ソケットを作成。

   「Socket」ソケット作成。※TCPソケットなのでSOCK_ STREAMを指定。

   ・type? (6:SOCK_STREAM, 17:SOCK_DGRAM, 0:SOCK_RAW)　→　通信方式。

   6:SOCK_STREAM　コネクション型　TCPを利用した通信

   ※# func_success ->の戻り値が作成されたソケット番号となります。

2. 接続相手の設定。

   「connect」ソケットへの接続を行う。

   ・remote-addr.

   ip　→　IPアドレスの設定

   port　→　ポート設定

3. 通信を行う。

   「recv」データ受信。※1

   「send」データ送信。※2

7.4.4 ノンブロッキング通信

この例では、サンプルアプリケーション“uNet3_nonblock”を使用します。
以下の手順を実施してください。

1. プログラムをコンパイル、ダウンロードし、アプリケーションを実行。
2. 作成されたソケットに対しデータ送信し動作確認。

本サンプルアプリケーションは，TCP/IPスタックの動作を確認するための簡易プログラムです。
ソケットのノンブロッキング API を使って、TCP ソケットと UDP ソケットのエコーサーバを 1 つのタスク上で動作させます。
プログラム起動後 TCP ソケットは 10000 番ポートをリスニング、UDP ソケットは 20000 番ポートを受信します。それぞれのソケットがデータを受信すると、送信元に受信データを折り返します。

※ノンブロッキング通信の概要については「2.1.20 ブロッキングとノンブロッキング」を参照してください。
※R-IN32 にて生成したソケットに合わせて PC(クライアント)側のソケットが別途必要となります。

No.	Ti	Source	Destination	Protocol	Length	Info
1	…	192.168.1.101	192.168.1.100	UDP	45	20000 → 20000
2	…	192.168.1.100	192.168.1.101	UDP	60	20000 → 20000

```
Frame 1: 45 bytes on wire (360 bits), 45 bytes captured (360 bits) on interface
Ethernet II, Src: BuffaloI_0d:15:91 (cc:e1:d5:0d:15:91), Dst: 12:34:56:78:9a:bc
Internet Protocol Version 4, Src: 192.168.1.101, Dst: 192.168.1.100
User Datagram Protocol, Src Port: 20000 (20000), Dst Port: 20000 (20000)
Data (3 bytes)
```

図 7.11 UDP での通信例

No.	Ti	Source	Destination	Protocol	Length	Info
1	…	192.168.1.101	192.168.1.100	TCP	57	[TCP segment of a
2	…	192.168.1.100	192.168.1.101	TCP	60	10000 → 50388 [AC
3	…	192.168.1.100	192.168.1.101	TCP	60	[TCP segment of a
4	…	192.168.1.101	192.168.1.100	TCP	54	50388 → 10000 [AC

```
Frame 1: 57 bytes on wire (456 bits), 57 bytes captured (456 bits) on interface 0
Ethernet II, Src: BuffaloI_0d:15:91 (cc:e1:d5:0d:15:91), Dst: 12:34:56:78:9a:bc (
Internet Protocol Version 4, Src: 192.168.1.101, Dst: 192.168.1.100
Transmission Control Protocol, Src Port: 50388 (50388), Dst Port: 10000 (10000),
```

図 7.12 TCP での通信例

8. 付録

8.1 パケット形式

(1) T_NODE

通信端点の情報

```
typedef struct t_node {
        UH      port;           /* ソケットのポート番号 */
        UB      ver;            /* IP バージョン（必ず IP_VER4 指定）*/
        UB      num;            /* デバイス番号*/
        UW      ipa;            /* IP アドレス */
} T_NODE;
```

(2) T_NET_ADR

ネットワークアドレスの情報

```
typedef struct t_net_adr {
        UB      ver;            /* IP バージョン（必ず IP_VER4 指定）*/
        UB      mode;           /* 予約 */
        UW      ipaddr;         /* IP アドレス */
        UW      gateway;        /* ゲートウェイ */
        UW      mask;           /* サブネットマスク */
} T_NET_ADR;
```

(3) T_NET_DEV

ネットワークデバイスの情報

```
typedef struct t_net_dev {
        UB          name[8];    /* デバイス名 */
        UH          num;        /* デバイス番号 */
        UH          type;       /* デバイスタイプ */
        UH          sts;        /*予約 */
        UH          flg;        /* 予約 */
        FP          ini;        /* dev_ini 関数へのポインタ*/
        FP          cls;        /* dev_cls 関数へのポインタ*/
        FP          ctl;        /* dev_ctl 関数へのポインタ*/
        FP          ref;        /* dev_ref 関数へのポインタ*/
        FP          out;        /* dev_snd 関数へのポインタ*/
        FP          cbk;        /* dev_cbk 関数へのポインタ*/
        UW          *tag;       /* 予約 */
        union {                 /* MAC アドレス */
         struct {
        UB          mac[6];
        }eth;
        } cfg;
        UH          hhdrsz;     /* デバイスヘッダ―サイズ */
        UH          hhdrofs;    /* ネットワークバッファ書き込み位置*/
        VP          opt;        /* ドライバ拡張領域 */
} T_NET_DEV;
```

(4) T_NET_BUF

ネットワークバッファの情報

```
typedef struct t_net_buf {
        UW          *next;      /* 予約 */
        ID          mpfid;      /* メモリプール ID */
        T_NET       *net;       /* ネットワークインタフェース */
        T_NET_DEV   *dev;       /* ネットワークデバイス */
        T_NET_SOC   *soc;       /* ソケット */
        ER          ercd;       /* エラーコード */
        UH          flg;        /* ブロードキャスト・マルチキャストフラグ */
        UH          seq;        /* フラグメントシーケンス */
        UH          dat_len;    /* パケットのデータサイズ */
        UH          hdr_len;    /* パケットのヘッダーサイズ */
        UB          *dat;       /* パケット(buf)内のデータ位置を指す */
        UB          *hdr;       /* パケット(buf)内のヘッダー位置を指す */
        UB          buf[];      /* 実際のパケット */
} T_NET_BUF ;
```

(5) T_HOST_ADDR

ホストアドレスの情報

```
typedef struct t_host_addr {
        UW      ipaddr;     /* IP アドレス */
        UW      subnet;     /* サブネットマスク */
        UW      gateway;    /* ゲートウェイ */
        UW      dhcp;       /* DHCP サーバーアドレス */
        UW      dns[2];     /* DNS アドレス */
        UW      lease;      /* DHCP アドレスのリース期間 */
        UW      t1;         /* DHCP アドレスのリニューアル期間*/
        UW      t2;         /* DHCP アドレスのリバインド期間 */
        UB      mac[6];     /* MAC アドレス */
        UH      dev_num;    /* デバイス番号 */
        UB      state;      /* DHCP クライアント状態 */
        SID     socid;      /* UDP ソケット ID */
} T_HOST_ADDR ;
```

(6) T_FTP_SERVER

FTP サーバー制御情報

```
typedef struct t_ftp_server {
        ・       ・
        UW       sec;            /* セキュリティポリシー */
        ・       ・
        ・       ・
        UH       dev_num;        /* デバイス番号*/
        SID      ctl_sid;        /* 制御用ソケット ID */
        SID      dat_sid;        /* データ用ソケット ID */
        ・       ・
        ・       ・
        ER (*auth_cbk)(UH, const char*, const char*);    /* 認証コールバック */
        VB*      syst_name;      /* SYST コマンド応答文字列 */
} T_FTP_SERVER ;
```

(7) T_HTTP_FILE

HTTP コンテンツ情報

```
typedef struct t_http_file {
        const char   *path;                     /* URL */
        const char   *ctype;                    /* コンテンツタイプ */
        const char   *file;                     /* コンテンツ */
        Int          len;                       /* コンテンツサイズ*/
                                                /* HTTP コールバック関数
        void(*cbk)(T_HTTP_SERVER *http);        or
                                                CGI ハンドラ */
        UB           ext;                       /* 拡張動作フラグ */
} T_HTTP_FILE ;
```

(8) T_HTTP_SERVER

HTTP サーバー制御情報

```
typedef struct t_http_server {
        UW                  sbufsz;             /* 送信バッファサイズ */
        UW                  rbufsz;             /* 受信バッファサイズ */
        UW                  txlen;              /*内部データ */
        UW                  rxlen;              /*内部データ */
        UW                  rdlen;              /*内部データ */
        UW                  len;                /*内部データ */
        UB                  *rbuf;              /* 送信バッファ*/
        UB                  *sbuf;              /* 受信バッファ*/
        UB                  *req;               /* 内部データ*/
        UH                  Port;               /* リスニングポート番号*/
        SID                 SocketID;           /* ソケット ID */
        T_HTTP_HEADER       hdr;                /* HTTP クライアントリクエスト */
        UB                  NetChannel;         /* デバイス番号 */
        UB                  ver                 /*IP バージョン */
        UB                  server_tsk_stat     /* HTTP サーバータスク起動状態*/
        ID                  server_tsk_id       /* HTTP サーバータスク ID */
        struct t_http_server *next              /* HTTP サーバーオブジェクト */
        UH                  kpa_max             /* HTTP KeepAlive max 値（制御用） */
} T_HTTP_SERVER;
```

(9) T_HTTP_HEADER

HTTP ヘッダ情報

```
typedef struct t_http_header {
        char                *method;            /* メソッド */
        char                *url;               /* パス名 */
        char                *url_q;             /* URL クエリ*/
        char                *ver;               /* バージョン */
        char                *host;              /* ホスト名 */
        char                *ctype;             /* コンテンツタイプ */
        char                *Content;           /* コンテンツ */
        char                ContentLen;         /* コンテンツ長 */
        char                kpa;                /* HTTP Keep Alive 用フラグ（制御用） */
        char                *auth;              /* Authorization ヘッダ */
        char                *cookie;            /* Cookie ヘッダ */
} T_HTTP_HEADER;
```

(10) T_RCV_PKT_INF

受信パケット情報

```
typedef struct t_rcv_pkt_inf{
        UW      src_ipa;        /* パケットの送信元 IP アドレス */
        UW      dst_ipa;        /* パケットの送信先 IP アドレス */
        UH      src_port;       /* パケットの送信元ポート番号 */
        UH      dst_port;       /* パケットの送信先ポート番号 */
        UB      ttl;            /* パケットの IP ヘッダ TTL */
        UB      tos;            /* パケットの IP ヘッダ TOS */
        UB      ver;            /* パケットの IP ヘッダーバージョン */
        UB      num;            /* パケットの受信デバイス番号 */
} T_RCV_PKT_INF;
```

(11) T_DNS_CLIENT

DNS クライアント情報

```
typedef struct t_dns_client {
        UW              ipa;            /* DNS サーバ IP アドレス */
        char            *name;          /* ホスト名(設定・参照用) */
        UW              *ipaddr;        /* IP アドレス(設定・参照用) */
        SID             sid;            /* DNS ソケット ID(UDP) */
        UH              code:           /* 要求 RR タイプ */
        UB              dev_num;        /* デバイス番号 */
        UB              retry_cnt;      /* リトライ回数 */
} T_DNS_CLIENT ;
```

(12) T_DHCP_CLIENT

DHCP クライアント情報

```
typedef struct t_dhcp_client {
        T_DHCP_CTL      ctl             /*内部データ*/
        UW              ipaddr;         /* IP アドレス */
        UW              subnet;         /* サブネットマスク */
        UW              gateway;        /* ゲートウェイ */
        UW              dhcp;           /* DHCP サーバーアドレス */
        UW              dns[2];         /* DNS アドレス */
        UW              lease;          /* DHCP アドレスのリース期間 */
        UW              t1;             /* DHCP アドレスのリニューアル期間*/
        UW              t2;             /* DHCP アドレスのリバインド期間 */
        UB              mac[6];         /* MAC アドレス */
        UH              dev_num;        /* デバイス番号 */
        UB              state;          /* DHCP クライアント状態 */
        SID             socid;          /* UDP ソケット ID */
        UB              arpchk;         /* IP 重複チェック有無 */
        T_DHCP_UOPT     *uopt;          /* DHCP オプション取得パラメータ */
        UB              uopt_len;       /* DHCP オプション取得パラメータの数 */
        UB              retry_cnt;      /* リトライ回数 */
} T_DHCP_CLIENT;
```

(13) T_DHCP_UOPT

DHCP 取得オプション情報

```
typedef struct t_dhcp_uopt {
        T_DHCP_CTL      ctl             /* 内部データ */
        UB              code;           /* DHCP オプションコード */
        UB              len;            /* オプション要素のサイズ／個 */
        UB              ary;            /* オプション要素の数 */
        UB              flag;           /* 要素判別・状態フラグ */
        VP              val;            /* 要素格納先ポインタ */
} T_DHCP_UOPT;
```

(14) T_PING_CLIENT

Ping クライアント情報

```
typedef struct t_ping_client {
        SID     sid;            /* ICMP ソケット ID */
        UW      ipa;            /* 宛先 IP アドレス */
        TMO     tmo;            /* 応答待ちタイムアウト（ms） */
        UH      devnum;         /* デバイス番号 */
        UH      len;            /* パケットサイズ（バイト） */
} T_PING_CLIENT;
```

(15) T_SNTP_CLIENT

SNTP クライアント情報

```
typedef struct   t_sntp_client {
        SID         sid;            /*  ソケット ID */
        UW          ipa;            /* SNTP サーバーIP アドレス  */
        TMO         tmo;            /*  タイムアウト設定  */
        UH          devnum;         /*  デバイス番号  */
        UH          port;           /*  ポート番号  */
        UB          ipv;            /* IP バージョン  */
        UB          stt;            /*  受信データ Stratum フィールド値  */
} T_SNTP_CLIENT;
```

8.2 定数とマクロ

(1) IP アドレス

ADDR_ANY	0 の IP アドレス
IP_VER4	IP バージョン 4

(2) ポート番号

PORT_ANY	0 のポート番号

(3) IP プロトコル

IP_PROTO_TCP	TCP プロトコル
IP_PROTO_UDP	UDP プロトコル
IP_PROTO_ICMP	ICMP プロトコル

(4) ネットワーク・インタフェース制御

NET_IP4_CFG	IP アドレス、サブネットマスク等の設定と確認
NET_IP4_TTL	TTL の設定と確認
NET_BCAST_RCV	ブロードキャストの受信の設定と確認
NET_MCAST_JOIN	マルチキャストグループへの参加
NET_MCAST_DROP	マルチキャストグループからの脱退
NET_MCAST_TTL	マルチキャスト送信で使用する TTL の設定

(5) ソケットのパラメータ

SOC_IP_TTL	ソケットの TTL の設定と確認
SOC_IP_TOS	ソケットの TOS の設定と確認
SOC_TMO_SND	snd_soc のブロッキングタイムアウトの設定と確認
SOC_TMO_RCV	rcv_soc のブロッキングタイムアウトの設定と確認
SOC_TMO_CON	con_soc のブロッキングタイムアウトの設定と確認
SOC_TMO_CLS	cls_soc のブロッキングタイムアウトの設定と確認
SOC_IP_LOCAL	ローカルホストの IP アドレスとポート番号を取得
SOC_IP_REMOTE	リモートホストの IP アドレスとポート番号を取得
SOC_CBK_HND	コールバック関数の登録
SOC_CBK_FLG	コールバックイベントの指定
SOC_RCV_PKT_INF	受信パケット情報を取得

(6) ソケットの接続モード

SOC_CLI	リモートホストに接続する (能動接続)
SOC_SER	接続を待ち受ける (受動接続)

(7) ソケットの終了モード

SOC_TCP_CLS	ソケットを切断する。（接続を終了する）
SOC_TCP_SHT	送信処理のみを無効にする。受信は可能。

(8) ソケットの中止モード

SOC_ABT_CON	con_soc()の中止
SOC_ABT_CLS	cls_soc()の中止
SOC_ABT_SND	snd_soc()の中止
SOC_ABT_RCV	rcv_soc()の中止
SOC_ABT_ALL	すべてのソケットの処理の中止

(9) コールバックイベント

EV_SOC_CON	con_soc()をノンブロッキングモードにする。
EV_SOC_CLS	cls_soc()をノンブロッキングモードにする。
EV_SOC_SND	snd_soc()をノンブロッキングモードにする。
EV_SOC_RCV	rcv_soc()をノンブロッキングモードにする。

8.3 エラーコード一覧

マクロ名	値	内容
E_NOSPT	-9	未サポート機能
E_PAR	-17	パラメータエラー
E_ID	-18	不正 ID 番号
E_NOMEM	-33	メモリ不足
E_OBJ	-41	オブジェクト状態エラー
E_NOEXS	-42	オブジェクト未生成
E_QOVR	-43	キューイングオーバフロー
E_RLWAI	-49	待ち状態の強制解除
E_TMOUT	-50	ポーリング失敗またはタイムアウト
E_CLS	-52	待ちオブジェクトの状態変化
E_WBLK	-57	ノンブロッキング受付
E_BOVR	-58	バッファオーバフロー
EV_ADDR	-98	デフォルト G/W 未設定

8.4 API 一覧

API 名	
A) ネットワーク・インタフェース	
net_ini	TCP/IP プロトコルスタックの初期化
net_cfg	ネットワーク・インタフェースのパラメータ設定
net_ref	ネットワーク・インタフェースのパラメータ参照
net_acd	IP アドレス重複検出
B) ネットワークデバイス制御	
net_dev_ini	ネットワークデバイスの初期化
net_dev_cls	ネットワークデバイスの解放
net_dev_ctl	ネットワークデバイスの制御
net_dev_sts	ネットワークデバイスの状態取得
C) ソケット	
cre_soc	ソケットの生成
del_soc	ソケットの削除
con_soc	ソケットの接続
cls_soc	ソケットの切断
snd_soc	データの送信
rcv_soc	データの受信
cfg_soc	ソケットのパラメータ設定
ref_soc	ソケットのパラメータ参照
abt_soc	ソケット処理の中止
soc_ext	ソケット処理の一斉停止
D) ネットワークアプリケーション	
dhcp_client	DHCP クライアントの開始
ftp_server	FTP サーバーの開始
ftp_server_stop	FTP サーバーの停止
http_server	HTTP サーバーの開始
http_server_stop	HTTP サーバーの停止
CgiGetParam	CGI 引数の解析
CgiGetParamN	CGI 引数の解析
CookieGetItem	Cookie ヘッダの解析
HttpSendText	テキストコンテンツの送信
HttpSendFile	ファイルコンテンツの送信
HttpSendResponse	指定コンテンツの送信
HttpSetContent	HTTP サーバ制御情報の送信バッファ追記
HttpSetContentKpa	送信バッファに HTTP Keep-Alive ヘッダ付与
HttpSetContentCookie	送信バッファに Set-Cookie ヘッダ付与
HttpSendBuffer	指定バッファ内容の送信
dns_get_ipaddr	ホスト名から IP アドレスの取得
dns_get_name	IP アドレスからホスト名の取得
dns_query_ext	DNS クエリの発行
dhcp_bind	DHCP リース情報の取得
dhcp_renew	DHCP リース情報の有効期間延長
dhcp_rebind	DHCP リース情報の再割り当て申請

API 名	
dhcp_reboot	DHCP クライアントの再起動
dhcp_release	DHCP リース情報の解放
dhcp_inform	DHCP オプションの取得
ping_client	ICMP Echo 要求の送信と応答の受信
sntp_client	NTP 時刻の取得
E) その他	
htons	16 ビット値をネットワークバイトオーダーへ変換
ntohs	16 ビット値をホストバイトオーダーへ変換
htonl	32 ビット値をネットワークバイトオーダーへ変換
ntohl	32 ビット値をホストバイトオーダーへ変換
ip_aton	ドット表記の IPv4 アドレス文字列を 32 ビット値に変換
ip_ntoa	32 ビット値の IPv4 アドレスをドット表記の IPv4 アドレス文字列に変換
ip_byte2n	IPv4 アドレスの配列を 32 ビット値に変換
ip_n2byte	IPv4 アドレスの 32 ビット値を配列に変換
arp_set	静的 ARP エントリの設定
arp_ref	ARP キャッシュの参照
arp_req	ARP リクエストの送信
arp_clr	ARP キャッシュのクリア
arp_del	ARP エントリの削除
net_atoi	文字列を int 型の数値変換
net_atol	文字列を long 型の数値変換
net_itoa	int 型の数値を文字列変換
net_strncasecmp	文字列の比較(英大文字・小文字同一視)
net_strcmp	文字列の比較
net_strncmp	文字列の比較
net_strcpy	文字列のコピー
net_strlen	文字列長の取得
net_strncat	文字列の連結
net_strcat	文字列の連結
net_strchr	文字の検索
net_strstr	文字列の検索
net_strcasestr	文字列の検索 英大文字・小文字同一視
net_strncpy	文字列の n 文字コピー

8.5 リソース一覧

8.5.1 カーネルオブジェクト

(1) Ethernet デバイスドライバで使用しているカーネルオブジェクト

オブジェクト	オブジェクトID	用途
タスク	ID_TASK_ETH_SND	Etherドライバ送信タスク（スタックサイズ: 1024Byte)
タスク	ID_TASK_ETH_RCV	Etherドライバ受信タスク（スタックサイズ: 1024Byte)
タスク	ID_TASK_PHY0_LINK	PHYドライバ制御タスク（スタックサイズ: 512Byte)
タスク	ID_TASK_PHY1_LINK	PHYドライバ制御タスク（スタックサイズ: 512Byte)
イベントフラグ	ID_FLG_ETH_RX_MAC	Etherドライバイベントフラグ
イベントフラグ	ID_FLG_ETH_TX_MAC	Etherドライバイベントフラグ
イベントフラグ	ID_FLG_PHY_STS	Ether ドライバイベントフラグ
イベントフラグ	ID_FLG_SYSTEM	Ether ドライバイベントフラグ
メールボックス	ID_MBX_ETH_SND	Ether ドライバメールボックス
メールボックス	ID_MBX_ETH_MEMPOL	Ether ドライバメールボックス

(2) TCP/IP プロトコルスタック(μNet3 互換)で使用しているカーネルオブジェクト

オブジェクト	オブジェクトID	用途
タスク	ID_TASK_TCP_TIM	R-IN32用TCP/IPスタック時間管理タスク
セマフォ	ID_SEM_TCP	プロトコルスタックリソース制御セマフォ

(3) TCP/IP プロトコルスタック(BSD 互換)で使用しているカーネルオブジェクト

オブジェクト	ID	用途
タスク	ID TSK_BSD_API	BSD Wrapper タスク
タスク	ID_LO_IF_TSK	ループバックデバイスタスク
メールボックス	ID MBX_BSD_REQ	BSD Wrapper タスク間通信
メールボックス	ID_LO_IF_MBX	ループバックデバイスタスク間通信

注 TCP/IP プロトコルスタック(μNet3 互換)で使用しているカーネルオブジェクトも使用されます。

(4) メモリ管理で使用しているカーネルオブジェクト

オブジェクト	オブジェクトID	用途
メールボックス	ID_MBX_ETH_MEMPOL	メモリ管理

8.5.2 Hardware ISR

表 8.1 TCP/IP スタックで使用している Hardware ISR

Hardware ISR要因	動作	用途
PHY0_IRQn	set_flg()	PHY ドライバ
PHY1_IRQn	set_flg()	PHY ドライバ
ETHTXDMA_IRQn	set_flg()	Ethernet ドライバ送信処理
ETHTXDERR_IRQn	set_flg()	Ethernet ドライバ送信処理
ETHTX_IRQn	set_flg()	Ethernet ドライバ送信処理
ETHTXFIFO_IRQn	set_flg()	Ethernet ドライバ送信処理
ETHTXFIFOERR_IRQn	set_flg()	Ethernet ドライバ送信処理
ETHRXDMA_IRQn	set_flg()	Ethernet ドライバ受信処理
ETHRXFIFO_IRQn,	set_flg()	Ethernet ドライバ受信処理
ETHRXDERR_IRQn	set_flg()	Ethernet ドライバ受信処理
ETHRXERR_IRQn	set_flg()	Ethernet ドライバ受信処理

改訂記録	R-IN32M4-CL3 シリーズ ユーザーズ・マニュアル TCP/IP スタック編

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2021.08.31	-	R-IN32M4-CL3 シリーズ版としてリリース

[メ　モ]

R-IN32M4-CL3 シリーズ ユーザーズ・マニュアル
TCP/IP スタック編

発行年月日　2021年08月31日　Rev.1.00

発行　　　ルネサス エレクトロニクス株式会社
〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

R-IN32M4-CL3 シリーズ
ユーザーズ・マニュアル TCP/IP スタック編

ルネサス エレクトロニクス株式会社