User's Manual

RZ/T1 グループ

μNet3/BSD ユーザーズマニュアル

・RZ/T1

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：R01US0204JJ0200
発行年月：2020.11.1

ルネサス エレクトロニクス
www.renesas.com

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。

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、BSDインタフェースライブラリの機能を理解し、それを用いた応用システムを設計するユーザを対象にしています。

このマニュアルは、BSDインタフェースライブラリの機能、動作、使用方法を理解していただくことを目的としています。

このマニュアルを読むにあたっては、マイクロコンピュータに関する一般知識を必要とします。

本LSI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。

目　次

1. はじめに

μNet3/BSD は、μNet3上でBSDアプリを動作させるためにBSDインタフェースを提供するものです。μNet3/BSD を使用することにより、Linuxや BSDで動作するソケットアプリは、μNet3上でシームレスに動作可能となります。

本ドキュメントではμNet3/BSDの使用方法や制約事項について説明します。

2. 仕様概要

2.1 POSIX仕様での位置づけ

µNet3/BSD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「4. 提供API」を参照してください。またµNet3/BSDを使用することでアプリケーションは、BSDソケットAPIと、µNet3独自APIを使用することができます。

2.2 µNet3との違い

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

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

2.3 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µNet3/BSDが提供するAPI、structure、マクロには独自接頭辞“unet3_”が冠してあります。

アプリケーションは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標準のシンボル名をこれら独自接頭辞付きのシンボルに置き換わります。そのため 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µNet3/BSDで動作します。なお、本書での表記は可読性を考慮してPOSIX標準シンボルを用いています。

3. 全体構成

3.1 モジュール構成

µNet3/BSDを構成するモジュールブロックを図3.1に示します。

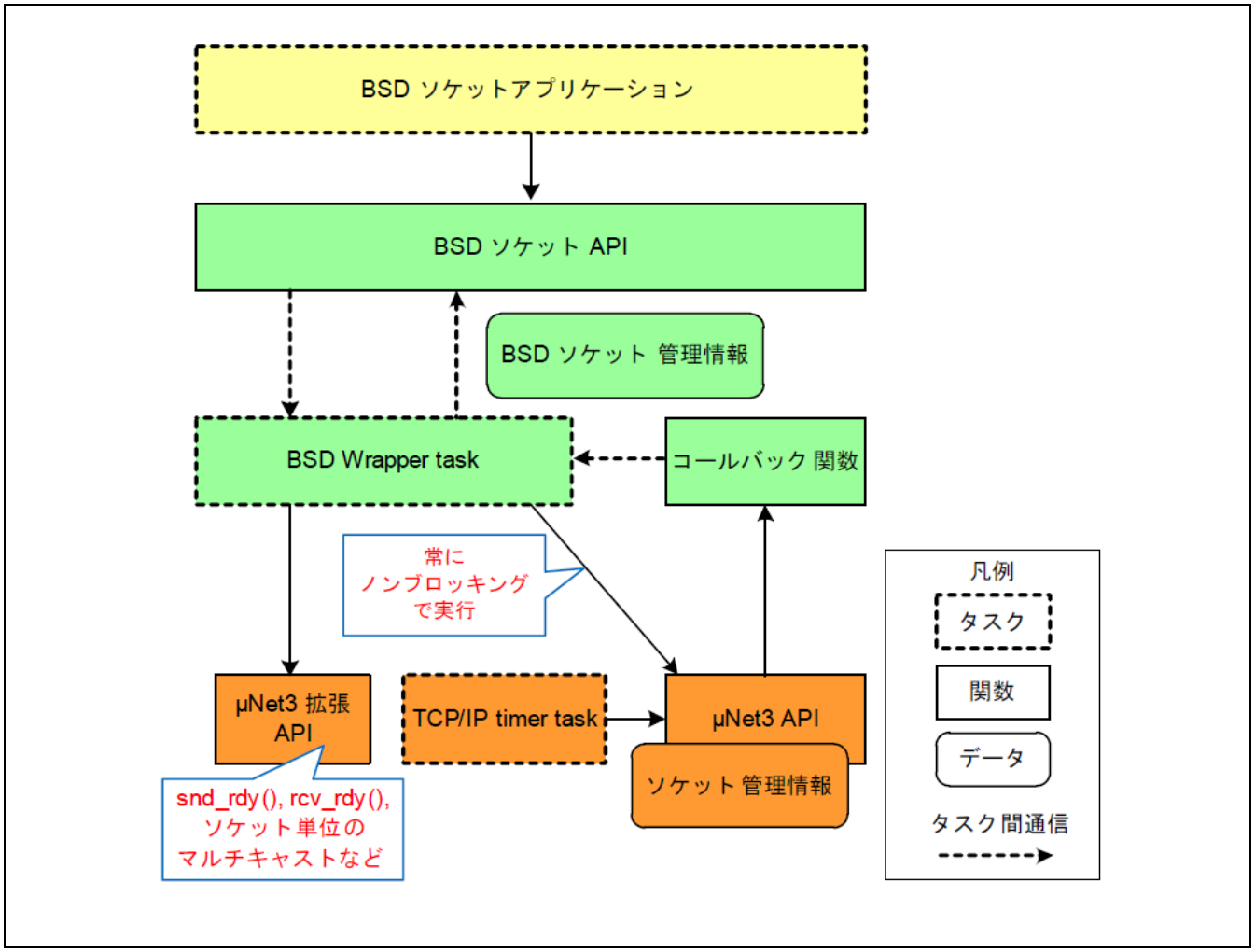

図 3.1

3.2　ヘッダ構成

µNet3/BSDでは POSIX 準拠のヘッダファイルをダミーとして扱います。これらのダミーファイルがµNet3/BSDオリジナルの公開ヘッダファイル(unet3_socket.h)をincludeします。表3.1にµNet3/BSDが提供するヘッダファイルの一覧を示します。

表 3.1　ヘッダファイル一覧

POSIX 準拠のヘッダファイル（ソケット関連）	
ヘッダファイル名	主な用途
arpa/inet.h	数値として IP アドレスを操作する機能の定義
netinet/in.h	AF_INET と AF_INET6 アドレスファミリ。インターネット上で広く使われ、IP アドレスと TCP/UDP のポート番号が含まれる
netinet/ip.h	IP レベルのオプションや IP パケットに関する定義
netinet/tcp.h	TCP レベルのオプションや TCP パケットに関する定義
sys/socket.h	BSD ソケットの中核となる関数とデータ構造
net/if.h	インタフェースに関する定義
POSIX 準拠のヘッダファイル（システム関連）	
sys/errno.h	エラー番号の定義
sys/ioctl.h	ioctl 関連の定義など
sys/select.h	select、fd_set 関連の定義など
sys/time.h	timeval 型の定義など
sys/times.h	timeval 型の定義など
sys/unistd.h	UNIX 標準に関する標準ヘッダ
µNet3/BSD のオリジナルヘッダファイル	
unet3_cfg.h	ユーザーコンフィグレーションの定義
unet3_socket.h	ソケット API などを定義した公開ヘッダファイル
unet3_sys.h	BSD プラットフォーム固有の型やマクロを定義したヘッダファイル。BSD アプリが include するシステム系ヘッダファイルはこのファイルに集約されます
unet3_wrap.h	内部制御用
bsd_param.h	内部制御用

3.3 ソースファイル一覧

µNet3/BSDのソースファイル一覧を示します。

アプリケーションはbsdフォルダ配下の *.cプログラムを組み込む必要があります。

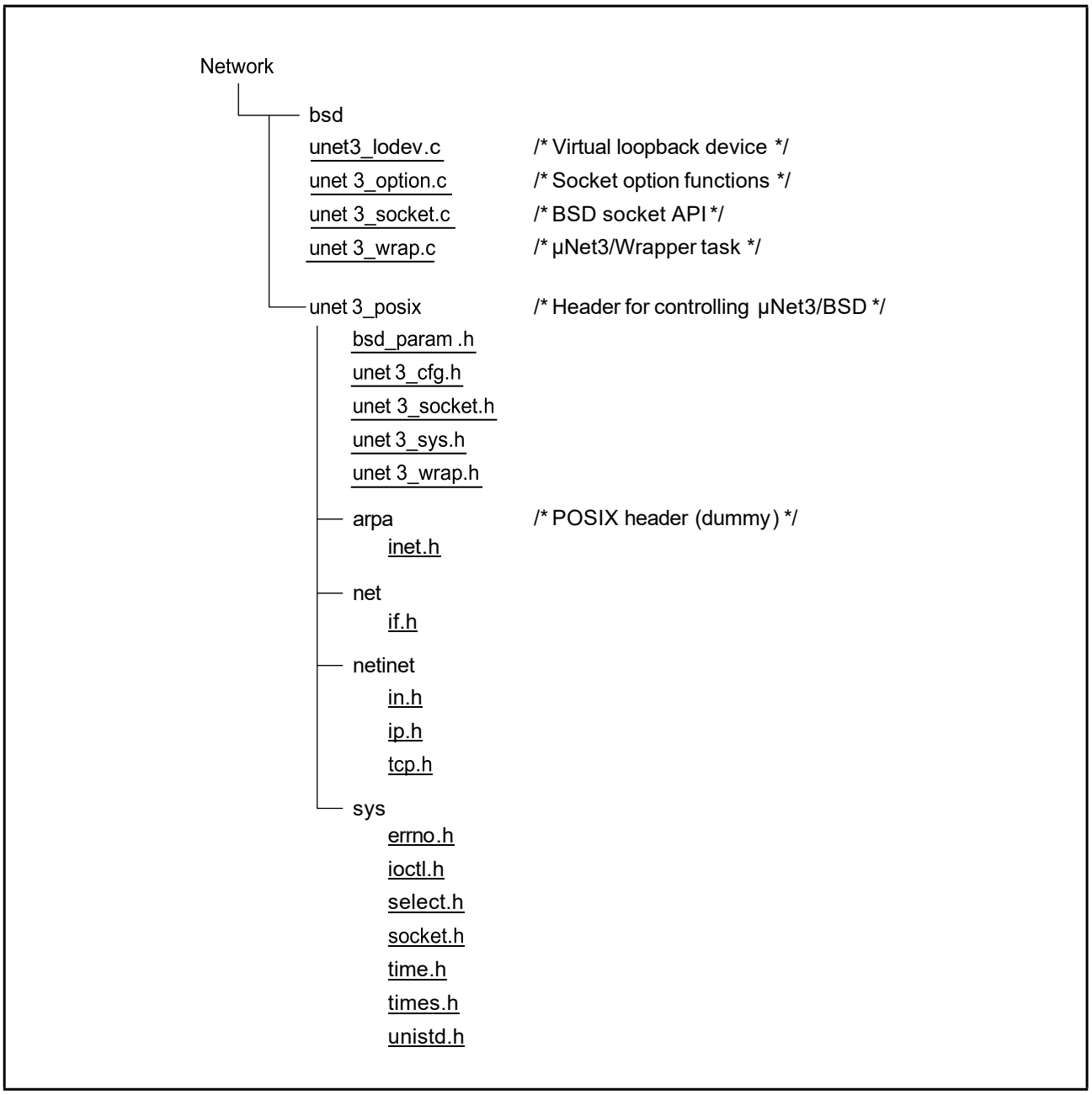

図 3.2

4. 提供API

4.1 API一覧

µNet3/BSDが提供するAPI一覧を表4.1に示します。

表 4.1 API一覧

API	機能	インクルードヘッダ
unet3_bsd_init	µNet3/BSD を初期化する	“sys/socket.h”
get_errno	タスクごとの 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う	“sys/socket.h”
setsockopt	ソケットのオプションの設定を行う	“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”

4.2　API詳細

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。したが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。

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	すで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

- 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。
- 受信動作を開始するには、対象のソケットを指定してbind()をあらかじ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。

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	サポートしていないプロトコル（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。

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（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struct sockaddr_in*型で設定します。
- 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。

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	すでに接続済み
	listenしているソケット
EALREADY	すでに接続中
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）を行わないため、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。

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
EPIPE	ソケットFDが不正な状態
EDESTADDRREQ	宛先未設定（UDPソケット）
ENOTCONN	接続されていない（TCPソケット）
EACCES	ブロードキャスト送信が未許可のため、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注1

注1.　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。

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
EPIPE	ソケットFDが不正な状態
EDESTADDRREQ	宛先未設定（UDPソケット）
ENOTCONN	接続されていない（TCPソケット）
EACCES	ブロードキャスト送信が未許可のため、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注1

注1.　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。

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
EPIPE	ソケットFDが不正な状態
ENOTCONN	接続されていない（TCPソケット）
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。したがって、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。
- 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。

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
EPIPE	ソケットFDが不正な状態
ENOTCONN	接続されていない（TCPソケット）
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。したがって、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。
- 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。
- 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。

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。

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。

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。（ただしすで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。

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 sockaddr_inのサイズ（16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()

  なお、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。

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sockaddr_inのサイズ（16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。

getsockopt（ソケットのオプションの取得を行う）

【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の指定可能です。
- 各オプションレベルで取得可能なオプション名は、「5.1 オプション一覧」を参照してください。

setsockopt（ソケットのオプションの設定を行う）

【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の指定可能です。
- 各オプションレベルで設定可能なオプション名は、「5.1 オプション一覧」を参照してください。

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「6.1 ノンブロッキング設定」を参照してください。
- 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。

inet_addr（インターネット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。

inet_aton（インターネット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。

inet_ntoa（インターネット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文字列は静的に割り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。

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μNet3のデバイス名(gNET_DEV[index-1].name[8])によって設定されます。

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

【errno】

ENXIO	存在しないインタフェースindex

- インタフェース名はμNet3のデバイス名 (gNET_DEV[index-1].name[8]) によって設定されます。

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

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。

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 *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】

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.　ソケットオプション

5.1　オプション一覧

setsockopt()/getsockopt()APIで取得・設定が可能なオプション一覧を「表5.1 オプション一覧」に示します。
それ以外のオプションを指定した場合、setsockopt()/getsockopt() は -1 を返却します。

表 5.1　オプション一覧

SOL_SOCKETレベル		
オプション名	値の型	用途
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 注 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		
オプション名	値の型	用途
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_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		
オプション名	値の型	用途
TCP_KEEPCNT 注 1	int	TCP Keep-Alive プローブ回数の設定。SET のみ。
TCP_KEEPIDLE 注 1	int	TCP Keep-Alive 起動無通信間隔の設定。SET のみ。
TCP_KEEPINTVL 注 1	int	TCP Keep-Alive プローブ送信間隔の設定。SET のみ。
TCP_MAXSEG	int	TCP パケットの MSS 値の設定。GET/SET。

注 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。

6.　サポート機能

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6.1 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。

なお、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μNet3/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 6.1　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ない）
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。

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。

なお、μNet3/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。

6.3　エラー処理

µNet3/BSDでは、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値は更新されます。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

- タスクごとの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。

6.4 errno一覧

使用するコンパイラによって、errnoの定義値は変わることがあります。

【errno定義パターン1 – 対象コンパイラ】

- Arm 社 RealView Developer Suite
- IAR 社 Embedded Workbench (EWARM)
- TI 社 Code Composer Studio
- GNU C Compiler

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
EACCES	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	すでに実行中の処理
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	アドレスがすでに使用中
EADDRNOTAVAIL	49	アドレスが使用できない
EISCONN	56	ソケットがすでに接続されている
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

【errno定義パターン 2 – 対象コンパイラ】

- ルネサスエレクトロニクス社 CubeSuite+ *

errno	値	説明
ENXIO	*	インタフェースが存在しない
EBADF	*	ソケット FD が無効
ENOMEM	*	メモリ不足
EACCES	*	要求された処理に対するアクセス拒否
EFAULT	*	パラメータ異常
ENODEV	*	システム内で致命的（もしくは不明）な異常
EINVAL	*	パラメータ誤り
EPIPE	*	ソケットオブジェクトが不正
EAGAIN	*	ブロッキング処理を実行した
EALREADY	0x1025	すでに実行中の処理
EDESTADDRREQ	0x1027	宛先設定が必要
EPROTONOSUPPORT	0x102B	機能が未サポート
EAFNOSUPPORT	0x102F	アドレスファミリが未サポート
EADDRINUSE	0x1030	アドレスがすでに使用中
EADDRNOTAVAIL	0x1031	アドレスが使用できない
EISCONN	0x1038	ソケットがすでに接続されている
ENOTCONN	0x1039	ソケットが接続されていない
ETIMEDOUT	*	タイムアウト
ECONNREFUSED	0x103D	接続が拒否された

- 制限事項
  - エラー格納定義名 errno は使用できません。
    CubeSuite+ の標準ライブラリに同名の errno 定義があり、µNet3/BSD 側で errno 定義名を使用出来ません。
    代わりに unet_errno を参照してください。

  - 一部のerrnoの定義値はコンパイラ側のものを使用します。(*)
    µNet3/BSDで使用する errno の値定義名が CubeSuite+ 側に存在する場合、そちらを使用しています。

7. BSDアプリの実装

7.1 ソースコード

µNet3/BSDを使用するアプリケーションは、Network/bsd/ 直下の4 つのソースコードをプロジェクトに取り込む必要があります。（「3.3 ソースファイル一覧」参照）

また µNet3本体となるライブラリも BSD 向けのライブラリ (uNet3BSDxxxx.lib) をリンクする必要があります。

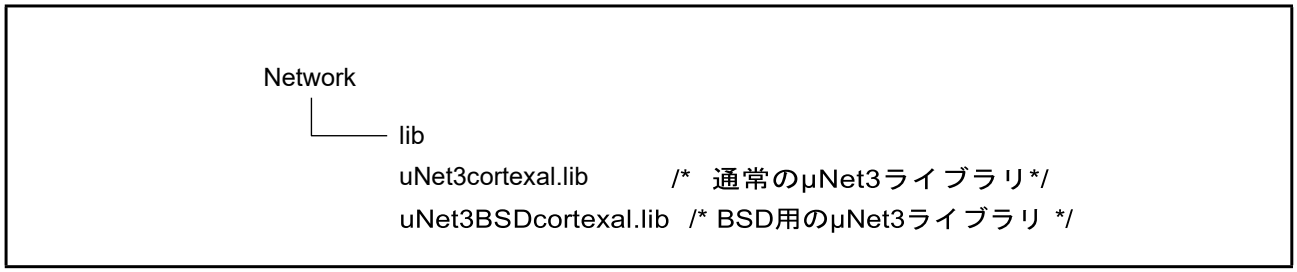

図 7.1

7.2 インクルードパス

ヘッダファイルは POSIX 準拠のファイルも含めて、Network/bsd/unet3_posixフォルダ配下にあります。µNet3/BSDを使用するアプリケーションはインクルードパスの設定を追加する必要があります。

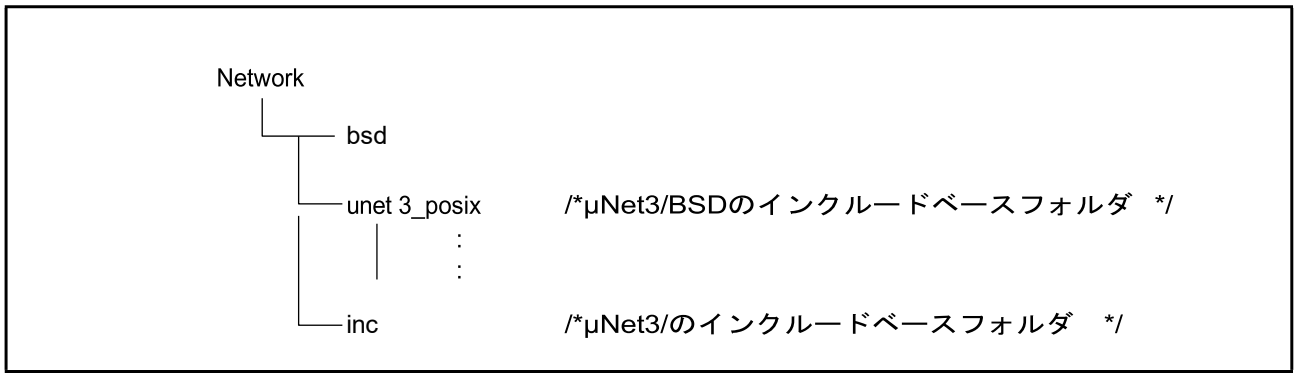

図 7.2

7.3　コンフィグレーション

µNet3/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あらかじめunet3_cfg.hにマクロ定義する必要があります。

最大ソケット数

#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、この値はµNet3のソケット最大数 (CFG_NET_SOC_MAX)と同じ値でなければなりません。

アプリケーションタスク数

#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µNet3/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ください。

7.4　リソース定義

µNet3/BSDを使用するアプリケーションは、µNet3/BSDを動作させるために必要なリソースを用意する必要があります。リソースはµNet3/BSDが情報を管理するためのテーブルです。

BSDソケット管理テーブル

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。

エラー番号管理テーブル

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。

7.5　カーネルオブジェクト

µNet3/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

7.6　初期化

アプリケーションはソケットAPIを使用する前に unet3_bsd_init() 関数を呼び出して µNet3/BSD モジュールを初期化する必要があります。なお、µNet3/BSD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

【errno】

E_SYS	カーネルオブジェクトの初期化処理に失敗

8. 付録

8.1 対応コンパイラ

µNet3/BSDでは次のコンパイラでの動作を保証します。

- Arm 社 RealView Developer Suite
- IAR 社 Embedded Workbench (EWARM)
- TI 社 Code Composer Studio
- GNU C Compiler
- ルネサスエレクトロニクス社 CubeSuite+

注. コンパイラによる制限事項あり

8.2 サンプルアプリ

Sample フォルダに µNet3/BSD を使ったサンプルアプリを収録しています。なお、これらのサンプルプログラムは、POSIX 環境 (Linux) でも動作します。

- APIコンソール (sample_sockcmd.c)

コマンドプロンプト（UART 接続）からソケットAPI名と各種パラメータを入力してAPIを実行することができます。詳しくは Readme_command.txt を参照してください。

8.3 コンパイラの制限事項

µNet3/BSDでは一部のコンパイラにおいて制限事項があります。

- ルネサスエレクトロニクス社 CubeSuite+
  - エラー格納定義名 errno は使用できません。
    CubeSuite+ の標準ライブラリに同名のerrno定義があり、µNet3/BSD 側でerrno定義名を使用出来ません
    代わりに unet_errno を使用してください。

改訂記録	RZ/T1 µNet3/BSD ユーザーズマニュアル

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00		—	初版発行
2.00	2020.11.1	4. 提供 API	
		11, 35, 36	getifaddrs()/freeifaddrs() 関数の追加
		12 ～ 36	「4.2 API 詳細」部分の追記、修正
		19, 20	MSG_PEEK オプション追加にともない、説明内容を追記
		28	FIONREAD オプション追加にともない、説明内容を追記
		5. ソケットオプション	
		37	TCP　Keep-Alive に関するソケットオプションの補足説明を追記
		37	SO_BROADCAST オプション設定時の動作変更により「5.1 オプション一覧」の「SO_BROADCAST」の説明を変更
		37	SO_REUSEADDR オプション追加にともない、「5.1 オプション一覧」に説明を追記
		6. サポート機能	
		40	「6.4 errno 一覧」の記述を変更
		40, 41	「6.4　errno 一覧」に「EACCES」と「EHOSTUNREACH」の説明を追記
		8. 付録	
		45	対応コンパイラに CubeSuite+ を追加
		45	「8.3 コンパイラの制限事項」を追加

RZ/T1 グループ　ユーザーズ・マニュアル
μNet3/BSD編

発行年月日　2013年06月27日　Rev.1.00
　　2020年11月01日　Rev.2.00

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）

RZ/T1 グループ ユーザーズ・マニュアル

μNet3/BSD 編

ルネサス エレクトロニクス株式会社

R01US0204JJ0200