

TOPPERS中級実装セミナー

(H8/3069F版:ネットワーク)

基礎編:2日目

TOPPERSプロジェクト
教育ワーキング・グループ

本教材の利用条件

NEXCESS中級コース02 リアルタイムOSを用いたソフトウェア設計技術

Copyright (C) 2006 by 名古屋大学 組込みソフトウェア技術者人材養成プログラム

Copyright (C) 2006 by 本田晋也, 高田広章, 山本雅基

上記著作権者は、以下の(1)～(4)の条件を満たす場合に限り、本コンテンツ(本コンテンツを改変・翻訳したものを含む、以下同じ)を使用・複製・改変・翻訳・再配布(以下、利用と呼ぶ)することを無償で許諾する。

- (1) この枠内の著作権表記等が、そのままの形でコンテンツ中に含まれていること。
- (2) 本コンテンツを再配布する場合には、再配布の形態等を、以下のウェブサイトから報告すること。
<http://www.nces.is.nagoya-u.ac.jp/NEXCESS/REPORT/>
- (3) 本コンテンツを改変・翻訳する場合には、コンテンツを改変・翻訳した旨の記述を、コンテンツ中に含めること。また、改変・翻訳者の著作権表記等は、この枠内の著作権表記等とは別に行うこと。
- (4) 本コンテンツの利用により直接的または間接的に生じるいかなる損害からも、上記著作権者を免責すること。

※ 本コンテンツの一部は、文部科学省 科学技術振興調整費により、名古屋大学 組込みソフトウェア技術者人材養成プログラム(NEXCESS)の一環として作成しました。

※ 本コンテンツ中に記載されている商品名やサービス名などは、各社の商標または登録商標です。

本ドキュメントに関して

1. 著作権に関しての表記

＜TOPPERS中級実装セミナー基礎編(H8-3069F版：ネットワーク) 2日目＞

Copyright (C) 2005-2006 by 高田広章 名古屋大学

Copyright (C) 2005-2006 by 山本雅基 名古屋大学

Copyright (C) 2005-2006 by 本田晋也 名古屋大学

上記著作権者は、以下の (1)～(3) の条件を満たす場合に限り、本資料（本資料を改変したものを含む。以下同じ）を使用・複製・改変・再配布（以下、利用と呼ぶ）することを無償で許諾する。

- (1) 本資料を利用する場合には、上記の著作権表示およびこの利用条件が、そのままの形で資料中に含まれていること。
- (2) 本資料を改変する場合には、資料を改変した旨の記述を、資料中に含めること。ただし、TOPPERSプロジェクトの活動の一環として本資料を改変する場合には、資料を改変した旨の記述を含める必要はない。
- (3) 本資料の利用により直接的または間接的に生じるいかなる損害からも、上記著作権者およびTOPPERSプロジェクトを免責すること。

2. 本ドキュメントに関するご意見・ご提言・ご感想・ご質問等がありましたら、TOPPERSプロジェクト事務局までE-Mailにてご連絡ください。

3. 本ドキュメントの内容は、内容の改善や適正化の目的で予告無く改定することがあります。

本ドキュメントでは、Microsoft社のClip Art Galleryコンテンツを使用しています。

TRONは"The Real-time Operating system Nucleus"の略称です。ITRONは"Industrial TRON"の略称です。
μITRONは"Micro Industrial TRON"の略称です。TOPPERS/JSPはToyohashi Open Platform for Embedded Real-Time System/Just Standard Profile Kernelの略称です。」

本ドキュメント中の商品名及び商標名は、各社の商標または登録商標です。

スケジュール

■ 1日目

- | | |
|-------------------|-------|
| 1. 開発環境の確認 | 0.5時間 |
| 2. 基本プログラミング | 1.5時間 |
| 3. ITRON仕様の概要 | 0.5時間 |
| 4. ITRON API解説 | 2.5時間 |
| 5. カップラーメンタイマーの開発 | 1.0時間 |

■ 2日目

- | | |
|------------------------|-------|
| 1. TCP/IPの概要 | 1.0時間 |
| 2. ITRON TCP/IP API 仕様 | 1.0時間 |
| 3. TINET (TCP/IPスタック) | 0.5時間 |
| 4. TCPサーバープログラミング | 2.0時間 |
| 5. TCPクライアントプログラミング | 1.0時間 |
| 6. まとめ | 0.5時間 |

TCP/IPの基礎

1. TCP/IPの基礎
2. ITRON TCP/IP仕様
3. TINET (TCP/IPスタック)
4. TCPサーバープログラム
5. TCPクライアントプログラム
6. まとめ

ネットワークプロトコル

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

2006/10/07

TOPPERSプロジェクト認定

6



ネットワーク上で通信を行う場合、意図したデータが正しく送受信できるように、データの形式、フォーマット、意味、手順などを、取り決める必要があります。このような、通信時の規約や手順をネットワークプロトコル、または、プロトコルと呼びます。ネットワークプロトコルといっても種類しかないわけではなく、使用目的や利用目的によって、いくつもの規約があります。プロトコルのなかでも、特定の目的に従って作成されたプロトコル群をプロトコルスイートと言います。プロトコルスイートには、OSI、TCP/IP、IPX/SPX、NetBIOS/NetBEUI、AppleTalkなど種々のものがあります。

TCP/IPは標準化団体が規定したものではなく、研究者や開発者を中心とした利用者が議論と開発を繰り返す中から、さまざまな改良が行われ、多くの機器で使われるようになったプロトコルスイートです。特に、ワークステーション、パソコンの普及により、インターネットの標準プロトコルスイートとなりました。IPはInternet Protocolの略で複数のネットワークが接続している状況で、ネットワークを介して相手のコンピュータにパケットを送るプロトコルです。TCPはアプリケーションプログラムとIPの間の仲介を行い、送信側からのデータが確実に届いていることを保証する信頼性確保のためのプロトコルです。

TCP/IPが普及したのは、バークレイ版のUNIXであるBSD UNIXに採用されたことが大きな要因であると言われています。

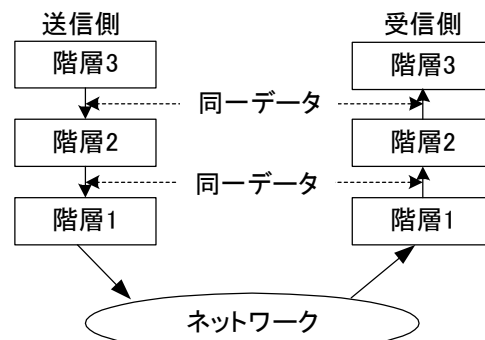
TCP/IPに関する論議はIETF (Internet Engineering Task Force)で行われています。IETFで行われた論議が実用的な段階になったら、RFC文書にまとめられて公開が行われます。

階層化の必要性

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

階層化原理

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



2006/10/07

TOPPERSプロジェクト認定

7



巨大なひとつのプロトコルで、通信の規約や手順を制御することは困難です。ネットワークにおける階層化のモデルとしては、OSIの参照モデルが有名です。OSIではプロトコルの策定を7階層に分けた通信モデルを作成しました。層単位に役割をもつプロトコルを策定し、各層は送信側と受信側で同じレベルのデータを扱う形になっています。また、下の層ほどハードウェアに近く、上の層ほどアプリケーションに近い構成となっています。

上位層	⑦アプリケーション層
	⑥プレゼンテーション層
	⑤セッション層
下位層	④トランスポート層
	③ネットワーク層
	②データリンク層
	①物理層

TCP/IPでは5階層のモデルとなっています。

TCP/IP インターネットのプロトコル階層は5階層

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

2006/10/07

TOPPERSプロジェクト認定

8

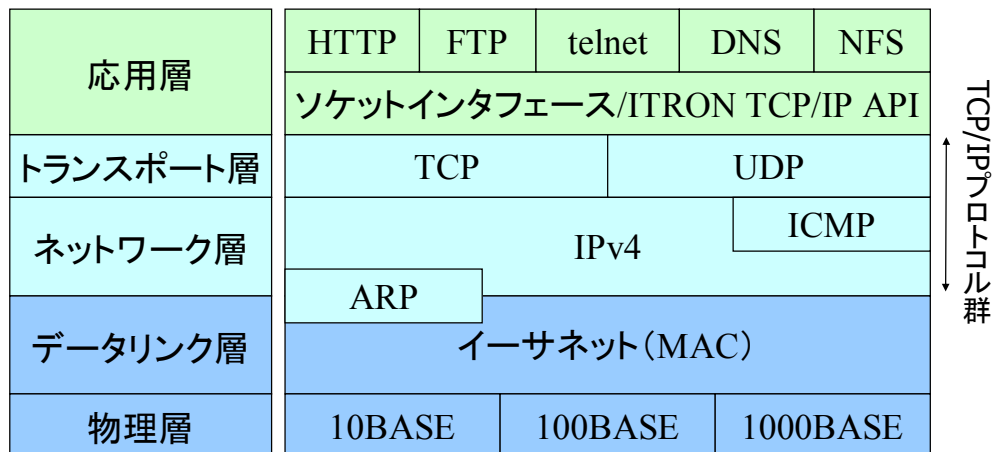


通常はTCP/IPプロトコルスイートでは5階層の構成となっています。2のデータリンク層と1のハードウェア層をひとつのネットワークインターフェース層として、4階層とする場合もあります。

OSI参照モデルとの比較にて説明を行うと、データリンク層では、LAN用がイーサネット、広域接続用にはアナログモデムやISDNを使ったPPPが広く使われています。ネットワーク層はIP、トランスポート層にはTCPとUDPが使われます。TCPはコネクション指向のプロトコルで信頼性重視であるのに対して、UDPはコネクションレスの軽快な通信を目的としています。

TCP/IPのアプリケーション層は、OSI参照モデルのセッション層、プレゼンテーション層、アプリケーション層に当たります。アプリケーション層の種々のプロトコルがあります。

階層図 (TCP/IPv4, イーサネット)



2006/10/07

TOPPERSプロジェクト認定

9



上記がIPv4のイーサネットを対象とした断層図です。

ICMPはIPパケットが受信された時に、正常に受信されず途中でエラーが発生した場合、送り元にエラーを伝えるプロトコルです。ARPはデータリンク層でイーサネットを使用する場合、IPアドレスをイーサネットのMACアドレス(物理アドレス)に変換するプロトコルです。

MACアドレスはMedia Access Control Addressの略で48ビットの長さを持ちます。このアドレスは通常はネットワークインターフェースカードの製造時ROMに書き込みが行われます。MACアドレスの先頭24ビットはイーサネット機器のメーカーやベンダの固有のコードで、後の24ビットは機器固有のシリアル番号です。

アプリケーション層のプロトコル

- HTTP : Webシステム用プロトコル
 - インターネットの爆発的普及のけん引役
- DNS : ホスト名とIP アドレスの解決
- telnet : リモート端末
- FTP : ファイル転送
- SMTP, POP3 : 電子メール
- NFS, SMB : ファイルシステム
- SNMP : ネットワーク管理

2006/10/07

TOPPERSプロジェクト認定

10



アプリケーション層の主なプロトコルについて、説明を行います。

HTTPは**Hyper Text Transfer Protocol**の略で、WWWのページの取得要求やHTMLファイル、画像ファイルの転送を行うプロトコルです。HTTP自体はページやファイルの取得要求や転送が主な仕事です。表示などはWWWブラウザが行います。

DNSは**Domain Name System**の略です。パソコンやワークステーションにはIPアドレス以外に、ホスト名がつけられます。通常の通信ではホスト名を対象に行いますので、ホスト名とIPアドレスを変換する仕組みが必要です。このような名前と番号情報を管理するサービスをネームサービスといい、このサービスを行う仕組みがDNSです。

telnetはTCP/IPネットワークを介して、他のコンピュータを遠隔利用するプロトコルです。

FTPは**File Transfer Protocol**の略で、TCP/IPネットワーク間のコンピュータ同士でファイルの転送を行うためのプロトコルです。

SMTPは**Simple Mail Transfer Protocol**の略で、相手のメールサーバーにメールを届けるためのプロトコルです。

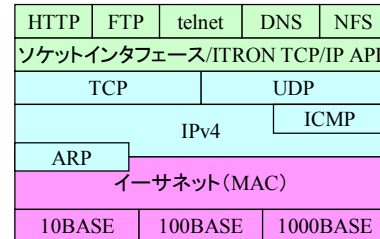
POPは**Post Office Protocol**の略で、メールサーバーからメールを読み出すためのプロトコルです。

NFSは**Network File System**の略、SMBは**Server Message Block**の略です。共に、ネットワークを介して他のコンピュータのローカルストレージを自分のファイルシステムとして扱う為のプロトコルですが、NFSはサン・マイクロシステムズ社が、SMBはマイクロソフト社が中心開発したものです。

SNMPは**Simple Network Management Protocol**の略です。その名の通り、ネットワーク上の機器の管理を行うプロトコルです。

データリンク層・物理層：イーサネット

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



2006/10/07

TOPPERSプロジェクト認定

11



イーサネット(Ethernet)は1973年にXerox社のパルアルト研究所で開発されたもので、1980年にXerox社、DEC社、Intel社によってDIXイーサネットとしてまとめられ、その後、IEEEにより、IEEE802.3規格として標準化されました。イーサネットの特徴はCSMA/CD方式によりパケットを送りあう方式です。これにより、ケーブル上で衝突が発生すると再送信することになります。初期のイーサネットはバス型の接続方式でしたが、高速化によりスター型に切り替わりました。

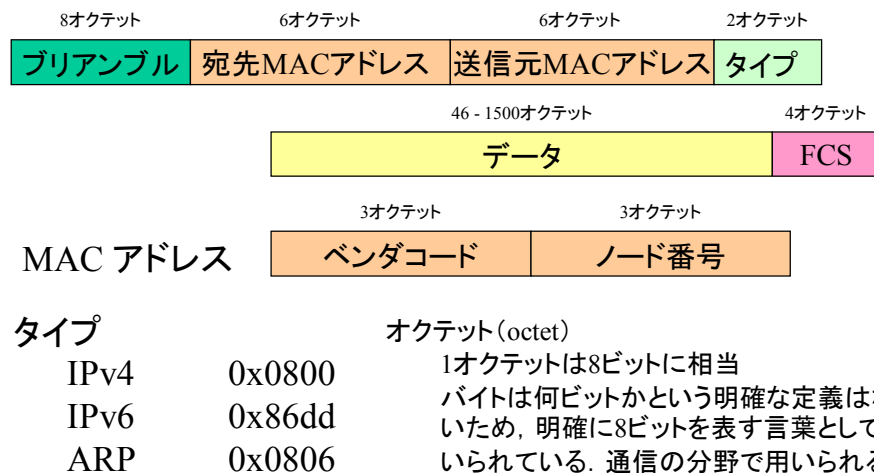
イーサネットの表記方法を以下に示します。

例えば、10Base-T以下の組み合わせで内容が示されます。

項目	意味
伝送速度	10:10Mbps
	100:100Mbps
	1000:1000Mbps(1Gbps)
伝送方式	Base:ベースバンド
	Broad:ブロードバンド
伝送メディア	-T:ツイストペアケーブル
	-F:光ファイバー
	-SX:光ファイバー(短波長)
	-LX:光ファイバー(長波長)
	-CX:同軸ケーブル

イーサネットフレーム

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



2006/10/07

TOPPERSプロジェクト認定

12

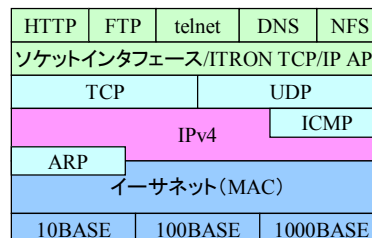


イーサネット上を通信するデータは、イーサネットフレームと呼ばれ、形式はフレームフォーマットとして決まっています。通信の世界では8ビットのデータをオクテットという単位で呼びます。コンピュータで使用する1バイトは必ずしも8ビットではない場合がありますので、明確にするために、この単位を用いています。イーサネットのフレームは64オクテット以上、1518オクテット以下の可変長です。

上記はDIXイーサネットのフレームフォーマットで、IEEE802.3ではタイプとして使用している部分が長さの設定となっています。プリアンブルは以下に続く信号がフレームであることを示すための区切り信号です。その後に宛先と送信元のMACアドレスがあり、最後にFCS (Frame Check sequence) と呼ばれる、フレームのデータが正しいかどうかを確かめるビット列で終わります。

ネットワーク層 : IP (Internet Protocol)

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



2006/10/07

TOPPERSプロジェクト認定

13



IPは通信したいデータをパケットという単位に分割し、受信先のIPアドレスを持った機器にデータを届ける機能を持ちます。このとき、複数のネットワークをまたがった環境でもデータを届けなければなりません。複数のネットワークを転送するうちに、パケットはルータによって捨てられる可能性もあり、最善努力型 (ベストエフォート) のプロトコルと呼ばれています。

IPは、複数のネットワークが相互接続されている環境で、パケットを送信するために、IPアドレスのネットワークアドレスを見て、経由すべきネットワークを決定していきます。このような機能を経路選択またはルーティングと言います。経路情報は、経路選択表に記載されます。この表は手動でも設定できますが、最近ではルーティングプロトコルを用いて、自動的に経路選択表をつくるのが主流となっています。

IPアドレスはIPv4の場合、32ビット (約43億個) であり、地球の人口の約60億人より少ない。有効に利用するために、DHCPにより、IPアドレスの自動割当やグローバルアドレスとプライベートアドレスを使い分ける等の工夫が行われています。

グローバルアドレスは世界中からアクセスが可能なアドレスで、インターネット上で重複してはならないアドレスを指します。プライベートアドレスは閉じたネットワーク内で使用するアドレスで以下の範囲が設定されています。

- 10.0.0.0～10.255.255.255
- 172.16.0.0～172.31.255.255
- 192.168.0.0～192.168.255.255

IPv4 データグラム

- データの基本転送単位
- ヘッダとデータ領域で構成されている



2006/10/07

TOPPERSプロジェクト認定

14



TCP、UDPからのデータは必要に応じて分割され、IPによりIPパケットが作られます。上の図はIPv4のデータグラムです。IPヘッダとデータで構成されています。以下にヘッダのフィールドの説明を行います。

Ver: IPのバージョンで、IPv4の場合、4が入ります。

ヘッダ長: IPヘッダそのものの大きさです。単位は4オクテット(32ビット)なのでオプションがない場合は5になります。

TOS: サービスタイプです。8ビット構成で以下の意味を持ちます。

ビット0、1、2(優先度)、ビット3(低遅延要求)、ビット4(高スループット要求)、ビット5(高信頼性要求)、ビット6(低コスト要求)、ビット7(未使用)

IPデータグラム長: IPヘッダとデータを含めた長さで、単位はオクテットです。

フラグメントID: 複数のIPパケットが一連のデータであることを示す識別子で、一連のパケットは同じIDを持ちます。

フラグ: パケットの分割に関する情報で、ビットごとに意味を持ちます。ビット0は未使用であるが0でなければならない。ビット1はさらに分割できるかのフラグ、ビット2は最後のパケットかどうかのフラグです。

フラグメントオフセット: フラグメント化されたIPパケットがオリジナルのどの位置にあったかを示す情報、8オクテット単位。

寿命: IPパケットのネットワーク上の生存時間、単位はhops。

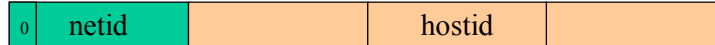
上位プロトコル: TCP、UDP、ICMPなどの上位プロトコルを示します。

ヘッダチェックサム: ヘッダのチェックサム値。

IPv4 のクラス型アドレス

- 各ホストには、32ビットのアドレスが割り当てられる

クラスA（ネットワーク7ビット、ホスト24ビット）



クラスB（ネットワーク14ビット、ホスト16ビット）



クラスC（ネットワーク21ビット、ホスト8ビット）



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

2006/10/07

TOPPERSプロジェクト認定

15



IPv4のIPアドレスはクラス分けされています。設計当初は以下の3つに分けられました。

- ・クラスA
 - 用途: 大規模ネットワーク
 - 範囲: 0.0.0.0～127.255.255.255
 - ひとつのネットワークに対応できるホストアドレスはホスト部オール0とオール1を除いた16,777,214台
- ・クラスB
 - 用途: 中規模ネットワーク
 - 範囲: 128.0.0.0～191.255.255.255
 - ネットワークの数は16,384、ひとつのネットワークに対応できるホストアドレスはホスト部オール0とオール1を除いた65,534台
- ・クラスC
 - 用途: 小規模ネットワーク
 - 範囲: 192.0.0.0～255.255.255.255
 - ネットワークの数は2,097,152、ひとつのネットに対応できるホストの数は254台

その後、マルチキャストアドレスとしてクラスD、将来の拡張用にクラスEが用意されました。

問題点として、クラス分けが固定的で、クラスBが不足しています。ホストアドレスもクラスCでは少ないが、クラスBでは多すぎる。また、IPアドレスは地理とは無関係で、経路情報が増える傾向があります。

応用層プログラムの識別:ポート番号

IPはコンピュータ間の通信

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

2006/10/07

TOPPERSプロジェクト認定

16



IPアドレスは通信相手のコンピュータを特定するために使用します。ポート番号はコンピュータ上のソフトウェアを特定する番号です。ポート番号のうち登録されているポート番号をWell-Known Portといいます。

ポート	TCP/UDP	内容	ポート	TCP/UDP	内容
7	TCP,UDP	エコー	80	TCP	HTTP(WWW)
21	TCP	FTP	110	TCP	POP
23	TCP	TELNET	119	TCP	NNTP
25	TCP	SMTP	123	UDP	NTP
43	TCP	WHOIS	143	TCP	IMAP2
53	TCP,UDP	ドメイン	161	UDP	SNMP
67	TCP	DHCPサーバー	179	TCP	BGP
68	TCP	DHCPクライアント	220	TCP	IMAP3
70	TCP	Gopher	443	TCP	HTTPS(WWW)
79	TCP	FINGER			

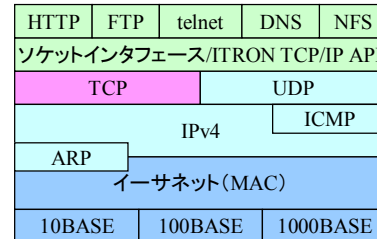
Windows用のポート番号定義ファイルはC:\windows\system32\drivers\etc\servicesにあります。

ポート番号の区分け

ポート番号	用途
0～1023	システム用のWell Known ポート番号
1024～49151	ユーザ用のWell Known ポート番号
49152～65535	動的割り当てもしくはプライベート用ポート番号

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

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



2006/10/07

TOPPERSプロジェクト認定

17



TCPはコネクション指向のプロトコルで、通信を行う場合、TCP同士でコネクションを張り、アプリケーションはTCP間の通信路に送信、受信を行うだけで通信が行える手順を提供します。通信路を閉じるには、コネクションの切断を行います。TCPを使って通信を行うと、通信路のデータは連続して流れるストリームデータとして扱われます。

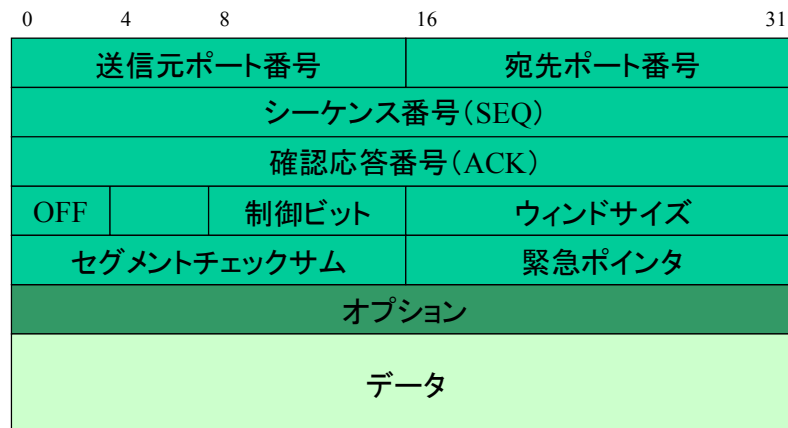
TCPは確認応答を行います。送信後、確認応答を受け取ってから次のデータを送ります。データが届かない場合は確認番号を使って再送要求できるばかりではなく、確認応答が届かない場合は、時間待ちを行って、再送信ができます。

TCPの送受信は送信側と受信側でシーケンス番号を取り交わし、TCPセグメントが進むごとにシーケンス番号をインクリメントを行う順序付け制御を行います。また、受信可能なデータサイズはウィンドウサイズで取り交わすフロー制御を行います。

コンピュータ上で複数のネットワークアプリケーションを実行する場合、複数の通信路が必要となります。TCPは複数の通信路を区別するために、IPアドレスとポート番号の組をエンドポイントといい、送信と受信のエンドポイントの組み合わせで通信路の区別を行っています。

TCP セグメント

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



2006/10/07

TOPPERSプロジェクト認定

18



TCPセグメントはTCPヘッダとデータで構成されます。

送信元ポート番号: データの送信元のポート番号。

宛先ポート番号: データの宛先のポート番号。

シーケンス番号: データが送信データストリームの中のどの位置にあるかの番号。

確認応答番号: 次に受信したいデータのオクテット位置。正常に送受信が行われる場合、確認応答で返ってくる確認応答番号と、その後送信を行うシーケンス番号は一致します。

OFF: TCPヘッダの先頭からデータの先頭までのオクテット数。

制御ビット: 通常のデータ送信ではなく特別な送信であることのビットで6ビットあります。

URG (Urgent)	緊急処理
ACK (Acknowledgement)	確認応答
PSH (Push)	転送強制
RST (Reset)	仮想通信路の強制切断
SYN (Synchronize)	同期通信
FIN (Fin)	転送終了

ウィンドウサイズ: 確認応答の際、次に送信可能なデータサイズ。

セグメントチェックサム: TCPデータが正しく届いたかどうかを判定するチェックサム情報

緊急ポインタ: 制御ビットのURGが1の場合有効、データの先頭からどこまでが緊急データかを示す。

コネクション確立

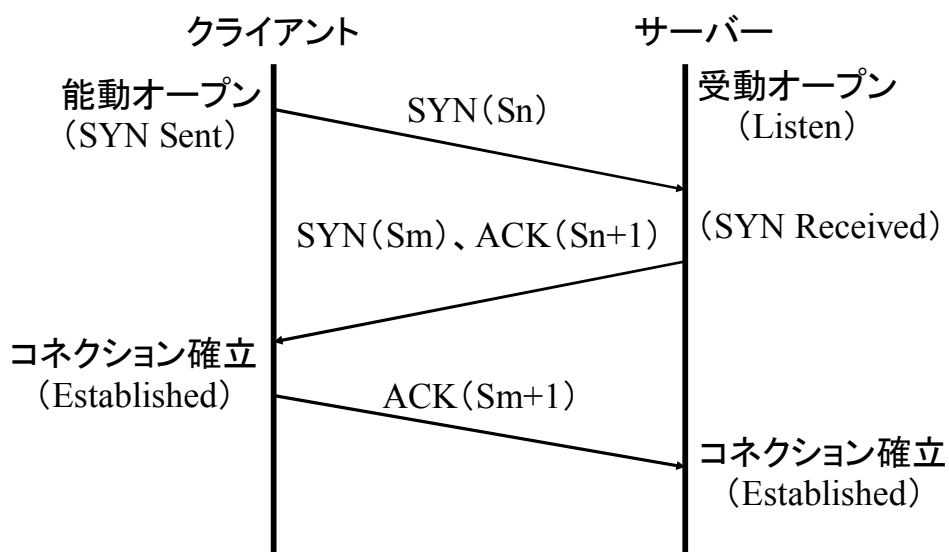
- 3 ウェイハンドシェーク
 - クライアントとサーバー間でセグメントを3回交換
- 受動(Passive)オープン
 - サーバー側、入ってくるコネクションを受け入れる
- 能動(Active)オープン
 - クライアント側、サーバーにコネクションを要求する
- お互いの初期シーケンス番号を交換

TCPでコネクションを確立する場合の手順を3ウェイハンドシェークといいます。まず、能動(Active)オープン側が最初のSYNセグメントを受動(Passive)オープン側に送ります。受動オープン側はこれを受け取ると、SYN+ACKセグメントを送り返します。能動オープン側がこれを受信すると能動オープンでのコネクションが確立します。能動オープン側がACKセグメントを受動オープン側にかえし、受動オープン側でコネクションが確立します。3つのセグメントでコネクションの確立を行います。(次のスライド)

能動オープンを行う側がクライアント、受動オープンを行う側がサーバーとなります。

最初の能動オープンのSYNセグメントのシーケンス番号が初期シーケンス番号で、受動オープンからのSYN+ACKセグメントの確認応答番号が能動オープンの初期シーケンス番号+1がセットされ、受動オープンの初期シーケンス番号をシーケンス番号にセットします。3つめの能動オープンの確認応答番号には、能動オープンの初期シーケンス番号+1がセットされ、シーケンス番号には、能動オープンの初期シーケンス番号+1がセットされます。このような手順で、お互いの初期シーケンス番号の交換を行います。

コネクション確立



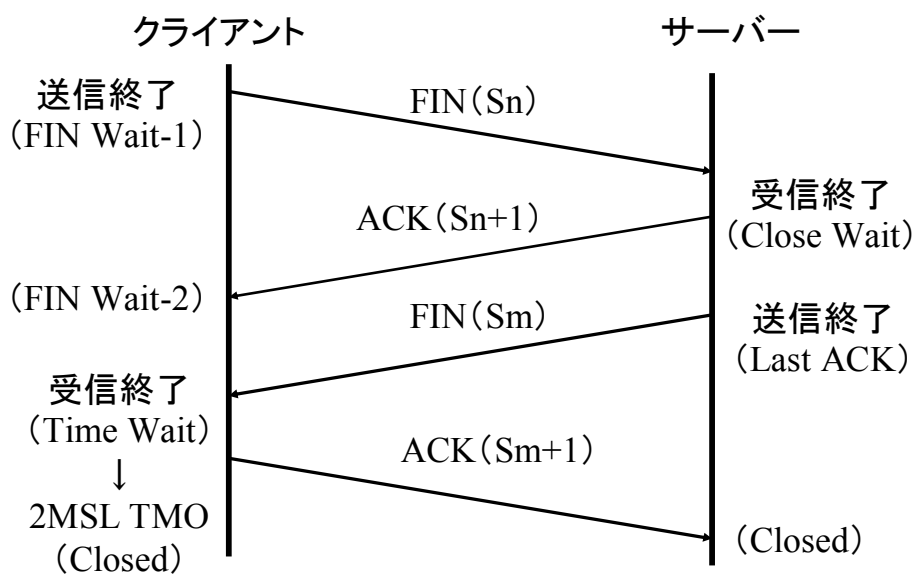
コネクション切断

- 送信するデータが無い事を相手に通知
 - クライアント側から切断するが多い
 - WWW はサーバーから切断
 - 2MSL タイムアウトが必要
 - MSL (Maximum Segment Lifetime): ネットワーク内でのセグメントの最大残存時間. RFC793では2分とされている
- FIN セグメントと ACK セグメントを交換
- ハーフクローズ
 - 相手の送信が終わっても、送信可能

コネクションの切断は、送信するデータがないことを相手に伝え、通信路を開放します。切断をサーバー側から行うか、クライアント側から行うかは応用アプリケーションにて違いますが、クライアント側から切断するが多いようです。切断要求を行った側は2MSLの間、待ちを行わなければならない約束事になっています。MSL (Maximum Segment Lifetime) はパケットがネットワークに滞留できる最大時間を示します。すなわち、MSLを越えてパケットが到着しない場合、パケットが何らかの理由で喪失したものとみなすことができます。つまり、切断のために、FINを発行してから往復として2MSL待機すれば、もう、受信データはないと判定します。

切断の手順はFINセグメントとACKセグメントの交換により行われます。また、完全に切断を行わず。送信データがなく、送信のみ停止するハーフクローズという状態もあります。

コネクション切断



2006/10/07

TOPPERSプロジェクト認定

22



高信頼性通信

- シーケンス番号 (SEQ)
 - データのオフセット
- 確認応答番号 (ACK)
 - 受信済のデータオフセット+1
つまり、次に受信を期待するデータのオフセット
- 送信と同時に再送タイマーをスタート
- 再送は最大12回
- 再送毎にタイムアウト値は2倍

TCPの通信路での通信では、TCPブロック上にシーケンス番号 (SEQ) と確認応答番号 (ACK) があります。送信側は現在の送信データの中のどの位置にあるかのオフセットを入れて送ります。受信側はこれを受け取ると、受け取れたデータの次のオフセットを確認応答番号に入れて確認応答として送り返します。正常に送受信が行われると、送信側のデータのシーケンス番号と確認応答の確認応答番号は一致します。

送信が正常に受信側に伝えられなかった場合は、確認応答番号は送信側のシーケンス番号より小さな値となります。送信側はシーケンス番号をその位置に戻して、データの再送を行います。

送信側は送信と同時に、再送タイマーをスタートします。そのタイマーのタイムアウトが発生しても確認応答が返らない場合、送信側は送信データを再送します。再送は最大12回行われ、再送毎のタイムアウト値は2倍に設定されます。

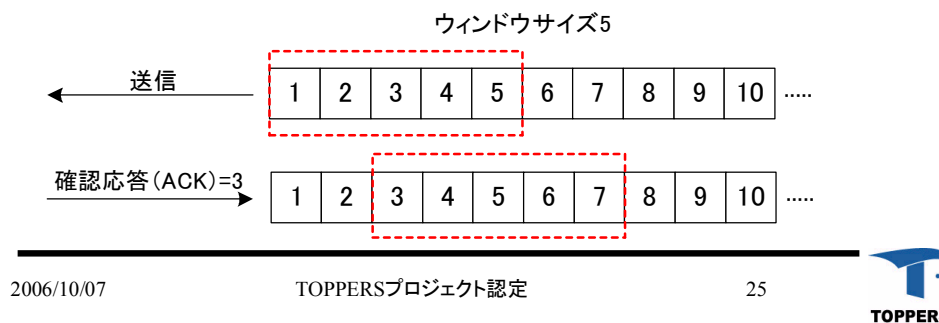
再送タイムアウトの決定

- 往復時間 (RTT) の測定
 - あるセグメントに付けた SEQ に対して、確認応答 ACK が帰ってくるまでの時間
- 再送タイムアウト値 (RTO)
 - $$RTO = \alpha RTO + (1 - \alpha) RTT$$
 - RTT が急に増減しても、過敏に反応しない
 - FreeBSD で実際に使用される式は複雑

往復時間 (RTT、Round Trip Time) は、あるセグメントに対応した、確認応答が返ってくるまでの時間です。TCP では、RTT を定期的に測定を行います。再送に使われるタイムアウト値を再送タイムアウト値 (RTO、Retransmission TimeOut) といいます。この値は RTT から算出します。

高効率通信とフロー制御

- スライディングウィンド制御
 - 受信側が、受信可能データ量を送信側に伝える
- 最初は1セグメント(1MSS)送信する
 - MSS(Maximum Segment Size) : 最大セグメント長
- スロースタート
 - 応答が来る度にウィンドサイズを1セグメントずつ増やす
- 倍数減少輻輳回避
 - セグメントが喪失するとウィンドウサイズを半分にする(最小1セグメントまで)



複数の通信路にて同時に送受信が発生する場合など、受信側はデータを受けきれない状態が発生します。そのために、受信側は受信可能データ量をTCPヘッダのウィンドウサイズを使って送信側に伝えることができます。

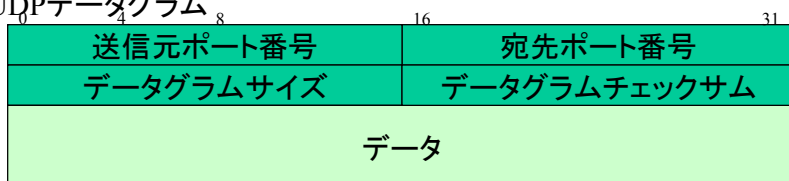
ひとつのTCPパケットで、IPの分割が発生させずに通信が可能な最大サイズをMSS(Maximum Segment Size)といいます。例えば、イーサネットの場合、最大1500オクテットのデータを送れますので、ここからIPヘッダとTCPヘッダのサイズを除いて、1460オクテットがMSSとなります。最初の送信は確実なMSSサイズの送信を行います。受信用のウィンドウサイズは最小のサイズを設定しておき、確認応答が戻るたびにウィンドウサイズを大きくしていきます。ウィンドウサイズは一括して受信できるデータ量ですので、多くのセグメントを一括して送って、確認応答を少なくした方が送信効率が向上していくわけです。

ウィンドウは確認応答が来るたびに、既に受け取られたセグメントから新しく受け取れるようになったセグメントにずれていきますから、スライディングウィンドウ制御と呼ばれます。(上の図の説明)

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

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

HTTP	FTP	telnet	DNS	NFS
ソケットインタフェース/ITRON TCP/IP API				
TCP			UDP	
IPv4				ICMP
ARP	イーサネット (MAC)			
10BASE	100BASE	1000BASE		



2006/10/07

TOPPERSプロジェクト認定

26



UDPはコネクションレス型のプロトコルで、TCPのようなコネクションを行う必要はありません。そのため、応用プログラムがデータを送信するたびに送信相手を指定しなければなりません。UDPはポート番号をもつ、応用プログラム間の通信となります。そのため、オーバヘッドが少なく、高速な通信が可能となります。その反面、ネットワーク上のデータの消失や、順序の入れ替わり等の問題が発生した場合は、応用プログラムで対応を行わなければなりません。

UDPのデータグラムはUDPヘッダとデータから構成されます。UDPヘッダの構成は以下の通りです。

送信元ポート番号: データの送信元のポート番号。

宛先ポート番号: データの宛先のポート番号。

データグラムサイズ: UDPヘッダとデータの合計サイズ、オクテットサイズ。

データグラムチェックサム: UDPの擬似ヘッダを使ったチェックサム値、UDPで唯一の信頼性確保のデータです。

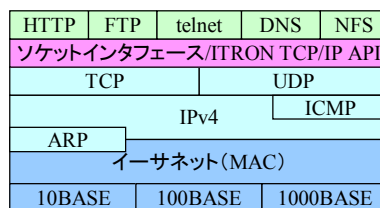
これからのネットワーク層 : IPv6

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

IPv6はTCP/IPネットワークの拡大にともない、128ビットのIPアドレスをもつ、新しいIPプロトコルです。

ソケットインタフェース

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



2006/10/07

TOPPERSプロジェクト認定

28

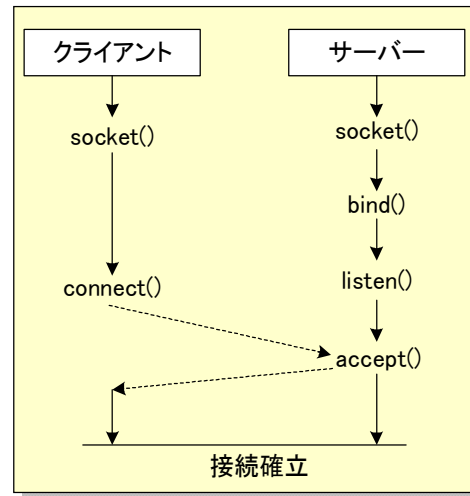


ソケットインターフェースは、BSD UNIXにおいて応用アプリケーションとTCP/IPプロトコル間のインターフェースで、多くのOSで採用されています。Windowsでは、WinSockと呼ばれ、UNIX用のソケットインターフェースとは若干仕様が異なります。

ソケットは応用アプリケーションがデータを送受信するための仕組みを抽象化したオブジェクトです。送受信の仕組みは、ファイルをオープンしデータを読み書きするようにプログラムが可能です。

ソケットインターフェース : API

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



2006/10/07

TOPPERSプロジェクト認定

29



ソケットインターフェースを用いたコネクションの確立の手順について説明を行います。サーバー側が`socket()`関数でコネクション用のTCPソケットを作成します。続いて`bind()`関数でソケットにポート番号を割り当てます。`listen()`関数にて、受動オープンを行い待ち状態になります。クライアント側は`socket()`関数でクライアント用のソケットを生成します。`connect()`関数で能動オープンを行います。サーバーをクライアントからの能動オープンにより、待ち状態を抜けます。`accept()`関数にて通信用のソケットが生成され、このソケットによりデータの送受信を行います。

ITRON TCP/IP API 仕様

1. TCP/IPの基礎
2. ITRON TCP/IP仕様
3. TINET (TCP/IPスタック)
4. TCPサーバープログラム
5. TCPクライアントプログラム
6. まとめ

組込みシステムとTCP/IP

- 組込みシステムのインターネット接続
 - PDA、プリンタ、DVDレコーダー、プロジェクター
- インターネットプロトコルの流用
 - ! インターネットプロトコルである必然性はなく、他に適切なプロコルがあればそれでもよい
 - ➡ パソコン側で既存のソフトウェアが利用可能
 - 例) Webブラウザ
 - 万能ユーザーインタフェースツール
 - ➡ パソコン側のユーザーインタフェースツールを作成する必要がない

TCP/IPの重要性はさらに高まる傾向

2006/10/07

TOPPERSプロジェクト認定

31



ITRON TCP/IP Ver.1.00.01はITRON上のTCP/IPプロトコルAPIとして、Embedded TCP/IP技術委員会の活動の結果を、1998年にITRON専門委員会の審査を経て、「組込みシステムのための標準TCP/IP API」として認定されました。このAPIはアプリケーション層とトランスポート層の間のインターフェースについての標準となっています。認定の時点で、組込みシステム用のTCP/IP APIとしてはUNIX用のソケットインターフェースを使用したソフトウェア部品がいくつかありましたが、この仕様は、ITRONをプラットフォームとしている小規模な組込みシステムを構築する上で、最適な仕様となっています。

組込みシステムでも、PDA、プリンタ、DVDレコーダー、プロジェクターなどインターネットやLANに接続して使用する機器が増えてきています。この場合、現在のインターネット接続機器の主体であるパソコンに接続でき、パソコン側のプログラムを変更することがなく、既存のソフトウェアが利用できます。これにより組込みシステムの価値を向上させることができます。

組込みシステムの特徴

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

2006/10/07

TOPPERSプロジェクト認定

32



ここで、組込みシステムで最適なネットワークインターフェースを考察するために、組込みシステムの特徴についてまとめなおしてみましょう。ITRONが対象とする小規模な組込みシステムについては以下のような特徴がありました。

- 1) 組込み機器自体が特定の機能をもち、行すべき処理が決まっているものが多い
- 2) 製品としての価格帯域があり、ハードウェア資源に対する制約が厳しい。特に、メモリの制約やより高い性能を求められることが多い
- 3) リアルタイム性を求められる機器を制御する必要がある
- 4) 製品化時点で、高い信頼性が求められる

組込みシステムのメモリ管理方式には、大別すると動的メモリ管理方式と静的メモリ管理方式があります。動的メモリ管理とは、組込みシステムが起動後、共有のメモリ管理領域から必要に応じてメモリを取得、開放することでメモリ管理を行う管理方式です。これに対して静的メモリ管理方式は、コンパイル、リンク(ビルド、生成)時に必要なメモリを決定してしまう管理方式です。ITRONが対象とする組込みシステムでは静的なメモリ管理を行っている場合が多く、これは開発方式が1リンク方式をとる場合が多いからです。システムによって、どうしても動的な管理が必要な場合でも、メモリ管理のポリシーをアプリケーションに任せる方が、最適化しやすい場合が多いです。

TCP/IPプロトコルスタックのAPI

- TCPとUDPのAPIが最も重要

➡ 以下、これに絞って議論

応用層	HTTP	FTP	telnet	DNS	NFS
	ソケットインタフェース/ITRON TCP/IP API				
トランスポート層	TCP		UDP		
ネットワーク層	IPv4			ICMP	
	ARP				
データリンク層	イーサネット(MAC)				
物理層	10BASE		100BASE		1000BASE

2006/10/07

TOPPERSプロジェクト認定

33



TCP/IPプロトコルスタックの層を参照すると、物理層からトランスポート層までは、パソコン側の層と機能対応しており、アプリケーションプログラムでネットワーク機能を実現するために特に意識する必要はありません。ここでは応用層とトランスポート層の間のAPI、特に、TCP/IPでは、TCPとUDPを実現するプログラムとのAPIを標準化することが重要となります。

以降では、TCPとUDPとのAPIに絞って議論を進めます。

ソケットインタフェースとその問題点

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

2006/10/07

TOPPERSプロジェクト認定

34



現状の組込みシステムでは、TCPとUDPのAPIではBSD UNIX用に設計されたソケットインターフェイス、または、RTOSによって、それを最適化したものが広く使われています。ソケットインターフェイスはUNIXを基本OSとして、設計しているため小規模な組込みシステムでは、いくつかの点で不向きであると指摘されています。

1)ソケットインターフェイスでのアドレス形式の指定時、アドレスファミリに準じてIPアドレスやポート番号を指定できる。また、TCPでもUDPでも同等の関数を使用します。組込みシステムではTCP/IPに特化したアドレス指定で十分であり、プロトコルにより最適化した方が良いと考えている。

2)UDPパケットの受信時等、プロトコルスタック内でパケットをキューイングするために動的なメモリ管理が必要となる。静的なメモリ管理でもプロトコルスタックが動作できるようなAPIの拡張が必要。

3)データのコピー回数が多くなる。

4)RTOSのタスクモデルとUNIXのプロセスモデルのシステムコールの違いにより、RTOSへの最適化が難しい。

Embedded TCP/IP技術委員会

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



標準化の必要性

Embedded TCP/IP技術委員会

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

2006/10/07

TOPPERSプロジェクト認定

35



仕様策定を行った1996年前後に、組込みシステム用にTCP/IPのミドルウェアを開発した各々の会社がソケットインタフェースを独自に最適化したAPIを用意し、ネットワークを使用するアプリケーションの互換性に問題が発生する可能性がおきてしまった。そのため、1997年からテンポラリにEmbedded TCP/IP技術委員会が発足し、ITRONプロジェクトにおけるソフトウェア部品のAPIの標準化活動を開始した、1998年5月に活動の成果を「組込みシステムのための標準TCP/IP API仕様」としてまとめ、ITRON専門委員会の審査を経て「ITRON TCP/IP API仕様 (Ver.1.00)」として認定された。

ITRON TCP/IP APIの設計仕様

(a) ソケットインタフェースをベースに

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

(b) APIの理解しやすさ、プログラムしやすさを重視

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

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

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

2006/10/07

TOPPERSプロジェクト認定

36



ITRON TCP/IP APIの設計方針

(a)ソケットインタフェースに慣れているプログラマが多いことや、ソケットインタフェースを用いて開発されたソフトウェアが多いことから、ソケットインタフェースをベースに設計仕様を作成する。また、定義したAPIの上にライブラリをかぶせることにより、ソケットインタフェースと互換のAPIを実現できるように設定する。

(b)複雑な、または、使いにくいAPIはなるべく避ける。API毎のエラーの詳細がわかることが望ましく、APIの理解のしやすさ、プログラミングのしやすさに重点をおいた設計を目指す。

(c)ハードウェア資源(プロセッサ能力、メモリ容量)に対する制約が厳しい。そのため、プログラムやデータサイズを小さく抑え、プロセッサの能力をフルに発揮できることが必要となる。バッファのためのメモリ領域が最小限であること、メモリ不足時の振る舞いをアプリケーションが制御できること、できる限りプロトコルスタック内部で動的なメモリ管理を使う必要がないこと。

ITRON TCP/IP APIの設計仕様

(d) プロトコル毎にそれに最適なAPIを定義

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

(e) リアルタイムシステムへの適用を考慮

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

(f) 静的な設定で十分なものを考慮

- 「静的API」

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

(g) ITRON仕様の作法を踏襲、他のRTOSにも適応可能

- サービスコール、パラメータ、エラーの名称等はITRON仕様の作法に準拠

2006/10/07

TOPPERSプロジェクト認定

37



(d) TCPとUDPのプロトコルごとに最適なAPIを定義する。また、API毎のエラーの詳細がわかることが望ましい。これにより、アプリケーション開発者が、必要な資源量を把握しやすくなる。

(e) 待ち状態に入るサービスコールには、一律タイムアウトとノンブロッキングコールを用意し、リアルタイムシステムの適用を考慮する。

(f) ITRON4.0仕様の「静的API」の設定の対応を行う。また、静的APIで十分なAPI仕様とする。

(g) サービスコール、パラメータ、エラー等の名称は、ITRON仕様の作法に準拠する。また、他のRTOSにも適応可能な作法とする。

ITRON TCP/IP の主概念

- 通信端点
 - ネットワークを介した通信サービスの端点
(ソケットに対応)
 - プロトコルの種類ごとに定義
 - UDPの通信端点 (UDP communication end point)
 - TCPの受付口 (TCP reception point)
 - TCPの通信端点 (TCP communication end point)
- ↑
ソケットインタフェースでは、TCP接続を待ち受けているソケットは、データの送受信には使われない
- 種類ごとにシステム全体でユニークなID番号で識別
- ← (e) (g)の方針

2006/10/07

TOPPERSプロジェクト認定

38



通信端点 (end point)とはソケットインタフェースのソケットに対応します。通信端点はIPアドレスとポート番号で識別され、ITRONカーネル仕様のオブジェクトに相当するものです。通信端点は、プロトコルや機能毎に別種類のもので、システム全体でユニークなID番号で識別されます。

上記の説明の解説を行います。TCPでサーバーを構成する場合、ソケットインタフェースでは、TCP接続を待ち受のためのソケットは用意するが、接続した時点で新しいソケットが生成されるため、データの送受信に使用されることはありません。ITRON TCP/IP API仕様での通信端点は、接続した時点で、接続状態の通信端点に遷移し、そのまま、通信用に使用されます。この点から、通信端点はソケットに比べ、ずっと、実装に近い扱いとなっています。

ITRON TCP/IP の主概念

- ノンブロッキングコール
 - サービスコールの中でブロックされる状況になった場合に、**処理を継続したまま**サービスコールからリターン
 - ↓ *UNIXの非同期I/Oとの違い*
 - 処理が完了した時点でコールバックを用いて通知
 - 処理を途中でキャンセルすることも可能
- コールバック
 - プロトコルスタックからのイベントをアプリケーションに通知する (OS非依存な) 方法
 - 通信端点毎にアプリケーションで定義
 - プロトコルスタック側のコンテキストで実行
 - メモリ保護のないOSを想定
 - ➡ 中でイベントフラグをセットするという使い方も可能

2006/10/07

TOPPERSプロジェクト認定

39



ITRON TCP/IP API仕様では、待ち状態に入る可能性のあるAPIには、タイムアウトとノンブロッキングコールを用意しています。

タイムアウトは一定時間経過しても処理が完了しない場合に、処理をキャンセルしてAPIからリターンする手順です(この場合、APIからはE_TMOUTエラーが返る)。そのため、タイムアウトが起こった場合には、APIを呼び出したことでのオブジェクト状態は変化していないことを原則としています。(APIの機能上、処理のキャンセル時に元の状態に戻せない場合は例外とする)

ノンブロッキングコールの設定は各サービスコールのタイムアウト指定にTMO_BNLKを設定することによりAPIに伝えられます。この設定によりAPIの中の待ち状態に入るべき状況になった場合に、処理を継続したままAPIからリターンします(このとき、APIからはE_WBLKエラーが返る)。そのため、APIからリターンしても処理の実行は継続しており、処理が完了した時点(または、処理がキャンセルされた時点)で、コールバックを用いてアプリケーションプログラムに処理完了を通知します。また、APIからリターンした後にも処理は継続しているため、データ領域へのポインタを渡すAPIをノンブロッキング指定で呼び出した場合、処理が完了するまで、それらの領域を別の目的で使わないようにしてください。

ITRON TCP/IP仕様でのコールバックは、プロトコルスタックで起こった事象をアプリケーションプログラムに伝えるために用いるOS非依存な手法をさします。コールバックを起こす事象は、大きく分けて、ノンブロッキングコールの処理完了通知とその他の事象に分類されます。

コールバックルーチンは、アプリケーションプログラムが定義し、実装依存のコンテキストで実行されます。そのため、どのようなコンテキストでも実行できるように記述することが望ましく、コールバックルーチン内で待ち状態に入ってはいけません。

コールバックルーチンは、各通信端点に対して1つずつ定義されます。コールバックを起こした事象の種類はコールバックルーチンへの引数として渡されます。同一の通信端点に対するコールバックルーチンはネストして呼ばれません。また、TCP受付口はコールバックルーチンは持ちません。

タイムアウト設定の詳細は「ITRON TCP/IP API仕様書」の1. 5. 3節の定数を参照してください。

ITRON TCP/IP の主概念

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

2006/10/07

TOPPERSプロジェクト認定

40



各APIの返値は、ITRON仕様のコンベンションに従っています。エラーが発生した場合には負の値のエラーコード、正常に実行された場合には0または正の値とします。正常実行された場合の返値の意味は各APIごとに定義しています。

エラーコードは、メインエラーコードとサブエラーコードで構成されています。メインエラーコード、サブエラーコード共に負の値とし、それらを組み合わせたエラーコードも負の値とします。メインエラーコードのニーモニックと値及び意味は、ITRONカーネル仕様のエラーコードと同じように標準化します。ただし、ITRONカーネル仕様で足りないエラーコード(E_WBLK, E_CLS, E_BOVR)は追加定義します。サブエラーコードについては実装依存です。以下のエラーコードは全てのAPIでエラーとして返す可能性があります。

- E_SYS システムエラー (プロトコルスタックの内部エラー)
- E_NOMEM メモリ不足
- E_NOSPT 未サポート機能
- E_MACV メモリアクセスエラー

静的APIは、受付口や通信端点を静的に定義できる。

TCPのサービスコールの一覧

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

2006/10/07

TOPPERSプロジェクト認定

41



ITRON TCP/IP API仕様でTCPを制御するために使用する通信端点は、相手からの接続要求を待ち受けするための受付口 (TCP Reception Point, repと略す)と、接続の端点として用いるTCP通信端点 (TCP Communication End Point, cepと略す)を用意しています。TCP通信端点は8つの状態を遷移することにより、接続、データの送受信、終了の各操作を行います。

ソケットインターフェースのread/write処理にあたるデータ送受信APIに加え、データ送受信時、データのコピー回数を減らす可能性のあるより効率的なAPIを用意しています。このAPIを省コピーAPIと呼んでいます。

各APIは必要の度合いに応じて標準機能と拡張機能に分類している。TCP/IPプロトコルの機能を実現するためには、少なくとも標準機能を実装する必要があります。

TCPのサービスコールの一覧

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

クライアント側の接続手順

- ソケットインターフェースとの対比

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

- プログラム例

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

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

2006/10/07

TOPPERSプロジェクト認定

43



ソケットインターフェースで、TCPクライアントプログラムを作成する場合、まず、**socket**関数にてTCPソケットを作成します。クライアントプログラムの場合、IPアドレスが単一の場合プロトコルスタック内で自動設定できますので、あえて**bind**関数で自ノードのアドレス設定を行う必要はありません。**connect**関数でサーバーのIPアドレスとポート番号により特定されるソケットとの間で接続を確立します。

ITRON TCP/IP API仕様では、**socket**関数にあたるのは**tcp_cre_cep**サービスコールでTCP通信端点の生成を行います。**bind**関数と**connect**関数にあたるのは**tcp_con_cep**サービスコールでTCP通信端点を用いて、相手側のIPアドレスとポート番号に対して接続を行います。

TCP_CRE_CEP TCP通信端点の生成

tcp_cre_cep

【標準】
【拡張】

【静的API】

TCP_CRE_CEP(ID cepid, {ATR cepatr, VP sbuf, INT sbufsz
VP rbuf, INT rbufsz, FP callback});

【C言語API】

ER ercd = tcp_cre_cep(ID cepid, T_TCP_CCEP *pk_ccep);

【パラメータ】

ID cepid TCP通信端点ID
T_TCP_CCEP *pk_ccep TCP通信端点生成情報
内容：TCP通信端点属性

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

2006/10/07

TOPPERSプロジェクト認定

44



通信端点の生成は、静的APIではTCP_CRE_CEPサービスコール、動的APIではtcp_cre_cepサービスコールを用いて行います。

pk_ccep内容は以下の通りです。

ATR	ceptr	TCP通信端点属性
		API仕様では内容の指定はないため、ゼロを指定すれば良い
VP	sbuf	送信用のウィンドウバッファの先頭アドレス
INT	sbufsz	送信用のウィンドウバッファのサイズ
VP	rbuf	受信用のウィンドウバッファの先頭アドレス
INT	rbufsz	受信用のウィンドウバッファのサイズ
FP	callback	コールバックルーチンのアドレス

API仕様で定めるエラーコードは以下の通りです。

E_OK	正常終了
E_ID	不正ID番号
E_RSATR	予約属性
E_PAR	パラメータエラー(pk_ccepのアドレス、sbuf, sbufsz, rbuf, rbufsz, callbackが不正)
E_OBJ	オブジェクト状態エラー(指定したIDのTCP通信端点が生成済み)

API仕様では、ウィンドウバッファの先頭アドレスとしてNADR(-1)を指定することでプロトコルスタック内でバッファを確保する実装を認めている。

tcp_con_cep 接続要求(能動オープン) 【標準】

【C言語API】

```
ER ercd = tcp_con_cep(ID cepid, T_IPV4EP *p_myaddr,
                      T_IPV4EP *p_dstaddr, TMO tmout);
```

【パラメータ】

ID	cepid	TCP通信端点ID
T_IPV4EP	myaddr	自分側のIPアドレスとポート番号
T_IPV4EP	dstaddr	相手側のIPアドレスとポート番号
TMP	tmout	タイムアウト指定T_TCP_CCEP

【特記事項】

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

【構造体】

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

2006/10/07

TOPPERSプロジェクト認定

45



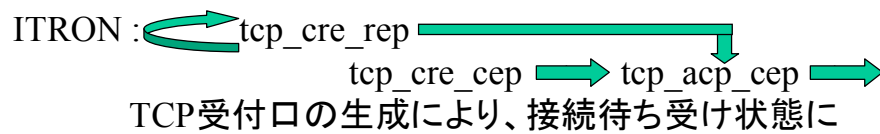
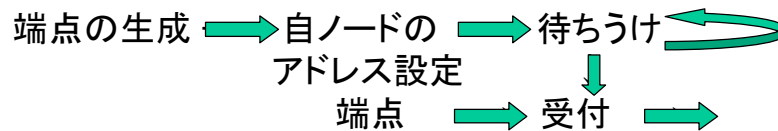
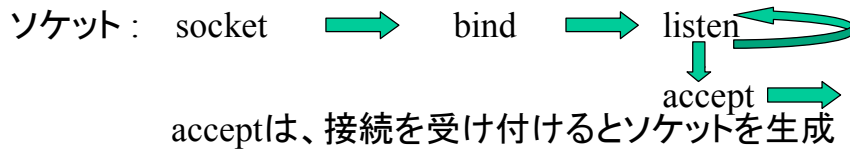
TCPクライアントプログラムでは、サーバーに対して能動的に接続を行う必要があります。ITRON TCP/IP API仕様ではtcp_con_cepサービスコールを用いて、この機能を行います。tcp_con_cepサービスコールは指定したTCP通信端点を用いて、指定した相手のIPアドレスとポート番号に対して接続を行い、接続が完了するまでサービスコール内で待ち状態となります。

myaddrにNADR(-1)を指定した場合、IPアドレスとポート番号はプロトコルスタック内で決定します。また、自分のIPアドレス(ipaddr)にIPV4_ADDRANY(=0)、ポート番号(portno)にTCP_PORTANY(=0)を指定した場合も同様に、プロトコルスタックで自動決定を行います。

tcp_con_cepを呼び出すことで、TCP通信端点は能動オープン状態となり、処理完了した時点で接続状態となります。タイムアウトやtcp_can_cepによってtcp_con_cepの処理がキャンセルされた場合、TCP通信端点は未使用状態に戻ります。

サーバー側の接続手順

- ソケットインターフェースとの対比



サーバープログラムをソケットインターフェースで作成する場合、**socket**関数を用いてTCPソケットを作成します。次に、**bind**関数を用いて指定したIPアドレスとポート番号を作成したソケットに関連付けを行います。**listen**関数にて、クライアントからの接続要求を許可します。**accept**関数はリスン状態にあるソケットのポート番号に接続要求が届くまで、関数内で待ち状態となります。接続要求を受けると、**accept**関数は新しいソケットのディスクリプタを作成して返します。

ITRON TCP/IP API仕様では、**tcp_cre_rep**サービスコールでTCP受付口を作成し、**tcp_cre_cep**サービスコールにてTCP通信端点を作成します。**tcp_acp_cep**サービスコールにてTCP受付口にてTCP通信端点にて接続されるまで待ちを行います。

サーバー側の接続手順

プログラム例

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

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

マルチタスクサーバー

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

2006/10/07

TOPPERSプロジェクト認定

47



複数のクライアントに対応したサーバープログラムを作成したい場合、マルチタスクサーバーの形態で実現が可能です。マルチタスクサーバーは同時に受け付けたいクライアントの数だけTCP通信端点とタスクを作成します。各タスクでtcp_acp_cepを用いて受付待ちを行えば、受け付けたTCP通信端点順に待ち状態が解除され、サーバー機能を実行することができます。

TCP_CRE_REP TCP受付口の生成

tcp_cre_rep

【標準】
【拡張】

【静的API】

```
TCP_CRE_REP(ID repid, {ATR repatr,  
                                {UW myipaddr, UH myportno}});
```

【C言語API】

```
ER ercd = tcp_cre_rep(ID repid, T_TCP_CREP *pk_crep);
```

【パラメータ】

ID repid TCP受付口ID
T_TCP_CREP *pk_crep TCP受付口情報
 内容 : TCP受付口属性
 自分側のIPアドレスとポート番号

【特記事項】

- 自分側のIPアドレスの自動決定指定あり

2006/10/07

TOPPERSプロジェクト認定

48



TCP受付口の生成は、静的APIではTCP_CRE_REPサービスコール、動的APIではtcp_cre_repサービスコールを用いて行います。

pk_crep内容は以下の通りです。

ATR cepatr TCP受付口属性
 API仕様では内容の指定はないため、ゼロを指定すれば良い
T_IPV4EPmayaddr 自分側のIPアドレス(myipaddr)とポート番号(myportno)

API仕様で定めるエラーコードは以下の通りです。

E_OK 正常終了
E_ID 不正ID番号
E_RSATR 予約属性
E_PAR パラメータエラー(pk_crepのアドレス、IPアドレスポート番号が不正)
E_OBJ オブジェクト状態エラー(指定したIDのTCP受付口が生成済み、ポート番号が既に使用済み、特権ポート違反)

tcp_cre_repは指定したIDのTCP受付口を生成し、指定したIPアドレスとポート番号で待ち状態にします。自分のIPアドレスにIPV4_ADDRANY(=0)を指定した場合、自分の持つ全てのIPアドレスに対する接続要求を待ち受けます。IPアドレスを指定した場合には、指定したIPアドレス宛の接続要求のみ受付を行います。

tcp_acp_cep 接続要求(受動オープン) 【標準】

【C言語API】

```
ER ercd = tcp_acp_cep(ID cepid, ID repid,
                      T_IPV4EP *p_dstaddr, TMO tmout);
```

【パラメータ】

ID	cepid	TCP通信端点ID
ID	repid	TCP受付口ID
TMP	tmout	タイムアウト指定T_TCP_CCEP

【リターンパラメータ】

T_IPV4EP p_dstaddr 相手側のIPアドレスとポート番号

【特記事項】

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

2006/10/07

TOPPERSプロジェクト認定

49



TCPサーバープログラムでは、クライアントからの受信を受動的に接続を行う必要があります。ITRON TCP/IP API仕様ではtcp_acp_cepサービスコールを用いて、この機能を行う、tcp_acp_cepは指定したTCP受付口に対する接続要求を待ちます。接続要求があった場合、指定したTCP通信端点を用いて接続を行い、相手のIPアドレスとポート番号を返します。tcp_acp_cepは接続が完了するまで待ち状態となります。

同じTCP受付口に対して、同時に複数のtcp_acp_cepを発行することができます。その場合、接続要求をどのTCP通信端点が受け付けるかは実装依存となります。

tcp_acp_cepを呼び出すことで、TCP通信端点は受動オープン待ち状態となり、処理完了した時点で接続状態となります。タイムアウトやtcp_can_cepによってtcp_acp_cepの処理がキャンセルされた場合、TCP通信端点は未使用状態に戻ります。

データの送受信

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

2006/10/07

TOPPERSプロジェクト認定

50



標準のデータの送受信は、`tcp_snd_dat`サービスコールと`tcp_rcv_dat`サービスコールを用います。このAPIは送信や受信にバッファを用いてデータ送受信を行い、送信領域に空きがない場合やデータ受信していない場合は待ち状態となります。

省コピーAPIのデータ送受信は、プロトコルが管理している送受信バッファに直接アクセスできるAPIを用意し、このAPIを用いて直接、送受信バッファをアクセスすることにより、通信効率を上げることを目的としたAPIです。このAPIは通信効率を上げる可能性もありますが、データの送受信プログラムが複雑化する可能性も高くなります。

tcp_snd_dat データの送信**【標準】****【C言語API】**

```
ER ercd = tcp_snd_dat(ID cepid, VP data, INT len,
                      TMO tmout);
```

【パラメータ】

ID	cepid	TCP通信端点ID
VP	data	送信データの先頭アドレス
INT	len	送信したいデータの長さ
TMO	tmout	タイムアウト指定

【リターンパラメータ】

ER	ercd	送信バッファに入れたデータの長さ /エラーコード
----	------	-----------------------------

【特記事項】

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

2006/10/07

TOPPERSプロジェクト認定

51



tcp_snd_datは指定したTCP通信端点からデータを送信するのに用います。データが送信バッファに入った時点でこのAPIからリターンします。空いている送信バッファ長が送信しようとしたデータ長よりも短い場合、送信バッファがいっぱいになるまで送信バッファにデータをいれ、送信バッファに入れたデータサイズを戻り値として返します。送信バッファに空きがない場合は空きが生じるまで待ち状態となります。同一のTCP通信端点に**tcp_snd_dat**か**tcp_get_buf**がペンディングしている間に**tcp_snd_dat**を発行すると**E_OBJ**エラーとなります。

エラーコードは以下の値となります。

正の値	正常終了(送信バッファに入れたデータの長さ)
E_ID	不正ID番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー (data , len , timeout が不正)
E_OBJ	オブジェクト状態エラー (指定したTCP通信端点が未接続または送信終了 tcp_snd_dat , tcp_get_buf がペンディング中)
E_WBLK	ノンブロッキングコール受付
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	処理のキャンセル、待ち状態の強制終了
E_CLS	TCP接続が切断された

tcp_rcv_dat データの受信**【標準】****【C言語API】**

```
ER ercd = tcp_rcv_dat(ID cepid, VP data, INT len,
                      TMO tmout);
```

【パラメータ】

ID	cepid	TCP通信端点ID
VP	data	受信データを入れる領域の先頭アドレス
INT	len	受信したいデータの長さ
TMO	tmout	タイムアウト指定

【リターンパラメータ】

ER	ercd	受信バッファに入れたデータの長さ /エラーコード
----	------	-----------------------------

【特記事項】

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

2006/10/07

TOPPERSプロジェクト認定

52



tcp_rcv_datは指定したTCP通信端点からデータを受信するのに用います。受信バッファに入ったデータを取り出した時点でこのAPIからリターンします。受信バッファに入っているデータ長が受信しようとしたデータ長よりも短い場合、受信バッファが空になるまでデータを取り出し、取り出したデータサイズを戻り値として返します。受信バッファが空の場合には、データを受信するまで待ち状態となります。相手側から接続が正常切断され、受信バッファにデータがなくなると、APIから0が返ります。

同一のTCP通信端点に**tcp_rcv_dat**か**tcp_rcv_buf**がペンディングしている間に**tcp_rcv_dat**を発行すると**E_OBJ**エラーとなります。

エラーコードは以下の値となります。

正の値	正常終了(取り出したデータの長さ)
0	データ終結(接続が正常切断された)
E_ID	不正ID番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー (data , len , timeout が不正)
E_OBJ	オブジェクト状態エラー(指定したTCP通信端点が未接続 tcp_rcv_dat , tcp_rcv_buf がペンディング中)
E_WBLK	ノンブロッキングコール受付
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	処理のキャンセル、待ち状態の強制終了
E_CLS	TCP接続が切断され、受信バッファが空

切断手順

ソケットインタフェースとの違い

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

ソケットインターフェースでの通信の終了は、**close**関数を用います。**close**関数ではソケットに、これ以上送受信できないというマークを付け、リターンします。ソケットは切断作業を開始し、切断が完了するまでソケットは残ります。ITRON TCP/IP API仕様での通信の終了は、**tcp_cls_cep**サービスコールで行います。**tcp_cls_cep**はTCP通信端点を切断し、TCP通信端子が未使用状態になるのを待ってリターンします。

tcp_cls_cep 通信端点のクローズ**【標準】****【C言語API】**

```
ER ercd = tcp_cls_cep(ID cepid, TMO tmout);
```

【パラメータ】

ID	cepid	TCP通信端点ID
TMO	tmout	タイムアウト指定

【特記事項】

- まだ送信終了していない場合は、送信バッファ中のデータを送信し終わるのを待ってFINを送り、接続を切断する
- 受信したデータは捨てる
- TCP通信端点が未使用状態になるのを待つ
- タイムアウトエラーになると、RSTを送って強制切断する。この場合でも、元の状態に戻らない(例外的なサービスコール)

2006/10/07

TOPPERSプロジェクト認定

54



tcp_cls_cepはTCP通信端点のクローズを行います。詳細な処理としては、まず、指定したIDのTCP通信端点はまだ通信終了となっていない場合は、送信バッファ中のデータを送信し終わるのを待ってFINを送り、接続の切断を行います。また、以後受信したデータは捨てます。TCP通信端点が未使用状態になるのを待ってリターンするため、**tcp_cls_cep**からリターンした後はTCP通信端子はすぐに再利用することができます。

タイムアウトや**tcp_can_cep**によって**tcp_cls_cep**の処理がキャンセルされた場合、指定したTCP通信端子からRSTを送信して接続を強制切断します。RSTをすぐに送信できない場合には、送信のみ手配をおこなうか、それもできない場合はRSTの送信を省略します。いずれにしても、**tcp_cls_cep**からリターンするとTCP通信端点未使用状態となっています。(ノンブロッキング指定を行った場合は、完了を通知するコールバックで未使用状態に移行します)**tcp_cls_cep**は、タイムアウトが起こった場合にはAPIを呼び出したことでオブジェクト状態は変化しないという原則の例外となっています。

通信端点は、ソケットインターフェースにおけるファイルディスクリプタと異なり、TCPの接続状態が完全に終了するまで再利用可能となりません。TCP/IPプロトコルの規格に従うと、接続状態が完全に終了するまでに数分かかる場合があります。

エラーコードは以下の通りです。

E_OK	正常終了
E_ID	不正ID番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー (timeoutが不正)
E_OBJ	オブジェクト状態エラー (指定したTCP通信端点が未接続)
E_WBLK	ノンブロッキングコール受付
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	処理のキャンセル、待ち状態の強制終了

tcp_sht_cep データ送信の終了(ハーフクローズ) 【標準】**【C言語API】**

```
ER ercd = tcp_sht_cep(ID cepid);
```

【パラメータ】

ID cepid TCP通信端点ID

【特記事項】

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

2006/10/07

TOPPERSプロジェクト認定

55



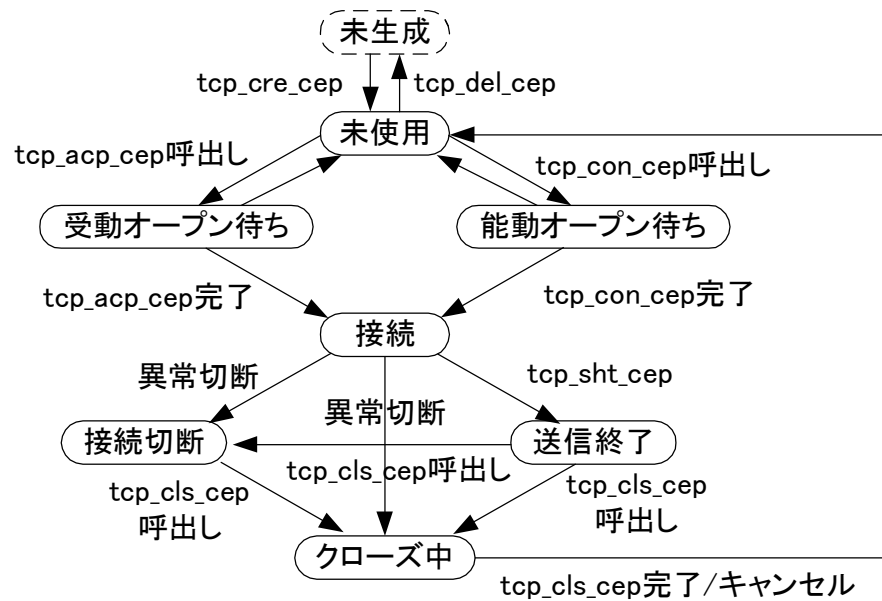
tcp_sht_cepは、指定したTCP通信端点に対してデータ送信を終了します。具体的には、送信バッファ中のデータを送信し終わった後、FINを送って接続の切断手順を行うように手配します。tcp_sht_cepは、切断する手配を行うだけであるため、API中で待ち状態になることはありません。

tcp_sht_cepを呼んで以降は、TCP通信端点は送信終了状態になり、それに対するデータは送信できません。送信しようとするとE_OBJエラーとなります。データの受信は可能です。

エラーコードは以下の値となります。

正の値	正常終了(送信バッファに入れたデータの長さ)
E_ID	不正ID番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー(data, len, timeoutが不正)
E_OBJ	オブジェクト状態エラー(指定したTCP通信端点が未接続)

TCP通信端点の状態遷移



2006/10/07

TOPPERSプロジェクト認定

56



TCP通信端点は、API上、「未生成」、「未使用」、「受動オープン待ち」、「能動オープン待ち」、「接続」、「送信終了」、「接続切断」、「クローズ中」の8つの状態を遷移します。未生成以外の7つの状態のいずれかにある場合を「生成済み」と呼びます。未生成および未使用以外の6つの状態のいずれかにある場合を「使用中」と呼びます。接続、送信終了、接続切断以外の5つの状態のいずれかにある場合を「未接続」と呼びます。

UDPのサービスコールの一覧

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

2006/10/07

TOPPERSプロジェクト認定

57



UDP (User Datagram Protocol) は、TCPと異なる種類のエンドツーエンドサービスを提供します。UDPではTCPのように接続を確立しなくても使用することができます。また、任意のアドレスから複数のアドレスにデータを送ることができます。また、その逆のデータ送受信も可能です。そのため、TCPのように受付け口を作る必要がありません。

UDPのAPIに使う通信端点として、UDP通信端点 (UDP Communication End Point, cepと略す) を用意します。UDPの通信端点に対して相手側のIPアドレスとポート番号をあらかじめ設定する機能は用意していません。

あるインターフェースのブロードキャストアドレス宛に送信されたパケットは、そのインターフェースに対してブロードキャストされます。255.255.255.255宛に送信されたパケットは、ブロードキャストをサポートしているすべてのインターフェースからブロードキャストされます。

その他のサービスコール

受付口・通信端点の削除

緊急データの送受信

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

ペンディング中の処理のキャンセル

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

通信端点オプションの設定/読出し

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

2006/10/07

TOPPERSプロジェクト認定

58



TCPのその他のサービスコールについて説明を行います。

•受付口・通信端点の削除

動的APIで生成されたTCP受付口やTCP通信端点を削除するサービスコールが拡張APIとして用意されています。

•緊急データの送受信

ITRON TCP/IP API仕様では、緊急データはout-of-bandデータとして扱うことを標準としています。実装依存で、TCP通信端点属性やTCP通信端点オプションの設定を行うことにより、in-bandのデータとして扱う方法もあります。緊急データの受信サービスコールtcp_rcv_oobは緊急データ受信のコールバックルーチン内で呼び出すことを想定しています。

•ペンディング中の処理のキャンセル

tcp_can_cepは指定したTCP通信端点にペンディングしている指定の処理の実行をキャンセルします。キャンセルされたAPIには、E_RLWAIエラーが返ります。また、ノンブロッキングをキャンセルした場合には、処理の完了を通知するコールバックルーチンが呼ばれます。

•通信端点オプションの設定／読出し

TCP通信端点のオプションに対して、読出しや書き込みを行うサービスコールを用意しています。TCP通信端点のオプションの種類と機能は実装依存であり、API仕様では設定していません。

TCPのコールバック

コールバックルーチンの設定と呼出し

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

コールバックを起こす事象

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

2006/10/07

TOPPERSプロジェクト認定

59



コールバックルーチンは、アプリケーションプログラムが定義して、実装依存のコンテキストで実行されます。また、TCP通信端点に対して1つずつの定義できます。但し、TCP受付口はコールバックルーチンをもちません。コールバックを起こした事象の種類は、コールバックルーチンへの引数として渡されます。

以下がコールバックルーチンの共通定義です。

【C言語API】

```
ER ercd = callback(ID cepid, FN fincd, VP p_parblk);
```

【共通パラメータ】

ID	cepid	TCP通信端点ID
FN	fincd	事象の種類

【固有パラメータ】

VP	p_parblk	事象に固有なパラメータブロックのアドレス
----	----------	----------------------

【返値】

事象の種類に依存

TCP通信端点のコールバックの事象は以下の2つです。

• ノンブロッキングコールの完了通知

ノンブロッキングコールの処理が完了した、または、キャンセルされた場合に呼び出されます。APIからの返値がパラメータブロックに入れて渡されます。APIからのその他のリターンパラメータは、APIを呼び出した時に指定した領域に格納されます。

• 緊急データの受信

緊急データを受信した場合に呼び出されます。コールバックルーチン内で、tcp_rcv_oobを使って緊急データを取り出す必要があります。取り出されなかった場合は、データは捨てられます。

省コピーAPIによるデータの送受信

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

TCP用APIに対して、省コピーによるデータの送受信手順をサポートしています。この手順は、プロトコルスタック上のウィンドウバッファ領域が連続して取られていることを前提としています。これを実現するために`tcp_cre_cep`にてウィンドウバッファ領域を与える必要があります。この場合、送受信APIにて最低1回のコピーが発生することになります。省コピーAPIはウィンドウバッファに直接読み書きが可能なAPIを提供することを目的とします。

送受信で、実際に何回のコピーが発生するかは、LANコントローラやプロトコルスタックの実装に依存します。

tcp_get_buf 送信用バッファの取得(省コピーAPI) 【標準】**【C言語API】**

```
ER ercd = tcp_get_buf(ID cepid, VP *p_buf, TMO tmout);
```

【パラメータ】

ID	cepid	TCP通信端点ID
TMO	tmout	タイムアウト指定

【リターンパラメータ】

VP	buf	空き領域の先頭アドレス
ER	ercd	送信バッファに入れることができるデータの長さ /エラーコード

【特記事項】

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

2006/10/07

TOPPERSプロジェクト認定

61



tcp_get_bufは送信バッファ中の、次に送信すべきデータを入れることができる空き領域の先頭アドレスと、連続した空き領域の長さを返します。送信バッファに空きがない場合は、空きが発生するまで待ち状態となります。tcp_get_bufを呼び出したことで、プロトコルスタックの内部情報は変化しません。そのため、tcp_get_bufを続けて呼び出しても同じ領域が返されます。逆に、tcp_snd_dat、tcp_snd_bufを呼び出すとプロトコルスタックの内部状態は変化します。これらのAPIを呼び出すと、それ以前に呼び出したtcp_get_bufが返した情報は無効となります。

エラーコードは以下の値となります。

正の値	正常終了(空き領域の長さ)
E_ID	不正ID番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー(p_buf, timeoutが不正)
E_OBJ	オブジェクト状態エラー(指定したTCP通信端点が未接続または送信終了 tcp_snd_dat, tcp_get_bufがペンディング中)
E_WBLK	ノンブロッキングコール受付
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	処理のキャンセル、待ち状態の強制終了
E_CLS	TCP接続が切断された

tcp_snd_buf バッファ内のデータ送信(省コピーAPI) 【標準】**【C言語API】**

```
ER ercd = tcp_snd_buf(ID cepid, INT len);
```

【パラメータ】

ID	cepid	TCP通信端点ID
INT	len	データの長さ

【特記事項】

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

2006/10/07

TOPPERSプロジェクト認定

62



tcp_get_bufで取り出したバッファに書き込んだ長さlenのデータを送信するように手配します。tcp_snd_bufは、送信する手配を行うだけであるため、API中で待ちになることはありません。lenが、送信すべきデータを入れることができる連続した空き領域の長さ(tcp_snd_bufを呼び出すことで出される値)よりも大きい場合にはE_OBJエラーを返します。

エラーコードは以下の値となります。

正の値	正常終了
E_ID	不正ID番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー(p_buf, timeoutが不正)
E_OBJ	オブジェクト状態エラー(指定したTCP通信端点が未接続または送信終了lenが長すぎる)
E_CLS	TCP接続が切断された

tcp_rcv_buf 受信バッファの取得(省コピーAPI) 【標準】**【C言語API】**

```
ER ercd = tcp_rcv_buf(ID cepid, VP *p_buf, TMO tmout);
```

【パラメータ】

ID	cepid	TCP通信端点ID
TMO	tmout	タイムアウト指定

【リターンパラメータ】

VP	buf	受信データの先頭アドレス
ER	ercd	受信データの長さ/エラーコード

【特記事項】

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

2006/10/07

TOPPERSプロジェクト認定

63



tcp_rcv_bufは受信したデータが入っているバッファの先頭アドレスと、そこから連続して入っているデータの長さを返します。受信バッファが空の場合には、データを受信するまで待ち状態となります。相手側から接続が正常終了され、受信バッファにデータがなくなると、APIから0が返ります。

tcp_rcv_bufを呼び出すことにより、プロトコルスタックの内部状態は変化しません。そのため、tcp_rcv_bufを続けて呼び出すと同じ領域が返されます。逆に、tcp_rcv_dat、tcp_rel_bufを呼び出すと、それ以前に呼び出したtcp_rcv_bufが返した情報は無効となります。

同一のTCP通信端点に対するtcp_rcv_datがtcp_rcv_bufがペンディングしてる間にtcp_rcv_bufを発行するとE_OBJエラーとなります。

エラーコードは以下の値となります。

正の値	正常終了(受信データの長さ)
0	データ終結(接続が正常切断された)
E_ID	不正ID番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー(p_buf, timeoutが不正)
E_OBJ	オブジェクト状態エラー(指定したTCP通信端点が未接続またはtcp_rcv_dat, tcp_rcv_bufがペンディング中)
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	処理のキャンセル、待ち状態の強制解除
E_CLS	TCP接続が切断され、受信バッファが空

tcp_rel_buf 受信バッファの開放(省コピーAPI) **【標準】****【C言語API】**

```
ER ercd = tcp_rel_buf(ID cepid, INT len);
```

【パラメータ】

ID	cepid	TCP通信端点ID
INT	len	データの長さ

【特記事項】

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

2006/10/07

TOPPERSプロジェクト認定

64



tcp_rcv_bufで取り出したバッファ中の長さlenのデータを捨てます。このAPI中では待ち状態となることはありません。lenが受信したデータが連続して入っているデータの長さ(tcp_rcv_bufで返される値)よりも大きい場合にはE_OBJエラー、原則的にどのような条件においても不正な値(例えば負の値)になる場合にはE_PARエラーとします。

エラーコードは以下の値となります。

正の値	正常終了
E_ID	不正ID番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー (lenが不正)
E_OBJ	オブジェクト状態エラー (指定したTCP通信端点が未接続、lenが長すぎる)

省コピーAPI : プログラム例

```

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

```

2006/10/07

TOPPERSプロジェクト認定

65



省コピーAPIのプログラム例

受信したデータをエコーバックするプログラムを省コピーAPIを使って記述しました。While文の条件の `tcp_rcv_buf` で受信データの先頭アドレスとサイズを取り出します。値が0の場合は正常切断として、マイナス値の場合エラーとしてwhile文を抜けます。次の `tcp_get_buf` にて送信可能なバッファ領域の先頭アドレスとサイズを取り出します。送信空きサイズと受信データサイズのうち小さいサイズを `len` に設定し、受信データを送信空きバッファに `len` サイズ分コピーします。このとき、大文字のキャラクタは小文字のキャラクタに変換します。

`tcp_snd_buf` でコピーしたデータを送信要求し、`tcp_rel_buf` にてコピーした `len` 分の受信データを捨てます。

TINET (TCP/IPプロトコルスタック)

1. TCP/IPの基礎
2. ITRON TCP/IP仕様
3. TINET (TCP/IPスタック)
4. TCPサーバープログラム
5. TCPクライアントプログラム
6. まとめ

TINETの特徴

苫小牧高専 情報工学科において開発された
ITRON TCP/IP API仕様に準拠したコンパクトな
TCP/IP プロトコルスタック

- FreeBSDをベースに
 - 枯れたソフトウェアで、他システムの手本
 - BSD ライセンスによる配布
 - 人的リソース制約
- 組み込みシステムのリソース制約に対応
- 対応 RTOS は TOPPERS/JSP カーネル
- API は ITRON TCP/IP API 仕様

2006/10/07

TOPPERSプロジェクト認定

67



TINETは苫小牧高専の情報工学科において開発されたITRON TCP/IP API仕様に準拠したコンパクトなTCP/IPプロトコルスタックです。TINETはFreeBSDのプロトコルスタックをもとの開発を行っています。また、実装はTOPPERS/JSPカーネルを対象に行われています。そのため、TINETを含むソフトウェアを他のソフトウェア開発に使用できない形で再配布を行う場合、TOPPERSのライセンスに加えて、FreeBSDおよびFreeBSDへのソフトウェアの寄贈者のライセンス規定に従って、再配布に伴うドキュメント(利用者マニュアルなど)に、ライセンス表示を行う等の必要が発生します。しかしながらFreeBSDライセンス自体は、他リソースに比べ制約の少ないライセンスとして知られています。

FreeBSDの元となったBSD UNIXのプロトコルスタックは枯れたソフトウェアであり、多くのシステムの手本としてなっています。

実装上考慮した事項

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

2006/10/07

TOPPERSプロジェクト認定

68



TINETの開発においては、メモリ容量の制約のきつい組込みシステム上の実行を想定して行っています。そのため、単一リンクの組込みシステムを対象とした実装となっています。最小構成を想定した場合、IPv4の実装では、RAMが約8Kバイト、ROMが約41Kバイトとなり、IPv6の実装では、RAMが約9Kバイト、ROMが約59Kバイトとなっています。IPv6に関しては、現状(2005年6月)の時点では、ITRON TCP/IP API仕様には、IPv6の仕様は標準化されておらず、仮仕様として実装されています。

TINET が提供する機能と実装ターゲット

提供機能

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

実装ターゲット

- 秋月電子通商製 AKI-H8/3069F LAN ボード
 - NE2000 互換 NIC
 - 内部ループバック
 - シリアルインタフェースを利用した PPP
- 北斗電子製 HSB7727ST (SH3)
 - SMSC 製 LAN91C111 NIC

2006/10/07

TOPPERSプロジェクト認定

69

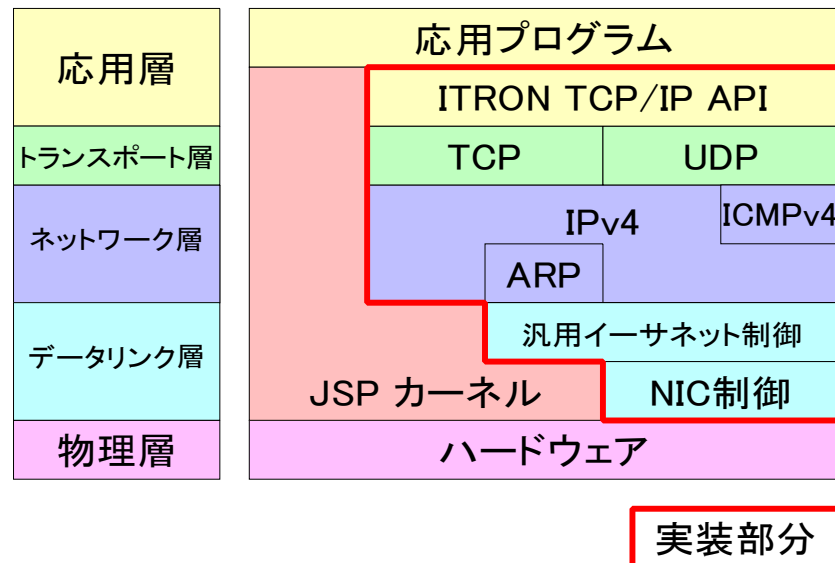


TINETはITRON TCP/IP API仕様に準じたアプリケーションを開発することにより、幅広い応用層に対応が可能です。TCPのオプションとしては最大セグメントサイズMSS (Maximum Segment Size)のみをデファイン文で指定が可能です。また、データリンク層へのインターフェースはネットワークシステムに依存したインターフェースで制御を行います。

TINETの実装ターゲットは、現状(2005年6月)は以下の2つです。

- 秋月電子通商製のAKI-H8/3069F LANボード以下の機能に対応しています。
 - NE2000互換NIC (REALTEK製RTL8019AS)
 - 内部ループバック機能
 - シリアルインターフェースを用いたPPP
- 北斗電子製HSB7727ST (SH3)ボードでは以下のEtherNetに対応しています。
 - SMSC製LAN91C111NIC

ネットワーク階層図 (IPv4)



2006/10/07

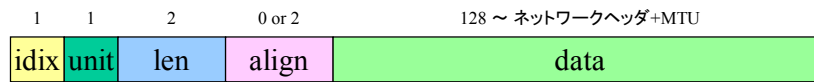
TOPPERSプロジェクト認定

70



上の図が、ネットワーク階層図に対応したTOPPERS/JSP及びTINETの階層図です。トランスポート層はTCPとUDPで、応用層とのインターフェースはITRON TCP/IP API仕様となっています。ネットワーク層は、IPv4とIPv6が選択できます。データリンク層はネットワーク・システムに依存しない汎用イーサネット制御を行います。

ネットワークバッファ(net_buf)



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

2006/10/07

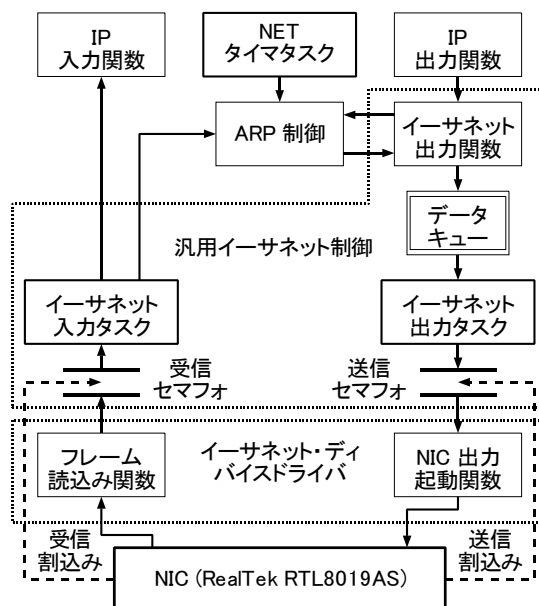
TOPPERSプロジェクト認定

71



TINETではプロトコル・スタック内の各階層のデータの受け渡しをネットワークバッファ(net_buf)を用いて行っています。これはITRON 4.0仕様の固定長メモリ・プールから取りだして割り当てを行っています。出力時はTCP/UDPで確保を行い、入力時はイーサネットドライバで確保を行います。いずれも、割り当て前にデータの確保を行い、プロトコル・スタック内では、ネットワークバッファの動的なメモリ操作は行っていません。これらのメモリリソースの設定はCPU依存のコンフィギュレーション・ディレクトリ中のTINET依存のインクルードファイルに設定されています。

イーサネット制御とデータの流れ



2006/10/07

TOPPERSプロジェクト認定

72

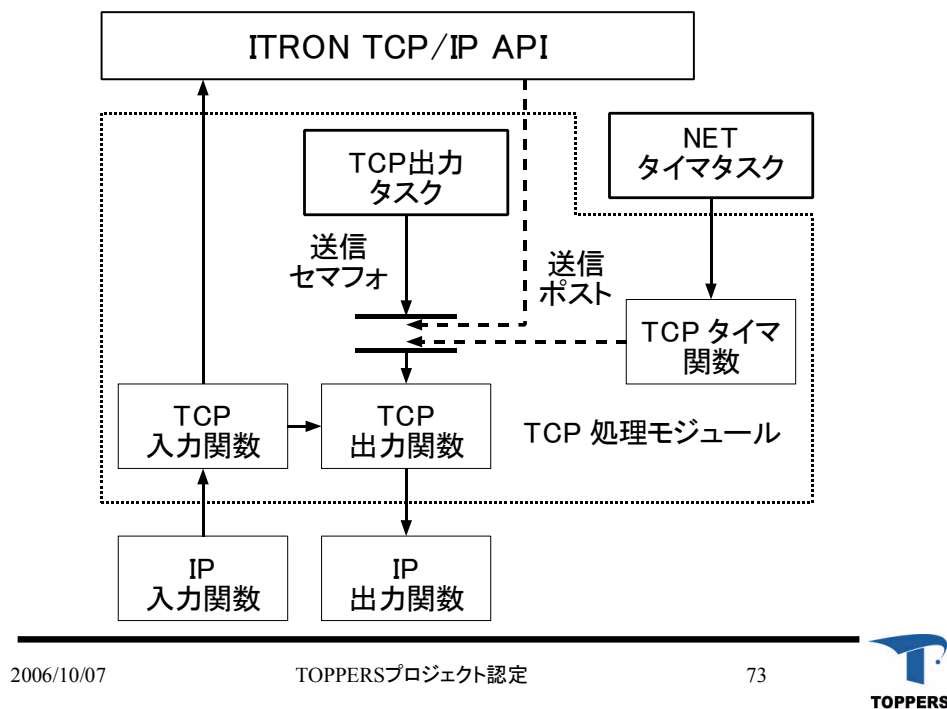


上の図がネットワーク層から物理層 (Ethernet) のデータの流れを示した図です。データ・リンク層は、NICに依存しない汎用のEthernet制御と、NICに依存するEthernetデバイスから構成されています。この図は汎用のEthernetのデータの流れを中心に説明を行っています。

Ethernetからの受信があった場合、NICからの割込みが発生します。この割込みはイーサネット入力タスクと同期をとり、このタスクが受信バッファからEthernetフレームを取り出します。そして、Ethernetヘッダから上位プロトコルを判定し、IP入力関数から上位プロトコルにデータを渡します。

Ethernetへ出力を行う場合は、IP出力関数は、ネットワーク・バッファにEthernetヘッダとIPデータグラム設定されて上位プロトコルから呼び出されます。イーサネット出力関数はネットワークバッファのEthernetバッファにMACアドレスを設定します。このとき、プロトコルがIPv4の場合は、ARPにより送信先のMACアドレスを解決します。解決後、ネットワークバッファをデータキューに投入します。イーサネット出力タスクはデータキューからネットワークバッファを取り出し、NIC割込みハンドラと同期してデータの書き込みを行います。

TCP の制御とデータの流れ



2006/10/07

TOPPERSプロジェクト認定

73



TCP制御では、IP入力関数からのネットワークバッファをITRON TCP/IP APIに準じた手順で汎用プログラムに渡し、ITRON TCP/IP API仕様に準じて受け取ったネットワークの送信データをIP出力関数に渡します。

ITRON TCP/IP API の実装

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

2006/10/07

TOPPERSプロジェクト認定

74



TINETのITRON TCP/IP API仕様に準じた実装を説明します。IPv4については、ITRON TCP/IP API仕様の標準機能を実装しています。但し、IPv6については、暫定的な仕様の標準機能の実装となっています。現在(2005年6月)は、トロン協会でITRON TCP/IP API仕様の改定を行っており、TINETの暫定の仕様と異なる可能性があります。

TCPについては、BSDの通信機能、最大セグメントサイズの設定インターフェース、ITRON TCP/IP API仕様で指定されている省コピーAPIとノンブロッキングコールの送受信インターフェースをサポートしています。UDPについては、ノンブロッキングコールの送受信機能をサポートしています。静的APIはTCP/UDPの通信端点の生成とTCP受付口の生成をサポートしています。

TCP のAPI

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

NBLK*: ノンブロッキングコール

2006/10/07

TOPPERSプロジェクト認定

75



ITRON TCP/IP API仕様でのTCPに対するサポートAPIは上記の通りです。ノンブロッキングコールと省コピーに対応したAPIはそれぞれ○を指定しています。

UDP のAPI

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

ITRON TCP/IP API仕様でのUDPに対するサポートAPIは上記の通りです。ノンブロッキングコールに対応したAPIはそれぞれ○を指定しています。

TINET ディレクトリ構成

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

TINETのディレクトリ構成について説明を行います。tinetディレクトリはjspのディレクトリ上にあります。./cfgディレクトリはITRON TCP/IP API仕様の静的APIを解決するTINETコンフィギュレータのソースが収められているディレクトリです。./docはTINETのドキュメント類が収められているディレクトリです。./netディレクトリには、ネットワークシステムに依存しない汎用ネットワーク制御に関するプログラムが収められています。./netappディレクトリはサンプルのネットワークアプリケーションプログラムが収められています。./netdevディレクトリにはEthernetデバイス・ドライバ中のJSPターゲットに依存しないプログラムファイルに収められています。./netdev/if_edディレクトリにはNE2000互換のRTL8019AS用のデバイスドライバが収められています。./netinetディレクトリはIPv4に依存したTINETの本体のプログラムが、./netinet6ディレクトリにはIPv6に依存したTINETの本体のプログラムが収められています。

TINET のドキュメント(./doc)

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

./docディレクトリ中のドキュメントの内容は上記の通りです。

コンフィギュレーション/パラメータ定義ファイル

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

2006/10/07

TOPPERSプロジェクト認定

79



次に、実装に依存したパラメータ依存ファイルに関して説明を行います。

TOPPERS/JSPターゲットに依存するファイルは、config/\$(CPU)/\$(SYS)中のJSPシステム依存アセンブリ関数ファイルのsys_support.Sと、システム依存のインクルードファイルのtinet_sys_config.hです。H8の場合、sys_support.SではRTL8019ASの割込みベクタの設定とNICハードウェアの初期化とH8の割込み実装依存の割込み許可、禁止関数が追加、変更となっています。tinet_sys_config.hはNICについての定義とRTL8019ASのハードウェア制御についての定義が行われています。

プロセッサ依存部のネットワーク設定は、config/\$(CPU)/tinet_cpu_config.hファイルにて定義されています。アライン等のプロセッサ依存の設定は、config/\$(CPU)/tinet_cpu_defs.hファイルにて定義されています。

ネットワークアプリケーション依存の定義は\$(APP_DIR)/tinet_app_config.hで定義します。このインクルードファイルにはボードのIPアドレス等を指定します。

ネットワークインターフェースに依存した定義は、tinet/netdev/\$(NIC)/tinet_nic_config.hとtinet/netdev/\$(NIC)/tinet_nic_defs.hで行われています。

応用プログラムファイル

`$(APP_DIR)/tinet_$(UNAME).cfg`

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

`$(APP_DIR)/tinet_app_cfg.h`

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

`$(APP_DIR)/route_cfg.c`

- 静的経路定義ファイル

2006/10/07

TOPPERSプロジェクト認定

80



`$(APP_DIR)/tinet_$(UNAME).cfg`はTINET用のコンフィギュレーションファイルで、`$(UNAME)`はネットワークアプリケーション名を指定します。このファイルに通信端点や受付口等の生成を行う静的APIの定義を行います。このファイルは`make tinet`によるビルドでTINETコンフィギュレータにより、C言語のプログラムに変換されます。

`$(APP_DIR)/route_cfg.c`は静的経路定数を定義するC言語のプログラムです。

tinet_app_config.h の例

```
//IP アドレスの設定
#define IPV4_ADDR_LOCAL          ¥
    0xac198158          /* 172.25.129.88 */
#define IPV4_ADDR_LOCAL_MASK     ¥
    0xffffffff          /* 255.255.255.0 */
#define IPV4_ADDR_DEFAULT_GW     ¥
    0xac19818c          /* 172.25.129.140 */
```

tinet_app_config.hファイルの記述例です。
IPV4_ADDR_LOCALはIPアドレスを16進数で指定します。
IPV4_ADDR_LOCAL_MASKはネットマスク値を指定します。
IPV4_ADDR_DEFAULT_GWは、デフォルトゲートウェイのIPアドレスを指定します。

静的経路定義ファイル route_cfg.c の例

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

route_cfg.cは静的なルーティング表をrouting_tblを用いて設定します。

TINETコンフィギュレータ生成ファイル

`$(APP_DIR)/tinet_cfg.c`

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

`$(APP_DIR)/tinet_kern.cfg`

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

`$(APP_DIR)/tinet_id.h`

- TCP 受付口、TCP 通信端点、及び UDP 通信端点のID 自動割付結果ファイル。

2006/10/07

TOPPERSプロジェクト認定

83



TINETコンフィギュレータは、アプリケーションディレクトリ中に以下のファイルを生成します。

• `$(APP_DIR)/tinet_cfg.c`

TINET用コンフィギュレーションファイルに記述したTCP受付口、TCP通信端点、及び、UDP通信端点などが、C言語の構造体データで記述したソースファイル。ビルド時、アプリケーションファイル等と一緒にリンクされます。

• `$(APP_DIR)/tinet_kern.cfg`

TINETの内部で使用するJSPカーネルのカーネルオブジェクトを静的APIで記述したコンフィギュレーションファイル。JSPのコンフィギュレーションファイルからインクルードされ、JSPのコンフィギュレータにより、C言語ファイルに変換されます。

• `$(APP_DIR)/tinet_id.h`

TCP受付口、TCP通信端点、および、UDPの通信端点のIDを自動割付した値が定義されているインクルードファイル。

ファイル tinet_kern.cfg の例

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

生成されたtinnet_kern.cfgファイルではITRON4.0仕様のカーネルオブジェクトの静的APIが定義されています。

TCPサーバープログラミング

1. TCP/IPの基礎
2. ITRON TCP/IP仕様
3. TINET (TCP/IPスタック)
4. TCPサーバープログラム
5. TCPクライアントプログラム
6. まとめ

TCPサーバープログラミング

ITRON TCP/IP API仕様を用いて,
TCPサーバープログラムを作成します

- TINETとTOPPERS/JSPカーネルを用いる
- 完成したプログラムはプログラムディレクトリの以下のディレクトリにあります
 - ./OBJ/AKIH8_3069F/
 - TCP_SRV : ベースサーバー
 - ECHO_SRV : エコーサーバー
 - DEV_SRV : デバイスサーバー
 - MULTI_SRV : マルチタスクサーバー

TINETを用いて, ITRON TCP/IP API

このセクションで作成するTCPサーバープログラムは、./OBJ/AKIH8_3069Fディレクトリ上に完成プログラムが作成されています。実習の後、参考になしてください。

TCPサーバプログラミング : PC側の設定

PCの有線側のネットワークの
IPアドレスを192.168.11.6に、ネットマスクを255.255.255.0に設定

- ボードとPCをイーサネットケーブルにより接続する
- ディスケット上の”マイネットワーク”のアイコンで右クリックし”プロパティ”を選択
- 開かれたウィンドウで”ローカルエリア接続”のアイコンで右クリックし”プロパティ”を選択
- インターネットプロトコル(TCP/IP)を選択してプロパティをクリック
- IPアドレスとサブネットマスクを指定
 - IPアドレス : 192.168.11.6
 - サブネットマスク : 255.255.255.0
- コンソールもしくはコマンドプロンプトでipconfigを実行して設定を確認

AKIH8-3069Fボードとパソコンはイーサネットのクロスケーブルで接続します。パソコン側では固定のIPアドレスを設定しなければなりません。パソコンの設定後、CygwinまたはDOS窓からipconfigコマンドを入力してパソコンのIPアドレスを確認します。

TCPサーバープログラミング

- 完成したプログラムはプログラムディレクトリの以下のディレクトリにあります
 - ./OBJ/AKIH8_3069F/
 - TCP_SRV : ベースサーバー
 - ECHO_SRV: エコーサーバー
 - DEV_SRV : デバイスサーバー
 - MULTI_SRV : マルチタスクサーバー
- TCP_SRVの内容をコピーしてMY_SRVディレクトリを作成してください
- 各プログラムでは、ボード側のIPアドレスを192.168.11.98に設定。IPアドレスは、tinet_app_config.h の IPV4_ADDR_LOCALマクロで設定があります

2006/10/07

TOPPERSプロジェクト認定

88



このセクションで作成するTCPサーバープログラムは、./OBJ/AKIH8_3069Fディレクトリ上に完成プログラムが作成してあります。実習の後、参考にしてください。

TOPPERS/JSP-1.4.1のH8の実装ではMACドライバの割込みの設定を有効にするためにDSUPPORT_ETHERをデファインして、再構築する必要がありましたが、1.4.2の実装でDEF_INHの設定を自動反映できるようになりました。以下はJSP-1.4.1の教材の場合の説明です。

libkernel中のMakefileの101行目を以下のように書き換えてください。

```
COPTS := $(COPTS) → COPTS := $(COPTS) -DSUPPORT_ETHER
```

CygwinでOBJ/AKIH8_3069F/libkernelディレクトリに移動して、以下のコマンドを実行してカーネルライブラリを再構築してください。

```
$ make clean
```

```
$ make libkernel.a
```

次にTCP_SRVをAKIH8_3069Fディレクトリ上にMY_TCPという名前でコピーしてください。

```
$ cd .. 'OBJ/AKIH8_3069Fディレクトリに移動
```

```
$ cp -r TCP_SRV MY_SRV 'ディレクトリごとコピー
```

ボードのIPアドレスは、tinet_app_config.hファイル中のIPV4_ADDR_LOCALマクロ、サブネットマスクはIPV4_ADDR_LOCAL_MASKの定義で行います。

TCPサーバープログラミング

1. ベースサーバー
2. エコーサーバー
3. デバイス操作サーバー
4. マルチタスクサーバー

ベースサーバー : TCP_SRV

- 実行されると、TCPを用いてポート番号"1234"でクライアントからの接続を待つ。
- クライアントと接続した後は、クライアントから送られてきたデータをシリアルコンソールに出力する
- プログラムをコンパイルして動作を確認する

```
$ cd MY_SRV
$ make realclean
$ ls
Makefile  tcp_srv.c  tcp_srv.h  tinet_tcp_srv.cfg
route_cfg.c tcp_srv.cfg tinet_app_config.h
$ make depend
$ make
```

2006/10/07

TOPPERSプロジェクト認定

90



ベースサーバーは、TCPサーバーを構成します。ポート番号"1234"にてクライアントからの接続を待ち、接続が確立したら、クライアントの入力データをシリアルコンソールに表示します。

MY_SRVディレクトリに移動します。ベースサーバープログラムを構築します。

\$ make tinet 'TINET-1.3より古いバージョンのTINETをお使いの場合は
TINETのコンフィグレータを起動する必要があります。

本教材は1.3を使用していますので、この操作は必要ありません

\$ make depend '依存関係の生成を行います。JSPのコンフィギュレーション

\$ make 'JSPのコンフィギュレーションとコンパイル、リンク

teraTermを立ち上げ、シリアルケーブルとイーサネットのクロスケーブル接続を確認し、電源を入れます。teraTermにROMモニターにプロンプトが表示されたら、ldコマンドを入力し、構築されたjsp.srecファイルをボードに転送します。

ベースサーバー：接続

- サーバーへの接続は telnet を用いる
- telnet はスタートメニューの"ファイル名を指定して実行"から実行する
telnet 192.168.11.98 1234
- telnet に入力した文字がシリアルコンソールに出力される

```

Tera Term - COM1 VT
File Edit Setup Control Window Help
JSP Kernel Release 1.4 (patchlevel = 1) for AKI-H8/3069F (Nov 4 2004, 23:50:48)
Copyright (C) 2000-2003 by Embedded and Real-Time Systems Laboratory
Toyohashi Univ. of Technology, JAPAN

System logging task is started on port 2.
[TCP SRV :8,0] started.
[ETHER INPUT: 5] started on MAC Addr: 00:02:cb:01:76:2f.
[ETHER OUTPUT:4] started.

TINET Release 1.2 for JSP Kernel Release 1.4 (Nov 24 2004, 18:30:23)
Copyright (C) 2001-2004 by Dep. of Computer Science and Engineering
Tomakomai National College of Technology, JAPAN

[NET/TIMER:2] started.
[TCP OUTPUT:3] started.
[TES:01 ACP] connected: 2, from: 192.168.2.1.1237
Recv Data g
  
```

2006/10/07

TOPPERSプロジェクト認定

91



プログラムの転送が終了後、TeraTermにてgoコマンドでベースサーバーを実行します。Cygwinまたは、DOS窓より以下のコマンドにて、ベースサーバーに接続します。

```
$ telnet 192.168.11.98 1234
```

この後、文字入力をEnterキーで入力した文字がTeraTermに表示されます。

telnetを終了させる場合は、CTRL-]の後、Enterキー入力で、以下のプロンプトが表示されます。

```
telnet>
```

このプロンプトにquit[enter]を入力すると、telnetを終了します。

```
telnet>quit
```

ベースサーバー : ファイル構成

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

TCP_SRV上のファイルは、上記の通りです。

ベースサーバー：メイクファイル(Makefile)

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

```
# ネットワークインタフェース
NET_IF = ether

# イーサネット・デバイスドライバの選択
NET_DEV = if_ed

# ネットワーク層の選択
SUPPORT_INET4 = true

# トランスポート層の選択
SUPPORT_TCP = true

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

2006/10/07

TOPPERSプロジェクト認定

93



TINETを含むMakeFileは、JSPの標準的なMakefileとほぼ同じです。TINET用の各種設定を行った後で、TINET用のMakefile.configをインクルードします。このあと、TINET用の応用アプリのプログラム設定を行います。

NET_IFはネットワークインターフェースの選択を行います。Loop,ppp,etherより選択

NET_DEVは使用するデバイスを選択します。AKIH8_3069Fボードではif_edのみ

SUPPORT_INET4=trueはネットワーク層にIPV4を選択

SUPPORT_TCP=trueはトランスポート層にTCPを選択します。SUPPORT_UDPを同時設定可能。

ベースサーバー：静的ルーティング設定ファイル(route_cfg.c)

- 静的ルーティング情報として、tinet/netinet/in.h で定義されているT_IPV4EP型の構造体の配列を作成。
- 詳細はTINETのメインマニュアル(tinet/doc/tinet.txt)の6章を参照
- デフォルトゲートウェイのみのシンプルなネットワークでは、サンプルアプリケーションのroute_cfg.cをそのまま流用可能
- 静的経路表の書式

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

2006/10/07

TOPPERSプロジェクト認定

94



静的ルーティング設定ファイル(route_cfg.c)は、静的ルーティングの定義を行います。IPv4のルーティングエントリは、./tinet/netinet/in.hファイルで定義されています。静的ルーティングエントリは、予め決められたルーティング情報です。なお、デフォルトゲートウェイのみのシンプルなネットワークでは、サンプルアプリケーションのroute_cfg.c(以下)をそのまま流用できます。以下の各でファインはtinet_app_config.hを参照してください。

```
T_IN4_RTENTRY routing_tbl[NUM_ROUTE_ENTRY] = {
```

```
    /* 異なる LAN、default gateway による間接配送 */
    { 0,          0,          IPV4_ADDR_DEFAULT_GW      },

    /* 同一 LAN 内、直接配送 */
    { IPV4_ADDR_LOCAL &
      IPV4_ADDR_LOCAL_MASK, IPV4_ADDR_LOCAL_MASK,      0          },

    /* 同一 LAN 内へのブロードキャスト、直接配送 */
    { 0xffffffff,      0xffffffff,      0          },

};
```

ベースサーバー：コンフィギュレーション定義ファイル

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

```
/* ネットワーク統計情報の取得 */
#define SUPPORT_MIB
/* IPアドレス */
#define IPV4_ADDR_LOCAL 0xc0a8b262 /* 192.168.11.98 */
/* サブネットマスク */
#define IPV4_ADDR_LOCAL_MASK 0xffffffff /* 255.255.255.0 */
/* デフォルトゲートウェイ */
#define IPV4_ADDR_DEFAULT_GW 0xc0a80b01 /* 192.168.11.1 */
/* ルーティング表の静的ルーティングエントリ数 */
#define NUM_STATIC_ROUTE_ENTRY 3
/* 向け直し(ICMP)によるルーティング数。0を指定すると向け直しを無視 */
#define NUM_REDIRECT_ROUTE_ENTRY 0
```

コンフィギュレーション定義ファイル(`tinet_app_config.h`)は、アプリケーションプログラムに依存するパラメータ設定を行います。

`SUPPORT_MIB`はSNMP用管理情報ベース(MIB)に準拠したネットワーク統計の取得を有効にします。但し、`TINET`は、管理情報ベース(MIB)に準拠したネットワーク統計を提供するだけで、SNMPはサポートしていません。

`IPV4_ADDR_LOCAL`は、自分のIPアドレスを指定します。但し、PPPを使用する場合、相手に割り当ててもらう場合は0を指定します。

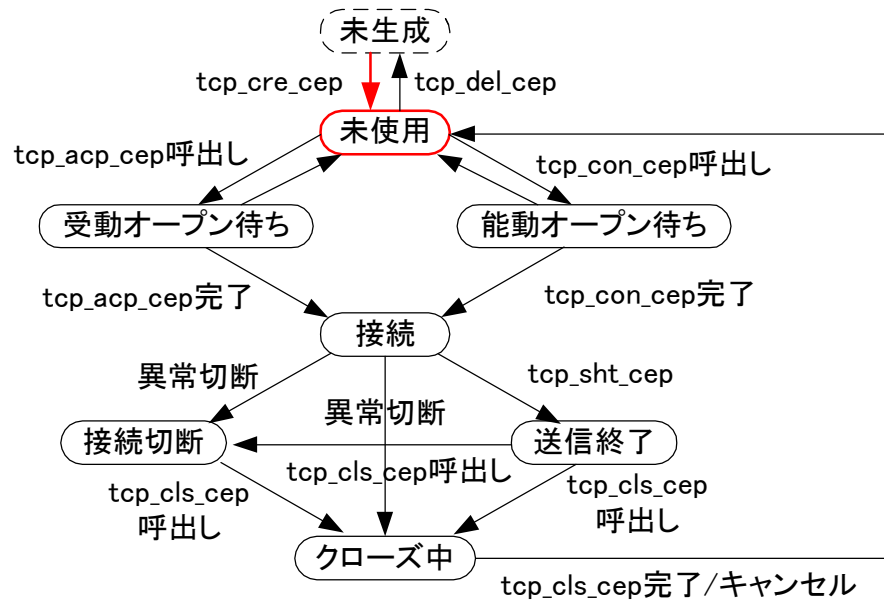
`IPV4_ADDR_LOCAL_MASK`は、ネットワークインターフェースがイーサネットの場合のみ有効で、サブネットマスクの指定を行う。

`IPV4_ADDR_DEFAULT_GW`は、ネットワークインターフェースがイーサネットの場合のみ有効で、デフォルトゲートウェイを指定します。

`NUM_STATIC_ROUTE_ENTRY`は、ルーティング表の静的ルーティングエントリ数を指定します。

`NUM_REDIRECT_ROUTE_ENTRY`は、向け直し(ICMP)によるルーティングエントリ数を指定します。0の場合、向け直し(ICMP)を無視する指定となります。

ベースサーバー：TCP通信端点の生成



2006/10/07

TOPPERSプロジェクト認定

96



TCP通信端点、TCP受付口の生成手順について説明を行います。TINETの場合、通信端点は静的APIで生成します。この場合、コンパイル、リンク時にオブジェクトは生成されますので、tcp_dep_cepを用いて、これを削除することはできません。

ベースサーバー：TINETコンフィギュレーション(tinet_tcp_srv.cfg)

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

- TCP受付口生成静的API

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

- ベースサーバーでは受付口を1個作成

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

自分の持つ全てのIP
アドレスに対する接
続要求を待ち受ける

1234

2006/10/07

TOPPERSプロジェクト認定

97



TCP受付口はTCPのサーバーアプリケーションのみ必要となります。クライアントからの接続要求を待つオブジェクトです。サーバーのIPアドレスとポート番号を属性として持ちます。複数のタスクから共用できます。

ITRON TCP/IP API仕様では以下の通りです。

【静的API】

```
TCP_CRE_REP(ID repid, {ATR repatr, {UP myipaddr, UH myportno }});
```

【パラメータ】

ID	repid	TCP受付口ID(自動割当)
ATR	repatr	TCP受付口属性
UP	myaddr	自分側のIPアドレス
UH	myportno	ポート番号

自分側のIPアドレスにIPV4_ADDRANY(=0)を指定した場合、自分のもつすべてのIPアドレスに対する接続要求を待ち受ける指定となります。

ベースサーバーではTCP受付口生成静的APIはTINETのコンフィギュレーションファイル(tinet_tcp_srv.cfg)に記述します。

ベースサーバー：TINETコンフィギュレーション(tinet_tcp_srv.cfg)

- TCP通信端点の生成のための静的API

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

- ベースサーバーではTCP通信端点を1個作成

```
TCP_CRE_CEP (TCP_SRV_CEPID, {
    0,
    tcp_srv_cep_sbuf, TCP_SRV_CEP_SBUF_SIZE,
    tcp_srv_cep_rbuf, TCP_SRV_CEP_RBUF_SIZE,
    NULL
})
```

2006/10/07

TOPPERSプロジェクト認定

98



TCP通信端点はTCP通信端点の生成のための静的APIを用いて生成を行います。TINETの場合、送信と受信のためのバッファは自分で確保しなければなりません。ITRON TCP/IP API仕様でのTCP通信端点の生成の静的APIは以下の通りです。

【静的API】

```
TCP_CRE_CEP(ID cepid, { ATR cepatr, VP sbuf, INT sbufsiz, VP rbuf, INT rbufsz,
                        FP callback});
```

【パラメータ】

ID	cepid	TCP通信端点ID(自動割当)
ATR	cepatr	TCP通信端点属性
VP	sbuf	送信用のウィンドウバッファの先頭アドレス
INT	sbufsz	送信用のウィンドウバッファのサイズ
VP	rbuf	受信用のウィンドウバッファの先頭アドレス
INT	rbufsz	受信用のウィンドウバッファのサイズ
FP	callback	コールバックルーチンのアドレス

ベースサーバーではTCP通信端点をひとつ使用します。

ベースサーバー：TINETコンフィギュレーション(tinet_tcp_srv.cfg)

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

```
#define TCP_SRV_CEP_SBUF_SIZE      (1024*4)
#define TCP_SRV_CEP_RBUF_SIZE      (1024*4)
```

- tcp_srv.c

```
UB tcp_srv_cep_sbuf[TCP_SRV_CEP_SBUF_SIZE];
UB tcp_srv_cep_rbuf[TCP_SRV_CEP_RBUF_SIZE];
```

TCP通信端点で使用する送受信のウィンドウバッファは、アプリケーションで用意する必要があります。ベースサーバーでは、ウィンドウバッファサイズをtcp_srv.hインクルードファイルで定義しています。また、送受信バッファ領域はユーザプログラムのtcp_srv.c中で確保しています。

ベースサーバー：TINETコンフィギュレータ実行

- make tinetでTINETのコンフィギュレーターを実行する
 - tinet_tcp_srv.cfg をプリプロセッサに通してtmpfile9にリダイレクト
 - 生成したtmpfile9に対してコンフィギュレータ(tinet_cfg)を実行

```
$ cd MY_TCP
$ make realclean
$ ls
Makefile      tcp_srv.c    tcp_srv.h    tinet_tcp_srv.cfg
route_cfg.c   tcp_srv.cfg  tinet_app_config.h
$ make depnd
h8300-hms-gcc -E -I. -I../include -I../config/h8/akih8_3069f -I../config/h8 -I../
../tinnet/netdev/if_ed -I../tinnet -DCPU_CLOCK=200000000 -DLABEL_ASM -DVEC
TOR_SIZE=64 -DUSE_PING -DUSE_NETAPP_SUBR -DUNDEF_TCP_CFG_PASSIV
E_OPEN -DUNDEF_TCP_NON_BLOCKING -DUNDEF_UDP_CFG_NON_BLOCKING
-DISUPPORT_INET4 -DISUPPORT_ETHER -DISUPPORT_TCP -DTCP_CFG_LIBRAR
Y -DUDP_CFG_LIBRARY -DTNCT_MONITOR -x c-header tinet_tcp_srv.cfg > tmpfile9
../tinnet/cfg/tinet_cfg -s tmpfile9
rm -f tmpfile9
:
$ ls
Makefile      kernel_id.h   tcp_srv.cfg   tinet_id.h
Makefile.depend kernel_obj.dat tcp_srv.h     tinet_kern.cfg
Kernel_cfg.c   route_cfg.c   tinet_app_config.h tinet_tcp_srv.cfg
Kernel_chk.c   tcp_srv.c     tinet_cfg.c   vector.S
```

2006/10/07

TOPPERSプロジェクト認定

100



上の図は、ベースサーバーの構築時のコマンドと表示です。TINETのコンフィギュレータは「make tinet」コマンドにて実行します。まず、TINETのコンフィギュレーションファイルtinnet_tcp_srv.cfgがプリプロセッサにより変換され、tmpfile9に書き込まれます。このtmpfile9に対して、TINETのコンフィギュレータが実行し、以下のファイルを生成します。

tinnet_id.h TINETの自動割付IDのファイル

tinnet_cfg.c TCP受付口やTCP通信端点などの領域設定

tinnet_kern.cfg TINETで使用するカーネルオブジェクトのコンフィギュレーションファイル

ベースサーバー：TINETコンフィギュレータ生成ファイル(1/2)

- tinet_id.h
 - TINETのオブジェクトのID自動割付け結果が定義されている

```
#ifndef _TINET_ID_H_
#define _TINET_ID_H_
#define TCP_SRV_CEPID 1
#define TCP_SRV_REPID 1
#endif /* of _TINET_ID_H_ */
```

- tinet_kern.cfg
 - 静的APIによる受付口、通信端点の生成により、TINET内部で使用するカーネルオブジェクト
 - カーネルコンフィギュレーションファイル(tcp_srv.cfg)からインクルードするTINETコンフィギュレーションファイル(./tinnet/tinet.cfg)からインクルードされる

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

2006/10/07

TOPPERSプロジェクト認定

101



自動生成された`tinnet_id`には、TCP受付口ID(`TCP_SRV_CEPID`)に1が、TCP通信端点(`TCP_SRV_REPID`)にも1が割り当てられています。

`tinnet_kern.cfg`にはTINET内部で使用するカーネルオブジェクトの静的APIが定義されています。このコンフィギュレーションファイルは、ユーザープログラムのコンフィギュレーションファイル(`tcp_srv.cfg`)からインクルードされ、JSPのコンフィギュレータにより、カーネルオブジェクトが生成されます。

ベースサーバーでは3つのセマフォとひとつのイベントフラグを使用します。

ベースサーバー：TINETコンフィギュレータ生成ファイル(2/2)

- `tinnet_cfg.c`
 - 静的APIによる受付口、通信端点の生成により、TINET内部で使用する構造体

```
...
#define TNUM_TCP_CREPID      1
const ID tmax_tcp_crepid = (TMIN_TCP_CREPID + TNUM_TCP_CREPID - 1);
T_TCP_CREP tcp_crep[TNUM_TCP_CREPID] = {
    {0, { IPV4_ADDRANY, 1234 }, },
};

#define TNUM_TCP_CCEPID      1
const ID tmax_tcp_ccepid = (TMIN_TCP_CCEPID + TNUM_TCP_CCEPID - 1);
T_TCP_CCEP tcp_ccep[TNUM_TCP_CCEPID] = {
    {0,
     tcp_srv_cep_sbuf,
     ( 1024 * 4 ),
     .....},
};
```

`tinnet_cfg.c`では、ベースサーバーで使用するTCP受付口とTCP通信端点の構造体領域が確保されています。

TCP受付口の構造体の`T_TCP_CREP`とTCP通信端点の構造体`T_TCP_CCEP`は`tinnet/netinet/tcp_var.h`に定義されています。

ベースサーバー : ユーザープログラム

- コンフィギュレーションファイル
 - TINETコンフィギュレーションファイルをインクルード

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

- タスクを1個生成

```
CRE_TSK(TCP_SRV_TASK, {
    TA_HLNG|TA_ACT, 0,
    tcp_srv_task,
    TCP_SRV_MAIN_PRIORITY,
    TCP_SRV_STACK_SIZE, NULL});
```

- TINET標準インクルードファイル

```
#include "tinnet_id.h"
#include <netinet/in.h>
#include <netinet/in_itron.h>
```

2006/10/07

TOPPERSプロジェクト認定

103



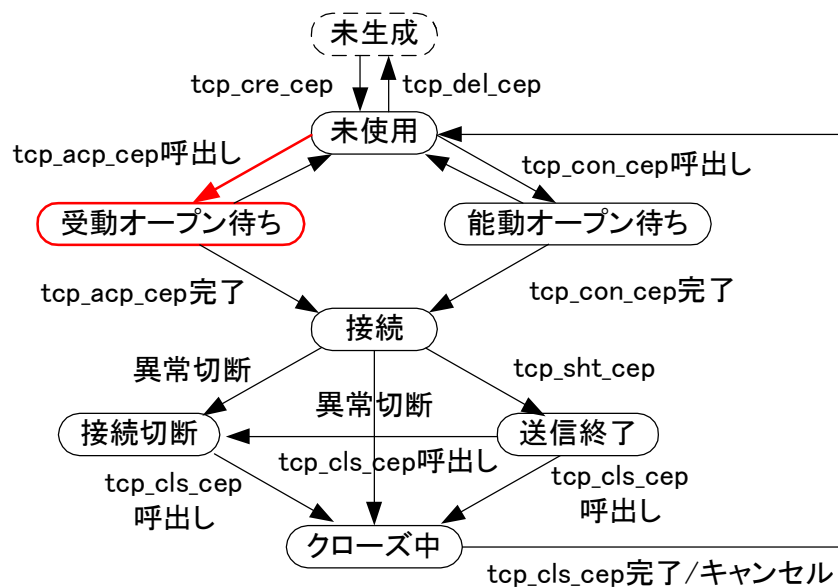
ベースサーバープログラムは以下の3つのファイルから構成されています。

- tcp_srv.cfg コンフィギュレーションファイル
- tcp_srv.h インクルードファイル
- tcp_srv.c ベースサーバー本体のC言語ファイル

コンフィギュレーションファイル (tcp_srv.cfg) は、TINETのコンフィギュレータで生成したカーネルオブジェクト用のコンフィギュレーションファイル (tinnet.cfg) をインクルードする必要があります。また、ベースサーバー機能を実行するタスク(プログラムはtcp_srv.cに記述)を生成するCRE_TSK静的APIを定義します。

ユーザープログラム (tcp_srv.c) には、上記のようにTINET用の標準インクルードファイルをインクルードする必要があります。

ベースサーバー：受動オープン



2006/10/07

TOPPERSプロジェクト認定

104



受動オープンについて説明します。以後のプログラムはユーザープログラム(tcp_srv.c)中の記述の説明となります。

ベースサーバー：受動オープン

- 受動オープンAPI

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

- ベースサーバー

```
T_IPEP dst;

tcp_acp_cep(
    TCP_SRV_CEPID, TCP_SRV_REPID,
    &dst,
    TMO_FEVR)
```

2006/10/07

TOPPERSプロジェクト認定

105



tcp_srv.cの44行目から64行目までを以下に載せます。59行目のtcp_acp_cepサービスコールにより、受動オープンを行います。クライアントからの接続が成立するとE_OKが返り、ip2strの行に移行します。エラーが発生しE_OKが返らない場合は、再度、tcp_acp_cepサービスコールを実行し受動受付を行います。

```
void
tcp_srv_task(VP_INT exinf)
{
    static UB addr[sizeof("0123:4567:89ab:cdef:0123:4567:89ab:cdef")];
    ID      tskid;
    T_IPEP  dst;
    ER      error = E_OK;
    char    c;
    int rlen;
    SYSTIM  now;

    get_tid(&tskid);
    syslog(LOG_NOTICE, "[TCP SRV :%d,%d] started.", tskid, (INT)exinf);
    while(TRUE){
        if (tcp_acp_cep(TCP_SRV_CEPID, TCP_SRV_REPID, &dst, TMO_FEVR) !=
E_OK){
            syslog(LOG_NOTICE, "[TES:%02d ACP] error: %s", TCP_SRV_CEPID,
itron_strerror(error));
            continue;
        }
        ip2str(addr, &dst.ipaddr);
        get_tim(&now);
```

ベースサーバー：データの受信

- データ受信API

```
tcp_rcv_dat(
    TCP通信端点ID,
    バッファ, バッファサイズ,
    タイムアウト)
```

- 戻り値

- 受信文字数
- 0の場合は、通信相手がコネクションを切断
- 負の場合はエラー

2006/10/07

TOPPERSプロジェクト認定

106



`tcp_srv.c`の63行目から80行目までを以下に載せます。63行目から66行目までは受信が確立したクライアントのIPアドレスと時間を表示します。69行目の`tcp_rcv_dat`サービスコールでクライアントからのデータ受信を行います。戻り値がゼロの場合はクライアントがコネクションを切断した場合の返回值です。マイナスはエラーの発生です。これらの場合、`break`文にて`while`文を抜け`err_fin`に移行します。ゼロより大きな返回值は受信のデータサイズです。この場合、`syslog`文にてコンソールの表示を行い、再び、`tcp_rcv_dat`サービスコールに入り、次のデータ待ちとなります。

```
ip2str(addr, &dst.ipaddr);
get_tim(&now);

syslog(LOG_NOTICE, "[TES:%02d ACP] connected: %6d, from: %s.%d", TCP_SRV_CEPID, now / SYSTIM_HZ, addr,
dst.portno);

while (TRUE) {
    if ((rlen = tcp_rcv_dat(TCP_SRV_CEPID, rbuffer, BUF_SIZE - 1, TMO_FEVR)) <= 0) {
        if (rlen != E_OK)
            syslog(LOG_NOTICE, "[TES:%02d RCV] error: %s,%d", TCP_SRV_CEPID, itron_strerror(rlen));
        break;
    }
    rbuffer[rlen] = '\0';
    syslog(LOG_NOTICE, "Recive Data %s", rbuffer);
}

err_fin:
    if ((error = tcp_cls_cep(TCP_SRV_CEPID, TMO_FEVR)) != E_OK)
```

ベースサーバー：データの受信

- ベースサーバー
 - 受信文字数をrlenに、受信データをrbufferに格納

```
rlen = tcp_rcv_dat(
    TCP_SRV_CEPID,
    rbuffer, BUF_SIZE - 1,
    TMO_FEVR)
```

tcp_rcv_datサービスコールのITRON TCP/IP API仕様を以下に載せます。
このサービスコールは標準(非省コピーAPI)の受信サービスコールとなります。

【C言語API】

```
ER ercd = tcp_rcv_dat(ID cepid, INT len, TMO tmout);
```

【パラメータ】

ID	cepid	TCP通信端点ID
VP	data	受信データを入れる領域の先頭アドレス
INT	len	受信したいデータサイズの長さ
TMO	tmout	タイムアウト指定

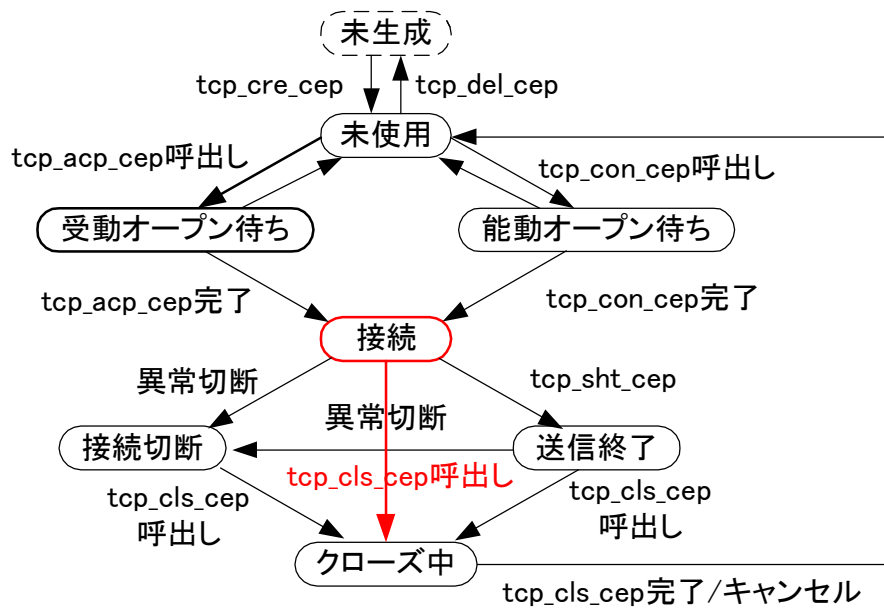
【リターンパラメータ】

ER	ercd	取り出したデータの長さ/エラーコード
----	------	--------------------

【エラーコード】

正の値	正常終了(取り出したデータの長さ)
0	データ終結(接続が正常切断された)
負の値	エラーコード

ベースサーバー：コネクションの切断



2006/10/07

TOPPERSプロジェクト認定

108



コネクションの切断について説明します。

ベースサーバー : コネクションの切断

- コネクション切断API

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

- ベースサーバー

```
tcp_cls_cep(TCP_SRV_CEPID, TMO_FEVR)
```

2006/10/07

TOPPERSプロジェクト認定

109

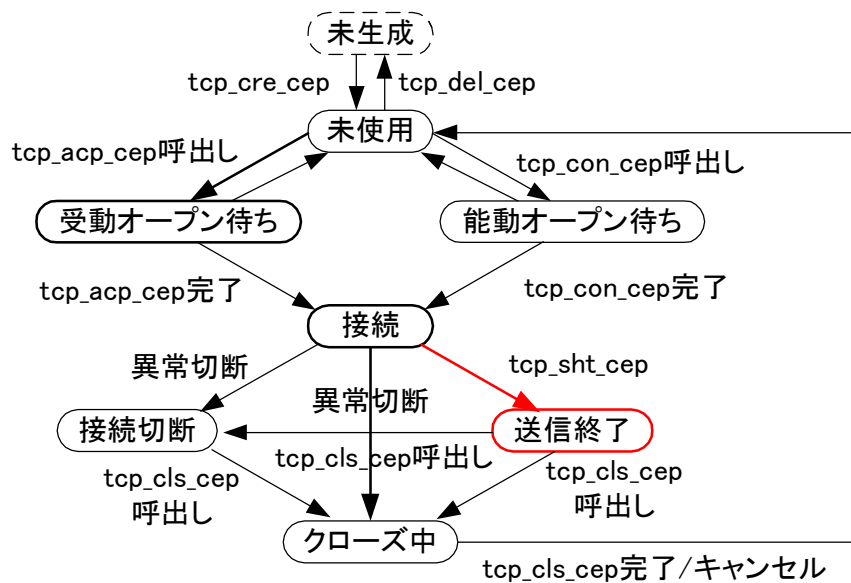


`tcp_srv.c`の78行目から87行目までを以下に載せます。コネクションの切断は`tcp_cls_cep`サービスコールを用いて行います。ベースサーバーでは、クライアントが正常に切断した場合、または、エラーが発生した場合にコネクションの切断を行います。`tcp_cls_cep`サービスコール実行後、現在時間を表示して、受動オープン待ちに移行します。

```
err_fin:
    if ((error = tcp_cls_cep(TCP_SRV_CEPID, TMO_FEVR)) != E_OK)
        syslog(LOG_NOTICE, "[TES:%02d CLS] error: %s", TCP_SRV_CEPID, itron_strerror(error));

    get_tim(&now);
    syslog(LOG_NOTICE, "[TES:%02d FIN] finished: %6d",
        TCP_SRV_CEPID, now / SYSTIM_HZ);
}
```

ベースサーバー：送信終了通知



2006/10/07

TOPPERSプロジェクト認定

110



ベースサーバーには実装されていませんが、送信終了通知について説明します。送信終了通知は `tcp_shut_cep` サービスコールを用いて行います。

ベースサーバー : tcp_sht_cep と tcp_cls_cep の違い

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

tcp_cls_cep サービスコールでは、通信端点をクローズし、これ以降通信端点を使用できません。tcp_cls_cep サービスコールではTCP通信端点が未使用状態になるのを待ってからリターンするため、リターン後はTCP通信端点を再利用できます。tcp_sht_cep サービスコールはデータ送信を終了します。そのため、これ以降のデータ送信は不可能となります。tcp_sht_cep は、切断の手配を行うだけで、API内で待ち状態とはなりません。このサービスコールは通信相手に送信を終了したことを通知するために使用します。

サーバープログラムはクライアントが要求終了したかの判定ができないので主導的にコネクションは切れません。クライアントは自分の要求が終了したことはわかりますが、サーバーからのデータがすべて到着したかの判定ができません。そこで、クライアントがサーバーへの要求が終了したことを通知するために、tcp_sht_cep サービスコールを使用します。

TCPサーバープログラミング

1. ベースサーバー
- [2. エコーサーバー](#)
3. デバイス操作サーバー
4. マルチタスクサーバー

エコーサーバー : ECHO_SRV

- ベースサーバーを改造して受信した文字を一つ進めて返すエコーサーバーを作成します
- ベースサーバーに以下の処理を加える必要があります
 - 受信文字の加算
 - データの送信
- 先に作成したMY_SRVを改造してエコーサーバーを作成してください

ベースサーバーを改造して、エコーサーバーを作成します。

エコーサーバーは、クライアントから受信したデータキャラクタに1を加えて、クライアントに送り返します。改造は以下のプログラムです。

`tcp_srv.c`のみです。ベースサーバーのプログラムに受信したデータ文字に1加算して、送信バッファに書き込み、送信を行うように改造します。

エコーサーバー：データ送信

- データ送信API

```
tcp_snd_dat(
    TCP通信端点ID,
    バッファ, 送信サイズ,
    タイムアウト)
```

- 戻り値

- 送信文字数
- 負の場合はエラー

- 送信バッファ

- 送信文字列を作成するためのバッファを確保

```
static UB sbuffer[BUF_SIZE];
```

2006/10/07

TOPPERSプロジェクト認定

114



データの送信は、tcp_snd_datサービスコールを用いて行います。以下にITRON TCP/IP API仕様を載せます。

【C言語API】

```
ER ercd = tcp_snd_dat(ID cepid, VP data, INT len, TMO tmout);
```

【パラメータ】

ID	cepid	TCP通信端点ID
VP	data	送信データの先頭アドレス
INT	len	送信したいデータの長さ
TMO	tmout	タイムアウト時間

【リターンパラメータ】

ER	ercd	送信バッファに入れたデータの長さ/エラーコード
----	------	-------------------------

【エラーコード】

正の値	正常終了(送信バッファに入れたサイズ)
負の値	エラーコード

送信文字列を作成するためのバッファをtcp_srv.cに追加します。

エコーサーバー：データ送信

- 受信した文字列の処理

```
for(i = 0; i < rlen; i++){
    if(rbuffer[i] >= ' ')
        sbuffer[i] = rbuffer[i] + 1;
    else
        sbuffer[i] = rbuffer[i];
}
```

- 送信処理

```
char *psdata;
....
slen_remain = rlen;
psdata = sbuffer;
do{
    if ((slen = tcp_snd_dat(TCP_SRV_CEPID,
                           psdata, slen_remain, TMO_FEVR)) < 0) {
        syslog(LOG_NOTICE, "[TES:%02d SND] .....");
        goto err_fin;
    }
    slen_remain -= slen;
    psdata += slen;
}while(slen_remain > 0);
```

2006/10/07

TOPPERSプロジェクト認定

115



受信した文字に1足します。コントロールコードは変えないようにするために、**rbuffer[i]**がスペースコード(' ')より大きい場合を対象に送信バッファ(**sbuffer**)にコピーします。送信には**tcp_snd_dat**サービスコールを使用します。送信サイズを**slen_remain**にセットしておき、**tcp_snd_data**の返り値を**slen**に入れておき、**slen**がゼロ以上の場合、**slen_rmain**から**slen**を引き、まだ、送っていないデータサイズを計算します。送信した分だけ**psdata**のポインタを進めます。**slen_remain**がゼロより大きい場合、また、送信が終了していないデータがあるので、再度、**tcp_snd_dat**サービスコールにて送信を行います。

TCPサーバープログラミング

1. ベースサーバー
2. エコーサーバー
3. [デバイス操作サーバー](#)
4. マルチタスクサーバー

TCPクライアントプログラム

ITRON TCP/IP API仕様を用いて,
TCPクライアントプログラムを作成します

- 完成したプログラムは以下のディレクトリにあります
 - ./OBJ/AKIH8_3069F/
 - TCP_CLIENT : ベースクライアント
 - DIP_MONITOR : DIP状態通知クライアント
- Windowsで動作するTCPサーバーが以下のディレクトリにあります
 - ./environment/tcp_server/tcp_server.exe

2006/10/07

TOPPERSプロジェクト認定

117



この章ではボードをクライアントとして、パソコンをサーバーにしたシステムを作成します。TCPクライアントの完成プログラムは./OBJ/AKIH8_3069F上のTCP_CLIENTとDIP_MONITORの2つのディレクトリ上にあります。パソコン上には、Windows用のサーバープログラムが必要になります。このプログラムはサーバーとして起動し、クライアントから送られたデータをコンソール上に表示するプログラムです。

デバイス操作サーバー : DEV_SRV

- エコーサーバーを改造してデバイス操作するサーバーを作成しましょう。
- データを受け取り
 - 0ならLED1,2共に消灯
 - 1ならLED1を点灯, LED2を消灯
 - 2ならLED1を消灯, LED2を点灯
 - 3ならLED1,2共に点灯
 - それ以外は無視
- データを受け取ったらディップスイッチの状態を4文字の文字列で返す
 - 例 ON, OFF, ON, OFF なら 1010を返す
- デバイスの操作は1日目に作成したdevice.cを流用
- ハードウェアの初期化を忘れずに！

2006/10/07

TOPPERSプロジェクト認定

118



エコーサーバーを改造して、デバイス操作サーバーを作ります。エコーサーバーがうまく作れなかった人は、正解例のエコーサーバーを元に作り変えましょう。

```
$ cd ..                                './OBJ/AKIH8_3068F'ディレクトリに移動
```

```
$ rm -rf MY_SRV                        'MY_SRV'を削除
```

```
$ cp -r ECHO_SRV MY_SRV               'ECHO_SRV'をコピー
```

改造の仕様は、

1) telnetから0から3までの数字を送信し、サーバーがデータを受信すると

0ならLED1とLED2を消灯

1ならLED1を点灯、LED2を消灯

2ならLED1を消灯、LED2を点灯

3ならLED1とLED2を点灯します。

2) LEDの処理の後、DIP-SWの状態を読み込みON, OFF状態に従って

DIP-SW1から4に対応した4文字の数字(0または1)を送り返します。

DIP-SWがONの時は1、OFFの時は0を返します。

デバイスの操作は1日目に作成したdevice.cを./OBJ/AKIH8_3069F/deviceからコピーして使用します。

デバイス操作サーバー：初期化ルーチン

- 初期化ルーチン
 - デバイスの初期化

```
void
device_init(VP_INT exinf){
    initial_switch(); /* スイッチの初期化 */
    initial_led();    /* LEDの初期化 */
}
```

- 初期化ルーチンとして呼び出す

```
ATT_INI({TA_HLNG, 0, device_init});
```

- Makefileの編集
 - device.cをコンパイル対象に加える

```
UTASK_COBJS = $(UNAME).o device.o
```

デバイスの初期化ルーチンdevice_init()をtcp_srv.cに追加し、device_initのプロトタイプ宣言をtcp_srv.hに追加します。

初期化ルーチンとして呼び出すために、tcp_srv.cfgにATT_INI静的APIを追加します。

MakefileのUTASK_COBJSにdevice.oを追加し、コンパイル対象に加えます。

デバイス操作サーバー : LED操作関数

- 受信した文字を引数にLEDを操作する

```
void  
led_control(char c){  
    switch(c){  
        case '0':  
            set_led(LED1,OFF);  
            set_led(LED2,OFF);  
            break;  
        case '1':  
            set_led(LED1,ON);  
            set_led(LED2,OFF);  
            break;  
        .....  
        default:  
            break;  
    }  
}
```

2006/10/07

TOPPERSプロジェクト認定

120



受信した文字からLEDを操作する関数`led_control()`を`tcp_srv.c`に追加します。引数は受信した文字列の最初1文字です。

デバイス操作サーバー : DIP状態取得

- DIPスイッチの状態を取得して文字列に変換

```
int dip_monitor(char *buffer){
    if (get_switch(SW1) == ON) {
        buffer[0] = '1';
    }else{
        buffer[0] = '0';
    }
    if (get_switch(SW2) == ON) {
        buffer[1] = '1';
    }else{
        buffer[1] = '0';
    }
    .....
    buffer[4] = '\n';
    buffer[5] = '\r';
    buffer[6] = '\0';
    syslog(LOG_NOTICE, "Dip state is %s", buffer);
    return 6;
}
```

2006/10/07

TOPPERSプロジェクト認定

121



DIPスイッチの状態を取得して文字列に変換する関数`dip_monitor()`を`tcp_srv.c`に追加します。引数はキャラクタへのポインタで、この文字列領域にDIP-SWの状態を書き込み、`syslog`文で内容をコンソールに表示します。戻り値として、作成した文字列のサイズを返します。

デバイス操作サーバー：データ送受信

- データを受信したらLEDを操作し、DIPの状態を取得してクライアントに送信する

```

rbuffer[rlen] = '\0';
syslog(LOG_NOTICE, "Recive Data %s", rbuffer);
/* LEDの操作 */
led_control(rbuffer[0]);
/* DIP状態の取得 */
slen_remain = dip_monitor(sbuffer);
psdata = sbuffer;
while(slen_remain > 0){
    if ((slen = tcp_snd_dat(TCP_SRV_CEPID, psdata,
                           slen_remain, TMO_FEVR)) < 0) {
        syslog(LOG_NOTICE, "[TES:%02d SND] .....);
        goto err_fin;
    }
    slen_remain -= slen;
    psdata += slen;
}

```

2006/10/07

TOPPERSプロジェクト認定

122



tcp_srv.cのtcp_srv_tskタスクを改造します。

TCPサーバープログラミング

1. ベースサーバー
2. エコーサーバー
3. デバイス操作サーバー
4. [マルチタスクサーバー](#)

マルチタスクサーバー : MUL_SRV

- デバイス操作サーバーをベースに一つの受付口で2個のクライアントからの接続要求を受け付けるように改造しましょう。
- 単一のTCP受付口で複数のタスクが待つことが可能。
- タスク毎にTCP通信端点を生成します。
- タスクは2個生成し、コードは共有する。
- タスクに個別の情報は配列に入れ、拡張情報(exinf)をインデックスに取得(sample1を参照)
- グローバル変数として確保しているバッファ等は2個分用意する。
- LEDの操作に関しては排他制御をしなくてもよい

2006/10/07

TOPPERSプロジェクト認定

124



デバイス操作サーバーを改造して、2つのクライアントに対応できるマルチタスクサーバーを作成します。改造方法は、ひとつのTCP受付口に2つのTCP通信端点を2つのタスクから対応付けを行います。

tinnet_tcp_srv.cfg、tcp_srv.cとtcp_srv.hを改造して2つのTCP通信端点を生成します。tcp_srv.cfgとtcp_srv.cを改造してtcp_srv_tskを2つ生成します。送受信バッファを配列として2重化して2つのタスクでTCP通信端点と送受信バッファは別のものを使用するように修正を行います。

LEDの操作については、排他制御をしなくても、良いこととします。

マルチタスクサーバー

: TINETコンフィギュレーション(tinet_tcp_srv.cfg)

- TCP通信端点をひとつ増やす
 - バッファをそれぞれ用意する

```
TCP_CRE_CEP (TCP_SRV_CEPID1, {
    0,
    tcp_srv_cep_sbuf1, TCP_SRV_CEP_SBUF_SIZE,
    tcp_srv_cep_rbuf1, TCP_SRV_CEP_RBUF_SIZE,
    NULL
});
```

```
TCP_CRE_CEP (TCP_SRV_CEPID2, {
    0,
    tcp_srv_cep_sbuf2, TCP_SRV_CEP_SBUF_SIZE,
    tcp_srv_cep_rbuf2, TCP_SRV_CEP_RBUF_SIZE,
    NULL
});
```

2006/10/07

TOPPERSプロジェクト認定

125



tinet_tcp_srv.cfgを改造して、TCP通信端点を一つ追加します。このとき、通信バッファを別々にするためにTCP_SRV_CEPID1用の送受信バッファを以下のように修正する。

送信ウィンドウバッファ:tcp_srv_cep_sbuf1

受信ウィンドウバッファ:tcp_srv_cep_rbuf1

TCP_SRV_CEPID2用の送受信バッファは以下の設定とします。その他はTCP_SRV_CEPID1と同等となります。

送信ウィンドウバッファ:tcp_srv_cep_sbuf2

受信ウィンドウバッファ:tcp_srv_cep_rbuf2

マルチタスクサーバー：バッファ確保

- 送受信バッファ(TCPウィンドウバッファ):tcp_srv.c
 - 2個分確保

```
UB tcp_srv_cep_sbuf1[TCP_SRV_CEP_SBUF_SIZE];
UB tcp_srv_cep_rbuf1[TCP_SRV_CEP_RBUF_SIZE];
UB tcp_srv_cep_sbuf2[TCP_SRV_CEP_SBUF_SIZE];
UB tcp_srv_cep_rbuf2[TCP_SRV_CEP_RBUF_SIZE];
```

- タスクで用いる送受信バッファ
 - 2個分確保

```
static UB rbuffer[2][BUF_SIZE];
static UB sbuffer[2][BUF_SIZE];
```

TCP通信端点に使用する送受信ウィンドウバッファをtcp_sev.cに追加修正します。同様に送受信ウィンドウバッファのプロトタイプ宣言をtcp_srv.hに追加修正します。

tcp_srv.c中の2つのサーバータスクで使用する送受信バッファを配列化します。

マルチタスクサーバー：TCP通信端点の情報取得

- グローバル変数を用意してTCP通信端点のIDを格納
- タスクへの引数(exifnf)をインデックスに情報を取得

```
int tcp_cep_id[] = {TCP_SRV_CEPID1, TCP_SRV_CEPID2};
```

- タスクの生成
 - 2個生成

```
CRE_TSK(TCP_SRV_TASK1, {
    TA_HLNG|TA_ACT, 0,
    tcp_srv_task,
    TCP_SRV_MAIN_PRIORITY,
    TCP_SRV_STACK_SIZE,
    NULL
});
```

```
CRE_TSK(TCP_SRV_TASK2, {
    TA_HLNG|TA_ACT, 1,
    tcp_srv_task,
    TCP_SRV_MAIN_PRIORITY,
    TCP_SRV_STACK_SIZE,
    NULL
});
```

2つのタスクのtcp_srv_taskへの引数を0, 1として、各タスクでTCP通信端点IDを別々に取得できるように、TCP通信端点IDを初期データとしたグローバル変数の配列を用意します。

マルチタスクサーバー：プログラム

- TCP通信端点のIDはexinfからタスク起動時に取得

```
INT cepid;
cepid = tcp_cep_id[(INT)exinf];
```

```
tcp_acp_cep(cepid, TCP_SRV_REPID, &dst, TMO_FEVR)
```

- バッファもexinfにより指定

```
tcp_rcv_dat(cepid, rbuffer[(INT)exinf], BUF_SIZE - 1, TMO_FEVR)
```

```
/* LEDの操作 */
led_control(rbuffer[(INT)exinf][0]);
/* DIP状態の取得 */
slen_remain = dip_monitor(sbuffer[(INT)exinf]);
```

tcp_srv_tsk関数では、起動時にexinfからグローバル配列を介して、2つのタスクで別々のTCP通信端点IDをcepidに取り込みます。また送受信バッファもexinfの配列を介して、別々のバッファを指定して、受動オープン、受信、送信を行います。

TCPクライアントプログラミング

1. TCP/IPの基礎
2. ITRON TCP/IP仕様
3. TINET (TCP/IPスタック)
4. TCPサーバープログラム
5. TCPクライアントプログラム
6. まとめ

TCPクライアントプログラム

ITRON TCP/IP API仕様を用いて,
TCPクライアントプログラムを作成します

- 完成したプログラムは以下のディレクトリにあります
 - ./OBJ/AKIH8_3069F/
 - TCP_CLIENT : ベースクライアント
 - DIP_MONITOR : DIP状態通知クライアント
- Windowsで動作するTCPサーバーが以下のディレクトリにあります
 - ./environment/tcp_server/tcp_server.exe

2006/10/07

TOPPERSプロジェクト認定

130



この章ではボードをクライアントとして、パソコンをサーバーにしたシステムを作成します。TCPクライアントの完成プログラムは./OBJ/AKIH8_3069F上のTCP_CLEINTとDIP_MONITORの2つのディレクトリ上にあります。パソコン上には、Windows用のサーバープログラムが必要になります。このプログラムはサーバーとして起動し、クライアントから送られたデータをコンソール上に表示するプログラムです。

TCPクライアントプログラミング

1. ベースクライアント

2. DIP状態通知クライアント

ベースクライアント : TCP_CLIENT

- 実行されるとシリアルコンソールからのキー入力を待ち、キー入力となされると、IPアドレス192.168.11.6のポート1234のサーバーに接続を試みる
- 接続が成功したら、シリアルコンソールからの入力をエンターが入力されるまで読み込む
- 読み込んだ入力をサーバーに送る。
- Ctrl-Cもしくは'Q'が入力されると終了通知("_end_")をサーバーに送り、サーバーからの終了通知を待つ
- 接続はサーバーから切断され、サーバーからの終了通知を受け取るとポートをクローズする

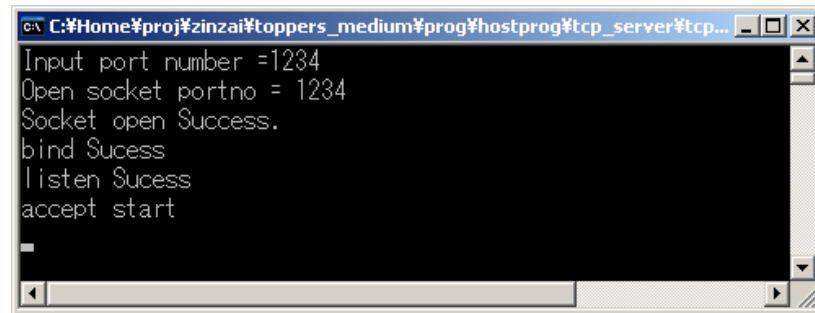
TCP_CLIENTプログラムは、ボードをクライアントとしてパソコンにアクセスするプログラムです。このプログラムは起動すると、「[TCP CLIENT] Push any key to connent」を表示してキー待ちとなります。キー入力により、サーバーに対して能動オープンを行います。サーバーとコネクションが成立すると、データ入力した文字列をサーバーに送信します。サーバーは文字列を表示します。

クライアントプログラムを終了させるには、Ctrl-Cまたは'Q'を入力します。これにより、サーバーに文字列"_end_"を送ります。その後クライアントは"_end_"受信待ちとなります。

サーバーは"_end_"を受信すると、クライアントに"_end_"を送り返し、シャットダウン、クローズします。クライアントは"_end_"を受信するとTCP通信端点クローズします。

ベースクライアント : サーバー

- プログラムディレクトリ
/environment/tcp_server/tcp_server.exe を実行
- 実行を開始するとポート番号の入力になるのでポート番号を"1234"と入力してエンター
- ポートのオープンが成功すると"accept start"となる



```

C:\Home\proj\zinzai\toppers_medium\prog\hostprog\tcp_server\tcp...
Input port number =1234
Open socket portno = 1234
Socket open Success.
bind Success
listen Success
accept start

```

2006/10/07

TOPPERSプロジェクト認定

133



パソコン上でサーバープログラムを実行します。このプログラムは/environment/tcp_server/ディレクトリ上に実行プログラムがあり、これをダブルクリックして実行してください。実行すると「Input port num =」を表示します。ここでポート番号"1234"を入力してEnterキーを押してください。以下の手順で受動オープン待ちとなります。

socket();	‘ソケットの生成
bind();	‘ソケットをポート番号に割り当てる
listen();	‘割り当てたポート番号に接続可能状態に移行する
accept();	‘受信接続待ち

クライアントからの接続が確立すると、「accept start」を表示して、受信待ち状態に移行します。

ベースクライアント：構築

- プログラムをコンパイルして動作を確認

```
$ cp -r TCP_CLIENT MY_CLIENT
$ cd MY_CLIENT
$ make realclean
$ ls
Makefile  tcp_client.c  tcp_client.h  tinet_tcp_client.cfg
route_cfg.c  tcp_client.cfg  tinet_app_config.h
$ make depend
$ make
```

- サーバーに接続してコンソールに入力したデータをサーバーに送り、サーバープログラムから送信したデータが表示されることを確認してください。

2006/10/07

TOPPERSプロジェクト認定

134



./OBJ/AKIH8_3069F/ディレクトリ上で、TCPクライアントプログラムディレクトリのTCP_CLIENTをMY_CLIENTにcpコマンドを用いてコピーします。MY_CLIENTディレクトリに移動して、サーバープログラムと同じ手順で、TCPクライアントプログラムを生成します。

生成したjsp.srecをTeraTermを介して、ボードにダウンロードし、実行して手順通りに接続とデータ送信ができることを確認してください。

ベースクライアント : ファイル構成

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

TCP_CLIENT上のファイルは、上記の通りです。

ベースクライアント

: TINETコンフィギュレーション(tinet_tcp_client.cfg)

- クライアントアプリケーションであるため、TCP受付口は作成せず、TCP通信端点を1個作成

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

```
TCP_CRE_CEP (TCP_CLIENT_CEPID, {
    0,
    tcp_client_cep_sbuf, TCP_CLIENT_CEP_SBUF_SIZE,
    tcp_client_cep_rbuf, TCP_CLIENT_CEP_RBUF_SIZE,
    NULL
});
```

TINETコンフィギュレーションファイル(tinet_tcp_client.cfg)について説明します。TCPクライアントプログラムでは、TCPサーバープログラムと異なり、受信待ちを行う必要がありません。そのため、**tinnet_tcp_client.cfg**でも、TCP受付口を生成をする必要がなく、TCP通信端点のみを作成します。生成の設定は、TCPサーバーとほぼ同等です。

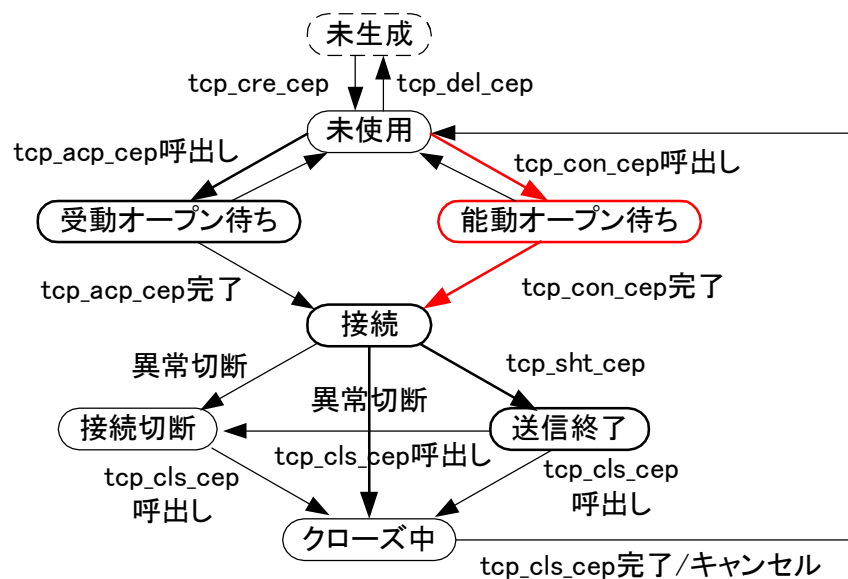
ベースクライアント : JSPコンフィギュレーションファイル

- タスクを1個生成

```
CRE_TSK(TCP_CLIENT_TASK, {  
    TA_HLNG|TA_ACT,  
    0,  
    tcp_client_task,  
    TCP_CLIENT_MAIN_PRIORITY,  
    TCP_CLIENT_STACK_SIZE,  
    NULL  
});
```

JSPコンフィギュレーションファイル (tcp_client.cfg) にクライアントプログラムを実行するタスク (tcp_client_task) の生成用、静的APIを記述します。

ベースクライアント：能動オープン



2006/10/07

TOPPERSプロジェクト認定

138



能動オープン待ち手順について説明を行います。以降は、`tcp_client.c`プログラムを対象に説明を行います。

ベースクライアント：能動オープン

- 能動オープンAPI

```
tcp_con_cep(
    TCP通信端点ID,
    自分のIPアドレス/ポートアドレス構造体,
    相手のIPアドレス/ポートアドレス構造体,
    タイムアウト)
```

- ベースクライアント

```
T_IPV4EP dst;
dst.ipaddr = DEST_ADDR;
dst.portno = DEST_PORT;
tcp_con_cep(TCP_CLIENT_CEPID,
    (T_IPV4EP*)NADR, &dst,
    TMO_FEVR)
```

IP及びポート番号は
プロトコルスタックで
決定

2006/10/07

TOPPERSプロジェクト認定

139



能動オープンはtcp_con_cepサービスコールにて行います。tcp_client.cの70行目から86行目までを以下に載せます。80行目のtcp_con_cepサービスコールにより、能動オープンを行います。サーバーへの接続が成立するとE_OKが返り、85行目のsyslog行に移行します。エラーが発生しE_OKが返らない場合は、再度、tcp_con_cepサービスコールを実行し能動オープンを行います。

```
while (TRUE) {
    /*
     * 端点への接続
     */
    do{
        syslog(LOG_NOTICE, "[TCP CLIENT] Push any key to connect");
        syscall(serial_rea_dat(TASK_PORTID, &c, 1));
        syslog(LOG_NOTICE, "[TCP CLIENT CEP] connected to %s:%d", addr, dst.portno);
        if (error = tcp_con_cep(TCP_CLIENT_CEPID, (T_IPV4EP*)NADR, &dst, TMO_FEVR) !=
E_OK) {
            syslog(LOG_NOTICE, "[TCP CLIENT CEP] error: %s", itron_strerror(error));
        }
    }while(error != E_OK);
    syslog(LOG_NOTICE, "[TCP CLIENT CEP] connection establish.");
```

ベースクライアント : コネクション切断

- コネクションを先に切断した方は2MSL待たなければならない
- TINETでは、TCP通信端点とコネクションが1対1に対応しているため、TCP通信端点が2MSL待つ
- Windowsはコネクション自体は2MSL待つが、ソケットは消滅する
- そのため、今回はWindows側からコネクションを切断するようにする

2006/10/07

TOPPERSプロジェクト認定

140



TCPの基本機能として、コネクションの接続を要求した側は、2MSL待たなければなりません。MSL (Max Segment Lifetime) は最大セグメント生存時間を表します。MSLは、パケットがネットワークに滞留できる最大時間を示します。すなわち、MSLを越えてパケットが到着しない場合、パケットが何らかの理由で喪失したものとみなすことができます。つまり、切断のために、FINを発行してから往復として2MSL待機すれば、もう、受信データはないと判定します。

通常の2MSLは120秒位とされますが、実装により、もっと短いケースが多いようです。TINETの場合、`TCP_TVAL_KEEP_COUNT(8) × TCP_TVAL_KEEP_INTERVAL (7.5秒)` で60秒に設定しているようです。

TCPクライアントでは、Windows上のサーバーが切断を行うため、サーバーが側で2MSL待ちを行う。WindowsのsocWinインターフェースでは、`accept();`により接続が確立した時点で、通信用のソケットが生成されます。コネクションを切断した場合、このソケットは2MSL待つて消滅します。

ベースクライアント : コネクション切断

- コネクション切断前に終了パケットを送る

```
#define END_STAR "_end_"
if (c == '\003' || c == 'Q'){
    if ((slen = tcp_snd_dat(TCP_CLIENT_CEPID,
        END_STAR, 5, TMO_FEVR)) < 0) {
        syslog(LOG_NOTICE, "[TCP CLIENT SND] .....");
        goto err_fin;
    }
}
```

- さらにサーバーから終了パケットが送られてくるのを待つ
- 終了パケットを受け取ったらポートを閉じる

```
tcp_cls_cep(TCP_CLIENT_CEPID, TMO_FEVR)
```

2006/10/07

TOPPERSプロジェクト認定

141



切断の説明を行うために、`tcp_client.c`の87行目から106行目を以下に載せます。コンソールからCtrl-Cまたは‘Q’を入力すると、97行目の判定で、`tcp_snd_dat`サービスコールを用いて、サーバーに`END_STAR(=“_end_”)`を送ります。サーバーから“_end_”の受信を待って、`tcp_cls_cep`サービスコールを用いてTCP通信端点をクローズします。

```
/*
 * データの送受信
 */
while (TRUE) {
    /*
     * エンターが入力されるまで文字列を読み込む
     */
    slen = 0;
    do{
        syscall(serial_rea_dat(TASK_PORTID, &c, 1));
        if (c == '\003' || c == 'Q'){
            /* 終了通知を送る */
            if ((slen = tcp_snd_dat(TCP_CLIENT_CEPID, END_STAR, 5, TMO_FEVR)) < 0) {
                syslog(LOG_NOTICE, "[TCP CLIENT SND] can not send end data",
                    itron_strerror(slen));
            }
            goto err_fin;
        }
        sbuffer[slen++] = c;
    }while(c != '\r');
}
```

TCPクライアントプログラミング

1. ベースクライアント

[2. DIP状態通知クライアント](#)

DIP状態通知クライアント : DIP_MONITOR

- ベースクライアントを基に、サーバーに接続後周期的にディップスイッチの状態を送るプログラムを作成しましょう
- サーバー側のプログラムはベースクライアントと同じものを使用します。
- ディップスイッチの状態はメインタスク(tcp_client_task)以外にもう一つDIP状態通知タスク(dip_monitor_task)を作成して、このタスクで周期的にDIPスイッチの状態を送ります。
- メインタスクはサーバーと接続した後、DIP状態通知タスクを起動し、シリアルコンソールからCtrl-CもしくはQが入力されたら、DIP状態通知タスクを終了させ、サーバーとの接続を切断します。
- ハードウェアの初期化を忘れないこと。

2006/10/07

TOPPERSプロジェクト認定

143



ベースクライアントプログラムを改造して、DIP状態通知クライアントプログラムを作成します。サーバー側のプログラムはベースクライアントと同等のプログラムを用いて変更はありません。

DIP状態通知クライアントは2つのタスクで構成します。メインタスク(tcp_client_task)は、サーバーと接続した後、もう一つのタスクであるDIP状態通知タスク(dip_monitor_task)を起動して、コンソールからのCtrl-Cまたは‘Q’の入力を待ち、入力後、DIP状態通知タスクを終了させ、その後、サーバーに切断を要求します。

DIP状態通知タスクは起動すると、周期的にDIP-SWの状態を読み取り、OFFまたはON状態を表す‘0’または‘1’の4文字列に変換してサーバーに送信します。周期は2秒といいます。

DIP状態通知クライアント : JSPコンフィギュレーションファイル

- 新たにDIP状態通知タスクを休止状態で生成

```
CRE_TSK(DIP_MONITOR_TASK, {  
    TA_HLNG,  
    0,  
    dip_monitor_task,  
    DIP_MONITOR_PRIORITY,  
    DIP_MONITOR_STACK_SIZE,  
    NULL  
});
```

JSPのコンフィギュレーションファイルにDIP状態通知タスク(dip_monitor_task)の生成APIを追加します。

DIP状態通知クライアント : DIP状態通知タスク

```

void dip_monitor_task(VP_INT exinf){
    int slen;
    syslog(LOG_NOTICE, "[DIP MONITOR] started.");
    while(TRUE) {
        if (get_switch(SW1) == ON) {
            sbuffer[0] = '1';
        } else {
            sbuffer[0] = '0';
        }
        .....
        if ((slen = tcp_snd_dat(TCP_CLIENT_CEPID,
                               &sbuffer, 4, TMO_FEVR)) < 0) {
            syslog(LOG_NOTICE, "[TCP CLIENT SND] .....");
        }
        dly_tsk(DIP_MONITOR_INTERVAL);
    }
}

```

2006/10/07

TOPPERSプロジェクト認定

145



tcp_client.cにdip_monitor_task();関数を追加します。

DIP状態通知クライアント：メインタスク

- サーバーとの接続後、DIP状態通知タスクを実行状態にする

```
syscall(act_tsk(DIP_MONITOR_TASK));
```

- シリアルコンソールからCtrl-CもしくはQを受け取るとDIP状態通知タスクを終了させる

```
do{
    syscall(serial_rea_dat(TASK_PORTID, &c, 1));
}while(c != '\003' && c != 'Q');
syslog(LOG_NOTICE, "[TCP CLIENT] terminate dip monitor task");
syscall(ter_tsk(DIP_MONITOR_TASK));
```

tcp_client.c中のメインタスク(tcp_client_task)にサーバーとの接続後、DIP状態通知タスクを実行状態にするact_tskサービスコールを追加します。また、シリアルコンソールからCtrl-Cまたは‘Q’を入力時、DIP状態通知タスクを停止するter_tskサービスコールを追加します。

不要なデータ入力と送信手順は削除します。

まとめ

1. TCP/IPの基礎
2. ITRON TCP/IP仕様
3. TINET (TCP/IPスタック)
4. TCPサーバープログラム
5. TCPクライアントプログラム
6. まとめ



カップラーメンタイマーのTOPPERS/JSP版を作成してみましょう。

この開発環境では、デバッガが使用できませんので、タスクモニターやシステムログのダンプ機能を使ってデバックしましょう。

1日目のまとめ

以下の事項について学習

- AKI-H8/3069FとTOPPERS/JSPカーネルの構築方法
- 周辺デバイスのプログラミング方法
- ITRON仕様
- ITRON APIを用いたプログラミング
 - タスク関連のAPI
 - 同期付属機能
- カップラーメンタイマー
 - 設計
 - コーディング

2日目のまとめ

- 以下の事項について学習
 - TCP/IPの概要
 - ITRON TCP/IP仕様
 - TINET
 - TCPサーバープログラミング
 - TCPクライアントプログラミング

教材の作成

参考資料

- 「TCP/IPによるネットワーク構築 Vol.I 原理・プロトコル・アーキテクチャー」
Douglas Comer著/村井純・楠木博之 訳
- 「平成16年度 マイコン応用研修 TINETによる TCP/IP プログラミング」
苫小牧工業高等専門 情報工学科 阿部 司