解 説

リアルタイム制御のためのTCP/IPプロトコルスタック —ソケット通信からリアルタイムOS用プロトコルまで—

TCP/IP Protocol Stack for Real-Time Control

—Starting from Socket Communications to Protocols for Real-Time OS—

内 村 裕* *芝浦工業大学

Yutaka Uchimura* *Shibaura Institute of Technology

1. は じ め に

コンピュータネットワークの普及に伴い、ロボットをはじめとする各種制御系の実装においても、ネットワークを通信経路とすることが広く行われるようになってきている。このような、ネットワークを介した通信においては、信号を伝えるハードウェアはもちろんのこと、通信相手との間の通信手順に関する決まりを定めたプロトコルが必要不可欠である。プロトコルは、英語や日本語といった言語のようなものであるので、プロトコルの異なる通信装置間には言葉が通じない（通信が行えない）。そこで、インターネットのような異機種間の相互接続においては、共通言語（プロトコル）を定めており、様々なアプリケーションは、この共通言語を前提として動作している。

このような共通言語の一つが米国の DARPA（国防高等研究計画局）のプロジェクトから生まれた TCP/IP プロトコルである。TCP/IP は仕様や開発過程が公開されていたため、多くの研究者やユーザに支持され、インターネットにおける事実上の標準プロトコルとなっている。TCP/IP は、狭義には送信側のホストコンピュータから、受信側のホストコンピュータまでの転送を実施する IP（Internet Protocol）[1] と、IP を利用して信頼性のある通信を実現するためのプロトコルである TCP（Transmission Control Protocol）[2] のことを指すが、UDP、ICMP、ARP などのプロトコルを含めた TCP/IP スタック全体を指す場合も単に TCP/IP と呼ぶことがある。

本解説では、初めに TCP/IP によるプログラミングの基本としてソケット通信について説明した後、組込用 OS 上の実装例やロボット制御に利用可能なリアルタイム Linux 向けのネットワークプロトコルの事例について解説する。また、TCP/IP 以外のプロトコルとして、リアルタイム Linux 上で動作し、時間制約の厳しい制御系のアプリケーション

に適用可能な RT-Messenger についても紹介する。

2. ソ ケ ッ ト 通 信

「ネットワークを使ったアプリケーションを作ってみたい」、「ロボットにネットワークを利用した機能を付加したい」などと日ごろから思っているが、何から手をつければよいかわからないという読者には、まずは本章で紹介するソケット通信から始めることを勧める。

ソケットとは、ネットワーク通信をプログラミングするためのインターフェース（API: Application Programming Interface）[†]である。カリフォルニア大学バークレー校において開発され、1983年にソフトウェアパッケージ（4.2 BSD）に含まれる形で公開された。ソケット API は、ネットワークを介して通信を行うためのプログラミング手順として現在も広く使われている^{††}。

ソケット通信においては、ネットワークもプロセス間通信の形態の一つにすぎず、プロセスと通信路の接続点であるソケットを介して、ファイルやパイプを読み書きすると同様に通信が行える。ソケットはプロセスを通信路に繋ぐためのコンセントのようなものであり、プロセス側からは、通信手段は抽象化されている。つまり、実際にイーサネットカードを使って通信するか、モデム経由で電話線を使って通信するか、など実際の通信手段については関知しない。また、ソケットは個々のコンピュータに割り当てられた IP アドレスに加えてポートと呼ばれる識別番号を割り当てる。1台のコンピュータ上で動作中の複数のプロセスにそれぞれ別のポートが割り当てられる。IP アドレスを住所とすると、ポートは部屋番号に相当するものである。

ソケットを生成する際には、プロトコルの種類（domain）と接続形態（type）を指定する。domain には、インターネットプロトコル IPV4（PF_INET）、ローカル通信（PF_UNIX）、Apple Talk プロトコル（PF_APPLETALK）などが指定できる。

原稿受付 2007 年 12 月 6 日

キーワード: Network, TCP/IP, Protocol, Real-Time OS

*〒135-8548 江東区豊洲 3-7-5

*Koto-ku, Tokyo

[†]プログラミングにおいて関数を呼び出す仕組み。^{††}ITRON TCP/IP API や XTI (X/Open Transport Interface) [3] のようにソケット以外の API も多数存在する。

接続形態 (type) には, `SOCK_STREAM`, `SOCK_DGRAM`, `SOCK_RAW` などが指定できる. `SOCK_STREAM` は, 信頼性の高いコネクション型の双方向接続性の通信であり, `SOCK_DGRAM` ソケットはコネクションレス型の固定最大長の通信である.

コネクション型とは, 通信相手との回線を常に維持して行う方式で, 通信状況に応じた速度の通信の調整や, 通信データ欠損時の再送処理などを行うことで信頼性を向上させた通信方法である. 一方, コネクションレス型は, コネクション型のような通信品質を確保するための処理は行わない.

一見, コネクション型のほうが優位にあるようだが, リアルタイム性と信頼性はトレードオフの関係にあり, 制御系のアプリケーションにおいては, コネクションレス型の通信が使われることも多い. 例えば再送による通信遅れが, 通信データの欠損以上に問題になる場合には, コネクションレス型の通信が適している. `INET` ドメイン (`PF_INET`)

リスト 1 サーバ側プログラム

```

1  /* server.c:*/
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <sys/types.h>
6  #include <sys/socket.h>
7  #include <netinet/in.h>
8
9  #define PORT 50001
10 #define nerr_msg(msg) do { perror(msg); exit(
    EXIT_FAILURE); } while (0)
11
12 int main(void)
13 {
14     int sd, fd;
15     struct sockaddr_in saddr, caddr;
16     int len, ret;
17     char buf[32];
18
19     if ((sd = socket(PF_INET, SOCK_STREAM,
20                     0)) < 0)
21         nerr_msg("socket");
22     bzero((char *)&saddr, sizeof(saddr));
23     saddr.sin_family = AF_INET;
24     saddr.sin_addr.s_addr = INADDR_ANY;
25     saddr.sin_port = htons(PORT);
26     if (bind(sd, (struct sockaddr *)&saddr, sizeof(
27                 saddr)) < 0)
28         nerr_msg("bind");
29     if (listen(sd, 1) < 0)
30         nerr_msg("listen");
31     len = sizeof(caddr);
32     if ((fd = accept(sd, (struct sockaddr *)&caddr,
33                     &len)) < 0)
34         nerr_msg("accept");
35     close(sd);
36     ret = read(fd, buf, 128);
37     printf("%s", buf);
38     close(fd);
39     exit(0);
40 }
```

の場合, `TCP` がコネクション型 (`SOCK_STREAM`) に, `UDP` がコネクションレス型 (`SOCK_DGRAM`) に相当する. ただし, 一般的に下層の `IP` とセットで, `TCP/IP`, `UDP/IP` と呼ばれることが多い.

リスト 1, 2 にコネクション型のソケット通信プログラムの `C` 言語ソースの例を示す. リスト 1 がサーバ側プログラムで, リスト 2 がクライアント側プログラムである. 本ソースの実行には, `Linux` など `Unix` 系の `OS` をインストールした `PC`, または `Cygwin` [4] をインストールした `Windows PC` が必要である. 2 台の `PC` で実行することが望ましいが, 1 台の `PC` 上にターミナルを二つ開いても実行可能である.

それぞれのプログラムを, サーバ側とクライアント側でリスト 3 のようにコンパイル, リンクを行ったのち, サーバ側プログラムを先に実行してから, クライアント側プロ

リスト 2 クライアント側プログラム

```

1  /* client.c */
2  #include <stdio.h>
3  #include <string.h>
4  #include <sys/types.h>
5  #include <sys/socket.h>
6  #include <netinet/in.h>
7  #include <netdb.h>
8
9  #define nerr_msg(msg) do { perror(msg); exit(
    EXIT_FAILURE); } while (0)
10 #define PORT 50001
11 #define SERVER_IP "192.168.0.1"
12 char buf[32] = {"Hello!\n"};
13
14 int main(void)
15 {
16     struct sockaddr_in addr;
17     int sd;
18
19     if ((sd = socket(PF_INET, SOCK_STREAM,
20                     0)) < 0)
21         nerr_msg("socket_fail");
22     bzero((char *)&addr, sizeof(addr));
23     addr.sin_addr.s_addr = inet_addr(SERVER_IP);
24     addr.sin_family = AF_INET;
25     addr.sin_port = htons(PORT);
26
27     if (connect(sd, (struct sockaddr *)&addr,
28                 sizeof(addr)) < 0)
29         nerr_msg("connect");
30     write(sd, buf, sizeof(buf));
31     close(sd);
32     exit(0);
33 }
```

リスト 3 コンパイル, 実行例

```

1  サーバ側
2  > gcc -o server server.c
3  > ./server
4
5  クライアント側
6  > gcc -o client client.c
7  > ./client
```

図1 ソケット通信の手順

グラムを実行すると、サーバ側の PC に Hello ! と表示されるはずである[†]。なお、コンパイルの前にクライアント側プログラムの "192.168.0.1" を、サーバ側 PC の IP アドレスに書き換える必要がある (1 台で実行する場合は、その PC の IP アドレス)。

TCP/IP によるソケット通信の手順は、図 1 に示すように、サーバ側では、(1) ソケットの生成 (socket), (2) ポートの割り当て (bind), (3) 接続待ち (listen), (4) クライアントとの接続 (accept), (5) データの送受信 (read, write), (6) 通信切断 (close) という流れで処理を行う。一方、クライアント側は、(1) ソケットの生成 (socket), (2) サーバへの接続 (connect), (3) データの送受信 (read, write), (4) 通信切断 (close) の処理となる。サーバがクライアントの接続要求 (connect) を受け入れて (accept) コネクションが確立すると、ファイルを読み書きすると同様に read/write が行える。

送受信のデータは、プロトコルの処理によって複数のパケットに分割されたのち、各パケットに宛先の IP アドレス、ポート番号が付加される。さらに IP アドレスを MAC アドレスに変換した値が付加されて、ネットワークデバイスから物理的な通信路に送り出される。

3. 組込用 OS の TCP/IP プロトコルスタック

前述したように、BSD ベースの TCP/IP プロトコルスタックは、仕様とソースがオープンであるため、様々な CPU をサポートする各種 OS 向けに移植され、OS のシステムコールもしくはミドルウェアの形で提供されている。その

中でも、いわゆる組込用途向けの OS においては、リアルタイム性を要求するアプリケーションへの適用が要求されるため、プロトコルの実装においてもメモリ管理に関わる制約を意識した実装が必要となる。そこで、本章では、代表的な組込用 OS である ITRON 系の実装を紹介すると同時に、リアルタイム OS におけるメモリの動的確保について解説する。

3.1 ITRON 系 OS における実装例

組込用途としては、トロン協議会が μ ITRON 上の実装 (ないしは組込用 MPU のミドルウェアの実装) の仕様として、ITRON TCP/IP API 仕様を定めている [5]。本仕様に即した TCP/IP の例としては、ルネサステクノロジーの Tiny シリーズマイコン用のプロトコルスタックである M3S-T2-Tiny や、同 M32R 用の M3S-TCP32R などがある。

また、ITRON 仕様のオープンソースソフトウェアを提供している TOPPERS [6] プロジェクトでは、FreeBSD バージョン 3.4 をベースに TINET [7][8] を開発している。原稿執筆時点では H8 系のマイコン (H8/3048^{††}および 3069) に適用可能であり、NE2000 互換のイーサネットデバイスがサポートされている。

組込用 MPU をターゲットとする場合は、メモリ使用量の厳しい制約のもとでの実装が求められる。実際に、ROM 8[kbyte], RAM 0.5[kbyte] という極めて少ないメモリ使用量で動作する TCP/IP スタックの開発事例も報告されている [9]。

3.2 リアルタイム OS におけるメモリの動的確保の問題

一般に、メモリ利用の効率化とメモリ確保にかかる時間はトレードオフの関係にある。例えば、メモリのフラグメンテーション (断片化) を避けるため、パディシステムやスラブアロケータなどのリスト構造で空きメモリを管理するが、動的メモリ確保と解放を長い時間にわたって行うと空きメモリのエリアの探索に時間がかかるおそれがある [10]。その結果、メモリの動的確保 (または解放) の実行時間の上限値の予測が困難となるため、時間制約の厳しいプロセスの実行中にメモリの動的確保を行うことは好ましくない。

そこで、リアルタイムプロセス実行前の段階であらかじめ静的にメモリを確保するか、もしくはリアルタイム性が高くスワップアウト^{†††}の起こらないメモリ管理を実装するなどの対策が必要である。前述の TINET や後述のリアルタイム Linux 上のプロトコルスタックも、あらかじめ用意した固定長のメモリプールからネットワークバッファを割り当てることで、プロトコルスタック内における各層間のデータの受け渡しを行っている。

4. リアルタイム Linux における実装例

Linux カーネルのリアルタイム性を向上しようとする試

[†]本プログラムは 1 回の送受信で終了するが、read/write をループさせれば繰り返しデータをやり取りできる。

^{††}イーサネットデバイスはサポートしない。

^{†††}時間制約の厳しいプロセスにおいては、デッドラインを超えた時間の経過すること (デッドラインミス) が許されない。タスク実行中に動的なメモリ確保を行うと、状況によっては外部記憶に仮想メモリが確保される (スワップアウト) の可能性もあり、高いリアルタイム性を要求するアプリケーションには不向きである。

みはいくつか行われており、カーネギーメロン大学で開発された資源予約をベースとした TimeSys Linux [11][12] や、時間駆動型の拡張である ART-Linux [13] などが開発されている。また、Linux カーネルの下にリアルタイム OS を潜り込ませた構造を有する RT-Linux [14] や RTAI [15], Xenomai [16] も、ロボットなどモーションコントロール系のアプリケーションに活用されている。

本章では、これらのリアルタイム Linux のうち RT-Linux および RTAI 上に実装されたプロトコルスタックを紹介する。実装の形態によって、TCP/IP をサポートするものもあれば、UDP/IP のみサポートのもの、またリアルタイム性を最重視し RAW パケットによる通信を行う実装例もあり、用途に応じて使い分けることとなる。

4.1 Rtsock

Rtsock は、RT-Linux 上から Linux のソケット API を利用するためのインターフェースを提供するプロトコルスタックであり、現状では UDP ソケットをサポートしている。ソケットの生成、バインドなどの処理は、ユーザモード側のプログラムで行い、RT-FIFO 経由でソケットおよびソケット・カーネル・バッファ (`sk_buff` 構造体) のポインタを受け渡すことで、送受信データ (ペイロード) への書き込みを可能としている。また、`sk_buff` の解放 (`kfree_skb`) を含む動的メモリ処理は、リアルタイムタスク中では行わずに、ソフトウェア割り込み経由で呼び出される通常の Linux のカーネルコンテキスト中で行っている。

デバイスドライバを含むデータリンク層以下の処理は Linux カーネルに任せているので、Linux をサポートするすべてのネットワークデバイスが利用可能であることがメリットである。

しかしながら、ユーザモード側プログラムと、カーネルモード側プログラムの双方の動作が必要なため手順が煩雑である。また、Linux 側のプロセスの実行に依存するためリアルタイム性に欠けるなどのデメリットも併せ持っている。

4.2 RT-lwIP

RT-lwIP [17] は、スウェーデンのコンピュータサイエンス研究所 (SICS) で開発された lwIP (Light Weight IP) を RT-Linux にポータリングしたものである。現状において、IPv4, IPv6, ICMP, UDP, TCP をサポートしている。本プロトコルスタックにおいても、通信に必要な動的メモリの管理を独自に行っており、動的メモリのリアルタイム性問題を回避している。また、BSD ソケットによく似た同期型の API のほかに、コールバックが可能な非同期型

の API (RAW-API) も用意されている。

ただし、独自のメモリ管理を実装するため、独自のデバイスドライバを供給する必要がある。このため、Linux カーネルに含まれるデバイスドライバが使用できず、サポートするデバイスが 3COM 3c59x, RealTek 8139, Intel Pro100 に限られる。

4.3 RTnet

RTnet [18] は、PC 用リアルタイム Linux である RTAI および Xenomai 上で動作するプロトコルスタックである。RT-lwIP 同様、Linux カーネルの TCP/IP スタックとは別に動作し、バッファ用のメモリ管理は独自に行っている。現状、UDP, IP, ICMP, ARP をサポートしており、ソケット形式の API を提供している。

特筆すべきは、イーサネット上でのリアルタイム性を確保するため[†]、TDMA (時分割多元接続: Time Division Multiple Access) を実装している点で、ネットワークに参加する PC 中の 1 台をマスターノードとしてメディアアクセス制御を行っている。Ethernet-over-1394 などサポートしており、サポートするネットワークデバイスは RT-lwIP よりは多いが、一部のデバイスに限定される。

4.4 RT-Messenger

RT-Messenger [19] は、慶應義塾大学矢向研究室で開発されたリアルタイム Linux 用のプロトコルである。TCP/IP のような重層 (マルチレイヤ) のプロトコルスタックと異なり、データリンク層以下で動作するプロトコルである。カーネル内部での処理のオーバーヘッドが小さく、スケジューラの影響による揺らぎも小さい。このため、リアルタイム性が要求されるアプリケーションに適しており、図 2 に示すようなネットワークを介したバイラテラル遠隔制御ロボットへの適用例では、1 ミリ秒周期のサンプリングタイムでの通信が実装されている [20]。

RT-Messenger においても、送受信データ用のバッファはリアルタイムタスク開始前に確保している。また、一度確保したバッファ (`sk_buff` 構造体) は、リアルタイムタスク終了時まで解放せず再利用する仕組みとしている。このため、パケットサイズはリアルタイムタスク実行前に確定する必要があり、実行中は固定のパケットサイズとなる。制御系のアプリケーションの場合、周期実行中の送受信データの個数は一定であることが多く妥当な実装である。

RT-Messenger では、図 3 に示すようなイーサネットパケットを使用するが、パケットを生成する際、フレームタイプに RT-Messenger 固有の ID を付与する^{††}。パケットを生成し、MAC アドレスを付与したパケットは、ネットワークデバイス中の送信関数 (`hard_start_xmit()`) に直接渡されて即時送信される。また、あらかじめ受信時ルーチン登録関数 (`dev_add_pack()`) によって、RT-Messenger のパケットのタイプ (固有 ID) と、パケット受信時に起動

[†]CSMA/CD によってアクセスするため、本来はリアルタイム性が保証できない通信線路である。

^{††}プロトコルのタイプは RFC1340 で規定されており、16 進数表現で IP では 0x0800, ARP は 0x0806, RARP は 0x8035 などとなっているが、RT-Messenger は RFC1340 で空き番号となっている 0x0807 を使用している。

図2 バイラテラル遠隔操作ロボット

8byte	6byte	6byte	2byte	46-1500byte	4byte
ブリアンブル	宛先MACアドレス	送信元MACアドレス	フレームタイプ	データ	FCS

図3 Ethernet (DIX 規格) フレーム

するスレッドを指定しておく。これによって、パケット受信時（ボトムハーフ起動時）に、パケットが受信処理タスク（リアルタイム Linux のスレッド）に直接渡されるのでプロトコル処理のオーバーヘッドが大幅に削減される。

また、RT-Messenger はデバイスドライバの変更は行っていないので、Linux がサポートするすべてのネットワークデバイスが利用可能である上に、既存の TCP/IP 等のプロトコルと共存できる。

現在、Linux カーネルバージョン 2.4.4 に対応した RT-Linux 版が http://www.comp.sd.keio.ac.jp/~yakoh/maruyama/rt_messenger/ で公開されている。また、筆者は Linux カーネルバージョン 2.6.19 上の RTAI に移植し動作を確認しているので、興味のある読者は連絡されたい。

このように RT-Messenger は、実装がコンパクトかつ扱いが容易な上、高いリアルタイム性を有することから、ロボットのようなモーションコントロール用途としては非常に有効なプロトコルであるといえる。

5. おわりに

コントローラ（制御装置）とセンサ、アクチュエータ間をネットワークで結んだネットワークベースの制御系は、ロボット制御はもとより自動車や建物の設備制御 [21] [22] などにも活用されている。ネットワーク技術の進歩はとどまるところを知らず、今後もハードウェア、ソフトウェアの両面における発展から目が離せない。一方で、4.2 BSD の登場から 20 年以上経っても、ソケットや TCP/IP の基本的な概念は脈々と引き継がれており、基本的なネットワークの構成、考え方は今後とも急激に変化することは少ない

と思われる。これらネットワークの基本を修得することが、読者が研究者、技術者として将来の技術進歩に追従するための礎となることを願いたい。

参考文献

- [1] RFC 791: "Internet Protocol DARPA Internet Program Protocol Specification," <http://www.ietf.org/rfc/rfc0791.txt>
- [2] RFC 793: "Transmission Control Protocol DARPA Internet Program Protocol Specification," <http://www.ietf.org/rfc/rfc0793.txt>
- [3] X/Open Transport Interface: <http://www.opengroup.org/openbrand/register/gsm4.htm>
- [4] Cygwin: <http://www.cygwin.com/>
- [5] ITRON 専門委員会 Embedded TCP/IP 技術委員会: "ITRON TCP/IP API 仕様", <http://www.sakamura-lab.org/TRON/ITRON/SPEC/FILE/tcpip-100j.pdf>
- [6] Toppers プロジェクト: <http://www.toppers.jp/index.html>
- [7] 阿部司: "組み込み機器向け ITRON TOPPERS と組み合わせて使える組み込み TCP/IP プロトコル・スタック TINET の詳解", Interface, vol.30, no.4, pp.96-107, 2004.
- [8] TINET: <http://www.toppers.jp/tinet.html>
- [9] 浅井 他: "TCP/IP プロトコル・スタックの省メモリ開発事例", <http://www.kumikomi.net/article/explanation/2004/22tcpip2/01.html>
- [10] D.P. Bovet 著, 高橋浩和 監訳: 詳解 Linux カーネル第 2 版, オライリー・ジャパン, 2003.
- [11] D. Lazenby: TimeSys Linux/RT (Professional Edition), Linux Journal, vol.2000, Issue 77, no.21, 2000, <http://www.linuxjournal.com/article/4142>
- [12] S. Ghosh and R.R. Rajkumar: "Resource management of the OS network subsystem," Proc. of IEEE Int. Symp. on Object-Oriented Real-Time Distributed Computing, pp.271-279, 2002.
- [13] 石渡陽一, 松井俊浩: "高度な実時間勝利能力を持つ Linux の開発", 第 16 回日本ロボット学会学術講演会予稿集, pp.355-356, 1998.
- [14] RT-Linux: <http://www.rtlinuxfree.com/>
- [15] RTAI: <https://www.rtai.org/>
- [16] Xenomai: <http://www.xenomai.org/>
- [17] RT-LWIP: <http://rtl-lwip.sourceforge.net/>
- [18] RTnet: <http://sourceforge.net/projects/rtnet>
- [19] 佐藤秀雄, 矢向高弘: "時間制約の厳しいマルチメディアのためのリアルタイム通信機構", 情報処理学会論文誌, vol.42, no.2, pp.189-196, 2001.
- [20] Y. Uchimura and T. Yakoh: "Bilateral Robot System on the Real-Time Network Structure," IEEE Transaction on Industrial Electronics, vol.51, no.5, pp.940-946, 2004.
- [21] BACnet: Official Website of ASHRAE SSPC 135, <http://www.bacnet.org/>
- [22] 吉川寛: "ビルシステムにおける BACnet 対応", 三菱電機技報, vol.75, no.7, p.496, 2001.

学会会員。

内村 裕 (Yutaka Uchimura)

1991 年慶應義塾大学院修士課程修了, 同年鹿島建設入社。2005 年慶應義塾大学院博士課程修了。2007 年より芝浦工業大学機械工学科准教授, ネットワーク経由のロボット制御, ネットワークセンシングの研究に従事。博士 (工学), 電気学会, 計測自動制御学会, 日本機械 (日本ロボット学会正会員)