特集

UNIX TCP/IP ネットワーク入門

UNIXワークステーションにとっては，ネットワークはもはやなくてはならない環境であるといってもよい状況ができつつあるが，ネットワークでの通信の実際についてはまだまだ知られていないことが多いようだ．そこで，今回はネットワークにプロトコルの直接操作を通じてアクセスしてみることで具体的なイメージを摑み，UNIXから見える仮想的な通信経路と物理的な接続部分であるケーブルやインターフェイスの間に横たわるギャップを埋めてみようと思う．

ソケットの使い方とアプリケーションプロトコル

TCP/IPネットワークの現在

まずはじめに現在のInternet環境について整理してみよう．もちろん，TCP/IPネットワークとInternet環境がまったく同一なわけではないが，重なる部分も多いので参考になるだろう．

1. Internet環境の現状

Internetとは

現在，日本でも学術研究用のネットワークなどを中心として，Internet環境が拡大し続けている．Internetとは，機能的な意味としては“ネットワークのネットワーク”として，比較的小規模な組織内ネットワーク（LAN）同士を接続した大規模ネットワークのことをさす．もちろん狭い意味というのもあって，米国ARPAネットから発展した特定のInternetのことを固有名詞的にさすこともある．ここでは，Internetという場合には“ネットワークのネットワーク”を指すものとして使うものとする．

さて，日本でのInternet環境はまだまだ実験段階であるともいえ，利用できるのはごく限られたユーザーだけである．しかし，米国などでは一般的な通信手段として利用されている．これは，米国の国土の広大さが普及の原動力になっているのかもしれないが，ほとんどリアルタイムに近い感覚で大量のデータをやりとりできるInternetの能力は，コンピュータを利用して仕事をしている人にとっては“1度使ったらもう2度と手放せない”魅力を持ったものである．ここでは，このInternetの能力を，そこで提供されるネットワークサービスに注目して確認してみることにしよう．

日本と米国は本当に継っているか

さて，日本にもInternetに接続できる環境があるんだといわれても納得できない方もあるかもしれないので，ここで本当に米国までデータを送ることができるのだということを確認しておこう（図1）．

これは，弊社社内のコンピュータから米国マサチューセッツ工科大学（MIT）のメディアラボにあるコンピュータに対してpingコマンドとtracerouteコマンドを実行した例である．pingは，相手のマシンに正しくパケットが到着するかどうかを確認するために使われる．一方，tracerouteはMITのマシンまで往復したパケットが，途中どんなマシンに中継されているかと，パケットが中継地点まで往復するためにかかった時間をすべて報告する．

tracerouteコマンドは，本来はネットワークを管理する人が必要に応じて使うものであり，ネットワークに対してそれなりの負荷をかける．用もないのにこんなことをするのは他人迷惑であるということは十分留意しておいていただきたい．

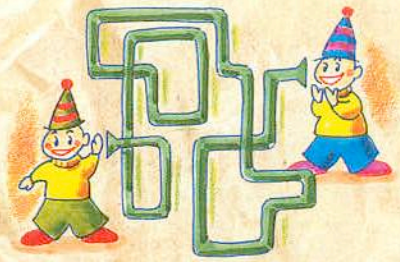
Internetの物理的実体

図1に出てくるのはデータの中継をしたコンピュータの名前だが，“中継”というのはInternetでは非常に重要なサービスである．Internetはネットワークのネットワークであり，ネットワーク同士を相互に接続していく形で構成されている．そのため，ネットワークからネットワークへデータを送る場合には，ネットワーク同士の接続点を通らなくてはならない．この接続点となっているのが，図1で出てきたデータの中継をするマシンである．

Internetは各ネットワークごとの接続点を構成しているマシン（IPルータ，もしくはgatewayと呼ぶ）と，接続点同士を結んでいる通信路によって基本的骨格ができあがっていると考えてよいだろう．

図1　MITメディアラボまでのパケットの往復時間と経路

```
[ascwide.ascii.co.jp]% ping media-lab.media.mit.edu
PING media-lab.media.mit.edu (18.85.0.2): 56 data bytes
64 bytes from 18.85.0.2: icmp_seq=0. time=406. ms
64 bytes from 18.85.0.2: icmp_seq=1. time=495. ms
64 bytes from 18.85.0.2: icmp_seq=2. time=418. ms
64 bytes from 18.85.0.2: icmp_seq=3. time=451. ms
64 bytes from 18.85.0.2: icmp_seq=4. time=418. ms
64 bytes from 18.85.0.2: icmp_seq=5. time=396. ms
64 bytes from 18.85.0.2: icmp_seq=6. time=407. ms
64 bytes from 18.85.0.2: icmp_seq=7. time=418. ms
^C
----media-lab.media.mit.edu PING Statistics----
8 packets transmitted, 8 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 396/426/495
[ascwide.ascii.co.jp]%
[ascwide.ascii.co.jp]% traceroute media-lab.media.mit.edu
traceroute to media-lab.media.mit.edu (18.85.0.2), 30 hops max, 40 byte packets
 1  jp-gate.wide.ad.jp (133.4.1.1)  11 ms  11 ms  11 ms
 2  jp-entry.wide.ad.jp (133.4.1.2)  11 ms  22 ms  11 ms
 3  menehune.Hawaii.Net (132.160.251.1)  132 ms  132 ms  132 ms
 4  imp.Hawaii.Net (132.160.1.20)  132 ms  132 ms  132 ms
 5  arc3.nsn.nasa.gov (132.160.249.2)  187 ms  198 ms  198 ms
 6  ARC2.NSN.NASA.GOV (192.52.195.11)  209 ms  187 ms  187 ms
 7  Palo_Alto.CA.NSS.NSF.NET (192.52.195.254)  198 ms  198 ms  198 ms
 8  Palo_Alto.CA.NSS.NSF.NET (129.140.13.129)  198 ms  209 ms  209 ms
 9  San_Diego.CA.NSS.NSF.NET (129.140.70.13)  220 ms  231 ms  220 ms
10  Houston.TX.NSS.NSF.NET (129.140.75.6)  308 ms  275 ms  275 ms
11  College_Park.MD.NSS.NSF.NET (129.140.73.11)  319 ms  363 ms  330 ms
12  Princeton.NJ.NSS.NSF.NET (129.140.72.9)  341 ms  363 ms  363 ms
13  Princeton.NJ.NSS.NSF.NET (129.140.8.130)  396 ms  385 ms  407 ms
14  w91-cisco-external-ether.mit.edu (192.54.222.1)  407 ms  462 ms  418 ms
15  E19-CISCO-FDDI.MIT.EDU (18.168.0.1)  385 ms  473 ms  440 ms
16  media-lab.media.mit.edu (18.85.0.2)  451 ms  462 ms  363 ms
[ascwide.ascii.co.jp]%
```



UNIX
TCP/IP
ネットワーク入門

2. 主要なネットワークサービス

さて，図1で日本からハワイを経由し，米国本土を横断してパケットが届く様子は実感できたので，次にInternetでどのようなサービスが提供されているのかをみていこう．もちろん，ここでいうサービスとは“何々してあげるとお支払いはいくらになります”といういわゆるサービス産業的な意味ではなく，ユーザーから見たInternetの機能，Internet環境で何ができるのかということである．

基本アプリケーション

まず，ネットワークである以上あって然るべきといえるような基本的なものから見ていくことにしよう．

電子メール：電子メールのサービスはネットワーク環境のアプリケーションとしては非常に有名であり，これを提供しないネットワークは皆無なのではないかという感さえある．Internet環境でも当然電子メールはサポートされているが，小規模なネットワークと違うのは，ユーザーが膨大な数になることである．Internet環境では，接続されているマシンの数が数百万台にもなると聞いている．こうした中で，電子メールを正しくあて先まで届けるには工夫が必要である．現状では，電子メールのあて先はマシン名で決定しているが，あて先となるユーザーがいるマシンはネットワークのはるか向こうかもしれないので，メールをネットワーク間で転送する手段が必要となる．このために“そのユーザーに届けたいならあのマシンに送りなさい”という指示をしてくれる，ドメインネームサーバと呼ばれるコンピュータが用意されている．

遠隔ログイン：遠隔ログインは，ネットワーク上の各マシンにログインするためのサービスである．このサービスが提供されることによって，自分の手元にあるマシンからネットワークの向こうにあるマシンを呼び出し，使用することができる．さらに，Xなどのネットワークトランスペアレントなウィンドウシステムと併用すれば，LANと変わらない分散処理環境を構築することができる．

ファイル転送：ネットワーク上にマシンが分散している状態では，各マシンでデータの共有をはかるうえでもファイル転送サービスが必要となる．あるマシンから別のマシンへファイルをコピーするという単純な機能なのだが，電子メールやネットワークニュースの転送など，多くのサービスがファイル転送機能を使っていることを考えれば，最も重要なネットワークサービスであるといえるだろう．

ネットワークの管理サービス

ユーザーの目に直接触れ，ネットワーク環境を実感させてくれるようなサービスである基本的なアプリケーションに比べると目立たないが，ネットワークを維持管理していくためにはなくてはならないサービスもある．

ルーティング：ルーティングとは，データを正しいあて先に転送するための機能である．Internetでは，あるマシンへ到達するための経路は複数ありうる．このため，複数のネットワークに接続されているネットワークからデータを送る場合には，どの接続点からデータを送り出したら正しくあて先に届けられるのかを判断する必要がある．このために使われるのがルーティング（経路制御）の技術であり，Internetを正常に運用していくためにはなくてはならない機能である．

ネームサービス：Internetでさまざまなサービスを運用するためには，多くの名前を見分けなくてはならない．たとえば，ネットワーク上のマシンの名前を知らなくては電子メールやファイル転送のあて先を指定することができない．しかし，現在の巨大なInternetでは，すべてのマシンがすべての名前を記憶しておくことなど到底できない．量も膨大なうえ，世界中のどこかで常に成長し続けているという状態では，名前の対応を正しく維持するためのデータのアップデートだけでネットワークの能力を使い切ってしまう．そのため，いくつかの重要なマシンが周辺のネットワークやマシン，ユーザーの名前を記憶しておき，ほかのマシンは必要に応じて問い合わせを行なうことで必要最小限の情報を入手できるようにしてあることが多い．こうした機能も，ネットワークの重要なサービスだといえる．

ネットワーク環境では，このほかにも各種のサービスが運用されており，その数は増える一方である．しかし，こうしたサービスを日常的に使用しているユーザーでも，こうしたサービスがどのようなメカニズムによって提供されているのかということについては意外に無関心なのではないだろうか．そこで，以降ではネットワークサービスを実現するためのメカニズムに注目していきたい．

電子メール

ネットワークニュース

ネットワークサービスのプロトコル

ここでは，具体的なプロトコルとしてSMTPとftpを取りあげ，そのプロトコルを直接操作してみる．この作業によって，プロトコルに対するイメージが変わるかも知れない．

1. 電子メールのメカニズム

メールを送るには

それでは，ここで電子メールを出す場合のことを考えてみよう．電子メールを出すには，普通何らかのメールプログラムを起動するだろう．たとえば，非常に一般的な（かつ古くもある）プログラムである，/usr/ucb/Mailを起動してみると，図1のようになる．

ここでは，Mailコマンドの引数としてあて先のユーザー名を指定している．すると，まずメールのサブジェクトを聞かれるので(Subject:)，適当なサブジェクトを入れる．すると，本文の入力ができるようになるので，メールの本文を書く．本文の入力を終了するには，行の先頭でピリオドを打つか，ctrl-Dを入力する．最後に，指定したあて先以外のところにも今のメールのコピーを送るかどうか聞かれるが(Cc:)，特に指定しない場合にはただリターンを打つとメールが発信される．この操作は，もちろん使っているメールプログラムによって多少の違いはあるが，入力すべき項目などはどれでもほぼ同じようなものである．

では，電子メールがどのように実現されているのかを見てみよう．とはいえ，いきなりメールプログラムのソースコードを読むというのも極端なので，まずは遠隔ログインコマンドtelnetを使ってちょっとした実験を行なうことにする．

SMTPサーバとは

SMTPとは，Simple Mail Transfer Protocolの略であり，TCP/IPネットワークで幅広く採用されているメール転送のプロトコルである．このプロトコルは一般的に用いられており，UNIXのメールはほとんどがこのプロトコルを用いて実装されている．また，Internetでも広く普及している．このプロトコルを実装し，実際にネットワーク間でのメール転送を行なってくれるのがSMTPサーバである．

先ほど紹介した/usr/ucb/MailなどのメールプログラムはすべてSMTPサーバではない．そのため，SMTPサーバがなければネットワーク経由でのメールの配送はできない．では，SMTPサーバはどこにあるかというと，UNIXでsendmailと呼ばれているメール転送デーモンがSMTPサーバとして動作するのである．

sendmailが動いているマシンでは，メールプログラムはsendmailに依頼することで，ネットワークを介してメールを配送することができる．sendmailは，メールプログラムからメールを受け取ると，そのあて先を調べる．あて先がローカルホスト上のユーザーであれば，/bin/mailコマンドを使ってローカルホストのスプールにメールを送り，ほかのマシンのユーザーあてであれば，ネットワーク経由で配送する．ローカルかリモートか，リモートであればどこのホストかなどといったメールの送り先は，管理者がsendmail.cfという設定ファイルで設定する．ネットワーク経由で配送する場合には，設定に従って，指定されたSMTPサーバに対してメールを配送し，あて先ユーザーへの配信を依頼するのである．

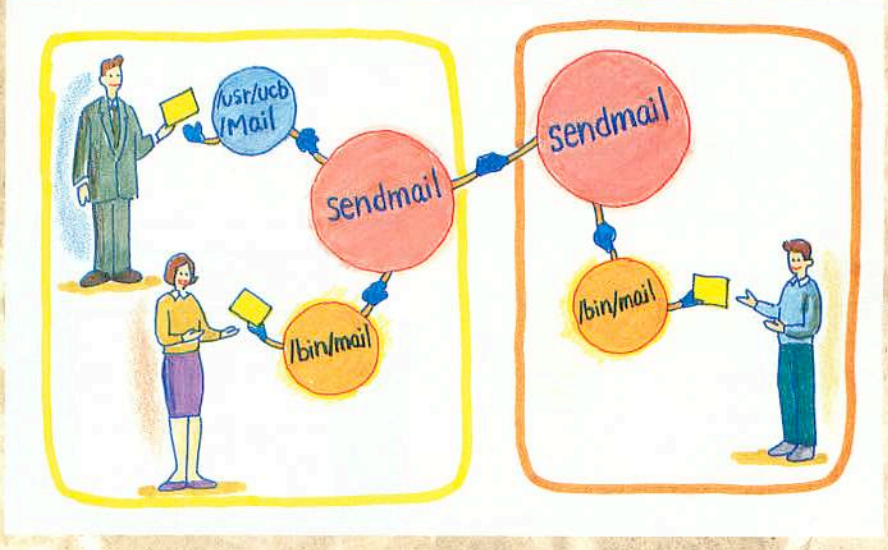
図1　/usr/ucb/Mailの実行例

```
%(toshi-w@pbguru) mail toshi-w
Subject: test

This is test mail.
.
Cc:
%(toshi-w@pbguru)

From toshi-w Tue Feb 18 06:34:31 1992
Date: Tue, 18 Feb 92 06:34:30 JST
From: Toshikazu Watanabe <toshi-w>
To: toshi-w
Subject: test

This is test mail.
```

ポートへのアクセス

では，SMTPのプロトコルの実際の仕組みをみてみよう．ここでは，マニュアル

●電子メールの転送の仕組み



UNIX
TCP/IP
ネットワーク入門

図2　telnetによる遠隔ログイン

```
(superascii)% telnet sass
Trying 133.152.XXX.XXX ...
Connected to sass.
Escape character is '^]'.


SunOS UNIX (sass)

login: editor
Password:
Last login: Tue Feb 18 11:14:54 from superascii
SunOS Release 4.1.1-JLE1.1.1 (GENERIC) #1:
 Tue Nov 13 16:34:44 JST 1990
Erase is Backspace
Interrupt is Delete

(sass)%
```

やドキュメントで調べるのではなく，実際にSMTPのプロトコルに従ってメールの転送を行ないながら調べてみることにする．ただし，/usr/ucb/Mailなどのメールプログラムは使用しない．メールプログラムを使用してメールを送ってみても，肝心のSMTPサーバとのやりとりは普通はユーザーには見えないようになっているからである．これは，SMTPサーバとのやりとりなどは普通は見える必要がないからであるが，SunOSでは，

```
% /usr/ucb/Mail -v
```

とすると，プロトコルに従ったやりとりがある程度見えるので，まったく見る方法がないということではない．

メールプログラム以外にも，SMTPサーバと通信する手段がある．ほとんどありとあらゆる通信に使用可能な汎用プログラムが実はある．それはtelnetである．本来遠隔ログインコマンドなのだが，このコマンドには意外に知られていない使い方がある．それは，指定した任意のポートに接続することである．

ポートについては3章で詳しく説明するので，ここでは大雑把なイメージの説明にとどめるが，要はネットワークに対して開かれたUNIXのサービス窓口であると思えばよい．この窓口（ポート）はたくさん用意されているが，それぞれどのようなサービスを提供するのかが決まっている．そのため，普通のネットワークサービスのクライアントは自分が希望するサービスを提供してくれる窓口をまっすぐ訪ねていくのだが，telnetの場合はどの窓口を訪ねるかをユーザーが選択することができるのである．もちろん，特に指定のない場合は遠隔ログインサービスを提供してくれるポートに直行するのはいうまでもない．

図2は，telnetを普通の遠隔ログインのために起動した例である．この場合は，引数として指定したホスト名を持つマシンに対してログインを実行する．しかし，ホスト名のあとに続けてポート名を指定すると，指定したマシンの指定したポートにアクセスするのである．通常の遠隔ログインの場合には，login: というプロンプトが表示され，通常のログインシーケンスが実行されるが，遠隔ログインサービスのポート以外のところにアクセスした場合は，もちろんlogin: プロンプトは表示されない．どのような返事が返ってくるか，または何の返事も返ってこないかは，どのポートに接続したのかによって異なるのである．

こうした仕組みはSMTPサーバについてもまったく同様である．メールを転送しようとするSMTPサーバは，ほかのマシンのSMTPサーバのポートにアクセスし，そこでプロトコルに従ってユーザーが存在するかどうかの確認などを行ない，ユーザーがそのマシンにいた場合はメールの転送処理を行なうのである．

2. telnetによるSMTPサーバへのアクセス

手作りメールの発信

さて，道具立てが整ったところで，いよいよSMTPサーバにアクセスしてみよう．まず，SMTPサーバが動いているマシンを探してホスト名を調べておく．“どのホストでSMTPサーバが動いているのか”などという具体的なサイトごとの事情については，ネットワークの管理者に確認する必要があるだろう．

今回の実験では，superasciiというホストから，mailserverというホストで動いているSMTPサーバにtelnetを使ってアクセスしてみた．SMTPサーバのポートの名前は，そのものずばりのsmtpとなっているので，実際のコマンドラインは以下のようになる．

```
% telnet mailserver smtp
```

もったいをつけた割には妙に呆気ないという印象を受けるかもしれないが，これでネットワーク上のホストmailserverのSMTPサーバのポートに接続することができるのである．

では，本来ならローカルのメールプログラムがやるはずの仕事をすべて肩代わりして，プロトコルを直接操作することでリモートのSMTPサーバに対してメールの転送を依頼してみよう（図3）．

SMTPサーバに接続すると，telnetでの通常の遠隔ログインと同様に接続先のIPアドレスが表示され，続いて接続に成功したことを示すConnected ...　というメッセージとtelnetのエスケープキャラクタの表示が行なわれる．しかし，ここから先はかなり趣が違っているので，図2と図3をよく見比べていただきたい．

通常の遠隔ログインでは，ここでOSのログインメッセージが表示され，続いてIDとパスワードの入力に移るのだが，SMTP

図3　telnetによるSMTPサーバとの対話例

```
(superascii)% telnet mailserver smtp
Trying 133.152.XXX.XXX ...
Connected to mailserver.ascii.co.jp.
Escape character is '^]'.
220 mailserver.ascii.co.jp Sendmail SMI4.1/smtpdemo-1.1 \
ready at Thu, 30 Jan 92 02:27:07 JST
helo superascii
250 mailserver.ascii.co.jp Hello superascii, pleased to meet you
mail from: <writer>
250 <writer>... Sender ok
rcpt to: <editor>
250 <editor>... Recipient ok
data
354 Enter mail, end with "." on a line by itself
Date: Thu, 30 Jan 92 02:27:11 JST
From: GENKOU generator <writer>
To: editor
Subject: smtp demo

smtp demo.
.
250 Mail accepted
quit
221 mailserver.ascii.co.jp delivering mail
Connection closed by foreign host.
(superascii)%
```

図4　SMTPサーバのエラーメッセージの例

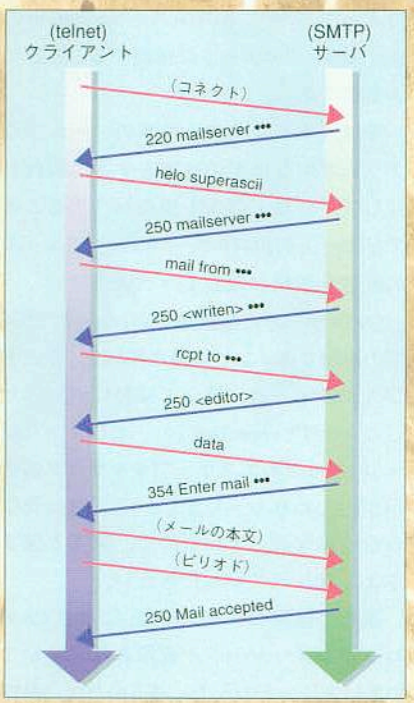
```
User unknown?
```

サーバへのアクセスでは，220という番号に続いてSMTPサーバのログインメッセージ（ウェルカムメッセージ）が表示されている．さらに，“Hello superascii，pleased to meet you”という挨拶までされてしまっている．

次の“mail from: <writer>”という行は，ユーザーがキーボードから入力したものである．意味は，“ユーザーwriterからのメールを配送します”ということである．続く250で始まる行は，“メールの発信元ユーザーを確認しました”というSMTPサーバからのメッセージである．続いて，メールの受け取り人（rcpt，メールのあて先ユーザー）を送ると，やはり確認メッセージが返ってきている．このメッセージには，“私は，ユーザーwriterあてのメールを受け取って正しく配送することできるので安心して送ってください”という確認の意味がある．これで，あて先がeditorのメールを送るには，このmailserverのSMTPサーバに送ればよいことが確認できたことになる．もしもmailserverが担当しているユーザーの中にeditorというユーザーがいなかった場合には，ここでSMTPサーバからエラーメッセージが返ってくる（図4）．これは，接続したSMTPサーバが間違

図5　SMTPサーバからのメールの内容

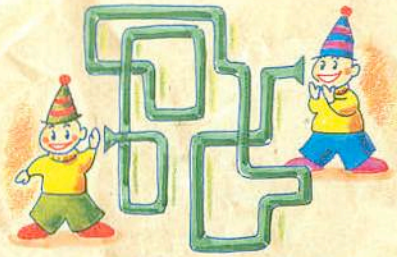
```
From writer Thu Jan 30 02:28:09 1992
Received: from superascii by mailserver.ascii.co.jp (SMI4.1/smtpdemo-1.1)
        id AA00752; Thu, 30 Jan 92 02:27:17 JST
Return-Path: <writer>
Message-Id: <9201291727.AA00752@mailserver.ascii.co.jp>
Date: Thu, 30 Jan 92 02:27:11 JST
From: GENKOU generator <writer>
To: editor
Subject: smtp demo

smtp demo.
```



●SMTPサーバとのやりとり

図6　SMTPサーバのhelpメッセージ表示

```
(superascii)% telnet mailserver smtp
Trying 133.152.XXX.XXX ...
Connected to mailserver.
Escape character is '^]'.
220 mailserver.ascii.co.jp Sendmail SMI4.1/smtpdemo-1.1 \
ready at Thu, 20 Feb 92 05:16:36 JST
help
214-Commands:
214-    HELO    MAIL    RCPT    DATA    RSET
214-    NOOP    QUIT    HELP    VRFY    EXPN
214-For more info use "HELP <topic>".
214-smtp
214-To report bugs in the implementation contact Sun Microsystems
214-Technical Support.
214-For local information contact postmaster at this site.
214 End of HELP info
quit
221 mailserver.ascii.co.jp closing connection
Connection closed by foreign host.
(superascii)%
```



UNIX
TCP/IP
ネットワーク入門

っていたということである．前述したように，どのユーザーあてのメールをどのSMTPサーバに送るかは，管理者が設定しておくことになるが，設定に誤りがあった場合には配送が失敗することになる．

さて，ユーザーの確認が無事に済むと，次はメール本体の転送である．キーボードから"data"と打つとメール本体の転送を要求したことになり，SMTPサーバからは転送方法の指示が返ってくる．ここで，普通の転送であればメールヘッダ付きの本文を単に転送するだけで済むのだが，今回は何もないので，メールが備えているべきメールヘッダを手で入力してみる．ヘッダの内容は，日付，差出人，あて先，サブジェクトである．普通は，メールヘッダはメールプログラムが自動的に作成し，本文の前につけてくれるものである．ヘッダができたら続いて本文を入力し，最後にピリオドだけの行を送るとメール本文の転送が終了する．SMTPサーバに転送されたメールは，SMTPサーバによって目的のユーザーまで配送される．図5は，この操作の結果ユーザーeditorに送られたメールである．さきほどの例で打ち込んだ日付などのヘッダの上に，さらに見かけないヘッダがついているが，これはSMTPサーバが配送の際に付け加えたものである．それ以下の部分を図3と見比べてみると，打ち込んだとおりに正しく転送されていることが分かるだろう．

これで，メールプログラムがなくてもtelnetでプロトコルどおりに通信を行なえば，ネットワーク経由でメールを送ることができることが確認できた．

SMTPプロトコルの実際

telnetを使ってSMTPサーバに直接接続し，メールを出してみるという実験は無事に成功したので，今度はSMTPのプロトコルの中身をもう少し詳しくみていこう．まずは復習になるが，SMTPのプロトコルが使われるのはどのような状況であったかを考えてみると，ほとんどがSMTPサーバ，つまりsendmail同士でのメールの転送処理である．つまり，このプロトコルは基本的にはプログラム同士で使うのであり，人間が直接操作するものではない．

直接手でメールの転送などをやってみると，まるで人間と対話しているかのような気になる"pleased to meet you"などというメッセージも，実は冗長なものであるといえる．実際，サーバから送られてくるメッセージの先頭には"220"とか"250"とかの数字が付いていたが，プログラム同士の通信であれば，これだけで十分明瞭であり，間違える恐れもない．では，なぜ人間が読める形でのメッセージが使われているのだろうか．

その理由は，今回の実験を振り返ってみればよく分かると思うが，ネットワークプログラムのデバッグを考えてのことだといわれている．つまり，数字しか返ってこないよりも，意味が一目で分かるメッセージが表示されるほうが動作の確認がしやすいということである．ネットワーク環境で動作するプログラムが不具合を起こすと，その原因を突き止めるのはたいへんなので，その手間を少しでも減らすためにメッセージをつけてあるのだ．

SMTPのプロトコルには，ヘルプメッセージの表示コマンドも用意されているので表示させてみると，使えるコマンドの一覧が出てくるが(図6)，コンピュータ同士の通信でヘルプを参照するとも思えない．これも，実は人間が直接操作する場合のためにあるといえるだろう．

また，asciiキャラクタベースの通信では，通信相手のハードやソフトに依存しないプロトコルにしておくことができる．実際，SMTPのプロトコルはTCP/IPベースのネットワーク環境であれば，機器ごとの特性からは独立して存在できる．このようなプロトコルにしておけば，たとえ環境が変わっても同じプロトコルを使うことができるので，汎用性が高くなる．また，最近では当たり前となった感のある"異機種間接続"のためにも役立つだろう．実際，SMTPのプロトコルは広く使

図7　ftpの実行例

```
(superascii)% ftp localhost
Connected to localhost.
220 superascii FTP server (SunOS 4.1) ready.
Name (localhost:testuser): testuser
331 Password required for testuser.
Password:
230 User testuser logged in.
ftp> pwd
257 "/" is current directory.
ftp> cd etc
250 CWD command successful.
ftp> dir
200 PORT command successful.
150 ASCII data connection for /bin/ls (127.0.0.1,1125) (0 bytes).
total 497
lrwxrwxrwx  1 root     staff          10 Oct 12 20:56 adm -> ../var/adm
-rw-r--r--  1 root     staff         901 Oct 12 23:04 aliases
-rw-rw-rw-  1 root     staff           0 Oct 12  1990 aliases.dir
-rw-rw-rw-  1 root     staff        1024 Oct 25 16:06 aliases.pag
lrwxrwxrwx  1 root     staff          14 Oct 12 20:56 arp -> ../usr/etc/arp
...
drwxr-sr-x  2 uucp     uucp          512 Oct 12  1990 uucp
lrwxrwxrwx  1 root     staff          15 Oct 12 20:56 vipw -> ../usr/etc/vipw
-rw-r--r--  1 root     staff         134 Feb 18 20:09 xtab
-rw-------  1 root     staff           0 Oct 12 22:35 ypbind.lock
226 ASCII Transfer complete.
7333 bytes received in 0.88 seconds (8.1 Kbytes/s)
ftp> get hosts
200 PORT command successful.
150 ASCII data connection for hosts (127.0.0.1,1126) (92261 bytes).
226 ASCII Transfer complete.
local: hosts remote: hosts
94029 bytes received in 1.3 seconds (73 Kbytes/s)
ftp> bye
221 Goodbye.
(superascii)%
```

われており，いろいろなマシンとのメールのやりとりが可能になっている．

これで，SMTPのプロトコルがどのようなものであるかはイメージがつかめたと思う．実際，プロトコルといっても意味不明のバイナリメッセージのやりとりばかりではなく，使われる局面に応じていろいろな形態がありうる．SMTPなどは，ある意味で典型的なコマンドタイプのインターフェイスを提供しているが，ネットワークサービスのサーバ／クライアントレベルで使われるプロトコルは，こうした形態のものも多い．

3. telnetによるftpへの接続

ftpへの接続

SMTPによる手動電子メール発信の実験は，単純にいってしまえばコマンドを使ってサーバ側の用意を整えておいてから，メールの本体（ファイル）を送るという手順であった．つまり，これはファイル転送の１種なのである．一方，ファイル転送プロトコルとしては，ftpが一般的に使われている．そこで，SMTPへの接続が成功したのだから，次はftpへの接続も実験してみよう．

まず，図７はftpコマンドを使ってファイル転送を行なった結果である．ここでは転送元ホストとしてlocalhost，つまり自分自身を指定しているが，もちろんネットワーク上のホストを指定した場合とまったく同じように実行できる．このレベルでは，"実際には自分自身の内部処理で話が済むから，cpを使おう"というように融通をきかせる（手を抜く？）ことはない．UNIXではループバックインターフェイスが用意されており，ネットワークに接続されていないマシンであっても通信相手に自分自身を指定すればネットワーク関連のアプリケーションをそのまま実行できるのである．

さて，図７をよく見ていただきたい．ごく普通のftpの実行例だが，各メッセージの先頭に数字が表示されているのがお分かりだろうか．普通にftpコマンドを使用しているときには意識しないことが多いと思うが，この数字はSMTPのときに表示されていたものと同様，サーバからのメッセージなのである．こんなものが表示されているということは，実はftpコマンドは非常に簡単なインターフェイスを提供しているだけであり，プロトコルに従った操作をほとんどそのままユーザーに見せているのではないかという気がしてくる．逆にいうと，特別な知識なしにいつもどおりの操作でftpのプロトコルをそのまま使えるのではないかと予想できる．それでは，実際に試してみよう．

図８telnetによるftpサーバへの接続

```
(superascii)% telnet localhost ftp
Trying 127.0.0.1 ...
Connected to localhost.
Escape character is '^]'.
220 superascii FTP server (SunOS 4.1) ready.
user testuser
331 Password required for testuser.
pass abcdef
230 User testuser logged in.
pwd
257 "/" is current directory.
cwd etc
250 CWD command successful.
list
425 Can't build data connection: Connection refused.
get hosts
500 'GET hosts': command not understood.
retr hosts
425 Can't build data connection: Connection refused.
quit
221 Goodbye.
Connection closed by foreign host.
(superascii)%
```

図９ ftpサーバのhelpメッセージ表示

```
(superascii)% telnet localhost ftp
Trying 127.0.0.1 ...
Connected to localhost.
Escape character is '^]'.
220 superascii FTP server (SunOS 4.1) ready.
user testuser
331 Password required for testuser.
pass abcdef
230 User testuser logged in.
help
214-The following commands are recognized (* =>'s unimplemented).
   USER    PORT    RETR    MSND*   ALLO    DELE    SITE*   XMKD    CDUP
   PASS    PASV    STOR    MSOM*   REST*   CWD     STAT*   RMD     XCUP
   ACCT*   TYPE    APPE    MSAM*   RNFR    XCWD    HELP    XRMD    STOU
   REIN*   STRU    MLFL*   MRSQ*   RNTO    LIST    NOOP    PWD
   QUIT    MODE    MAIL*   MRCP*   ABOR    NLST    MKD     XPWD
214 Direct comments to bugs@Sun.COM.
quit
221 Goodbye.
Connection closed by foreign host.
(superascii)%
```

ファイル転送はできるか

図８は，ftpサーバにtelnetで接続してみた結果である．パスワードがそのまま表示されてしまうところはいささか情けない気もするが，そのほかは本当に，普段ftpコマンドを使ってやっているのと同じ操作でできてしまう．

しかし，注目していただきたいことがいくつかある．まず，コマンド名がいつもとは違っていることである．図８では，getコマンドが失敗し，正しいコマンド名はretrであった．ヘルプメッセージ（図９）

UNIX
TCP/IP
ネットワーク入門

をみると正しいコマンド名は分かるのだが，なんでこんなところが変えてあるのかはよく分からない．見たところ，コマンド名はいくつか3文字のものもあるが，ほとんどは4文字で揃えてあるようだ．

それよりも重要な点は，失敗するコマンドがいくつかあることだ．pwd，cwdはそれぞれ現在のワーキングディレクトリの表示，ワーキングディレクトリの変更(cd)を行なうコマンドだが，これらは問題なく実行できた．次にlist(ls)コマンドを実行したところ，data connectionが確立できないというメッセージが表示され，ファイル名一覧をみることはできなかった．念のためお断わりしておくが，これは偶然のエラーや不具合ではない．ftpの仕様によるのである．

ftpの仕組み

telnetでftpサーバに接続すると，ftpサーバと通信することができる．しかし，実際のファイル転送など，データの送受信はできないのである．図8の例では，こちらからコマンドを送ることはできているし，サーバからの返事を受けとることもできている．しかし，これはプロトコルに従った交信であり，データではない．つまり，これらの交信は“データを転送するための準備として必要なやりとり”として別扱いになっているのである．

SMTPサーバとの通信では，プロトコルに規定されているコマンドのやりとりとデータのやりとりとは特に区別されていなかった．これは，実は使っているポートの数の問題なのである．SMTPサーバが使っているポートは1つである．しかも，データのやりとりはasciiキャラクタとして行なわれるため，telnetで接続しても何も不自由のない通信が可能であった．

一方，ftpサーバはコントロール用とデータ用の2つのポートを持っている．図8の実験でtelnetで接続したのは，このうちのコントロール用のポートである．データ用のポートは，コントロール用のポートを使ってデータ転送のコマンドを送ると，そのたびごとに用意され，接続されるようになっている．しかし，telnetは同時に2つのポートを使って通信するようにはなっていないため，この場合はデータ用のポートと接続することはできないのである．

UNIXのネットワークサービスの中でも，2つのポートを使って通信する例はまずない．SMTPのように1つのポートですべての作業を行なうのが合理的であり，2つのポートを使用するのは特別な例であるといえる．

ftpがデータ転送のために専用のポートを使用するのは，ftpが転送データの終わりを伝える方法と密接に関連している．これは，ファイルの終わりをどうやって相手に伝えるかということである．ftpでは，ファイル転送が終了したことを相手に伝えるために，実に単純な方法を採用している．それは，“データ転送用のポート同士の接続を閉じたら，そこでファイル末であるとみなす”という方法である．このやり方では，ファイル末になったことを示す情報をわざわざ送る必要はない．ファイルを送り出す側が，必要なデータをすべて送り出した後でポートの接続を閉じればよいのである．このような方法だと，データの転送を行なうたびに接続が切れてしまうため，ポートを1つしか使わずに通信を行なっている場合はほとんど使いものにならなくなってしまう．そのため，任意に接続を切れるポートを別に用意したということである．

ftpのコマンドの中で，ls (list)，get (retr)，put (stor) などは，データ転送用のポートを必要とするため，telnetで接続してみた場合にはデータ転送用のポートを接続できずにエラーとなってしまう．

しかし，私見ではあるがこの実装は奇妙な方法のようにも思える．ファイルの終了を知るためにいちいちポートの接続を閉じなくても，最初に転送するデータ量（バイト数）を知らせておけば，それでファイル転送の終了を受け側で確認することはできるように思う．なぜそうしなかったのか，そういう方法ではなにか不都合があるのかどうかについてはよく分からないが，ポートを2つ使用することなくファイル転送ができるほうが便利なような気がする．

4. ネットワークサービスのプロトコル

これまで，SMTPとftpの2つのプロトコルについて，telnetで直接操作してみた．その具体的な方法や結果については違いもあるものの，基本的な部分については共通しているといえるだろう．

つまり，

- コマンドとそれに対する応答というスタイルである
- やりとりはテキストベースで，人間にも分かる形で行なわれる

ということである．もちろん，ネットワークサービスのすべてのプロトコルがこうなっているわけではない．しかしこうした性質は，具体的な処理を行なうアプリケーションのプロトコルとしては比較的一般的なものである．ともあれ，一口に“プロトコル”といっても絶対理解不可能な奇怪なやりとりが行なわれているわけではないことはお分かりいただけたものと思う．

こうしたプロトコルには，無駄な点も多々あるのだが，やりとりを直接確認できるので，不具合が起こった場合に障害を見つけ出すためにはたいへん便利であり，さまざまな環境に対応できる．また，やりとりが分かりやすいため，ネットワークサービスのプログラムを自作する際にも対応が楽だろう．

ただし，プログラミングを考えるなら，プロトコルを知っているだけではさすがに材料不足である．今まで使ってきたポートや，ポートをアプリケーションから利用するためのインターフェイスであるソケットについての具体的な知識も必要となるだろう．そこで，次の章ではプログラムの観点からネットワークをみてみることにしよう．

UNIXのネットワークサービスの実装

UNIXのネットワークサービスは，ソケットを使って実装されていることが多いので，ここではソケットの使い方について実際のプログラムに即して見ていくことにしよう．

1. ネットワークサービスのサーバ/クライアント

アプリケーションの起動

ここでは，まずネットワークアプリケーションを起動した場合に何が起こるのかを見ていこう．

ネットワークアプリケーションの特徴は“自分だけでは必要な処理が完結しない”ことである．ネットワークを通じて処理を行なう場合，ネットワークの先にあるマシンを操作する必要がある．しかし，現在のUNIXカーネルだけでは，ネットワーク経由でマシンをリモートコントロールすることはできない．どうしても，要求を受け付けるためのカーネル以外のプログラムが必要となる．そこで，UNIXではこれをデーモンとして実装し，サーバ/クライアントモデルでのサーバとしていることが多い．つまり，クライアントがネットワークの向こうにいるサーバに対して処理を依頼し，サーバが実際に必要な処理を行なうのである．

ネットワーク経由でマシンを直接操作するためのツール，リモートログインコマンドであるtelnet, rloginなどのコマンドも仕組みはまったく同じである．これらのコマンドは，ネットワークの先にあるマシンのコンソールをあたかも手元に持ってきてしまったかのような環境を提供してくれるが，本当に持ってきてしまっているわけではない．これは，telnetでほかのマシンにログインしても，そのマシンのコンソールで作業していたユーザーには何の支障もないことからも明らかであろう．telnetは，ユーザーのキー入力をネットワークを介してサーバに送り，サーバがそのキー入力をシェルに送る．また，シェルの出力はサーバが受けとり，ネットワークを介してtelnetに送り返されるのである．このような仕組みになっているため，たとえばシングルユーザーモードで動作しているようなサーバがないマシンには，telnetでログインすることはできないのである．

ネットワークアプリケーションを有効に活用するために不可欠な要素であるサーバは，UNIXではたいていデーモン(daemon)として実装されている．

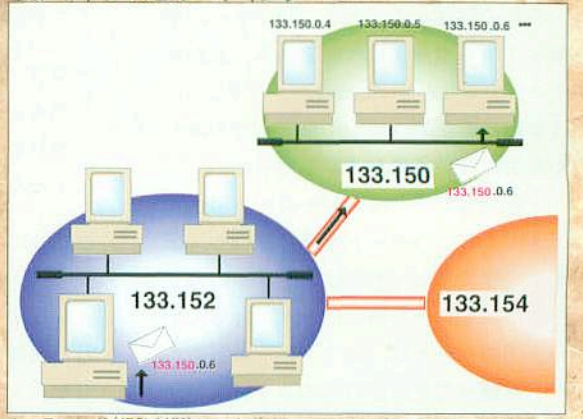
図1　動作中のデーモンの例

```
  PID TT STAT  TIME COMMAND
    0 ?  D     0:03 swapper
    1 ?  S     1:07 /sbin/init -
    2 ?  D     0:04 pagedaemon
   54 ?  S     2:46 portmap
   62 ?  S    15:52 gated /var/log/gated.log
   80 ?  R    70:08 in.named
   87 ?  I     0:31  (biod)
   88 ?  I     0:30  (biod)
   89 ?  I     0:30  (biod)
   90 ?  I     0:30  (biod)
  136 ?  S     8:38 syslogd
  155 ?  IW    0:00 rpc.mountd -n
  158 ?  I     0:00  (nfsd)
  164 ?  I     0:00  (nfsd)
  165 ?  I     0:00  (nfsd)
  166 ?  I     0:00  (nfsd)
  168 ?  I     0:00  (nfsd)
  169 ?  I     0:00  (nfsd)
  170 ?  I     0:00  (nfsd)
  171 ?  I     0:00  (nfsd)
  181 ?  I    38:22 update
  185 ?  I     2:17 cron
  199 ?  S     1:50 inetd
  204 ?  IW    0:00 /usr/lib/lpd
11295 ?  S     0:28 -accepting connections (sendmail)
20513 ?  R     0:27 -AA16126 ascgw.ascii.co.jp: user open (sendmail)
20897 ?  I     0:00 -AA20838 ascgw.ascii.co.jp: user open (sendmail)
20900 ?  S     0:00 -sh.wide.a (nntpd)
18714 ?  IW    0:00 in.rshd
   58 co IW    0:00 keyserv
   65 co IW    0:00 in.snmpd
   67 co IW    0:00 in.snmptrapd
  159 co IW    0:00 rpc.statd
  160 co IW   42:54 rpc.lockd
```

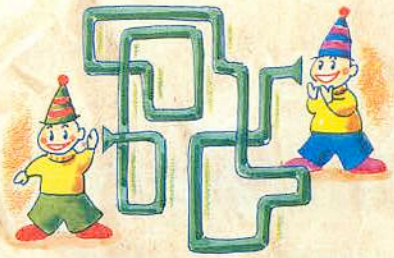
デーモンは待っている

ネットワークサービスのサーバとして動作するデーモンは，ネットワークからのサービス要求が来るのを受けて必要な処理を行なう．そのため，ネットワークサービスを提供しようとするマシンでは，あらかじめ必要なデーモンを起動し，待

●IPアドレスとルーティング



ルーティング(経路制御)は，まずはIPアドレスの内のネットワークアドレスに従って行なわれる．この方法では，ルーティングテーブルの管理はネットワーク単位で行なわれるので，1台1台のホストへの経路をいちいち指定するのに比べて格段に楽になる．



UNIX
TCP/IP
ネットワーク入門

図2　/etc/hostsファイルの例

```
127.1           localhost loghost
#
133.152.12.1    cisco0 b-router         # backbone router
133.152.12.2    pyramid mailhost
#
133.152.12.5    sun0 jserver
133.152.12.6    sun1
#
133.152.16.1    cisco1 e-router         # engineering router (cisco0)
133.152.16.2    vax vax8650
133.152.16.8    news0
133.152.16.9    news1 riscnews
133.152.16.10   luna riscluna
133.152.16.64   ncd0
133.152.16.65   ncd1
133.152.16.66   xmint
#
133.152.24.1    cisco2 m-router         # marketing router (cisco0)
133.152.24.12   ews0
133.152.24.64   fastpath4
133.152.24.80   mac0 fx
133.152.24.81   mac1 cx
```

たせておく必要がある．

デーモンは基本的には１つのサービスを提供し，対応するコマンド名の末尾にデーモンであることを示すdを付加した名前になっていることが多い．たとえば，telnetに対してin.telnetd，ftpに対してin.ftpdがあるといった具合である．最近ではネットワークサービスの種類も増えてきたので，必要なデーモンの数もかなり多くなっている．デーモンが常時要求を待ち続けているのは，システムにとってはリソースを消費されていることを意味する．しかも，サービスは基本的に増える一方なので，デーモンの消費するリソースだけでもかなりの量となってきた．そのため，ネットワークサービスのためのデーモンを統括し，効率的に運用するための仕組みとしてネットワークスーパーデーモンinetd（後述）が使われるようになってきている．

マシン指定のメカニズム

さて，ネットワークサービスを行なうためのソフトウェアとしてサーバ（デーモン）とクライアント（ユーザーコマンド）が揃った．最後に必要となるのは，ネットワーク上で特定のマシンを指定する手段である．

TCP/IPでは，IPアドレスによるホスト指定のメカニズムを提供している．IPアドレスは32bitで表わされる数値であり，普通は8bitずつ４つに区切り，それぞれを10進で表記する．これは，dotted decimal notation という表記法であり，アドレスは133.152.12.34といった形になる．

図3　rwhoコマンドの実行例

```
aki       news0:ttyp0     Feb 18 11:52  5:48
hiro      sun1:ttyp1      Feb 18 12:33  1:07
jun       ews0:ttyp0      Feb 19 21:57  1:53
katsu     pyramid:ttyp8   Feb 18 09:49   :12
kenji     sun1:ttyp4      Feb 17 16:13  3:55
kimi      news0:ttyp8     Feb 18 15:12  1:47
michi     pyramid:ttyp9   Feb 18 11:33  1:01
miho      pyramid:ttyqb   Feb 18 12:11
naoki     pyramid:ttyq9   Feb 17 16:44   :07
rie       news0:ttyp3     Feb 18 09:49   :07
root      ews0:ttyp1      Feb 17 13:12  4:57
tomo      pyramid:ttyq1   Feb 18 17:38   :04
toshi     sun1:console    Feb 18 01:52 15:52
toshi     sun1:ttyp0      Feb 18 01:52
youko     news1:ttyp8     Feb 18 15:33
```

IPアドレスは，意味的にはネットワークアドレス部とホストアドレス部の２つのアドレス指定が含まれた形になっており，IPアドレスだけで“どのネットワークのどのホストか”が特定できるように構成されている．ネットワークアドレス部は，Internet環境での基本的なアドレス指定として使われることが想定されているため，“世界共通の限られたリソース”となっている．そのため、ネットワークアドレス部はユーザーが勝手につけることはできず，申請して割り当てを受けることになっている．一方，ホストアドレス部に関しては各ネットワークのユーザーが自由に設定できるようになっている．

IPアドレスとは別に，ネットワーク上のマシンにはそれぞれ任意の名前が付けられている．たとえば

```
% telnet yours
```

とすればネットワーク上のホストyoursにログインすることができる．この場合，/etc/hostsファイルにIPアドレスとホスト名（例ではyours）の対応が書かれているので，その情報を元にIPアドレスを調べれば，目的とするアクセス先のマシンを特定することができる．

また，ホストを指定してサービス要求するやり方以外に，“誰でもいいから答えてください”というサービス要求もありうる．NIS環境で使われているクライアントマシンでは，いろいろな情報をNISサーバから教えてもらわなくてはならないが，ネットワーク上のどのマシンがサーバなのかは分かっていない．そのため，サーバを見つけるためのypbindコマンドは，ネットワーク内のすべてのホストに一斉に問い合わせを送り，サーバからの答えを待つという方法をとる．

こうした，全ホストに対してデータを送る場合にはブロードキャストという方法が使われる．このために，ブロードキャストアドレスという特別なIPアドレスが用意されており，このアドレスに発信された情報はネットワーク上のすべてのマシンが受けとれるようになっている．

さて，これで通信に必要な道具立ては整った．ネットワークサービスを利用するためにユーザーコマンドを起動し，必要なデーモンが待っているマシン名を指定してやれば，必要なサービスが受けられるはずである．

そこで，次は通信の具体的なメカニズムについてみていこう．

マルチキャストとは

ブロードキャストを使用すると，無関係なノードもパケットを受信する．このパケットは不必要なので，結局は捨てることになるが，この判断自体，無駄な動作といえる．そこで，特定多数のノードだけが受信できるマルチキャストが最近使用されている．これは特殊なEthernetアドレスを宛て先に指定することで実現されており，IPの世界ではCLASS Dがマルチキャストアドレスとして使用されている．

2. portとソケット

UNIXのプロセス間通信

UNIXでは，プログラムはすべてプロセスという単位で実行される．そのため，通信といった場合はまずプロセスとプロセスの通信（プロセス間通信）が基礎となる．プロセス間通信のための機構としては，BSDで使われるソケットとSystem Vで使われるストリームがあるが，ソケットを使うほうが一般的なので，ソケットについて説明しよう．

プロセスは，まずソケットを生成して通信相手のソケットと接続する．コネクションが確立されれば，後はファイルに対するアクセスと同様にread, writeなどでデータのやりとりをすることができる．

まず，ソケットの生成はsocket( )システムコールで行なう．

```
s = socket(domain, type, protocol)
```

ここで，domainはソケット名の通用範囲を指定するためのものであり，一般的には，AF_UNIX，AF_INET，AF_NSの3つから選ぶ．これは，そのソケットをネットワーク環境で使用するか，1つのUNIXシステム内だけで使用するかの指定である．ソケットをほかのソケットと区別するためにはそれぞれのソケットに名前が必要となる．1つのUNIXシステム内であれば，ソケットの名前をファイルシステム内に作ることでこうした名前を区別できるが，ネットワーク環境でソケットを区別するためには“どのマシンのどのソケットか”という区別が必要となるので，名前の付け方を変える必要がある．“どのマシンか”という指定にはIPアドレスが使用されるので，domain指定は実際にはソケットの名前をIPアドレス付きのものにするか，ファイルシステム内に作るかの指定だと考えてよいだろう．

次にtypeだが，これはソケットのデータのやりとりの方法を指定するものである．普通は，byte streamかdatagramと呼ばれるパケット単位の方法を指定する．

streamは通信相手のところまでまっすぐパイプを引いてしまい，その中にデータを流し込んでいくような通信形態である．一度コネクションを確立すれば，それ以降送信したデータは送出したとおりの順番で間違いなく相手に届くことが保証され，同じコネクションで相手からのデータを受けとることもできる．

dgram（datagram）はパケット交換であり，アプリケーションが扱うパケット1つ1つにすべてあて先を明示しておく必要がある．目的の通信相手にたどり着くまでの経路が複数ある場合は各パケットがどの経路を通るか分からないため，送ったパケットがそのままの順序で相手に届くとは限らない．

最後のprotocolは，データ転送に使用するプロトコルを指定するために用意されたものである．しかし，各データ転送タイプ（type）ごとにデフォルトのプロトコルが用意されているため，この引数にはほとんどの場合デフォルト指定である0が入る（囲み参照）．

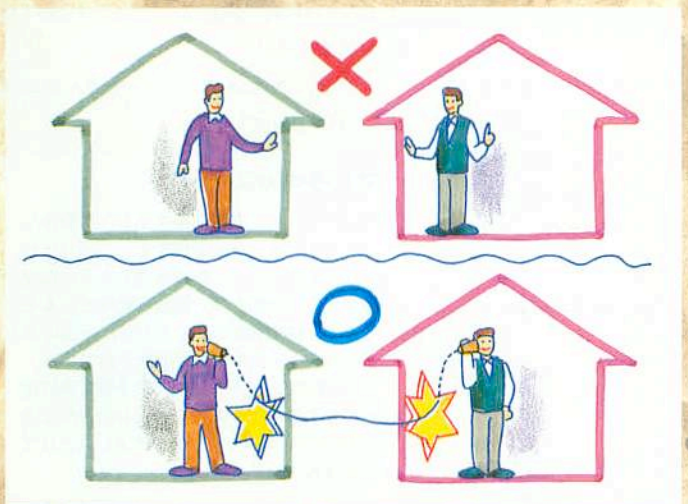
さて，どのようなソケットでも，“どのソケットか”を識別する方法が必要になる．Internetドメインのソケットでは，プロトコルポート番号が使われる．ポートは，1つのネットワークインターフェイスから入ってくるデータを必要なプロセスごとに振り分けるためのデータの出入り口で，ソケットはポートと結合されて使われる．この場合，ポート番号をどうやってソケットに割り当てるかが問題だが，well-knownポート（既知ポート）と呼ばれるポート番号をネットワークサービスごとに固定的に割り当て，この番号割り当てを一括管理するという方法が使われている．これは，電話での110番や119番

protocolパラメータについて

SOCK_STREAMとSOCK_DGRAMは，それぞれTCPとUDPの1つずつしかプロトコルをサポートしていない．したがって，socket(2)の第3パラメータは一般的に0を指定する．しかし，SOCK_RAWは，IPPROTO_RAWとIPPROTO_ICMPの，2つのプロトコルをサポートしており，プロトコルを指定しないとIPPROTO_RAWが選択される．ping(8)のようにICMPを使用する場合，RAWソケットを生成する時に明示的にICMPを指定する．

```
proto = getprotobyname("icmp");
s = socket(AF_INET, SOCK_RAW,
        proto->p_proto);
```

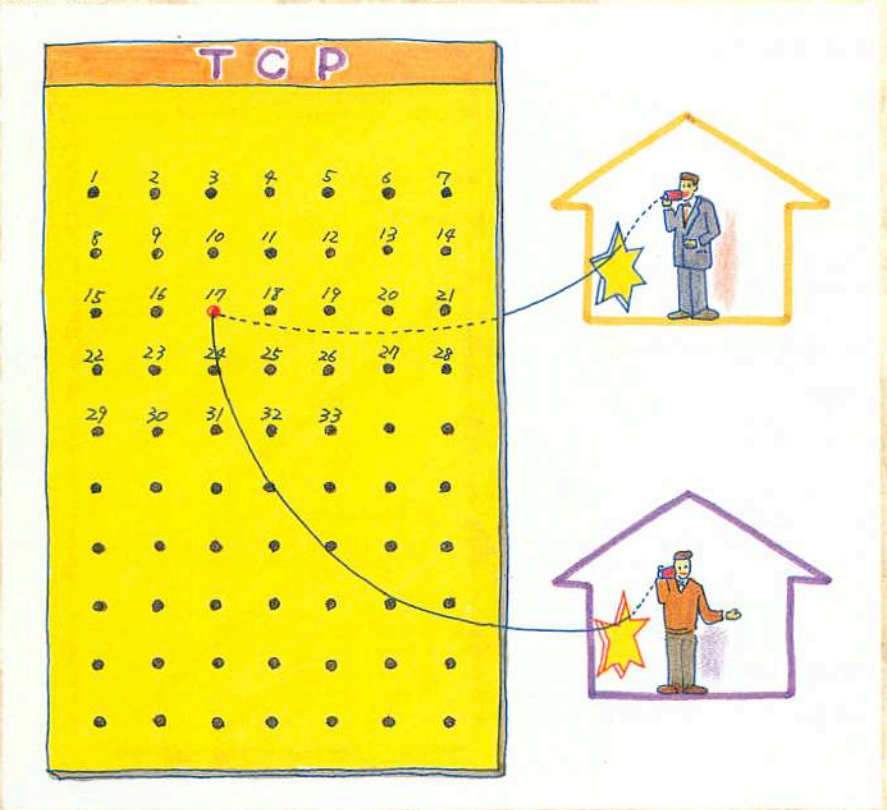
●ソケットのイメージ



●ソケット名の通用範囲

UNIX
TCP/IP
ネットワーク入門

●ポートのイメージ



イアントで違っているのでそれぞれに分けて説明しよう.

サーバは, クライアントからの要求を受けるためにwell-knownポートで待つ. そのため, サーバはbind( )で自分が生成したソケットに対してサービス固有のwell-knownポートの名前を割り当てる.

一方, クライアントにはサーバのような決められたポート番号は必要ないので, そのとき空いているポートを適当に確保すればよい. そこで, 空いているポートをシステムに割り当ててもらい, それを自分が設定したソケット名とする. bind( )でつけた名前は通信相手となるサーバから返事をもらうときに指定してもらうためにあると思えばよいだろう.

次は, コネクションの設立だが, このようなもので, /etc/servicesというファイルに書かれている. ここを調べることで, サービスを提供しているポート番号が分かるので, サーバ(デーモン)の使っているソケット名も同時に分かるという仕組みになっている.

ソケットを生成すると, システムからファイルディスクリプタsが返される. 以降のソケット操作はすべてこのファイルディスクリプタを介して行なう. これで, ネットワーク用のソケットと通常ファイルをまったく同じに扱うための方法が提供されたことになる.

ソケットを生成したら, bind( )システムコールでソケットに名前を付ける.

e=bind(s, myname, mynamelen)

sはsocket( )で得たファイルディスクリプタである. mynameは自分のソケット名の構造体を示すポインタで, mynamelenにname構造体の長さを入れる. name構造体には, domainに対応した種類がある (図5). eにはシステムコールの実行結果を示すエラーコードが返される.

bind( )の処理の意味は, サーバとクラ

図4 代表的なwell-knownポート番号

1～255	オフィシャルなwell-known ports
256～511	UNIXでは未使用
512～1023	UNIX特有のネットワークサービス. もしくは, UNIXでのauthentication用の特権ポート(スーパーユーザの特権を持つプロセスだけが確保できる)
1024～4999	動的にカーネルから割り当てられる(一般的なクライアントが使用する)
5000～	ユーザーリザーブエリアと呼ばれ, スーパーユーザ以外もwell knownポートとして使用できる

リスト1 rshコマンドのソース(基本構造)

```
>>> ${BSDNETWORK}/ucb/rsh/rsh.c 5.7 (Berkeley) 9/20/88 <<<

main(argc, argv0)
        int argc;
        char **argv0;
{
        pwd = getpwuid(getuid());                          # 現在のプロセスUIDからユーザー名を取得する
...
        sp = getservbyname("shell", "tcp");                # rshのサーバのポート番号を取得する
...
        rem = rcmd(&host, sp->s_port, pwd->pw_name,        # in.rshdのshell/tcpにコネクトする
                user ? user : pwd->pw_name, args, &rfd2);
}

>>> ${BSDNETWORK}/lib/libc/net/rcmd.c   5.20 (Berkeley) 1/24/89 <<<

rcmd(ahost, rport, locuser, remuser, cmd, fd2p)
        char **ahost;
        u_short rport;
        char *locuser, *remuser, *cmd;
        int *fd2p;
{
        int lport = IPPORT_RESERVED - 1;                   # 自分のポート番号として，1023以下の特権ポートを確保する
                                                           # このポートが確保できるのはsuperuserのみである
                                                           # つまり，特権ポートからコネクトすればin.rshdに信用してもらえる

        hp = gethostbyname(*ahost);                        # サーバのホスト名からIPアドレスへ変換
        s = rresvport(&lport);                             # 標準入力/標準出力用のソケットを確保
        sin.sin_family = hp->h_addrtype;                   # ソケットアドレス構造体のタイプを設定
        bcopy(hp->h_addr_list[0], (caddr_t)&sin.sin_addr, hp->h_length);   # サーバのIPアドレスを設定
        sin.sin_port = rport;                              # サーバのポート番号を設定
        connect(s, (caddr_t)&sin, sizeof (sin));           # rresvport( )で確保した特権ポートからin.rshdにコネクト
        lport--;
        return (s);
}

>>> ${BSDNETWORK}/lib/libc/net/rcmd.c   5.20 (Berkeley) 1/24/89 <<<

rresvport(alport)
        int *alport;
{
        sin.sin_family = AF_INET;
        sin.sin_addr.s_addr = INADDR_ANY;
        s = socket(AF_INET, SOCK_STREAM, 0);
        for (;;) {                                                    # 特権ポートを確保できるまで繰り返す
                sin.sin_port = htons((u_short)*alport);               # ポート番号として適当な特権ポート番号を設定
                if (bind(s, (caddr_t)&sin, sizeof (sin)) >= 0)        # bind( )が成功すればよし
                        return (s);                                   # 失敗したらポート使用中として，次の番号を試す
                if (errno != EADDRINUSE) {                            # エラーの理由が"ポート使用中"以外であれば
                        (void) close(s);                              # 処理を中断
                        return (-1);
                }
                (*alport)--;
                if (*alport == IPPORT_RESERVED/2) {                   # 未使用の特権ポートがなかったときの処理
                        (void) close(s);
                        errno = EAGAIN;                               # 後で再試行してね，で終了
                        return (-1);
                }
        }
}
```

作業も，サーバとクライアントではやり方が異なる．クライアントは，connect ( ) システムコールを使ってサーバにコネクション要求を送る．

e=connect(s, peername, peernamelen)

先ほどのbind( )とほぼ同様の構文だが，違っているのは，peernameで指定する名前が，コネクトしたい相手(サーバ)のソケット名になることである．

一方，サーバではconnect( )の代わりにlisten( )システムコールを使用する．

e=listen(s, backlog)

こちらは，どのソケットで要求を受けつけるかを示すディスクリプタと，最大いくつの要求をためておけるかを示す値backlogを指定する．これで，コネクションを受け付ける準備が整うので，次にaccept( ) システムコールを発行する．

ns=accept(s, addr, addrlen)

accept( )は，sで指定されたソケットに対するコネクト要求を待つ．コネクト要求がくると，accept ( )はクライアントとのコネクションのために新しいソケットを生成し，ファイルディスクリプタnsを返す．最初のソケットsはwell-knownポートにバインドされているため，ここでコネクションを確立してしまうと，それ以降のコネクト要求を受け付けることができなくなってしまうのである．なお，このコネクションの通信相手となるクライア

UNIX
TCP/IP
ネットワーク入門

リスト2　ライブラリ関数の使用法

```
struct  hostent {
        char    *h_name;        /* ホストのオフィシャルな名前 */
        char    **h_aliases;    /* ホスト名の別名リスト */
        int     h_addrtype;     /* ホストのアドレスタイプ( sin.sin_familyへ代入する) */
        int     h_length;       /* ホストアドレスの長さ */
        char    **h_addr_list;  /* ホストのアドレス( 複数あり得る) */
#define h_addr  h_addr_list[0]  /* 4.2BSDとコンパチブル */
};

struct hostent *gethostbyname(hostname)
char *hostname;

ホスト名からIPアドレスへ変換する関数．通常は以下のように使用する．

        struct sockaddr_in sin;
        struct hostent *hp, *gethostbyname();

        hp = gethostbyname(hostname);
        sin.sin_family = hp->h_addrtype;
        bcopy(hp->h_addr, (char *)&sin.sin_addr, hp->h_length);
==

struct  servent {
        char    *s_name;        /* オフィシャルな名前 */
        char    **s_aliases;    /* サービスの別名リスト */
        int     s_port;         /* ポート番号( ネットワークバイトオーダー) */
        char    *s_proto;       /* 使用するプロトコル( 例えば "tcp") */
};

struct servent *getservbyname(service, protocol)
char *service;
char *protocol;

ネットワークサービスの名前( well-knwon port) から，ポート番号へ変換す
る関数．メンバー s_port は，ネットワークバイトオーダーで返されるので，
そのまま sin.sin_port へ代入できる．

        struct  sockaddr_in sin;
        struct  servent *sp;

        sp = getservbyname("shell", "tcp");
        sin.sin_port = sp->s_port;      /* network byte order */
```

ントのソケット名はポインタaddr，名前の長さはポインタaddrlenで示される整数にそれぞれ格納される．

以上の手順で，ソケットを利用したコネクションが確立できる．

ソケットの具体的な利用法

ここでは，参考までにバークレイ・ネットワークリリースの中に含まれる，リモートシェル/usr/ucb/rshのソースの骨格部分を掲載する(**リスト1**)．これは，エラー処理部分などを取り除き，基本的な構造だけを追えるようにしたものなので，そのつもりで見ていただきたい．

ソースで使われているシステムコールのうち，socket( )，bind( )，connect( )，listen( )，accept( )，の5つについてはすでに説明したので，これらに注目すると具体的な使い方が分かるだろう．また，ポートの使い方に関して注目すべき関数が2つある（**リスト2**）．

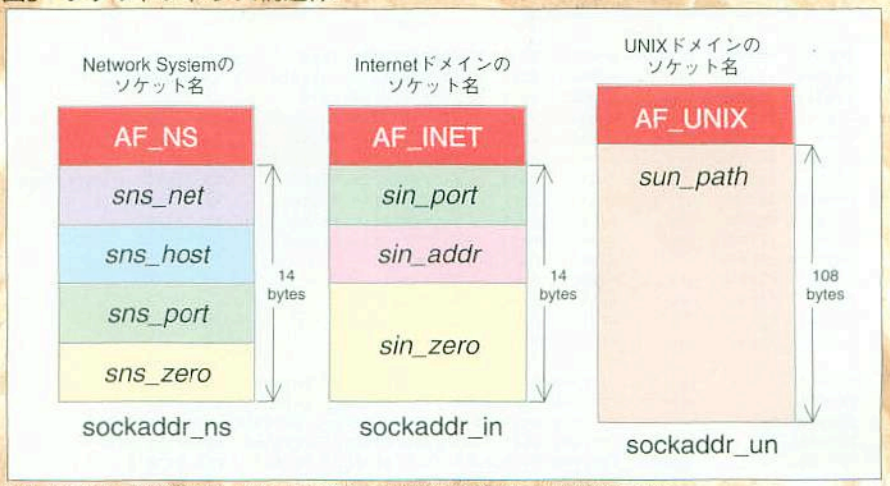
gethostbyname( )は，引数としてホスト名を渡すとそのホストのIPアドレスなどの情報をhostent構造体に格納してくれる．つまり，ユーザーが指定したホスト名から簡単にIPアドレスに変換できるのである．前に，ホスト名とIPアドレスの対応は/etc/hostsを見れば分かると説明したが，NIS環境やドメインネームサーバを利用した環境では，/etc/hostsに正しい情報があるとは限らないので，プログラムはこの関数を使ってIPアドレスを取得する．

同様に，getservbyname( )はサービス名からwell-knownポート番号を得る．

リスト1では，まずmain( )関数でユーザー名やコネクトすべきサーバのポート名に関する情報を取得した後，rcmd( )関数を呼んでいる．rcmd( )は，サーバのIPアドレスを取得した後，rresvport( )関数で確保したポートを使い，サーバにコネクトしている．このあたりの処理は，前に説明した通りなので，リストを見ればお分かりいただけるだろう．

rresvport( )関数はrsh特有の処理で，特権ポートを確保するための関数である．コネクトしたサーバに対してのオーセンティケーションの問題を回避するため，rshは自分のポートとしてスーパーユーザーしか確保できない特権ポートを使用する．しかし，特権ポートの何番が空いているか分からないため，順次違うポート番号を指定しながらbind( )システムコールを繰り返し実行することで，空いている特権ポートを確保している．コメントを参考にしながら，ソースをじっくり眺めていただきたい．

図5　ソケットアドレス構造体



ソケットアドレス構造体には，ソケット名のデータが格納される．AF_NSはXNSネットワークで使用されるものである．AF_INETはポート番号とIPアドレスを含み，AF_UNIXではパス名となっていることが分かる．なお，zeroとあるのはパディングである．

3. ネットワークスーパーデーモンとは

上位インターフェイスとしてのinetd

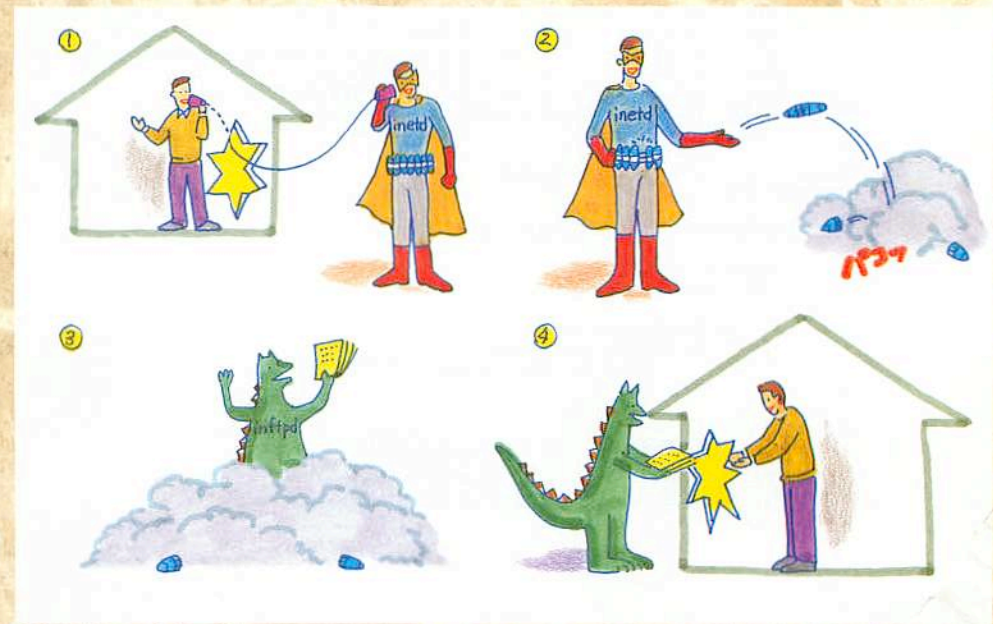
最近ではネットワークサービスの種類も何十という数になり，対応するデーモンを起動しておくだけでもシステムに対して負荷がかかるようになった．デーモンの数が増えることで生じるデメリットとしては，以下のようなものがある．

- デーモンはシステムブート時に起動されるため，ブートが遅くなる
- リソース（プロセステーブル，メモリ，スワップ領域など）を占有される

inetdは，こうしたネットワークサービスのデーモンが増えることによる弊害を最小限に食い止めるために工夫されたもので，ネットワークサービスのための共通インターフェイスとして機能する．

inetdは，クライアントからのサービス要求に応じて必要なデーモンをそのつど起動する．そのため，システム起動時にinetdだけを起動しておけば，あとは要求があるごとに必要なデーモンが起動され，サービスが終わればデーモンも終了してくれる．このため，システム内には現在提供中のサービスに対するデーモンしかプロセスとして存在しなくなるので，システムの負荷が軽くなる．

しかしながら，どんなデーモンでもすべてinetdから起動できるわけではない．



●inetdのイメージ

デーモン自体がinetdから起動されるように書かれていなくてはならないし，必要なサービスの提供が終わったら自分自身も終了するようになっていないと意味が半減する．とはいえ，最近では主要なネットワークサービスのデーモンはほとんどがinetd対応になっているため，inetdを使うのが当然という状況になっている．

inetdのしくみ

inetdは，設定ファイル（通常inetd.conf）に従って，デーモンを起動する．設定ファイルには，実際にサービスを提供するデーモンの名前や，必要なソケットを生成するための情報などが書かれている（図6）．

最初に書かれているのが，公式なサービス名である．この名前は/etc/servicesというファイルにあり，well-knownポートの番号と対応している．

次に，必要なソケットのタイプとプロトコルが書かれている．その後は，実行フラグ（囲み参照）とデーモン実行時のプロセスUIDの指定，実行するデーモンのパス名と続いている．

inetdの動作を追ってみると，以下のようになる．

- サービス名をキーとして/etc/servicesを検索することで，監視する必要のあるwell-knownポートの番号を得る．
- inetd.confに書かれたタイプとプロトコルのソケットを生成し，well-knownポートにbindする
- ソケットに要求が来たら指定されたデーモンを起動し，以降の処理はデーモンに任せる

inetdは，well-knownポートとサーバプロセスが基本的に1対1対応になっていることを利用してwell-knownポートの監視を一

図6 inetd.confファイルの例

```
ftp     stream  tcp     nowait  root    /usr/etc/in.ftpd        in.ftpd
telnet  stream  tcp     nowait  root    /usr/etc/in.telnetd     in.telnetd
shell   stream  tcp     nowait  root    /usr/etc/in.rshd        in.rshd
login   stream  tcp     nowait  root    /usr/etc/in.rlogind     in.rlogind
exec    stream  tcp     nowait  root    /usr/etc/in.rexecd      in.rexecd
finger  stream  tcp     nowait  root    /usr/etc/in.fingerd     in.fingerd
tftp    dgram   udp     wait    root    /usr/etc/in.tftpd       in.tftpd -s /tftpboot
comsat  dgram   udp     wait    root    /usr/etc/in.comsat      in.comsat
talk    dgram   udp     wait    root    /usr/etc/in.talkd       in.talkd
name    dgram   udp     wait    root    /usr/etc/in.tnamed      in.tnamed
daytime stream  tcp     nowait  root    internal
time    stream  tcp     nowait  root    internal
echo    dgram   udp     wait    root    internal
discard dgram   udp     wait    root    internal
time    dgram   udp     wait    root    internal
#
mountd/1        dgram   rpc/udp wait root /usr/etc/rpc.mountd   rpc.mountd
rexd/1          stream  rpc/tcp wait root /usr/etc/rpc.rexd     rpc.rexd
ypupdated/1     stream  rpc/tcp wait root /usr/etc/rpc.ypupdated rpc.ypupdated
rquotad/1       dgram   rpc/udp wait root /usr/etc/rpc.rquotad  rpc.rquotad
rstatd/1-3      dgram   rpc/udp wait root /usr/etc/rpc.rstatd   rpc.rstatd
rusersd/1-2     dgram   rpc/udp wait root /usr/etc/rpc.rusersd  rpc.rusersd
sprayd/1        dgram   rpc/udp wait root /usr/etc/rpc.sprayd   rpc.sprayd
walld/1         dgram   rpc/udp wait root /usr/etc/rpc.rwalld   rpc.rwalld
```

UNIX
TCP/IP
ネットワーク入門

inetd.confの動作フラグ

動作フラグには，nowaitとwaitがある．サーバは，accept( )でコネクト要求を受け付けると，サービスのための新しいソケットを受けとり，fork( )で子プロセスを生成する．このとき，nowaitが指定されていれば，サービスの提供は子プロセスに任せ，親プロセスは元のwell-knownポートで新たなサービス要求を待つ．一方，waitが指定されていると，親プロセスは子プロセスの終了を待ち続け，新たなサービス要求は受け付けない．これは，トランザクション型のサービスなど，1つのサービスが短時間で終わり，1つのコンテクストだけで処理をまかなえてしまう場合に使われる．

手に引き受けており，それ以外の処理はしない．つまり，inetd自身はクライアントとのデータのやりとりはせず，単に必要なデーモンを起動するだけである．

現在では，ネットワークサービスのほとんどがinetdを利用するようになっている．そのため，新しいネットワークサービスをインストールする場合でも，必要なデーモンなどをインストールしたら，あとはinetd.confに登録してinetdを再起動するだけでサービスを提供できるようになるので，管理者にとっても便利である．

ケーブルからinetdへ

ここまでで，UNIXが提供するネットワークサービスについての概要と，ネットワークサービスをプログラムから利用する際に必要となるインターフェイスについての説明がほぼ終了したと思う．そこで，最後に少々趣味的にはなるが，ケーブルを流れるパケットがinetdに受け取られるまでの経路を見ていこう．これは，TCP/IPと呼ばれるプロトコルが実際にどのような形で実装されているのか，どのような階層構造になっているのかを確認することでもある．ネットワークとUNIXの物理的な接点であるネットワークインターフェイスからプロセスに至るまでのパケットの経路をたどってみることで，TCP/IPの構造とそこで果たしている役割が分かるだろう．

物理層：まず，ケーブル上を流れている電気信号が，パケットとして各ホストのEthernetインターフェイスに入ってくる．このときのパケットは，Ethernetフレームと呼ばれるものである．パケットのヘッダには，あて先と送信元がEthernetアドレスで書かれている．Ethernetアドレスは48bitで表わされるインターフェイス固有のアドレスであり，通常は製造時に固定的に割り当てられる．Ethernetアドレスは重複しないように割り当てられているため，1つ1つの機器を識別するための最も確実な手段である．

インターフェイスのコントローラは，ネットワークケーブルから入ってくるパケットをすべて取り込む．そのうえで，取り込んだパケットのヘッダを見て，パケットが自分あてのものでなかった場合は捨てる．Ethernetが一斉同報媒体であるといわれるのはこうした性質による．パケットが自分あてであった場合には，上位プロトコルである，カーネル内のIP層にパケットを渡す．

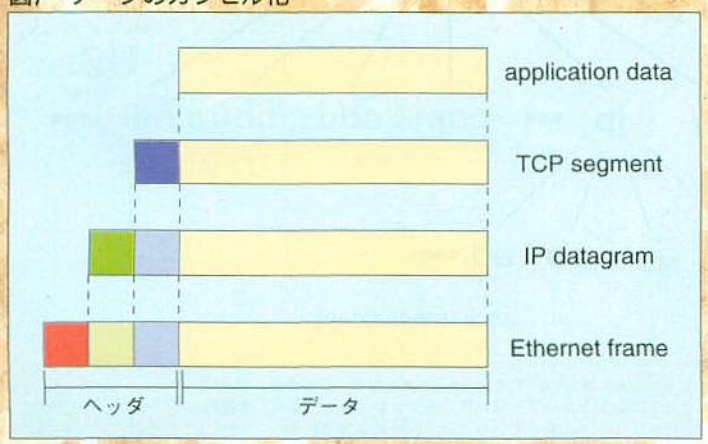
IP層(ICMP層)：IP層では，もはや物理アドレスは使われず，以後の処理はすべてIPアドレスに従って行なわれる．IP層では，主としてIPアドレスの解釈とデータグラムの取り扱いを担当する．IPアドレスにはネットワークアドレスが含まれており，この情報はパケットのルーティングに使われる．このルーティング情報に従ってパケットの当面の送り先を決定するのはIP層の仕事である．また，IP層は上位層から受け取ったデータをIPデータグラムと呼ばれるパケットに変換してパケット単位での転送を制御する．転送制御に関する処理の大部分は，IPのサブモジュールであるICMPプロトコルによって行なわれる．また，IPデータグラムを受け取った場合には，必要なデータグラムを集め，IP層で使用しているヘッダを取り除いて上位層に渡すべきデータを組み立てる．

TCP／UDP層：この層はIP層の上位に位置する．TCPはコネクションベースの通信をプロセスに提供するためのポートなどの管理や信頼性の確保，フロー制御などを行なう．IP層から送られてきたデータを，通信相手から送られたとおりの順序に並べ替えて，正しく上位層に渡すのもTCPの役目である．

socket層：ここでは，ソケットという形でTCP／UDPとアプリケーション(プロセス)とのインターフェイスを提供している．各プロトコルポートに対する操作をファイルシステムとまったく同じように操作できるのは，この層が提供しているインターフェイスを使うからである．

以上の層はUNIXカーネル内部に実装されているため，プロセスはソケットに対する操作だけでネットワークを利用した通信ができるようになっている．そのため，ネットワークを利用したプログラムは，ソケットを通じてデータを送受するだけでネットワークを利用できる．プロトコルの階層化によって，上位層にはネットワークでの通信を抽象化して取り扱うための環境が提供されているのである．

図7　データのカプセル化



プロトコルの階層ごとにデータの前にヘッダを付けるが，このヘッダは異なる階層では意味を持たない．そのため，ヘッダは下の階層に渡されるとデータの一部として扱われる．こうして，ヘッダの階層構造ができあがる．

より高度なネットワークサービス

ここでは，今まで触れられなかったネットワークサービスについて概要を紹介したい．これで，現在のInternet環境について，具体的なイメージが湧くのではないかと思う．

1.ドメインネームサーバ

ドメインとは

Internetはドメインに分割され，ドメインの情報の問い合わせに答えるドメインネームサーバが用意されている．

ドメインとは，Internetを区分する単位であり，図1のようなドメインに分かれている．現在の区分では，国ごとにまず大きく分けられている．ただし，米国だけはほかの国の1レベル下の階層にあたる区分がトップレベルの階層となっている．これは，Internetがもともと米国で発展したものであるため，最初のうちは米国内だけの接続だったためである．しかし，最近では各国でInternetの普及が始まっているため，バランスをとる意味で米国もトップレベルにUSをつけようという動きもあるようだ．

ドメイン名はアドレス指定として使用でき，ユーザーのメールアドレスなどを表記するためにも使われる．たとえば，弊社内のユーザーeditorにメールを送る場合，あて先アドレスはユーザー名と弊社のドメイン名を組み合わせて

editor@ascii.co.jp

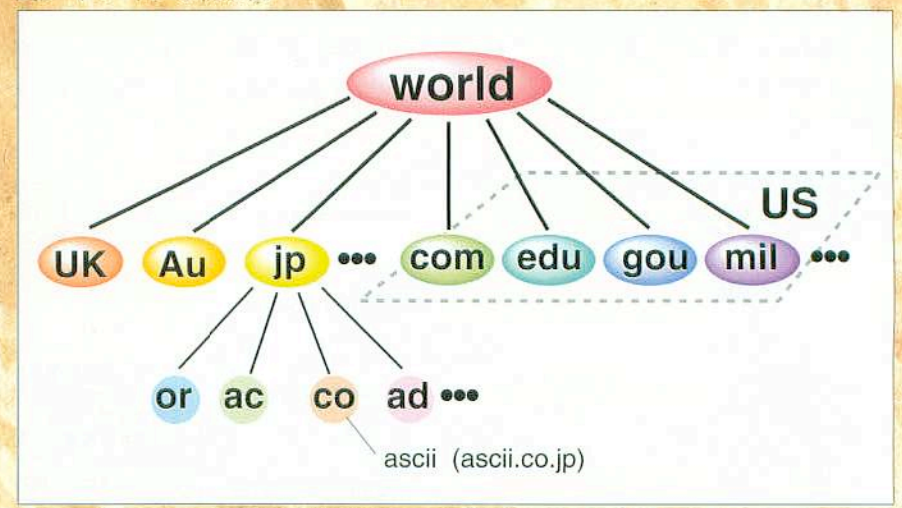
と指定することができる．

この表記では"@"の前がユーザーのログイン名で，後がドメインネームサーバに対する問い合わせのときの検索キーとなる．また，ドメインは"."で区切られる．メールの配送をする場合には，SMTPのところでも触れたが，ホストを目標に配送を行ない，ユーザー名は気にしない．上のアドレスに米国からメールを送る場合を考えると，まずドメイン名がascii.co.jpなので，このドメインに関する情報をドメインネームサーバに問い合わせ，メー

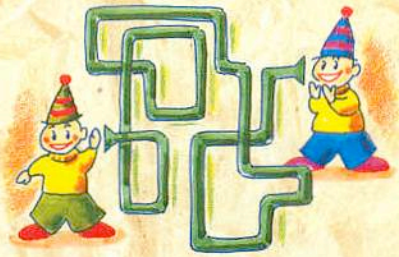
図1　ドメインネームサーバから得られる情報

```
[ascwide.ascii.co.jp]% nslookup
Default Server:  ascii.co.jp
Address:  133.152.32.11

> ascii.co.jp.
Server:  ascii.co.jp
Address:  133.152.32.11

Name:    ascii.co.jp
Address:  133.152.32.11

> set type=mx
> ascii.co.jp.
Server:  ascii.co.jp
Address:  133.152.32.11

ascii.co.jp     preference = 10, mail exchanger = ascwide.ascii.co.jp
ascii.co.jp     preference = 100, mail exchanger = rena.dit.co.jp
ascwide.ascii.co.jp     inet address = 133.152.32.11
rena.dit.co.jp  inet address = 133.156.1.1
> set type=hinfo
> ascii.co.jp.
Server:  ascii.co.jp
Address:  133.152.32.11

ascii.co.jp     CPU=Sun4/330    OS=SunOS4.0.3EXPORT
> exit
[ascwide.ascii.co.jp]%
```

図2　ドメインの構成例



ドメイン名は階層構造になっている．弊社の例では，ascii.co.jpとなる．この場合，ドメイン名を後ろからみていくことで階層を上から下へ降りてくることができる．なお，米国内のドメイン名が日本などと比べて1レベル違っているのは，歴史的事情である．



UNIX
TCP/IP
ネットワーク入門

ルの配送先を決定するのに必要な情報を取得する．この配送先は，各ドメインがメール受け取り用に指定したホストとなる．必ずしもこの配送先となったホスト上にあて先となったユーザーがいるとは限らないが，その場合，その先の配送に関してはドメイン内（サイト内）の責任で行なわれるので，発信元が関知する必要はない．

ドメインネームサーバを利用すると，ホストごとに持っていなくてはならない情報の量を少なくできるうえ，変更に対する対応も数が減る分楽になる．先ほどの例では，メールを最初に発信する米国のマシンでは，ascii.co.jpドメインにメールを配送するための情報を事前に持っている必要はない．問い合わせができるドメインネームサーバを知っていさえすればよいのである．

さらに，ドメインを分けることによって分散管理が可能になるので，分け方を工夫することで異なる管理方針を混在させることもできる．たとえば，企業のネットワークなどでは，機密性の高いデータが流れるネットワークと一般連絡用のネットワークとを分離しておくといった管理方法がよく使われるが，Internet環境でも同様のことが可能になる．実際，日本国内のサブドメインは，ac(大学，教育機関)，or（政府機関），co（一般法人）などというように各ネットワークの運営主体の性格に従って分類されているため，それぞれの方針にあった管理方法をほかと独立して採用することも可能である．

ドメインネームサーバ

ドメインネームサーバは，ドメインごとにホスト名などの情報を持ち，クライアントからの要求に従って情報を提供する．ドメインネームサーバの持っている情報は，さまざまなタイプを指定して検索することができる．中でもA, MX, HINFOの3種類がよく使われる（図1）．

タイプAでドメイン名を指定すると，その組織を代表するホストのIPアドレスを知ることができる．この情報を使えば，組織内に向けて直接データを送ることができる．タイプMXでは，メール配送に必要となるプライマリ／セカンダリのメールハブ(SMTPサーバ)の情報が得られる．図1の例では，ascii.co.jpドメインあてのメールはホストascwide.ascii.co.jpのメールハブに送ればよいとされている．もし，このホストがダウンしていたりして配送ができなければ，セカンダリのハブに配送を依頼することができる．

HINFOはネットワーク管理者向けの情報で，マシンのCPUタイプやOSの種別などを表示する．これは，人間が見て情報を得るためのものである．

2. NFS

NFSとは

NFS (Network File System) は別段新しいサービスではない．むしろ，ネットワーク環境での基本的なサービスとして広く一般的に使われているものである．

NFSはネットワーク環境で透過的なファイルシステムを提供する．NFSを使用すると，ネットワーク上のファイルサーバのディスクを，自分のローカルディスクと同様にマウントして使用できるようになる．そのため，NFSでマウントしたディスクも元々の自分のローカルディスクも，ユーザーからは区別なく使用できる点が特徴である．

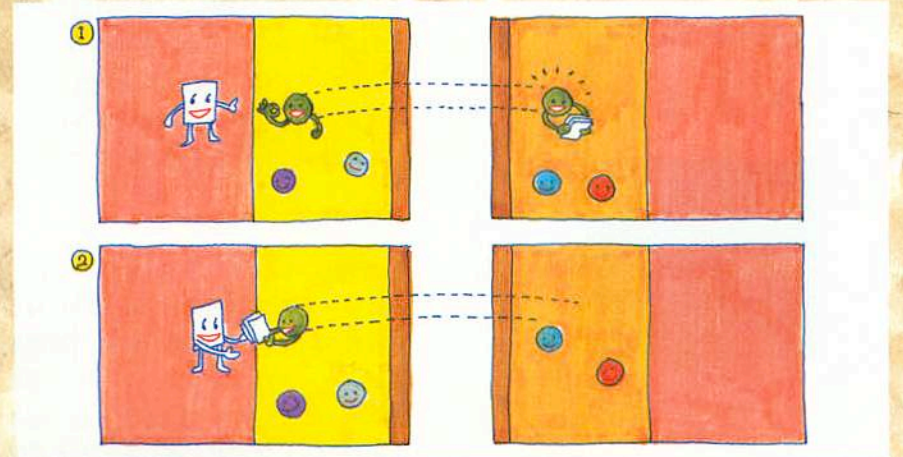
NFSをアプリケーションの観点から考えてみると，ローカルディスクにアクセスするアプリケーションが，まったく同じ方法でNFSでマウントされたディスクにアクセスできなくては価値がない．たとえば，lsコマンドのような基本的なコマンドがNFSに対して使用できる必要がある．NFSでマウントされたファイルシステム上でファイルの一覧を見る場合にnfslsなどというコマンドが用意されるようでは，ユーザーがいちいちNFSマウントされたファイルシステムをローカルディスクと区別しなくてはいけなくなる．

NFSとSunRPC

そのため，NFSにアクセスするために特別な関数やシステムコールを用意することはできない．通常のシステムコールを使ったアプリケーションをサポートするためには，カーネルの内部でNFSでマウントされたディスクを識別し，NFSマウントであった場合には，アプリケーションに意識させることなく必要な処理を実行しなくてはならない．そのための機構が，RPC (Remote Procedure Call) としてカーネル内部に実装されている．

SunRPCは，コネクションレスのプロトコル (UDP) を使い，リモートシステム上の必要なプロシージャを呼び出してくれる．しかし，これだけでは分かりにくいので，NFSの例で具体的に見てみよう．

●RPCのイメージ



NFSマウントされたファイルシステム上のファイルの中身をcatコマンドで表示する場合を考えてみよう．catコマンドは，カーネルに対してopen( )システムコールを使ってファイルをオープンしようとする．しかし，ファイルの実体はリモートのファイルサーバ上にあるため，open( )システムコールもファイルサーバのカーネル内で実行されなくてはならない．データの読み出しを行なうread( )システムコールも同様である．この場合，RPCはカーネルへのシステムコール（プロシージャ）の実行要求を，そのままリモートシステムに伝え，リモートシステム上で同じシステムコールを発行するのとほぼ同じ動作をする．つまり，システムコールの要求がネットワークを越えてリモートシステムに伝達されると考えられる．そして，リモートでの実行結果も当然ネットワーク越しにアプリケーションに返される．この結果，アプリケーションからはシステムコールがどこで実行されたかの情報は隠されてしまい，すべてローカルシステム上で処理されたように見える環境が提供される．

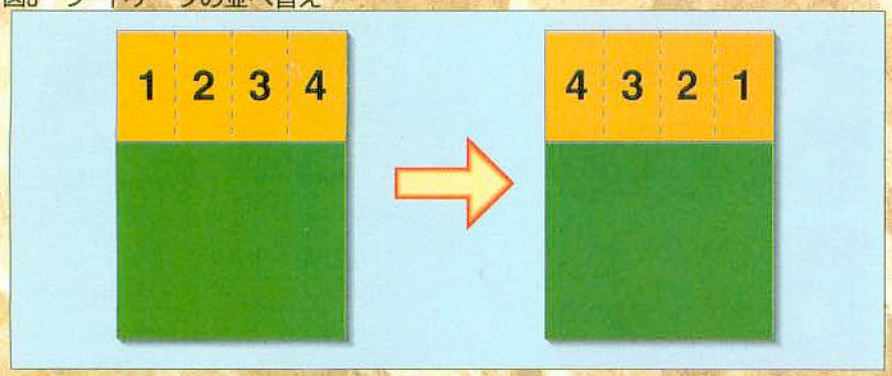
RPCを使った開発環境

システムコールなどの呼び出しにRPCを使用する場合，データ形式をどう表現するかという問題がある．たとえば，システムコールの引数に整数型データ（int）を渡す場合を考えてみよう．通常のシステムコールにintを渡す場合，そのバイト数は当然分かっている．一方，RPCを使用する場合，このintデータをネットワーク経由でリモートシステムに転送することになるが，このとき，intデータは複数バイトからなるバイト列として転送されることになる．データを受け取ったリモートシステムは，このバイト列から元のintデータを再構成しなくてはならないが，マシンのワード構成がlittle endianかbig endianかといった違いがあるため，バイト列の並び順を変更しなければワードデータとして正しく復元できないことがある．そのため，転送されるデータがバイトデータかワードデータかをローカル／リモート双方で合意していなければ正しい転送はできない．

この問題を簡単に解決するために，rpcgenというコマンドが用意されている．rpcgenは，データ型のプロトタイプを受け取ると，そのデータをネットワーク経由で転送するためのCのソースコードを出力する．このCのソースコードをローカル／リモート双方でコンパイルすると，双方共通のデータ型に関する知識を持った入出力ルーチンを作成することができる．これを利用すれば，自作のプログラムでRPCを利用する場合にも，データ型の保存をコーディングする必要はなくなる．

また，実際にこうした型を持つデータ（バイナリデータ）を転送するために，xdrというライブラリルーチンが用意されている．rpcgenが作るソースは，xdrに含まれるルーチンを自動的に呼び出してくれるので正しい転送ができる．

図3　ワードデータの並べ替え



ワードデータをバイト単位で見た並び順はCPUによって異なるため，並べ替えが必要になる．その場合，転送されてきたデータが4つの1byteデータなのか1つの1wordデータなのかが分からなくてはならない．そのため，データの型を双方が知っている必要がある．

3. ネットワークの時間同期

ネットワークの時間管理

ネットワーク環境で，NFSなどを利用して分散処理環境を構築する場合，各マシンごとの時計を合わせる必要が生じる．NFSの場合は，ファイルのタイムスタンプはサーバの時計で設定されるため，クライアント／サーバ間で時計の設定が違っていた場合は現在時刻とタイムスタンプが食い違ってしまうことになる．

時間が食い違っていると，makeなどのように時間に依存する処理を正しく行なうことができなくなることがある．makeは，ファイルのタイムスタンプを比較してファイルの生成順序を調べ，処理を行なっている．そのため，NFSマウントされたファイルシステムのファイルを参照しながらmakeを行なう場合，時間の設定が食い違っていると，ファイルの生成順序の前後関係を誤って認識してしまう可能性がある．そのため，ネットワーク内のマシンがすべて正しい時刻を保持するように揃えるのが望ましいが，これはなかなか容易ではない．

rdateコマンド

SunOSで提供されているrdateコマンドは，ネットワークで時刻を問い合わせるためのコマンドである．コマンド自体は新しいが，使われているプロトコルは時刻を合わせるためのものとしては古くからあったものである．SunOSのrdateは，起動されると指定されたサーバに時刻を問い合わせ，自分の時刻をサーバに合わせる．rdateは，起動時に1度だけ時刻を設定するだけなので，その後のハードウェアクロックの狂いなどにはまったく対応できないが，コマンドを実行するだけで済み，簡便なのでよく使われる．

timed

timedは，LAN（Local Area Network）内の各ホストの時刻を合わせるために考えられたプロトコルである．UNIXの時間管理は，ハードウェアクロックから一定間隔（とされている）で発生する割り込みをカウントして行なっている．たとえば，

UNIX
TCP/IP
ネットワーク入門

割り込みが100分の1秒ごとに発生するハードウェアクロックを使っていた場合には，割り込みを100回カウントするごとに時刻を1秒進めるという形で時間を計測している．

しかし，ハードウェアクロックからの割り込みがLAN内のマシンですべて等間隔で発生しているとは限らないので，場合によってはあるマシンの時刻が少しずつ遅れる／進むということが起こる可能性がある．そこで，timedでは，一定間隔でサーバが時刻をブロードキャストし，クライアントはその時刻に従って自分の時刻を補正する．何度か補正作業を行なうと，通常はサーバの時刻とほぼ正確に同期することができるようになるはずだが，クロックのタイミングが微妙にずれている場合は，毎回予想よりも遅れているなどといった結果になる．timedでは，こうした場合にハードウェアクロック割り込みのカウント(tick)数を増減させることによって，時刻のずれを補正する．具体的には，クロック100カウントごとに1秒時刻を進めるという設定になっていたがどうも少しずつ遅れる，という場合には，クロック99カウントで1秒時刻を進めるように補正を行なうのである．このため，rdateよりは精度の高い時刻合わせが可能になる．

NTP

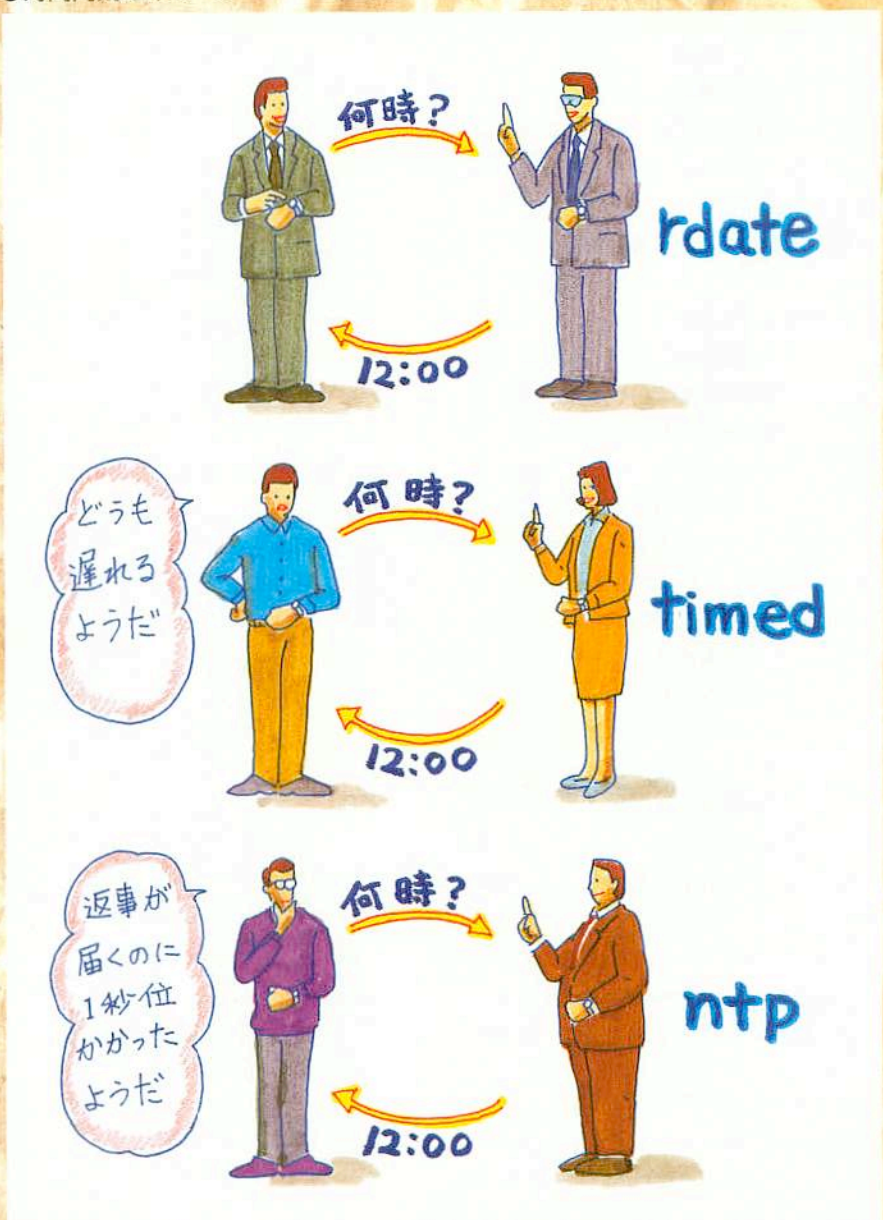
NTP(Network Time Protocol)は，大規模なネットワークで使うための時刻合わせのプロトコルであり，Internet全体の時刻を同期させるような場合を想定している．NTPもサーバに問い合わせて時刻を合わせるが，ネットワークの伝達遅延を利用してサーバが伝えてくる時刻の信頼性を吟味する点に特徴がある．

ネットワークでデータを送る場合，どうしても伝達のための時間がかかる．どの程度の時間がかかるのかは，ホスト間の距離，回線の速度や品質に依存するほか，ネットワークの込み具合いにも影響される．このため，伝達遅延を完全に把握することが不可能である．

伝達遅延を正確に予測することはできないため，NTPではサーバ／クライアント間での応答にかかった時間から伝達遅延を繰り返し計測する．また，その応答で報告された時刻のずれも測定する．

NTPでは，セシウム発信を利用した，時刻決定の基準クロックに同期できるホストが時刻の基準ホストとして利用されることを前提としており，基本的に各ホストの時刻は同期していると考える．そのため，さきほど調べた伝達遅延と時刻のずれが極端に大きいサーバがあった場合，このサーバとの回線が不安定か，またはサーバの時刻が信頼できないと判断し，そこからのデータは捨ててしまう．そして，ほかの信頼できるサーバからの時刻を採用して時刻を同期させる．このような仕様のため，NTPではサーバが複数存在し，さらにサーバ同士で時刻を同期させる機能もある．実際には，基準ホストを中心にサーバが木構造を構成し，上のレベルのホストに時刻を問い合わせて同期しつつ，同一レベルのホスト同士で微調整を図るという構造になっている．

●時間同期のイメージ



パケットの実際

最後に，ネットワーク上のパケットの中身を直接見てみよう．これは，ネットワークプログラムのデバッグなどをする必要がある人は，いつかはやるはめになる作業である．

1. パケットのモニタリング

パケットとは

これまで，ネットワーク上にパケットが流れるなどという表現を使ってきたが，そもそもパケットとはどのようなものなのかということについては何も言わずにきた．そこで，最後にパケットの実体を目に見える形で整理してみよう．

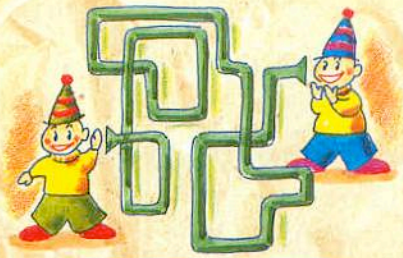
Ethernetは，ご存じのとおり一斉同報媒体である．これは，ブロードキャストを見ても分かるとおり，１つのパケットを複数のマシンで同時に受け取ることができるということである．よく似た例として，電波による通信をイメージしていただきたい．送信者から発信された電波は複数の受信者が同時に受け取ることができるが，発信者は誰が受信したかを知ることはできない．また，受信者相互でも，自分のほかに誰が受信したかを知る方法もない．こうした，電波による通信の性質はEthernetにもそのまま当てはまる．

Ethernetのブロードキャストの場合では，こうした性質が非常に有利に働き，１つのパケットを複数のホストに同時に届けることができた．必要なホストにそれぞれパケットを届けるのに比べ，１つのパケットを受信するホスト全部が同時に受け取るほうが，ネットワーク上を流れるパケットの数も少なくて済み，効率よくネットワークを使用できるといえる．

しかし，逆にいうと，特定のホストだけに届けたいデータを盗聴から守るのは非常に難しいということになる．具体的には，ユーザーがネットワーク上のホストにログインするときに入力するパスワードなども，文字データとしてネットワーク上を流れることになる．こうしたデータの盗聴を防ぐためには，より上位の

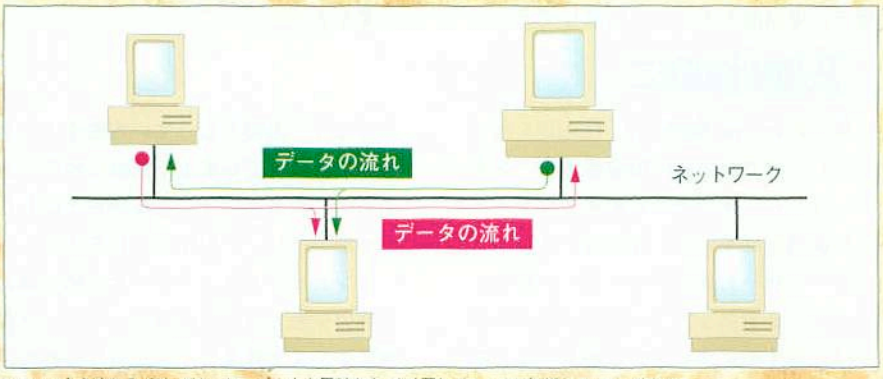
図1　SMTPサーバと通信中のパケットのダンプ

```
 148  tcp mailserver superascii       smtp       1040                  <greeting>
08 00 14 f0 07 63 08 00 20 0b 93 2a 08 00 45 00 .....c.. ..*..E.
00 86 00 ef 00 00 3c 06 71 51 85 98 80 80 85 98 ......<.qQ......
80 81 00 19 04 10 38 4e 10 02 00 0e 70 c2 50 18 ......8N....p.P.
10 00 a9 c2 00 00 32 32 30 20 6d 61 69 6c 73 65 ......220 mailse
72 76 65 72 2e 61 73 63 69 69 2e 63 6f 2e 6a 70 rver.ascii.co.jp
20 53 65 6e 64 6d 61 69 6c 20 53 4d 49 34 2e 31  Sendmail SMI4.1
2f 73 6d 74 70 64 65 6d 6f 2d 31 2e 31 20 72 65 /smtpdemo-1.1 re
61 64 79 20 61 74 20 53 75 6e 2c 20 32 33 20 46 ady at Sun, 23 F
65 62 20 39 32 20 30 33 3a 31 38 3a 30 34 20 4a eb 92 03:18:04 J
53 54 0d 0a                                     ST..

  71  tcp superascii mailserver       1040       smtp                  <helo>
08 00 20 0b 93 2a 08 00 14 f0 07 63 08 00 45 00 .. ..*.....c..E.
00 39 00 ce 00 00 3c 06 71 bf 85 98 80 81 85 98 .9....<.q.......
80 80 04 10 00 19 00 0e 70 c4 38 4e 10 7a 50 18 ........p.8N.zP.
10 00 cd b0 00 00 68 65 6c 6f 20 73 75 70 65 72 ......helo super
61 73 63 69 69 0d 0a                            ascii..

 120  tcp mailserver superascii       smtp       1040
08 00 14 f0 07 63 08 00 20 0b 93 2a 08 00 45 00 .....c.. ..*..E.
00 6a 00 f1 00 00 3c 06 71 6b 85 98 80 80 85 98 .j....<.qk......
80 81 00 19 04 10 38 4e 10 7a 00 0e 70 d5 50 18 ......8N.z..p.P.
10 00 cf 4a 00 00 32 35 30 20 6d 61 69 6c 73 65 ...J..250 mailse
72 76 65 72 2e 61 73 63 69 69 2e 63 6f 2e 6a 70 rver.ascii.co.jp
20 48 65 6c 6c 6f 20 73 75 70 65 72 61 73 63 69  Hello superasci
69 2c 20 70 6c 65 61 73 65 64 20 74 6f 20 6d 65 i, pleased to me
65 74 20 79 6f 75 0d 0a                         et you..

  75  tcp superascii mailserver       1040       smtp                  <mail from>
08 00 20 0b 93 2a 08 00 14 f0 07 63 08 00 45 00 .. ..*.....c..E.
00 3d 00 d0 00 00 3c 06 71 b9 85 98 80 81 85 98 .=....<.q.......
80 80 04 10 00 19 00 0e 70 d5 38 4e 10 bc 50 18 ........p.8N..P.
10 00 4f ee 00 00 6d 61 69 6c 20 66 72 6f 6d 3a ..O...mail from:
20 3c 77 72 69 74 65 72 3e 0d 0a                <writer>..

  81  tcp mailserver superascii       smtp       1040                  <sender ok>
08 00 14 f0 07 63 08 00 20 0b 93 2a 08 00 45 00 .....c.. ..*..E.
00 43 00 f2 00 00 3c 06 71 91 85 98 80 80 85 98 .C....<.q.......
80 81 00 19 04 10 38 4e 10 bc 00 0e 70 ea 50 18 ......8N....p.P.
10 00 c4 72 00 00 32 35 30 20 3c 77 72 69 74 65 ...r..250 <write
72 3e 2e 2e 2e 20 53 65 6e 64 65 72 20 6f 6b 0d r>... Sender ok.
0a                                              .

  73  tcp superascii mailserver       1040       smtp                  <rcpt to>
08 00 20 0b 93 2a 08 00 14 f0 07 63 08 00 45 00 .. ..*.....c..E.
00 3b 00 d2 00 00 3c 06 71 b9 85 98 80 81 85 98 .;....<.q.......
80 80 04 10 00 19 00 0e 70 ea 38 4e 10 d7 50 18 ........p.8N..P.
10 00 bc 25 00 00 72 63 70 74 20 74 6f 3a 20 3c ...%..rcpt to: <
65 64 69 74 6f 72 3e 0d 0a                      editor>..

  84  tcp mailserver superascii       smtp       1040                  <recipient ok>
08 00 14 f0 07 63 08 00 20 0b 93 2a 08 00 45 00 .....c.. ..*..E.
00 46 00 f3 00 00 3c 06 71 8d 85 98 80 80 85 98 .F....<.q.......
80 81 00 19 04 10 38 4e 10 d7 00 0e 70 fd 50 18 ......8N....p.P.
10 00 12 c6 00 00 32 35 30 20 3c 65 64 69 74 6f ......250 <edito
72 3e 2e 2e 2e 20 52 65 63 69 70 69 65 6e 74 20 r>... Recipient
6f 6b 0d 0a                                     ok..

  60  tcp mailserver superascii       smtp       1040                  <data>
08 00 14 f0 07 63 08 00 20 0b 93 2a 08 00 45 00 .....c.. ..*..E.
00 28 00 f4 00 00 3c 06 71 aa 85 98 80 80 85 98 .(....<.q.......
80 81 00 19 04 10 38 4e 10 f5 00 0e 71 03 50 10 ......8N....q.P.
10 00 d5 24 00 00 64 61 74 61 0d 0a             ...$..data..
```



UNIX
TCP/IP
ネットワーク入門

図2　パケットの傍受



Ethernet上を流れるパケットは，ケーブル上を電波とまったく同じイメージで伝播していく．そのため，通信中のデータを第三者が傍受することも技術的には可能である．もっとも，こうした作業にはセキュリティ上の問題が存在することはいうまでもないが．

レイヤーで暗号化などを行ない，セキュリティを確保するなどといった対策も考えられている．弊誌1992年２月号の記事「INSIDE SOFTWARE」で紹介したネットワークユーザー認証システムKerberosも，そうした対策の１つである．

一方，特定の２つのホスト間で通信中のパケットを第三者が傍受しても通信の当事者には知る術がなく，何の影響もないということを利用すると，パケットを傍受して内容を調べることでネットワークプログラムのデバッグや障害の発見を行なうことができるということにもなる．ネットワークの不具合には，“ちょっと状況が変わると出なくなる”という対応が難しいものもあるが，パケットを傍受しても通信状態には何の影響も与えないのであれば，これは通信状態のモニタリングの手段としては非常に強力なものになりえる．

以前にtelnetによるSMTPサーバとの通信を例にあげたが，図１はその通信中にやりとりされたパケットを，ネットワーク上の別のホストで傍受し，データをダンプしたものである．ただし，ここではTCPのACKパケットは割愛し，文字データ（プロトコルに従った応答）をやりとりしている部分だけを取り出してあるので，データ部分をよく見ていただきたい．どこかで見たような文字列が含まれていることがお分かりいただけるだろう．ダンプ部分の先頭には，あて先のアドレスと送信元のアドレスが，それぞれ48bitのEthernetアドレスで書かれている．それに続いて，IPヘッダ，TCPヘッダなどが順次書かれており，最後にSMTPプロトコルの文字列が書かれている．さらに，このヘッダの部分を詳細に調べると，各レイヤーごとに付けられるヘッダの構造についても具体的に理解できるだろう．

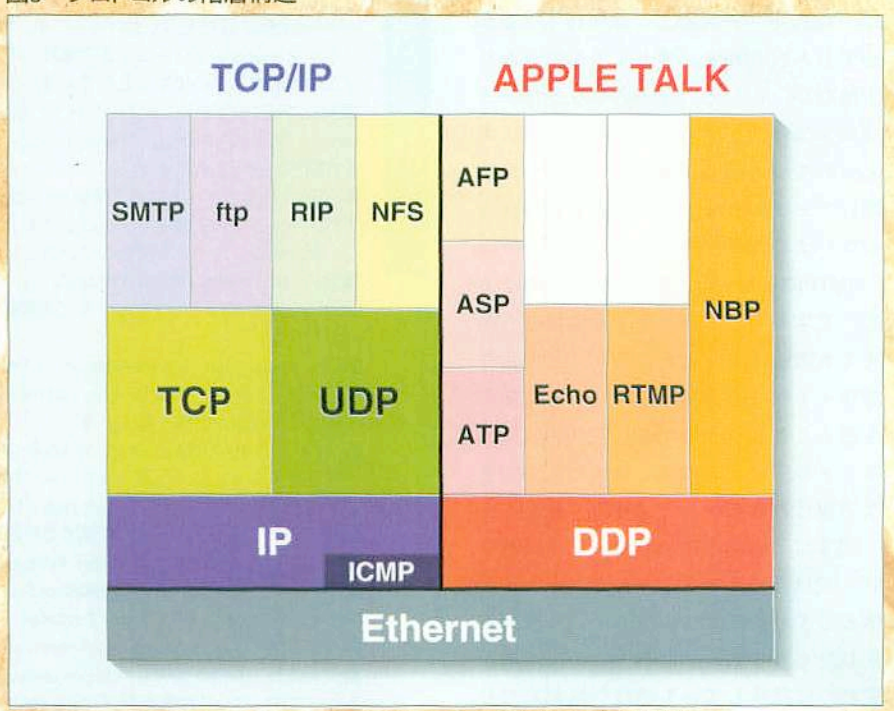
SMTPのプロトコルが，asciiキャラクタによるやりとりをベースとしていることが幸いして，やりとりの様子を直接目で見ることができるので，もしプロトコルを知らなくてもこれで詳細を知ることができる．先のtelnetを使ったポートとの接続方法を知っていれば，試行錯誤でプロトコルを解析することさえできる．

プロトコルの階層構造

次に，プロトコルの階層構造について整理してみたい．今まで，いくつかプロトコルを取りあげ，プロトコルに従ったやりとりの例を紹介してきた．たとえば，メール配送のためのSMTP，ファイル転送のためのftp，時間同期のためのntpなどである．一方，ここでの話はTCP/IPネットワークに限定しているわけだが，ここでいうTCP/IPも実はIP，TCP，UDPなどといったプロトコルのことを指している．これほどたくさんのプロトコルが一度に出てくると“結局プロトコルとは何なのか”という疑問もあるかもしれない．

プロトコルとは，通信規約のことであるとよくいわれる．通信規約とは，通信を行なう当事者同士の合意ということであり，どうやって通信を始めてどうやって終わるかという基本的な部分から，デ

図3　プロトコルの階層構造



プロトコルの階層構造を簡素化して示した．ここで，物理層のEthernetとTCP/IPのプロトコルは比較的独立しているので，Ethernet以外の物理層（FDDIなど）でTCP/IPを使うことも，AppleTalkなど異なるプロトコルをEthernetで動作させることも可能である．

ータをいつどのような形式で送るかということまで含まれる．

TCP/IPのネットワークでは，基本となるのはなんといってもIPであり，このプロトコルによって，Internet環境に対応できるパケット交換の機能が規定されている．だが，IP単独では転送の順序や信頼性は保証されていないため，上位プロトコルとしてTCPやUDPが規定されている．この，IPとTCP，UDPを基本的なプロトコルとして用意しているのが，今日のTCP/IPをベースとしたネットワーク環境であるということになる．しかし，これだけでは基本的なデータ転送の方法が規定されただけであり，個々のアプリケーションで必要となるような詳細な処理については何も決まっていない．そのため，データはやりとりすることができるが，そのデータをどう解釈したらよいのかについては分からないということになる．メールを転送するのであれば，メールの転送をサーバに依頼しなくてはならないが，どうやったらサーバに依頼の意思表示をすることができるのだろうか．そこを規定するのがアプリケーションレベルのプロトコルである．

アプリケーションレベルのプロトコルは，TCP/IPベースの上に作られているが，必ずしもTCP/IPを必要とするものばかりではない．むしろ，ほかのネットワークプロトコルの上でも利用できるように考えられている場合が多く，そのために実際にデータ転送をする部分の実装に依存しないような形で作られている．

SMTPのプロトコルを見ても分かるように，文字列でコマンドを送るのはどう考えても無駄なように思えるが，どのようなネットワークを利用していても文字列を送ることはできるので，TCP/IP以外のネットワークにも非常に簡単に対応できて汎用性があるというメリットがある．

図2は，TCP/IPでのプロトコル階層と，TCP/IP以外のネットワークプロトコルの例としてAppleTalkでのプロトコル階層を示したものである．プロトコルの階層が変われば提供している機能も当然変わり，上の階層になればなるほど，目で見て分かるほどの具体的な処理を規定するようになる．そのため，プロトコルを考える場合には階層をきちんと意識しておかないと混乱のもとになるので，注意を払う必要がある．

2. 終わりに

現在のInternet環境では，TCP/IPをベースとしてプロトコルの階層構造ができあがっており，その上でさまざまなサービスが実現されている．今回は，まずネットワークサービスのプロトコルを操作してみることでネットワークサービスのサーバに対するアクセスがどのようなものなのかを確認した後，アプリケーションからネットワークにアクセスするためにカーネルが提供しているインターフェイスとして，ソケットについてみてきた．

今回の特集は，ネットワーク環境での基本的な知識をまとめたものである．すでにネットワーク環境を利用している方にとってはあまりに当たり前のことだったかもしれない．しかし，Internet環境は現在では特別なものではなく普通に利用されているものだとはいえ，実際にプログラミングレベルでネットワークにアクセスしているユーザーはまだまだそれほど多くはないように思える．そこで，ネットワーク環境といっても別段特別なことはなく，基礎となっている概念さえきちんと理解してしまえば簡単に利用できるものであるということを明らかにしたかったのだが，いかがだったろうか．

コンピュータネットワークの持つ強力な能力を考えると，今後ネットワーク環境はますます普及し，ごく当たり前の環境となることが予想される．今はまだ"ネットワーク環境で何ができるのか"が重大問題扱いされているが，コンピュータがカラーディスプレイを備えているのと同様，ネットワークも"あって当たり前"と考えられる日が1日も早くやってきてほしいと思う．特に，プログラマの方にはネットワーク環境に親しんでいただき，有用なアプリケーションを多く世に送り出していただきたいと思う．

解決すべき技術的問題もまだたくさんあるが，広大なInternetの世界と孤立したコンピュータを比べてみれば，コンピュータが本来の能力を発揮するにはどちらの環境がよいかはもう明らかだろう．

●参考文献

Internet全般の話題を広くカバーし，IP，TCP，UDPなどの基本プロトコルからSMTPやftpなどのアプリケーションプロトコルまで網羅している．じつは，今回の特集で扱ったテーマのほとんどはこの本の中に見い出すことができる．

■INTERNETWORKING WITH TCP/IP Volume 1;Principles, Protocols, and Architecture Second Edition, Comer, Prentice-Hall, 1991
日本語訳も出版されている．

■TCP/IPによるネットワーク構築 第2版 Vol.1，村井純・楠本博之訳，bit別冊，1991
また，ネットワーク環境でのプログラミングの優れた教科書もある．ソースコードを豊富に掲載し，具体的なノウハウを解説している．

■UNIX NETWORK PROGRAMMING, W. RICHARD STEVENS, Prentice-Hall, 1990
UNIXカーネル内部でのソケットなどの実装の詳細について知りたいのであれば，以下の書籍が最適であろう．

■The Design and Implementation of the 4.3BSD UNIX Operating System, Leffler, McKusick, Karels, Quarterman, Addison-Wesley, 1989
本書も日本語訳が出版されている

■UNIX 4.3BSDの設計と実装，中村明・相田仁・計宇生・小池汎平共訳，丸善，1991
アプリケーションレベルのプロトコルに関しては，RFCにあたるのが最も確実だろう．RFCは，インターネットコミュニティの研究報告といった感じのオンラインドキュメントの集合である．今回の特集で扱ったプロトコルに関する記載があるものについて，以下に紹介する．

■RFC821 Simple Mail Transfer Protocol
■RFC854 Telnet Protocol Specification
■RFC959 The File Transfer Protocol
■RFC1034 Domain names -Concepts and Facilities-
■RFC1035 Domain names -Implementation and Specification-
また，NTPについては弊社刊「UNIX MAGAZINE」1992年2月号に詳細な解説記事がある．

■UNIX Communication Notes 44 NTP，山口英，UNIX MAGAZINE，1992年