計算機科学基礎'11 ＃5 – ネットワークサービスとセキュリティ

久野　靖*

2011.5.18

1　情報セキュリティ

1.1　セキュリティとは/情報セキュリティとは

セキュリティ(安全性) とはその名前の通り、安全が保たれるようにすること、たとえば身体的危険や財産の窃盗その他の危険にさらされないようにすることを言います。一般には、家屋にきちんと施錠したり、ガードマンに見回ってもらったり、財産なら安全な場所に預けるなどの対策があります。

では**情報セキュリティ**はどうでしょうか。つまり「情報に関する安全性」ですが、それは一般のセキュリティとどう違うのでしょうか。情報社会の今日では「情報」自体に価値が置かれるようになっていますから、一般の財産と同じに考えればいいでしょうか。たしかに違わない面もありますが、次のような点は違っています:

- コピーが容易であり、盗まれたかどうか分かりづらい。
- 同様に、書き換えも容易であり、変更されても分かりづらい。
- ネットワーク経由でのセキュリティ侵害が容易かつ一般的。

とくに最後の点は、今日の急速なネットワーク普及に社会が追い付いていない面があり、大きな問題です (なのでネットワークの回に取り上げているわけです)。

では、情報セキュリティ対策を行うとして、その目標は何であるべきでしょうか? 一般的には、次の 3 目標が挙げられます (図 1):

- **機密性** — 情報が、取得できるべきでない対象に漏洩しないこと。
- **完全性** — 情報が、完全な状態である、つまり改ざんされないこと。
- **可用性** — 情報が、必要なとき利用可能、つまりシステムがきちんと動作すること。

これらを見てみると、情報セキュリティ対策をきちんと行うことは、その多くの部分が人的な問題、管理上の問題だと分かります。

たとえば機密性について言えば、ある情報が「誰は」見てもよくて「誰は」見ては困るか、という範囲が明確でなければ適正な管理のしようがありません。

完全性について言えば、ある情報をその権限を持つ人が業務の必要に応じて書き換えることは当り前というか必要なことですが、権限がない人が書き換えたり、権限がある人でも業務上の必然性がないのに不正に書き換えるのは問題です。実際、企業等におけるセキュリティ侵害の大多数は外部の人間ではなく内部の人間によるという調査結果があります。

可用性についても、いくらネットワーク経由の攻撃に対して防御していても、サーバ室に誰かが入ってきて電源を落としたり装置を破壊したらおしまいなわけです。地震や火事など自然災害への対

*経営システム科学専攻

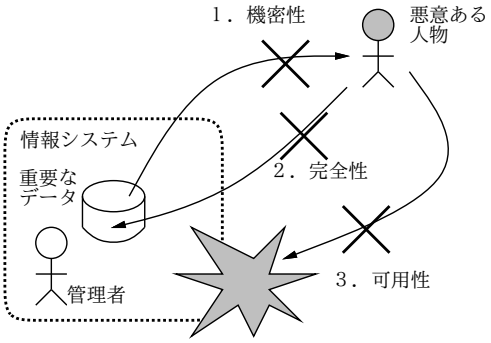

図 1: 情報セキュリティの 3 要素

策も重要です。なお、可用性に対する対策としては、システムを多重化してどちらかが壊れても運用を続行可能とする、重要なデータは定期的にバックアップして安全な場所に保管する、などがあります (このバックアップが流出すると機密性が損なわれるので注意)。

こうして見ると、組織の情報セキュリティ対策の責任を担うべきなのは、必ずしも情報システム部門だとは言えません。情報システム部門は技術的な対策には責任があるでしょうけれど、情報の取り扱い規定、機密度の分類、管理体制などはより上位の管理部門の仕事であるべきではないでしょうか。

この点については一応釘を挿しましたので、以下では技術的な話題に進みます。

1.2 不正アクセス

不正アクセスとは、大まかに言うと (1) 他人の ID/パスワードを使ったり、(2) ソフトの欠陥などを突き通常のアクセス制限を回避して、システムを利用するような行為を言います。このような行為は、**不正アクセス禁止法** (1999 成立) によって禁止されています (この成立年代の新しさが、まだ情報セキュリティに対する法制度的取り組みが始まって間もないことを示していますね)。

では、世の中にはどのような形の不正アクセスが多いのでしょうか? 特定の企業等から秘密を盗み出すためというものも無くはないでしょうが、大抵は「イタズラ」と「個人情報収集目的」です。逆に言えば自分のマシンに大したデータも入れていないので侵入されることはないだろう、と思うのは間違いなわけです。侵入されても構わない? 侵入者はマシンを乗っ取ると、次にそのマシンを踏み台に他のマシンにイタズラを仕掛けます。ですから、侵入されたものをほっておくと「お前のマシンからうちに攻撃が来ているぞ! 何とかしろ!」という警告をもらってしまうかも知れません。

次に、そういう侵入を行うのは高度な技術が必要かということですが、パスワード推測やセキュリティホールの活用によって「自動的に」他のサイトへの侵入を試みるプログラムが流通していて、これらを使うと自分では高度な技術を持っていなくても侵入が可能になります (そういうものを使う輩を Script Kiddle と呼びます)。

では、不正アクセスの被害に逢わないためにはどうするべきでしょうか? まず、本当に技術力のある侵入者に狙われた場合、完全に不正アクセスから逃れるのは難しいです。そのような場合は警察に相談するとともに、対策についてはその道の専門家をたのむべきでしょう。

実際には多くのケースは上述の Script Kiddle などの「いたずら」ですから、その場合は「あけっぱなし」「不注意」をなくし、侵入への「敷居を高く」することが有効です。あちらはどこでもいいのですから、ラクにイタズラできなければよそを試しに行くでしょう。そのための対策としては、次のようなものがあります (図 2):

- パスワード対策 — パスワードとユーザ ID をともに防御する (たとえばユーザ ID の一覧などもできるだけ外部には公開しない)。パスワードは推測されにくいものとし、定期的に変更させる。

- セキュリティホール対策 — 使用している OS やアプリケーションのセキュリティ情報には気を配り、問題があればソフトを更新してそれを解消する。Windows なら Microsoft Update、Mac ならソフトウェアアップデートなどを動かし、重要度の高い修正は必ず適用する。
- 防火壁、パケットフィルタ — 外部ネットワークと内部マシンの間に防火壁 (ファイアウォール) を入れ、そこで「不要な」(提供するサービスやその管理のため以外の) ネットワーク接続を禁止する。

Web サービスなどの場合はさらに、サービス構築時に注意すべき点が多くありますが、ここでは触れません。

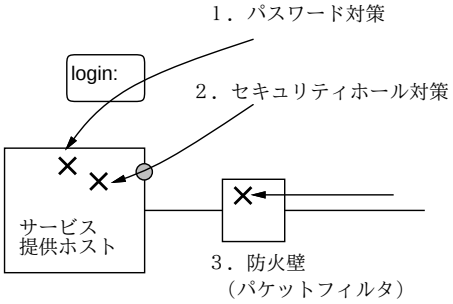

図 2: 代表的なセキュリティ対策

重要なのは、防火壁「さえ」あれば済むというわけではない、ということです。なぜなら、サービスの提供やその管理に必要な通信は防火壁を通過させる必要があり、これらの通信を経由した攻撃も必ず存在するからです (それ以前に防火壁をきちんと設定していないという論外な状態も実在するようですが)。さらに、内部者側の問題 (悪意ある内部者や、ウィルスへの感染など) にはまったく役に立ちません。

ですから、セキュリティのためには特定の 1 つの対策で十分ということはなく、さまざまな面から総合的に侵入への敷居を高くするべきだ、ということです。たとえば、図 2 では 3 つの攻撃が並列に書かれていますが、実際にはパスワードの不備や OS のセキュリティホールがあっても防火壁でこれらの不備を突く通信を遮断できれば被害を食い止められる、という関係があるわけです。

なお、敷居を高くすると、正規のユーザにとっても不便になることがあるため、ユーザが不満を持って「穴」をあけようとしたりすることがあります。このような問題については結局上で述べた「管理方針」の問題で、それがユーザにもきちんと納得されていなければうまく行かない、ということになるのではないでしょうか。

1.3 マルウェア (俗にいうウィルス)

Winny を通じて広まるウィルスによる相次ぐ情報流出事件からも分かるように、コンピュータウィルス (以下単に**ウィルス**) は今日の情報セキュリティ上の脅威として無視できないものとなっています。ではウィルスとは何でしょうか?

実は、ウィルスと言う場合、広義の意味と狭義の意味とがあります。広義のウィルスは狭義のウィルスとの区別のため**マリシャスコード** (Malicious Code、悪意のあるコード)、**マルウェア** (malware) と呼ばれることもあり、次のような性質を持つプログラム全般を言います。

- 何らかの手段で個人の PC などにとりつき、所有者の意図しないことを行うようなプログラム。

そして、広義のウィルスの中身はたとえば次のように分類できます (網羅的とは限りません):

- 狭義のウィルス — プログラム、文書などのファイルの中に自分自身を埋め込み、そのファイルを開くことで他のファイルに感染し増殖するようなもの。
- **ワーム** — 単独のプログラムであり、ネットワークを通じて自分自身の複製を他のホストに送り込んで増殖しようとするもの。システムのセキュリティホールを突いて増殖するものと、メールなどで自分自身を他のユーザ宛に送り込み、そのユーザをだまして自分自身を起動させようとするものとがある。
- **トロイの木馬** — 一見有用なプログラムのふりをして配布されており、起動するとそのように動作するが、その裏で悪事をはたらくようなもの。

悪事の内容としては、単に画面にイタズラを表示させる程度のものから、ファイルを勝手に削除する、マシンを起動できなくする、マシンに裏口 (ウィルスを放った人がこっそり侵入できる手段) を設置する、利用者の打鍵をこっそり記録する、などがあります。

ワームでは他ホストへの感染のために大量のネットワークトラフィックを生成して迷惑となるものもありますし、勝手に大量のメールを他人あてに送るものもあります。この場合、感染したマシンのアドレス帳から宛先を取り出して送信するので、知合いに迷惑が掛かる (あなたもウィルスつきメールを受け取ったことがあると思います。しかも From:は別人に偽装されているので見た目の送信者に文句を言っても見当違いとなるわけです)。

近年とくに問題になっているのは、マシン内の情報を外部に送信するようなもので、多くの情報流出事件を引き起こしています。この流出手段も、Winny などのファイル交換機能によるもの、メールで送出するもの、勝手に Web サーバになって送出するものなどさまざまです。

では、ウィルスに対する防御はどうしたらよいのでしょうか。いわゆるウィルス対策ソフトは当然使用するのがよいですが、その基本的な動作原理は、ファイルが書き込まれる時などにその内容を調査して、**パターンファイル**に含まれる既知のウィルスのパターンと一致しないかどうか調べ、怪しければ書き込みを止めるなどです。

ということは、パターンファイルを新しいものに絶えず更新して行かないと、新たなウィルスには効果がないわけです。また、ファイルへの書き込みを行わないワーム (メモリ上でだけ自分の複製を作る) ではパターンファイルによる検出ができないという弱点もあります。

なお、Winny の漏出事件のときは、Winny が主に日本で使われているファイル交換ソフトだったためパターン更新が遅れたことが問題だったという説があります (さらに Winny の交換機能によるウィルスの拡散速度が速かったという面もあるでしょう)。

多くのウィルスはユーザが間違って実行してしまうことにより感染しますから、もっとも基本的な防御手段は、要するに「未知のプログラムは実行しない」ということに尽きます。しかし、Winny で広まったウィルスの場合、自分自身のアイコンを「画像」「フォルダ」「テキスト」などの無害そうなもののアイコンにすり替え、名前もそれらしいものにしておくことでユーザが「うっかり」開いて (実行させて) しまうようになっていますから、十分な慎重さが必要なわけです。[1]

万一、ウィルスに感染した場合は、他のホストへの感染を防ぐためにネットワークから切り離し、駆除を試みますが、駆除できない場合はシステムを初期状態から入れ直すことになります。このような場合のためにも、重要なデータは常にバックアップを取る必要があるわけです。

1.4 暗号技術とセキュリティ対策への応用

暗号技術は昔から、情報の機密性を保つために使われて来ました。そのことは、コンピュータ時代の今日でも変わりません。もちろん、計算機による高速な計算のため、使われる暗号方式はずっと高

[1] これから防御するには、すべてのファイルの拡張子をきちんと表示させること、長い名前で省略されているようならきちんとチェックすること、ネットワークから来たものを入れる場所はセキュリティレベルを上げて (Windows なら「インターネット」ゾーンに設定) 十分チェックさせること (こうするとプログラムの実行は常に警告が出る)、などがあります。

度なものになっています。しかし、暗号を破ろうとする側も同じ高速な計算が使えるわけですから、暗号を作る側と破ろうとする側のせめぎ合いは昔と変わっていないとも言えます。

どんな暗号であっても、十分な時間を掛ければ…すべての鍵の可能性をしらみ潰しに試すことで、破ることができます。ただしそれには、太陽が燃え尽きてなくなるまで掛かるだろうということで、実質破られないだろうとして扱うわけです。しかしこれには、計算機が高速になって試す時間が短くなってしまったり、また暗号方式に弱点があって全部試さなくても部分的に試せば済んでしまう (従って短時間で破られてしまう) 可能性がある、という問題があります。

暗号には大きく分けて、次の 2 種類があります:

- **対称鍵暗号** — 暗号化と復号化に同じ鍵を用いる。
- **公開鍵暗号** — 2 つの鍵の対を使い、片方で暗号化、他方で復号化を行う。

歴史的には対称鍵暗号の方が古くからありますが、これには「鍵をどうやって安全に伝達するか」という問題がつきまといます。ネットワークで送ればいいと思うかも知れませんが、送っているところで盗聴されたらその後の暗号化通信も解読されてしまいます。フロッピーディスクなどに入れて会って手渡たせば安全ですが、面倒すぎますね。

そこで考案されたのが公開鍵暗号で、こちらの場合は鍵を**秘密鍵**と**公開鍵**の対で生成し、公開鍵を皆に広く知らせてしまいます。暗号を使って通信したい人はこの公開鍵を使って通信を暗号化して送り、受け手は自分だけが持つ秘密鍵で解読します。他の人は秘密鍵を知らないので、他人には解読できないわけです (図 3)。

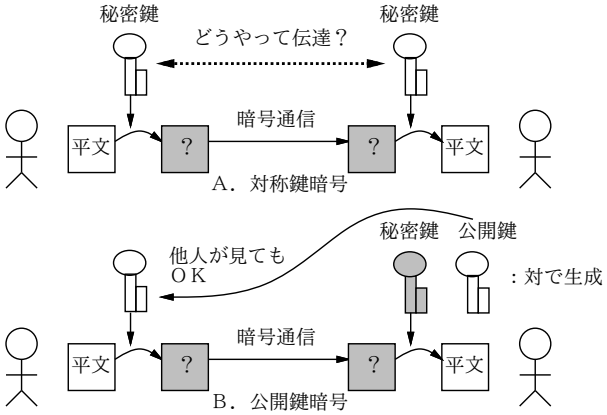

図 3: 対称鍵暗号と公開鍵暗号

また、公開鍵暗号の別の用途として、持ち主が持つ秘密鍵によってある情報に印をつけ、対応する公開鍵を用いるてそれが OK であることを検証する (偽物の秘密鍵だと NG となる) ことにより、情報を発信したのが確かに秘密鍵を持つ本人だということを証明する、というものもあります。こちらは書類にサインするのと似ていることから、**電子署名**と言います。

ところで、公開鍵をネット経由で送ることに危険はないとしても、受け取った公開鍵が確かに本人のものかどうかを確認するという問題がまだ残っています。公開鍵は長い数値ですが、それをもとに生成した**フィンガープリント**という 20〜40 桁くらいの数値があり、それを電話などで読み上げて照合することで間違いがないかどうかをチェックすることができます。しかしこれはこれで大変ですね…

そこで**PKI**(Publick Key Infrastructure) という枠組みが用意されました。これは、**CA**(Certification Authority) という組織が他の組織や個人の公開鍵を「正しいと保証」するという制度で、保証してもらいたい側は CA に自分の出自を示す書類を送るなどして確認してもらいます (もちろん、CA も商売なのでお金が掛かります)。では、CA 自体の正しさは…それは、その CA を保証する「親 CA」があり、そのまた親…とつながって行き、最後は「根元」の CA(**ルート CA**) にたどりつきます。ルー

トCAはそう沢山はないので、あらかじめ確認しておくなどの手段が取れます。なお、CAの仕事は「当該組織が存在していること」を確認し保証することまでであり、その組織の業務内容や社会的信頼性を調査して保証するわけではない点は注意しておいてください。

たとえば、ブラウザで暗号化されたデータ転送を行う **SSL** プロトコルでは、公開鍵暗号を使用し、このPKIを用いたチェックを組み込んでいます。そのため、代表的なルートCAについて、ブラウザを配布する時に最初からその公開鍵情報がブラウザ内に組み込まれています。ブラウザでSSLのページ (`https://`で始まるページ) を開いた時は、このページの証明書情報からルートCAまでの連鎖をたどれるかどうかをブラウザが自動的にチェックしています (図4)。

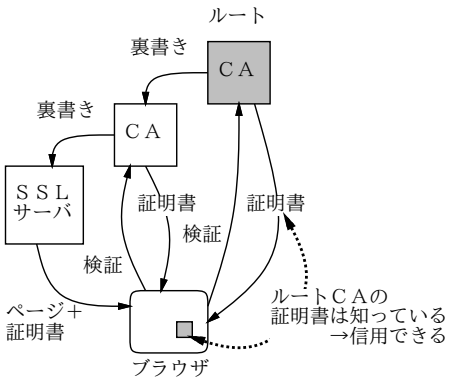

図 4: PKIとCAの連鎖

ところが、CAにお金を払うのが惜しいとか、海外のCAに行政機関が依存できないだろうとか (そうなんでしょうかね?) の理由で、このような連鎖に加わらずにSSLのページを動かしているサイトもあります。このようなサイトの証明書を (勝手に自分でいいだろと言っているというニュアンスで)「**オレオレ証明書**」と呼びます。そのようなサイトを表示しようとすると、ブラウザは「ルートCAへの連鎖がたどれません」という警告を出し、次の選択肢を提示します:

(1) このサイトを恒久的に信用して今後警告なしで表示する。

(2) とりあえず今回だけ信用して表示する。

(3) 表示とりやめ。

このとき、基本は (3) ですが、どうしても見る用事があるなら「そのサイトの内容は信用できないかも知れない」という覚悟の上で (2) で見ることもありでしょう。決して (1) は選ばないようにしてください。

ところが、このようなサイトでは「SSLでは暗号化はできているのだから、別に構わないので (1) を選んでください」とか「自分のところをルートCAに追加してください」とか呼び掛けていることが多いようです。いくら暗号化ができていても、見ているページそのものをどこが公開しているかの身元保証がないわけなので、決して従わないようにしましょう (行政機関などがこのようにせっかくの認証技術の価値を否定させるような行動を取るのも問題だと思われます)。

最後に、普段のシステムへのログインなどパスワードに基づく**個人認証技術**も暗号技術が基本になっていることを注意しておきます (もちろん、指紋や静脈などの生体認証技術は別です)。ですから、長く同じパスワードを使い続けるのは望ましくないわけです。

2 ネットワークサービスの構成

ネットワークを経由して通信を行う場合、前章で見たように、最終的には2つ以上のプロセスどうしがソケットなどの機能を使ってデータをやりとりすることになります。このとき、(前章の`send.c`/`recv.c`

でやったように) あるユーザが 2 つのプロセスをうまく噛み合わせて起動する、というのは現実にはありそうもないことです。なぜなら、ユーザは 1 つのマシンの上にいて、もう 1 つの遠隔地にあるマシンとやりとりするためにネットワークを使うわけだからです。ではどうしたらよいのでしょう?

この問題に対する回答は、プログラムを次の 2 種類に分けることです:

- サーバ — サービスを提供するマシンで常時稼働していて、サービス要求があるまで待ち、要求があったらサービスを提供する。
- クライアント — サーバに要求を出して、そのサービスを受ける。

このような方式を**クライアントサーバ方式**と呼びます (図 5)。クライアントサーバ方式では、上述の「噛み合わせ」の問題が自然な形で解決できますし、サービス提供のために必要な資源 (データ等) はサーバのところで一括管理できるので、各種サービスの実現が比較的簡単に行えます。このため、クライアントサーバ方式はネットワークの初期から今日に至るまでネットワークアプリケーションの構成方式として広く使われています。

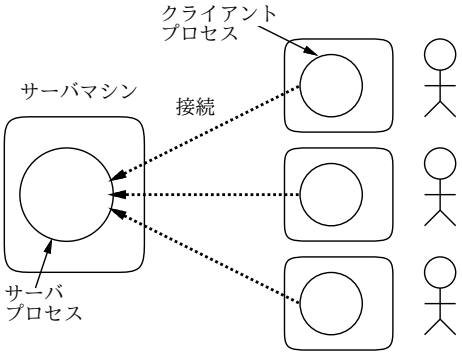

図 5: クライアントサーバ方式

なお、クライアントは実際には 1 つではなく、そのサービスを利用するユーザの数に応じて多数動いているのが普通です。つまり、ユーザがネットワーク上のサービスを使おうとしたときは、クライアントに相当するプログラムを起動し、このプログラムがサーバと接続してサービスのためのやりとりを行うわけです。言い替えれば、クライアトサーバ方式のシステムは、クライアントはそれぞれのユーザがいる多数のマシンで動き、サーバはサービスを提供するための専用マシンで 1 つだけ動く、という非対称的な構造を持ちます。

Unix では伝統的に、サーバのことを**デーモン** (daemon、守護神の意味) と呼びます。先にプロセスを観察したときに `nfds` とか `ntpd` とか最後に「`d`」のつくプロセスが多数走っていたのをご記憶でしょうか。これらが「なんとかデーモン」、つまりネットワークサービスのために待機しているサーバプロセスだったわけです。

なお、ネットワークサービスはとても多数あるため、それぞれのサーバを常時動かしていると待機プロセスばかりが多くなりすぎることがあります。このため、Unix では **inetd**(Internet Daemon) と呼ばれるサーバが稼働していて、専用のサーバが動いていないサービスの要求を「代理で」待ち受けてくれています。そして実際に要求が到着すると、`inetd` は「本物の」サーバを起動してサービスを行わせるわけです。もちろん、頻繁に使うサーバは inetd に頼まず常時動くようにします。

ところで、サーバを動かすマシンに求められる特性とは何でしょう? 答えはある意味簡単で、「ネット経由でサービス要求が来たとき応えられる必要がある」ですね。それをもう少し具体的に書くと:

1. 常時動いていること。だから「使ってない時は止める」マシンは駄目。(Wake On Lan とかでパケットが来ると立ち上がるというのもありますが、ちょっと無理がありますね。)

2. 外部ネットワークから到達可能。固定 IP アドレスを持つか、DNS で名前が検索できる。(基本的にはインターネットに直接接続されている必要がありますが、ファイアウォールに転送設定をしておくことで、ファイアウォールの内側で動かすこともできなくはありません。)

ところで、上で述べたことにもかかわらず、クライアントサーバ方式ではない構成のネットワークアプリケーションもいくつかあります。これらは**ピアツーピア方式**と呼ばれ、特定のサーバはなく、各プログラムが対等な立場で通信することが特徴です。このようなシステムで大規模なものの代表例としては、後述する Napster、Winny、Skype などが挙げられます。

もっと身近な例でいえば、遠隔会議システムのようなものでは、「私とあなたが通信する」ことが目的ですから、互いに相手のマシンの IP アドレスが分かりさえすれば相互に音や画像をやりとりして会話ができるわけです。ただ、ピアツーピア方式であっても上に述べた「噛み合わせ」の問題を解決するために、「どのアドレスでサービスに参加しようとするプログラムが動いているか」という情報を交換するための登録機能の部分で、クライアントサーバ方式を援用しているものもあります (Napster がそうです)。

では次節以降で、各種のネットワークアプリケーションとその構造について順次見ていくことにします。

3 遠隔ログイン

3.1 遠隔ログインの原理

計算機ネットワークが作られたごく初期の時点から、「わざわざ遠くのマシンのところまで歩いて行かなくても手元のマシンの前に座ったままでそのマシンを利用できるようにしたい」という要望が存在し、それを可能にするソフトウェアが用意されていました。これを**遠隔ログイン**、ないし (手元のマシン上のソフトに遠隔端末の働きをさせることから) **ネットワーク仮想端末**と呼びます。そのためのコマンドとして今日の Unix では次の 3 つがおもに使われています:

- **telnet** 相手先 — TELNET プロトコルで他ホストに接続
- **rlogin** 相手先 — rlogin プロトコルで他ホストに接続 (Unix のみ)
- **slogin** 相手先 — ssh プロトコルで他ホストに接続

いずれも基本的には接続先ホストを指定して起動し、接続先ホストにおける自分のユーザ ID とパスワードを打ち込むとログイン認証が行われ、ログインすると接続先ホストのシェルが普通に使えます:

```
% telnet smb          ←ホスト指定して接続
Connected to smm.
Escape character is '^]'.
UNIX ...
login: kuno           ←ユーザ ID を入力
Password: *******     ←パスワード入力
Welcome ...
smb%                  ←接続完了
...                   ←普通に相手ホストで作業
smb% exit             ←終る
Connection closed ...
%                     ←元ホストに戻る
```

歴史的には、telnet が最も古くから存在し、多くの機種で用意されています (TELNET プロトコルを使った PC 向けの仮想端末ソフトも多数存在します)。続いて、Unix マシン相互でもっと簡単に使えるように、ユーザ ID を省略でき (手元のマシンを使っている時のユーザ ID を送る)、さらに特別な設定がなされていればパスワードも省略できる rlogin が作られ使われはじめました。[2]

遠隔端末ソフトでは、クライアント側はユーザがキーボードから打ち込んだ文字をそのまま相手ホストのサーバに送り、サーバ側ではそれらをシェル (ないしそれに対応するプログラム) に渡して実行させ、その出力をネットワーク経由で送り返してくるのでクライアントがそれを受け取って表示する、という形で動作しています。サーバ側ではログインした人の権限でコマンドを実行する必要があるため、ログイン認証が終わるとサーバ側で子プロセスを生成し、その子プロセスがユーザの権限でシェルを実行しています (図 6)。

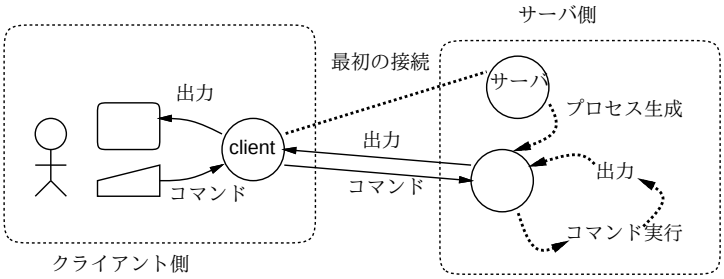

図 6: 遠隔ログインの原理

3.2 SSH の機能

telnet [3] と rlogin における問題は、これらのプロトコルでは往復する文字情報をそのままパケットに載せているので、(たとえばネットワーク診断用の機器やソフトを使って) パケットを傍受されるとパスワードも通信内容もすべて盗まれてしまうことです。

これでは余りに危険なので、通信路の両側で暗号化を行い、傍受があってもパスワードや通信内容が盗まれないようにしたのが **SSH**(Secure SHell) と呼ばれるソフト群と対応するプロトコルです。加えて SSH ではサーバ=クライアント間で公開鍵を利用して互いの正統性の確認も行います。

slogin はこのプロトコルを使って telnet 同様の遠隔ログインを行います。SSH は通常のパスワード認証の他に公開鍵暗号を用いた認証もサポートしていて、こちらであればパスワードが盗まれる心配をさらに減らすことができ、より安全です (ただし、最初に公開鍵と秘密鍵の対を生成し、公開鍵を自分でログイン先に持って言って設置する必要があるため、設定は面倒です)。

SSH が提供するもう 1 つの機能として、**ポート転送** (port forwarding) があります。これは、接続先ホスト内でしか使えないネットワークサービスを手元のマシンで使えるように接続を延長する「延長ケーブル」のようなものです (図 7)。たとえば、大学内の Web サーバには学内掲示が載っているため、そのサーバのコンテンツは外部には公開されていない、などの状況はよくあります。しかし大学まで行かなくても手元のマシンで学内掲示が見たいですよね? そのとき、次のように「延長ケーブル」を使うことができます:

```
% slogin -L8080:smb:80 接続先   ← smb のポート 80 を手元の 8080 に
password: *******
Welcome ...
```

[2] rlogin および後述の rsh で相手先でのユーザ名が手元のマシンでのユーザ名と違う場合は「-l ユーザ名」というオプションを指定することで相手先のユーザ名を指定できます。また slogin と ssh では相手先の指定方法を「ユーザ名@相手先」という形にすることでもユーザ名を指定できます。

[3] 最近の telnet では通信を暗号化する機能を提供しているものも一部あります。

```
>                                    ←接続完了
```

つまり、学内サーバのホスト名が「smb」だとして、接続先のマシンで smb のポート 80 番 (WWW サービスの標準ポート番号) に接続し、それを「延長」してきて手元のマシンのポート 8080 番として取り出すわけです。そこで手元のマシンでブラウザを起動し、http://localhost:8080 のページを開くと…学内サーバのページが読み出せるわけです。ここでは WWW を例に取りましたが、この方法で任意の TCP 接続を手元に延長してくることができます。

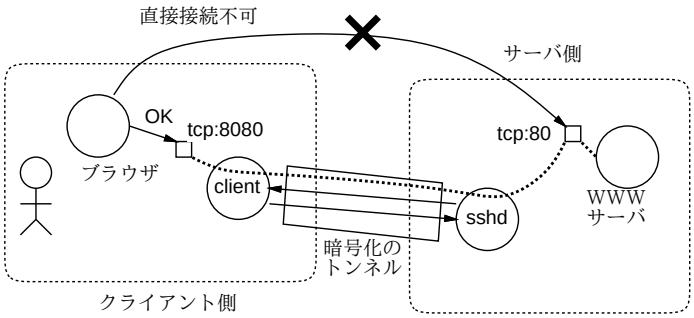

図 7: SSH によるポート転送

なお、勝手に学内サーバのアクセス範囲を延長してもよいのか疑問に思う人もいるかも知れませんね。この場合は、正当な利用権を持つあなたが認証を経て接続し、SSH の暗号化された通信路を経由してあなたの使っているマシンだけに延長してくるわけですから、それは構わないはずです。ただ、あなたのマシンのポート 8080 番に他人が接続できるようだと、他人も学内サーバが見えてしまいますから、それは避けるようにしましょう (SSH はとくに指定しない限り、-L オプションで延長してきた接続は手元のマシン内からしか接続を許さないようになっています)。

3.3 単一コマンドの実行 option

話は戻りますが、1 つだけのコマンドを実行させたいとき、わざわざログインして相手シェルに向かってキーボードからコマンドを打ち込み、終ったらログアウトする、というのは繁雑ですから、rlogin も slogin も「コマンドを 1 つだけ実行する」仕組みを用意しています。そのためにはコマンドとして **rsh**、**ssh** を使います:

- rsh 相手先 コマンド… — rexec プロトコルで 1 コマンド実行
- ssh 相手先 コマンド… — SSH プロトコルで 1 コマンド実行

これは例えば、次のように使うことができます:

```
% ssh as301.ecc.u-tokyo.ac.jp 'ps ax' >data  ←別マシンの ps 記録
Password: ******              ←注: パスワード応答機能は rsh にはない
%
```

つまり、相手先でのコマンド実行の出力は標準出力に出て来ますから、それを出力リダイレクションで手元のファイルに保存できるわけです (逆に入力リダイレクションで手元側からデータを相手先に送り込むこともできます。

4　ファイル転送

4.1　FTP と rcp option

ネットワークの機能が遠隔地間での通信である以上、それを利用してデータ (つまりファイル) をやりとりする、というのは当然あっていい使用法です。**ファイル転送**についても次の 3 つがあります:

- **ftp** 相手先 — **FTP** プロトコルでのファイル転送
- **rcp** 相手先:パス名　相手先:パス名 — Unix 間のファイル転送
- **scp** 相手先:パス名　相手先:パス名 — SSH によるファイル転送

FTP はネットの初期から存在していて、Unix に限定されないファイル転送プロトコルであり、PC 向けのクライアントソフトも多く存在しています。`ftp` コマンドでは接続後にパスワード応答による認証を行い、次のコマンド群を駆使して対話的にファイルをやりとりします:

- `cd` パス名 — 「向こう側で」現在位置を移動
- `lcd` パス名 — 「こちら側で」 〃
- `type text` — テキスト転送モードにする
- `type binary` — バイナリ転送モードにする
- `get` ファイル名 — 向うからこちらにファイルを転送
- `put` ファイル名 — こちらから向うにファイルを転送
- `bye` — `ftp` コマンドを終わる

たとえば**匿名 FTP** [4]で接続してサーバからファイルを取り寄せる対話例は次のようになります。

```
% ftp ftp.iij.ad.jp
Connected to ftp.iij.ad.jp.
220 ftp.iij.ad.jp FTP server ready.
Name (ftp.iij.ad.jp:myname): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password: myname@example.com
          ↑本当は打ち返されないので見えない
...
230 Guest login ok, access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd pub/rfc
250 CWD command successful.
ftp> get rfc822.txt
local: rfc822.txt remote: rfc822.txt
227 Entering Passive Mode (202,232,2,54,5,69)
150 Opening BINARY mode data connection for rfc822.txt (106299 bytes).
...
226 Transfer complete.
106299 bytes received in 11.21 seconds (9.26 KB/s)
ftp> bye
221-You have transferred 106299 bytes in 1 files.
221-Total traffic for this session was 107922 bytes in 1 transfers.
221-Thank you for using the FTP service on ftp.iij.ad.jp.
221 Goodbye.
%
```

[4]ユーザ名「ftp」ないし「anonymous」で接続し、パスワードとして自分のメールアドレスを打ち込むと、「誰にでも配布してよいファイル」を取り寄せるためのアカウントに接続できるという慣習をいいます。

現在では WWW の発達により、ファイルの配布に FTP を使う場面は少なくなっています。しかし PC とサーバの間でファイルを送受するのには今でも FTP プロトコルが多く使われています (クライアントソフトは PC 用のさまざまなものが使われることが普通ですが)。

rcp は FTP のような繁雑さなしに Unix システム間でファイルをコピーするために開発され、scp はそれを安全にしたもの、という経緯は遠隔ログインの場合と同様です (プロトコルもそれぞれ rexec と SSH を使っています)。[5]簡単な例を示しておきます:

```
% scp myname@examle.ac.jp:\*.txt .
password: ********     ←注: パスワード応答機能は rcp にはない
t1.txt  100% |*****************************|   363KB 00:00
t2.txt  100% |*****************************|  1127   00:00
%
```

なお、「*」などのメタキャラクタは手元で展開されては意味がないので (この場合、相手先ホストの「.txt」で終るファイルすべてを取り寄せたい)、「\」をつけて指定していることに注意してください。

4.2 ダウンローダとファイル交換ソフト option

厳密にいえば「ファイル転送」ではありませんが、ネット上のさまざまなデータを取り寄せる場合には、WWW サーバからデータを取り寄せる場合も含まれています。つまり、Web ブラウザをデータの取り寄せに使うこともできるわけです。さらに、データの取り寄せのみを目的とする場合は **wget** や **fetch** など、URI(後述) を指定するとそこからデータを取り寄せてファイルに格納してくれる**ダウンロード用コマンド**も使われます:

- wget *URI* — *URI* のデータを取り寄せてファイルに格納

fetch も基本的な使い方は同じです (それぞれ固有のオプションが指定でき、その部分では違いがあります)。また、これらのプログラム (Web ブラウザも含めて) は URI として FTP URI(後述) を使えば FTP サーバからの取り寄せにも利用できます。

ここまではクライアントサーバ型のデータ転送アプリケーションについて説明してきましたが、このほかにピアツーピア型の**ファイル交換ソフト**と呼ばれるものもあります。

これを最初に有名にしたのは **Napster** と呼ばれるシステムで、音楽データを交換するのに広く使われました — つまり、多くの人が手持ちの音楽 CD からデータを取り出してこのシステムで公開し、代わりに他人から自分の持っていない曲をもらったわけです (まさに「交換」ですね)。もちろんこれは**著作権法**に触れる行為であり、音楽産業各社が Napster 社を訴えました。Napstar 側は「交換は各個人がやっていることであり、自社は何ら悪いことをしていない」と弁明しましたが認められず、敗訴してシステム全体を有償化して 1 曲ごとに著作権料を払うことになりましたが、そうなるとユーザは「金を払うくらいならやめる」となって一気に下火になりました。[6]

Napster ブームは終りましたが、それが示した「ピアツーピアファイル交換」の次のような特徴は多くの人に強い印象を残しました:

- ユーザどうしが直接データを交換するので、誰が何を交換しているか追跡するのが難しい
- サーバがボトルネックにならないので、全体として大量のデータ交換が可能となる。

このため、今日では **WinMX** や **Winny** など多くのファイル交換ソフトが作成され広く利用されています。その一定割合は音楽データ、映像データ、有償ソフトの無断配布など、著作権に触れるよう

[5]手元のマシンと相手先でユーザ名が違う場合は rcp でも scp でも相手先の前に「ユーザ名@」をつけて指定できます。
[6]Napster は一旦破産しましたが、2003 年現在、復活して新しい形で音楽配信に復帰しようとしています。

な用途に使われている可能性があり、実際にそのような行為を行っていたユーザが逮捕される、などの事件も起きています。[7][8]

5 遠隔ファイルアクセス option

ファイル転送は複数のマシンで情報を共有する有力な手段ですが、意識して「あのファイルをこちらへコピーして」などと考えるのは繁雑ですし、間違えて悲惨な目に会ったりします。そこで、LANなどで高速なやりとりが可能な環境においては、「ファイルは1つのサーバ上に置いておき、どれかのマシンでファイルにアクセスするとネットワーク経由でサーバ上のファイルを読み書きする」ことが多く行われます。これを一般に**遠隔ファイルアクセス**ないし**ファイル共有**と呼びます (図 8)。自分がよく使うファイル群をファイル共有機能を使って各マシンでアクセスするようにすれば、ファイルはあくまでもサーバだけに置かれているため、あちこちのマシンで個別にファイルを管理する繁雑さがなくなります (大学などの環境では学生が多数のマシンのどれにログインするか分からないため、最初からこのような仕組みを前提としています)。

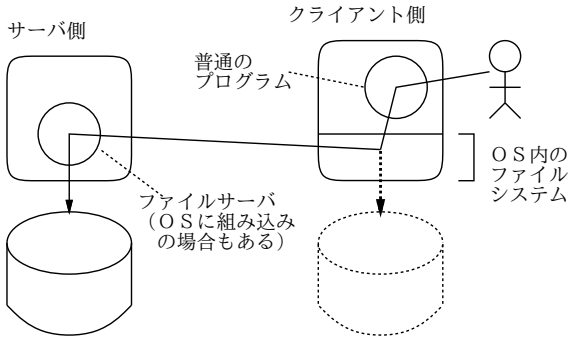

図 8: 遠隔ファイルアクセス

ファイル共有機能を Unix で最初に広めたのは Sun Microsystems 社の **NFS**(Network File System) で、現在でも Unix ではこれがファイル共有機能の標準です。そのほか、Windows のファイル共有プロトコルである **SMB** のサービスを提供するサーバソフト **Samba** を動かすことで、Windows マシンに対するファイル共有機能も提供できます。

なぜ他のサービスと違って OS の種別ごとにプロトコルが別なのでしょう? それは、ファイル共有の場合、OS の内部でファイルシステムを通じてファイルにアクセスする時、通常はディスク上にあるデータを読み書きしますが、ネットワーク共有されたファイルの場合はその本体はディスク上にあるわけではなく、ネットワーク経由でサーバマシンにアクセスに行かなければならないからです。つまり、他のネットワークサービスと違い、OS のファイルシステムの中にファイル共有プロトコルが組み込まれてしまっているため、「よく使われるプロトコルに統一しよう」などと簡単に決めることはできないわけです。とはいえ、WWW のプロトコルを元にした **WebDAV** と呼ばれるファイル共有方式は現在、Windows を含む複数の OS で共通に使われるようになってきています。

さて、Unix では上述のように NFS が標準で使われますが、Unix ではファイルシステム全体は既に説明したようにマウント機能により張り合わせられたディレクトリの木構造として構成されています。ここで、NFS サーバになっているマシンのディレクトリをマウントすることでそのディレクトリ以下のファイルがサーバと共有されるようになります。ファイル共有の状況はマウント状況を調べる

[7] とくに Winny は交換データを暗号化するため第3者が不法行為をチェックしにくいと言われていましたが、警察なども対応した技術開発を行っているようで、実際に逮捕された例があります。

[8] 筑波大学では Winny をインストールした PC を学内ネットワークに接続することを禁止していますのでよろしく。

mount コマンド[9]で表示させられます:

```
% /sbin/mount
dev/ad4s2a on / (ufs, local)
devfs on /dev (devfs, local)
/dev/md0 on /tmp (ufs, local, soft-updates)
/dev/ad4s2d on /var (ufs, local)
/dev/ad4s2e on /t0 (ufs, local)
procfs on /proc (procfs, local)
smo:/fbsd/r62usr on /usr (nfs, read-only)
pid623@smsi16:/vol on /vol (nfs)
linprocfs on /usr/compat/linux/proc (linprocfs, local)
sma:/u1 on /.amd_mnt/sma/u1 (nfs)
utogw:/sb on /.amd_mnt/utogw/sb (nfs)
```

マウントされているファイルシステムのうち、「nfs」と表示されているものが NFS により共有されているディレクトリに対応します。

6 電子メールとネットニュース

6.1 メールとニュースの原理

電子メールと**ネットニュース**はネットワークの初期からの情報交換サービスで、インターネット以外に **USENET** と呼ばれる電話線や低速の通信回線により接続されたネットワーク上でも広く使われていました。[10]メールは今日でも広く使われていますが、ネットニュースについては日本では Web 上の掲示板などが主流になり利用が少なくなっています (しかし、米国などでは依然として活発に利用されています)。

メールとニュースは一緒になって発達してきたため、交換されるメッセージの形式なども良く似ていますし、システムの構造もよく似ています。すなわち、メールでもニュースでも遠隔地との情報交換は基本的に「サーバどうしで」行う設計であり、ユーザは「自分の手元の」サーバと通信してメッセージを投入したり取り出したりします (図 9)。

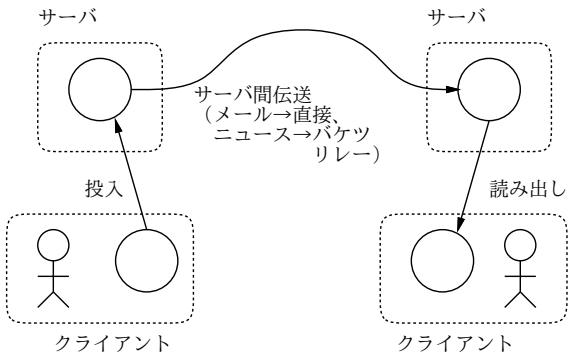

図 9: メール/ニュースの伝送経路

[9]通常、ディレクトリ/sbin、/usr/sbin などに置かれているので、実行パスに入っていない場合は/sbin/mount など絶対パスで指定する必要があります。

[10]日本では **JUNET** と呼ばれるネットワークが中心となって日本各地のサイトを結び、これらのサービスを相互運用していました。

なぜそうなっているかというと、インターネット以前のネットワークでは遠隔地との通信は時間がかかり複雑な制御を必要とする操作だったので、今日のように「ユーザがクライアントを起動して遠隔地のサーバと直接やり取りする」ことは困難だったからです。このため、そのような仕事は手元のサーバに依頼し、あとはサーバどうしでゆっくり情報を伝達してもらう、という構造になったわけです。なお、手元のサーバとのやりとりについても、古いシステムではサーバとユーザは同一のマシン上でファイルを共有することでメッセージを投入したり取り出していましたが、これはさすがにLANの発達とともに少なくなり、ネット経由で手元のサーバにアクセスする構造に変わっています。[11]

なお、これらのシステムではサーバは普通に**メールサーバ**、**ニュースサーバ**と呼ばれますが、ユーザがメッセージを読み書きするために使うクライアントは伝統的に**メールリーダ**、**ニュースリーダ**と呼ばれます。

電子メールの場合は、基本的にはユーザがメッセージを送信すると、そのメッセージは (メールリーダから直接にせよ、手元のサーバ経由にせよ) 宛先のサーバに届けられ、そこに格納されます。送信時のデータ伝達は **SMTP**(Simple Mail Transfer Protocol) と呼ばれるプロトコルで行われます。宛先のサーバはメールアドレスから分かるようになっています。たとえばメールアドレスは次のような形をしています:

```
someone@example.com
```

つまり「`@`」の後ろ側はドメインアドレス (この場合は `example.com`) になっています。ドメインアドレスはDNSを用いてIPアドレスに変換できることは既に説明しました。[12]IPアドレスが得られたら、そのアドレスのSMTPポートに接続してそこに「`someone` さん宛ですよ」といってメッセージを送り込めばよいのです。

メールがメールサーバに到着したら、ユーザはそのメールサーバから自分のメッセージを取り出して読むわけです。このときユーザがサーバとやりとりするのには、大きく分けて2通りの方法があります。

- ユーザが手元のマシンにメッセージを格納し管理する方法 — メッセージの投入はSMTPで行い、サーバから手元のマシンにメッセージを取り寄せる時には **POP**(Post Office Protocol) を使う。
- メッセージは基本的に常時サーバ内に格納しておき、ユーザはメッセージを読み書きするつどそのメッセージだけを手元のマシンとの間でやりとりする方法 — サーバと手元のメールリーダの間での通信に **IMAP**(Internet Message Access Protocol) を使う。

前者ではメッセージが手元のマシンにあってその場で読んだり返信したりでき、サーバと通信するのはメッセージを送受信する短時間だけで済むという利点がありますが、あるマシンで読んでしまったメールはサーバから手元のマシンに移動してしまうので、後で別のマシンから同じメッセージを読もうとしてもそれはサーバにはなくなっています。ですから、マシンを1つだけ決めて常にそこでメールを読み書きするという使い方になり、場合によっては不便です。後者ではこのような問題がありませんが、その代わりネットワーク経由でサーバとつながっていないとメールの読み書きができません。

ネットニュースの場合は、メールと異り個々のメッセージ (ニュース記事) には「宛先」はなく、ニュースグループと呼ばれる分類単位だけが指定されています。そこで、ユーザが手元のサーバにメッセージを投入すると、そのサーバは隣接するサーバに記事を転送し、そのサーバはさらに先のサーバに記事を転送し、というふうに「バケツリレー式」に記事が転送されていきます (逆に遠方からの記事も

[11] また、現在では遠隔地のメールサーバに直接アクセスしてメッセージを送信することもできます。ただ、向こう側のメールサーバが忙しかったりすると待たされるので、手元のサーバに投入してあとはサーバに任せる方が楽だし自然ということです。

[12] ただし、メール伝送の時はDNS上で「MX(Mail Exchange)」レコードと呼ばれるメール専用種別のデータを検索します。これは、メールアドレスの場合、あるアドレス (つまりメールサーバ) が停止していたら代替のサーバに送るなど、メール固有の扱いをする場合があるためです。

同様にして手元へ送られてきます)。このとき、記事の伝送の「堂々めぐり」が起きると大変なので、すべての記事には固有の**メッセージ ID** と呼ばれる識別名を割り当て、各サーバは手元に持っている記事のメッセージ ID 一覧を常に管理して、まだ持っていない記事だけを受け取ってリレーします。

ネットニュースではサーバに常時最新の記事群が大量に蓄積され相互に流通しているため、全部のメッセージを手元のマシンに取り寄せてから読むとういのは非現実的で、読みたいメッセージだけを選んで個別に読むことになり、メールで IMAP を使うのと似た形になります。ただしプロトコルについては、サーバどうしの通信もサーバとユーザが使うクライアントの通信も同じ **NNTP**(NetNews Transfer Protocol) を使って行います。

6.2 メッセージヘッダと SMTP

メールでもニュースでも、伝達されるメッセージの冒頭部分には**メッセージヘッダ**と呼ばれる部分があり、ここに各種の管理情報が格納されています。その後ろにメッセージ本文がありますが、ヘッダと本文の間は 1 行の空白行 (長さが 0 の行) で区切られることになっています。ヘッダ部分の情報を見ると、そのメッセージがいつどこで投入され、どのように中継されてきたかが記録されていることが分かります。メールヘッダの一例を見てみましょう。

```
Return-Path: kuno@mail.ecc.u-tokyo.ac.jp
Delivery-Date: Tue, 16 Dec 2003 18:24:48 +0900
Return-Path: kuno@mail.ecc.u-tokyo.ac.jp
Return-Path: <kuno@mail.ecc.u-tokyo.ac.jp>
Delivered-To: kuno@gssm.otsuka.tsukuba.ac.jp
Received: (qmail 38329 invoked from network); 16 Dec 2003 09:24:48 -0000
X-qmail-remote-mx: 0
Received: from ns.ecc.u-tokyo.ac.jp (133.11.171.253)
  by utogwpl.gssm.otsuka.tsukuba.ac.jp with SMTP; 16 Dec 2003 09:24:48 -0000
Received: from m.ecc.u-tokyo.ac.jp (mail.ecc.u-tokyo.ac.jp [133.11.171.196])
  by ns.ecc.u-tokyo.ac.jp (Postfix) with ESMTP id 5D84017EC82
  for <kuno@gssm.otsuka.tsukuba.ac.jp>; Tue, 16 Dec 2003 18:24:48 +0900 (JST)
Received: from mail.ecc.u-tokyo.ac.jp (as301.ecc.u-tokyo.ac.jp)
 by m.ecc.u-tokyo.ac.jp
 (Sun Internet Mail Server sims.3.5.2000.03.23.18.03.p10) with ESMTP id
 <0HPZ00E4CE5BEV@m.ecc.u-tokyo.ac.jp> for kuno@gssm.otsuka.tsukuba.ac.jp;
 Tue, 16 Dec 2003 18:24:48 +0900 (JST)
Date: Tue, 16 Dec 2003 18:24:47 +0900
From: kuno@mail.ecc.u-tokyo.ac.jp
Subject: test
To: kuno@gssm.otsuka.tsukuba.ac.jp
Message-id: <0HPZ00E4DE5BEV@m.ecc.u-tokyo.ac.jp>

This is a test.
```

見て分かるように、ヘッダは「フィールド名: 値」という形をした行が並んだもので、メールの Subject: や From:などの情報もすべてヘッダフィールドとして伝達されています。ただし、誰が送信したかの情報はこれとは別に SMTP プロトコルでも伝達しています。この情報を**エンベロープ From** と呼び、Return-Path:ヘッダに格納されます。また、Received: ヘッダを見ると、このメッセージがどのような経路を通って中継されてきたかを追跡することができます (途中のサーバがここに嘘を書き込んでいない限り)。

SMTP や NNTP などのプロトコルは、通常はメールリーダやニュースリーダがサーバとやり取りするために使うわけですが、通常の文字を使ったやりとりであり、簡単な構造をしているので、人間が手で打ち込んでみることもできます。たとえば、`telnet` クライアントでメールサーバに接続して自分あてに簡単なメッセージを送っている例を示しましょう:

```
% telnet utogw smtp  ← SMTP ポートを指定してメールサーバに接続
Trying 192.xx.xx.x...
Connected to utogw
```

```
Escape character is '^]'.
220 gssm.otsuka.tsukuba.ac.jp ESMTP
MAIL FROM:<kuno>  ←自分が誰かを示す (エンベロープ From)
250 ok
RCPT TO:<kuno>  ←メール宛先を示すコマンド
250 ok
DATA          ←「以下本文」コマンド
354 go ahead
From: kuno    ←メールヘッダが最低 1 つは必要
              ←空っぽの行がヘッダ終わりを示す。
test...       ←本文も 1 行以上あった方がよい。
.             ←「.」だけの行があるとおしまいを表す。
250 ok 989909466 qp 19829
QUIT          ←「これでおしまい」コマンド
221 gssm.otsuka.tsukuba.ac.jp
Connection closed by foreign host.
%
```

なお、ここで「嘘」を打ち込めばその嘘はそのまま相手に伝えられていくことに注意してください。メールヘッダの信頼性とはある意味その程度である、ということです。[13][14]ネットニュースの転送や読み書きに使われる NNTP もこれと同様の簡単なプロトコルです。

6.3 符号化と MIME option

上の通信例から分かるように、SMTP(や NNTP) ではメッセージを「生の」ままで文字として送るため、バイナリデータをそのまま送ることはできません (規格で 7 ビット文字のみと定められていますから、日本語テキストも JIS7 ビットコードで送らなければならず、SJIS や EUC も許されません)。このため、バイナリデータを送る場合はそれを一旦文字の集まりに直して (**符号化**、エンコード) 送ることが古くから行われていました (Unix では符号化形式として **UUENCODE 形式**、Macintosh では **BinHEX 形式**が一般に使われてきました)。

その後電子メールの普及とともに、さまざまなシステムで共通に符号化データを流通させることが望まれるようになり、特定プラットフォーム向けでない符号化方式として **MIME**(Multipurpose Internet Message Extension) と呼ばれる規格が作られました。この規格では、1 つのメールメッセージは複数のコンテンツ (内容) を集めたものとなり、それぞれのコンテンツは「そのまま」「**base64 符号化**」「**quoted-printable 符号化**」などの選択肢からどれか 1 つの方式を選んでメッセージに組み込まれます。たとえば「あいうえお」という 1 行のファイルを 4 通りの方式で 1 つのメッセージに組み込んだ例を見てみましょう (送信前の状態なのでメールサーバによるヘッダはまだ付加されていません):

```
To:  kuno
Subject:  test...
MIME-Version: 1.0
Content-ID: <Sun_Dec_21_14_47_29_JST_2003_0@sma>
Content-type: multipart/mixed;
    boundary="sma.52473.Sun.Dec.21.14:47:29.JST.2003"

This is  a multimedia message in MIME  format....
--sma.52473.Sun.Dec.21.14:47:29.JST.2003
Content-ID: <Sun_Dec_21_14_47_29_JST_2003_1@sma>
Content-type:text/plain
Content-Transfer-Encoding:7bit      ←そのまま
```

[13] しかし世界中のメールサーバを乗っ取ることは難しいでしょうから、メールヘッダを解析してやればどこまでが嘘でどこからが本当かはそれなりに分かるわけです。

[14] GSSM のサーバではローカルサイトからの送信ではエンベロープ From は実在するユーザかどうかチェックされます。ということは存在しないユーザは書けませんが他人になりすますことはできます。

```
あいうえお
--sma.52473.Sun.Dec.21.14:47:29.JST.2003
Content-ID: <Sun_Dec_21_14_47_29_JST_2003_2@sma>
Content-type:text/plain
Content-Transfer-Encoding:base64     ← Base64 符号化

GyRCJCIkJCQmJCgkKhsoQgo=
--sma.52473.Sun.Dec.21.14:47:29.JST.2003
Content-ID: <Sun_Dec_21_14_47_29_JST_2003_3@sma>
Content-type:text/plain
Content-Transfer-Encoding:quoted-printable ← quoted-pprintable 符号化

=1B$B$"$$$&$($*=1B(B
--sma.52473.Sun.Dec.21.14:47:29.JST.2003
Content-ID: <Sun_Dec_21_14_47_29_JST_2003_4@sma>
Content-type:text/plain
Content-Transfer-Encoding:x-uue       ← UUENCODE 符号化

begin 600 test.txt
1&R1")"(D)"0F)"@D*ALH0@H‘
‘
end
--sma.52473.Sun.Dec.21.14:47:29.JST.2003--
```

つまり、メッセージ全体の形式は **multipart/mixed** 形式と指定され、区切り文字列が併せて指定されます。続いてその区切り文字列で区切られた内容が複数つきますが、それぞれの内容の冒頭にもヘッダがついていて、内容の種別 (この場合はプレーンテキスト) と、どの方式で符号化されているかなどが記載された後に、本体がついています。

実際にはこのようなメッセージをユーザが手で書くわけではなく、メールリーダが生成してくれます。一般には「記事本文」に「追加のファイルをつけて送る」使い方が多いため**添付ファイル**と呼ばれていますが、実際には上の例のように、最初の本文も追加のファイルも一緒に束ねられています。

なお、メールリーダによっては日本語の SJIS や EUC を送ろうとすると 8 ビット目の立った文字が含まれるため自動的に符号化して送ってくれてしまい (少なくともそのような設定が標準のものがあります)、受け手のメールリーダが MIME に対応していないと内容が読めないなどの不都合を起こすことがあります。注意しましょう。

7 World Wide Web

7.1 ハイパーテキストと WWW

今日、最も多くの人に利用されているネットワークアプリケーションといえば電子メールと **WWW**(World Wide Web、以下 Web と略します) だといえますから、少なくとも本文を読んでいる人のなかで Web を見たことがない人はいないと思われます。しかし、その見慣れた Web 画面の裏側がどのようにできているのかを考えてみたことのある人は、それほど多くないかも知れません。まずこの点から見ていきましょう。

Web の基本的な枠組みは**ハイパーテキスト**です。ハイパーテキストとは何か説明するとすれば、次のようなものになるでしょう:

- 計算機の画面上にテキストや画像などの内容 (コンテンツ) が表示されている。
- コンテンツの中には、他のコンテンツやその特定箇所を「指し示して」いる箇所が埋め込まれている。これを**リンク**という。
- リンクの箇所を何らかの方法で選択すると、画面はそのリンク先の内容に切り替わる。

- このようなリンクで互いに結び合わされたコンテンツの集まりは、リンクを自由にたどりながら読み進んでいくことができる。

ハイパーテキストの概念そのものは WWW よりずっと前から存在しました。たとえば Macintosh で広く利用されてきたアプリケーションである**ハイパーカード**や、Windows の**ヘルプ**などはリンクで結び合わされたコンテンツの集合体ですから、ハイパーテキストの典型例だといえます。Web も複数の**ページ**と呼ばれる単位がリンクによって結び合わされた構造を持っているので、ハイパーテキストの一種だといえます (図 10)。なお、Web をみるためのプログラムを一般に **Web ブラウザ**ないし単に**ブラウザ**と呼びます。[15]

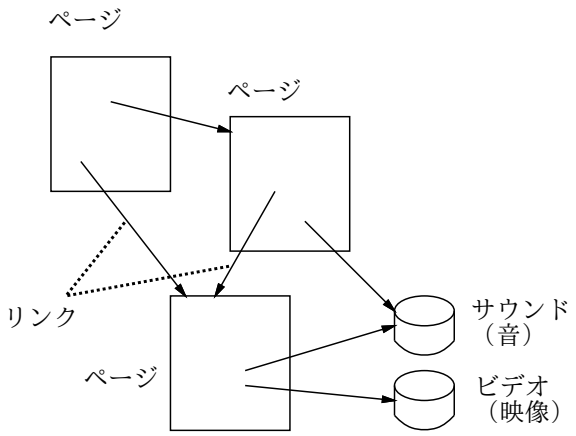

図 10: ハイパーテキストと WWW

ではハイパーテキストには普通の (紙の) 文書とくらべてどんな利点があると言えるでしょうか?

- 最初から順番に読まなくても、必要な箇所だけ選択して読むことができる。
- 計算機の機能を活用して、さまざまな支援ができる。
- 紙ではないので、画像や動画や音などの**マルチメディア**が取り入れられる。

最後の点に関連して、単なる文字 (テキスト) だけでないことを強調する場合には**ハイパーメディア**という言い方をすることもあります。

では、Web 独自の部分、つまりハイパーカードや Windows のヘルプにない部分というのは何でしょうか? それは次のような点だといえます:

- Web では、リンクは「ネットワーク上の任意のページ」を指すことができ、従って世界中の情報源を行き先とすることができる (World Wide の意味。ちなみに Web は「くもの巣」の意味で、リンクが網目状にはりめぐらされていることを表しています)。
- さらに、リンクはページだけでなく、「ネットワーク上のさまざまなモノ (資源)」を指すことができる。
- それらの資源の中には、単なる受動的なデータではなく、自らが能動的に動くようなもの、さらには「読み手」から情報を受け取るものも含まれている。
- これらの情報源はリンクをたどった瞬間にネットワーク経由で取り寄せられるので、ディスクや CD-ROM の容量などの制限とは無縁であり、また常に「発信されている最新の情報」が取り出せる。

[15] 本来ブラウザというのは「色々なものを眺め回るためのツール」という意味であり、Web とは関係ないブラウザもあるのですが、Web があまりにも普及したため、ブラウザと言えば Web ブラウザの意味で通用するようになってしまいました。

- Unix、Macitosh、Windows という主流のプラットフォームすべてにブラウザが用意されており、どのプラットフォームでもどこから来た情報でも表示できる (**クロスプラットフォーム**)。

1990 年に WWW をはじめてつくり出したのは、当時スイスの **CERN**(欧州粒子物理研究所) に所属していた Tim Berners-Lee です。彼の発明以前からインターネットはずっと存在し、電子メール、ネットニュース、FTP などの手段で情報を流通することは可能だったのですが、これらは操作方法もバラバラで、どこに何があるか熟知していなければ情報にアクセスすることができませんでした。ここにハイパーテキストの考え方を取り込み、情報のありかや種別を知らなくてもページに内包されたリンクを選択するだけでその情報にアクセスできるようにした、というのが WWW の偉大な発明だったわけです。

CERN で作られた最初の Web ブラウザはテキストしか扱うことができませんでしたが、**NCSA**(米国アリゾナ州のスーパーコンピュータ応用センター) の学生たちが多くの Unix システムで動き、ページ中に画像を含められるブラウザ **Mosaic** を開発し公開したことで、Web は急速に普及し始めました。

その後、Mosaic の開発者たちは Netscape Communications 社に移り、より高度で Unix 以外に Windows や Macintosh でも動く **Netscape** ブラウザを発売し、巨額の利益をあげました。しかし他の企業にインターネットという巨大市場を取られまいとした Microsoft 社が Windows にブラウザ **Internet Explorer** を無償で付属させる戦略を取ったため、Netscape 社はブラウザで儲けることができなくなり没落しました。[16]

一方、WWW のさまざまな規約 (プロトコルや記述言語) の標準に対するニーズも急激に高まり、これを受けて **WWW コンソーシアム** (W3C) と呼ばれる共同体が発足し、標準の取りまとめや参照用のソフト開発を行うようになりました (Tim Berners-Lee も W3C の発足とともにここに移っています)。

歴史の話は詳しくするときりがないのでこれくらいにして、先に挙げた WWW の特徴が実際にどのような形で現われているかを、もう少し見て行くことにしましょう。

7.2 URI とリンク

WWW では「さまざまなモノ (資源、resource) を指すポインタ」として **URI**(Uniform Resource Identifier) と呼ばれる形式を使用しています。URI は一般に次の形をとります:

スキーム:アドレス

ここで**スキーム**部分は資源の種別を表し、**アドレス**部分はその種別のなかのどれであるかを識別します。たとえば電子メールアドレスという種別の資源であれば次の形になります (メールアドレスは宛先となる人を一意的に指定できますから):

`mailto:`メールアドレス

これを **mailto URL** と呼びます。また、FTP でファイルを取り寄せる場合には次のようになります:

`ftp://FTP` サーバ/ディレクトリ/.../ディレクトリ/ファイル

FTP サーバと、そこのどのファイルかを指定すればファイルが一意的に指定できるので、これでよいわけです。こちらは **FTP URL** です。そして、Web サーバからコンテンツを取り寄せるためのプロトコルである **HTTP**(HyperText Transfer Protocol) を使う場合は、上とほとんど同様ですが、次のようになります:

`http://`ホスト指定/ディレクトリ/.../ディレクトリ/ファイル

[16] さらに Microsoft は、パソコン製造各社に Netscape を搭載しないよう働きかけるなど公正でない活動をしたとして訴えられたりもしています。

これが **HTTP URL** で、当然ながら Web 上ではこれがもっとも多く使われています。なお、サーバとブラウザの間で暗号化通信を行うために HTTP に暗号を入れた **HTTPS** プロトコルもあり、その場合は上の `http:` を `https:` に取り換えます。

ここまでに上げた URI の例はいずれも、資源が「どこに」あるかを含んだ情報であり **URL**(Uniform Resource Locator) と呼ばれる種類のものです。これ以外に、資源がどこにあるかに関係なく対象を指すものもあり、これは **URN**(Uniform Rerource Name) と呼ばれます。たとえば「`urn:isbn:4-621-04373-0`」というのは ISBN(書籍番号) を表す URN です (書籍はどこの本屋さんで購入しても ISBN が同じなら同じ本ですからこれでいいわけです)。そして URI は URN と URL を合わせたものをいいます。[17]

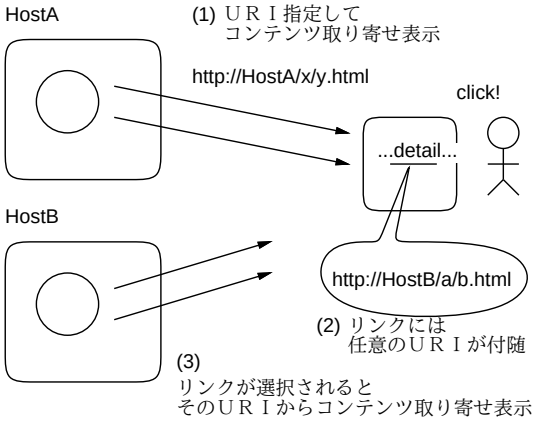

図 11: リンクの仕組み

なぜここで URI について長々と説明しているのかというと、URI こそが WWW の「肝」だからです。後で見るように、WWW でコンテンツを伝送しているのはごく簡単なプロトコル (実質はファイル転送のようなもの) であり、ブラウザがやっていることは基本的には次の 2 点だけです:

a. 指定された URI からコンテンツを取り寄せる。

b. 取り寄せたコンテンツを適切な形で画面に表示する。

ただ、ここでコンテンツを取り寄せたとき、それが **HTML** 形式のファイルであれば中にリンクが埋め込まれていることがあります (つまり、そこに別のコンテンツを指す URI が埋め込まれています)。そして、ユーザがそのリンクを選択すると、ブラウザはそのリンク先のコンテンツを取り寄せ、表示画面をそちらに切り替えます (つまり上記 a の動作を行います)。それにより、いつもやっているように「別のページへ飛ぶ」ことができるわけです (図 11)。これだけで済むのは、「リンク先」が URI の形で表されていて、世界中のどのサーバのどのコンテンツであっても自由に指し示すことができるから、なのです。

7.3 HTML の初歩

ではここで、ごく簡単に HTML とはどんなものか説明し、自分のページを作って見ましょう。GSSM では各自のホームディレクトリの下に `www` という名前のサブディレクトリを作ってそこに HTML ファイルを入れるだけで、内部サーバ経由にページを公開することができます。以下の作業は最初の 1 回だけ行ってください。

[17] Web の初期には URN という考え方はなかったため、URI という用語もなく、URL だけが使われていました。今日でも URN が使われる機会はあまりないため、依然として URN や URI という言葉を知らない人が多数います。

```
% mkdir WWW
% chmod a+rx WWW
% touch WWW/index.html
% chmod a+r WWW/index.html
```

あとはこの WWW/index.html を Emacs などで編集するだけです。とりあえず、次のような内容を入れてみてください (○○のところは自分の名前をどうぞ)。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title>○○'s page</title>
</head>
<body>
<h1>○○です。</h1>
<p>…挨拶ないし自己紹介を書く…</p>
</body>
</html>
```

これを作成した後、ブラウザで http://w3in/~ユーザ/を表示してみてください (図 12)。

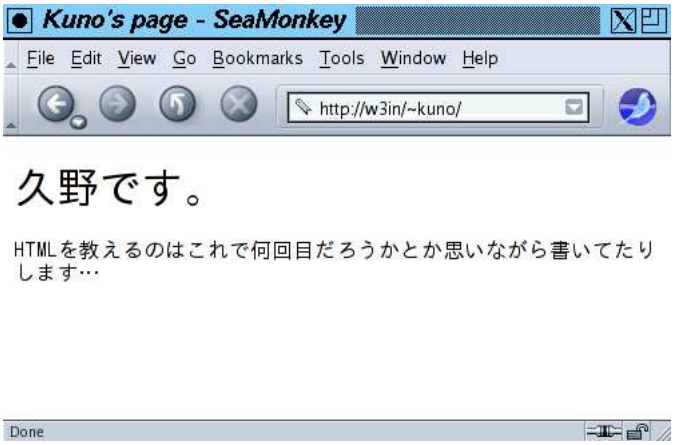

図 12: HTML をブラウザで表示

ではこの概要を説明しましょう。1 行目は DOCTYPE 宣言といい、この文書が HTML 文書 (バージョン 4.01) であることを宣言しています。2 行目からが HTML です。

HTML では<名前>…</名前>という形で範囲を指定します。これを**要素**、最初の<名前>は開始タグ、最後の</名前>を終了タグと言います。つまり開始タグから終了タグまで 1 つの要素が構成されるわけです。要素の中には別の要素が入れられます (入れ子構造)。

上に出て来る HTML の要素 (タグ) の意味は次の通り。

- <html>…</html> — HTML 文書全体をあらわす。
- <head>…</head> — ヘッダ (この文書に関する情報を記述する部分) をあらわす。
- <title>…</title> — 文書のタイトルをあらわす。
- <body>…</body> — 文書本体 (ブラウザの窓の内側に表示される内容全体) をあらわす。
- <h1>…</h1> — レベル 1 の見出し (大見出し) をあらわす。

- … — 通常の段落をあらわす。

なお、段落等の中はすべて窓の幅に合わせて詰め合わせされますから、空白や改行を入れても意味がありません。空白や改行で整形したい場合は pre 要素を使います。

- <pre>…</pre> — この内側は空白や改行をそのまま残す。

次のような形のものを末尾 (ただし</body>の前) に入れてみると図 13 のようになります。

```
<pre>
  プログラム
        教える手間も
                また楽し。
</pre>
```

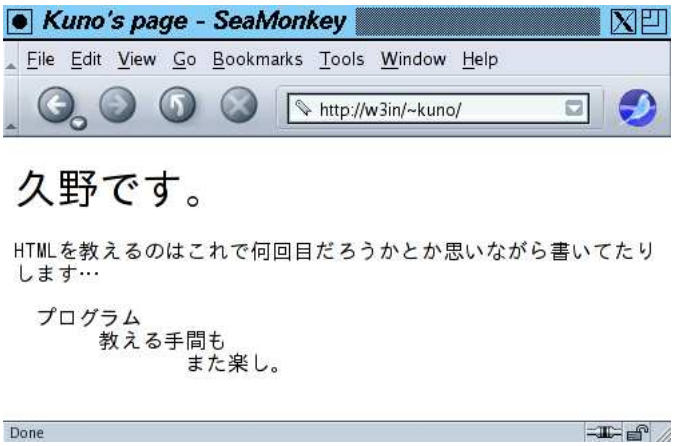

図 13: pre 要素は空白と改行が残る

最後にリンクのつけ方をやってみましょう (h2 要素はおまけ)。

- リンクテキスト — 「リンクテキスト」の部分がリンクとして表示され、そこを選択すると *URI* のページを表示する。
- <h2>…</h2> — 第 2 レベルの見出し (中見出し) を表す。

このように、a 要素の開始タグは「href=...」という指定 (**属性**と呼びます) がつけられ、それによってリンク先を指定します。これを利用して、「リンク集」を作ってみましょう。1 項目を 1 段落とします (リンクは段落等の中に入れることになっています)。例を示しましょう (図 14)。

```
<h2>私のリンク集</h2>
<p><a href="http://www.fujitv.co.jp/">フジテレビ</a>のサイト
はよく見に行きます。ドラマ好き。</p>
<p><a href="http://www.shes.net/dramacheck/">ドラマチェック</a>
なんていうのも愛用しています。</p>
```

なお、ここまでの説明だと WWW は HTML を取り寄せて来て表示するだけのシステムということになりますが、実際にはもっと対話的に、ブラウザ内に入力してそれをサーバに送って処理するなどのことも行えます (そういうページを誰でもよく使っているはずです)。そのあたりは「計算機ソフトウェア」で扱います。

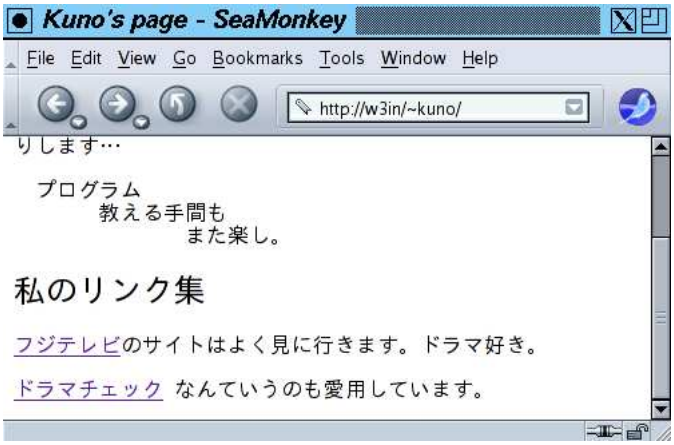

図 14: リンクを入れたページ

7.4 WWW サーバとブラウザのやりとり (HTTP)

最後に、WWW のしめくくりとして、Web サーバとブラウザがどうやって通信しているのかを見てみましょう。SMTP と同様、HTTP もごく簡単なプロトコルで、`telnet` コマンドを使って直接打ち込んで試すことができます。

```
% telnet w3in http        ← HTTP プロトコルで接続
Trying 10.1.0.3...
Connected to smb.         ←サーバに接続された
Escape character is '^]'.
GET /~kuno/           ← GET で URI のサーバ名より後ろの部分を指定
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title>Kuno's page</title>
</head>
<body>
<h1>久野です。</h1>
<p>HTML を教えるのはこれで何回目だろうかとか思いながら書いて
たりします…</p>
(途中略)
</body>
</html>                                  ← HTML が返送され
Connection closed by foreign host.   ←おしまいになる
```

つまり、サーバにコンテンツのパスを打ち込むと直ちにその内容が返送され、接続が切断されます。このように簡単にしておくことで、ブラウザはあちこちのサーバのページをロードしまくれるし、サーバは現在どこのブラウザが自分につながっているかをいちいち管理しなくても済むため高速なサービスが行え、大量のクライアントを扱えるようになるのです。[18]

8 その他のネットワークアプリケーション option

今回の冒頭でも述べたように、ネットワークアプリケーションは非常に多様なものがあり、ひととおりであっても全てを解説し尽くすことはできません。残ったものの中から代表的なものだけ選んで、ごく簡単に説明しておきましょう。

[18] 毎回 TCP 接続を張り直すのは無駄が大きいので、1 つ取り寄せた後も次の内容を取り寄せるために接続を切らずに応答を続けるような指定も可能になっています。

- **チャット** — チャットとは「おしゃべり」の意味で、ネットワーク上で複数のユーザどうしが短いメッセージを交換するようなネットワークアプリケーションをいいます。インターネット上では古くから **IRC**(Internet Relay Chat) と呼ばれるシステムが稼働していますが、これは大量のユーザが参加できるように複数のサーバが連係してメッセージを相互に中継し、ユーザはどれかのサーバに接続して会話に参加します。一方 **ICQ** と呼ばれるシステムはピアツーピア型で、会話したい相手どうしのマシン間で直接メッセージを交換します。
- **ストリーミング** — 音声のみ、あるいは音声と画像のデータを実時間的にクライアントに流す (つまり、クライアントはファイルを蓄積するのでなく、取り寄せつつその場で直ちに再生する) ようなサービスで、いわばネット上のラジオやテレビです。**RealSystem** と **WindowsMedia** が双璧をなしています。

  従来のストリーミングが専用クライアントを使うのに対し、YouTube やニコニコ動画などの**動画サイト**では **FlashVideo** 形式を使います。現在はほとんどのブラウザが **Flash** の再生ソフトを同梱しているので、新たにクライアントを入れなくてもそのまま見られ、また Web ページに埋め込まれるため Web ページ側にコメントや投票などさまざまな仕掛けを入れることができ、このため急速に広まりました。
- **テレビ会議** — 音声と画像をクライアント間で双方向に流す「テレビ電話」です。**NV**、**NetMeeting** などが代表的です。
- **プリントサービス** — プリンタがつながっていないマシンからでもネットワーク経由でプリンタに印刷できるようなサービスをいいます。バークレー版 Unix に由来する **lpd** が基本的で昔からありますが、より高度な機能を提供する **CUPS** などのシステムもあります。
- **ネームサービス** — 一般に「名前」から対応するデータを検索させてくれるサービスをいいます。**DNS** も IP アドレスを中心としたネームサービスだといえます。Unix 系のサイトでは LAN 内でユーザ管理情報やホスト情報などを共有するために、Sun が開発した **NIS** と呼ばれるサービスを用いているところが多くあります。特定プラットフォームに依存しない、汎用的なネームサービスのプロトコルとして **LDAP**(Lightweight Directory Access Protocol) があり、このプロトコルが使えるサーバやクライアントも多くなっています。

9 まとめと演習

今回は、実際にネットワーク上で私達がお世話になるようなネットワークアプリケーションとその仕組みについて、代表的なものを選んで概観してきました。これらをざっと見ただけでも、ネットワークには新しいものが生まれて来る余地が無限にあることが分かると思います。

5-1. ブラウザに設置されているルート CA の一覧を表示させて見なさい。次に、SSL サーバのサイトに接続し (普段自分が使っているショッピングとかでも沢山あるはずです)、その状態で「鍵」アイコンを選択して CA の証明書情報を確認してみなさい。また、「オレオレ証明書」のサイトだとどうかも試してみなさい。[19]

5-2. 携帯や別のアドレスから GSSM のメールアドレスにメールを送って (または友人に送ってもらって)、そのメッセージのヘッダを調べてみなさい。そのメッセージは、どのようなサーバを経て到着していますか。それらには送り元によってどのような違いがありますか。 また、ファイルを添付したメールを送った場合はどうですか。

5-3. `telnet` コマンドを使って手元のメールサーバの SMTP ポートに接続し、手で SMTP コマンドを打ち込んで自分あて (または知合いあて) にメールを送ってみなさい。「嘘」を書いたらそれはそのまま送られるか、その他普通にメールを送るのと違うことはないかどうか、試しなさい。

[19] たとえばここ:`http://eap.pref.nagasaki.jp/e-apply/index.php`

5-4. 自分の Web ディレクトリ (`~/WWW`、保護モード `rwx--x--x`) にファイル `index.html` を置いて誰でも読めるように (保護モード `rw-r--r--`) 設定し、内容として簡単な HTML を入れてブラウザからアクセスして確かに Web ページとして見えることを確認しなさい。HTML の上で空白や改行を入れても窓の幅で詰め合わされて意味がないこと、ただし pre 要素の中だけはその通りになることも確認しなさい。

5-5. 自分の Web ディレクトリの index.html に「リンク集」を入れてみてください。つまり、自分がよく行く外部のページにリンクを張るわけです。

5-6. `telnet` で内部サーバや外部の Web サーバに接続して、内容を取り寄せてみなさい (ブラウザで表示させながら同じ URI から取るようにすると打ち間違いを避けやすいと思います)。どのようなコンテンツに対してどのようなものが返送されてくるかまとめなさい。

今回も外部のサーバに接続する場合は `utogw` で実施すること。