解　説

UNIX のネットワーク機能†

村　井　　純††

1. はじめに

UNIX オペレーティングシステムは，その主な利用者層である計算機研究者や高度な計算機技術の開発者の要求に対応すべく発達してきた．また，その発達は，歴史的な利用者の技術が蓄積されるという形で実現されている傾向がある．

UNIX オペレーティングシステムにおけるネットワーク技術もこうして開発された技術の一つであり，非常に多種多様の技術が実現されてきている．そこで，ここでは，これらの技術のうち，広く利用され，安定した評価を得ている部分について解説し，最後に今後の展望を述べることにする．

2. UNIX ネットワークの応用機能

UNIX ネットワークの応用機能は，その機能を提供するために必要なオペレーティングシステムの機能の段階によって以下のような三つのレベルに分けることができる．

（1） メッセージ交換の機能
（2） ホストの存在を意識するネットワーク機能
（3） 分散環境を実現するための機能

2.1 メッセージ交換の機能

メッセージ交換とは電子メイル，電子ニュース，電子掲示板などのように有限長の文字データを交換する機能である．UNIX は時分割システムで複数の利用者に対するサービスを提供することができるので，このような機能は一つのシステムの中での情報交換機能として本来備わっていた．ネットワーク環境におけるメッセージ交換機能はネットワークで結合された遠隔システムとの間での情報交換が実現できるものである．

この機能を提供するための下位機能としては有限長のデータ（フィルタ）とその取り扱いを指示する命令とが目的システムに転送され，目的システムにおいてそのファイルが指示に従って正しく処理できればよい．したがって，通信機能としてバッチ式のファイル転送が提供されていれば実現が可能となる．

2.2 ホストの存在を意識するネットワーク機能

独立した複数のシステムがネットワークによって結合された場合に要求される機能としてファイル転送機能，遠隔ログイン機能，遠隔コマンド実行，がある．これらの機能は，ネットワーク上の遠隔システム（ホスト）の識別子を指定することにより目的の処理が実現できるという共通の特徴を持っている．

これらの一部の機能はバッチ式の通信によって実現することも可能だが，一般的には遠隔システム間での双方向のバイト列が端末回線と同等またはそれ以上の速度で交換できる通信機能が要求される．

2.3 分散環境を実現するための機能

ネットワークで結合した複数のシステムが管理する資源を効率的に利用するためには利用者からネットワークの存在が隠ぺいされる分散環境の実現が要求される．分散環境を形成する応用技術は分散ファイルシステム，分散データベースなどの基礎技術に加えて，並列処理記述用プログラミング言語を基盤とした広範囲の可能性が考えられる．

このような機能を一般的に実現するための基礎機能としては高速のプロセス間通信が提供されることが必要である．資源の位置が意識されないためには自システムの資源に対するアクセスの速度と遠隔システムの資源に対するアクセスの速度との差が実用上無視できる範囲でなければならない．これを通信速度に対する具体的な目安として考えることができる．

3. UNIX オペレーティングシステムとネットワーク

UNIX における上記応用機能の実現はバッチ方式と階層的通信プロトコルの組み込みによる方式の二つ

† Network Functions of UNIX Operating System by Jun Murai (Computer Center, Tokyo Institute of Technology).

†† 東京工業大学

の方式によっている．バッチ方式の通信機能は UUCP [Nowitz 78] と呼ばれ，シリアル回線を介したファイル転送を行う機能である．シリアル回線を用いると端末の制御のための資源を用いることができるので通信用ソフトウェアは端末と入出力を用いて記述することができる．そのためオペレーティングシステムの構造としても特別な仕組みは必要なく，通常の応用プログラムとしてネットワーク機能を作成することができる．

uucp は自動発呼装置と自動受信装置の組み合せによって公衆電話回線を使用することができるために，特に，組織間を結合するネットワークを簡単に構築する目的で広く利用されている．バッチ形式であるために使用目的はメッセージ交換と簡単なファイル転送に限られ，会話型のネットワーク機能のために利用することはできない．

UNIXオペレーティングシステムが分散型オペレーティングシステムとして発展していくためには，遠隔システム間でのプロセス間通信を基本に進化していく必要があった．バージョン7までの UNIX におけるプロセス間の通信としては pipe と呼ばれるシステムコールが提供されているが，これは親子関係を持つプロセス同士でメモリを共有することによって実現されているために本質的に遠隔システム間のプロセス間通信に利用することはできない．また，System III 以降の AT & T 版の UNIX において提供されている namedpipe と呼ばれる機能は，ファイルシステム内に共有するメモリをアクセスするためのノードを設定したもので遠隔システム間でファイルシステムが共有された場合に限って遠隔システム上のプロセス同士による通信に利用することを可能としている．このためにはファイルシステムを遠隔システム間で共有するための機構の実現を目的とした通信機能が別に開発されなくてはならない．さらに，AT&T 版の System V において提供されているプロセス間通信の機能 IPC [AT & T 86] も同一システム内のメモリの共有を利用した機構であるためにネットワーク通信を用いたプロセス間通信にそのまま利用することはできない．

そこで，汎用の分散機能を提供するためには汎用のネットワークプロトコルを用いた通信が可能なプロセス間通信機能が定義される必要がある．4.2 BSD に組み込まれたネットワーク機能では，socket と呼ばれるプロセス間通信モデルとカーネル空間に組み込まれたプロトコル階層によってネットワークを介したプロセス間通信を可能にしている．UNIX オペレーティングシステムにおいては二つの空間，利用者空間とシステム空間にそれぞれ存在する基本的な構成要素，プロセスとカーネル，にネットワーク階層構造をどのように組み込むかが効率を左右する重要な問題であり，この面での新しい技術が開発されている．

4. UUCP ネットワークの構造

歴史的な視点における UNIX ネットワークは，ベル研究所で開発された UUCP ネットワークの出現によって一般的に利用されるようになった．UUCP ネットワークは，直接結合またはダイアル呼出の電話回線を利用したシリアル回線を基調としたネットワークでバッチモードのファイル転送と UNIX コマンドの遠隔実行の機能を提供している．利用者プロセスとして実現されていること，プロセス間の情報交換はスプールファイルを用いて行っていること，UNIX バージョン7で供給されているシステムコールを用いて実現されていること，などの理由から現存するすべての UNIX 間での通信が可能となっている点に特徴がある．

4.1 UUCP ネットワークの機能

UUCP ネットワーク機能の最上位レベルのインタフェースは，uucp と呼ばれるファイル転送機能，および，UUX と呼ばれる遠隔コマンド実行機能の二つのコマンドによって提供されている．

一般的に UUCP 機能における遠隔資源の指定は，次のような表現を用いて行う．

システム名!資源名

4.2 UUCP ネットワークの機構

uucp や uux コマンドはスプールディレクトリ内に次の三つの情報を格納する．

（1） 対象となるシステム名

（2） 通信すべきファイルの指定と，遠類システムでのそのファイルの取り扱いの指示．

（3） データファイル

これらのうち，（2），（3）は，ファイルとして作成され，（1）の情報は，それらのファイル名として伝達される．

こうしてスプールされた情報は，uucico と呼ばれる通信プログラムによって処理される．uucico は，対象システムごとに定められたシナリオに従って対象システムの呼び出しを行う．呼び出しは自動発呼装置に対する命令列および対象システムに対するログイン手

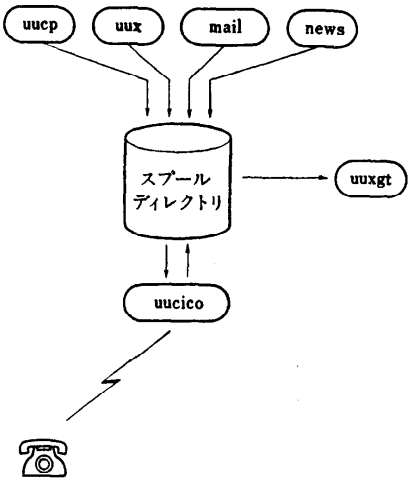


図-1 UUCP の構造

続きが含まれているのが普通である.

UNIX ではログイン時に起動されるプログラム(通常はコマンドインタプリタ)をログイン名ごとに定めることができる. 呼び出されるシステムでは, これを利用して uucico を起動するように設定する. これによってシリアル回線を介した uucico 同士の結合が確立され, 同プログラムによってプロトコルに基づいた通信が行われる.

受信されたファイルは, 再びスプールディレクトリに格納され, uuxqt と呼ばれるプログラムによって送信システムにおいて指定された指示に従って処理される. バッチ式の通信機能である UUCP の構造を図-1 に示す.

5. 4 BSD* [Leffler 83]

4 BSD は, カリフォルニア大学バークレイ校において開発が行われている UNIX を基盤としたソフトウェア開発/研究用のオペレーティングシステムである. ネットワーク機能の実現という視点から見た 4 BSD の一般的な特徴は, デマンドページングの機能が含まれているので, パケットを取り扱うために必要なカーネル空間での動的なバッファの割り当てを効率的に行うことができる点と, 端末処理機能を疑似装置として取り扱う pty の機能である.

* BSD (Berkeley Software Distribution) 第4版の意味. 4 BSD には 4(.0) BSD, 4. 1 BSD, 4. 2 BSD, 4. 3 BSD とバージョンの変遷があるが本解説では, 特に指定しない限りこれらの総称, あるいは最新版である 4. 3 BSD を示すことにする.

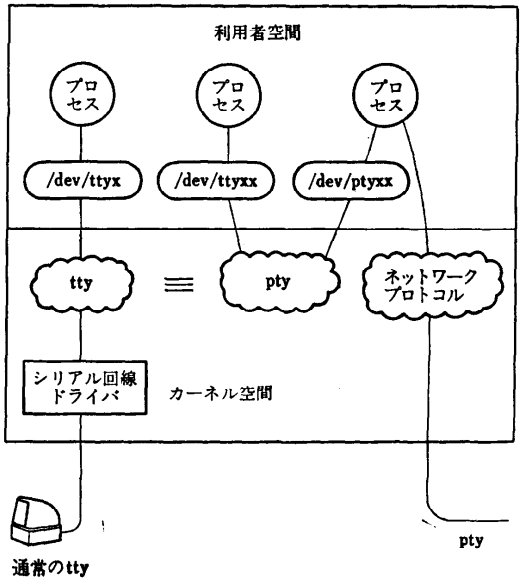


図-2 pty の構造

pty は双方向バイトストリームのプロセス間通信を行うための機能だがエコー処理, 行単位文字単位の消去処理, 制御文字による割り込み処理などの端末文字列処理の機構が組み込まれている点に特徴がある. これによりネットワークを介した遠隔ログインをシリアル回線からの端末による局所的ログインと全く同様に扱うことが可能になっている. pty の構造を図-2 に示す.

4 BSD は 1981 年に配布を開始した 4.2 BSD からカーネル構造を大幅に改造した. この改造の中にはカーネル空間におけるネットワーク通信機能の実現, あるいは分散オペレーティングシステムへの発展を目的とした箇所が多い.

5.1 4 BSD のネットワーク構造

ネットワーク機能の設計に階層的プロトコル構造を用いることの利点は, 各層を独立して取り扱うことができるので論理的仕様の変更, 物理的変更をいずれも全体への影響を考えずに実現することができ, 設計も容易になる点である. オペレーティングシステムにおけるネットワーク機能の実現にもこれらの特徴が生かされる構造を確立し, かつ, 高い効率を提供する必要がある. そのためにはオペレーティングシステムの構造とネットワーク機能の構造の対応を考慮し設計しなければならない.

4 BSD のネットワーク構造はこの問題の解決に重

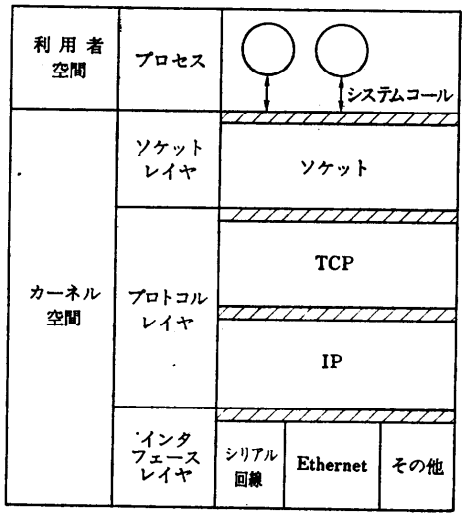
点をおいて設計されており，ネットワーク論理構造に対応した各層のモジュール性，プロトコルモジュールとハードウェアインタフェースの独立性を実現するための技術が導入されている．

4.2 BSD 以降の 4 BSD UNIX のネットワーク構造の設計の特徴をまとめると次のようになる．

（1） カーネルでは，プロセスのファイル入出力と互換性のあるシステムコール，socket と呼ばれるプロセス間通信機能を提供している．この機能はシステム内のプロセス間通信と，プロトコルモジュールを介したネットワークによるプロセス間通信に利用することができる．

（2） セッション層以下のプロトコルモジュールはカーネルに組み込む．組み込まれたプロトコルモジュールは，複数の層から成り立っていてもよく，そのそれぞれは UNIX のデバイスドライバと同様に独立したモジュールとして取り扱われる．そのため，新しいプロトコルの付加，層単位の変更はほかの部分に影響を与えずに独立して行うことができる．

（3） 通信用ハードウェアインタフェースのドライバは，既存の UNIX デバイスドライバと異なった方法で定義される統一的な仕様で記述される．これにより，ハードウェアインタフェースはそのハードウェアを使用するプロトコルに依存せず設計することができる．



▨ 定義されている層間インタフェース

図-3 4BSD のネットワーク構造

クライアント側		サーバ側
socket	ソケットの解消	s : socket
bind	名前の定義	bind
	受信準備	listen
connect	接続要求	accept
	接続受諾	accept
(通信	read, write, sendto, receive from)	
close	接続解除	close

図-4 Socket を使ったプログラムの例

この構造を示したのが図-3 である．

5.2 カーネル基礎機能

上記に示した構造面での特徴はカーネルが提供するいくつかの機能に基づいて実現されている．

5.2.1 socket

socket は UNIX のプロセス間通信を提供するオペレーティングシステム基礎機能である．プロセスが socket を用いてプロセス間通信を行うためには，（1）プロセス間通信を行うための端点となる実体，**ソケット**，を確保し，（2）そのソケットが属する通信ドメイン内で一意に定まる名前を付け，（3）ソケットに付けられた名前を用いて相互に通信を開始する，という手順を用いる．この手順を用いたプログラムの例を図-4 に示す．

構造的に見ると socket は，UNIX のプロセスがファイルと入出力を行う機構と同じデータ構造を用いて実現されている．したがって open システムコールによって準備される**ファイル記述子**の取り扱いと socket システムコールによって獲得され，所定の手続きを完了したソケットの取り扱いはほぼ同様である．このことは，fork システムコールによって生成された子プロセスに親プロセスが確立した入出力経路が伝承されるという点で大きな意味を持つ．

socket システムコールは，その名前付けのドメイン，型，プロトコルという三つの引数をともなって呼び出される．

s＝socket（ドメイン，型，プロトコル）

ドメインには自システム内のプロセスとの通信，DARPA インタネットプロトコルを用いているプロセスとの通信，XEROX ネットワークシステムを用いているプロセスとの通信，にそれぞれ対応した定数を指定する．これによって，それぞれのプロトコル規約で定めたプロセス間通信の端点のアドレス付けの約束に従ってサービスプロセスなどの作成をすることができ，UNIX 以外のシステムとの通信を実現することも可能となる．

型はストリーム型（バーチャルサーキット型），デー

タグラム型などの通信形態を表現し，プロトコルはUNIX のカーネル内に組み込まれているプロトコルモジュールの識別子を記述し，そのソケットが使用するプロトコルの指定に使用される．

これらの仕様により 4 BSDUNIX 上のプロセスは，システム内外のプロセスとの通信が可能となり，必要なプロトコルの選択，そのプロトコルにおける通信端点のアドレス付け，などの機能によりオペレーティングシステムに依存しない通信をも実現している．

5.2.2 プロトコルモジュール

4 BSD では階層的プロトコルの処理を行うモジュールはカーネル内に組み込まれている．各モジュールの組み込みは，通常のデバイスドライバと同様に，モジュール内の関数の集合の種類をオペレーティングシステムのレベルで統一し，各関数のエントリポイントをオペレーティングシステム内のテーブルに格納することにより動的なモジュール間の接続が可能なようになっている．

このテーブルは各プロトコルモジュールごとに一つ存在し，隣接しているプロトコルとのインタフェースに使用されるテーブルと，ソケットの機能と接続のために使用されるテーブルとからなっている．このテーブルに登録される情報には図-5 のような項目がある．これらのうち，興味深いのは二つのタイムアウト処理のためのエントリである．プロトコルモジュールを実現するためにはタイムアウト処理は避けられない．このため 4 BSD では従来の UNIX のようにクロックの割り込みをプロセスのコンテキスト切り替えのみに使用するのではなく，一般的なカーネル内でのタイマ処理を始末するように変更している．ネットワークモジュールもこれを利用して，テーブルにタイムアウト関数のエントリポイントが指定されている場合には高速および低速タイムアウト関数をそれぞれ 200 ms ごと 500 ms ごとに呼び出すように設計されている．したがってプロトコルの作成者は呼びだされる間隔に応じて適当なタイムアウト処理を行うようにプロトコルごとに設計することができる．

このモジュールの属性に関するもの：
- 適合するソケットの型
- 提供する通信の種類
- プロトコル識別子
- プロトコルのグループ識別子

隣接プロトコルとのインタフェース：
- 下層プロトコルからの受信関数
- 上層プロトコルからの受信関数
- 下層プロトコルからの制御関数
- 上層プロトコルからの制御関数

ソケット機能とのインタフェース：
- ソケット機能からの要求を処理する関数

その他の関数：
- プロトコルモジュールの初期化
- 高速タイムアウト関数
- 低速タイムアウト関数
- 空間解放の関数

図-5 プロトコルモジュールの定義項目

5.2.3 インタフェースモジュール

4 BSD ではネットワーク通信に用いるハードウェアインタフェースとプロトコルモジュールとの独立性を実現するために，ハードウェアインタフェースのドライバの取り扱いを規格化するためのテーブルを持っている．カーネル内の最下層プロトコルモジュールはこのテーブルを介してハードウェアのアクセスを行う．

このテーブルによる定義はカーネル内の最下層プロトコルモジュールが利用する機能を一般的に定義したもので，図-6 のような項目からなっている．

5.2.4 経路制御

発信パケットまたは通過パケットの経路決定はネットワーク層から呼び出される経路管理モジュールによって処理される．したがって経路管理モジュールとそこで利用される経路制御テーブルはカーネル空間内に

対応する一般的なインタフェースの属性
- インタフェースの論理名
- MTU* (このインタフェースの最大転送単位)
- インタフェースの型

プロトコルに依存したアドレスの情報**
- 上位レベルプロトコルがこのインタフェースに指定したホストアドレス
- ブロードキャスティング型の場合はそのためのアドレス
- ポイント-ポイント型の場合は相手のアドレス

出力データの待ち行列

取り扱い関数
- 初期化関数
- 送信関数
- ioctl (制御関数)
- リセット関数
- タイマ関数

統計情報
- 受信パケット数
- 受信パケットエラー数
- 送信パケット数
- 送信パケットエラー数
- CSMA 型インタフェースの衝突数

図-6 インタフェースの定義項目

★ Maximum Transmission Unit

★★ これらのアドレスはインタフェース初期化の際に動的に書き込まれるので結合されるプロトコルとの独立関係は保たれていることになる．

存在する．経路管理の主な機能は次のとおりである．

（1） 与えられたアドレスへの経路を検索しゲートウェイを経由する場合はそのゲートウェイのアドレスを発見する．これによって同じネットワーク内での目的ホストが決定される．

（2） 得られた経路が複数存在する場合は最適な経路を選択する．現在の 4 BSD では通過パケット数が最も少ない経路を選択するアルゴリズムになっている．

（3） (1), (2)で得られたホストに到着するために使用するインタフェースモジュールを選択する．

カーネル内の経路制御テーブルの情報はネットワーク間結合の状態の変化に対応して遷移すべきである．各ホストに分散されたこのような情報を動的に変化させ，かつ矛盾がないようにするためには複雑な機構が必要となる．4 BSD では経路制御情報の現実的な管理を実現するためにこのテーブルを修正するシステムコールを用意し，利用者空間のプログラムから変更，更新を行うように設計してある．このようなプログラムの一つとして XEROX 社の経路情報更新プロトコル [Xerox 81] に基づいた routed が用意され，動的なネットワーク間結合の状態変化に対応するようになっている．

6. ネットワーク機能の実現

前に述べた UNIX ネットワークの応用機能は UUCP や socket を用いたプロセス間通信によって実現されている．一般に UNIX に組み込まれているプロトコルは米国の防衛先端技術計画局（DARPA）によっ提唱されている「インタネットプロトコル」[Crocker 72] と XEROX 社で提唱されている「XEROX ネットワークシステム」[Xerox 85] が組み込まれている．

6.1 メッセージ交換の機能

UNIX での電子メイル機能は，利用者インタフェース，メイル配送プログラム，通信プログラムの三つの層からなる構造を持っている．利用者インタフェースは電子メイルの送信，受信，整理などの目的に直接利用者によって使用されるプログラムによって構成されている．最も基本的なプログラムとしては「ucbmail」と呼ばれている /usr/ucb/mail が提供されているが，整理の点で優れた MH システム [Borden 79] が実際には多く利用されている．電子メイル送信の際には利用者インタフェースのプログラムは電子メイル本体に定められた書式のメイルヘッダを付加しメイル配送プログラムに引き渡す．ここで用いられている書式は原則として ARPA の「標準インタネットテキストメッセージ」[Crocker 82] に従ったものである．

配送プログラムは *sendmail* [Allman 83] と呼ばれ，次の処理を行う．

・利用者インタフェースによって作成されなかった標準ヘッダの要素の作成．

・標準ヘッダに含まれる宛先アドレスの正規表現への書き換え．

・正規表現となったアドレスの解釈と，通信プログラムの選択．

・選択された通信プログラムに適合したアドレスなどの書き換えと通信プログラムの引数の作成．

・通信プログラムへの電子メイル本体，ヘッダの引渡しと起動．

メッセージのアドレス表現はネットワークごとに異なっており，使用可能な通信プログラムの種類もシステムによって異なる．そのため，配送プログラムのこれらの処理は外部から与える規則に従って行うように設計されている．

通信プログラムとして標準的に使用されるのは次の 4 つである：

（1） /bin/mail
　自システムへの配送
（2） UUCP
　UUCP を用いた配送
（3） SMTP
　ARPA の SMTP [Postel 82] を用いた配送

UNIX におけるメッセージ交換に関する構造を総合的に示したのが図-7 である．

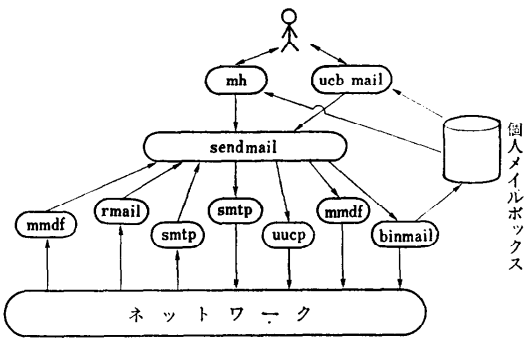


図-7 Unix のメッセージ交換機能

6.2 ホストの存在を意識するネットワーク機能

分散環境においてオペレーティングシステムが解決しなければならない問題は資源の名前付けに関する取り扱いである．独立稼動を前提としたオペレーティングシステムはそのシステム内の資源に一意に定まる名前またはアドレスを定義している．ネットワークで結合されたシステムの資源を互いにアクセスするためには全体を一つの定義域とした名前空間が形成される必要がある．このためには位置に関する情報を隠ぺいすることによって名前空間を形成する方法と各システム自体に名前（ホスト名）を付け，ホスト名と資源名の二つの階層を用いて名前空間を定義する方法がある．

提供される環境の面では前者が優れているが構築技術の面から見れば前者は環境全体の設計と開発が必要なのに対して後者は個々の計算機システムを独立に設計することが可能であることから単純な機構で実現でき，拡張性，信頼性を容易に得ることができる．4 BSD の提供するネットワーク環境は基本的にホスト名を意識して通信機能を利用するもので，分散環境を提供するためには機能の付加，改造が必要である．

4 BSD の応用機構は2種類のモデルに基づいて構築されている．一つはブロードキャスト型の通信に基づいたもので，システム状態に関する情報の配布を行う目的などに使用されている．もう一つのモデルはクライアント／サーバモデルと呼ばれている形式で，サービスを要求するプロセスと提供するプロセスの役割を定義し，応用機能を実現する．実際の応用機能はほとんどが後者の形式で構築されている．4 BSD で提供されているネットワーク機能を表-1 に示す．

6.2.1 クライアント／サーバモデル

クライアント／サーバモデルはサービスを提供する

表-1 4BSD のネットワーク機能

ftp	DARPA ファイル転送
telnet	DARPA リモートログイン
rcp	ファイル転送
rlogin	リモートログイン
rdump	リモートダンプ
rwho	システム情報
ruptime	〃
lpr	プリンタスプーラ
rsh	リモートコマンド実行
routed	経路制御
named	ネームサーバ
timed	時刻調整機構

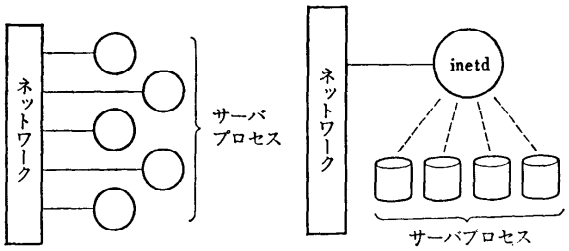


図-8 サーバの構造

サーバとサーバにサービスを要求するクライアントの二者による単純なマスタ／スレーブ関係の通信方式である．サーバとなるプロセスの起動方式にはシステムのブート時に起動される場合と要求受付時に起動される場合と二とおりある．

前者は 4.2 BSD で用いられた方式で，あるシステムで提供するサービスを担当するプログラムはそれぞれブート時に起動されるように設定しておき，システムが稼働している限り要求受付待ち状態で待機する．後者はシステムに登録されているすべてのサービスを一つのプロセス，inted がいったん受け付け，対応するサーバプログラムを起動し，そのプログラムにクライアントからの呼そのものを引き渡す方式である．この方式は，4.3 BSD や SUN マイクロシステムズ社の 4.2 BSD で用いられている．これらの違いを示したのが図-8 である．

クライアントからの要求を受け付けるために，各サービスにはプロトコルごとに定義される．ポート番号が割り当てられており，各サーバは自分のポート番号において要求を待つことになる．4 BSD で標準に用いられている DOD のプロトコルでは [Reynolds 85] において主なサービスのポート番号が定義されているのでこれらを用い，UNIX 独特のサービスに関してはこの規準内で未定義になっている番号を割り当てて用いている．

inetd が代表的に待機する場合と各サーバが独立に待機する場合のいずれの場合も，待機しているプロセス自体が要求の受け付けからサービスの処理までを行うことはサービス処理中に発生する要求に対応できないために望ましくない．このため，要求を受け付けた待機プロセスは fork システムコールを発生し，それによって生成した子プロセスにサービスの処理を担当させている．これによって親プロセスは待機状態に速やかに復帰できる．この方式は 4 BSD におけるプロ

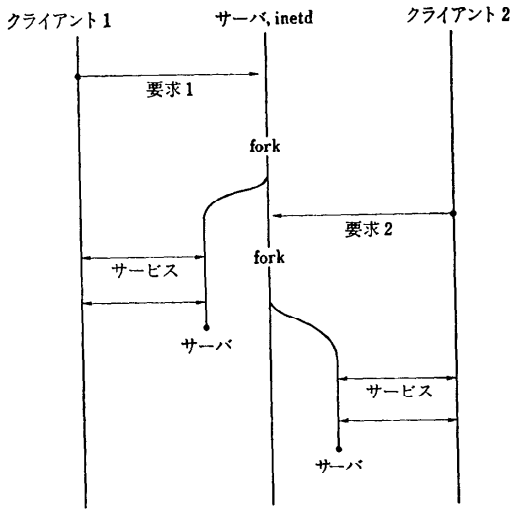


図-9 サーバとクライアントの関係

セス間基本機能である socket 関連の情報が fork によって子プロセスに引き継がれる特徴を利用している．このような処理の構造を図-9 に示す．

6.3 分散環境

UNIX ではプロセスとその入出力の対象となるファイルとの関係としてほとんどの処理が定義される．そして，各ファイルに一意の名前を定義する空間を提供するのが UNIX のファイルシステムである．したがって，分散環境の組み立てが UNIX において議論される場合にはネットワークを用いたファイルシステムの拡張という形をとる場合が多い．個々のシステムは独立した木構造によるファイルシステムを持っている．これらを分散環境としてとらえるためには各ファイルシステムを仮想的に結合した空間を提供する機構が必要となる．これが UNIX における分散ファイルシステムの概念である．このような分散ファイルシステムの技術として大きく二つの代表的な方式が存在する．

最初の方式は独立した各ファイルシステムにおける木構造のそれぞれの根（ルート）を要素とする一つの仮想的なディレクトリ，「全域的ルート」を設定する方式である [Rowe 82] [Brownridge 82] 一般的にこの方式ではファイルパス名に特殊な解釈を行い，各システムの独立性を保存しつつ遠隔ファイルシステムのアクセスが透過的に実現できるようになっている．たとえば，[Brownridge 82] では，あるファイルを含んでいる親ディレクトリへのリンクの名前として用いられる「…」が，各ファイルシステムのルートにおいては全域的ルートを示すという規則を設けることにより，図のような全域的構造を実現している．ここでは「／…／ホスト名／」で始まるパス名のファイルへのアクセスは，ネットワークアクセスにより該当ホストのファイルサービス機能の呼び出しを起動することになる．

この方式の特徴は，全域的に一意に定まるファイルの名前空間を提供することにある．したがってファイル名を認識する主体が各システムに分散するような場合，たとえば，実行中プロセスの移動などを必要とする分散処理の形態を想定するとこの特徴の利点は大きい．反面，利用者は常にパス名の表現としてファイル

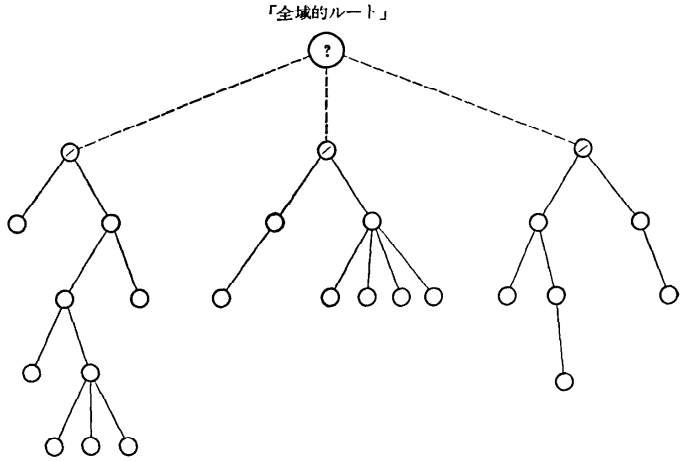


図-10 「全域的ルート」方式

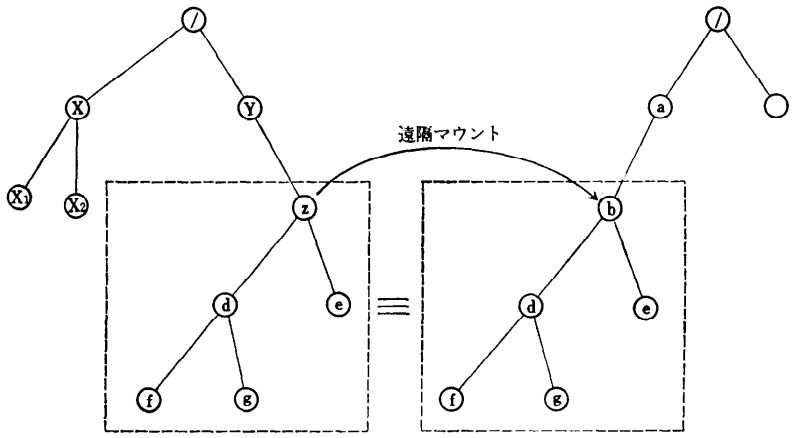


図-11 遠隔マウント方式

の位置を意識する必要がある．この方式によるファイルシステムの結合を図-10 に示す．

これに対してディスクの論理ボリュームを結合し，ファイルを結合するためのシステムコール *mount* を拡張した形で遠隔ファイルシステムの部分木を結合する方式が遠隔マウント方式である(図-11)．遠隔マウント方式の特徴は，各システムごとに結合が定義されるために全域的に一意に定まる名前空間を持たないことと，物理的に一つの部分木を環境内で共有することができる点にある．この方式は独立した既存の UNIX との整合性に優れているために非常に広く利用されているのが現状である [Sandberg 85] [Rifkin 86].

7. UNIX ネットワークの現状と今後

UNIX のネットワーク機能は既存の環境を拡張していきながら分散環境を提供する方向に発展していることが特徴である．オペレーティングシステム構造からみた通信技術の取り込みは 4 BSD の *mbuf* や System V の *stream* [Ritchie 84] といったメモリ管理に関する機能を利用することによって効率的な処理がカーネル内で実現できるようになっている．そのため，分散環境を提供するための技術の発展は，おもに利用者空間上のプロセスを利用した設計によって実現していくことが可能となった．そのために，分散環境の構築，分散処理の実験などの目的で実用的に利用されはじめている．

UNIX オペレーティングシステムは柔軟な環境を提供しているためにソフトウェア開発や流通の共通基盤として利用されているのが現状である．本解説で示したように通信面での柔軟性が提供されるようになったことによって，各種の計算機資源が存在する研究開発環境においての基幹ネットワークのためのシステムとしての役割を UNIX に期待することが可能となった．

参 考 文 献

[Allman 83] Allman, E.: SENDMAL—An Internetwork Mail Router, Unix Programmer's manual, CSRG U.C. Berkeley (July 1983).

[AT & T 86] Interprocess Communication, UNIX System V Release 3 Programmer's Guide, Chapter 9, AT & T (1986).

[Borden 79] Borden, S., Gaines, R.S. and Shapiro, N.Z.: The MH Message Handling System: User's Manual, Rand Corporation (Oct. 1979).

[Brownridge 82] Brownbridge, D.R., Marshall, L.F. and Randell, B.: The Newcastle Connection or UNIXes of the World Unite!, Software-Practics and Experience, Vol. 12, pp. 1147-1162 (Dec. 1982).

[Crocker 72] Crocker, S.D., Heafner, J.F., Metcalfe, R.M. and Postel, J.B.: Functionalorienteed Protocols for the ARPA Computer network; Proceedings of Spring Joint Computer Conference, AFIPS, p. 271 (1972).

[Crocker 82] Crocker, D.H.: Standard for the ARPA Internet Text Messages, RFC 822, Univ. of Delaware (Aug. 1982).

[Reynolde 85] Reynolds, J. and Postel, J.: Assigned Numbers, RFC 960 (Dec. 1985).

[Rifkin 86] Rifkin, A.P., Forbec, M.P. and Hamilton, R.L.: Remote File Sharing Architectural

Overview, Proceedings of USENIX Summer Conference, USENIX Associations (June 1986).
[Rowe 82] Rows, L. A. and Braiman, K. P.: A Local Network Based on the UNIX Operating System, IEEE Trans. on Software Engineering, SE-8, No. 2 (Mar. 1982).
[Leffler 83] Leffler, S. J., Joy, W. N. and Fabry, R. S.: 4.2 BSD Networking Implementation Notes, Unix Programmers' Manual, Vol. 2c, CSRG, University of California, Berkeley (July 1983).
[Postel 82] Postel, I. B.: Simple Mail Transfer Protocol, RFC 821, USC/Information Sciences Institute (Aug. 1982).
[Ritchie 84] Ritchie, D. M.: A Stream Input-Output System, AT & T Bell Laboratories Technical Journal, Vol. 63, No. 8, pp. 1897-1901 (Oct. 1984).
[Sandberg 85] Sandberg, R. et al.: Design and Implementation of the Sun Network Filesystem, Proceedings of USENIX Summer Conference, USENIX Associations (June 1985).
[Xerox 81] Routing Information Protocol,"Internet Transport Protocol, XSIS 028112, XEROX Corp. (Dec. 1981).
[Xerox 85] Xerox Network System Architecture General Information Manual, XNSG 068504, XEROX Corp. (1985).

(昭和 61 年 10 月 24 日受付)