記　号　処　理 72－4
（1994．1．14）

記号処理計算機 SILENT のネットワークアーキテクチャ

村上健一郎

NTT 基礎研究所

本論文では、記号処理計算機 SILENT とその記号処理言語 TAO 上に実現されるネットワークアーキテクチャについて述べる。そして、基本的な設計思想をソフトハード両面について明らかにする。TAO のネットワークは、オブジェクト指向によるストリームの実装を基本とする。ここでは、ストリームは受動オブジェクトとなり、イベント割り込み、および、ユーザプロセスからのメッセージによってストリーム上で処理が進められる。処理にあたっては、パケット予測、メモリアクセスの抑制と効率化、プロセススイッチの抑制などによってオーバヘッドを減らすことをねらっている。また、2つのストリームオブジェクトを密結合してバッファを共有化し、これで効率のよいパイプ機構を実現することもねらいの一つである。一方、ハードウェアでは、SILENT プロセッサが直接ネットワークインタフェースを制御しており、フロントエンドプロセッサによるオーバヘッドを避けている。

Network Architecture in TAO/SILENT

Ken-ichiro Murakami

NTT Basic Research Laboratories

This paper describes a network architecure built on top of the TAO language and the symbol processing machine SILENT. The TAO/SILENT system is a dedicated real-time symbolic processing kernel which applies AI to real world problems.

The network system designed is based on the connection oriented implementation model employing object oriented programming paradigm. On a network stream object, several processes run simultaniously and perform operations on its packet buffer. The network system improves its performance by a packet prediction technique, a memory access and process switch suppression technique, and a shared buffer implemented by pipelined stream objects. In terms of the hardware, network interfaces are controlled directly by the SILENT CPU. This results in reduced processing overhead compared to indirect operations through Front-End-Processor.

1. はじめに

ネットワークシステムは、リアルタイム性、プロトコルの階層性、多重処理などに特徴がある。近年、これを並行オブジェクトによって実装するいくつかの試みがなされてきた[1][2][3]。しかし、このような実装を行う場合、マシンアーキテクチャやオペレーティングシステムによるサポートが、効率的な開発やシステムの効率を左右する。現在開発中の記号処理計算機SILENTでは、その豊富なプロセス制御プリミティブやリアルタイム性を利用し、オブジェクト指向をベースとしながら、高速ネットワークにも耐えられるような効率の良い実装を目標としてネットワークの設計を行っている。

本論文では、記号処理計算機SILENTとその記号処理言語TAOをベースとしたネットワークシステムについて説明する。そして、基本的な設計思想をソフトハード両面について明らかにする。TAOのネットワークは、オブジェクト指向によるストリームの実装を基本とする。ここでは、ストリームは受動オブジェクトとなり、イベント割り込み、および、ユーザプロセスからのメッセージによってストリーム上で処理が進む。処理にあたっては、パケット予測、メモリアクセスの抑制と効率化、プロセススイッチの抑制などによってオーバヘッドを減らす。また、ストリームオブジェクトを密結合してバッファの共有化を行い、効率のよいパイプ機構を提供する。一方、ハードウェアでは、SILENTプロセッサが直接ネットワークハードウェアを制御しており、フロントエンドプロセッサによるオーバヘッドを避けている。

2. TAO/SILENTシステムの概要

記号処理カーネルTAO/SILENTは、専用VLSIを中心としたタグアーキテクチャマシンSILENTと、その上の豊富な記号処理プリミティブとコンカレンシプリミティブを備えた機械語TAOから構成される[4][5]。

SILENTは、8ビットタグと32ビットポインタを1語とするタグアーキテクチャの専用VLSIチップ(SILENTチップ)を搭載したボードコンピュータである(図1)。タグ部、ポインタ部それぞれが専用のALUをもっており、ポインタ部の比較や演算と同時に、タグ部分の比較や、タグ生成ができる。SILENTチップは、2語からなるセルを一度に読み書きできるよう、80ビット幅のシステムバスを通して、主記憶、浮動小数点演算器TARAI等と通信する。SILENTチップ制御のマイクロコードを収めるWCS、スタックメモリはそれぞれ専用の高速メモリチップを用いる。どちらも主記憶のアドレス空間にアドレスをもつが、SILENTチップ直結のバスも備えている。つまり、主記憶、マイクロコード、スタックを同時にアクセスできる3バスアーキテクチャとなっている。

Ethernet、FDDI(Fiber Distributed Data Interface)、ISDNなどのネットワークインタフェースは、32ビットデータ幅のI/Oバスに接続されている。そして、I/Oバスは、バスカプラを通してシステムバスに接続されている。これらのインタフェースは、I/Oバス上のメモリをバッファとして使用する。なお、SILENTは、I/Oバス上を転送されるパケットをスヌープする特別なLSIを持ち、DMA中にチェックサムなどの前処理が行えるように設計されているが、この機構については別の機会に報告する。

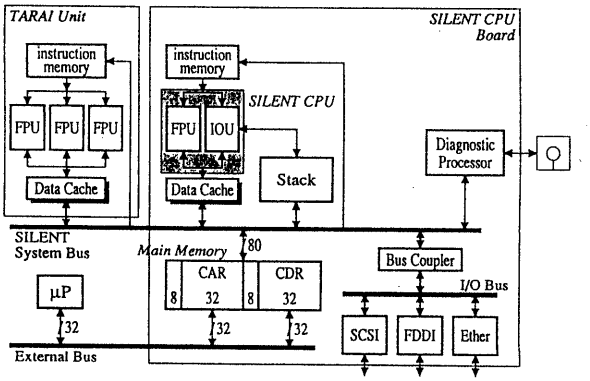

図1. SILENTハードウェア

TAOはこのSILENTの核言語である。豊富な記号処理プリミティブとコンカレンシプリミティブを備え、実時間処理に対処している。機能的には、TAO/ELIS[6]で提唱したLisp、オブジェクト指向、論理型のパラダイム融合をより深め、さらに、これら記号処理とTARAIチップによる実世界向き3次元数値処理との統合を狙っている。また、SILENTを新たなネットワークプログラミングパラダイムやネットワークのアルゴリズムの実験を行うテストベッドとして使用することも狙っている。実装するプロトコルは、事実上の標準TCP/IP(Transmission Control Protocol/Internet Protocol)プロトコル群である。将来的には、このネットワークを大規模システムの自己安定性や自己組織化といったヘテロ並列システムに

よる機能の質の向上をねらった計算原理の探究のためのプラットフォームとしても使用する予定である。

3. ネットワーク実装のポリシー

TAO/ELIS のネットワークシステム [1] では、オブジェクト指向による実装方式を確立することに主眼をおいて実装を行った。これは、バーチャルサーキットを能動的オブジェクトとするものである。TAO/SILENT の実装では、これをより洗練されたものとし、オブジェクト指向に求められる機構を明らかにすることと、最適化を行って 100Mbps オーダの高速ネットワークにも耐え得るものとすることの 2 つを目標としている。

そして、具体的には、以下のような目標の下に設計を進めている。

- チェックサムやコード変換、メモリコピーなどのメモリアクセスを最小限に押える
- プロセススイッチを最小限に押える
- ヘッダ予測などの統計的な手法を利用してパフォーマンスを向上させる
- ストリームオブジェクトの融合や再構成が動的に行える柔軟性を確保する
- ネットワークで使用されるさまざまな発見的手法をストリームオブジェクト内の知識として制御機構から明確に分離する

4. 実装方式

4.1 ベースとなるネットワーク実装方式とその問題点

図 2 に TCP/IP のプロトコル階層、図 3 にコネクション指向実装モデルのクラス階層を示す [1]。各クラスは、それぞれプロトコル階層に対応した抽象クラスとなっており、これらを多重継承したクラスのインスタンスとしてコネクションオブジェクトが生成される。(プロトコル階層とクラス階層が逆転していることに注意。) このオブジェクトにデーモンやユーザプロセスからメッセージが送られ、プロトコル処理が行われる。この実装によって、システムの拡張や変更 (例えば、マルチプロトコル化) が柔軟に行える。また、基本ストリームのクラスを継承することにより、他のストリームと同様にネットワークストリームへも高速なアクセスが可能となっている。

しかしながら、いくつかの問題もある。まず、独立して作成された他のストリームとの効率の良い結合ができないことである。例えば、FTP (File Transfer Protocol) では、TCP のストリームとファイルストリームを結合しなければならないが、この間のデータ転送は、別のエージェントを介する方法しかない。即ち、バッファの共有ができないためにコピーというオーバヘッドの大きい手段を利用せざるを得ない。

次の問題は、オブジェクトが生成されたあとにスーパクラスを変更することができないため、予想されるスーパクラスをすべて継承しなければならない点である。しかし、そのメソッド名が同じであれば、それも不可能である。例えば、Ethernet と FDDI の両方に接続されているようなホストでは、ドライバに相当する Ethernet の抽象クラスと FDDI の抽象クラスを用意する。しかし、それらはインタフェースを標準化するために同じメソッド名を持つので、両方の継承は困難である。実際のネットワークでは、FDDI を通ってくるルートが障害になっても Ethernet を通ってくるような冗長な接続が可能である。FDDI でコネクションを確立した後に、障害が発生すると Ethenet 側に経路を切り替える場合がある。ところが、これを行うためには、ストリームのスーパークラスを FDDI から Ethernet に動的に変更しなければならない。

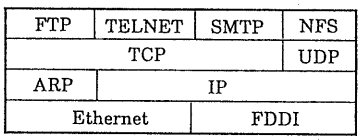

図 2. TCP/IP プロトコル群の階層

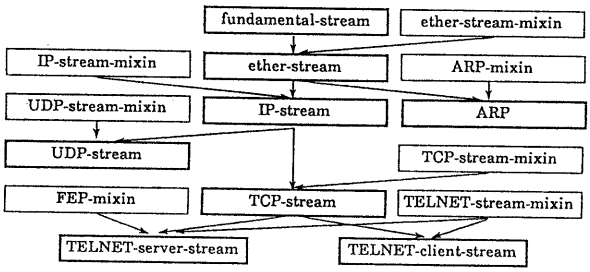

図 3. クラス階層

4.2 密結合されたオブジェクト – Fat Pipe

ファイル転送では、ネットワークから入ってきたデータを処理し、内部形式に変更したあとにディスクに書き出すというパイプライン処理が必要である。高速なパイプ (Fat Pipe) を実現するためには、それぞれのストリー

ムオブジェクト内にある基本ストリームのインスタンス(バッファやポインタなど)を共有すれば良い。その一つの方法は、ファイルとネットワークのクラスを継承するようなクラスを作ることである。しかし、この方法では、あらゆる組み合わせを考えなければならない。そこで、各オブジェクトが共通に持っている基本ストリーム部分のスロットを共有してしまう。これを密結合オブジェクト(図4)と言う。

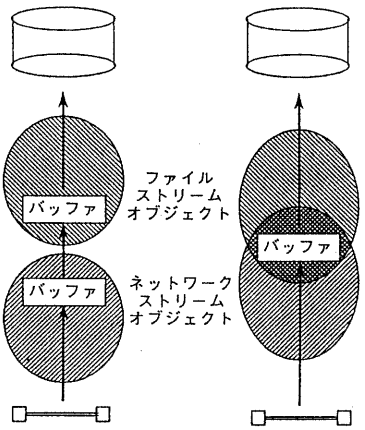

図4. 疎結合オブジェクトと密結合オブジェクト

密結合オブジェクトでは、バッファが共有されるので、データのコピーを避けることができる。この効果は、高速なネットワークではより顕著となる。それは、高速のネットワークが大きなサイズのパケット(例えば、FDDIは4352バイト)を使用しているためである。もちろん、パイプライン処理がブロックされないように、バッファにはリングバッファを使う必要がある。このようなFat Pipeは、ファイルとネットワークストリームにだけ必要なものではない。ネットワークのループバックインタフェースでは、ネットワークストリーム同士が密結合される。また、ルータでは、異なるインタフェースを使用するストリームを密結合してパケットの中継を行う。つまり、密結合の機構は、汎用的に利用できるものである。

4.3 動的なオブジェクトの再構成

ネットワークのストリームオブジェクトを生成する時には、相手のホストがどのインタフェースを通じて通信できるのかをチェックする。この際、ルーティングテーブルを使用して、最も近い距離(距離として、中継段数、速度、ディレイなどのさまざまなパラメータを考慮した値が使用される)となるような経路を選択する。例えば、FDDIのインタフェースとEthernetのインタフェースの両方から到達可能で、FDDIインタフェース側の経路の距離が近ければ、これを選択する。そこで、FDDIの抽象クラスを継承したインスタンスを生成する。ところが、通信途中で、この経路に障害が起きれば、Ethernet側の経路を使用して通信を継続しなければならない。

そこで、以下のようなメタな機構が必要となる。まず、ルーティングテーブルを管理しているルーティングデーモンは、障害を検出すると、その経路を使用していたストリームオブジェクトのインタフェース部分を無効とする。一方、ストリーム側では、パケットを転送しようとした時に、メソッドが無いことがわかり、missing-method handlerというメタなメソッドが起動される。このメソッドでは、ルーティングデーモンとの通信で、Ethernetならば通信が続行できることを知る。そして、デレゲーションによって、あたかもスーパクラスとしてEthenetを継承しているように振舞う(図5)。

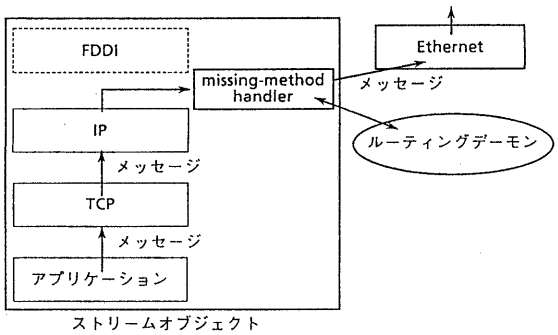

図5. missing-method handler

4.4 ネットワークの制御機構と知識の分離

TCP/IPには、2400bit/secのような低速からサブGbit/secのような高速に至るまでの非常に広い適応性が要求されている。このため、パケットの送信には、多くの発見的手法が用いられている。しかも、パケットの往復時間を観測することによってネットワークの状態を推定し、これに応じて動的に送信のアルゴリズムを変更する。

例えば、telnetで遅い回線の帯域を有効に利用するためには、パケットに載せるデータ量をできるだけ多くする必要がある。しかし、いつまでもバッファにためておくわけにもいかない。全二重方式では、クライアントか

ら転送しない限り、当然、サーバからのエコーバックも得られないからである。そこで、直前にタイプしたキャラクタのエコーバックがサーバから送られてきた契機で貯めていたキャラクタを転送する。つまり、エコーバックに対する確認応答(ACK)と新たなデータをピギーバックする(Nagleのアルゴリズム)。実際には、これ以外にも多くのアルゴリズムが適用されるので、実装は非常に複雑なものとなる。

そこで、ネットワークの制御機構から知識をルールとして分離することによってこの問題を解決する。ルールは、各ストリームオブジェクトで状態(例えば、応答時間など)や条件が異なるので、オブジェクトに閉じたものでなければならない。また、ルールの変更や拡張を行う場合にも、ルールがオブジェクトに閉じるほうが好ましい。

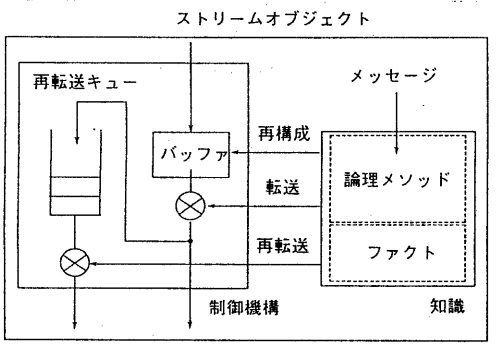

図6. 制御機構と知識の分離

各クラスでは、それに特有のルールを定義し、インスタンスは、そのルールを継承しなければならない。SILENT上のネットワークではこの部分を論理メソッドで記述する予定である。この場合、相手のホストの応答時間などは、各インスタンス(ストリームオブジェクト)に固有のファクトとなる。なお、ユニフィケーションによって決定されるのは、例えば以下のような事項である。

(1) 実際にパケットを送るか、空振りしてバッファに溜めておくか

(2) 複数のパケットが受理されていない場合、再転送で、バースト的に転送するか、インターバルをあけるか

(3) 相手が受理したデータだけをバッファから除いてパケットを再構成するかどうか

(4) 受理を示すACKパケットを直ちに送るか、あるいは次のパケットに対するACKまで待つか(delayed ACK)

† ICMP source quenchによるパケット生成の抑制要求が原因の場合もある

5. 高速ネットワークへの対応

ネットワーク実装上のオーバヘッドは、メモリコピーやチェックサム計算などによるメモリアクセス、リスト化されたバッファの操作、プロセススイッチ、ストリームへの受信パケットの振り分けなどを原因としている†。細分化された短いパケットを多量に処理しなければならない場合には、ヘッダ部分の処理が問題になり、逆に、ファイル転送のような長いパケットの場合には、データ部分の処理が問題となる。即ち、さまざまな場合に適応できるためには、両方に対応する必要がある。

5.1 パケット予測とデーモンプロセス

受信したパケットのプロトコル処理でオーバヘッドとなり得るものとしてTCPにおけるストリームへの分離がある。この場合、ヘッダに埋めこまれているストリーム識別子(これは、TCPのリモートポート番号、ローカルポート番号、相手のIPアドレスのペア)からストリームへのハッシュが必要となる。また、このハッシュの時間が問題とならなくとも、その次のステップである受信したパケットを該当するストリームのバッファへコピーあるいはスワップするオーバヘッドもある。

そこで、TAO/SILENTの実装では、デーモンエージェントを使用せず、次に到着するパケットのストリームを予測する。そして、そのストリームのバッファにパケットをいきなり受信する。その後、ヘッダをチェックし、予測がミスヒットした場合には、それを本来受け取るべきストリームのバッファとスワップすると共に、処理を行うスレッドをそのストリーム上に移動する(図7)。

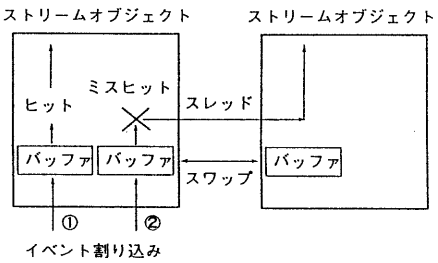

図7. パケット予測とスレッドの移行

予測には、以下の統計的性質を利用する。即ち、受信

したパケットの転送先ストリームと転送元ストリームは、その直前に受信したパケットのそれと同じである確率が高い。例えば、文献[7]ではメールサーバで89.8%、汎用ワークステーションで93.2%が、直前のパケットと同じストリームに属するものであったと報告されている。

5.2 バッファ長予測と割り当て

一般に高速な伝送媒体では、大きいMTU (Maximum Transmission Unit = IPパケットの長さと考えれば良い)が使用される。例えば、FDDIの場合、4352byteである。また、ATM (Asynchronous Transfer Mode)上のIPの場合、9180byteである。これは、ヘッダ処理のオーバヘッドを避けてスループットを向上させるためである。ところが、このために大きなバッファを用意するとメモリの使用効率が著しく低下する。それは、telnetなどのようにインタラクティブな利用の場合、1パケットに1キャラクタ程度しかデータが入っていないためである。逆に、短かいバッファをリンクさせて使用すれば、フルにMTUを使い切るNFS (Network File System)やFTPのような場合、そのリンクの処理が大きなオーバヘッドになる。即ち、高速から低速までをカバーするような適応性が要求される場合には、適切な単一のバッファ長を選択するのは困難である。

そこで、以下のような長さのバッファを用意し、オブジェクトの生成時に利用するものを決定する。

(1) FTPクライアントのGETやサーバのPUTのようにACKしかしない場合、あるいは、telnetのクライアントのように入るデータサイズがせいぜい数バイトの場合には、バッファ長は伝送媒体固有のヘッダとIPおよびTCPヘッダの長さ程度(例えば、64バイト)あれば十分である。

(2) FTPクライアントのPUTやサーバのGETの場合、フルサイズのデータを転送するため、媒体のMTUサイズを越えるバッファを使用する。

(3) telnetサーバからクライアントへのストリームやSMTPなどのインタラクティブな通信では、速度はあまり重要な要素ではない。従って、128から256バイト程度の長さがあれば十分であろう。

ストリームの使用するバッファサイズは、ストリームを生成する時にユーザからのオプション指定またはシステムの自動的な検出で決定される。実際、ユーザはストリームをオープン(生成)する時に、入力ストリームであるか、出力ストリームであるか、あるいは双方向のストリームであるかを指定する。また、利用するサービス名も指定するので、これらをアドバイスとして利用することができる。もし、実際に使用されるバッファ長が予測された長さよりも短かい場合には、上記のバッファをリンクで接続して一つのバッファとして取り扱う。今後、これらのバッファの長さについては、実環境での統計を更に調査する必要がある(図8は、当研究所のあるイーサネットで観測したもの)。

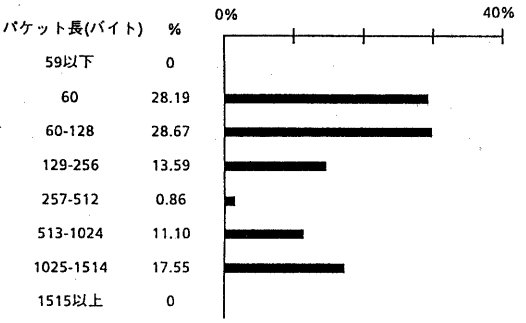

図8. パケットの長さの統計(Ethernetの場合)

5.3 ヘッダ予測とチェックサム

チェックサムの計算やバッファコピーなどのメモリアクセスは、大きいパケットを処理する際には、パフォーマンスを低下させる要員となる。TAO/SILENTでは、メモリのキャッシュにライトスルー方式を採用しているのでメモリ書きこみ時にはキャッシュが効かない。また、パケットの処理を行うためには、I/O空間にあるネットワークインタフェースのバッファとセル空間内のベクタとの間で、どうしてもメモリコピーを行わなければならない。さらに、チェックサム計算やデータ部分のコード変換のようなパケットのオクテットをすべてアクセスする必要のある処理を別に行うとメモリアクセスの回数は更に増大する。これに対処するため、I/O空間のバッファとセル空間のベクタとの間のコピー時にチェックサム計算やコード変換の前処理を行う(図9)。

ところが、チェックサムを計算するには、パケットのどこからデータ部が開始されているかを前もって知る必要がある。しかし、オプションや伝送媒体固有のヘッダ長が異なるため、場所はあらかじめわからない。そこで、標準的なヘッダを想定してチェックサム計算を行っておき、後に補正して正しいチェックサムを求める。これに

よってメモリアクセスの増加を最小限に押える。

一方、コード変換で最も頻繁に発生するのは、テキストの転送時におけるデリミタ (Carriage Return や Line Feed) の変換である。例えば、UNIX では、LF が行のデリミタである。また、Macintosh では、CR がデリミタである。ネットワーク上では、CRLF の 2 連続キャラクタを標準デリミタとすることになっている。これらの変換を行うには、バッファをすべてサーチする必要があり、かなりのメモリアクセスが発生する。これを避けるため、I/O バッファからベクタへの転送の際に変換してしまいたい。ところがデータの開始位置がわからない。このため、変換した位置を記憶しておき、ヘッダ中のデータを誤って変換している場合には、後に元へ戻す。この手法は、漢字コードの変換にも適用することができる。

もう一つ、バッファでサーチが発生するのは、telnet のネゴシエーションのためのエスケープシーケンスである。これは、telnet が一般のキャラクタに混ぜてネゴシエーションのためのエスケープシーケンスを送ってくることに起因する。これに対しては、どの位置にエスケープキャラクタがあったかをチェックしておく。

なお、以上の方式は、パケット受信には利用できるが、パケット送信の場合には、送信前に予め処理を終わっていなければならないので利用できない。

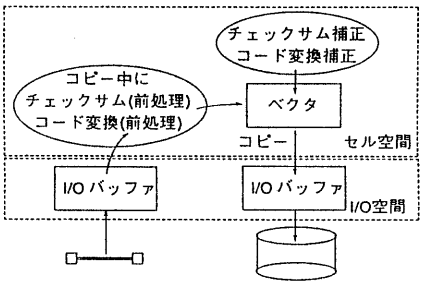

図 9. バッファとベクタ間のコピー

5.4 ヘッダの再利用

パケットの送信処理を行う場合には、パケットを作成する度にヘッダも作成するのが一般的である。しかし、ユーザが通信を開始すると、その通信相手はほとんど変わらない。(障害や、より良い経路の発見によって中継するルータを変更する場合を除く。) このことは、ヘッダ部分はほとんど変更されないことを意味している。そこで、一旦作成したヘッダ部分を再利用すれば、ヘッダ作成のオーバヘッドを減らせるということが期待できる。

再利用には 2 つの方法がある。まず、1 つ目の方法は、パケット全体を同一のバッファに入れておき、そのバッファのヘッダ部分を再利用する方法である。この場合、密結合オブジェクトでファイルを相手のホストへ転送する時には、バッファの先頭にあるヘッダ部分を避けてファイルからのデータを入れるようにする必要がある。もう一つの方法は、ヘッダだけを別バッファとする方法である。送信時には、まず、このヘッダを送信した後にデータ部を連続して送信する。どちらを採用するかは、今後の検討課題である。

6. プロセス制御

6.1 デーモンプロセス

プロトコル処理のデーモンプロセスを実装する場合、ひとつのデーモンプロセスがいくつものストリームの処理を行う方法がある。これは、プロセス数を押えることができるという利点がある。しかし、あるストリーム内でプロセスがサスペンドすると全体の処理が止まってしまうという問題もある。そこで、SILENT 上の実装では、TAO の軽いプロセスを生かし、各ストリームごとに独立したデーモンプロセスを走行させる。

なお、デーモンの中には、すでに 5.1 で説明したインタフェースのパケット受信を行うデーモンのように、実体を持たず、スレッドがストリームオブジェクト上を周回するものもある。

6.2 排他制御

TAO/SILENT のネットワーク実装においては、プロセススイッチによるオーバヘッドを減らすため、レイヤが異なってもメソッドを連続的に呼び出して同一プロセス上で処理を行う。この方法では、いくつものプロセスが同一オブジェクト上を走行するために排他制御が必要となる。また、TCP プロトコル自体が送信データと ACK(受信データに対する確認) を同一パケットにピギーバックしていることも排他制御を複雑にしている。例えば、telnet のサーバの場合、出力ストリームでは以下の 4 つものプロセスが走行する。

(1) ACK を返すためのプロセス
(2) タイムアウト発生による再転送のためのプロセス
(3) フラッシュ要求を発生したユーザからのプロセス

(4) 一定時間内に次のデータが発生しなかったためにオートフラッシュを行うプロセス

このような競合する資源を操作する場合には、対応するプロセスを別に作成してシリアライズする方法もあるが、ここでは、プロセススイッチを極力少なくするために必要最小限の排他制御を行う。例えば、TCP ストリームでは、その状態を保持するスロットの排他制御が必要となる。それは、ユーザからのプロセスによるアクセス(例えば、リセットやアボート処理)とインタフェースの受信割り込みのプロセスによるアクセス(例えば、コネクションの設定やリセットなど)が競合するからである。排他制御の時間は、このスロットの極めて短かいアクセス時間で済むので、ここではTAO のビジーウェイト型排他制御のロッカを使用する。

上記のような場合と異なり、複数の資源の操作が競合する場合にはセマフォを利用する。また、バッファメモリへのアクセスなどのように極めて短かい時間で操作が完了する場合には、割り込み禁止で走行することによって排他制御を行う場合もある。

7. おわりに

本論文では、記号処理計算機 SILENT とその記号処理言語 TAO をベースとしたネットワークシステムについて説明し、基本的な設計思想をソフトハード両面について明確にした。TAO/SILENT のネットワークは、オブジェクト指向によるストリームの実装を基本としている。処理にあたっては、パケット予測、メモリアクセスの抑制と効率化、プロセススイッチの抑制などによってオーバヘッドを減らす。また、ストリームオブジェクトを結合してバッファの共有化を行えば、効率のよいパイプ機構を提供することができる。一方、ハードウェアでは、SILENT プロセッサが直接ネットワークハードウェアを制御しており、フロントエンドプロセッサによるオーバヘッドを避けている。

TAO/SILENT のネットワークソフトウェアは、現在基本設計の段階にある。今後、詳細設計にあたっては、クリチカルな関数のマイクロコード化やオブジェクトの標準インタフェースの規定などを行う必要がある。また、周期的(isochronous)にトラフィックを処理するようなリアルタイムプロトコルのサポート、パイプライン処理に適したバッファの構造、いくつものタイマの操作や管理方法、応答時間 RTT (Round Trip Time) の計算方法なども検討してゆく予定である。

[文献]

[1] 村上: オブジェクト指向による TCP/IP プロトコルの実現、コンピュータソフトウェア、Vol.6、No.1、pp.30-40、1989.

[2] N.C. Hutchuinson and L. L. Peterson: The χ-kernel: An Architecture for Implementing Network Protocols, IEEE Trans. on Software Engineering, Vol.17, No.1, 1991.

[3] 村田、横手、所: 平行オブジェクトによるネットワークシステムの構築、日本ソフトウェア科学会第 10 会大会論文集、pp.157-160、1993.

[4] 天海、山崎、中村、吉田、竹内: TAO のコンカレンシーコントロール、記号処理研究会、69-3、1993.

[5] 竹内、吉田、天海、山崎: TAO/SILENT の実時間性について、記号処理研究会、61-1、1991.

[6] I. Takeuchi, H. G. Okuno, and N. Ohsato: A List Processing Language TAO with Multiple Programming Paradigm, New Generation Computing, 4-4, 1986.

[7] D. D. Clark, V. Jacobson, J. Romkey, H. Salwen : An Analysis of TCP Processing Overhead, IEEE Comm. Magazine, June 1989.