RT-Machにおける分散リアルタイム通信の実現†

1Z－1　喜多山卓郎　斉藤鉄也　徳田英幸
慶應義塾大学 SFC 研究所　慶應義塾大学大学院 政策メディア研究科　慶應義塾大学 環境情報学部

1　はじめに

近年のマイクロカーネルの技術では，従来カーネル内で行なわれていたプロトコル処理を，カーネル内のパケットフィルタを用いることによりユーザレベルで行なうことを可能にした．これにより，マイクロカーネル方式を用いた場合に問題となっていた性能を犠牲にすること無しに，プロトコル処理の高い柔軟性を実現することが可能となった．同時に，この方式は従来オペレーティングシステムのようにカーネル内でプロトコル処理をする場合や，マイクロカーネル上のサーバにより処理をする場合と比べて，高いプリエンプタビリティを提供することが可能となった．我々は，分散リアルタイム環境での通信メカニズムとして，この方式に拡張を加えクライアントの要求の優先度を使ってサーバ側のプロトコル処理を行なう方式を提案し，この方式を RT-Mach[8] 上への実装を進めている．

2　通信モデル

我々のリアルタイム通信のモデルは，概念的には処理に対して優先度を付ける．したがって，マシン間での優先度の伝搬が必要となる．モデルの特徴を以下に挙げる．

- サーバの優先度は，サービス実行時はクライアントの優先度と同じものを使用する．
- 全てのサーバとクライアントのホストにおける優先度は一貫性があるものとする．
- クライアント-サーバ間の point-to-point 通信．
- サーバはシングルまたは，マルチスレッドで実装されている．
- 通信は同期，非同期をサポートする．

クライアントの優先度がサーバの優先度として使用されている期間は，同期通信の場合，サーバがサービスを行なっている期間はサービスの要求を受けとってからリプライを返すまでであり，非同期通信の場合は，サービスの要求を受けとってから次の要求を待つまでの期間とする．

3　従来のプロトコル処理方式

本章では，従来行なわれてきたプロトコル処理について簡単に述べる

3.1　カーネル内で行なう方式

4.3BSD などのモノリシックカーネルで使用されてきた方式で，全てのプロトコル処理をカーネル内で行なう．

この方式では，プロトコルの処理にシステム内で一番高い優先度が割り当てられる．性能面では高いスループットと優れたレスポンスを実現することが可能である．しかし，高い優先度のスレッドが，低い優先度のスレッドやネットワークから入ってきたパケットのプロトコル処理によって待たされる状況が発生する．また，次に説明するマイクロカーネルを用いた方式に比べ柔軟性が低い．

3.2　プロトコルサーバを用いた方式

従来のマイクロカーネル方式のオペレーティングシステムで用いられた方式で，Mach 上の UNIX サーバや，x-Kernel[1] がこの方式を用いている．

この方式は，プロトコル処理をユーザ空間で実現しているため高い柔軟性を提供できるが，プロトコルサーバとアプリケーション間のローカルな通信のオーバヘッドが生じる．

パケットを送る時のプロトコル処理に使用される優先度はアプリケーションにより指定することができ，この優先度よりも高い処理はプロトコル処理により待たされることはない．しかし，ネットワークから入ってくるパケットに対しては，要求の優先度がわからないため一番高い優先度で実行する必要が生じる．

この問題に対して，IP のオプションを使用し優先度を伝搬させる Prioritized-IP(PIP) が提案され [7]，RT-Mach 上の NPS (Network Protocol Server) に実装された．これにより NPS では UDP の処理は要求の優先度で実行することが可能となった．

3.3　プロトコルライブラリを用いた方式

この方式では，カーネル内のパケットフィルタ [4, 9] によりパケットの受けとるユーザプログラムを探し，プロトコルの処理はユーザレベルのライブラリとして提供され，実行時はユーザプログラムの一部として処理される．このため，高い性能と柔軟性の両方を実現している．この方式を用いると，プロトコル処理はサーバの優先度を使用して行なうことができ，また，高いプリエンプタビリティを提供することが可能である．この方式を，プロセッサリザベーション [5] に適応したモデル [3, 6] も提案されている．

しかしながら，サーバの優先度でプロトコル処理が行なわれるため，低い優先度のサーバに高い優先度の要求が来た場合に処理が待たされたり，高い優先度のサーバに低い優先度の要求が来た場合に他の高い処理が待たされてしまうなどの問題が生じる．

4　リアルタイム通信メカニズム

本章では，我々が実装中のリアルタイム通信メカニズムについて述べる．

Distributed real-time communication in RT-Mach
Takuro KITAYAMA[1], Tetsuya SAITO[2] and Hideyuki TOKUDA[3]
[1]Keio Research Institute at SFC,Keio University
5322 Endo Fujisawa 252 Japan
E-Mail: <takuro@sfc.keio.ac.jp>
[2]Graduate School of Media and Governance, Keio University
[3]Faculty of Environmental Information, Keio University
†この研究は，情報処理振興事業協会（IPA）が実施している創造的ソフトウェア育成事業「次世代マイクロカーネル研究プロジェクト」のもとに行われた．

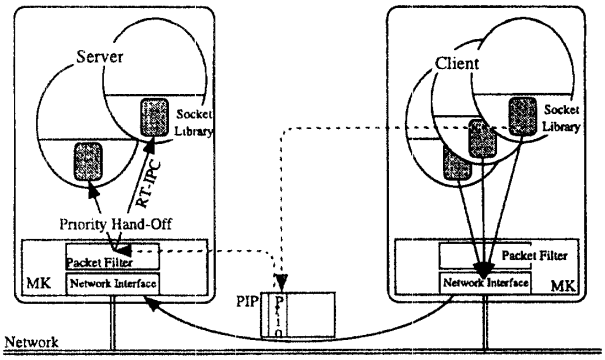

図 1: リアルタイム通信メカニズム

4.1 設計

図 1にリアルタイム通信メカニズムの様子を示す. 3.3章で述べたプロトコルライブラリを用いた方式をベースとして拡張を加えている. このため, プロトコルライブラリを用いた場合の利点である, 柔軟性, 性能, プリエンプタビリティを損なうこと無しに, クライアントからサーバへの優先度の伝搬を効率良く行なうことが可能になっている.

優先度の伝搬は次のように行なわれる. クライアントの優先度をサーバに伝搬させるために, クライアントからサーバへの要求はPIP を用いて行なわれる. クライアントがサーバへの要求を出す時, PIP ヘッダ内の優先度にクライアントの優先度を設定する. 一方サーバ側では, 受けとったパケットがPIP かどうかを確かめる. これはカーネル内のパケットフィルタが行なう. 受けとったパケットがPIP だった場合は, PIP パケット内の優先度をサーバの優先度としてセットし, パケットがサーバのプロトコルライブラリ内のプロトコル処理ルーチンに渡される. これにより, クライアントの優先度をサーバへ引き継ぐことが可能となる.

4.2 実装

我々は, リアルタイム通信メカニズムを RT-Mach 上に実装中である. ソケットライブラリはソケットの管理等のために 4.4BSD Lite Server(Lites) を使用しているため, Lites サーバにも一部変更を加えている. この部分は別のサーバとして実装し Lites サーバが無い場合でも使用できるような変更を予定している. 主な変更点を以下に挙げる.

socket library クライアントがサーバに要求を出す際に, PIP のパケットを投げるように変更を行なう.
サーバ側の対応は, オリジナルのソケットライブラリがIPC のポートを使ってカーネルからパケットを受けとっているのに対し, RT-IPC[2] のポートを使用して受けとるように変更する. RT-IPC は RT-Mach のリアルタイム拡張機能の一つであり, ローカルなスレッド間のリアルタイム通信手段である. これを利用することによりメッセージの送り側から受け側へ優先度の伝搬を行なうことが可能となる.

packet filter 現在の RT-Mach に実装されているパケットフィルタは, パケットの受け手側のポートを探すために使用されている. これを拡張し, PIP 内の優先度のフィールドを指定できるような変更を行なう. 具体的には, パケットフィルタがパケット内の指定されたアドレスを呼び側へ返す事が可能なよう変更を行なう.

microkernel カーネル内からユーザプログラムへパケットを渡す際に IPC を使用している. この部分の変更を行ない, パケットフィルタにより取り出された優先度をメッセージ内に設定し, RT-IPC のメッセージを送るようにする.

4.4 BSD Lite Server(Lites) ユーザのプログラムがパケットを受けとる IPC のポートは Lites サーバ内でアロケートされている. これを RT-IPC のポートをアロケートするように変更する.
また, パケットフィルタの設定をする部分で, パケットにPIP ヘッダがあるかどうかを確かめ, PIP ヘッダがあった場合はその中の優先度フィールドを返すように設定する.

5 おわりに

本稿では, いくつかのプロトコル処理の方式について比較検討を行ない, リアルタイム環境でのプロトコル処理方式の提案を行なった. この方式は現在 RT-Mach 上で実装が進められており, 今後評価を進めていく予定である.

参考文献

[1] N. C. Hutchinson and L. L. Peterson. The *x*-Kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, January 1991.

[2] T. Kitayama, T. Nakajima, and H. Tokuda. RT-IPC: An IPC extension for Real-Time Mach. In *Proceedings of the USENIX Symposium on Microkernel and Other Kernel Architectures*, 1993.

[3] C. Lee, K. Yoshida, C. Mercer, and R. Rajkumar. Predictable communication protocol processing in real-time mach. *In the proceedings of IEEE Real-time Technology and Applications Symposium*, June 1996.

[4] S. McCanne and Van Jacobson. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In *Proceedings of the 1993 Winter USENIX Conference*, January 1993.

[5] C. W. Mercer, S. Savage, and H. Tokuda. Processor Capacity Reserves for Multimedia Operating Systems. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, May 1994.

[6] C. W. Mercer, J. Zelenka, and R. Rajkumar. On Predictable Operating System Protocol Processing. Technical Report CMU-CS-94-165, Carnegie Mellon University, 1994.

[7] T. Nakajima and H. Tokuda. User-level Real-Time Network System on Real-Time Mach. In *Proceedings of 4th International Workshop on Parallel and Distributed Real-Time System*, 1996.

[8] H. Tokuda, T. Nakajima, and P. Rao. Real-Time Mach: Towards a Predictable Real-Time System. In *Proceedings of USENIX 1st Mach Workshop*, October 1990.

[9] M. Yuhara, B. N. Bershad, C. Maeda, and J. E. B. Moss. Efficient Packet Demultiplexing for Multiple Endpoints and Large Messages. In *Proceedings of the 1994 Winter USENIX Technical Conference*, January 1994.