3K−1 バックプレイン・ネットワークによるプロセッサ間通信とリモートシステムコール

下島 健彦*, 関 啓一*, 有野 康仁**, 山名 千秋**, 古城 隆*
* 日本電気(株) C&C共通ソフトウェア開(本) 基本システム開発部
** 日本電気マイコンテクノロジ(株) 共通ソフトウェア部

1 はじめに

組み込み/制御システムでは, バックプレイン(バス)で接続されたマルチプロセッサ構成が多く使われている. このようなシステムは各プロセッサのローカルメモリ上でOSやアプリケーション・プログラムが動作し, プロセッサ間共有メモリを通信媒体として使用する疎結合マルチプロセッサが多い. 共有メモリによる通信は広域ネットワークやLANなどとは速度や信頼性が大きく異なっており, より単純で高速なプロトコルが適している.

我々はリアルタイムOS RX616[3]をベースにプロセッサ間共有メモリによる小型で高速な通信プロトコルを実現し, システムコールのリモートプロセッサ上での実行という拡張を行なった.

2 通信プロトコル

バックプレイン・ネットワークでは, 広域ネットワークやLANなどのネットワークとは転送レート, 遅延時間, 通信路上のエラー率, セキュリティの考え方などが異なるため, LANなどを前提に設計されたものよりも単純で高速な通信プロトコルが適している. 一方, アプリケーション・インタフェースは, アプリケーションの共有などを考えると, 標準インタフェースが望ましい. 我々は, アプリケーション・インタフェースはUNIXのソケットをベースにそのサブセット化を行ない, 通信プロトコルはバックプレイン専用の軽いものを実現した.

2.1 アプリケーション・インタフェース

バックプレイン・ネットワークでは, 通信路はメモリであり, データの誤り率は十分に低いため, 確認応答はなくてもよい. また共有メモリ上のバッファ(パケット)は送信側と受信側で共有しているため, 受信側の処理が遅れてくると, パケットがアロケートできなくなり, 送信側はブロックされるので, フロー制御も不要である. そこで, バックプレイン・ネットワークではデータグラムのみをサポートした.

2.2 バックプレイン通信

基本的に, 固定長パケットの送受信のみをサポートした. パケットサイズはユーザが指定可能にすることとし, メッセージの複数パケットへの分割はサポートしていない. データ構造としては, プロセッサ間共有メモリ上に固定長のメモリブロック(パケット), そのフリーリストとプロセッサ対応の着信パケットリストを用意した. それぞれのリストに対

Inter-processor Communication and Remote System Call on a Backplane Network.
Takehiko SHIMOJIMA*, Keiichi SEKI*, Yuuji ARINO**, Chiaki YAMANA**, Takashi KOJO*
*NEC Corporation.
**NEC Microcomputer Technology, Ltd.

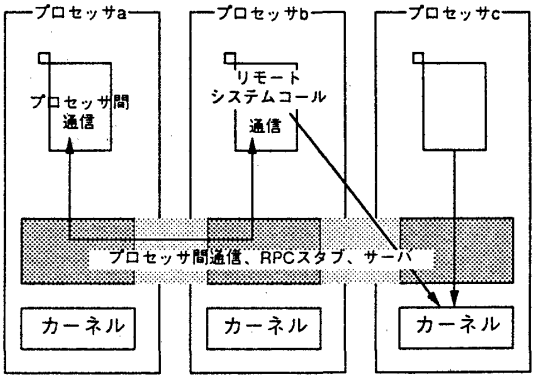

図1: プロセッサ間通信とリモートシステムコール

する操作は, 不可分に比較と交換を行なうcompare-and-exchange命令(V60/V70のCAXI命令や68020/30のcas命令)で行なうことにより, リストのアトミックな操作を1命令で実現し[2], 共有メモリに対するロック時間を小さく押えている.

3 システムコールの拡張

プロセッサ間通信を利用してシステムコールをリモートプロセッサのカーネルに対して発行するような拡張が可能になる. これによりリモートプロセッサ上のタスクの生成, 起動を行なったり, プロセッサ間で同期, 通信を行なったり, クロックを同期させるなどが可能となる.

システムコールをリモートに発行するように拡張する場合, アプリケーションがプロセッサ番号を意識するかどうかというロケーションの透過性が問題となる. 組み込み/制御システムでは, 特定の制御対象物は特定のプロセッサに接続されていたり, タスクとプロセッサの対応はシステム構成時に固定的であることが多く, プロセッサを意識せざるを得ないことが多い. そこで, リモートシステムコールでは常にプロセッサ番号を指定することにした.

実現はBirrellのRPCの実現[1]と同様の方法で, 図2に示したようにクライアント・スタブ, サーバ, 2で述べた通信モジュールから構成される.

RX616のシステムコールでは, 中断を含まず, 割り込みレベルから発行できるものがある. これらをリモートに実行する場合は, リクエストをサーバまで渡さず, パケット到着の割り込み処理からシステムコールを発行し, 結果をクライアントに返すというショートカットを行なっている.

4 性能評価と改善

実現したプロセッサ間通信とリモートシステムコールのオブジェクトサイズを表1に示す.

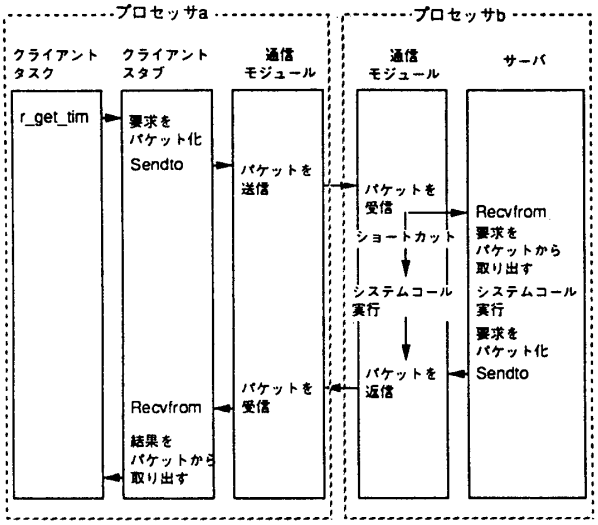

図 2: リモートシステムコールの実現

モジュール	サイズ (バイト)
プロセッサ間通信	2.7k
クライアントスタブ (1 システムコール当たり)	115
サーバ	2.3k

表 1: オブジェクトサイズ

オブジェクトサイズは V60/V70 でのサイズを示した. 特に通信モジュールは機能を単純化したことにより, 小型化されている.

End-to-end の通信遅延時間は 785μsec であった. これは, 受信側タスクが recvfrom でメッセージを待っている状態で, 送信側タスクが sendto をコールしてから受信側タスクが recvfrom からリターンするまでの時間で, 16MHz で動作する V60 2 台をマルチバスで接続した環境で, ICE を用いて測定した. 送信したデータは, リモートシステムコールでのリクエストのデータサイズである 11 ワードである.

さらに, 通信モジュール内の処理時間の詳細を同じ環境で測定した. 結果は表 2に示す通りである.

処理	時間 (μsec)
パケット獲得, メッセージコピー	50
パケットのリンク, プロセッサ間割り込み	79
割り込み処理, パケットの取りだし	44
タスクの待ち解除, タスク切り替え	430
待ちの後処理	145
パケット解放	37

表 2: 通信時間の詳細

バックプレインのような単純なプロトコルのネットワークでは, 通信時間の中で, 待ち状態のタスクを起こす処理の占める割合いが大きい. つまり高速な RPC を実現するためにはプロトコルの高速化だけでなく, カーネルの基本性能の改善が必要なことが分かる.

リモートシステムコールのオーバーヘッドは, サーバとのメッセージの往復時間になるため 1570μsec 程度になる. 中断を含まないシステムコールで, ショートカットを行なった場合のオーバーヘッドは, 908μsec であった. このケースでは, リモートシステムコールの要求を待っているサーバへのコンテキスト・スイッチ時間程度高速化される. さらに, クライアントタスクがリプライをビジーウェイトで待つようにしてクライアント側のコンテキスト・スイッチもやめるとオーバーヘッドは 352μsec になる.

カーネルがプリエンプティブ・スケジュリングを行なっている場合, クライアントタスクはリモートシステムコール発行時点でそのプロセッサ中で最高プライオリティのタスクである. 数百 μsec 程度の時間経過であれば, リプライ受信後もそのタスクは最高プライオリティである可能性が高い. この場合, 一旦他のタスクにプロセッサを渡しても, リモートでの処理が終了するとすぐに元のタスクにスイッチすることになり, タスクレベルのプロセッサ利用率はあまり向上しない. ビジーウェイトでリプライを待つ方法は, 高プライオリティのタスクのブロック時間が短くなる. システムコールのリモート実行のように実行時間の上限がおさえられる RPC では, ビジーウェイトは有効である.

5 まとめ

プロセッサ間共有メモリによる通信プロトコルとリモートシステムコールを実現し, 性能評価を行なった. その結果,

- 高速な通信を実現するためには, プロトコルの高速化だけでなくイベント待ちタスクの待ち解除というカーネルの基本性能の高速化が重要なこと,
- ビジーウェイトが有効であること

を示した.

これは, プロセッサ間共有メモリによる通信だけでなく, イーサネットや FDDI のようなネットワーク上に専用の高速なプロトコルを実現し, 高速な分散システムを設計する場合にも共通する.

参考文献

[1] Andrew D. Birrell and Bruce J. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, February 1984.

[2] Henry Massalin and Calton Pu. Threads and input/output in the synthesis kernel. *the 12th ACM Symposium on Operating System Principles*, 23(5):191–201, December 1989.

[3] 市瀬 他. リアルタイム OS における高速応答性能の実現と 透過的な OS モデルの実現. 第 *42* 回情処全国大会, March 1990.