共有メモリを介したストリーム通信ライブラリの構築と性能評価

3F－5

古賀靖人　上原敬太郎　益田隆司
東京大学大学院理学系研究科情報科学専攻

1　導入

共有メモリをプロセス間通信に用いることで効率の良い通信を行えることが知られている[1]。しかし単に共有メモリがあるだけではプログラマが明示的に共有メモリの管理や同期を行う必要があり、UNIXのソケットを用いる場合に比べて手間がかかる。

そこで、共有メモリの管理や同期を内部で行う通信ライブラリを構築してソケットと同じストリーム形式のインターフェースを提供することが考えられる。これにより、ソケットと同じ感覚でより効率のよい通信を行うことができ、現在ソケットを用いているプログラムを共有メモリを利用するように書き換えることも容易になる。

本研究では、実際にSolaris上でmmapと同期ライブラリを組み合わせてストリーム形式の共有メモリ通信ライブラリを実装し、ソケットを用いた場合と性能を比較する実験を行ってその有効性を検証した。

本論文の残りの部分は次のように構成される。2節で本ライブラリが提供するインターフェースについて述べ、3節でその実装について説明する。4節で性能評価のための実験とその結果を示し、5節で結論を述べる。

2　インターフェース

共有メモリで通信の効率を上げるという考えは古くからあり、mmapの機能を利用した研究もなされている。例えばKriegerらの研究[2]では、mmapの機能を利用して主にファイル入出力の効率を上げている。プロセス間通信の機能も提供しているが、従来のソケットの機能を保つことに主眼を置いているため扱うデータはバイト列で基本的に1対1の通信である。

本ライブラリはバイト列ではなく任意サイズのメモリオブジェクトを単位としてsend、recvを行うストリーム形式のプロセス間通信を提供する。メモリオブジェクトのデータ領域と通信チャンネルを分離することにより、データをコピーせずに任意の相手にマルチキャストすることができる。本ライブラリの使用例を図1に示す。以下ではインターフェースの詳細について述べる。

```
// memory object
    memory_object_t m;
// send
    send_port sp(2); /* port id == 2 */
    m = new memory_object(16);
    // write m ...    /* size == 16 */
    sp.send(m);
    delete m;
// receive
    recv_port rp(2); /* port id == 2 */
    m = rp.recv();
    // read m ...
    delete m;
```

図1: 本ライブラリの使用例

2.1　メモリオブジェクト

バイト列を扱うソケットとは異なり、本ライブラリではサイズを持ったメモリオブジェクトを単位として送受信を行う。メモリオブジェクトのサイズは任意である。プログラマがサイズを指定してメモリオブジェクトを構築すると、そのメモリオブジェクトのデータ領域は共有メモリに確保される。get_ptr()はそのデータ領域へのポインタを返し、これを用いてデータの読み書きを行う。使用後、不要になったメモリオブジェクトはdeleteする。

2.2　ポートと送受信

本ライブラリはポートと呼ぶ1対1単方向の通信チャンネルを提供する。送信側と受信側はそれぞれポートidを指定してsend_port、recv_portを構築する。送受信は共有メモリ上のデータへのポインタの受け渡しを行い、データのコピーは不要である。メモリオブジェクトをdeleteする前に複数のポートにsendすることにより、データをコピーせずにマルチキャストすることができる。

A Library for Stream IPC via Shared Memory
Nobuhito Koga, Keitaro Uehara, and Takashi Masuda
The Department of Information Science, Graduate School of Science, the University of Tokyo

3 実装

上述のような通信ライブラリをSolaris上に実装した。Solarisはプロセス間でも同期を行えるライブラリを提供しており、その内のmutexと条件変数を使用している。共有メモリはmmapによって実現した。

3.1 メモリオブジェクトの実装

メモリオブジェクトを生成すると共有メモリに領域が確保される。各メモリオブジェクトにはリファレンスカウントが付けられており、生成時に1にセットされる。sendするとインクリメントされ、deleteするとデクリメントされる。0になるとメモリ領域が解放される。

3.2 ポートと送受信の実装

ポートは共有メモリ上のメモリオブジェクトへのポインタを保持するキューを持つ。キューは共有メモリ上の連結リストとして実装されている。キューの操作のためのmutexや、送受信の同期を行う条件変数を持つ。

3.3 selectの実装

各ポートに1つの条件変数だけでは、複数のポートからの受信を待つ場合にポートの数だけスレッドが必要になる。そこで本ライブラリはselectの機能を提供し、select用の条件変数を用意して受信を待つポートからポインタを張っておく。ポートにメモリオブジェクトを送信した時にポートの条件変数とselect用の条件変数の両方をsignalする。これにより、1つのスレッドで複数のポートからの受信を待つことができる。

4 性能評価

本ライブラリとソケット(TCP、UDP)を用いた場合の性能を比較する実験を行った。実験は、SPARC Station 20 (4CPU 50MHz、Memory 128MB) 上で1ホスト内のプロセス間におけるround-trip timeを測定した。以下では本ライブラリをMSTRMと表現する。MSTRMでは確保したメモリ領域にダミーのデータを書き込む時間も含めているのに対し、ソケットではユーザーバッファに書き込む時間は除いている。これは、ソケットではすでにデータがあればそのままwriteすることができるからである。

1対1でメモリオブジェクトのサイズを変化させた場合の結果を図2に示す。MSTRMはTCPやUDPよりも約3倍以上速くなっている。メモリオブジェクトのサイズを64KBに固定してマルチキャストのround-trip timeを測定したものが図3である。データのコピーが不要なMSTRMがほぼ一定であるのに対して、ソケットでは送信先の数に応じて増加していることがわかる。

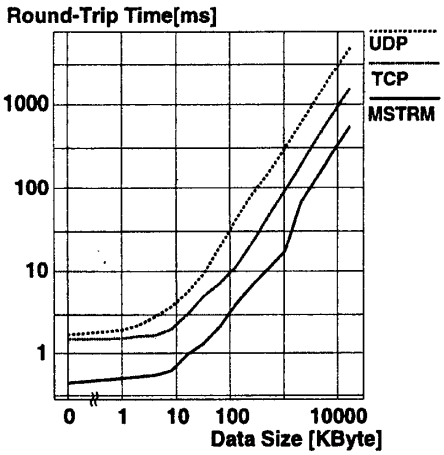

図 2: 1対1 round-trip time

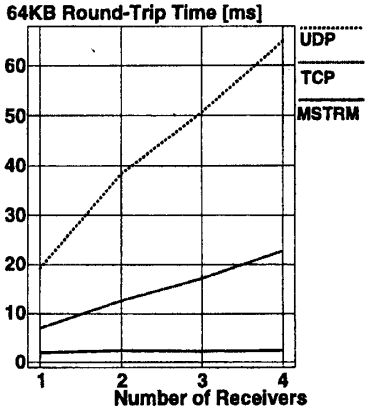

図 3: マルチキャスト時のround-trip time

5 結論

mmapによる共有メモリと同期ライブラリを組み合わせて効率のよい通信を実現することができた。ソケットに似たストリーム形式のインターフェースにより、プログラマは共有メモリの管理や同期を意識することなく、容易に効率のよい通信を利用することができる。

本研究ではポートを単方向としたが、これを双方向に拡張することは容易である。また、現段階ではホスト間の通信にはソケットを用いるほかないため、同じインターフェースを分散に拡張することを検討中である。

参考文献

[1] R. Fitzgerald and R. Rashid. The integration of virtual memory management and interprocess communication in accent. *ACM Transactions on Computer Systems*, 4(2), May 1986.

[2] Orran Krieger, Michael Stumm, and Ron Unrau. Exploiting the Advantage of Mapped Files for Stream I/O. In *Proceedings of the USENIX 1992 Winter Conference*, pages 27–42. USENIX Association, January 1992.