2J-05

非対称マルチプロセッシングシステムにおける効率的なリアルタイムOS間同期RPC

新谷 正太郎 †　　矢代 武嗣 ‡　　坂村 健 †‡
† 東京大学大学院学際情報学府　　‡YRP ユビキタス・ネットワーキング研究所

1 はじめに

近年の組込みシステムの高度化に伴い，消費電力を抑えつつ性能を向上させる手段として，組込みリアルタイムシステムにおいてもマルチコアプロセッサが広く用いられている．また，個々のプロセッサコア上で動作し，他のOSと通信する機能を有するリアルタイムOS（RTOS）は「非対称マルチプロセッシング（Asymmetric Multiprocessing; AMP）型RTOS」と呼ばれる．

多くのAMP型RTOSは，他OSのシステムコールを呼び出すために，RPC (Remote Procedure Call) に基づく通信機構を備えている．一般に，RPCの応答の通知方式は，要求を出したタスクが通知を待ち続ける「busy wait」方式と，別のタスクに実行が切り替わり通知は割り込みにより受ける「block wait」方式に分かれる．

OS間RPCの実行方式は，処理に応じて最適な方式が異なる．block waitはコンテキストスイッチが発生するため，常に一定のオーバヘッドが生じる．要求する処理が非常に短い場合は，このオーバヘッドはボトルネックになりうる．busy waitはこの問題は生じないが，要求する処理が長いときに，クライアントOSにおけるCPUの利用効率が低下し，スループット悪化に繋がる．従って，システムの平均的なリアルタイム性を向上させるためには，両者を適切に使い分ける必要がある．

本研究では，最適なOS間RPC実行方式を自動選択することで，追加の開発コストを強いることなく，平均的なリアルタイム性を向上させることを目的とする．

2 関連研究

両方式を組み合わせる研究として，OS間通信の開始時はbusy waitで待ち，ある期間が経過するとblock waitに移行する手法が提案・利用されている [1][2]．この手法を組込みRTOSに適用する場合，RPC時に一定期間busy waitを行うことから，動作環境や処理によってはリアルタイム性を満たせない可能性が懸念される．

3 異なるOS上のタスク同士のデータ送受信

OS間RPCの代表的な処理として，異なるOS上で動作するタスク同士のデータ送受信がある．これには，OS間共有領域とOS内共有領域が用いられる．OS間のデータ送信関数 (send_buf() とする) は以下の通りである．まず，クライアントOSのsend_buf()がOS間共有領域にデータコピーを行い，サーバOSに対して通知を行う．次に，サーバOSのsend_buf()がOS間共有領域からOS内共有領域にデータコピーを行う．最後に，クライアントOSに処理完了通知が返される．send_buf()の処理の流れを図1と図2に示す．OS間のデータ送信後に，サーバOS内で動作するタスクから受信関数が呼び出される．これにより，OS内共有領域から受信タスクの領域にデータがコピーされ受信が完了する．

4 提案

本研究では，OS間RPCによるsend_buf()について，データ転送量に応じて最適な実行方式を選択する手法を提案する．具体的には，両方式の性能が逆転するデータ転送量の条件を求め，それを満たす値をしきい値として，RPC実行方式をOSで切り替える機能を実装する．

条件を求めるために，send_buf()が用いられるアプリケーションの処理時間をモデル化する．クライアントOSに「タスク1」と「タスク2」が存在するアプリケーションを想定する．タスク1はOS間のsend_buf()のみを行う通信専用の処理とし，実行時間を $T_1(x)$ とする．ここで，x はデータ転送量 [byte] を表す．タスク2は他のOSと通信をしない処理とし，実行時間を T_2 とする．また，タスク2にCPUを割当て可能な時間を T_{cpu} とする（常に $T_2 > T_{\mathrm{cpu}}$ とする）．最後に，busy waitとblock waitの両方式について，両タスクの完了にかかる時間をそれぞれ，T_{bsy}，T_{blk} とする．上記以外の記号については，図中で示すものに対応する．

busy waitでの処理時間は図1より，

$$T_{\mathrm{bsy}} = T_1(x) + T_2 = (T_{\mathrm{c\_cpy}}(x) + T_{\mathrm{s\_cpy}}(x) + T_{\mathrm{c}} + T_{\mathrm{s1}} + T_{\mathrm{s2}} + T_{\mathrm{s\_hdr}}) + T_2$$

block waitでの処理時間は図2より場合分けが生じる．

1. $T_{\mathrm{s\_cpy}}(x) + T_{\mathrm{s2}} < T_{\mathrm{ctx}}$ のとき，

An Efficient Synchronous RPC Between Real-Time Operating Systems on Asymmetric Multiprocessing Systems
†Shotaro SHINTANI　‡Takeshi YASHIRO　†Ken SAKAMURA
†Graduate School of Interdisciplinary Information Studies, The University of Tokyo
‡YRP Ubiquitous Networking Laboratory

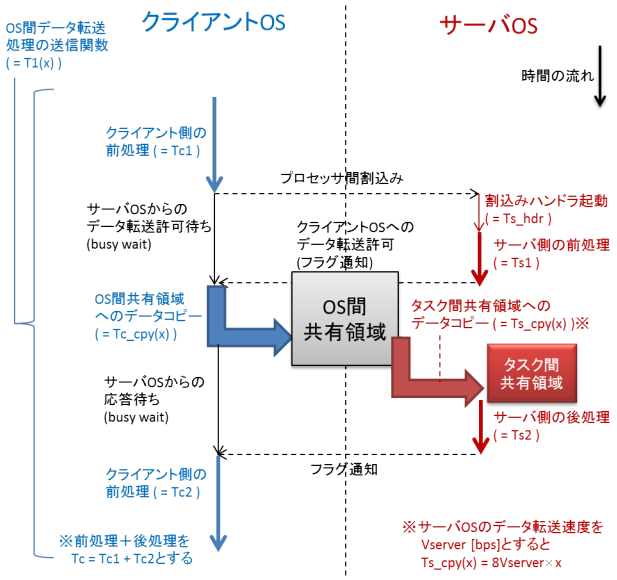

図 1: busy wait 方式の OS 間 send_buf() の処理の流れ

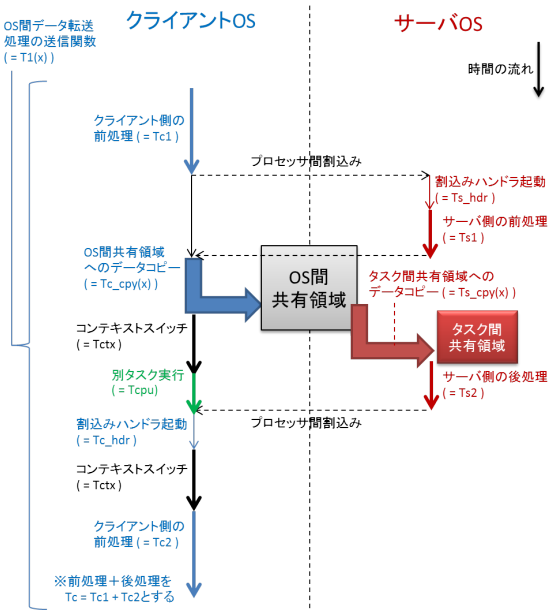

図 2: block wait 方式の OS 間 send_buf() の処理の流れ

$$T_{\mathrm{blk}} = T_1(x) + T_2$$
$$= (T_{\mathrm{c\_cpy}}(x) + T_{\mathrm{c}} + T_{\mathrm{s1}} + T_{\mathrm{s\_hdr}} + T_{\mathrm{c\_hdr}} + 2T_{\mathrm{ctx}}) + T_2$$

2. $T_{\mathrm{s\_cpy}}(x) + T_{\mathrm{s2}} \geq T_{\mathrm{ctx}}$ のとき，

$$T_{\mathrm{blk}} = T_1(x) + T_2 - T_{\mathrm{cpu}}$$
$$= (T_{\mathrm{c\_cpy}}(x) + T_{\mathrm{s\_cpy}}(x) + T_{\mathrm{c}} + T_{\mathrm{s1}} + T_{\mathrm{s2}} + T_{\mathrm{s\_hdr}}$$
$$+ T_{\mathrm{c\_hdr}} + T_{\mathrm{ctx}}) + T_2 - (T_{\mathrm{s\_cpy(x)}} - T_{\mathrm{s2}} - T_{\mathrm{ctx}})$$

1. では，常に $T_{\mathrm{blk}} - T_{\mathrm{bsy}} < 0$ となるので，この範囲では busy wait を用いるのが適切である．

2. では，$T_{\mathrm{blk}} - T_{\mathrm{bsy}} = T_{\mathrm{c\_hdr}} + 2T_{\mathrm{ctx}} - T_{\mathrm{s\_cpy}}(x) - T_{\mathrm{s\_stub\_2}}$ となるので，この式の符号が逆転する $x = x_{\mathrm{opt}}$ が両方式の性能が逆転するデータ転送量である．以上より，しきい値として，$x_{\mathrm{opt}} = 8V_{\mathrm{server}}(2T_{\mathrm{ctx}} + T_{\mathrm{c\_hdr}} - T_{\mathrm{s2}})$ を定めると，最適な OS 間 RPC の実行方式を自動選択できる．

導出したしきい値により実行方式を選択する機能を，AMP 型 RTOS である「AMP T-Kernel」に実装した．

5 実験と評価

OS 間のイベント通知を想定して，タスク 2 が生成したデータを，タスク 1 を通して他 OS に送信するベンチマークプログラムを作成し，実験を行った．生成データのサイズを 1 byte から 50 kbyte の範囲で変更させ，T_{bsy} と T_{blk} の値を計測し，両者の比を取った．計測は，「MP T-Kernel/NE1 評価ボード」上で行った．

本環境においては，$(2T_{\mathrm{ctx}} + T_{\mathrm{c_hdr}}) = 336\ \mu$s，$T_{\mathrm{s2}} = 81.4\ \mu$s，$V_{\mathrm{server}} = 0.833$ bps なので，しきい値は $x_{\mathrm{opt}} = 1696$ byte $\simeq 1.70$ kbyte と設定した．

実験結果を表 1 に示す．表より，導出したしきい値付近で両方式の処理時間の比率が逆転している．よって，導出したしきい値を用いると，データ通信量に応じて最適な RPC 実行方式を選択できる．以上より，既存の OS 間システムコールと比べて，一定期間 busy wait を行わずに自動選択ができるので，平均的なリアルタイム性が向上できる．

表 1: 計測結果

転送量 [byte]	1	10	100	1k	10k	50k
busy wait 結果 [ms]	0.442	0.428	0.420	0.660	3.16	14.5
block wait 結果 [ms]	0.618	0.660	0.656	0.668	2.52	10.2
busy 結果 / block 結果	0.715	0.648	0.640	0.988	1.25	1.42

6 まとめ

本研究では，2 つの実行方式が存在する OS 間 RPC について，データ転送量に応じて実行方式を選択する手法を提案した．このために，全てのデータ範囲で処理時間が最適となるしきい値の条件を示した．

評価の結果，導出したしきい値を用いることで平均的なリアルタイム性を向上できることが確認された．

参考文献

[1] Anna R. Karlin et al. Empirical Studies of Competitve Spinning for A Shared-Memory Multiprocessor. *SIGOPS Oper. Syst. Rev.*, September 1991.

[2] Andrew Baumann et al. The Multikernel: A new OS architecture for scalable multicore systems. *SOSP'09.*