4P-6

密結合マルチプロセッサ用OS MUSTARD における プロセッサ間通信方式

渡邊 明子** 川口 浩美** 広屋 修一* 宮地 利雄*
* 日本電気(株) ** 日本電気技術情報システム開発(株)

1. はじめに

MUSTARD は, V60/V70 で構成される密結合型マルチプロセッサ・システムをターゲットとする, 組込みシステム用リアルタイム OS である. 交換機システムなどでの利用実績をもつシングルプロセッサ・システム用 RX616 [1] に対し, マルチプロセッサ・システム対応の機能拡張をおこなったものである.

本稿では MUSTARD で採用したプロセッサ間通信方式と, その実装について述べる.

2. プロセッサ間通信の種類

MUSTARD では以下にあげる機能で, プロセッサ間通信を使用している.

2.1 異なるプロセッサ上のタスクへの通信

マルチプロセッサ・システムにおいては, そのシステムコールの対象となるタスクが, 実際に実行中である場合がある. 従って異なるプロセッサ上で実行中のタスクを制御が必要となり, プロセッサ間通信を使用する.

(例) ● ras_exc : 指定タスクへの例外送付
● ter_tsk : 指定タスクの強制修了
● sus_tsk : 指定タスクの強制一時停止

2.2 システム内の全/一部のプロセッサの状態変更

マルチ化に伴い, 他のプロセッサのレジスタのクリアやセーブ, または稼働の開始/停止の実現のためにプロセッサ間通信を使用する.

(例) ● mnt_prc : 指定プロセッサの稼働開始
● umt_prc : 指定プロセッサの稼働停止
● rst_sys : レベルDによる再開処理 [2]

2.3 異なるプロセッサ上の I/O 操作

密結合マルチプロセッサでは, プロセッサはすべて均一と考えられているが, ハード構成により, 特定のプロセッサ上の I/O 操作が必要となり, プロセッサ間通信を使用する.

(例) ● rhd_cll : 指定プロセッサ上での処理ルーチン起動 (リモートハンドラ機能) [3]

3. ブロック型/ノンブロック型システムコール

RX616 におけるタスク間通信は, 構成上, すべて中断中のタスクに対しておこなわれ, 通信システムコールが修了したあとには, 対象タスクの状態はすでに変えられているとみなすことができた. MUSTARD において RX616 との互換性の観点から, 対象となるタスクが実際に実行中の場合には, 一般にはタスク間の同期が必要となるが, システム性能の向上にも配慮し, それぞれについて, 下記のような2種類のシステムコールを提供することとした.

(1) 要求した, 対象タスクの実行状態の変化が生じるまで待つ**ブロック型システムコール**

(2) 対象タスクに対して, 状態を変えるように要求をだすだけで, 実際に実行状態が変わるのを待つことなく, 直ちに終了してリターンする**ノンブロック型システムコール**

(1)のシステムコールをディフォルトとし, (2)を拡張システムコールとすることで, RX616 との安全な互換性を維持している.

4. プロセッサ間通信の統一的な扱い

システムにおいて, プロセッサ間通信を必要とするモジュールは多数あるが, これは概念的にはプロセッサ管理モジュール (図1参照) に対する次のような要求発行と見なすことができる.

I_P_Req (プロセッサ番号, 要求, パラメータ);

・プロセッサ番号　各プロセッサがユニークにもつ**プロセッサ番号**を通信相手の識別子となる
・要求　対象プロセッサへの処理要求フラグ (表1. に要求の一部をあげる.)
・パラメータ　要求処理に必要な**パラメータ**

表1. おもな要求フラグとその処理

フラグ	要求処理	システムコール
REQ_TER	カレントタスクのターミネイト (ブロック型)	ter_tsk
REQ_RHD	リモートハンドラの起動 (ブロック型)	rhd_cll
REQ_SUS	カレントタスクのサスペンド (ノンブロック型)	nw_sus_tsk
REQ_EXC	カレントタスクに例外発生 (ノンブロック型)	nw_ras_exc
REQ_UMNT	プロセッサの稼働停止	umt_prc
REQ_TLB	各プロセッサのアドレス変換キャッシュのクリア	cre_mpl

A Method for Inter-processor Communication on the Multiprocessor Operating System MUSTARD
Akiko WATANABE**, Hiromi KAWAGUCHI**, Shuuichi HIROYA*, Toshio MIYACHI*
*NEC Corporation, **NEC Scientific Information System Development Ltd.

このように，プロセッサ間通信をカーネルの最下層部のプロセッサ管理モジュールで扱うことにより，各モジュールの改造を最小限に押さえることができた．

MUSTARD の全体モジュール構造を，図 1 に示す．

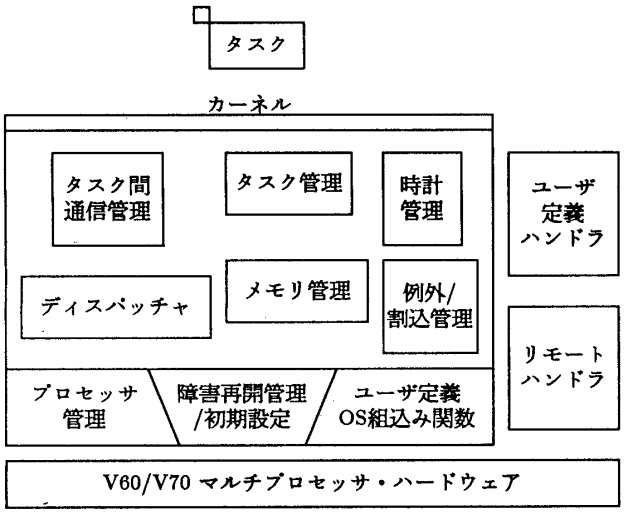

図 1．全体のモジュール構成とプロセッサ管理の位置

5. プロセッサ間通信の実現方式

5.1 プロセッサ制御ブロック

MUSTARDでは，プロセッサ制御ブロックPCB(Processor Control Block) によって各プロセッサを管理している． PCB は，各プロセッサの番号，ID，通信をおこなうための要求ビット列，要求処理に必要なパラメータ等を格納するフィールドを含んでいる．このプロセッサ制御ブロックにより，すべてのプロセッサからシステム内の任意のプロセッサの状態を知ることが可能である．プロセッサ間通信は PCB とプロセッサ間割込みを利用して実現されている．

5.2 要求の出し方

プロセッサ間の要求は，共有メモリ上の対象 PCB 内にある要求フィールドの該当ビット位置にフラグを立てることによって表示される．フラグを立てた後，要求を受ける側のセンスを促すために，対象プロセッサに対しプロセッサ間割込みを発生させる．

5.3 要求の受信

各プロセッサにだされた要求をセンスするため，プロセッサ間割込み，クロック割込み，およびシステムコールの出入口で，自プロセッサの PCB の要求フィールドをチェックし，なんらかのフラグが立っていることで，要求の存在が認識される．要求がセンスされた場合は，

クロック割込み ＞ 要求処理 ＞ システムコール

の優先順位で処理をおこなう．

次に，受信のトリガになる割込みについて述べる．クロック／プロセッサ間割込みについては，ハードウエア構成により一様ではないが， MUSTARD ではできるだけ柔軟に対応できるよう考慮している．

(1) クロック割込み

クロックの割込み方法は次の 3 つのいずれかを想定している．

(A) 固定で単一プロセッサに割込む
(B) 可変で単一プロセッサに割込む
(C) 全プロセッサに割込む

上記(A)，(B)の場合は，要求のセンスの遅れを縮小するため，プロセッサ間割込みの併用をおこなう．

(2) プロセッサ間割込み

プロセッサ間割込み機能の無いシステムにおいては，システムコールの出入口とクロック割込みでしか要求がセンスされないため，システムコール発行後，最悪の場合には，次の基本クロックの割込みがあるまで処理が遅延させられることになる．

5.4 要求の処理

要求をセンスしたプロセッサはその要求を記憶し，PCB の要求フラグをクリアし，待ち状態になっているタスクを解放する．続いて，記憶した要求のフラグに従った処理をおこなう．

5.5 ブロック型／ノンブロック型システムコールの処理の違い

3 で述べたように，要求を出した後の要求側の動きには 2 通りある．

タスク状態からブロック型システムコールが発行された場合には， PCB に要求フラグを立てた後，自ら PCB 内にあるキューに入って，待ち状態になる．このフィールドは，ブロック型システムコールを発行したタスクが，その要求をセンスされるのを待つために設けられている．このキューに入ったタスクは，要求がセンスされた時点で解放される．非タスク状態からブロック型システムコールが発行された場合には，立てた要求フラグが倒されるまでビジーウェイトで待つ．

ノンブロック型システムコールは，対象プロセッサの PCB に要求フラグを立てるだけである．

6. 終わりに

本稿では，MUSTARD のプロセッサ間通信について述べた．今後は実際のアプリケーションを起動させた場合のプロセッサ間通信発生頻度などを評価し，プロセッサ間通信要求の処理手順の改善をおこなう予定である．

参考文献

[1] 古城他, "V60リアルタイムOSの設計", 情処第33回全国大会1C-4, 1986.
[2] 川口他, "密結合マルチプロセッサ用OS MUSTARDの耐故障性の強化と障害処理機構", 情処第39回全国大会, 1989.
[3] 広屋他, "種々のハードウェアに適用可能な密結合マルチプロセッサ用OS MUSTARD", 情処第39回全国大会, 1989.