C-028

分散制御システムを対象とした分散リアルタイム OS のアスペクト指向プログラミングによる実現

A Distributed Real-Time Operating Systems Based on Aspect-Oriented Programming for Distributed Control Systems

石川 大貴 † 齋藤 亘弘 ‡ 兪 明連 † 横山 孝典 †
Hiroki Ishikawa Nobuhiro Saito Myungryun Yoo Takanori Yokoyama

1. はじめに

組み込み制御システムは多くの用途に使用されているため, アプリケーションに応じて多様な機能を持つリアルタイム OS(RTOS) が求められてきている．しかし, 組み込み制御システムには厳しいリソース制約があり, 要求されるすべての機能を一つの RTOS で提供することは困難で，アプリケーションに応じて必要最低限の機能を持つリアルタイム OS を用いることが望ましい．

そこで，アスペクト指向プログラミング (AOP)[1] を用いて, 用途向けに RTOS をカスタマイズする研究がなされている [2][3]．しかし，近年増えている分散制御システムに必要な分散処理機能には対応していない．

本論文では，自動車制御分野向けの標準 RTOS である OSEK OS[4] をベースに，アスペクト指向プログラミングを用いて分散リアルタイム OS を実現する手法を提案する．

2. 実現する分散リアルタイム OS

2.1. 分散リアルタイム OS の仕様

我々は既に，OSEK OS 仕様を拡張し，自ノード上のタスクのみでなく他ノード上のタスクに対しても同一のシステムコールを用いてタスク管理やイベント制御を可能とする，位置透過性のあるシステムコールを有する分散 RTOS を開発している [5]．本研究では，これと同一仕様の分散 RTOS を AOP により実現する．また，ネットワークとして FlexRay を用いたシステムを対象とする拡張 [6] と，CAN を用いたシステムを対象とする拡張を行い，広範囲の組み込み制御システムに対応可能とする．

OSEK OS 仕様で規定されているタスク管理，イベント制御，リソース管理，アラーム，割り込み管理，OS 実行制御，フックルーチンの 7 つのカテゴリのうちタスク管理とイベント制御に関するシステムコールに位置透過性を持たせる拡張を行う．具体的には，対象とするタスクを引数で指定する ActivateTask()，ChainTask()，GetTaskState()，SetEvent()，GetEvent() の 5 つのシステムコールについて，他ノード上のタスクを指定できるよう拡張する．また，OSEK OS 仕様ではタスク ID のデータ長は 1Byte と規定されているが，分散システム全体で一意にタスクを指定するため，1Byte のノード ID を付加した 2Byte のグローバルタスク ID に拡張した．上位 1Byte でノードの判別，下位 1Byte でタスクの判別を行う．

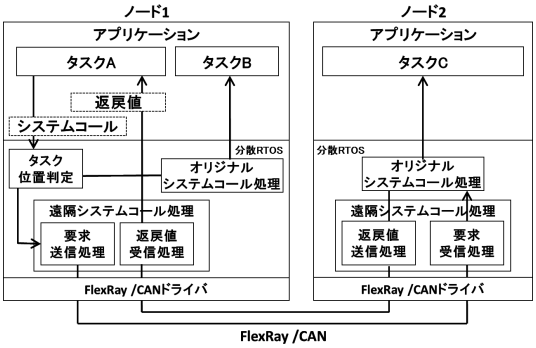

図 1: 遠隔システムコールを実現に必要な処理

2.2. 分散リアルタイム OS の構成

他ノード上のタスクに対するシステムコールを遠隔システムコールと呼ぶ．図 1 に示すように，遠隔システムコールを実現するには，タスク位置判定処理，要求送信処理，要求受信処理，返戻値送信処理，返戻値受信処理，ネットワークドライバ (FlexRay ドライバまたは CAN ドライバ) を追加する必要がある．

タスク位置判定処理はシステムコールの対象タスクがどのノード上にあるかを判定する．対象タスクが自ノード上の場合，OSEK OS 本来のシステムコールを実行する．

対象タスクが他ノード上の場合，遠隔システムコールを実行する．すなわち，要求送信処理が要求メッセージを生成し，ネットワークドライバを呼び出して送信する．要求メッセージを受信したノード上の要求受信処理は，ネットワークドライバを呼び出してメッセージを受け取り，要求されたシステムコールを実行する．システムコール実行後，返戻値送信処理が返答メッセージを生成し，ネットワークドライバを呼び出して，送信する．要求元ノードの返戻値受信処理は，ネットワークドライバを呼び出して，返信メッセージを受け取り，システムコール発行元のタスクへ返戻値を返す．

3. アスペクト指向プログラミングによる分散リアルタイム OS の実現

3.1. 実装方針

OSEK OS 仕様に基づく TOPPERS/ATK1[7] をベースに，アスペクトにより遠隔システムコールに必要な処理を織り込む．アスペクト指向言語には ACC(AspeCt-oriented C)[8] を用いる．

2.2 で述べた遠隔システムコールの実現に必要な処理のうち要求受信処理，返戻値送信処理，返戻値受信処

†東京都市大学
‡現在，スタンレー電気株式会社

```
StatusType around(GlbTaskType glbtskid):
call(StatusType ActivateTask(GlbTaskType)) && args(glbtskid) {

    if( check_target_location( glbtskid ) ) { /* タスク位置判定処理 */
        lock_cpu();
        global_syscal_flag = TRUE;
        request_sending_service( glbtskid,
              OSServiceId_ActivateTask, 0xFFFFFFFF ); /* 要求送信処理 */
        ercd = (UINT8)(get_return_value());
        if(ercd == E_OK){
            unlock_cpu();
            goto exit;
        } else {
            goto d_error_exit;
        }
    }
    tskid = (TaskType)( glbtskid & 0x00FF ); /* タスクID変換処理 */
    ercd = ActivateTask(tskid); /* オリジナルのシステムコール */
 exit: return(ercd);
    error_exit: lock_cpu();
 d_error_exit: エラー処理
    goto exit;
}
```

図 2: ActivateTask() のためのアスペクト

理は割り込みで実行するため，アスペクトを用いずに追加できる．

したがって，アスペクトにより織り込む処理は，タスク位置判定処理と要求送信処理の 2 つである．また，FlexRay と CAN で，メッセージ内容とドライバのインタフェースを同一とすることで，同じアスペクトを用いることができる．

3.2. アスペクトの記述

ActivateTask() の場合のアスペクトの記述を図 2 に示す．ソースコードは一部省略または簡略化している．call ポイントカットにより，織り込み箇所として ActivateTask() の呼び出しを指定し，args ポイントカットで引数 glbtskid(グローバルタスク ID) を取り出す．そして，around アドバイスにより，ActivateTask() の呼び出しをアドバイス内に記述した以下のような一連の処理に置き換える．

タスク位置判定処理の関数 check_target_location() を呼び出すと，対象タスクが他ノード上に存在する場合は true，自ノード上に存在する場合は false が返される．

対象タスクが他ノード上の場合は，lock_cpu() で割り込みを禁止し，遠隔システムコール発行中を示すフラグを立てる．そして，関数 request_sending_service() を呼び出して要求の送信を行う．関数 get_return_value() で返戻値を受け取り，正常な場合は unlock_cpu() で割り込み禁止を解除し exit ラベルへジャンプする．返戻値がエラーの場合は d_error_exit ラベルへジャンプする．

また，対象タスクが自ノード上の場合，グローバルタスク ID を元のタスク ID に変換して，オリジナルのシステムコールを呼び出し，返戻値をリターンする．

他のシステムコールについても同様にしてアスペクトを記述できる．

4. 実装及び評価

FlexRay を対象にした場合については既に報告している [6] ので，本論文では CAN を使用した場合について述べる．ハードウェアには H8S/2638 を搭載した評価ボードを用いた．CPU のクロック周波数は 20MHz，CAN の通信速度は 1Mbps である．アスペクトにより追加した，タスクが遠隔システムコールを発行してから要求の送信が完了するまでの処理について，その実行時間を測定した結果を表 1 に示す．AOP によるオーバヘッドを評価するため，直接ソースコードを修正した場合の値も示している．処理時間の増大は 0.8%以下であり，オーバヘッドは十分小さいと考えている．

表 1: 要求送信処理実行時間

システムコール	計測結果(μsec)	
	アスペクト指向プログラミングによる場合	直接ソースコードを修正した場合
ActivateTask	71.6	71.6
ChainTask	73.8	74.4
GetTaskState	72.9	72.3
SetEvent	73.6	73.6
GetEvent	72.9	72.3

5. おわりに

本論文では，AOP により分散処理機能を追加することで，ソースコードを直接書き換えることなく分散 RTOS を実現する手法について述べた．また，実装評価の結果，AOP によるオーバーヘッドは十分小さいことを確認した．今後は，AOP によるマルチコアプロセッサ向け RTOS の実現について検討する予定である．

謝辞

本研究で使用した TOPPERS/ATK1 の開発者と ACC の開発者に感謝する．本研究の一部は JSPS 科研費 24500046 の助成を受けたものである．

参考文献

[1] Kiczales, G., Lamping, J., Mendhekar A., Maeda, C., Lopes, C.,Loingtier, J. and Ir- win,J., Aspect-Oriented Programming, Proceeding of European Conference on Object-Oriented Programming 97, pp.220-242, 1997.

[2] F. Afonso, C. Silva, S. Montenegro and A. Tavares, Applying Aspects to a Real-Time Embedded Operating System, Proceedings of the 6th Workshop on Aspects, Components, and Patterns for Infrastructure Software, Article No.1 2007.

[3] D. Lohmann, W. Hofer, W. Schröder-Preikschat, and O. Spinczyk, Aspect-Aware Operating System Development, Proceedings of the 10th International Conference on Aspect-Oriented Software Development 2011, pp.69-80, 2011.

[4] OSEK/VDX, OSEK/VDX Operating System Version 2.2.3, 2005

[5] 知場貴洋，齊藤政典，伊丹悠一，兪明連，横山孝典 : 位置透過性のあるシステムコールを有する組み込み制御システム向け分散リアルタイム OS，情報処理学会論文誌，Vol53，No.12，pp.2702-2714,2012.

[6] 齋藤亘弘，兪明連，横山孝典 : アスペクト指向プログラミングに基づく分散リアルタイム OS，情報処理学会研究報告 Vol.2014-EMB-28, No.4,2014.

[7] TOPPERS/ATK1:http://www.toppers.jp/atk1.html

[8] AspeCt-oriented C, https://sites.google.com/a/gapp.msrg.utoronto.ca/aspectc/