6H-08

アスペクト指向プログラミングによる 並列・分散リアルタイムＯＳの共有リソース管理機能の実現

小平 裕太† 兪 明連† 横山 孝典†
†東京都市大学

1. はじめに

組み込みシステムは様々な用途に使用されるため，アプリケーション分野によってリアルタイム OS に要求される機能は異なる．ところが，組み込みシステムにはリソースの制約があり，要求されるすべての機能を単一のリアルタイム OS で提供することは困難であり，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．

そこで我々は，アスペクト指向プログラミング(AOP)[1]により，リアルタイムOSをカスタマイズすることで，リアルタイムOSのファミリー化を行ってきた．アスペクト指向プログラミングを用いることで，ソースコードを直接修正せずにカスタマイズできるとともに，重複した記述をなくすことができ，リアルタイムOSファミリーの構成管理が容易になる．これまでにスケジューリングアルゴリズムのカスタマイズ[2]や，並列・分散リアルタイムOS向けのタスク管理機能やイベント制御機能の追加等を行ってきた[3]．

本論文では，並列・分散リアルタイム OS 向け機能で残されている共有リソース管理機能を対象に，アスペクト指向プログラミングを用いて，マルチコアプロセッサ並列処理及び分散処理向けの共有リソース管理機能を実現する手法について述べる．

2. 並列・分散リアルタイム OS の機能

2.1 カスタマイズ対象機能

本研究では OSEK OS 仕様[4]に基づく TOPPERS/ATK1[5]を対象に，オリジナルのソースコードを直接書き換えることなくアスペクト指向プログラミングを用いて，並列・分散リアルタイムOSを実現する．我々は既にタスク管理とイベント制御に関するシステムコールを並列・分散向けにカスタマイズするアスペクトを提案している．

本論文では，リソース管理に関するシステムコールGetResource()とReleaseResource()をカスタマイズするアスペクトを提案する．これらのシステムコールは，引数で操作対象のリソースIDを指定して，そのリソースの獲得・解放を行うことで，排他制御を実現する．

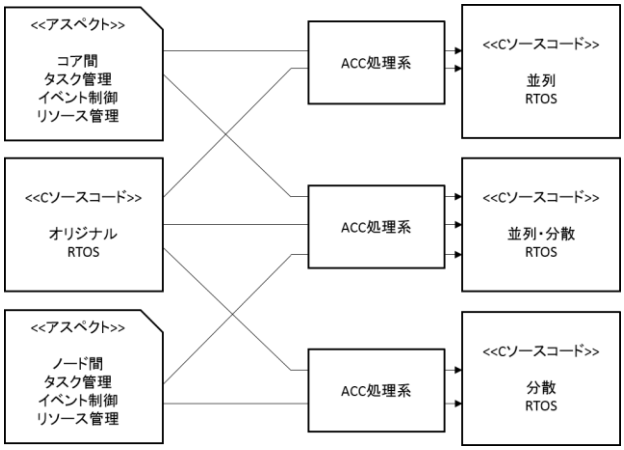

図 1 並列・分散機能の織り込み

2.2 アスペクトで追加する機能

我々は既にソースコードを直接修正して，同一CPU 内のリソース管理のみでなく，コア間のリソース管理機能[6]とノード間のリソース管理機能[7]を持つリアルタイム OS を開発している．リソース管理はコア内とコア間とノード間で処理が異なる．また対象リソースがコア内リソースかコア間リソースかノード間リソースかを判定する処理も必要である．そこで GetResource()，ReleaseResource()にコア間リソース管理処理，ノード間リソース管理処理，リソース判定処理の 3 つの処理をアスペクトによって織り込む．

コア間のリソース管理は MSRP(Multiprocessor Stack Resource Policy)[8]をベースに実装する．MSRP はコア間で共有するコア間リソースを獲得するタスクの優先度をそのタスクが存在するコア内で最高の優先度とする．また，獲得したいコア間リソースがほかのタスクに占有されていた場合は，スピンロックでコア間リソースの開放を待つ．コア間リソースを解放すると，タスクの優先度は元に戻る．

ノード間のリソース管理はメッセージの衝突が発生しない CAN ネットワークの機能を利用して排他制御を行う．具体的には，ノード間リソースの獲得・解放要求メッセージをネットワーク上に送信した順にリソースの獲得・解放を行う．また MPCP(Multiprocessor Priority Ceiling Protocol)[9]に基づき，ノード間リソースの上限優先度をそのノード間リソースを利用する全ノード上のタスクの中で最高の優先度とし，ノード間リソースを獲得したタスクの優先度を上限優先度に変更する．

Aspect-Oriented development of Parallel and distributed Real-time Operating Systems with Shared Resource Management Mechanisms

† Yuta Kodaira ,Myungryun Yoo ,Takanori Yokoyama , Tokyo City university

```
around():execution(StatusType GetResource(ResourceType)) {

//変数宣言
StatusType ercd = E_OK;
ResourceType resid_glbget = 0, resid_distget =0;

LOG_GETRES_ENTER(resid);//デバッグ用

CHECK_CALLEVEL(TCL_TASK | TCL_ISR2);コールレベルの確認

if(resid < dnum_node) {
  resid_distget = resid;
} elseif ((resid&MIN_GLBRESID) == MIN_GLBRESID) {
  resid_glbget = resid;
} else {
  ローカルのリソース管理処理
}

終了処理;
}
```

リソース判定処理

図 2 リソース判定処理アスペクト

```
after():set(ResourceType resid_glbget)&&infile("common_get$") {

リソースIDの確認;
タスクの優先度が上限優先度以下か確認;
獲得したいリソースが他のタスクに占有されていないか確認;
タスクが既に共有リソースを獲得していないか確認;
優先度変更;
}
```

リソース管理処理

図 3 コア間リソース管理処理アスペクト

3. アスペクト

3.1 アスペクトによる織り込み

図 1 にアスペクトによる並列・分散機能の実装のイメージを示す．並列・分散両者を対象としたリアルタイム OS のみでなく，並列のみを対象としたリアルタイム OS，分散のみを対象としたリアルタイム OS も実現する．

コア間のリソース管理機能とノード間のリソース管理機能をそれぞれアスペクトで定義し，コア間リソース管理機能のアスペクトを織り込むことで並列リアルタイム OS，ノード間リソース管理機能のアスペクトを織り込むことで分散リアルタイム OS，両者を織り込むことで並列・分散リアルタイム OS を実現する．アスペクト指向言語には ACC(AspeCt-oriented C)[10]を用いる．

3.2 アスペクト記述

GetResource()を例として，アスペクトの記述について説明する．図 2 はリソース判定処理のアスペクトを示す．このアスペクトは，GetResource()の実行部分を execution ポイントカットで指定し，around アドバイスによって処理を置き換えている．リソース判定処理ではリソース ID がノード間リソースであれば resid_distget にリソース ID を格納し，コア間リソースであれば resid_glbget にリソース ID を格納する．コア内リソースであればオリジナルのシステムコールを実行する．

図 3 にコア間リソース管理処理のアスペクトを示す．リソース判定処理アスペクト内の変数 resid_glbget にリソース ID を格納する箇所を指定し，after アドバイスでその後にリソース管理処理を織り込む．

ノード間リソース管理処理のアスペクトも同様に変数 resid_distget にリソース ID を格納する箇所を指定し，after アドバイスでその後にノード間のリソース管理処理を織り込む．

4. おわりに

アスペクト指向プログラミングを用いてリアルタイム OS ファミリーを実現する研究の一環として，本論文では，並列・分散処理環境におけるリソース管理機能を追加する手法を提案した．現在，提案した手法で実装したアスペクトの性能評価を行っている．

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JP15K00084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., Lopes, C.Loingtier, J.M.and Ir-win, J., Aspect-Oriented Programming, Proceed-ings of the 11th European Conference on Object-Oriented Programming, pp.220-243(1997).

[2] 原田祐輔，阿部一樹，兪明連，横山孝典，アスペクト指向プログラミングによるリアルタイム OS スケジューラのカスタマイズ，情報処理学会論文誌 Vol.57,No.8,pp.1752-1764(2016)

[3] 原田祐輔，石川大貴，兪明連，横山孝典，アスペクト指向プログラミングによるリアルタイム OS ファミリーの実現，情報処理学会研究報告，Vol.2017-EMB-44,No.32,pp.1-6(2017)

[4] OSEK/VDX : Operation System Version 2.2.3 , (2005)

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

[6] Ishibashi, K., Yokoyamam K., Yoo, M. and Yokoyama T., A Real-Time Operating System with Location-Transparent Shared Resource Management for Multi-Core Processors, Proceedings of the 13th IEEE International Conference on Embedded Software and Systems, pp.93-98(2016)

[7] Ishibashi, K., Yoo, M. and Yokoyama T., A Real-Time Operating System with CAN-Based Inter-Node Shared Resource Management and Distributed Shared Memory, Proceedings of the 14th IEEE International Conference on Embedded Software and Systems, pp.798-805(2017)

[8] Gai, P., Lipari, G.and Natale, M.D.: Minimizing Memory Utilization of Real-Time Task Sets in Single and Multi-Processor Systems-on-a-chip, Proceedings of the 2001 IEEE Real-Time Systems Symposium,pp.71-83(2001)

[9] Rajkumar, Real-time Synchronization Protocols for Shared Memory Multiprocessors, Proceedings of the 10th International Conference on Distributed Computing Systems,pp.116-123(1990)

[10]https://sites.google.com/a/gapp.msrg.utoronto.ca/aspectc/