Vol.2022-EMB-60 No.2
2022/7/8

アスペクト指向プログラミングによる 並列・分散アラーム管理機能を有する RTOS の実現

溝呂木裕規 [1]　横山孝典 [1]　兪明連 [1]

概要：我々は，アプリケーションの要求に応じて必要最小限の機能を備えた RTOS を提供するため，並列および分散処理に対応できる分散マルチコア RTOS ファミリを，アスペクト指向プログラミングを用いて開発している．本論文では，これまで実現したタスク管理やイベント制御に加え，異なる CPU コアや異なるノードに対して，時刻を指定したタスクの起動やイベント設定を可能とするため，アラーム管理用システムコールを CPU コア間およびノード間で発行可能とするためのアスペクトを提案する．具体的には，2 種類のアスペクトによる実現方法を提案して，2 つのバージョンの RTOS を開発し，CPU 実行時間と記述量の比較を行うことで評価する．そして，実用上問題のないオーバヘッドで実現可能であることを示す．

キーワード：組込みシステム，リアルタイム OS．アスペクト指向プログラミング，並列処理，分散処理

1. はじめに

多くの組み込み制御システムは，制御処理が時間制約を満たせなかった場合に重大な損害が発生する可能性のあるハードリアルタイムシステムである．ハードリアルタイムシステムをサポートするためにリアルタイム OS(RTOS)が用いられる．ただし，組み込み制御システムは自動車や産業用機器等の多様なアプリケーションに使用されるため，RTOS に要求される機能はアプリケーションによって異なることが多い．特に，近年，組み込み制御システムにおいてマルチコアプロセッサが使用されるとともに，ネットワークで複数のコンピュータを接続し，分散させて制御処理を行う分散システムが増加傾向にある．そのため，組み込み制御システムに搭載されているソフトウエアは大規模化，複雑化し，組み込みソフトウエアの開発量が増大している．しかし，リソース消費量の制約が大きい組込みシステムにおいて，多様なアプリケーションに要求されるすべての機能を備えた RTOS を搭載する事は望ましくない．したがって，単一の RTOS ではなく，それぞれのアプリケーションの要求に応じて必要最小限の機能を備えた RTOS を提供できる RTOS ファミリが望まれる．

横断的関心事をアスペクトとして分離するアスペクト指向プログラミング(AOP)[1]を用いた RTOS の拡張やファミリ化の研究が行われている．Beuche, Spinczyk らは、アーキテクチャに依存しないコンポーネントで RTOS ファミリを開発する手法を提案した[2][3]．Afonso らは，RTOS の同期とロギングに対して AOP を適用した[4]．Park らは、プログラミング言語に依存しないプログラミング環境 AOX を備えたカスタマイズ可能な RTOS を開発した[5]. Lohmann らは，アスペクト指向プログラミング言語 AspectC++[6]を用いた RTOS ファミリの開発を行い，そのオーバヘッドが小さいことを示した[7]．また，はじめからアスペクトを意識した設計である Aspect-Aware Design を提唱し [8]，AUTOSAR OS[9]仕様で定義されている機能の取外しが可能な RTOS ファミリの開発を行った[10]．しかし，Lohmann らの研究は AUTOSAR OS の仕様内でのカスタマイズを目的としたもので，RTOS の仕様で定義されていない機能の拡張には対応していない．

そこで我々は，アスペクト指向プログラミングを用いて，当初の RTOS とする仕様では規定されていない機能拡張を可能とする RTOS のファミリ化の研究を行ってきた．そしてこれまでに OSEK OS[11]仕様に基づく RTOS の TOPPERS/ATK1 [12]を対象に, 固定優先度スケジューリングから動的優先度スケジューリングである EDF(Earliest Deadline First)スケジューリングに変更するアスペクトや RMCL(Rate Monotonic Critical Laxity)スケジューリング[13]に変更するアスペクトを提案した[14] [15] [16]．また，組み込み制御システム分野では，分散制御システムや，非対称型あるいは機能分散型のマルチコアプロセッサ向けの RTOS が求められていることから，アスペクト指向プログラミングを用いて，マルチコア並列処理向けの CPU コア間

1　東京都市大学
Tokyo City University, Tokyo 158-8557, Japan
{g2181486, tyoko, myoo}@tcu.ac.jp

システムコール，分散処理向けのノード間システムコールを追加するアスペクトを提案した[17][18].

本論文では，アスペクト指向プログラミングを用いることで、ベースとする RTOS のソースコードを直接修正せずに，重複記述が少なく構成管理が容易な RTOS ファミリを実現する手法を提案する．

2. 拡張機能対象

2.1 拡張対象のシステムコール

RTOS ファミリの機能はフューチャモデルで表現することができる．Lohmann らの RTOS のフィーチャモデルの一部を図 1 に示す．“RTOS”と“システムコール”のようにフィーチャ間が実線で結ばれている場合，親フィーチャにとって子フィーチャが必須の要素であることを表している．また，“システムコール”と“タスク管理”のようにフィーチャ間が実線と円形で結ばれている場合，親フィーチャにとって子フィーチャが取捨選択可能な要素であることを表している．このように Lohmann らは AUTOSAR OS の仕様内での機能の取捨選択を可能とした．

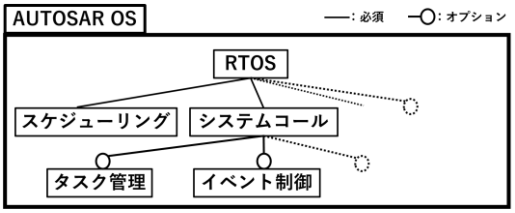

図 1　Lohmann らの RTOS ファミリのフィーチャモデル

一方，我々は OSEK OS 仕様では規定されていない CPU コア間システムコールやノード間システムコールをアスペクトにより実現する研究を行ってきた．OSEK OS はタスク管理，イベント制御，アラーム管理，リソース管理，割り込み管理，OS 実行制御等の機能を提供している．これらのうち，マルチコア並列分散処理や分散処理に対応するための拡張が考えられるのは，タスク管理，イベント制御，アラーム管理，リソース管理である．それらのうち，これまでに実装した機能と未実装である機能を表 1 に示す．OSEK OS の機能のうち，タスクの起動と停止等を行うタスク管理とイベントの設定等を行うイベント制御を CPU コア間およびノード間で発行可能とするアスペクトをこれまでに提案した．本論文では，残された機能のうち，周期タスクの設定を行う際に用いられるアラーム管理を対象に機能の拡張を行う．具体的には，アラーム管理用システムコールを CPU コア間およびノード間で発行可能とするためのアスペクトを提案する．

本論文で開発する RTOS ファミリのフィーチャモデルの一部を図 2 に示す． OSEK OS では提供されない CPU コア間およびノード間でのアラーム管理用システムコールを発行する機能を取捨選択して追加することを可能とする．

表 1　アスペクトによる機能拡張の対応状況

OSEK OSの機能	CPUコア間 システムコール対応	ノード間 システムコール対応	主な用途
タスク管理	○	○	タスクの起動・停止
イベント制御	○	○	イベントの設定
アラーム管理	×	×	周期タスクの設定
リソース管理	×	×	リソースの獲得・解放

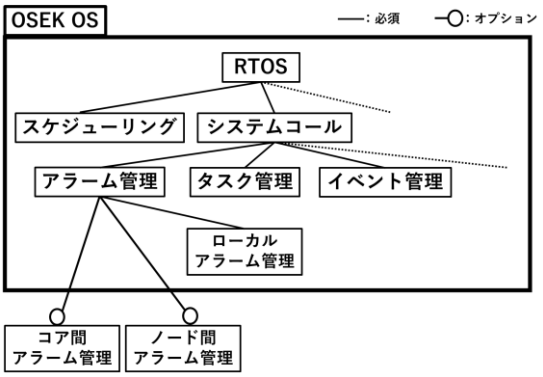

図 2　本 RTOS ファミリのフィーチャモデル

2.2 アラーム管理機能の拡張

アラームは特定のタイミングで発生するイベントを扱う機能である．アラームの代表的な使い方として，OS のシステム時刻をカウントするシステムカウンタを用いて，設定された時刻に周期タスクの起動やイベントの設定を行うことが挙げられる．

OSEK OS のアラーム管理機能には，5 つのシステムコールがある．本論文での拡張対象のシステムコールを表 2 に示す． GetAlarm は，指定するアラームが満了するまでの相対ティック値を取得する．GetAlarmBase は，指定するアラームの最大許容カウント値等の 3 つのティック値を含む構造体のポインタを取得する．SetRelAlarm は，指定ティック値経過後，指定するアラームを指定周期でアラームを設定する．SetAbsAlarm は，絶対時刻と指定ティック値が同値の時，指定するアラームを指定周期でアラームを設定する．CancelAlarm は，指定するアラームを取り消す．これらのシステムコールを対象に CPU コア間およびノード間で発行できるよう拡張する．また，CPU コア間システムコールおよびノード間システムコールをサポートするため，アラーム ID に CPU コア ID とノード ID を含むように拡張して，対象アラームの位置を判定できるようにする．

表 2 拡張対象のシステムコール

システムコール	入力パラメータ	出力パラメータ
GetAlarm	アラームID	相対ティック値
GetAlarmBase	アラームID	3つのティック値を含む構造体のポインタ
SetRelAlarm	アラームID，相対ティック値，周期	
SetAbsAlarm	アラームID，絶対ティック値，周期	
CancelAlarm	アラームID	

2.3 CPU コア間システムコール

CPU コア間システムコールとは，ある CPU コア上のタスクが同一マルチコアプロセッサ上の異なる CPU コア上のアラームやタスクを対象に発行するシステムコールである．CPU コア間システムコールの動作について図 3 を用いて説明する．ノード 0 の CPU コア 0 上のタスク 1 が，ノード 0 の CPU コア 1 上のアラーム 2 を指定して，システムコール SetRelAlarm を呼び出してからリターンするまでの動作を示す．点線は左から右方向に時間経過を示す時間軸である．

CPU コア 0 上のタスク 1 がシステムコールを発行すると，RTOS はアラーム固有の番号であるアラーム ID を参照して，指定するアラームの位置を対象のアラームの位置を特定する．アラーム 2 は同一のノード上の別の CPU コア上にあるため，共有メモリ経由で要求送信処理を行う．共有メモリ経由の要求送信処理では，大きく 2 つの処理を行う．まず，入力や要求されたシステムコールを示すシステムコール ID や対象のアラーム等を含む要求データを CPU コア 0 と CPU コア 1 で共有するメモリに書き込む. 次に，CPU コア 1 に対して割込みを発行し，タスク 1 はビジーウエイトで待機する．コア間システムコールの場合は，タスクスイッチの時間の方が CPU コア間通信の時間より長くなる場合があるため，ビジーウエイトを採用する．

CPU コア 1 上の割込み処理は共有メモリ上の要求データを取得し，システムコールを発行し，アラームの設定を行う．設定後，CPU コア 1 上の割込み処理は， CPU コア 0 上のタスク 1 への返戻値を共有メモリに書き込む．タスク 1 のビジーウエイトは解除され，共有メモリ上の返戻値を読み出し，リターンする．

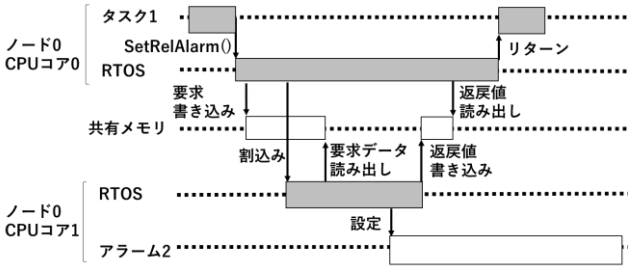

図 3　CPU コア間システムコールの動作

2.4 ノード間システムコール

ノード間システムコールとは，あるノード上の CPU コア上のタスクが同一のネットワーク上の異なるノード上の CPU コア上のアラームやタスクを対象に発行するシステムコールである．

ノード間アラーム管理システムコールの動作について，図 4 を用いて説明する．ノード 0 の CPU コア 0 のタスク 2 が，ノード 1 の CPU コア 1 のアラーム 3 を指定して，システムコール SetRelAlarm を呼び出してからリターンするまでの動作を示す．点線は左から右方向に時間経過を示す時間軸である．

ノード 0 の CPU コア 0 のタスク 2 がシステムコールを発行すると RTOS はアラーム固有の番号であるアラーム ID を参照して，指定するアラームの位置を特定する．アラーム 3 は異なるノード上の CPU コア上にあるため，ネットワーク経由で要求送信処理を行う．ネットワーク経由の要求送信処理では，以下の処理を行う．最初に，入力や要求されたシステムコールを示すシステムコール ID，対象のアラーム等を含む要求データを CAN 通信で送信する. そして，タスク 2 を待ち状態に遷移させる．ノード間システムコールでは，通信時間が長くなるため，待ち状態に遷移してほかのタスクを実行可能とする．

ノード 1 の CPU コア 1 の CAN 受信割込み処理は要求データを取得して，システムコールを実行し，アラームの設定を行う．設定後，CAN 受信割込み処理はネットワーク経由の返戻値を ノード 0 の CPU コア 0 に対して送信する．ノード 0 の CPU コア 0 の CAN 受信割込み処理はタスク 2 を実行可能状態に遷移させる. タスク 2 は返戻値を取得し，リターンする．

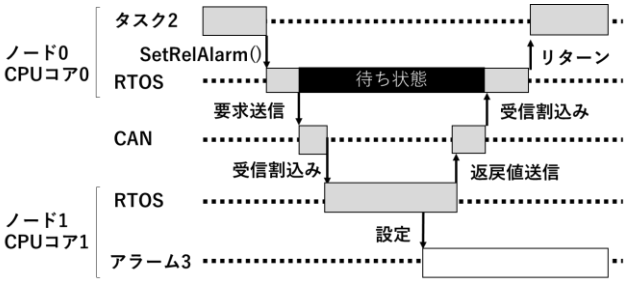

図 4　ノード間システムコールの動作

3. アスペクトによる機能拡張

3.1 アスペクト指向プログラミングによる RTOS の拡張

本論文では，アスペクト指向プログラミングを用いて，拡張する機能をオリジナルのソースコードと分離してアスペクトで記述し，織り込むことでソースコードを直接修正することなく機能を拡張する．具体的には、図 3 の送信側であるノード 0 の CPU コア 0 の RTOS の要求書き込みおよび返戻値読み出しの処理と、図 4 の送信側であるノード 0 の CPU コア 0 の RTOS の要求送信および返戻値取得の処理を、アスペクトを用いてオリジナルのシステムコールに織り込む．なお、割込み処理は既存のソースコードに織り込む必要はないため、アスペクトを用いずに追加する．必要機能のアスペクトを取捨選択して織り込むことで，アプリケーションの要求に応じて必要最小限の機能を備えた RTOS を提供できる．

カスタマイズ RTOS のオブジェクトファイルを生成するまでの流れを図 5 に示す. ベースとする TOPPERS/ATK1 のソースファイルが C 言語で記述されているため，C 言語ベースのアスペクト指向プログラミング言語 ACC(AspeCt-oriented C)[19][20]を用いる．そして ACC コンパイラにより織り込み後，C コンパイラを用いてオブジェクトファイルを生成する．

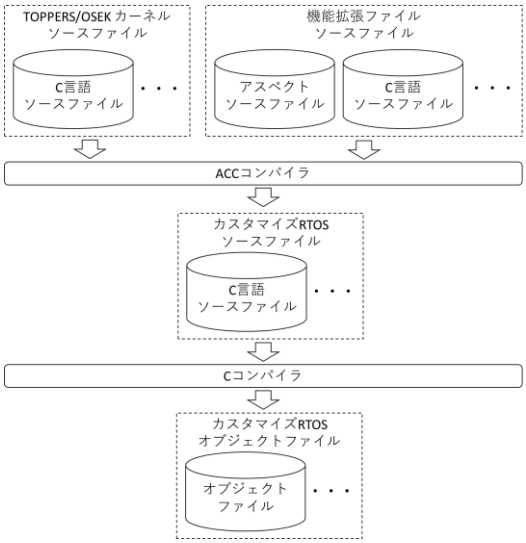

図 5　アスペクトによる機能の拡張

アスペクトには，オリジナルのソースコード中の処理を織り込む箇所であるジョインポイントを指定するポイントカット，織り込む処理の内容であるアドバイスを記述する．ポイントカットには，メソッドの実行を指定する execution やメソッドの呼び出しを指定する call, 指定する変数への書き込みを指定する set，指定する変数への読み出しを指定する get 等がある．アドバイスには，ジョインポイントと代替して織り込む around アドバイス, ジョインポイントの直前に処理を織り込む before アドバイス，ジョインポイントの直後に処理を織り込む after アドバイスの 3 種類がある．

3.2 アラーム管理機能用アスペクト

本論文では，オリジナルのシステムコールに対して around アドバイスを用いて機能を拡張するアスペクトと，オリジナルのシステムコールに対して before アドバイスを用いて機能を拡張するアスペクトの 2 種類を提案する．以下，前者を用いて開発する RTOS を第 1 アスペクト版 RTOS, 後者を用いて開発する RTOS を第 2 アスペクト版 RTOS と呼ぶ．

3.2.1 第 1 アスペクト版 RTOS

第 1 アスペクト版 RTOS では, 1 つのシステムコールに対して，共通アスペクトと CPU コア間要求アスペクト，ノード間要求アスペクトの 3 つのアスペクトを記述する．対象のシステムコールは 5 つなので，全部で 15 のアスペクトがある．

オリジナルのシステムコール SetRelAlarm に対して around アドバイスを用いて，織り込まれる CPU コア間システムコールとノード間システムコールで共通する内容を記述した第 1 共通アスペクトを図 6 に示す．1 行目の around アドバイスは，処理内容を代替することを表し，execution(StatusType SetRelAlarm(GlbAlarmType, TickType, TickType)) で，execution ポイントカットを用いて，SetRelAlarm の関数が実行されるときを指定している．args ポイントカットを用いて，アスペクト内で引数を利用するために指定する引数を受け取る. 4 行目の if 文で引数として与えられたアラーム ID(glbalmid)を参照して自ノードかどうかの位置判定処理を用いて行っており，14 行目で位置判定処理関数 check_alarm_location で位置を判定し, その結果を変数 alm_location に書き込んでいる．

```
1  StatusType around(GlbAlarmType glbalmid, TickType incr, TickType cycle):
   execution(StatusType SetRelAlarm(/*引数 省略*/))&&args(/*引数 省略*/) {
2  /*エラーチェック　省略*/
3
4  if( (NODE_ID(glbalmid) == MY_NODE_ID) || (NODE_ID(glbalmid) == 0)){
5          /*エラーチェック 省略*/
6          almcb_almval[ALMID(glbalmid)] = add_tick(cntcb_curval[cntid],
           incr,cntinib_maxval2[cntid]);
7          almcb_cycle[ALMID(glbalmid)] = cycle;
8          enqueue_alarm(ALMID(glbalmid), cntid);
9          goto exit;
10 }else{
11         glbalmid_store = glbalmid;
12         cycle_store = cycle;
13         incr_store = incr;
14         alm_location = check_alarm_location(glbalmid);
15         /*エラーチェック 省略*/
16 }
17 /*エラーフックルーチン,exit 等　　省略*/
18 }
```

図 6　第 1 共通アスペクト

そのほかのシステムコールについても, SetRelAlarm と同様に，各アスペクトが ACC によって織り込まれる．

CPU コア間システムコールに依存する内容を記述した第 1 CPU コア間要求アスペクトを図 7 に示す．このアスペクトは第 1 共通アスペクトに記述された変数 alm_location への書き込みに対して after アドバイスを用いて, CPU コア間システムコールの処理を織り込む. set(StatusType alm_location) は, ポイントカット set を用いて, 前述した変数 alm_location への書き込みを織り込み箇所に指定している．また，ポイントカット infile を用いて，織り込む対象のソースファイルを指定するソースファイルに限定している．これにより，図 7 の 2 行目から 5 行目までの処理が，図 6 の 14 行目の位置判定処理関数 check_alarm_location() 実行終了直後に織り込まれる. 図 7 の 2 行目の if 文では, 位置判定結果が，同一ノード上の異なる CPU コアであることを表す CALL_MULTI であるかを判定している．4 行目の request_sending_multi_service() は，共有メモリにデータを書

き込み，ほかの CPU コアに対して割込みを入れる関数である．6 行目の関数 get_multi_return_value(runtsk) は，他の CPU コアで実行されたシステムコールの返戻値を取得する関数である．

```
1  after(): set(StatusType alm_location)&&infile(/*ファイル位置 省略*/) {
2      if(alm_location == CALL_MULTI) {
3          request_sending_multi_service(glbalmid_store,
           OSServiceId_SetRelAlarm, incr_store, cycle_store);
4          ercd = (UINT8)(get_multi_return_value( runtsk ));
5      }
6  }
```

図 7　第 1 CPU コア間要求アスペクト

そのほかのシステムコールも，SetRelAlarm と同様に，各アスペクトが ACC によって織り込まれる．

ノード間システムコールに依存する内容を記述した第 1 ノード間要求アスペクトを図 8 に示す．このアスペクトは第 1 共通アスペクトに記述された変数 alm_location への書き込む．2 行目の if 文では，位置判定結果が，異なるノード上の CPU コアであることを表す CALL_DIST かを判定している．3 行目では，ノード間システムコール発行中を表すフラグを立てている．4 行目の関数 request sending service() は，メッセージボックスにデータを書き込み，CAN 通信でメッセージを送信するノード間要求関数である．

```
1  after(): set(StatusType alm_location)&&infile(/*ファイル位置 省略*/) {
2      if(alm_location == CALL_DIST) {
3          global_syscal_flag = TRUE;
4          request_sending_service( glbalmid_store,
           OSServiceId_SetRelAlarm, cycle_store , incr_store );
5          ercd = (UINT8)(get_return_value(runtsk));
6      }
7  }
```

図 8　第 1 ノード間要求アスペクト

ほかのシステムコールも，SetRelAlarm と同様に，各アスペクトが ACC によって織り込まれる．

3.2.2 第 2 アスペクト版 RTOS

第 2 アスペクト版 RTOS は，第 1 アスペクト版 RTOS よりソースコードの記述量を削減することを目的としている．第 1 アスペクト版 RTOS とは異なり，システムコール毎に記述するのは共通アスペクトのみである． CPU コア間要求アスペクトとノード間要求アスペクトはシステムコールに依存しないため，2 つのアスペクトで記述できる．対象のシステムコールは 5 つなので全部で 7 つのアスペクトがある．

オリジナルのシステムコール SetRelAlarm に対して before アドバイスを用いて，CPU コア間システムコールとノード間システムコールで共通する内容を記述した第 2 共通アスペクトを図 9 に示す．1 行目の before アドバイスは，処理内容を代替することを表し， execution(StatusType SetRelAlarm(GlbAlarmType*, TickType, TickType)) で，execution ポイントカットを用いて，SetRelAlarm の関数が実行されるときを指定している．アスペクト内で引数を利用するため，args ポイントカットを用いて，指定する引数を受け取る．4 行目でアラーム固有の番号であるアラーム ID を参照して特定する位置判定処理関数 check_alarm_location を実行する．5 行目で対象アラームが同一ノード上の他の CPU コアに存在するかを判定し，その場合は，CPU コア間要求 request_sending_multi_service を行う．また，6 行目で対象のアラームが別のノード上の CPU コアに存在するかを判定し，その場合は，ノード間要求 request_sending_service を行う．

```
1  before(GlbAlarmType *glbalmid, TickType incr, TickType cycle):
   execution(StatusType SetRelAlarm (/*引数 省略*/))&&args(/*引数 省略*/) {
2      StatusType alm_location;
3      lock_cpu();
4      alm_location = check_alarm_location(*glbalmid);
5      if(alm_location == CALL_MULTI)
       request_sending_multi_service(*glbalmid,
       OSServiceId_SetRelAlarm, incr_store, cycle_store);
6      if(alm_location == CALL_DIST) request_sending_service( *glbalmid,
       OSServiceId_SetRelAlarm, incr_store, cycle_store );
7      unlock_cpu();
8  }
```

図 9　第 2 共通アスペクト

そのほかのシステムコールについても，SetRelAlarm と同様に，各アスペクトが ACC によって織り込まれる．

CPU コア間システムコールに依存する内容を記述した第 2 CPU コア間要求アスペクトを図 10 に示す．このアスペクトは第 2 共通アスペクトに記述された CPU コア間要求関数 request_sending_multi_service に対して before と after アドバイスを用いて，CPU コア間システムコールの処理を織り込む．第 2 CPU コア間要求アスペクトはシステムコールによらず，共通である．2 行目では，CPU コア間システムコール発行中を表すフラグを立てている．6 行目の関数 get_multi_return_value(runtsk) は，異なる CPU コアで実行されたシステムコールの返戻値を取得する．7 行目で，オリジナルのシステムコール SetRelAlarm の処理が行われることを防ぐため，CPU コア間要求またはノード間要求を行った場合に限り，アラーム ID を NULL にする．

```
1  before(GlbAlarmType *glbalmid, OSServiceIdType srvc_id, UINT32 argv,
   UINT32 argv2): execution(void request_sending_service(/*引数 省略*/)) &&
   args(/*引数 省略*/){
2      multi_syscal_flag = TRUE;
3  }
4
5  after(GlbAlarmType *glbalmid, OSServiceIdType srvc_id, UINT32 argv1,
   UINT32 argv2): execution(void request_sending_multi_service(/*引数 省略
   */)) && args(/*引数 省略*/){
6      ercd = (UINT8)(get_multi_return_value( runtsk ));
7      if(alm_location == CALL_MULTI || alm_location == CALL_DIST)
       *glbalmid = NULL;
8  }
```

図 10　第 2 CPU コア間要求アスペクト

ノード間システムコールに依存する内容を記述した第 2 ノード間要求アスペクトを図 11 に示す．このアスペクトは第 2 共通アスペクトに記述されたノード間要求関数 request_sending_service に対して before と after アドバイスを用いて，ノード間システムコールの処理を織り込む．第 2 ノード間要求アスペクトはシステムコールによらず，共通である．2 行目では，ノード間システムコール発行中を表すフラグを立てている．8 行目では，図 10 の 7 行目と同様にオリジナルのシステムコール SetRelAlarm の処理が行われることを防ぐため，CPU コア間要求またはノード間要求を行った場合に限り，アラーム ID を NULL にする．

```
1  before(GlbAlarmType *glbalmid, OSServiceIdType srvc_id, UINT32 argv,
   UINT32 argv2): execution(void request_sending_service(/*引数 省略*/)) &&
   args(/*引数 省略*/){
2      global_syscal_flag = TRUE;
3  }
4
5  after(GlbAlarmType *glbalmid, OSServiceIdType srvc_id, UINT32 argv,
   UINT32 argv2): execution(void request_sending_service(/*引数 省略*/)) &&
   args(/*引数 省略*/){
6      ercd = (UINT8)(get_return_value(runtsk));
7      if(alm_location == CALL_MULTI || alm_location == CALL_DIST)
       *glbalmid = NULL;
8  }
```

図 11　第 2 ノード間要求アスペクト

第 2 アスペクト版 RTOS では，CPU コア間システムコールまたはノード間システムコールが呼び出された場合，コンテキストの確認のエラーチェックを送信側 CPU では行わず，CPU コア間要求またはノード間要求を実行したのちに，受信側 CPU で行う．そのため，送信側 CPU で不正なコンテキストであった場合でも，システムコールを発行することが可能となってしまう問題があり，その解決は今後の課題である．

4. 評価

4.1 評価方法

評価に用いる実験環境を表 3 に示す．ベースとする OS は，TOPPERS/ATK1 Release 1.0 である．コンパイラには，C/C++ compiler package for SuperH RISC engine familyV.9.04 Release 03 を用いる．エミュレータには，SH2A-DUAL E10A-USB Emulator version 2.02.00 を用いる．AOP コンパイラは，ACC ver.0.9 を用いる．

表 3　実験環境

項目	内容
マイクロコンピュータボード	M3A-HS50
プロセッサ	SH7205 × 2
内蔵メモリ	高速内蔵 RAM0 (64K バイト)
	高速内蔵 RAM1 (32K バイト)
共有メモリ	内蔵 RAM (16K バイト)
ネットワーク	RCAN
統合開発環境	High-performance Embedded Workshop
ベースOS	TOPPERS/ATK1 Release 1.0
コンパイラ	C/C++ compiler package for SuperH RISC engine family V.9.04 Release 03
エミュレータ	SH2A-DUAL E10A-USB Emulator version 2.02.00
AOP コンパイラ	ACC ver.0.9

評価項目は 2 つで，1 つ目は CPU 実行時間の比較である．ソースコードを直接修正して機能を拡張した直接修正版 RTOS (以下，直接修正版)，第 1 アスペクト版 RTOS，第 2 アスペクト版 RTOS の 3 つの RTOS について，CPU コア間およびノード間システムコールの CPU 実行時間を 100 回ずつ測定して，その平均を求め，比較する．測定の精度は 0.03 μ 秒である．

2 つ目の評価項目はソースコードの記述量の比較である．直接修正版と第 1 アスペクト版 RTOS と第 2 アスペクト版 RTOS の 3 つの場合について記述量を比較する．

4.2 CPU 実行時間と応答時間の評価

CPU コア間システムコールの CPU 実行時間は，システムコールが呼び出されてから値が返されるまでを計測する．測定結果を表 4 に示す．

表 4　CPU コア間システムコールの CPU 実行時間

システムコール	CPU実行時間[μ秒]		
	直接修正版	第1アスペクト版	第2アスペクト版
GetAlarm	3.52	3.55	3.55
GetAlarmBase	4.61	4.61	4.67
SetRelAlarm	4.06	4.09	4.12
SetAbsAlarm	4.06	4.09	4.12
CancelAlarm	3.45	3.45	3.48

ノード間システムコールの CPU 実行時間は，ノード間要求時にタスクを待ち状態にするため，送信側と受信側の CPU で CPU 実行時間を計測する．送信側ではシステムコールを呼び出してからタスクを待ち状態に遷移するまでとタスクを実行可能状態に遷移してから，リターンするまでの合計時間を測定する．測定結果を表 5 に示す．

表 5　ノード間システムコールの CPU 実行時間

システムコール	CPU処理時間[μ秒]								
	直接修正版			第1アスペクト版			第2アスペクト版		
	送信CPU	受信CPU	計	送信CPU	受信CPU	計	送信CPU	受信CPU	計
GetAlarm	2.70	2.12	4.82	2.70	2.12	4.82	2.73	2.12	4.85
GetAlarmBase	3.39	2.91	6.30	3.42	2.91	6.33	3.45	2.91	6.36
SetRelAlarm	6.52	4.21	10.73	6.55	4.21	10.76	6.58	4.21	10.79
SetAbsAlarm	6.52	4.33	10.85	6.55	4.33	10.88	6.55	4.33	10.88
CancelAlarm	2.61	2.18	4.79	2.61	2.18	4.79	2.64	2.18	4.82

表 4，表 5 より CPU コア間システムコール，ノード間システムコールともに，直接修正版と第 1 アスペクト版 RTOS，第 2 アスペクト版 RTOS で最大 0.06μ 秒の差が生じた．2 種類のアスペクト版と直接修正版で CPU 実行時間に差が生じたのは，アスペクトを織り込むことによって，直接修正版よりアスペクト版のほうが関数を呼び出す回数と変数に書き換える回数が多くなることが要因と考えられる．

また，第 1 アスペクト版 RTOS に対して第 2 アスペクト版 RTOS は多くのジョインポイントにアスペクトを織り込んでいるため，CPU 実行時間が増加したと考えられる．直接修正版の CPU 実行時間に対して，CPU コア間システムコール SetRelAlarm，SetAbsAlarm のときに最大 1.47%の増大が生じているが，アスペクト化によるオーバヘッドは十分に小さく，実用上は問題ないと考えている．このため，アスペクト指向プログラミングはアラーム管理以外の機能の拡張にも有効であると考えられる．

次に応答時間について考察する．CPU コア間システムコールの場合の応答時間はビジーウエイトを採用しているため，CPU 実行時間と同値である．マルチコア並列･分散 RTOS が適用対象と考えている制御システムの多くは，タスクの周期は最小で 1m 秒程度と推定している．第 1 アスペクト版 RTOS，第 2 アスペクト版 RTOS の CPU 実行時間は最大で 4.67μ秒で，1m 秒に対して十分に小さいため，実用上問題のない性能であると考える．

ノード間システムコールの場合の応答時間は，CPU 実行時間に，ネットワーク通信時間が加わる．送信するデータ量と送信回数がシステムコールごとに異なるため，CAN の通信時間はシステムコールごとに異なる．データ転送速度 500kbps の場合に要する CAN の通信時間を表 6 に示す．システムコール GetAlarmBase のとき，通信時間は 1008μ秒であり，アスペクト版の応答時間は，最大で 1.01m 秒程度である．一般に，自動車制御用アプリケーションの場合，CAN 通信を用いるタスクの 10m 秒から 100m 秒程度である制御周期に対して，応答時間は十分に小さく，実用上問題のない性能であると考える．

表 6　CAN の通信時間

システムコール	CANの通信時間[μ秒]
GetAlarm	480
GetAlarmBase	1008
SetRelAlarm	760
SetAbsAlarm	760
CancelAlarm	416

4.3 記述量の評価

直接修正版のシステムコールごとの追加記述量を表 7 に示す．また，第 1 アスペクト版と第 2 アスペクト版の各アスペクトの記述量を表 8 と表 9 に示す．第 2 CPU コア間要求アスペクトと第 2 ノード間要求アスペクトはシステムコールによらず，共通なので，common の欄にその記述量を記載している．

記述量の合計は直接修正版で 76 行，第 1 アスペクト版と第 2 アスペクト版のアスペクトの記述量はそれぞれ 299 行と 60 行であった． 第 1 アスペクト版は直接修正版における追加記述に加えて，ポイントカットやアドバイスを指定するための記述等が必要であるため，記述量は多くなる．一方，第 2 アスペクト版は，CPU コア間システムコールおよびノード間システムコールの各システムコールで共通の処理を 1 つのアスペクトで記述しているため，第 1 アスペクト版に対して記述量は少なくなる．

表 7　直接修正版の記述量

システムコール	行数[行]
GetAlarm	15
GetAlarmBase	15
SetRelAlarm	16
SetAbsAlarm	16
CancelAlarm	14
計	76

表 8　第 1 アスペクト版の記述量

システムコール	行数[行]			計
	共通アスペクト	コア間要求アスペクト	ノード間要求アスペクト	
GetAlarm	45	6	7	58
GetAlarmBase	39	6	7	52
SetRelAlarm	49	6	7	62
SetAbsAlarm	65	6	7	78
CancelAlarm	36	6	7	49
計	234	30	35	299

表 9　第 2 アスペクト版の記述量

システムコール	行数[行]			計
	共通アスペクト	コア間要求アスペクト	ノード間要求アスペクト	
GetAlarm	9	-	-	9
GetAlarmBase	10	-	-	10
SetRelAlarm	9	-	-	9
SetAbsAlarm	9	-	-	9
CancelAlarm	9	-	-	9
common	-	7	7	14
計	46	7	7	60

5. おわりに

本論文では，アスペクト指向プログラミングを用いて，構成管理の容易な RTOS ファミリを開発することを目的とし，RTOS のソースコードを直接修正することなく，アラーム管理機能の CPU コア間システムコール，ノード間システムコールを実現するアスペクトを提案した．そして，CPU 実行時間を測定して実用上問題のない性能で実装できるこ

とを示した．また，5 つのシステムコールに共通する処理を一つのアスペクトで記述する第 2 アスペクト版 RTOS では，アスペクトの記述量を直接修正版 RTOS の追加記述量を削減できることを示した．

今後の課題はリソース管理用システムコールを CPU コア間システムコールおよびノード間システムコールに対応させるためのアスペクトを提案することである．

謝辞

本研究で使用した ACC および TOPPERS/ATK1 の開発者に感謝する．本研究は JSPS 科研費 JP18K11225，JP21K11815 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.-M. and Irwin, J.: Aspect oriented programming, ECOOP'97 - Object-Oriented Programming (Ak,sit, M. and Matsuoka, S., eds.), Lecture Notes in Computer Science, Vol. 1241, pp. 220-242, 1997.

[2] Beuche, D., Fröhlich, A. A., Reinhard, M., Papajewski, H., Schön, F., Schröder-Preikschat, W., Spinczyk, O. and Spinczyk, U.: On Architecture Transparency in Operating Systems, Proceedings of the 9th Workshop on ACM SIGOPS European Workshop: Beyond the PC: New Challenges for the Operating system, EW 9, pp. 147–152, 2000.

[3] Spinczyk, O. and Lohmann, D.: Using AOP to Develop ArchitectureNeutral Operating System Components, Proceedings of the 11th workshop on ACM SIGOPS European workshop Article, No.34, 2004.

[4] Afonso, F., Silva, C., Montenegro, S. and Tavares, A.: 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.

[5] Park, J. and Hong, S.: Building a Customizable Embedded Operating System with Fine-Grained Joinpoints Using the AOX Programming Environment, Proceedings of the 2009 ACM symposium on Applied Computing, SAC '09, pp.1952–1956, 2009.

[6] Spinczyk, O., Gal, A. and Schröder-Preikschat, W.: AspectC++: An Aspect-oriented Extension to the C++ Programming Language, Proceedings of the Fortieth International Conference on Tools Pacific: Objects for Internet, Mobile and Embedded Applications, CRPIT '02, pp. 53-60, 2002.

[7] Lohmann, D., Scheler, F., Tartler, R., Spinczyk, O. and Schröder-Preikschat, W.: A Quantitative Analysis of Aspects in the eCos Kernel, Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems, pp.191–204, 2006.

[8] Lohmann, D., Hofer, W., Schröder-Preikschat, W. and Spinczyk, O.: Aspect-aware Operating System Development, Proceedings of the Tenth International Conference on Aspectoriented Software Development, AOSD '11, pp. 69-80, 2011.

[9] AUTOSAR: Specification of operating system, version 2.0.1., pp. 101, 2006.

[10] Lohmann, D., Spinczyk, O., Hofer, W. and Schröder-Preikschat, W.: The Aspect-aware Design and Implementation of the CiAO Operating-system Family, Transactions on Aspect-Oriented Software Development IX, Lecture Notes in Computer Science, Vol.7271, pp. 168-215, 2012.

[11] OSEK/VDX: Operating System, Version 2.2.3, 2005.

[12] TOPPERS Project: http://www.toppers.jp/.

[13] 加藤真平，山崎信行 : Linux カーネル用リアルタイムスケジューリングモジュール, 情報処理学会論文誌コンピューティングシステム（ACS）, Vol. 2, No. 1, pp. 75‑86, 2009.

[14] Abe, K., Yoo, M. and Yokoyama, T.: Aspect-Oriented Customization of the Scheduling Algorithm and the Resource Access Protocol of a Real-Time Operating System, Proceedings of the 2013 IEEE 16th International Conference on, Computational Science and Engineering (CSE), pp. 627–634, 2013.

[15] Harada, Y., Abe, K., Yoo, M. and Yokoyama, T.: Aspect-Oriented Customization of the Scheduling Algorithms and the Resource Access Protocols of a Real-Time Operating System Family, Proceedings of the 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity), pp. 87–94, 2015.

[16] 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.

[17] Saito, N., Yoo, M. and Yokoyama, T.: A distributed real-time operating system built with aspect-oriented programming for distributed embedded control systems, Proceedings of the 2014 20th IEEE International Conference on Parallel and Distributed Systems, pp. 436–443, 2014.

[18] Harada, Y., Yoo, M., Yokoyama, T.: A Distributed Multicore Real-Time Operating System Family Based on Aspect Oriented Programming, Proceedings of the 19th IEEE International Conference on Industrial Technology, pp.1389-1394, 2018.

[19] Gong, M. Zhang, C. and Jacobsen, H. A.: Systems Development with AspeCt-oriented C (ACC), Connections 2007 (ECE Graduate Symposium, University of Toronto), Talk 5.6, 2007.

[20] AsceCt-oriented C: https://sites.google.com/a/gapp.msrg.utoronto.ca/aspectc/.