情報処理学会論文誌 Vol.53 No.12 2702–2714 (Dec. 2012)

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

知場 貴洋[1,a)] 齊藤 政典[1,b)] 伊丹 悠一[1,†1,c)] 兪 明連[1,d)] 横山 孝典[1,e)]

受付日 2012年3月5日, 採録日 2012年9月10日

概要：本論文では，異なるノード上のタスク管理やタスク間の同期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リアルタイム OS について述べる．本論文の対象は，自動車制御などの分野で用いられている分散型の組み込み制御システムである．まず，全ノードで統一した時刻に基づく動作を可能とするため，ネットワークに TDMA プロトコルに基づくリアルタイムネットワークである FlexRay を用いてノード間で OS が管理する時刻の同期を行う手法を提案する．また，ノードの違いを意識しないアプリケーション開発を可能とするため，異なるノード間でタスクの起動や同期が可能な位置透過性のあるシステムコールを提案する．そして，自動車制御向けのリアルタイム OS である OSEK OS を拡張し，提案した機能を有する分散リアルタイム OS を実装する．OSEK OS のタスク管理やイベント管理に関わるシステムコールに位置透過性を持たせ，異なるノード上のタスクを対象にシステムコールを発行可能にする．通信量を考慮して FlexRay 通信のパラメータを設計した場合，位置透過性のあるシステムコールのネットワーク通信時間を含む最悪応答時間は予測可能である．

キーワード：リアルタイム OS，分散システム，組み込みソフトウェア，組み込み制御システム

A Distributed Real-Time Operating System with Location-Transparent System Calls for Embedded Control Systems

Takahiro Chiba[1,a)] Masanori Saito[1,b)] Yuichi Itami[1,†1,c)]
Myungryun Yoo[1,d)] Takanori Yokoyama[1,e)]

Received: March 5, 2012, Accepted: September 10, 2012

Abstract: The paper presents a distributed real-time operating system that provides location-transparent system calls for task management and inter-task synchronization. The target application of the operating system is hard real-time embedded systems such as automotive control systems. The operating system manages distributed tasks based on the global time supported by the clock synchronization of FlexRay, which is a real-time network based on a TDMA (Time Division Multiple Access) protocol. By using the operating system, we can develop a distributed control application program with location-transparent system calls. The distributed real-time operating system is an extension to OSEK OS, which is a standard operating system in the automotive control domain. The worst case response time of a remote system call of the operating system is predictable if the FlexRay communication is well configured.

Keywords: real-time operating systems, distributed systems, embedded software, embedded control systems

[1] 東京都市大学
Tokyo City University, Setagaya, Tokyo 158–8557, Japan
[†1] 現在，日立情報通信エンジニアリング株式会社
Presently with Hitachi Information & Communication Engineering, Ltd.
[a)] g1181525@tcu.ac.jp
[b)] g1281520@tcu.ac.jp
[c)] yuichi.itami.tm@hitachi.com
[d)] yoo@cs.tcu.ac.jp
[e)] yokoyama@cs.tcu.ac.jp

1. はじめに

自動車制御やファクトリーオートメーションなどの分野では，複数の組み込みコンピュータをネットワークでつないだ分散型の組み込み制御システムが用いられている．たとえば自動車制御分野では，複数の ECU（Electronic Control Unit）をネットワーク接続した分散制御システム

構成が用いられている．組み込み制御システムの多くはハードリアルタイムシステムであり，分散型の組み込み制御システムではネットワーク通信も含めたシステム全体で制御処理のデッドラインを守る必要がある．

また，組み込み制御システムのアプリケーションの多くはタスク単位で実装され，組み合わされる．マルチタスクによるアプリケーションでは，あるタスクが他のタスクを起動したりタスク間で同期をとったりする必要があり，1 プロセッサ上で動作するアプリケーションの場合は，リアルタイムオペレーティングシステム（Real-Time Operating System，RTOS）の機能を用いてタスクの管理を行うのが一般的である．組み込みシステムで用いられる RTOS には，μITRON [1] や OSEK OS [2] のようなリアルタイムカーネル型と呼ばれるものと，組み込み Linux のような汎用 OS 型と呼ばれるものがある [3]．リソースの制約が厳しい自動車制御のような組み込み制御システムでは，リアルタイムカーネル型の RTOS が用いられている．

分散型の組み込み制御システムの場合はネットワークを通じて他のノード上にあるタスクと同期をとる必要がある．たとえば，自動車のパワートレイン系の制御システムでは，一定周期で処理を行うタスク（周期タスク）と，エンジン回転に同期して処理を行うタスク（回転同期タスク）が存在する．従来の分散制御システムでは，ECU ごとに独立の RTOS で管理されていたため，同一周期のタスクとして設計しても，ECU 間で周期に微妙なずれが生じる．このため，複数のタスクがそれぞれの制御対象のアクチュエータを同時に制御するなど，複数の周期タスクを同期させて処理を行う必要がある場合には，それらのタスクを単一 ECU 上に配置する必要があった．一方，回転同期タスクは，クランク角センサの割込み処理からタスクを起動することで実装される．回転同期タスクは 1 回転に 1 回とは限らず．複数回転に 1 回実行するようなタスクもあり，それらは RTOS のタスク起動や同期機構を利用して実装される．以上のような，同期実行する必要のある周期タスクや回転同期タスクを分散配置可能とするには，複数のノード上にあるタスクを統一的に管理できる分散 RTOS が必要になる．

分散処理環境向けの基盤ソフトウェアとしては RPC（Remote Procedure Call）ベースのソフトウェアが広く用いられている．RPC ベースの基盤ソフトウェアとして，CORBA [4] や CORBA をリアルタイムシステム向けに変更した Real-Time CORBA [5] や Minimum CORBA [6] がある．また，RPC 機能を提供する分散 OS も提案されている [7], [8]．しかし，RPC による分散処理は必ずしも分散型組み込み制御システムのアプリケーションには適しているとはいえない．たとえば，時間駆動に基づくシステムや，時間駆動処理とイベント駆動処理が混在するようなシステムでは RPC は使用されない [9], [10]．

これまでに多くの分散 OS が提案されており，リアルタイムシステム向けの分散 OS の研究も行われている [11]．複数のコンピュータ資源を位置透過に扱う手法もいくつか提案されており，たとえば，メモリ，CPU，割込み処理，周辺デバイスの 4 つのコンピュータ資源を抽象化し，分散システム全体で統一的な資源管理を行うことが可能な分散 OS も提案されている [12]．しかし，これまでの研究の多くは主に汎用コンピュータを用いた分散システムを対象としており，リアルタイムシステムを対象とした場合でも汎用 OS 型のものがほとんどである．リソース制約の厳しい組み込み制御システム向けには，リアルタイムカーネル型の分散 OS が望まれるが，そのような研究はほとんどなされていない．

我々は，分散処理環境においてノードとタスクの位置関係を意識せずにアプリケーションを記述可能な，組み込み制御システム向けの分散 RTOS に関する研究を行ってきた [13]．我々が提案する分散 RTOS は，自動車制御分野の標準的な RTOS である OSEK OS をベースとして，異なるノード上にあるタスクを対象にできる位置透過性のあるシステムコールを提供する．システムコールが発行された際に対象のタスクがどのノード上にあるか確認する機能を備え，対象のタスクが他ノード上にある場合はネットワークを通じて処理要求を送信する．また，分散 RTOS を搭載する全ノード間で時刻同期を行う機能も備える．ネットワークには，自動車制御分野で普及しつつある TDMA プロトコルに基づくリアルタイムネットワーク FlexRay [14] を用いる．同期した時刻に基づいて処理を行うことで，分散システム全体で同一のグローバル時刻に基づくスケジューリングアルゴリズムを採用することも可能になる．また，通信量を考慮して FlexRay 通信のパラメータを設計した場合，ネットワーク通信時間を含んだシステムコールの最悪応答時間は予測可能である．

本論文では，提案する分散 RTOS の機能と構成について述べる．そして，提案する分散 RTOS を実装して性能評価を行い，組み込み制御システムに適用可能であることを示す．本論文の構成は以下のとおりである．まず 2 章で提案する分散 RTOS の機能と構成について述べる．3 章では提案した分散 RTOS の実装方法とネットワーク通信を含むシステムコールの応答時間について述べる．そして，4 章で実装した分散 RTOS の性能評価を行い，5 章で関連研究について論じ，6 章で本論文のまとめと今後の課題について述べる．

2. 分散 RTOS の機能と構成

2.1 ノード間の時刻同期

全ノードのアプリケーションを統一された時刻に基づいて動作可能とするため，分散 RTOS は全ノード間で同期されたシステム時刻（グローバル時刻）を提供する．全ノードで同期したシステム時刻は，TDMA（Time Division Multiple Access）方式に基づくネットワーク FlexRay を

用いて実現する．各ノードの FlexRay コントローラは全 FlexRay コントローラ間で同期したネットワーク時刻を持ち，ネットワーク時刻を元に周期的な通信を行う．ネットワーク時刻を分散 RTOS に供給することで，全ノードの分散 RTOS が同期したシステム時刻を持つことが可能となる．これにより，Phase-Modification Protocol [15] のような，分散システム全体で同期した時刻に基づくスケジューリングアルゴリズムを採用することが可能になり，複数ノードにまたがる処理全体のレスポンスタイム（End-to-End Response Time）のデッドライン保証が容易になる．

分散 RTOS に供給するネットワーク時刻には，コミュニケーションサイクルを用いる．コミュニケーションサイクルは FlexRay コントローラが持つ最も大きな時間単位であり，このサイクルごとに静的に設定されたスケジューリングに基づいた通信を行う．本研究で提案する分散 RTOS コミュニケーションサイクル開始時に同期して分散 RTOS 内のシステム時刻を更新する．そして，このシステム時刻を OS が周期タスクを管理するために必要なタイムティックとして用いる．

一般に，自動車などの組み込み制御システムで使用される RTOS のタイムティックは 1 msec 程度である．本分散 RTOS のタイムティックを 1 msec とする場合，FlexRay のコミュニケーションサイクルも 1 msec となる．FlexRay におけるクロック同期機構は，コミュニケーションサイクルより細かいマクロティック・レベルでなされる．典型的なマクロティックの時間長は 1 μsec 程度である．このため，FlexRay における時刻同期の精度は 1 μsec 以下となる．計算機ネットワークにおいて広く利用されている NTP（Network Time Protocol）による時刻同期では精度が不十分な場合への対応として，近年 GPS（Global Positioning System）を用いた時刻同期が注目されており，同期の精度を 1 μsec 以下にできるといわれている [16]．すなわち，FlexRay は GPS と同程度の精度を得られることになる．本論文が対象としている自動車のような制御システムにおけるタスクの周期は短いものでも msec オーダであり，時刻同期としては十分な精度を得ることができる．

2.2 遠隔システムコール

本研究では OSEK OS のシステムコールのうち，他ノードのタスクを対象とする可能性があるシステムコールに位置透過性を持たせる．他ノードに対して発行するシステムコールを遠隔システムコールと呼ぶ．開発者は，対象のタスクがどのノード上にある場合も同じシステムコールを用いてアプリケーションを記述できる．遠隔システムコールが発行されたと分散 RTOS が判断した場合，他ノード上のタスクに対する処理のために必要な通信処理を行う．

OSEK OS 仕様では，OS が持つ機能の部分的な実装，モジュール化を可能にするためにコンフォーマンスクラスが定義されている．コンフォーマンスクラスは 4 種類定義されており，最も基本的な機能集合である BCC1 のほかに BCC2 や ECC1，OSEK OS 仕様のすべての機能を網羅した ECC2 がある．本研究では ECC2 を対象に，遠隔システムコール向けの拡張を行う．

OSEK OS が備えるシステムコールには，タスクの管理，タスク間の同期に用いるイベントの管理，主に周期タスクの制御に用いるアラームの管理などがある．このうち，タスクを対象とするシステムコールはタスク管理とイベント管理に関するものだけである．そのため，本研究ではタスク管理とイベント管理に関わるシステムコールを遠隔システムコールの対象とする．

タスク管理とイベント管理に関わるシステムコールの一覧を**表 1** に示す．タスク指定という欄は，システムコールの引数として対象タスクを指定する必要があるかどうかを示している．パラメータ取得という欄は，パラメータを取得するためのシステムコールかどうかを表している．

システムコール名の欄には引数も記しているが，引数の Task は対象タスクの ID であり，Event は設定するイベントマスクを表す．TaskRef はタスク ID を格納する変数へのポインタ，StateRef はタスク状態を格納する変数へのポインタ，EventRef はイベントマスクを格納する変数へのポインタである．システムコール処理が正常に終了すると，ポインタが示す変数に取得したパラメータが格納される．タスク管理とイベント管理に関わるシステムコールのうち，引数でタスクを指定する必要があるシステムコールは，ActivateTask()，ChainTask()，GetTaskState()，SetEvent()，GetEvent() の 5 つである．よって，これら 5 つのシステムコールを拡張し，遠隔システムコールに対応可能とする．

また，OSEK OS では，要求した処理を正しく完了できたかどうかを示す返戻値をシステムコールを発行したタスクに返す．OSEK OS 仕様で定義されているシステムコールの返戻値は 0 から 8 までの整数値である．システムコールの処理が正常に完了した場合は E_OK を返し，エラーが起きた場合はエラー発生を表す値を返す．どのエラーを返すかはシステムコールによって異なる．

本研究で提案する遠隔システムコールの基本的な仕様は，遠隔システムコールを発行したタスクは返戻値を取得するまで待ち状態に遷移することとした．しかし，遠隔システムコールを発行するたびにタスクが待ち状態に遷移することで，アプリケーションが要求する応答性能を実現することが困難になる可能性がある．また OSEK OS の場合，返戻値の主な利用方法はアプリケーションのデバッグである [17]．OSEK OS には，Extended Status Mode と Standard Status Mode があり，システムコールで使用できる返戻値が異なる．前者はシステム開発時に用いるモードで，デバッグに有効ないくつかのエラーコードを使用できる．後者は開発完了後に用いるモードで，一部のシ

表 1 タスク管理とイベント管理に関わるシステムコール
Table 1 System calls for task management and event control.

分類	システムコール名	タスク指定	パラメータ取得
タスク管理	ActivateTask(Task)	○	×
	TerminateTask()	×	×
	ChainTask(Task)	○	×
	Schedule()	×	×
	GetTaskID(TaskRef)	×	○
	GetTaskState(Task, StateRef)	○	○
イベント管理	SetEvent(Task)	○	×
	ClearEvent(Event)	×	×
	GetEvent(Task, EventRef)	○	○
	WaitEvent(Event)	×	×

ステムコールを除き，返戻値として E_OK しか返さない．表 1 に示したシステムコールの中では，ActivateTask() と ChainTask() の 2 つのみが E_OK 以外に E_OS_LIMIT（制限回数以上の起動要求を行った）を返す可能性がある．また，OSEK OS にはデバッグに利用可能なフックルーチンと呼ばれる仕組みがある．システムコールのエラーをフックルーチンで検出可能なため，返戻値を用いずに，フックルーチンでエラー処理を行うことも可能である．エラー E_OS_LIMIT に対しては，起動したタスクが存在するノードでエラー処理を行うのが普通であるため，エラーフックルーチンによる対応で問題ないと考える．

そこで本研究の分散 RTOS では，遠隔システムコールを発行したタスクが待ち状態に遷移しないモードも提供する．これを No-Wait モードと呼ぶ．No-Wait モードでは，遠隔システムコールの処理要求を対象ノードに送信した後，待ち状態に遷移せずに即座に返戻値として E_OK を返す．ただし，表 1 に示したシステムコールのうち，GetTaskState() と GetEvent() はパラメータを取得してアプリケーションに返す必要があるため，No-Wait モードの対象外とし，つねに対象ノードからの返答を待つ．前述のように，ActivateTask() と ChainTask() のエラー E_OS_LIMIT のみについてエラーフックルーチンで対応すれば，No-Wait モードでも実用上大きな問題はないため，システム開発完了後は，Standard Status Mode を用いるとともに No-Wait モードを用いるのが基本であると考えている．

No-Wait モードを用いるかどうかは，OSEK OS における Extended Status Mode と Standard Status Mode の選択と同様に，システム構築時に静的に設定する．また，1 つの分散システム内では，同一モードの分散 RTOS を使用するものとする．

2.3 遠隔システムコールの動作

通常の遠隔システムコールの動作を**図 1** に示す．図 1 は，ノード CallerNode 上のタスク ApplicationTask が，ノード CalleeNode 上のタスクに対するシステムコールを発行してから，返戻値を受け取って処理を再開するまでのタイムチャートを示している．

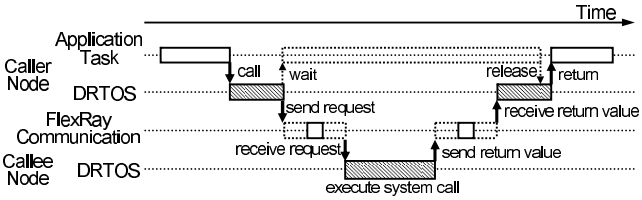

図 1 遠隔システムコール
Fig. 1 Remote system call.

CallerNode の ApplicationTask がタスクを対象にシステムコールを発行すると，分散 RTOS は対象タスクがどのノード上にあるか確認を行う．対象タスクは CalleeNode にあるため，分散 RTOS は CalleeNode にシステムコールの処理要求を送信する．そして，ApplicationTask を待ち状態に遷移させる．処理要求を受け取った CalleeNode の分散 RTOS は，要求された処理を行って返戻値を CallerNode に送信する．返戻値を受け取った CallerNode の分散 RTOS は，ApplicationTask の待ち状態を解除する．待ち状態を解除された ApplicationTask は，返戻値を受け取って処理を再開する．

なお，遠隔システムコール発行後，何らかの異常により返戻値が戻らず，待ち状態が解消されない可能性がある．周期タスクや回転同期タスクは繰り返しタスクが起動されるため，次回タスクが起動されたときの多重起動エラーより異常を検出し，エラーフックルーチンによりこのような状況に対応することとする．

次に，No-Wait モードにおける遠隔システムコールのタイムチャートを**図 2** に示す．No-Wait モードの場合，分散 RTOS は CalleeNode にシステムコールの処理要求を送信すると即座に ApplicationTask に返戻値 E_OK を返す．ApplicationTask は返戻値を受け取り，処理を継続する．

2.4 拡張 OIL

OSEK OS では，アプリケーションの設定（コンフィ

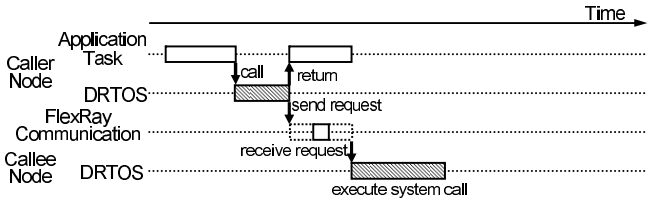

図 2 遠隔システムコール No-Wait モード
Fig. 2 Remote system call in No-Wait mode.

```
<file> ::=
    <OIL_version>
    <implementation_definiticn>
    <application_definition_list>
. . . . . .
<application_definition_list> ::=
    <application_definition>
    | <application_definition_list> <application_definition>
<applicarion_definition> ::=
    "CPU" <name> "{" <object_definition_list> "}" <description> ";"
. . . . . .
```

図 3 拡張 OIL の構文
Fig. 3 Syntax of extended OIL.

```
OIL_VERSION = "2.5";
IMPLEMENTATION Standard {
. . . . .
}
CPU cpu0 {
    TASK task00 { . . . . . }
    TASK task01 { . . . . . }
}
CPU cpu1 {
    TASK task10 { . . . . . }
    TASK task11 { . . . . . }
}
```

図 4 拡張 OIL の記述例
Fig. 4 Example extended OIL description.

ギュレーション）のために OIL（OSEK Implementation Language）[18] と呼ばれる専用言語を用いる．OIL によりタスクの宣言やイベントの設定などを記述し，SG（System Generator）に入力することでアプリケーション依存の構成情報を記述したソースコードを出力する．本研究では，分散 RTOS 向けに 1 つのファイルに全ノードのアプリケーションの設定を記述できるように OIL を拡張する．また，拡張 OIL に対応した SG を開発する．

図 3 は，拡張 OIL の構文を BNF 記法で表記したものである．<application definition>が 1 ノード分のアプリケーションの設定情報を表す．従来の OIL では 1 つのファイルに<application definition>は 1 つしか含まれなかったが，拡張 OIL では<application definition list>を追加し，1 つのファイルに複数の<application definition>を含むことができるように拡張した．

次に，拡張 OIL の記述例を**図 4** に示す．図 4 は，cpu0 と cpu1 の 2 ノード分の設定情報を記述した拡張 OIL の例である．cpu0 のアプリケーションは task00 と task01 で構成され，cpu1 のアプリケーションは task10 と task11 で構成されている．

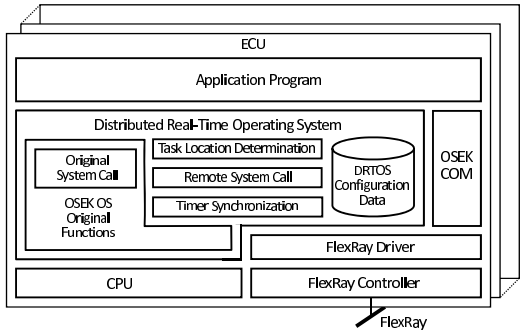

図 5 分散 RTOS の構成
Fig. 5 Structure of distributed real-time operating system.

2.5 システム構成

本研究で提案する分散 RTOS は，OSEK OS に分散処理のための機能を追加することで実現する．提案する分散 RTOS の構成を**図 5** に示す．分散 RTOS は，タスク位置確認機能（Task Location Determination），遠隔システムコール機能（Remote System Call），システム時刻の同期機能（Timer Synchronization）を備える．SG により生成した情報は，分散 RTOS 構成データ（DRTOS Configuration Data）として保持される．

タスク位置確認機能は，システムコールの対象タスクがどのノード上にあるか確認を行う．対象のタスクが自ノードにある場合，分散 RTOS は OSEK OS のシステムコールを実行する．対象のタスクが他ノードにある場合，分散 RTOS は遠隔システムコール機能を呼び出す．遠隔システムコール機能は対象タスクがあるノードに処理要求を送信し，受け取った返戻値やパラメータを遠隔システムコール要求元タスクに返す．また，他ノードから遠隔システムコールの処理要求を受け取った場合は，要求された OSEK OS のシステムコールを実行して返戻値やパラメータを遠隔システムコール要求元ノードに送信する．システム時刻の同期機能は，全ノード共通のグローバル時刻を提供する．

なお，分散 RTOS の機能を用いずにアプリケーション間で直接メッセージ通信を行うには，OSEK COM [19] を用いる．

3. 実装

3.1 概要

OSEK OS 仕様の RTOS である TOPPERS/OSEK カーネル [20] をベースに分散処理機能を追加することで，提案した分散 RTOS を実装する．

分散 RTOS のシステムコール発行時に，分散システム全体でタスクを一意に指定可能とするため，全ノードで共通のグローバルタスク ID を定義する．グローバルタスク ID は 2 Byte の値であり，文献 [21] と同様に所属するノードを表すノード ID と，ノード内で識別するための ID で構成される．グローバルタスク ID の上位 1 Byte がノード

ID であり，下位 1 Byte がノード内における識別 ID である．本研究で主な対象とする自動車のパワートレイン系やシャーシ系のシステムでは，ノード数は多くても 10 ノード程度である．また，バス型のトポロジを採用した場合の FlexRay の最大ノード数は 22 である．一方，オーバヘッドを低減するため，自動車制御では同一周期や同一起動要因の処理は 1 つのタスクとすることが多いため，1 ノードあたりのタスク数は一般に 20 以下のことが多い．2 Byte のグローバルタスク ID であれば，システム全体で 256 ノード，1 ノードあたり 256 タスクまで対応できるため，今後ノード数やタスク数が増大する可能性を考慮しても十分であると考えている．

ネットワークとして用いる FlexRay のコミュニケーションサイクルは，周期的なデータ送信に適したスタティックセグメントと非周期的なイベントによるデータ送信に適したダイナミックセグメントに分けられる．タスクがシステムコールを発行するタイミングは周期的とは限らないため，本分散 RTOS はダイナミックセグメントを用いて遠隔システムコールのデータを送信する．

3.2 システム時刻

本研究ではシステム時刻をアップカウンタとして実装し，FlexRay のコミュニケーションサイクル開始時に同期した割込みを用いて更新する．システム時刻の同期処理は FlexRay 通信が確立されると開始される．FlexRay 通信を確立するためには，最低でも 2 ノードが必要である．

FlexRay 通信が確立されると，マスタとなるノードが決定され，マスタノードはシステム時刻同期用の値を送信する．FlexRay 通信確立後に参入したノードも，マスタノードから送られてくるシステム時刻に合わせる．同期用の値は毎サイクル送信する必要があるため，ダイナミックセグメントではなくスタティックセグメントを用いて送信する．また，システム時刻の同期処理はコミュニケーションサイクル開始のタイミングで行われるため，同期処理の際は 1 サイクル前にマスタノードから送信された値を用いる．よって，マスタノードが送信する値は現在のシステム時刻に 1 を加えた値である．他ノードは受け取った値にシステム時刻を合わせ，同期完了フラグを立てる．同期完了フラグが立ったノードは，コミュニケーションサイクル開始時に同期してシステム時刻の更新を行う．

3.3 タスクの位置確認

分散 RTOS は，システムコールの引数として受け取るグローバルタスク ID の上位 1 Byte と，各ノードに割り当てられたノード ID を照合してタスクの位置確認を行う．タスクの位置確認の際，分散 RTOS を搭載するノード数を表す定数と，各ノードのタスク数を保持する配列を参照し，存在しないノードやタスクを指定していた場合はエラーを返す．各ノードに割り当てるノード ID やエラーチェック用のデータは，3.6 節で述べるコンフィギュレーションによって決定される．タスクの位置確認の結果，対象のタスクが自ノードにある場合は TOPPERS/OSEK カーネルの処理を実行する．対象のタスクが他ノードにある場合は，本研究で追加した遠隔システムコールの処理を実行する．

3.4 遠隔システムコールの実装

本分散 RTOS における遠隔システムコール処理の流れを**図 6** に示す．ノード CallerNode 上のタスクがシステムコールを発行すると，分散 RTOS は対象タスクのグローバルタスク ID 中のノード ID を参照して，対象タスクがどのノード上のタスクか確認を行う．対象タスクが他ノード上にある場合，分散 RTOS は遠隔システムコールの処理要求メッセージを生成する．処理要求メッセージは，処理要求タスク ID，処理対象タスク ID，システムコールの種類，システムコールの引数，処理要求メッセージを表すメッセージ ID で構成する．そして，FlexRay ドライバにメッセージの送信要求を出す．その後，分散 RTOS は遠隔システムコール発行元タスクを待ち状態に遷移させる．

送信要求の呼び出しを受けた FlexRay ドライバは，送信するメッセージ本体にヘッダ情報などを付加したフレームを生成する．フレームのサイズは 20 Byte である．そして，FlexRay コントローラのメッセージ RAM にフレームを書き込む．実際の通信は FlexRay コントローラが行う．通信が完了すると，受信側ノード CalleeNode の FlexRay コントローラのメッセージ RAM に処理要求メッセージが格納される．

FlexRay コントローラのメッセージ RAM に格納されたメッセージのチェックは，サイクル開始処理で行う．サイクル開始処理は，FlexRay のコミュニケーションサイクルに同期した割込み処理によって開始される．割込み処理には，OSEK OS 仕様で定義されている ISR カテゴリ 2 の割

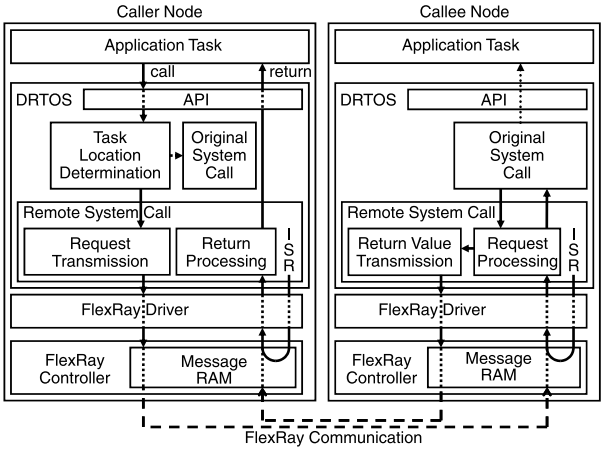

図 6 遠隔システムコール処理の流れ
Fig. 6 Remote system call processing flow.

込みを用いる．

CalleeNode の分散 RTOS はサイクル開始処理で FlexRay コントローラのメッセージ RAM をチェックし，受信した処理要求メッセージを読み出して，メッセージの内容を解読する．そして要求されたシステムコールの処理を行う．

システムコールの処理が終了したら，No-Wait モードでない場合や No-Wait モードの対象外のシステムコールの場合は，処理を行って得られた返戻値を含む返信メッセージを生成する．返信メッセージは，処理要求タスク ID，返信データ，返信メッセージを表すメッセージ ID で構成する．要求された処理が ActivateTask()，ChainTask()，SetEvent() のいずれかの場合，返信データはシステムコールの返戻値である．GetTaskState() の場合，返信データはタスク状態を表す値である．GetEvent() の場合，エラーが発生した場合の返信データは 0xFFFFFFFF であり，正常に処理が完了した場合の返信データは取得したイベントマスクである．そして，FlexRay ドライバを呼び出し，返信メッセージにヘッダ情報などを付加して 20 Byte のフレームを生成し，FlexRay のメッセージ RAM に書き込む．

メッセージ RAM のチェックからフレームの書き込みまでが割込みで処理される．通信が完了すると，CallerNode の FlexRay コントローラのメッセージ RAM に返信メッセージが格納される．

CallerNode の分散 RTOS は，CalleeNode と同様に割込みで開始されたサイクル開始処理で返信メッセージを読み出す．そして，受け取ったデータを返信データ用バッファに格納する．返信データ用バッファのサイズは 1 つのタスクにつき 32 bit である．そして，返信メッセージの格納が完了すると，遠隔システムコール発行元タスクの待ち状態を解除する．

待ち状態を解除されたタスクは，返信データ用バッファのデータを読み出して処理を再開する．ActivateTask()，ChainTask()，SetEvent() の場合はバッファのデータをシステムコールの返戻値として返す．GetTaskState() の場合，バッファのデータをタスク状態を格納する変数に渡し，システムコールの返戻値として E_OK を返す．GetEvent() の場合，まず返信データの内容が 0xFFFFFFFF と等しいか確認する．等しかった場合はエラーが発生したと判断し，システムコールの返戻値として E_OS_STATE を返す．異なる場合は正常にイベントマスクが取得できたと判断し，イベントマスクを格納する変数にバッファのデータを渡す．そして，システムコールの返戻値として E_OK を返す．

図 7 は，遠隔システムコールのタイムチャートを表している．遠隔システムコールの処理は，サイクル開始処理（cycle start processing），要求送信処理（request transmission），返戻値送信処理（return value transmission），リターン処理（return processing）の 4 つに分けられる．サイクル開始処理は FlexRay のコミュニケーションサイクルの開始時に同期して行われ，システム時刻の更新と，他ノードからの遠隔システムコールの処理要求があるかどうかをチェックする．もし処理要求があれば，要求された処理を行う．要求送信処理で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確認し，他ノード上にあれば処理要求を対象のノードに送信する．そして，他ノードのタスクに処理要求を出したタスクを待ち状態に遷移させる．返戻値送信処理では，要求された遠隔システムコールの処理によって得られた返戻値を遠隔システムコール要求元のノードに送信する．リターン処理では，受け取った返戻値をバッファに格納し，遠隔システムコール発行元タスクの待ち状態を解除する．

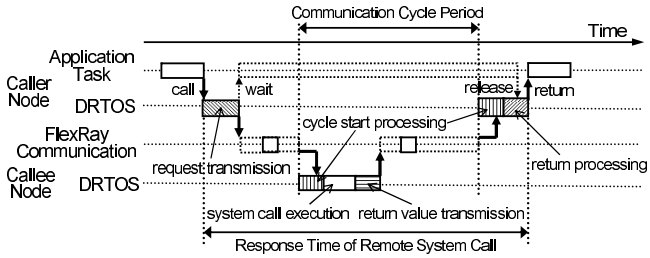

図 7 遠隔システムコールのタイムチャート
Fig. 7 Time chart of remote system call.

3.5 遠隔システムコールの最悪応答時間

遠隔システムコールの応答時間には，分散 RTOS の処理時間のほかに FlexRay コントローラのメッセージ RAM に送信フレームを書き込んでから送信完了までの遅延時間が含まれる．

FlexRay のコミュニケーションサイクルは複数のタイムスロットで構成されており，コミュニケーションサイクルを構成するタイムスロットの数や各スロットの ID は全サイクルで共通である．FlexRay ドライバが生成する送信フレームのヘッダ情報の 1 つにフレーム ID があり，FlexRay コントローラはフレーム ID とスロット ID が一致する場合にそのフレームを送信する．メッセージ RAM に書き込まれた送信フレームは，ID が一致するスロットの時刻まで保持される．また FlexRay では，通信速度やコミュニケーションサイクルの設定によって 1 サイクル中に送信可能な総データサイズが決まる．同じサイクル中に送信を要求したフレームの合計サイズが送信可能な総データサイズを超えた場合，送信できなかったフレームは次サイクルに持ち越される．

3.6 節で述べるように，FlexRay 通信のパラメータは通信量を考慮して設計される．すなわち，同一サイクル中に 1 サイクルで送信可能な総データサイズを超える通信を要求しないように設計される．この場合，あるサイクルで対応するタイムスロットが終了する前に送信フレームの書き込みが完了すれば，同じサイクルでフレームを送信できる．

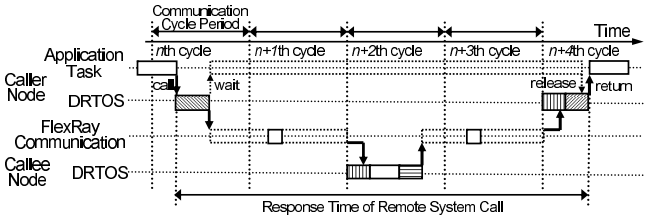

図 8 遠隔システムコールの最悪応答時間
Fig. 8 The worst case time chart of remote system call.

対応するタイムスロットが終了した後に送信フレームの書き込みが完了した場合は，次サイクルのタイムスロットでフレームを送信できる．よって，FlexRay 通信の最悪遅延時間は 2 サイクルである．

通信量を考慮して FlexRay 通信のパラメータを設計すれば，本分散 RTOS の遠隔システムコールの最悪応答時間は予測可能である．遠隔システムコールの最悪応答時間を表すタイムチャートを**図 8** に示す．この図は，CallerNode の分散 RTOS が要求送信処理で生成した処理要求メッセージを，生成したサイクル中に送信できない場合を示している．この場合は，CallerNode のタスクが遠隔システムコールを発行してから処理要求メッセージの送信完了までは最大で 2 サイクルかかる．またこの図は，CalleeNode が返戻値送信処理で生成した返信メッセージも，生成したサイクル中に送信できていない場合を示している．この場合は，CalleeNode がサイクル開始処理を開始してから返信メッセージの送信完了までは最大 2 サイクルかかる．よって，タスクが遠隔システムコールを発行してから返戻値を得るまでの最悪応答時間は 4 サイクルにサイクル開始処理とリターン処理の処理時間を加えた値となる．

一方，遠隔システムコールを発行してから，対象ノード上でシステムコールの実行を開始するまでの最悪遅延時間は，2 サイクルにサイクル開始処理時間を加えた値となる．異なるノード間でタスクの起動や同期を行う場合は，この遅延時間を考慮する必要がある．たとえば，自動車のパワートレイン系の制御システムの回転同期タスクを ActivateTask()，ChainTask()，SetEvent() などの遠隔システムコールを用いて実行する場合，上記遅延時間が許容される範囲で使用する必要がある．一般に，自動車のエンジン回転数は高回転時には 6,000 rpm 程度になるが，このとき 1 回転にかかる時間は 10 msec である．したがって，コミュニケーションサイクル 1 msec の場合の上記遅延時間は 2 msec 強となり，エンジンのクランクシャフトの角度に換算すると 70〜90 度程度になる．この遅延時間は，タスクの実行回数をエンジン回転数に一致させることが目的の回転同期タスクにとっては許容範囲と考える．一方，点火や燃料噴射のように，クランクシャフトの角度まで厳密に同期をとる必要がある角度同期処理については，本分散 RTOS を用いても分散化は困難である．しかしこのような角度同期処理は，現在のように単一 ECU で実装されている場合でも，タスクではなくデバイスドライバの割込み処理で実現するのが普通である．システムコールを用いてタスクの起動や同期を行う対象となるのは回転同期タスクであり，本分散 RTOS の狙いである位置透過なタスク管理という点では，十分有用性があると考える．

3.6 分散 RTOS のコンフィギュレーション

本分散 RTOS を用いてシステムを構築する際には，2 つのコンフィギュレーションが必要となる．1 つは分散 RTOS 自体の構成情報を決定することで，もう 1 つは FlexRay のパラメータを決定することである．

分散 RTOS 自体の構成情報には，TOPPERS/OSEK カーネルがもともと必要としていた構成情報に加え，本分散 RTOS で追加した分散処理機能向けの構成情報がある．我々は，2.4 節で述べた拡張 OIL の記述からそれらの 2 種類の構成情報を生成できる SG を開発した．

TOPPERS/OSEK カーネル本来の構成情報は，文献 [20] で提供されている TOPPERS/OSEK カーネル用 SG に従来の OIL 記述を入力することで出力できる．分散処理機能向けの構成情報にはノード ID，グローバルタスク ID，返信データ用バッファのサイズ，タスク位置確認で用いるエラーチェック用のデータが含まれ，本研究で開発した SG を用いて生成する．ノード ID は，符号なし 8 bit 整数を 0 から順に割り当てる．グローバルタスク ID は，TOPPERS/OSEK カーネル用 SG で出力された 1 Byte のタスク ID とノード ID を結合することで生成する．返信データ用バッファは符号なし 32 bit 整数の配列であり，配列の要素 1 つにつき遠隔システムコール 1 つ分の返信データが格納される．遠隔システムコールを発行したタスクは返信データを取得するまで処理が中断されるため，1 つのタスクにつき遠隔システムコール 1 回分の返信データを保持できればよい．そのため，配列の要素数はノード内のタスク数と同じとなる．エラーチェック用のデータは，拡張 OIL の記述からノード数とタスク数を取得して決定する．分散 RTOS を搭載するノード数を表す定数は#define で定義される．各ノードのタスク数を保持する配列は符号なし 8 bit 整数の配列であり，要素数はシステム内の分散 RTOS を搭載するノード数と同じである．

本分散 RTOS 向けの SG は，入力として拡張 OIL ファイルを受け取ると，ノードごとの記述に分割して TOPPERS/OSEK カーネル用 SG に入力し，全ノード分の TOPPERS/OSEK カーネル向けの構成情報を生成する．そして，生成した構成情報をもとに分散処理機能向けの構成情報を決定し，ノードごとの分散処理向け構成情報をまとめたソースファイルを生成する．

一般に FlexRay のパラメータは，システム構築者が通信量を考慮して決定する必要がある．これまでに，最悪応答

時間を考慮したFlexRayのパラメータ最適化手法については，様々な提案がされている [22], [23]．本分散RTOSを用いたシステムでも同様な手法でFlexRayのパラメータを決定できるが，アプリケーション自体が行う通信量のほか，分散RTOSが行う通信量，すなわち，遠隔システムコールの発行頻度も考慮して設計する．遠隔システムコールの通信フレームのサイズはすべて20 Byteに固定されているため，1サイクル中の遠隔システムコールの最大発行数を決めることで分散RTOSが行う通信量が分かる．

自動車のパワートレイン系の制御システムを例に，遠隔システムコールの発行頻度を考慮したFlexRayパラメータ設計の概要を説明する．1章で述べたように，パワートレイン系システムには周期タスクと回転同期タスクがある．周期タスクについては，統一されたシステム時刻で管理されるため，同期のために遠隔システムコールを発行する必要はない．回転同期タスクの場合は，クランク角センサの割込み処理を扱うノードが，他ノードに対しActivateTask(), ChainTask(), SetEvent()などの遠隔システムコールを発行してタスクの起動や同期を行う．回転同期のための遠隔システムコールの頻度は，エンジン回転数に依存する．一般に，自動車のエンジン回転数は通常で2,000 rpm程度，高回転時で6,000 rpm程度である．1回転に1回同期をとる場合，エンジン回転6,000 rpmのとき，システムコールの発行頻度は10 msecに1回となる．パワートレイン系システムの場合は一般にノード数は10ノード以下であるので，10ノード構成のシステムについて考えると，全ノードで同期をとるための遠隔システムコール数はトータルで9となる．したがって，エンジン回転6,000 rpmで，1回転に1回同期をとる場合の遠隔システムコールの発行頻度は10 msecに9回，すなわち，1 msecあたり0.9回となる．しかし最悪の場合，これら9回のシステムコールが1コミュニケーションサイクル内に発生する可能性がある（No-Waitモードでは十分可能な回数である）．1サイクル中に遠隔システムコール9回分の通信を行うには，ダイナミックセグメントで20 Byteのフレームを9回以上送信できるように設定する必要がある．FlexRayの転送レートを10 MHz，コミュニケーションサイクル長を1 msecとした場合，コミュニケーションサイクルの15%をダイナミックセグメントに割り当てれば，1サイクルで遠隔システムコール9回分のフレームを送信できる．なお，この場合，遠隔システムコールによる平均の負荷は1.5%であり，残り13.5%はアプリケーションプログラム間の通信に利用できる．

また，コミュニケーションサイクル長を決定する場合は，遠隔システムコールの最悪応答時間への影響も考慮する必要がある．FlexRayのコミュニケーションサイクル長が長くなるほど遠隔システムコールの最悪応答時間が大きくなる一方，FlexRay通信の発生頻度に対してコミュニケーションサイクル長が短いと，3.5節で述べた送信フレームの次サイクルへの持ち越しが発生しやすくなる．

4. 評価

4.1 評価環境

本研究で提案した分散RTOSを，V850E/PHO3プロセッサとFlexRayコントローラを搭載した評価ボードGT201N10上に実装し，自ノード上のタスクを対象としたシステムコールのオーバヘッドと，遠隔システムコールにおける処理時間の評価を行った．V850E/PHO3は32 bit RISCプロセッサで，クロックは128 MHz，ROM容量992 kByte，RAM容量92 kByteである．FlexRayコントローラのクロックは80 MHzである．そして，2ノード構成上で評価用アプリケーションを動作させて，分散RTOSの性能評価を行った．FlexRayでは各ノードからの通信が衝突するということはないので，今回の測定項目については，ノード数を増大させても値に影響することはない．

FlexRayの転送レートは10 MHz，コミュニケーションサイクル長は2 msecとして測定を行った．前述のように，自動車制御などの組み込み制御システムで用いられるRTOSでは，タイムティックを1 msec程度とすることが多いが，今回は計測処理が1タイムティック以内に十分収まるように，余裕をみてタイムティックすなわちコミュニケーションサイクルを2 msecとした．今回測定した実行時間は，コミュニケーションサイクルの値に依存しないため，問題はない．

実行時間の計測にはハードウェアカウンタを用いた．ハードウェアカウンタのクロックは32 MHzで，1カウントは31.25 nsecである．いずれの計測もハードウェアカウンタを用いて実行時間を50回計測し，50回中の最悪値と平均値を最終的な計測結果として扱う．計測結果は0.01 μsec（10 nsec）単位で示す．

4.2 オーバヘッドの評価

本分散RTOSの自ノード内のタスクを対象にしたシステムコール処理が，TOPPERS/OSEKカーネルの処理と比べてどの程度オーバヘッドがあるかを計測した．この計測に用いた評価用アプリケーションは，高優先度タスクから同じノード内の低優先度タスクを対象にシステムコールを発行する処理を，システム時刻に同期して繰り返し行う．そして．高優先度タスクがシステムコールを発行してから返戻値を取得するまでの時間を計測する．ただし，ChainTask()の場合はシステムコールを発行してからディスパッチャが起動する直前までの時間を計測する．これは，ChainTask()では正常にシステムコール処理が行われるとディスパッチャが起動し，返戻値を返さずに起動タスクが切り替わってしまうためである．

計測結果を**表 2**に示す．かっこ外の値は平均値，かっこ

表 3 遠隔システムコールの実行時間
Table 3 Execution time of remote system call.

システムコール	要求送信	リターン処理	サイクル開始	返戻値送信
ActivateTask()	19.19 (19.19)	47.41 (47.41)	63.31 (63.31)	17.27 (17.28)
ChainTask()	19.21 (19.22)	47.47 (47.47)	63.31 (63.31)	17.25 (17.28)
GetTaskState()	19.23 (19.25)	47.47 (47.47)	63.31 (63.31)	17.28 (17.31)
SetEvent()	19.24 (19.25)	47.47 (47.47)	63.31 (63.31)	17.25 (17.25)
GetEvent()	19.27 (19.28)	47.41 (47.41)	63.31 (63.31)	17.27 (17.28)

かっこ内は最悪実行時間 [μsec]

表 2 自ノード上のタスクを対象としたシステムコールのオーバヘッド
Table 2 Overhead of local system call.

システムコール	分散 RTOS	TOPPERS/OSEK カーネル
ActivateTask()	4.77 (4.78)	4.55 (4.56)
ChainTask()	4.42 (4.43)	4.03 (4.03)
GetTaskState()	2.31 (2.31)	1.91 (1.91)
SetEvent()	3.74 (3.75)	3.50 (3.50)
GetEvent()	2.31 (2.31)	2.03 (2.03)

かっこ内は最悪実行時間 [μsec]

内の値は最悪値である．自ノード上のタスクを対象とした場合は，No-Wait モードであるかどうかによらず，同じ値である．

本評価で計測したオーバヘッドには，グローバルタスク ID から 1 Byte のタスク識別用 ID を取り出してタスク位置確認を行う処理時間が含まれる．どのシステムコールの場合も分散 RTOS の処理のオーバヘッドは TOPPERS/OSEK カーネルの処理時間の 5%未満であり，十分に小さい値である．また，計測結果の平均値と最悪値の差はいずれも，計測に用いたタイマの精度である 0.03125 μsec（31.25 nsec）以下であり，また，従来の TOPPERS/OSEK カーネルと比較しても大きな差はなく，実行時間の予測性の点でも問題ないと考える．

4.3 遠隔システムコールの評価

本分散 RTOS の遠隔システムコールにおける処理の実行時間を計測した．具体的には，図 7 における要求送信処理，返戻値送信処理，サイクル開始処理，リターン処理について計測を行った．この計測に用いた評価用アプリケーションでは，分散 RTOS のシステム時刻 10 サイクルに 1 回の周期で遠隔システムコールを発行する．そして，その際の各処理の実行時間を計測する．

表 4 メモリ消費量
Table 4 Memory consumption.

	コード	定数	データ	合計
分散 RTOS	12,930	143	3,927	17,000
TOPPERS/OSEK カーネル	9,684	120	3,742	13,546

[Byte]

計測結果を**表 3** に示す．かっこ外の値は平均値，かっこ内の値は最悪値である．表に示したのは No-Wait モードを使用しない場合の値である．遠隔システムコール送信ノードの要求送信実行時間およびリターン処理実行時間の合計は約 66 μsec，受信ノードのサイクル開始処理実行時間および返戻値送信実行時間の合計は約 80 μsec である．遠隔システムコールにおける分散 RTOS の処理の実行時間はコミュニケーションサイクルの値に依存しないため，開発者は分散 RTOS の処理時間とアプリケーションに要求される応答時間を考慮してコミュニケーションサイクル長を決定する．たとえば，自動車制御アプリケーションの場合，通信をともなう処理の制御周期は 10 msec から 100 msec 程度である．コミュニケーションサイクル長を 1 msec とすれば遠隔システムコールの最悪応答時間は 4.066 msec になり，制御周期の半分以下の値となる．

なお，No-Wait モードの場合は，遠隔システムコール送信ノードでは要求送信処理が完了すると即座にタスクの処理に戻るため，リターン処理は行われない．受信ノードでは，サイクル開始処理で呼び出したシステムコール処理が完了すると，返戻値送信処理を行わずに次の処理を行う．

4.4 メモリ消費量

本研究で実装した分散 RTOS と従来の TOPPERS/OSEK カーネルの実行ファイルのメモリ消費量を比較した．どちらもタスク数を 2 として OS の構成情報を出力してコンパイルを行った．また，今回示すメモリ消費量には，通信用のデバイスドライバは含まれていない．メモリ消費量を**表 4** に示す．分散 RTOS のメモリ消費量は，TOPPERS/OSEK カーネルの約 1.3 倍程度となった．増

加した要因は，本研究で追加した分散処理機能と，追加した分散処理機能向けの構成情報である．

タスク数が増加した場合は TOPPERS/OSEK カーネル本来の構成情報が増加する．分散 RTOS の場合はそれに加え，符号なし 32 bit 整数として実装した返信データ用バッファの要素数が増える．そのため，分散 RTOS は TOPPERS/OSEK カーネルに比べ，タスク 1 つにつき 32 bit 配列の要素 1 つ分メモリ消費量が大きくなる．また，TOPPERS/OSEK カーネルにはノード数に依存する構成情報は存在しなかったが，分散 RTOS は各ノードのタスク数を保持する符号なし 8 bit 整数の配列を持つ．よって，分散 RTOS は TOPPERS/OSEK カーネルと比べて，ノード 1 つにつき 8 bit 配列の要素 1 つ分メモリ消費量が大きくなる．

5. 関連研究

自動車制御システム向けの基盤ソフトウェアの仕様として OSEK/VDX 仕様があり，RTOS の仕様である OSEK OS [2] や，分散システムに対応するための通信ソフトウェアの仕様である OSEK COM [19] などを定義している．OSEK OS はタスクの起動やイベントによるタスク間同期などの機能を提供し，タスクはシステムコールを発行することで OSEK OS の機能を呼び出すことができる．しかし，OSEK OS は単一ノード内のタスク管理を対象としており，複数のノード上のタスクを統一的に管理することはできない．たとえば，他ノード上のタスクを対象に起動や同期の処理を行うには，OSEK COM を用いてメッセージ通信を行い，対象タスクが存在するノード上のアプリケーションに要求を送り，そのアプリケーションからシステムコールを発行し，そのノード上の OSEK OS に処理を要求する必要がある．これに対し本論文で提案した分散 RTOS では，位置透過性のあるシステムコールを提供することで，どのノード上のタスクに対しても同一の記述で起動や同期が行え，ノードの違いを意識せずにアプリケーションを開発可能になる．また，アプリケーションタスクのノードへの割当てを変更する場合も，ソースコードの書き換えは必要なく，OIL 記述の修正などのコンフィギュレーションの変更で対応できる．

組み込みシステム向けの位置透過性のあるシステムコールを有する RTOS として, TOPPERS/FDMP カーネル [21] がある．この RTOS は，組み込み機器向けに策定された μITRON 仕様 [1] の OS を各プロセッサで実行するタスクを静的に割り当てる機能分散マルチプロセッサシステム向けに拡張したものである．また，自動車分野の業界標準仕様の策定を行っている AUTOSAR が，マルチコア向けの RTOS の仕様を定めている [24]．しかしこれらは，すべてのプロセッサで共有するメモリを持つ密結合型のシステムを対象としたものである．これに対し本論文で提案した分散 RTOS は，複数の組み込みコンピュータをネットワークで接続した疎結合型の分散システムにおいて，位置透過性のあるシステムコールを提供している．

AUTOSAR では，分散処理環境におけるアプリケーションコンポーネント間インタフェースを標準化する VFB（Virtual Function Bus）という仕様を策定している [25]．これにより，分散処理を意識しないアプリケーション開発が可能になる．AUTOSAR の分散処理実行環境は RTE（Run-Time Environment）と呼ばれるミドルウェアと RTOS やデバイスドライバにあたる基本ソフトウェア（Basic Software）からなるソフトウェア階層で実装される．したがって，分散処理環境で厳しい時間制約を守るには，RTE や基本ソフトウェアでリアルタイム性を保証する必要がある．VFB では回転同期のための Triggering Interface と呼ばれるインタフェースも定義しており，ネットワーク接続されたノード間で，最悪遅延時間や最悪応答時間が予測可能な環境を実現することが大きな課題となっている．AUTOSAR のアーキテクチャを採用する場合も，本論文で提案した分散 RTOS 上に RTE を実装すれば，時間制約を守れる分散処理環境の実現が容易になると考える．

6. おわりに

本論文では，分散型の組み込み制御システムを対象に，複数のノード上にあるタスクを統一的に管理できる分散 RTOS について述べた．本分散 RTOS は，自動車制御向けの RTOS である OSEK OS を拡張してシステムコールに位置透過性を持たせ，自ノード上のタスクと同様に他ノード上のタスクを対象にしたシステムコールの発行を可能にした．また，全ノードで統一された時刻に基づいた動作を可能とするため，ネットワークに FlexRay を用いてノード間で OS が管理する時刻の同期を行う機能を実装した．これにより，分散システム全体で同期したスケジューリングが可能になる．通信量を考慮して FlexRay 通信のパラメータを設計した場合，他ノードのタスクに対するシステムコールの最悪応答時間は予測可能である．

今後の課題としては，まず FlexRay のパラメータ設計を支援するコンフィギュレーション環境の整備があげられる．また，システムの耐故障性を向上させるための機能拡張も重要課題である．FlexRay は耐故障性の向上もその狙いとしており，2 つの通信チャネルを用いた二重系ネットワークをサポートしている．しかし，本分散 RTOS の対象が安全性が重視される分野のシステムであるため，ネットワーク通信やいずれかのノードの処理に異常が発生した場合にも対応できることが必要である．しかしこの課題は分散 RTOS 単体で検討し，解決することは困難である．アプリケーションを含むシステム全体で対故障性への対応について検討を行い，その検討結果を分散 RTOS 仕様に反映する必要があり，今後の課題である．

謝辞 本研究のベースとして使用したTOPPERS/OSEKカーネルの開発者に感謝する．

参考文献

[1] Takada, H. and Sakamura, K.: μITRON for Small-Scale Embedded Systems, *IEEE MICRO*, Vol.15, No.6, pp.46–54 (1995).
[2] OSEK/VDX: OSEK/VDX Operating System Version 2.2.3 (2005).
[3] 阪田史郎，高田広章（編著）：組込みシステム，オーム社 (2006).
[4] Object Management Group: The Common Object Request Broker: Architecture and Specification, Version 3.0, OMG Technical Document formal/02-06-01 (2002).
[5] Object Management Group: Real-Time CORBA Specification, Version 1.1, OMG Technical Document formal/02-08-02 (2002).
[6] Object Management Group: Minimum CORBA Specification, Version 1.0, OMG Technical Document formal/02-08-01 (2002).
[7] Mullender, S.J., van Rossum, G., Tananbaum, A.S., van Renesse, R. and van Staveren, H.: Amoeba: A Distribute Operating System for the 1990s, *IEEE Computer*, Vol.23, No.5, pp.44–53 (1990).
[8] Shin, K.G., Kandlur, D.D., Kiskis, D.L., Dodd, P.S., Rosenberg, H.A. and Indiresan, A.: A Distributed Real-Time Operating System, *IEEE Software*, Vol.9, No.5, pp.58–68 (1992).
[9] Kopetz, H.: Should Responsive Systems be Event-Triggered or Time-Triggered?, *IEICE Trans. Information & Systems*, Vol.E76-D, No.11, pp.1325–1332 (1993).
[10] 石郷岡祐，伊丹悠一，兪 明連，横山孝典：時間駆動処理とイベント駆動処理が共存する組み込み制御システムのための分散処理環境，情報処理学会論文誌，Vol.51, No.3, pp.1056–1067 (2010).
[11] Stankovic, J.A. and Ramamritham, K.: The Spring Kernel: A New Paradigm for Real-Time Systems, *IEEE Software*, Vol.8, No.3, pp.62–72 (1991).
[12] 芝 公仁，大久保英嗣：分散オペレーティングシステムSolelcの設計と実装，電子情報通信学会論文誌，Vol.J84-D-1, No.6, pp.617–626 (2001).
[13] 伊丹悠一，兪 明連，横山孝典：分散型組み込み制御システムのための分散リアルタイムOS，情報処理学会研究報告，Vol.2010-EMB-16, No.33 (2010).
[14] Makowitz, R. and Temple, C.: FlexRay – A Communication Network for Automotive Control Systems, *Proc. 2006 IEEE International Workshop on Factory Communication Systems*, pp.207–212 (2006).
[15] Sun, J. and Liu, J.: Synchronization Protocols in Distributed Real-Time Systems, *Proc. 16th International Conference on Distributed Computing Systems*, pp.38–45 (1996).
[16] De Vito, L., Rapuano, S. and Tomaciello, L.: One-Way Delay Measurement: State of the Art, *IEEE Trans. Instrumentation and Measurement*, Vol.57, No.12, pp.2742–2750 (2008).
[17] Lemieux, J.: *Programming in the OSEK/VDX Environment*, CMP Books, Lawrence (2001).
[18] OSEK/VDX: OSEK/VDX System Generation OIL: OSEK Implementation Language Version 2.5 (2004).
[19] OSEK/VDX: OSEK/VDX Communication Version 3.0.3 (2004).
[20] TOPPERSプロジェクト：TOPPERS/OSEKカーネル，available from ⟨http://www.toppers.jp/osek-os.html⟩.
[21] 本田晋也，高田広章：ITRON仕様OSの機能分散マルチプロセッサ拡張，電子情報通信学会論文誌，Vol.J91-D, No.4, pp.934–944 (2008).
[22] Pop, T., Pop, P., Eles, P., Peng, Z. and Andrei, A.: Timing Analysis of the FlexRay Communication Protocol, *Proc. 18th Euromicro Conference on Real-Time Systems*, pp.203–216 (2006).
[23] Ben, J., Yongming, B. and Anhu, L.: A Method for Response Time Computation in FlexRay Communication System, *Proc. IEEE International Conference on Intelligent Computing and Intelligent Systems*, Vol.3, pp.47–51 (2009).
[24] AUTOSAR: Specification of Multi-Core OS Architecture V1.1.0 R4.0 Rev 2 (2010).
[25] AUTOSAR: Virtual Function Bus V2.1.0 R4.0 Rev2 (2010).

知場 貴洋 （学生会員）

2011年武蔵工業大学知識工学部情報科学科卒業．現在，東京都市大学大学院工学研究科情報工学専攻修士課程在学中．組み込みシステム向けソフトウェアの研究に従事．

齊藤 政典 （学生会員）

2012年武蔵工業大学知識工学部情報科学科卒業．現在，東京都市大学大学院工学研究科情報工学専攻修士課程在学中．組み込みシステム向けソフトウェアの研究に従事．

伊丹 悠一 （正会員）

2008年武蔵工業大学工学部コンピュータ・メディア工学科卒業．2010年同大学大学院工学研究科電気工学専攻修士課程修了．同年日立情報通信エンジニアリング株式会社入社．通信機器のソフトウェア開発に従事．

俞 明連

1994 年安東国立大学校工学部コンピュータ工学科卒業．1996 年浦項工科大学大学院情報通信専攻修士課程修了．同年安東情報大学講師．2002 年嶺南大学コンピュータ工学専攻博士課程修了．2006 年早稲田大学大学院情報生産システム研究科博士後期課程修了．2007 年武蔵工業大学．現在，東京都市大学准教授．スケジューリング理論の研究に従事．博士（工学）．電子情報通信学会，IEEE 各会員．

横山 孝典 （正会員）

1981 年東北大学工学部通信工学科卒業．1983 年同大学大学院工学研究科電気及通信工学専攻修士課程修了．同年（株）日立製作所入社．1987 年から 1990 年まで（財）新世代コンピュータ技術開発機構出向．2004 年武蔵工業大学．現在，東京都市大学教授．組み込みシステム，分散システム，ソフトウェア工学等の研究に従事．博士（情報科学）．電子情報通信学会，IEEE，ACM 各会員．