Vol.2010-SLDM-144 No.33
Vol.2010-EMB-16 No.33
Vol.2010-MBL-53 No.33
Vol.2010-UBI-25 No.33
2010/3/27

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

伊丹悠一[†1]　兪　明連[†1]　横山孝典[†1]

本論文は組み込み制御システムを対象とした分散リアルタイム OS について述べる．本論文の対象は厳しい時間制約が課せられ，CPU やメモリなどのリソースが制限される自動車制御などの組み込み制御システムである．まず，分散制御システム全体で統一的な時刻に基づいたタスク管理を実現するために，ノード間の時刻同期方法を提案する．次に，位置透過性のあるタスク管理及びタスク間同期用のシステムコールを提案する．そして，自動車制御分野の標準 OS である OSEK OS をベースに，提案した機能を持つ分散リアルタイム OS を実装する．ネットワークには TDMA プロトコルに基づく FlexRay を使用する．また，実装したシステムコールの最悪実行時間の予測性について述べる．

A Distributed Real-Time Operating System for Distributed Embedded Control System

YUICHI ITAMI,[†1] MYUNGRYUN YOO[†1]
and TAKANORI YOKOYAMA[†1]

The paper describes a distributed real-time operating system for distributed embedded control system. The target domain of the paper is embedded control systems such as automotive control systems with hard real-time constraints and limited resources such as CPU performance and memory size. we present a inter-node time synchronization mechanism to realize task management with synchronized time. we also present location transparent system calls for a task execution control and inter-task synchronization. We have developed distributed real-time operating system as an extension to OSEK OS, which is a standard OS for automotive control systems. We use FlexRay network based on TDMA protocol. We also describe a predictability of the worst execute time of the system calls.

1. はじめに

自動車や航空機，ファクトリオートメーションなどの分野で組み込み制御システムが利用されている．これらは制御処理に時間制約が存在するハードリアルタイムシステムであり，制御処理を完了すべきデットラインを過ぎると，事故の発生など重大な損害が生じる恐れがある．また，ネットワーク接続した複数の組み込みコンピュータで制御処理を行う分散制御が増えており，ネットワークも含めたシステム全体で時間制約を満たすことが要求されている．このため，分散処理環境で時間制約を満たすことが可能な基盤ソフトウェアが必要である．

情報処理分野では CORBA[1] を始めとした RPC（Remote Procedure Call）をベースとする分散ミドルウェアが広く用いられており，リアルタイムシステム向けの Real-Time CORBA[2] も提案されている．また，分散リアルタイム OS として，Linux をベースにリアルタイム機能を追加することで分散リアルタイム OS を構築する方法が提案されている[3]．しかし，これら分散ミドルウェアや分散 OS は，分散処理環境での最悪実行時間の予測は必ずしも容易ではない．また，CPU やメモリなどのリソースが限られている組み込み制御システムに適用することも困難である．

リアルタイム OS のようなカーネルレベルで，分散処理機能を実行する手法も提案されている[4]．この手法では，クライアントサーバモデルで構成されたコンピュータ間で，分散処理を行うことを目的としている．呼び出すシステムコールとその実アドレスを実行時に関連付けることで，リアルタイム OS に拡張性を持たせることが可能である．しかし，我々が対象としている組み込み制御システムのアプリケーションの多くは，クライアントサーバモデルで構成されているわけではない．タスクの集合として構成されている制御アプリケーションに適した，リアルタイム分散処理を実現できる基盤ソフトウェアが求められている．

自動車制御システム向けの基盤ソフトウェア仕様として OSEK/VDX 仕様がある．この仕様には，リアルタイム OS （RTOS）仕様である OSEK OS[5] や，タスク間の通信仕様である OSEK COM[6] などがある．前述のように，組み込み制御システムではアプリケーションをタスクという単位に分割して実行するが，OSEK OS ではタスクの起動やイベント同期などのタスク管理機能を提供する．一方，OSEK COM では通信データをメッセージ

†1 東京都市大学（旧武蔵工業大学）
Tokyo City University

Vol.2010-SLDM-144 No.33
Vol.2010-EMB-16 No.33
Vol.2010-MBL-53 No.33
Vol.2010-UBI-25 No.33
2010/3/27

という形で表し，送受信する方法が定められている．OSEK COM はメッセージを自ノード上のタスクに送るか，別のノード上のタスクに送るかをアプリケーションに意識させない，位置透過性があるメッセージ通信を提供している．

しかし制御アプリケーションを分散化する場合，メッセージ通信だけでなく別のノード上のタスクの起床や異なるノード上のタスク間での同期といった機能も必要となる．しかし，OSEK OS は単一ノード上でのタスク管理や同期の機能しか提供していない．このため，異なるノード間でのタスクの起床や同期を実現するには，OSEK COM の機能と組み合わせる必要があり，位置透過性が失われる．

本研究は組み込み制御システムを対象に，分散処理環境において位置透過性とリアルタイム性の両者を実現する分散リアルタイム OS を開発することを目的とする．特に自動車制御を主な対象分野とするため，OSEK/VDX 仕様をベースに上記の目的を達成するための拡張を行う．

まず，自ノード上のタスクの起動や同期を行う場合と同一の API を用いて，他ノード上のタスクの起動や同期を可能とする位置透過性を持ったシステムコール（システムサービス）を実現する．本分散リアルタイム OS は，分散処理環境内にあるノードとタスクの位置関係に関する情報を記憶しており，それを参照して発行されたシステムコールの処理対象タスクがどこのノードに割り当てられているかを検索する．他ノード上のタスクの場合は，ネットワークを通じてその要求を送信し，送信先のノードで実行を行う．

また，分散システム全体で同一の時間に基づいたタスク管理を実現するため，ノード間で時刻の同期を行う機能も実現する．提案する分散リアルタイム OS では，システムコールが発行されてから別のノードで実行されるまでにネットワーク通信による遅れ時間が発生するが，同期された同一の時刻を持たせることでこの遅れ時間も予測可能となる．

前述のように本研究では，自動車制御分野においての標準仕様である OSEK/VDX 仕様に基づく OSEK OS をベースに上記で述べた拡張を行う．ネットワークとしては，TDMA（Time Division Multiple Access）に基づいたネットワークで近年自動車分野で普及しつつある，ノード間で同期した通信が可能な FlexRay[7] を用いる．異なるノード上のアプリケーション間で直接通信を行う場合は，OSEK COM を使用する．前述のように OSEK COM は位置透過性のあるタスク間通信を実現しているので，今回の拡張は OSEK OS のみを対象とする．

2. 分散リアルタイム OS の機能と構成

2.1 ノード間の時刻同期

分散処理環境では各ノードが同一の時間に基づいて動作する必要がある．そこで，本研究では各ノードの分散リアルタイム OS で同一の時刻を表すシステム時刻を定義する．FlexRay のネットワーク時刻を用いてシステム時刻を実現する．FlexRay はネットワーク時間全体をある周期 t を基に分割し，この周期ごとに決められたスケジューリングに従って通信を行うリアルタイムネットワークである．そのため，FlexRay では各通信ノード間で同期したネットワーク時刻を持つ．ネットワーク時刻は FlexRay コントローラによって同期されている．このネットワーク時刻を用いて，分散リアルタイム OS のシステム時刻に対して時刻を供給する．供給するネットワーク時刻は，FlexRay が持つ最も大きな時間単位であるコミュニケーションサイクルを用いる．これにより，分散リアルタイム OS が同一の時間を持つことが可能である．分散リアルタイム OS は，このシステム時刻に基づいた分散システム全体で統一したタスク管理が可能となる．

2.2 位置透過性のあるシステムコール

本研究では OSEK OS が持つシステムコールのうち，分散型制御アプリケーションの実行制御に用いられると考えられるシステムコールを選択した．OSEK OS が提供するタスク関連のシステムコールにはタスクの起床や終了などのタスク管理，イベントを用いたタスク間の同期，共有リソースを管理する相互排除などがある．これらのうちタスク管理やタスク間同期は，分散処理環境においても必要と考える．しかし，異なるノード間でリソースを共有する必要性は少ないと考える．そこで，タスク管理とタスク間同期に関するシステムコールに対して位置透過性を持たせることとした．

OSEK OS が持つタスク管理及びタスク間同期を要求するシステムコールを**表 1** に示す．表 1 でタスク指定とは，システムコールが他ノードにあるタスクを指定する可能性があることを示している．タスク指定がある場合は，システムコールが発行された際にネットワーク通信を行い，他ノードで実行する場合がある．そのため ActivateTask，ChainTask，SetEvent の 3 つのシステムコールに対し，他ノードで実行可能とする機能を加える．ただし，ChainTask は自タスクを終了後に対象タスクの起床を行うもので，他ノードへの要求としては ActivateTask と同等の処理であるため，要求を受け取ったノードでは ActivateTask と同一に扱うこととする．

提案する位置透過性のあるシステムコールの動作を**図 1** に示す．図 1 では FlexRay ネッ

Vol.2010-SLDM-144 No.33
Vol.2010-EMB-16 No.33
Vol.2010-MBL-53 No.33
Vol.2010-UBI-25 No.33
2010/3/27

表 1 対象とするシステムコール
Table 1 Intended system calls.

システムコール名	機能	タスク指定
ActivateTask(*Task*)	*Task* を起床する	あり
ChainTask(*Task*)	自タスクを終了し，*Task* を起床する	あり
TerminateTask()	自タスクを終了する	なし
SetEvent(*Task*, *Event*)	*Task* に *Event* を通知する	あり
WaitEvent(*Event*)	*Event* を待つ	なし
ClearEvent(*Event*)	*Event* で自タスクのイベントをクリアする	なし

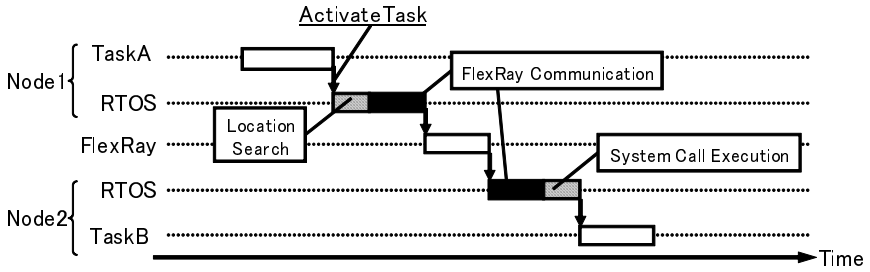

図 1 位置透過性のあるシステムコールの動作
Fig. 1 Behavior of a location transparent system call.

トワークで接続された Node1 と Node2 があり，それぞれ TaskA と TaskB を持つ．この例では，TaskA が TaskB を起床する動作を表している．TaskA は分散リアルタイム OS に対してシステムコール（ActivateTask）を発行する．このとき，分散リアルタイム OS は静的に定義したタスクとノードの位置情報を用いて，TaskB の位置を検索する．TaskB が Node2 にあるため，分散リアルタイム OS はシステムコールの要求を Node2 に送信する．Node2 では要求を受け付け，システムコールを実行し TaskB を起床する．

2.3 システムコールの実行時間の予測可能性

ここでは前述したシステム時刻に基づいた，分散リアルタイム OS の処理に関する実行時間の予測可能性について述べる．分散リアルタイム OS による他ノードへのシステムコールの発行には通信が含まれるため，システムコールが発行され他ノードで実行されるまでに遅れが発生する．この遅れがアプリケーションを設計する際にどの程度のものなのか予測できなければならない．そこでこの遅れ時間を分散リアルタイム OS のシステム時刻を用いて予測する．システム時刻に FlexRay のコミュニケーションサイクルを用いているため，遅れ

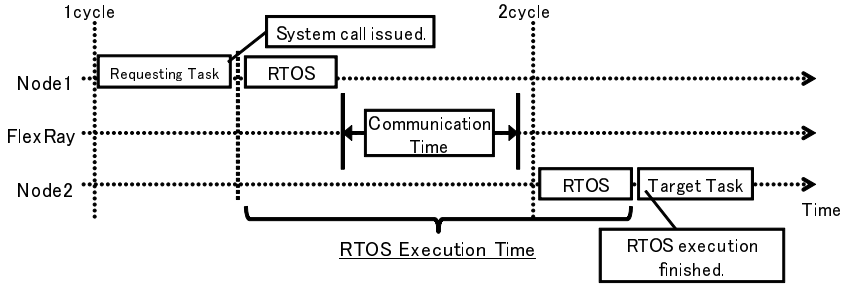

図 2 要求と同じサイクルで送信が行われる場合のタイムチャート
Fig. 2 Time chart of the case of sending a system call during the same cycle as issued.

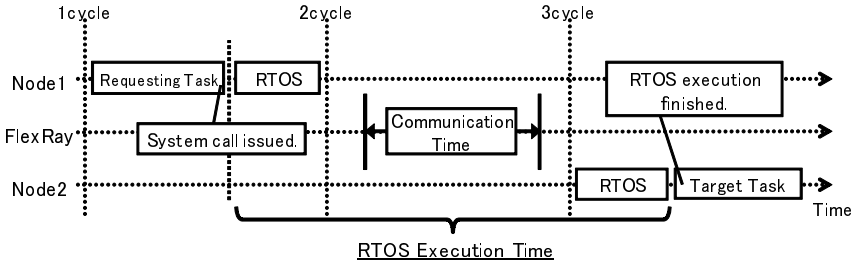

図 3 要求と同じサイクルで送信が行われない場合のタイムチャート
Fig. 3 Time chart of the case of sending a system call after the issued cycle ended.

時間の単位にはサイクルを用いて説明する．

分散リアルタイム OS の実行時間には，システムコールが発行されてから要求を送信するまでの時間と，FlexRay 通信にかかる時間と，システムコール要求を受信してからシステムコールの実行が完了するまでの時間の 3 つに分割できる．このうち，FlexRay 通信にかかる時間はハードウェアによって保障される．

他ノードへのシステムコールの実行には，2 つの場合があり得る．ある時点で発行した他ノードへのシステムコールがネットワークの状況によって，1) 発行したサイクルと同じサイクルで送信できた場合，2) 発行したサイクルと同じサイクル中に送信できなかった場合である．受信側での分散リアルタイム OS は，受信した時点で即座に処理を開始するのではなく，FlexRay のコミュニケーションサイクルの開始時に処理を行うこととする．1) の場合のタ

Vol.2010-SLDM-144 No.33
Vol.2010-EMB-16 No.33
Vol.2010-MBL-53 No.33
Vol.2010-UBI-25 No.33
2010/3/27

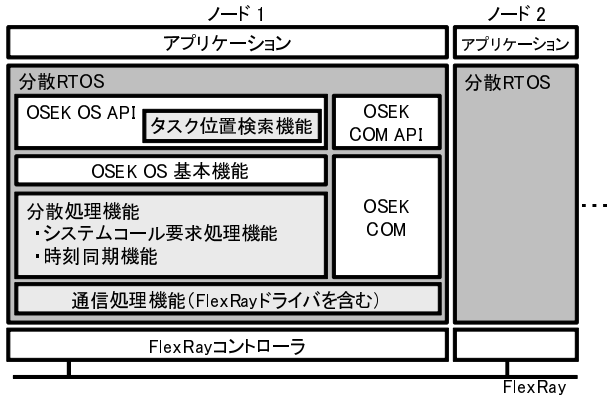

図 4 分散リアルタイム OS の構成
Fig. 4 Distributed real-time operating system architecture.

イムチャートを**図 2** に示す．この例では Node1 の RequestTask が他ノードへのシステムコールを発行した次のサイクルには，Node2 でシステムコールが実行され，TargetTask が起床している．このとき，Node1 の RequestTask が他ノードへのシステムコールを発行してから，Node2 の TargetTask が実行を始めるまでの期間が分散リアルタイム OS が処理を行う時間である．1) の場合はこの期間の長さが 1 サイクル以内におさまるため，他ノードへのシステムコールを発行した際の遅れ時間は，1 サイクルとなる．2) の場合のタイムチャートを**図 3** に示す．この例では 1 サイクル目で他ノードへのシステムコールを発行しているが，1 サイクル目には送信が行えなかったため，次の 2 サイクル目で送信を行っている．この場合，システムコールは 3 サイクル目で実行されることになり，Node1 で要求が発生してから Node2 で実行が行われるまで 2 サイクル遅れて処理されることになる．このように，他ノードへのシステムコールの発行にかかる遅れ時間は，最も短い場合が 1 サイクルであり，最も長い場合が 2 サイクルである．そのため分散リアルタイム OS がシステムコールを他ノードで実行するまでの時間は，最悪 2 サイクルである．この時間に，送信側の RTOS の処理時間と受信側の RTOS の処理時間を加えた時間をシステムコールの最悪実行時間としてアプリケーションを設計することで時間保障が可能である．

2.4 分散リアルタイム OS の構成

提案する分散リアルタイム OS のシステム構成を**図 4** に示す．本分散リアルタイム OS は OSEK OS の持つ基本機能を中核とし，その機能をアプリケーションから利用する OSEK OS API を持つ．この OSEK OS API を拡張し，発行されたシステムコールが他ノードへのシステムコールかどうかを判断するタスク位置検索機能を追加した．また，分散処理を担う部分として分散処理機能を追加した．分散処理機能にはシステムコール要求処理機能と時刻同期機能がある．システムコール要求処理機能は他ノードの対してシステムコールを発行する機能と，他ノードから受け付けたシステムコール要求を実行する機能を持つ．時刻同期機能はネットワーク時刻を FlexRay コントローラからシステム時刻に供給する機能を持つ．また，通信処理機能は FlexRay ドライバを含み，他ノードに要求を出す通信機能を持つ．

3. 分散リアルタイム OS の実装

3.1 実装の方針

提案した分散リアルタイム OS を実現するために，OSEK OS 仕様のリアルタイム OS である TOPPERS/OSEK カーネル[8] をベースに分散処理機能を追加する．システムコール要求処理機能はシステムコール送信処理，システムコール受信処理の 2 通りがある．システムコール送信処理はアプリケーションからシステムコールが発行されたときに行われる．システムコール受信処理は FlexRay のコミュニケーションサイクル開始時に割り込みから起動される．分散処理機能ではデータとして，ノードとタスクの位置関係を表す位置情報と，発行されたシステムコールを保持するシステムコールバッファと，システム時刻を持つ．タスクの位置情報とシステムコールバッファはシステムを構築する段階で静的に決定される．さらに，ユーザが他ノードのタスクを指定可能とするため，分散リアルタイム OS が管理するノード間で共通なタスク ID であるグローバルタスク ID を定義する．タスクの位置情報はこのグローバルタスク ID とノードの位置関係を表す．

3.2 ネットワーク通信

前述のようにネットワークとして FlexRay を用いる．本研究では各ノードが同期しているネットワーク時間を持つ点を利用し，分散リアルタイム OS のシステム時刻を供給するために使用する．このネットワーク時刻の最も大きい単位がコミュニケーションサイクルである．コミュニケーションサイクルはスタティックセグメント，ダイナミックセグメント，シンボルウィンドウ，ネットワークアイドル時間から構成される．

アプリケーションはスタティックセグメントとダイナミックセグメントを使って通信をすることができる．スタティックセグメントはシステム構築時にノードごとに通信で使用できる時間が静的に決められる．その時間は一定間隔で区切られ，スロットカウンタ値（スロッ

Vol.2010-SLDM-144 No.33
Vol.2010-EMB-16 No.33
Vol.2010-MBL-53 No.33
Vol.2010-UBI-25 No.33
2010/3/27

ト ID）を使って管理される．一方，送信データにはフレーム ID が割り当てられ，スロットカウンタ値とフレーム ID が合うと送信が行われる．そのため，ネットワーク通信時間が予測しやすい．

ダイナミックセグメントでは任意のタイミングで送信を行え，送信するフレーム長も可変である．ダイナミックセグメントでもスタティックセグメントと同様にスロット ID を持つ．ダイナミックセグメントでは送信データがない場合は，静的に決められた最小通信間隔だけ待った後にスロット ID を進める．このような特徴から，ダイナミックセグメントではフレーム ID が小さいものが優先度が高いことになる．

分散リアルタイム OS のシステム時刻はコミュニケーションサイクルと同期しているため，同システム時刻内ではどのタイミングで送信が行われても構わない．しかしスタティックセグメントでは，他ノードへシステムコールを実行したタイミングと，そのノードが通信を行える時間が必ずしも一致しない場合がありうる．そこでシステムコール用の通信にはダイナミックセグメントを用いる．

3.3 システムコール送信処理

送信処理はアプリケーションのシステムコール呼び出しを起点として処理される．アプリケーションは他ノードに対してシステムコールを発行するときも OSEK OS API を用いる．API の形式は OSEK OS 仕様と同一であるが，指定するタスク ID にグローバルタスク ID を使用する．

要求を受け付けた分散リアルタイム OS はタスク位置検索を行う．タスク位置検索では，分散リアルタイム OS 中に記憶されたタスク位置情報からタスクが割り当てられたノードを探す．タスク位置情報の詳細は 3.5 項で述べる．指定されたタスクが他ノードにある場合は，そのノードに対してシステムコールを発行する．

他ノードへのシステムコールを実行するときは，**表 2** に示したパラメタを決定する．送信元タスク ID はシステムコールを発行したタスクの ID を，送信先タスク ID はシステムコールに指定されたタスク ID を使用する．発行されたシステムコールの種類は，OSEK OS が持つ呼び出された API の ID を定義し，使用する．また，システムコールによっては引数を持つが，その場合は引数そのものを渡す．これらの情報を，システムコール送信処理を呼び出すときに渡す．システムコール送信処理では発行された他ノードへのシステムコールをシステムコールバッファに書き込み，その後，FlexRay ドライバを呼び出す．FlexRay ドライバでは，FlexRay コントローラ内のメッセージ RAM に変換した送信データを書き込む．通信自体は FlexRay コントローラが代行し，システムコールが送信される．

表 2 送信するシステムコールパラメタ
Table 2 System call parameters.

パラメタ	データサイズ (Byte)
処理要求タスク ID（送信元タスク ID）	1
処理対象タスク ID（送信先タスク ID）	1
発行されたシステムコールの種類	1
システムコールの引数	4

3.4 システムコール受信処理

受信処理は FlexRay のコミュニケーションサイクル開始時に割り込みを起点として行われる．この割り込みは OSEK OS 仕様で規定された OS が関知する割り込みである ISR カテゴリ 2 を用いる．この割り込み処理内では，まずシステム時刻の更新を行う．次に前サイクルで受信したデータを FlexRay コントローラから読み出す．そしてそのデータを解読しシステムコール要求であれば，そのシステムコールをシステムコールバッファに書き込む．システムコールバッファの詳細は 3.7 で述べる．バッファに書き込まれたシステムコールは FlexRay のスロット ID が小さい順に読み出され，その後に受信した順序に従って要求されたシステムコールを処理する．これにより，要求元が発行した順序でシステムコールを実行可能である．

3.5 タスク位置情報

タスクの位置情報は分散リアルタイム OS 全体で共通のグローバルタスク ID と，その内で自ノードに割り当てられているタスクが持つグローバルタスク ID のテーブルで表される．このテーブルは一次元配列で表され，全てのノード内にあるタスク数分の要素を持つ．この配列の添え字がグローバルタスク ID に対応し，その添え字が示す要素には自ノードのタスクを表す値か，他ノードのタスクを表す値かのいずれかが格納される．この配列はノードごとに合わせて静的に記述する．

タスク位置情報は後述するコンフィギュレーションにより生成される．タスク位置検索では，発行されたシステムコールの引数として渡されたグローバルタスク ID をインデックスとして，配列から値を取り出す．これにより，指定されたタスクがどのノードに割り当てられるかを検索する．

3.6 システム時刻

システム時刻は FlexRay のコミュニケーションサイクルを用いて表される．そのため，FlexRay のコミュニケーションサイクルの開始に同期して更新される．このタイミングで

Vol.2010-SLDM-144 No.33
Vol.2010-EMB-16 No.33
Vol.2010-MBL-53 No.33
Vol.2010-UBI-25 No.33
2010/3/27

FlexRay コントローラからネットワーク時刻を取得し，システム時刻を更新する．

システム時刻の同期は，FlexRay 通信が確立されたタイミングで開始される．3 つのノードが同期を取る場合を例にとって説明する．各ノードは電源が投入された際に，分散リアルタイム OS と FlexRay コントローラの初期化を行う．このときシステム時刻は初期状態になる．その後で FlexRay 通信を確立するが，そのためには最低でも 2 ノードが必要である．2 ノードが FlexRay 通信が確立するとコミュニケーションサイクルが開始される．このときシステム時刻はこの 2 ノード間で同期状態になる．続いて 3 つ目のノードが FlexRay 通信を確立し，そのノードでも FlexRay 通信が同期される．このとき，3 つ目のノードは前の 2 つのノード間での時刻に同期し，3 つのノード間でシステム時刻が同期される．このとき，システム時刻の現在値は必ず同じコミュニケーションサイクル中に渡されなければならないため，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で送信する．同期状態にあるノード内では，マスタとなるノードがおり，他ノードにマスタノードのシステム時刻を送信する．マスタノード以外はこの時刻とずれていた場合，同期がずれていると判断しマスタノードにシステム時刻を合わせる．

3.7 システムコールバッファ

システムコールバッファは，他ノードへのシステムコールが発行された際に，送信側と受信側でそれぞれ別の領域を設け，そこに対してシステムコールの情報を書き込む．送信側は送信したシステムコールの情報を保持し，対象ノードからの返戻値を受け取ったときにバッファから削除される．加えてサイクル開始時点でチェックを行い，送信が行われていないシステムコールがあった場合は優先して送信を行う．受信側は受信したシステムコールの情報を保持し，システムコールを実行し戻り値を送信したときに削除される．分散処理機能が使用するシステムコールバッファは全てのコミュニケーションサイクルにおいて，最も送信される場合が多いタイミングに合わせて用意する．

3.8 分散リアルタイム OS のコンフィギュレーション

本研究の分散リアルタイム OS を用いてアプリケーションを構築する場合は，前節までに述べた，タスクの位置情報やシステムコールバッファといった分散リアルタイム OS の機能を提供する際に使用する情報と，FlexRay 通信を行う際に分散リアルタイム OS が占有できる時間を決める必要がある．これを行うのが分散リアルタイム OS のコンフィギュレーションである．本研究では大きく分けて OSEK OS の構成の決定，分散処理機能が使用する情報の決定，FlexRay の通信時間の決定の 3 つがある．

OSEK OS におけるアプリケーションに依存する構成情報は，タスクの数や優先度といったタスクに関する情報や，イベントの定義やイベントに関連付けるタスクといったイベント機構に関する情報などがある．これらは OSEK OS 仕様では OIL（OSEK Implement Language）[9] を用いて，アプリケーション開発者が記述する．記述された OIL は SG（System Generator）を用いて OSEK OS の実装に合わせたプログラムコードに変換される．ノードごとの OSEK OS の構成が決定された後に，その情報をもとに本研究が提案する分散リアルタイム OS が使用する情報が決定される．3.5 項で述べたタスク位置情報は，OIL から生成されたプログラムコードをもとに決定される．

FlexRay 通信において，分散リアルタイム OS が行う他ノードへのシステムコール実行の他に，アプリケーションが独自に通信を行う場合がある．そのため，FlexRay の通信時間の決定では分散リアルタイム OS が占有する時間を提示することを行う．これにより，分散リアルタイム OS とアプリケーションが独自に行う通信をまとめてスケジューリングすることが可能となる．分散リアルタイム OS では，スタティックセグメントに対して，分散リアルタイム OS が管理するノード数分だけフレームを要求する．さらに，3.7 項で述べた，発行される可能性があるシステムコールの数だけフレームを要求する．これらの情報をに加え，アプリケーションが使用する可能性があるフレームの数を考慮することで FlexRay の通信時間のスケジューリングを行う．その結果として，分散リアルタイム OS が利用可能なフレーム ID を得る．

なお現時点では，本分散リアルタイム OS において新規に必要となったタスク位置情報や FlexRay 関連情報については手作業で生成している．今後は OIL を拡張し，それらの情報を自動生成することを計画している．

4. 評 価 実 験

4.1 実 験 環 境

本研究では実装した分散リアルタイム OS が自動車制御における X-by-Wire などの組み込み制御システムに適用可能かどうかを，性能を計測し評価する．一般に組み込み制御システムでは，必要最低限の計算能力を持った CPU を選ぶなどリソースに制約がある．そのため，実装した分散リアルタイム OS が占有する CPU 処理時間を計測し，処理時間を予測するための基礎データを取得することを目的とする．実験環境には V850E/PHO3 プロセッサと E-Ray（FlexRay コントローラ）搭載した，GT200N10 を用いた．計測を行った実験環境は**表 3** の通りである．本研究では，FlexRay のコミュニケーションサイクルは 2msec と設定した．従って，分散リアルタイム OS が持つシステム時刻も 2msec である．

Vol.2010-SLDM-144 No.33
Vol.2010-EMB-16 No.33
Vol.2010-MBL-53 No.33
Vol.2010-UBI-25 No.33
2010/3/27

表 3 実験環境
Table 3 Experiment environment.

CPU	V850E/PHO3(32bit RISC 128MHz)
内臓メモリ	992KB（ROM），92KB（RAM）
FlexRay	コントローラ 80MHz（通信速度 10Mbps）
計測用ハードウェアタイマ	クロックカウンタ (32MHz)

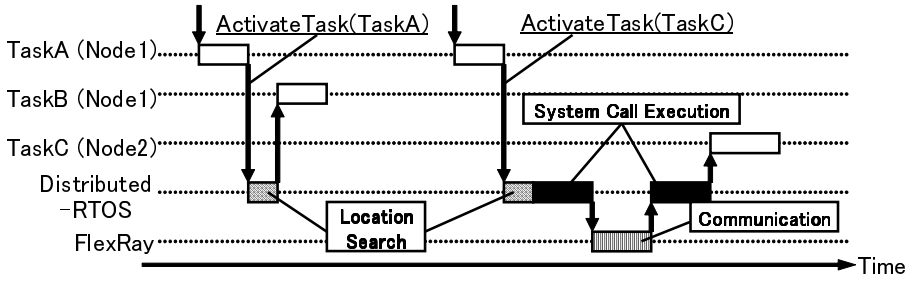

図 5 評価用プログラムの動作
Fig. 5 Behavior of evaluation program.

4.2 評価項目

実験では分散リアルタイム OS を用いて評価用プログラムを実行する．その評価用プログラムでは**図 5** に示すように，複数のノードにそれぞれタスクを配置する．評価用プログラムは，Node1 上の TaskA が同一ノード上にある TaskB と，異なるノード上にある TaskC にそれぞれシステムコールを発行する．

システムコールが発行され分散リアルタイム OS の実行が終了するまでの，各ノード上での処理時間を計測する．1)TaskC を対象としたシステムコール発行要求の送受信処理を行い他ノードで実行する場合と，2)TaskB を対象としたシステムコール発行要求を同ノード内で処理する場合の 2 通りの場合について計測する．

1) では分散リアルタイム OS の処理時間を計測し，性能を評価することが目的である．このとき，システムコールの送信処理と受信処理を FlexRay ドライバの処理時間を含め，それぞれ計測する．2) では同ノード内での実行であり，分散処理機能を付加した場合に既存環境と比べどの程度オーバヘッドが増加したかを評価するのが目的である．

どの場合も 50 回計測を行い，その平均値を求めた．計測に使用したハードウェアタイマは，32MHz のクロック信号をカウントするタイマで， カウントは 31.25nsec である．

表 4 他ノードへのシステムコールの実行時間
Table 4 Execution time of system calls to other node.

システムコール名	送受信	RTOS 実行時間	FlexRay ドライバ
SetEvent	送信	21.7	12.9
SetEvent	受信	21.8	12.9
ActivateTask	送信	21.6	12.9
ActivateTask	受信	22.9	12.9
ChainTask	送信	24.5	12.9
ChainTask	受信	22.9	12.9

4.3 システムコール送信処理，受信処理の計測結果

1) の場合の計測結果を**表 4** に示す．2.2 項で述べたように，ChainTask は他ノードで実行される際には ActivateTask として扱う．従って，表 4 の ActivateTask と ChainTask の受信処理にかかった時間は同じである．

システムコール送信処理は SetEvent と ActivateTask ときに，20μsec 強である．一般に自動車制御アプリケーションの制御周期は 1〜100msec 程度であり，これらの対して十分に小さく，実用上問題ない値と考える．同じシステム時刻の中で複数回のシステムコールを発行する場合は，1 つのシステムコールの送信時間と，アプリケーションの制御周期と，他ノードへのシステムコールの実行にかかる時間を考慮し，対象となるアプリケーションが使用できるシステムコールの回数の上限を判断する必要がある．

4.4 分散処理機能によるオーバヘッドの比較

2) の場合は，開発した分散リアルタイム OS のベースとした TOPPERS/OSEK カーネルに比べて，どの程度オーバヘッドが発生したかを計測する．計測は，Node 1 において Task A から Task C に対して SetEvent，ActivateTask，ChainTask を発行した場合にかかる時間を，それぞれ開発した分散リアルタイム OS と TOPPERS/OSEK カーネルで計測する．その結果を**表 5** に示す．

この結果より，分散処理機能の追加による，自ノードのタスクに対する処理のオーバヘッドは 0.1〜0.3μsec 程度で，5%程度の増加に抑えられており，実用上問題ないと考える．このオーバヘッドは分散リアルタイム OS が行うタスクの位置検索にかかる時間である．OSEK OS はタスク構成を静的に定義しており，本研究でも OSEK OS の思想に基づき，ノードとタスクの位置を静的に定義することにより，タスクの位置情報を少ないオーバヘッドで検索することが可能であった．

Vol.2010-SLDM-144 No.33
Vol.2010-EMB-16 No.33
Vol.2010-MBL-53 No.33
Vol.2010-UBI-25 No.33
2010/3/27

表 5 分散処理機能のオーバヘッド
Table 5 Overhead of distributed processing functions.

システムコール＼OS	TOPPERS/OSEK カーネル	開発した分散リアルタイム OS
SetEvent	3.51	3.67
ActivateTask	4.58	4.81
ChainTask	6.71	6.99

5. おわりに

組み込み制御システムを対象として，分散処理環境内の各ノード上のタスクを統一的に扱うことが可能な，位置透過性のある分散リアルタイム OS を開発した．本分散リアルタイム OS は自動車制御分野の標準 OS を拡張したもので，ネットワークとしては TDMA に基づく FlexRay を用いる．本分散リアルタイム OS では，分散処理環境内の他ノードに対して自ノード内と同一のシステムコールによりタスク管理やタスク間同期の要求を発行可能とした．また，ノード間で同期したシステム時刻を導入し，他ノードのタスクに対するシステムコールの処理時間を予測可能とした．評価実験では他ノードへのシステムコールの実行にかかる時間を計測し，実用上問題のない性能であると判断した．

現時点では，本分散リアルタイム OS のコンフィギュレーションは手作業で行っている．OSEK/VDX 仕様では OIL を使ったシステム構成定義が行われるため，現在 OIL を拡張して本研究の分散リアルタイム OS の設定を定義できるようにするとともに，拡張した OIL を解釈しソースコードを生成するツール（システムジェネレータ）を開発中である．これにより効率的な分散処理システム開発を実現できる．

参 考 文 献

1) OMG Technical Document formal/02-06-01：The Common Object Request Broker: Architecture and Specification, Version 3.0, 2002.
2) OMG Technical Document formal/02-08-02：Real-Time CORBA Specification, Version 1.1, 2002.
3) 徳田英幸, 西尾信彦, 永田智大, 間博人; 分散リアルタイム OS 技術. IPSJ Magazine Vol.44 No.1 pp.19-26, Jan. 2003.
4) Boo-Geum Jung, Young-Jun Cha, Hyung-Hwan Kim, Sung-Ik Jun, Ju-Hyun Cho, ; Dy- namic Code Binding for Scalable Operating System in Distributed Real-Time Systems, Proceedings of Real-Time Computing Systems and Applications, pp.96-100, 1995
5) OSEK/VDX, OSEK/VDX Operating System Version 2.2.3 , OSEK/VDX, February 17th 2005.
6) OSEK/VDX, OSEK/VDX Communication Version 3.0.3, July 20 2004.
7) Makowitz. R, and Temple. C, ; FlexRay . A Communication Network for Automotive Control Systems, Proceedings of 2006 IEEE International Workshop on Factory Communication Systems, pp.207-212, 2006.
8) TOPPERS/OSEK カーネル http://toppers.jp/osek-os.html.
9) OSEK/VDX System Generation OIL: OSEK Implementation Language, Version 2.5, July 1, 2004.