2つのOS機能を共存させた組込みシステムにおけるタスク優先順位制御方法

齊藤雅彦[†] 加藤直[†] 大野洋[†]
井上太郎[†] 上脇正[†] 中村智明[†]

1つのハードウェアプラットフォーム上にリアルタイムOSと汎用OSとを同時動作させる組込みシステム環境を開発した．リアルタイムOS上で既存アプリケーションを動作させ，汎用OS上でオープンソフトウェアを実行する．2つのOS機能の実行によるリアルタイム性低下を避けるため，異なるOS間での優先順位体系の差違を吸収し，より高い重要性を有するタスクを実行しているOSを動作させる「優先順位統一モデル」を実現した．優先順位統一モデルでは，システム全体で共通の「正規化優先順位」という概念を導入する．各OS上で実行されるタスクの優先順位は，いったん正規化優先順位に変換・比較され，比較結果に応じて実行すべきOSを選択する．これにより，より高い正規化優先順位を有したタスクを実行しているOSが優先的に動作する．すなわち，いずれのOSのタスクであっても，最も高い正規化優先順位のタスクがつねに処理を実行する．優先順位統一モデルを搭載した組込みシステム環境を実現し，200 MHz動作可能な組込み向けプロセッサSH-4上で動作させた結果，割込み処理は最大9.6 μ秒のオーバヘッド増で，タスクスケジューリングは最大6.5 μ秒のオーバヘッド増で実行できることを確認した．

Method for Controlling Task Priorities on Embedded Systems Incorporating Two Different Operating Systems

Masahiko Saito,[†] Naoshi Kato,[†] Hiroshi Ono,[†] Taro Inoue,[†] Tadashi Kamiwaki[†] and Tomoaki Nakamura[†]

This paper describes an embedded system environment that executes a real-time OS and a general purpose OS concurrently on one hardware platform. The real-time OS executes legacy applications. The open software will run on the general purpose OS as well. Even on such a hybrid OS environment, it is necessary to preserve the real-time capability for embedded applications. We have introduced a core function for embedded systems: "Priority Unification Model" which switches a pair of OS in accordance with the priorities of tasks executed by each OS. The priority unification model introduces the notion of "Normalized Priority" into which a priority of each OS is translated. By comparing the normalized priorities of a pair of OS, it executes the OS that has the higher priority task. Accordingly, the highest task will run among those of a pair of OS. By developing the system on 200 MHz SH-4 embedded processor, it is able to process interrupts with the maximum overhead of 9.6 microseconds and to execute task scheduling with the maximum overhead of 6.5 microseconds.

1. はじめに

インターネットや家庭内ネットワークの普及に従い，家電品，カーナビゲーションシステム等の組込みシステムにおいて，オープンかつ汎用的なOS（オペレーティングシステム）を採用する動きがさかんになっている．組込みシステムにおける汎用OSの代表例としてMicrosoft®社が開発したWindows® CEがある．また，Linus Torvaldsらが開発したLinux®も組込み分野に進出しつつある．これらの汎用OSは，μITRON，VxWorks®等の既存リアルタイムOSに比べて，圧倒的な数のアプリケーション開発者に支持されており，Webブラウザ/サーバ，文書作成ソフトウェアといったアプリケーションも多い．しかしながら，前述した組込みシステムは既存リアルタイムOSと一体で構築されたものが多く，これらをただちに汎用OS上に移植することは費用，時間の面からみて困難である．

上記のような問題点を解決するための1つの方法として，ハイブリッドOS[1)]と呼ばれる実行環境がある．

† 株式会社日立製作所
Hitachi, Ltd.

ハイブリッド OS とは 2 つ以上の異種 OS を 1 つのプロセッサで動作させるシステムである．VMware™ [2]，RTX™ [3)~5)]，DARMA（Dependable Autonomous Hard Real-time Management）技術[6] 等のパソコン（PC）向けのシステムが有名であるが，組込み向けにも適用できるものとして Accel-μ [7]，RT-Linux [8] 等がある．Accel-μ は μITRON と Windows CE とを共存させる技術であり，RT-Linux はリアルタイムカーネル上に非リアルタイム Linux を搭載したシステムである．リアルタイム OS と汎用 OS とを 1 つのプロセッサ上で共存させることにより，既存組込みアプリケーションをリアルタイム OS 上で，インターネットアクセス等のオープンアプリケーションを汎用 OS 上で動作させることができる．

多くの組込みシステムが備えるべき特徴として，応答性・リアルタイム性がある．組込み向けハイブリッド OS 実行環境を構築する際においては，2 つの OS を搭載することによるリアルタイム性低下を避けなければならない．たとえば，リアルタイム OS と汎用 OS との間で処理の重要性を考慮したスケジューリングを行うことが必要となる．

本研究では，組込み向けハイブリッド OS 実行環境の検討を行い，より重要なタスクを実行する OS を優先して動作させる「優先順位統一モデル」を実装した組込みシステム環境を開発した．これにより，ハイブリッド OS 実行環境においても，従来の組込みシステムと同様，リアルタイム性を保証することが可能となる．

本稿では，開発した組込みシステム環境の概要について，上記特徴を中心として述べる．上述した DARMA 技術を組込み向けハードウェアプラットフォームに適用し，その上で，優先順位統一モデルを実現した．組込み向けプロセッサ SH-4（200 MHz 動作）[9] を搭載したリファレンスプラットフォーム上で μITRON と Windows CE とを共存させ，タスクの優先順位に応じた OS 切替えを実行できる．本システムでは，割込み処理を最大 9.6 μ 秒のオーバヘッド増で，タスクスケジューリングを最大 6.5 μ 秒のオーバヘッド増で実行できることを確認した．

2. 関連研究

ハイブリッド OS 実行環境は PC 向けに多数開発されている[10)~12)]．VMware™ [2]，RTX™ [3)~5)]，INtime® [4]，DARMA 技術[6] 等，PC アーキテクチャ上で複数の OS 機能を実行する数多くのシステムが提案されている．また，組込みシステム向けのハイブリッド OS 実行環境としては，Accel-μ [7] がある．

VMware は，大型計算機における仮想計算機（Virtual Machine）技術と同様，単一の PC 上で複数の仮想 PC を模擬するものである．Windows®，Linux®に代表される汎用 OS を複数個搭載することができる．これにより，異なる OS 上のアプリケーションを同時に実行することや，別 OS をターゲットとしたアプリケーション開発を行うことが可能となる．しかし，VMware には，特にリアルタイム性を確保するための機能は存在していない．仮想 PC は，仮想計算機技術と同様，TSS（Time Sharing System）スケジューリング形式で順次実行されるため，個々の汎用 OS 上で動作するアプリケーションについてリアルタイム性を保証することは困難である．また，VMware を除くハイブリッド OS 実行環境の特徴として，リアルタイム OS は必ず汎用 OS に優先して動作するという点があげられる．すなわち，リアルタイム OS 上で何らかのタスクが動作している場合には，汎用 OS 上のタスクは実行されない．なお，リアルタイムカーネル上で Linux を動作させる RT-Linux [8] も同様のハイブリッド OS 実行環境であるといえる．これらのハイブリッド OS 実行環境では，リアルタイム OS 上で動作するタスクの応答性・リアルタイム性を保証することは可能であるが，汎用 OS の応答性に関しては注意が払われていない．しかしながら，汎用 OS においても適度な応答性が必要となることがある．

汎用 OS における応答性について，**図 1** に示すカーナビゲーションシステムを例として説明する．カーナビゲーションシステム上には，経路探索・経路誘導を行う既存ナビゲーションソフトウェアに加え，最近では，渋滞・事故情報を外部から入手するための手段として，インターネットアクセス・モバイル通信等を行うオープンソフトウェアが搭載される傾向にある．このような既存ソフトウェア資産とオープンソフトウェアとが混在するシステムにおいては，既存ソフトウェアをリアルタイム OS 上で，オープンソフトウェアを汎用 OS 上で実行するハイブリッド OS 実行環境が適している．たとえば，GUI，ネットワーク，マルチメディア等の機能に優れた汎用 OS 上でユーザインタフェースを実行し，経路探索・経路誘導・緊急通信等の既存かつ信頼性を必要とする処理をリアルタイム OS 上で動作させる方法が自然である．

ここで，カーナビゲーションにおける経路探索は通常数秒から数十秒オーダの処理であり，上記ハイブリッド OS 実行環境を用いると，経路探索中には汎用 OS のアプリケーションが動作しないという問題が発

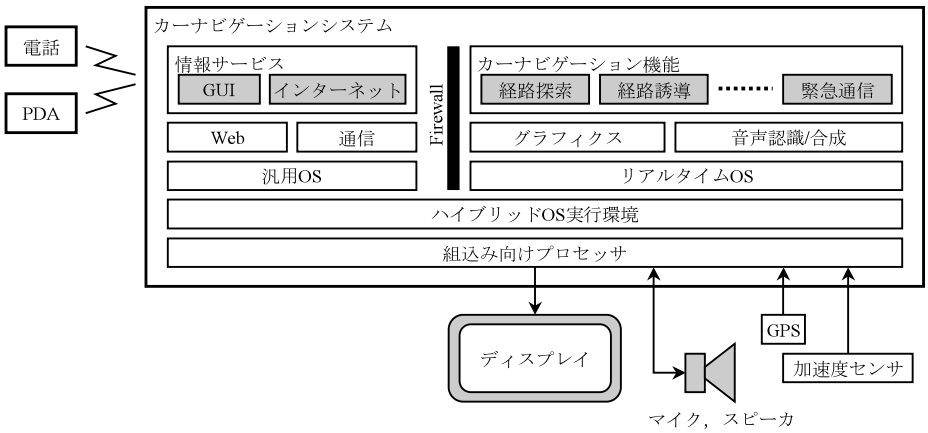

図 1 カーナビゲーションシステム
Fig. 1 Car navigation system.

生する．GUI，タッチパネル等を用いたユーザインタフェースの処理を汎用 OS 上で動作させている場合には，これらのユーザインタフェース処理が数秒から数十秒停止することになる．カーナビゲーションシステムにおける望ましいユーザインタフェースの応答性を得ることができない．

本研究の目的は，ハイブリッド OS 実行環境のいずれの OS においても応答性・リアルタイム性を保証できるようにすることである．近年の OS の傾向として，組込み Linux，Windows CE に代表されるように，汎用 OS にもある程度のリアルタイム性が導入されている．これらの汎用 OS と既存リアルタイム OS とを組み合わせたハイブリッド OS 実行環境では，さらに，汎用 OS 側の応答性・リアルタイム性の保証が要求される．

3. 2 つの OS 機能を共存させた組込みシステム環境

我々は，ハイブリッド OS 実行環境の 1 つである DARMA 技術を組込み向けマイクロプロセッサ SH-4 に適用し，リアルタイム OS である μITRON と汎用 OS である Windows® CE とを並行して動作させるシステムを開発した[13]．μITRON 上で既存アプリケーションを動作させ，Windows CE 上でオープンソフトウェアを実行する．本システムの概要を**図 2** に示す．DARMA 技術本来の特徴として，以下の 3 点がある[6]．

(1) **資源分割機能**

メモリ，入出力機器を分割して各 OS に割り付ける．プロセッサは時分割，タイマは仮想化して，各 OS で共用する．プロセッサ，タイマ以外のハードウェアデバイスは，いずれか一方の OS から占有して使用される．組込みシステム向け特有の機能としては，ハードウェアデバイスからの割込みを動的に各 OS に割り振る機能を搭載している[13),14)]．

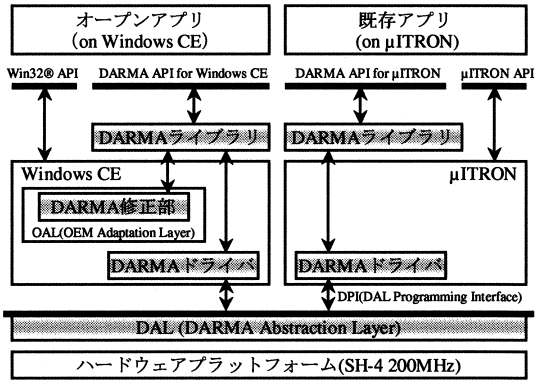

図 2 2 つの OS 機能を共存させた組込みシステム
Fig. 2 Embedded system incorporating two different operating sytems.

(2) **OS 間連携機能**

異なる OS 上のタスク間で共有メモリ，メッセージ通信，セマフォ（排他制御）を使用できる．

(3) **障害監視，回復機能**

OS の動作状況を監視する機能，および，OS に障害が発生した場合，該当 OS のみをリスタートさせる機能等を実現する．

上記 3 つの機能を実現するため，DAL（DARMA Abstraction Layer）というソフトウェア階層を実現し，これに対応するライブラリ・デバイスドライバを両 OS 上に実装した．DAL は SH-4 プロセッサのオブジェクト形式で約 32 KB のテキスト領域および 32 KB 程度のデータ領域から構成される（データ領域のサイ

表 1 主要 DPI 関数
Table 1 Main DPI functions.

機能	関数
資源分割機能	割込み割り振り変更 仮想タイマ生成・起動・停止 タスク優先順位通知
OS 間連携機能	共有メモリ作成 メッセージキュー作成・削除 メッセージ送受信 セマフォ作成・削除 セマフォ獲得・開放
障害監視・回復機能	OS 異常監視 特定 OS リスタート 特定 OS シャットダウン

ズはシステム内の最大タスク数に依存する).

DAL は主要なハードウェア（プロセッサ，タイマ）のみを管理する．これ以外のハードウェアは，基本的には，DAL 上で動作するいずれかの OS に占有される．VMware™や大型計算機で用いられる仮想計算機とは異なり，ハードウェアを仮想化して提供するものではない．各 OS は直接ハードウェアを制御することができ，組込み向けプロセッサにおいても極端な性能低下は発生しない．

DAL と OS との間の連携は DPI（DAL Programming Interface）と呼ぶインタフェースで規定されている．DPI に属する主要な関数を**表 1** に示す．

各 OS 上には，DPI に従ったライブラリ・デバイスドライバを作成する必要がある．DARMA ライブラリは，表 1 に示した関数を各 OS 上のタスクに提供するためのモジュールである．一方，メッセージ送受信を行った場合等，DAL から各 OS に対して処理を依頼することがある．このとき，DAL からの処理依頼によって各 OS 上でタスクスケジューリングが行われる可能性がある（したがって，OS 自体の切替えも行われる可能性がある)．コールバック関数によってこのような処理を実現することはできないため，DAL はソフトウェアによる擬似的な割込みによって各 OS に処理を依頼する．各 OS 上には，通常の割込み処理と同様の形式で，DAL からのソフトウェア割込みを処理する DARMA ドライバを実装しなければならない．

μITRON と Windows CE とを SH-4 上で動作させる組込みシステムでは，DARMA ライブラリ・デバイスドライバは両 OS とも 5KB 程度のプログラムで実装できている．

前述したように，組込みシステムに DARMA 技術を導入する際には，リアルタイム OS と汎用 OS にまたがるリアルタイム性保証を行う必要がある．このため，本研究では，優先順位統一モデルと仮称する新たな実行環境と，それに基づく優先順位統一スケジューリング機能とを実現した．優先順位統一スケジューリング機能とは，異なった優先順位体系を有した OS 間で，より重要性の高いタスクを実行している OS を優先して動作させる OS 切替え機能である．

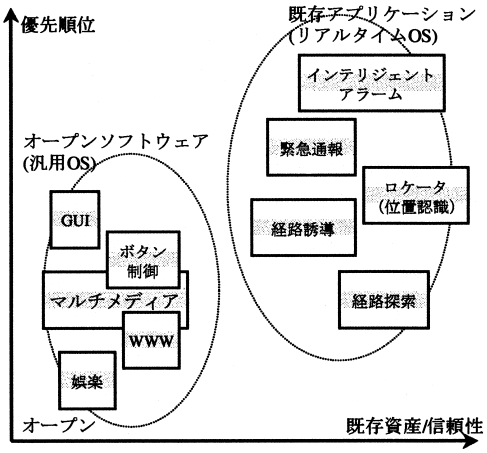

図 3 優先順位と信頼性との関係
Fig. 3 Relation between priority and reliability.

優先順位統一モデルは，表 1 に示すタスク優先順位通知関数，および，発生した割込みを DAL がいったん受理し，各 OS に割り振る機能から構成されている．以下，優先順位統一モデルとその実装について詳細を説明する．

4. 優先順位統一モデル

各 OS へのプロセッサ割当て（スケジューリング）に関して最も単純な方法は，既存リアルタイム OS がアイドル状態に移行した場合のみ汎用 OS に切り替えるという方式である．通常，既存アプリケーションや信頼性を要するソフトウェアをリアルタイム OS 上で，オープンソフトウェアを汎用 OS 上で動作させるため，この方式で十分な場合が多い．しかしながら，2 章に示したカーナビゲーションシステムに関する例を考えると，経路探索等のように時間のかかる処理を実行している最中には，ユーザインタフェースの応答性が低下するといった問題点が生じる．すなわち，既存アプリケーションのリアルタイム性は達成されるが，オープンソフトウェアの応答性が保証されない．この問題は，いい換えると，**図 3** に示すように，既存資産/信頼性と優先順位とが必ずしも相互に一致しないということである．このため，本研究では，優先順位の高いタスクを実行している OS を優先して動作させるという「優先順位統一モデル」（Priority Unification Model）を提案する．

4.1 正規化優先順位

異なる OS 間では一般に異なる優先順位体系を有しており，かつ，それぞれの OS に課せられた役割自体も異なる．たとえば，多くのリアルタイム OS では 128 レベル以上の優先順位を使用することができ，きめ細かな実行順序制御を行うことができる．汎用 OS では，8 レベルないし 32 レベルの優先順位で制御を行い，スループットを重視するものが多い．また，OS の種類によって，下記のように「優先順位」と「優先順位の数値」との関連が異なっている．

- “The smaller, the higher” モデル：優先順位の数値が小さいほど，優先順位が高い．
- “The greater, the higher” モデル：優先順位の数値が大きいほど，優先順位が高い．

なお,「優先順位」と「優先順位の数値」とは以下の関係にある．

- 優先順位が高い/低い ⇔ 優先してタスクが実行される/実行されない．
- 優先順位の数値が大きい/小さい ⇔ 単に数値が大きい/小さい．

このような差違を吸収するため，本研究で開発した組込みシステム環境では，正規化優先順位（Normalized Priority）と呼ぶ概念を導入した．正規化優先順位は，OS 間で共通の優先順位体系である．各 OS における優先順位をいったん正規化優先順位に変換して比較することにより，より重要性の高いタスクを実行している OS を優先して動作させることができる．正規化優先順位には，一般のリアルタイム OS で使用されている “The smaller, the higher” モデルを採用し，−32767〜32768 という 65536 レベルの優先順位を使用することができる．ただし，実際にはこれほど多くの優先順位レベルを使用する必要はない．本研究では，下記に示す方針で正規化優先順位への割当てを決定した．

(1) リアルタイム OS と汎用 OS に対応する正規化優先順位が重ならないように定義する．

(2) 一般的に，リアルタイム OS のタスク優先順位は，128〜256 レベル程度である．このため，2 つの OS 機能を共存させるシステムにおいては，タスク用に割り当てる正規化優先順位を 512 レベル程度とする．

(3) カーナビゲーションシステムの例（図 3）に示すように，最も緊急度の高いタスクはリアルタイム OS で実行することが多い．このため，リアルタイム OS の最高優先順位が汎用 OS の最高優先順位より高くなるように設定する．

(4) 割込み処理は外部イベントに対応するためのものであり，正規化優先順位を最も高く設定する．逆に，アイドル状態，OS 停止状態として，いかなるタスクよりも低い正規化優先順位を設ける．

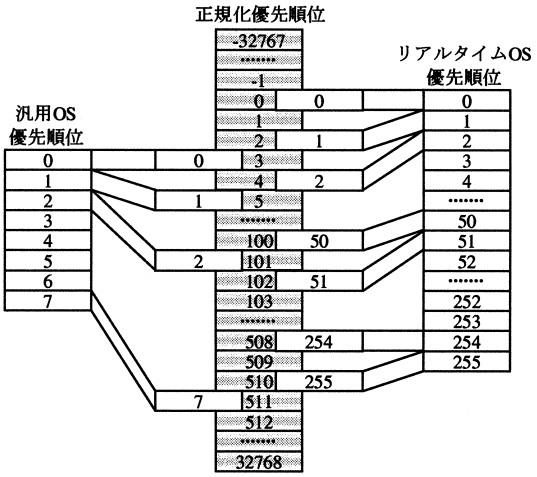

図 4 正規化優先順位
Fig. 4 Normalized priorities.

表 2 正規化優先順位へのマッピング
Table 2 Mapping to normalized priorities.

正規化優先順位	マッピング内容
−32767	割込み処理
−32766〜−3	未使用
−2	リアルタイム OS カーネル処理
−1	汎用 OS カーネル処理
0〜511	タスクの正規化優先順位
512	リアルタイム OS のアイドル状態
513	汎用 OS のアイドル状態
513〜32767	未使用
32768	OS 停止状態（両 OS 共通）

具体的には，**図 4**，**表 2** に示す方法により正規化優先順位への割当てを行う．図 4 の例では，リアルタイム OS の優先順位がレベル 0〜255（合計 256 レベル）であり，かつ，汎用 OS の優先順位がレベル 0〜7（合計 8 レベル）であるとしている．両 OS とも “The smaller, the higher” モデルに従う．リアルタイム OS の優先順位と汎用 OS の優先順位をそれぞれ，以下の計算式で正規化優先順位に割り当てる．

$$NP_{Real} = P_{Real} * 2 \quad (1)$$

$$NP_{Open} = NPA[P_{Open}] \quad (2)$$

ただし，

NP_{Real}：リアルタイム OS 用正規化優先順位

NP_{Open}：汎用 OS 用正規化優先順位

P_{Real}：リアルタイム OS での優先順位

P_{Open}：汎用 OS での優先順位

であり，かつ，正規化優先順位対応配列 NPA は下記の要素を有している．

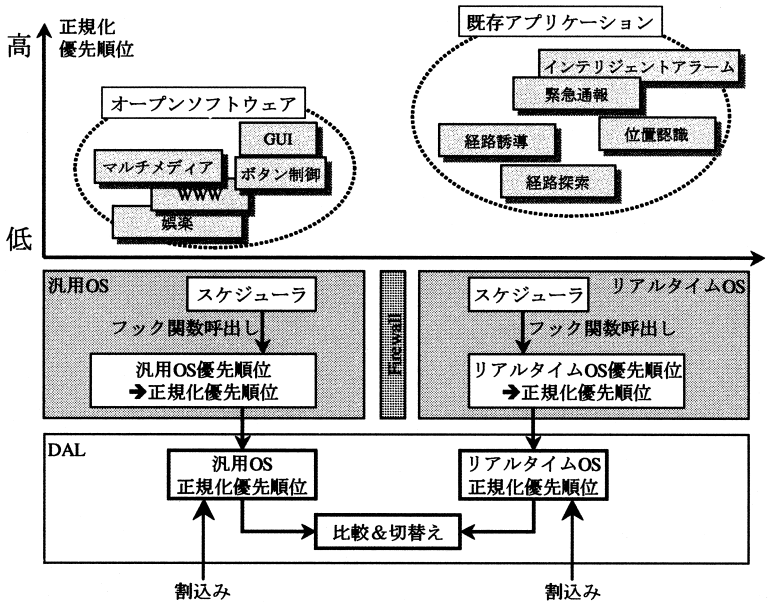

図 5 優先順位統一スケジューラ
Fig. 5 Priority unification scheduler.

$NPA[8] = \{3, 5, 101, 201, 301, 401, 501, 511\}$

リアルタイム OS は偶数の正規化優先順位を有し，汎用 OS は奇数の正規化優先順位に対応するため，OS 間で正規化優先順位が重複することはない．

表 2 に示したように，各 OS の状態として，タスク実行状態以外に，割込み処理，カーネル処理，および，アイドル状態という 3 つの状態を設けた．割込み処理に対応する正規化優先順位が最も高い．カーネル処理とは，タスクスケジューリング等の OS 内部の処理を意味する．カーネル処理状態の正規化優先順位は全タスクの正規化優先順位よりも高い．なお，OS の種類によってはカーネル処理状態であるか否かの認識を行えないものがあり，この場合には，カーネル処理状態に対応する正規化優先順位を使用しない．本研究で開発した組込みシステム環境においてもカーネル処理状態に対応する正規化優先順位を使用していない．アイドル状態はタスクがまったく動作していない状況を示す．

正規化優先順位を用いて複数の OS の優先順位を統一的に管理・比較する方法を「優先順位統一モデル」と仮称し，本方式に基づいて OS 切替えを実施するプログラムを「優先順位統一スケジューラ」と呼んでいる．

4.2 優先順位統一スケジューラ

前節で説明した正規化優先順位に基づく優先順位統一スケジューラを**図 5** に示すように構築した．リアルタイム OS 並びに汎用 OS では，OS 内でのタスクスケジューリング実行中に，次に実行すべきタスクの優先順位を検出し，それを正規化優先順位に変換して DAL に通知する．DAL は個々の OS から通知された正規化優先順位を比較し，より高い正規化優先順位を有している OS を動作させる．

たとえば，図 4，図 5 の例において，緊急通報タスク，経路探索タスクがそれぞれリアルタイム OS の優先順位 1，優先順位 51 で動作し，かつ，ボタン制御タスクが汎用 OS の優先順位 2 で動作すると仮定する．このとき，緊急通報，経路探索，ボタン制御の各タスクの正規化優先順位は，それぞれ，2，102，101 である．したがって，汎用 OS 上でボタン制御タスクが実行され，リアルタイム OS 上で緊急通報タスクが動作している場合にはリアルタイム OS が選択される．このとき，リアルタイム OS 上でタスクスケジューリングが行われ，リアルタイム OS が経路探索タスクを実行する場合を考える．リアルタイム OS 上での優先順位 51 が正規化優先順位 102 に変換され，これが DAL に通知される．汎用 OS は引き続きボタン制御タスクを実行しているため，汎用 OS の正規化優先順位がリアルタイム OS よりも高くなり，汎用 OS が選択される．このような方式によって，より正規化優先順位の高いタスクを実行している OS が動作する．複数の OS が搭載されているシステムにおいて，OS 間にわたってアプリケーションの優先順位付けを行うことができ，システム全体でのリアルタイム性保証を行うこ

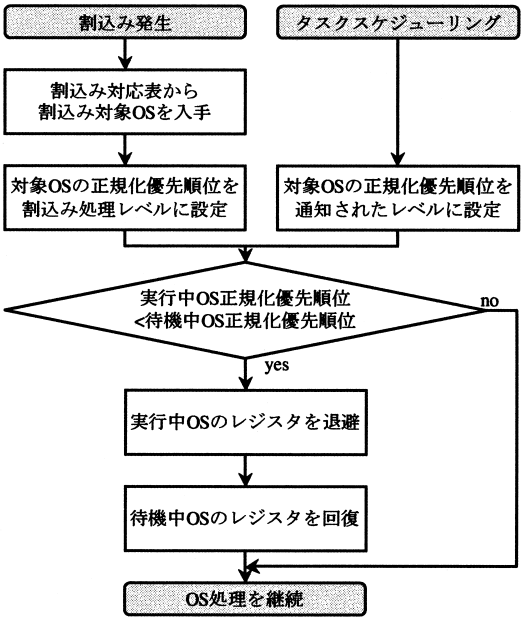

図 6 OS 切替え処理
Fig. 6 OS switch.

とが可能となる．

優先順位統一スケジューラを実装するためには，各 OS 上でのタスク切替え時に正規化優先順位を通知する必要がある．本システムでは，通常，組込み向け OS のプロファイリング機能として搭載されているフック関数機能を利用する．フック関数とは，ユーザが定義した関数が特定のタイミングで呼び出されるコールバック関数である．タスク切替え時に実行されるフック関数において，次にスケジューリングされるタスクの優先順位を取得し，これを正規化優先順位に変換して DAL に通知する．DAL は必要に応じてより高い正規化優先順位のタスクを動作させている OS に切り替えて処理を継続させる．処理が中断した OS は，あたかも，スケジューリング処理が長引いているかのような状態となる．

前節で述べたように，割込み処理状態に対応する正規化優先順位も存在する．このため，DAL は，割込みをいったん受理し，割込み対象 OS を判定したうえで，該当 OS の正規化優先順位を強制的に最上位レベルに設定する処理も行う．

4.3 OS 切替え処理

4.2 節で示した機能に基づく OS 切替え処理の内容を**図 6** に示す．開発した組込みシステム環境においては，割込み（タイマ割込みを含む），および，タスクスケジューリングのタイミングで OS 切替えが行われる．

割込み発生時には，DAL が管理している割込み割当て表に基づき，該当割込みをどちらの OS に通知するべきかを決定する．ここで，割込み対象 OS の正規化優先順位を割込み処理状態に設定する．タスクスケジューリング時には，各 OS から通知されたタスクの正規化優先順位を新たに記録する．割込みであってもタスクスケジューリングであっても，同時に 2 つ以上の OS の正規化優先順位が変化することはない．したがって，変化した正規化優先順位ともう一方の OS の正規化優先順位とを比較し，待機中 OS が実行中 OS よりも高い正規化優先順位を所有する場合，OS 切替えを実施する．なお，割込み処理状態は，通常，最も高い正規化優先順位を有しているため，割込みが発生すると，基本的には，割込み対象 OS に切り替えられて処理が行われることになる．

OS 切替えは，一方の OS が使用しているレジスタをメモリ上に退避し，実行すべき OS のレジスタを回復することで行うことができる．

5. 性能評価

組込み向け小型マイクロプロセッサ SH-4 上で本システムの性能評価を行った．SH-4 は 200 MHz クロックで動作し，ピーク性能 266 MIPS を実現するプロセッサである．ここでは，リアルタイム性を測定するため，DAL が割込みを受け付けてからリアルタイム OS の割込みハンドラが起動するまでの時間を測定した．割込み発生から割込みハンドラ起動までの処理フローは下記に示すものとなる．

(1) DAL が割込みを受理する．
(2) いずれの OS への割込みであるかを判定する．
(3) 実行中 OS と割込み対象 OS とが異なっていれば，割込み処理の正規化優先順位が高いため，割込み対象 OS に切り替える．
(4) 割込み対象 OS に制御を移行させる．
(5) OS 内で割込み要因を判定し，対応する割込みハンドラに分岐する．

上記 (1)〜(4) が DAL 内処理時間，(5) が OS 内処理時間である．2 つの処理時間をそれぞれ測定した．

測定条件は下記の 2 種類である．これらの条件下でリアルタイム OS の割込みを発生させた．

- 汎用 OS 実行中
- リアルタイム OS 実行中

汎用 OS 動作中にリアルタイム OS の割込みが発生すると，OS 切替え処理が発生する．両測定条件における DAL 内処理時間の差違は OS 切替えを行うか否かの違いである．OS 切替えを行う場合，実行中 OS の

表 3 性能評価
Table 3 Evaluation.

条件		処理時間		
負荷	OS 切替え	DAL	OS	合計
無	無	0.44 μs	0.93 μs	1.37 μs
無	有	1.40 μs	1.05 μs	2.45 μs
有	無	3.07 μs	5.22 μs	8.29 μs
有	有	9.54 μs	5.18 μs	14.72 μs

レジスタをメモリ上に退避し，割込み対象 OS のレジスタをメモリから回復しなければならない．したがって，レジスタ退避・回復にともなうメモリ参照がオーバヘッドの大部分を占めることになる．オーバヘッドの最大値を導出するため，本性能評価においては，キャッシュに負荷をかけるプログラムを動作させている．

測定結果を**表 3** に示す．なお，キャッシュ負荷あり時の処理時間はその最大値である．

割込み処理時間は，OS 切替えなしの場合，最大 3.1 μ 秒，OS 切替えありの場合，最大 9.6 μ 秒のオーバヘッド増加がある．また，OS 切替え時間は，OS 切替えなし時と OS 切替えあり時の割込み処理時間の差から導出することができる．したがって，最大 6.5 μ 秒程度の時間で OS 切替えが行われると計算できる．タスクスケジューリング中に優先順位統一スケジューラによって別 OS に切り替えられた場合でも同様の処理が行われるため，これと同程度のオーバヘッドで OS スケジューリングが実施可能である．すなわち，割込み処理およびタスクスケジューリングにおけるオーバヘッドの増加時間は 9.6 μ 秒以下，6.5 μ 秒以下と予測可能であり，この範囲でのリアルタイム性を保証できる．ディスプレイ，GPS，加速度センサといった比較的低速な入出力デバイスから構成されるカーナビゲーションシステムでは，10 μ 秒以内のオーバヘッド増加は，システムの性能・動作に大きな影響を与えるものではない．

なお，上記割込み処理時間には，各 OS が割込みを禁止している時間は含まれていない．割込み応答時間を導出するためには，このような割込み禁止時間を加算しなければならないことに注意する必要がある．

6. おわりに

リアルタイム OS と汎用 OS とを単一プロセッサ上で共存させる DARMA 技術を組込みシステム向けに適用した．本研究では，組込みシステム特有の条件である応答性・リアルタイム性保証を行うため，異なる OS 間での優先順位体系の差違を吸収し，より高い重要性を有するタスクを実行している OS を動作させる「優先順位統一モデル」を実現した．優先順位統一モデルでは，複数 OS 間にまたがって共通の体系を有する「正規化優先順位」を導入する．各 OS の優先順位をいったん正規化優先順位に変換して比較することにより，優先順位体系の異なる OS 間で，つねに最も正規化優先順位の高いタスクを実行させることが可能である．

本研究を SH-4 マイクロプロセッサ（200 MHz 動作）に適用した組込みシステム環境を実現し，割込み処理を最大 9.6 μ 秒のオーバヘッド増で，タスクスケジューリングを最大 6.5 μ 秒のオーバヘッド増で実施できることを確認した．

本研究では，SH-4 と μITRON，Windows® CE とから構成されるシステムを実現した．しかし，各 OS とのインタフェースを DPI と呼ぶインタフェースで規定しているため，これに従ったライブラリ・デバイスドライバを構築すれば，他 OS，CPU への実装も可能である．

μITRON は Micro Industrial TRON の略称です．TRON は The Realtime Operating system Nucleus の略称です．

Windows®，Win32®は米国 Microsoft Corporation の米国およびその他の国における登録商標です．Windows® CE の正式名称は，Microsoft® Windows® CE Operating System です．

その他，本稿で記載するシステム名・製品名は一般に各開発メーカの登録商標あるいは商標です．

参考文献

1) 日経 BP 社：Windows CE がリアルタイム OS と一つの機器中に共生，日経エレクトロニクス，2000 年 1 月 3 日号，No.760, pp.155–168 (2000).
2) Walters, B.: VMware Virtual Platform, *Linux Journal*, No.63 (1999).
3) Carpenter, B., et al.: The RTX Real-Time Subsystem for Windows NT, *Proc. Usenix Association Windows NT System Engineering Workshop* (1997).
4) Varhol, P.: Real-time extensions satisfy high-end RTOS requirements, *Computer Design*, pp.78–83 (June 1998).
5) Zimmerman, M.: Adapting Windows NT to Embedded Applications, *Computer Design's Electronic Systems Technology & Design*, pp.53–58 (April 1999).
6) 新井利明ほか：異種 OS 共存技術「DARMA」の開発と制御システムへの適用，計測技術，Vol.27, No.07, pp.39–44 (1999).

7) 日経 BP 社：Windows CE を ITRON と統合してリアルタイム化，日経エレクトロニクス，1999 年 9 月 20 日号，No.752, pp.176–177 (1999).
8) Barbanov, M. and Yodaiken, V.: Introducing Real-time Linux, *Linux Journal*, No.34, pp.19–23 (1997).
9) 株式会社日立製作所：SH7750 シリーズハードウェアマニュアル.
10) Timmerman, M., et al.: Windows NT Real-Time Extensions better or worse?, *Real-Time Magazine*, 3Q98, pp.11–18 (1998).
11) Obenland, K., et al.: Comparing the Real-Time Performance of Windows NT to an NT Real-Time Extension, *Proc. 5th IEEE Real-Time Technology and Applications Symposium* (1999).
12) Obenland, K. and Rosen, L.: The Performance Trade-Offs of Implementing a Large Scale Real-Time Application Using the Windows NT Operating System, *Proc. 6th IEEE Real-Time Technology and Applications Symposium* (2000).
13) 齊藤雅彦ほか：組込み向けデュアル OS 実行システム DARMA の開発，情報処理学会第 59 回全国大会特別セッション講演論文集，4B–03 (1999).
14) 奥出真理子ほか：ITS における車載情報システムの検討，情報処理学会論文誌，Vol.42, No.7, pp.1736–1743 (2001).

(平成 13 年 4 月 4 日受付)
(平成 14 年 3 月 14 日採録)

齊藤　雅彦（正会員）

昭和 39 年生．昭和 63 年京都大学大学院工学研究科情報工学専攻修士課程修了．同年株式会社日立製作所日立研究所入社．並列/分散処理システムの研究開発に従事．近年，主としてリアルタイム OS の実装に関する研究に従事．IEEE，電子情報通信学会各会員．

加藤　　直

昭和 44 年生．平成 5 年千葉大学大学院理学部物理学専攻修士課程修了．同年株式会社日立製作所入社．以後，情報制御システムおよび組込み機器向けのソフトウェア開発を担当．

大野　　洋

昭和 45 年生．平成 7 年東京大学大学院工学系研究科電気工学専攻修士課程修了．同年株式会社日立製作所入社．以後，情報制御システムおよび組込み機器向けのソフトウェア開発を担当．現在，同社 RAID システム事業部にて，ディスクアレイサブシステムの製品企画に従事．IEEE 会員．

井上　太郎（正会員）

昭和 36 年生．昭和 61 年京都工芸繊維大学大学院工芸学研究科生産機械工学専攻修士課程修了．同年株式会社日立製作所システム開発研究所入社．計算機システムの研究開発に従事．近年，主として計算機運用管理システムに関する研究に従事．IEEE 会員．

上脇　　正

昭和 37 年生．昭和 62 年東京工業大学大学院工学部情報工学専攻修士課程修了．同年株式会社日立製作所日立研究所入社．車載情報システム，情報制御システムの研究開発に従事．近年，車載端末，テレマティクスシステムの実装に関する研究に従事．SAE 会員．

中村　智明（正会員）

昭和 30 年生．昭和 54 年東京大学大学院原子力工学専攻修士課程修了．同年株式会社日立製作所に入社し，情報制御システム向けリアルタイム OS の製品開発を担当．昭和 60 年米国 Stanford 大学計算機科学科修士課程修了．近年，SH プロセッサを応用したカーナビゲーション，テレマティクス分野向けのシステム LSI 製品の開発，マーケティングに従事．IEEE，ACM 各会員．