システムソフトウェアと
オペレーティング・システム 84－7
（２０００．５．２５）

組込み用 OS『開聞』の割込み管理機構

萱嶋 志門　　並木 美太郎

東京農工大学大学院工学研究科

組込みシステムのアプリケーションは，自身の大規模化や複雑化によって生産性が低下する．生産性を向上させるための一つの方法として，OS の使用がある．しかしながら，組込みシステムでは応答性を重視するので，応答速度を維持しながら OS を使う仕組みが必要である．この問題を解決するため，組込み用 OS『開聞』を開発した．『開聞』では，割込み処理を３段階に分割している．ユーザはこの段階を選択することで，割込み処理での内容とオーバヘッドの大きさを選択することができる．本論文では，『開聞』における特徴的な機能である割込み処理について述べる．

Interrupt Management Mechanism in Embedded OS KAIMON

Shimon Kayashima　Mitaro Namiki

Graduate School of Technology,
Tokyo University of Agriculture and Technology

In embedded systems, the productivity of application software becomes low, as they are more complex and larger scales. OSes are needed to improve the productivity. However, overheads of OSes must be reduced because responsibility is very important in embedded systems. To solve these problems, we designed and developed the embedded operating system KAIMON. A process for interrupt handling is divided into 3 levels in the KAIMON. The function and the overheads of each level are different. Users can select the level as their needs. In this paper, interrupt management mechanism, which is characteristic function of KAIMON, is described.

1. はじめに

現在，ディジタルテレビや携帯電話に代表されるマルチメディア系の組込みシステムや，ネットワーク上の各種サーバなどの組込みシステムが登場している．これらの組込みシステムでは，ソフトウェアの大規模化や複雑化から OS を使用する傾向がある．しかし，組込みシステムでは OS を用いたときのオーバヘッドが問題となる．OS を用いることで想定した時間以内に処理を終了できないことや，想定した以上のメモリを消費することがありうる．OS が行うコンテキスト保存，復元などオーバヘッドを伴う要因だけを，プログラマの目的に特化した処理に置き換えることができれば，応答速度を維持することが可能である．

本研究で作成した『開聞』は割込み処理の応答性に着目し，OS で行うコンテキスト保存，復元や割込みハンドラ自体の処理を，プログラマの作成した処理へ置き換えることを可能にした．本稿ではこのような機能を実現する割込み管理機構を中心に述べる．以下，2 章で『開聞』の概要，3 章でタスク管理機構，4 章で割込み管理機構，5 章で評価を述べる．

2. 『開聞』の設計

2.1 組込み用 OS への要求

組込みシステムの性質は汎用コンピュータとは大きく異なる．それに応じて OS への要求事項も異なる．次の (1)，(2) で，組込み用 OS への要求事項を示す．また (3) では，近年の組込みシステムの動向から OS に対する要求事項を示す．

(1) 安価なハードウェアへの適応性があること

組込みシステムでは製品コストの上昇を嫌う．つまり，各組込み機器の製造価格を切り詰めるため安価なハードウェアが使用される．パーソナルコンピュータと比べると，特に CPU とメモリでの制約が著しい．このことから，安価なハードウェアへの適応性から OS に要求されるのは，カーネルサイズが小さいこと，無駄な処理を行わずオーバヘッドを小さくすることの 2 点である．サイズについては市販されている組込み用 OS の大半で最小構成が 1K バイトである[1].

(2) ハードリアルタイム性を保てること

組込みシステムでは，外部からのデータを処理して，その結果を一定時間以内に外部へ送信するリアルタイム処理を行う．そのためには実行時間を見積もる必要がある．つまり，OS の各種処理に要する時間を見積もれる必要がある．

処理を行う一定時間とは数 μs から数 ms まであり必ずしも短いわけではない．しかしより安価な CPU を利用する要求や高速な応答が必要なときのことを考慮すれば，ハードリアルタイム性を保つ必要性からも，OS のオーバヘッドは小さい必要がある．

(3) 組込みシステムのマルチメディアやネットワークへの用途拡大に対応すること

組込みシステムでは，その用途の拡大からマルチメディアモジュールやネットワークプロトコルなどの実装が行われている．このような大規模，複雑なソフトウェアを利用する必要から，組込みシステムではソフトウェアの再利用性や保守性を維持する目的で OS を利用する．

2.2 設計方針

これらの要求を満たすことを目標に『開聞』の設計方針として，次のことを定めた．また OS のユーザとしてはエンドユーザを考慮せず，OS 上で組込みシステムを開発するプログラマを対象とする．以下，システムを開発するプログラマのことを単にプログラマと呼ぶ．

(1) 必要最低限の機能だけを実装する

必要最低限の機能だけを実装することは，カーネルサイズを小さくすることにつながり，さらにカーネルの持つ機能が少なければカーネル内への処理移行を減らすことが可能である．つまりサイズと速度の両面でオーバヘッドを削減することができる．しかし，これは機能を限定し，汎用性の低い処理はプログラマが実装することになる．必要最低限と考えた機能は次の 2 点である．

(a) タスク管理機能

マルチタスクの概念を導入することで，並行処理を行うプログラムを容易に作成することができ

る．タスク管理は，他タスクとの相互作用を考えると，カーネル内で実行する必要がある．マルチタスクの利便性を向上させるには，自（他）タスクの処理を中断，再開させる機能が必要である．そこで，『開聞』ではタスク間同期機能を含めたタスク管理機能を設計した．

(b) 割込み管理

ハードウェアの割込みベクタテーブルは，アーキテクチャごとに異なる．ソフトウェアの再利用性を求めるならば，OS が独自にベクタテーブルを用意する必要がある．割込み管理は，OS が用意した仮想的な割込み処理用のベクタテーブルを管理することで行う．

(2) カーネルでの各種処理時間を短くする

OS の機能を使用することでプログラミングが容易になるが，性能が悪いとシステムの想定する時間以内に応答できない．スケジューリングやシステムコールの処理時間は短くするような設計を行う．

このとき，メモリを無駄に消費するアルゴリズムであっても，処理速度を優先して採用した．これはソフトウェア規模が増大しているため，カーネルのサイズが多少増えたとしても，システム全体への影響は速度を犠牲にした場合より少ないと考えたからである．

(3) カーネルでの処理時間を見積もれる

リアルタイムシステムを構築するために処理時間を見積もらなくてはならない．したがって，OS の各種処理時間を計測する必要がある．しかし，パイプラインの乱れやキャッシュミスなどの影響で処理時間を厳密に見積もるのは難しい．今回は一定命令数以内に処理を終わらせることを設計方針にした．

(4) ユーザプログラムでのハードウェアの操作も許可する

ハードリアルタイムシステムを構築するときは，OS のオーバヘッドによって想定していた時間以内に終了しない処理が一つでもあれば，システムの再設計を行う必要がある．そこで，プログラマの責任でデバイスの持つレジスタや CPU の持つ割込みベクタテーブルへのアクセスを許可した．これにより，高速な応答が求められるときにカーネル内への処理移行時に生じるオーバヘッドを削減することが可能となる．

2.3 『開聞』の全体構成

『開聞』を用いたシステムの全体構成を図 1 に示す．カーネルはタスク管理部と割込み管理部からなる．カーネルへのインタフェースとして，システムコールを設計した．またハードウェアへのインタフェースとしてライブラリ関数を設計した．

(1) タスク管理部

タスクの実行，中断を制御し，そのために必要なデータを保持する．またタスクの実行順序を決めるため，スケジューリングを行う．

(2) 割込み管理部

割込みベクタテーブルを管理する．

(3) システムコール

カーネル内部の情報や CPU の持つ特権命令，資源を利用するときに使用する．

(4) ライブラリ関数

ユーザプログラムからレジスタにカーネルを介さず直接アクセスすることで，高速な処理を行うプログラミングを可能にする．

3. タスク管理部

『開聞』でのタスクは並行処理の基本単位である．並行処理はタスクを複数作成して行う．『開聞』は，タスクコンテキストをカーネル空間に置くことによ

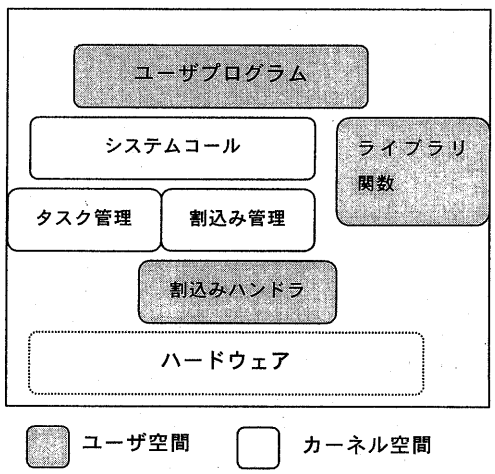

図 1 『開聞』を用いたシステムの全体構成

り，それぞれのタスクがカーネルを介さず相互作用しないことを保証する．

タスク管理部で行うことはスケジューリングと各タスクの生成，削除，実行，中断の制御である．これらの制御には，タスクコンテキストの情報を用いる．タスクコンテキストの内容を表 1 に示す．

最高優先度と最大生成タスク数はシステム生成時に静的に決定する．タスクの情報を格納するメモリ領域をタスク生成時に取得すると実行時間が予測不可能となる原因となる．そこで，『開聞』のシステム起動時に，タスクが最大生成タスク数システム上に生成されたときに必要なメモリを静的に確保する．タスク管理で用いるシステムコールの一覧を表 2 に示す．

3.1 タスクの状態とスケジューリング

タスク管理部はタスクを状態ごとに待ち行列に格納して管理する．タスクの状態遷移図を図 2 に示す．『開聞』ではタスクの状態として次の四つの状態を定めた．

(1) 実行状態

CPU に割り当てられ処理を実行しているタスクの状態である．

(2) 実行可能状態

実行可能状態となっているタスク．スケジューラで CPU へ割り当てられることを待っている状態である．

(3) 強制中断状態

サスペンドやセマフォで待ち状態となっているタスク．複数の要因での待ち状態も可能である．

表 1 タスクの持つ属性

名称	意味
優先度	スケジューリングのときに参照
状態	タスクの状態
カウンタ	サスペンドカウンタとして使用
コンテキスト	レジスタなどの値

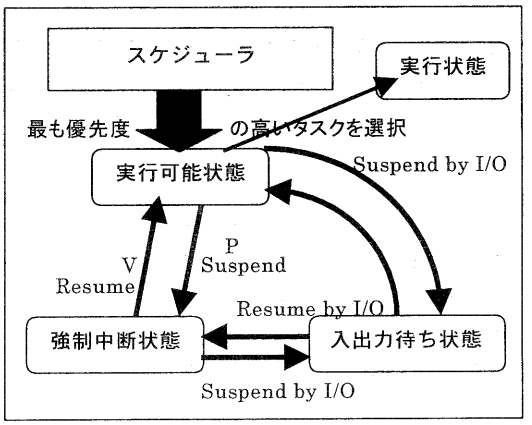

図 2 タスクの状態遷移図

表 2 タスク管理に用いるシステムコール

名称	引数	処理
CreateTask	タスクのエントリポインタ 優先度 スタックサイズ タスクへ渡す文字列	引数にしたがって，タスクを生成する
KillTask	タスク ID	タスクを消滅させる タスクの持っている資源の解放は行わない
Suspend	タスク ID	サスペンドカウンタを減らす． サスペンドカウンタが 0 以下で，強制中断状態
Resume	タスク ID	サスペンドカウンタを増やす． サスペンドカウンタが 1 以上で，準備完了状態
SuspendByI/O	タスク ID	入出力待ちのため中断させる
ResumeByI/O	タスク ID	入出力待ち解除
P	セマフォ ID	セマフォの P 命令
V	セマフォ ID	セマフォの V 命令
InitSemCnt	セマフォ ID 初期カウンタ値	セマフォカウンタの初期値の設定

(4) 入出力待ち状態

外部デバイスからの入出力を待っているタスク.

スケジューラは実行可能状態のタスクから最も優先度の高いタスクを選択して CPU に割り当てる. 最も優先度の高いタスクが複数存在する場合は, はじめに実行可能状態となったタスクを CPU に割り当てる. システムコールを期にスケジューリングを行うが, これは優先度を変更するには, システムコールを発行する他に手段がないからである.

3.3 タスク間同期

タスク間同期として自（他）タスクの中断, 再開の制御と, 資源占有のためのセマフォを備える. セマフォの最大個数はシステム構築時に静的に決定する.

4. 割込み管理部

一般の OS では, 一部の割込み処理を OS で行うことで, プログラマがハードウェアを意識せずにプログラミングすることを可能にする. 具体的な OS で行う一部の割込み処理とはコンテキスト保存, 復元と割込みを引き起こした要因の解析処理である. 一般の OS の割込み発生時の処理の流れを図 3 に示す.

次の項目については, OS 作成時に知ることができなくても, プログラマは知ることが可能である.

(1) 各タスクや各種割込み処理で使用するレジスタの数

OS 作成時にはこのようなレジスタの数を知ることができない. したがって, コンテキスト保存処理を OS が保証するときは, すべてのレジスタを保存する必要がある.

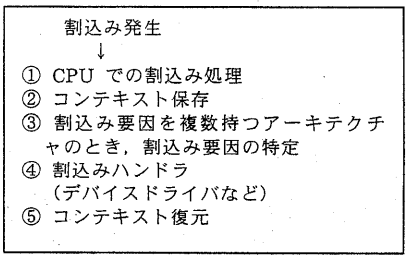

図 3 通常の割込み処理

(2) システムが使用しない割込み要因

ターゲット CPU のアーキテクチャによっては, 一つの割込みに対して複数の要因が割り当てられる. 例えば外部割込みに対して複数のデバイスが割り当てられているときである. このようなアーキテクチャでは, 割込みが発生した要因を解析する必要があるが, システムが使用しない割込み要因の解析処理は無駄である. しかしシステムが使用しない割込み要因を OS 作成時に知ることはできないため, OS では可能性のある割込み要因をすべて解析する必要がある.

RISC CPU はレジスタの数が多いため, コンテキスト保存, 復元で全汎用レジスタを操作すると実行効率が落ちる. そのためコンテキスト保存, 復元や割込み要因の解析処理を OS で固定処理として行うと, 汎用性を考慮しているのでオーバヘッドが大きくなる. 場合によっては性能を向上させるために移植性を落としてでも高速な応答が期待できる割込みハンドラを使用する必要がある. そこで, このような処理は OS で行わずにプログラマが作成したものを使用できる機構を提供する. 『開聞』では, OS が行う割込み処理のレベルを次の三つに分け, プログラマが必要に応じて選択することが可能である.

(a) 割込みを直接ユーザプログラムで処理する

ハードウェアの持つ割込みベクタテーブルにユーザプログラムを登録することで, 割込み発生後すぐにユーザプログラムへ処理を移行させることが可能である. ただしこの場合は, プログラマの責任でコンテキスト保存や割込み要因の特定といったハードウェアに依存した処理を実装する必要がある.

この方法は割込み前後でタスクスイッチを引き起こさず割込みハンドラでの処理に多くのレジスタを使わない場合に有効である. 有効な処理例として外部割込みでデータをメモリに保存することだけを行う割込みハンドラがある.

(b) カーネルで, コンテキスト保存だけ行う

コンテキスト保存後にユーザプログラムへ処理を移行させる. ただしこの場合は, 割込み要因の特定を行う処理をユーザプログラムで実装する必要があ

る．

この方法は，割込み前後にタスクスイッチがあるとき，すなわちコンテキストを全て保存する必要があり，割込み要因が限られているときに有効である．有効な処理例として，事前に使用するデバイスを限定することが可能なときで，外部割込みを契機にタスクを起動させる割込みハンドラがある．

(c) カーネルで，コンテキスト保存，割込み要因の特定を行う

OS が定めた割込み要因を利用できることから，プログラマはハードウェアを意識することなくプログラムが可能である．ただし，コンテキスト保存や割込み要因特定を行うのでオーバヘッドが大きくなる．プログラムの実行速度を求めない場合には有効である．

割込み処理で OS が行う処理について，それぞれに対応したベクタテーブルを用意することで実装した．このベクタテーブルを用いて割込み処理の流れを管理する．表 3 にそれぞれのベクタテーブル例を示す．この例のハードウェアは，PowerPC403GA アーキテクチャ[2]である．

割込み管理部は，このようなベクタテーブルへの登録と表 4 に示す内容の保証を行う．割込み管理部へアクセスするシステムコールを表 5 に示す．プログラマはどの割込みがどのテーブルへ割り付けられているのか把握している必要がある．

表 3 各方式ごとのベクタテーブルの内容
(a)，(b)は，PowerPC403GA での例

方式	ベクタテーブルの内容
(a) (b)	外部割込み，システムコール割込み，プログラム例外，タイマ割込みなど
(c)	COM1，COM2，パラレル，IDE，キーボードなど

表 4 各方式ごとの保証内容

方式	保証内容
(a)	OS としては何も行わない
(b)	コンテキストの保存，復元
(c)	コンテキストの保存，復元 『開聞』の定めた割込み ID の使用

5. 実現と評価

『開聞』の実現には CQ 出版から販売されている CQ-RISC 評価用ボード PowerPC403 を用いた．表 6 に実装環境を示す．次に評価方法を述べ，評価結果を表 7 以降に示す．

(1) カーネルサイズ

カーネルサイズは実際に計測したのではなく，静

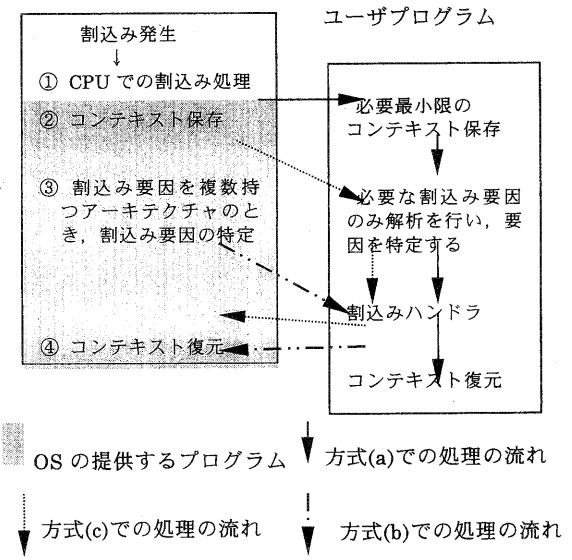

図 4『開聞』での割込み処理

表 5 割込み管理に用いるシステムコール

名称	引数	処理
ChIntVec	割込み ID エントリポインタ	割込み ID で，該当するベクタテーブルにエントリポインタを格納する
ChCpuIntVec	割込み ID エントリポインタ コンテキストの前後どちらか	割込み ID と，コンテキスト前後のどちらかを指定するフラグを参照して，該当するベクタテーブルにエントリポインタを格納する

的なサイズである．計測した条件を次に示す．

・最大生成タスク数 1
・最高優先度 1
・最大セマフォ数 1

また，最大生成タスク数一つあたりのサイズも計測した．表7にそれぞれの値を示す．

(2) タスクスイッチに要する時間

Null システムコールを発行して各優先度ごとに計測した．コンテキスト保存，復元とスケジューリングを行う時間が含まれている．すべてのレジスタを保存したときの計測結果を表8に示す．

(3) 割込み発生時にカーネル内で要する時間

割込み発生から割込みハンドラに処理が移行するまでの時間を各方式ごとに計測した．割込みが生じて最初に実行する命令で時間を取得し，ユーザ定義の割込みハンドラの最初で時間を取得して計測した．表8にその結果を示す．

コンテキストを保存するだけにほとんどの時間を費やしている．今回の実装では，汎用レジスタ 32 個と専用レジスタ6個を保存した．

(4) 各システムコールに要する時間

システムコールを発行する前後に時間を計測するインラインアセンブラ関数を配置して計測した．表9に計測結果を示す．キャッシュミスやパイプラインの乱れから計測値に数μs（数 10〜200 サイクル）の幅がある．またサスペンドやレジューム，セマフォ関連のシステムコールはタスクの状態遷移を引き起こす場合に待ち行列を操作するので，そうでない場合との間で 5μs から 10μs 近くの差が生じている．

6. 関連研究

関連研究としては，組込み用 OS の仕様としてμITRON[3] がある．μITRON 仕様は，仕様をゆるくすることで，幅広い分野に適応することを目標に策定された．そのため現在では，多くの組込み用 OS で採用されている．割込み関連の機能は，『開聞』と比べても多機能である．しかし，仕様レベルでは，割込み処理時のオーバヘッドを削減する手段はない．

表6 『開聞』の実装

ターゲット CPU	PowerPC403GA
ターゲットボード	評価用 RISC ボード PPC403 シリアルポート， パラレルポート， 10BaseT コネクタ IDE コネクタなど搭載
コンパイラ	HighC/C++
システムコール数	12
プログラム行数（言語 C）	1335 行
プログラム行数（アセンブラ）	267 行

表7 カーネルサイズ

スタートアップルーチン	1360 バイト
テキスト領域	4952 バイト
データ領域	528 バイト
BSS 領域	424 バイト
合計（スタートアップを除く）	7264 バイト
タスク一つあたりのサイズ	(78+stack size) バイト

表8 各種オーバヘッドの結果

タスクスイッチ所要時間（コンテキスト保存）	(10 + 4×Priority) μs (5μs)
割込み処理時間 (a) まで	計測不能（1命令）
割込み処理時間 (b) まで	5μs（61命令）
割込み処理時間 (c) まで	8μs（88命令）

表9 各種システムコールの処理時間

名称	測定時間（最小値／最大値）
Null	7μs／7μs
CreateTask	18μs／24μs
KillTask	22μs／22μs
Suspend	13μs／20μs
Resume	11μs／20μs
SuspendByIO	18μs／20μs
ResumeByIO	17μs／19μs
P	14μs／18μs
V	14μs／18μs
InitSemCnt	10μs／10μs
ChIntVec	12μs／13μs
ChCpuIntVec	12μs／13μs

OS のオーバヘッドを削減する方法として，OS をLSI化するアプローチがある．オンチップリアルタイム OS STRON-I [4] では，μITRON 仕様をターゲットとしており，システムコール発行時のオーバヘッドが μs オーダ未満になることを達成している．しかし，割込み処理については触れられていない．

7. おわりに

本論文では，『開聞』の設計を割込み管理を中心に述べた．本研究では基本的な機能を備えた組込み用OS『開聞』を実現した．『開聞』では各種オーバヘッドを少なくした．また複数の割込みベクタテーブルを用意することで，OS を用いていても高速な割込み応答を行える枠組みを実現した．

今後の課題として，『開聞』の PowerPC アーキテクチャ以外の CPU への移植と『開聞』上にアプリケーションを構築することが挙げられる．しかし，今回の方式では，高速性を求めるためにハードウェアを意識せざるを得なく，アプリケーションの構築に支障をきたす可能性がある．そこで，ソフトウェアの再利用性や保守性を維持するための開発支援環境を作成することが必要である．

参考文献

[1] Special Report, Embedded System Programming, Mar. 1999

[2] PowerPC 403GA User's Manual Second Edition, IBM, Mar. 1995

[3] http://www.itron.gr.jp/home-j.html

[4] リアルタイム OS の VLSI 化とその評価，仲野 巧，アンディ ウタマ 他，電子情報通信学会論文誌 D-I Vol.J78-DI No.8 pp679-686, Aug.1995