社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－ＳＬＤＭ－114 （８）
2004／3／19

LinuxとITRONによるハイブリッドOSの設計と実装

保田 信長† 飯山 真一† 冨山 宏之†† 高田 広章†† 中島 浩†††

†,††† 豊橋技術科学大学 情報工学系 〒441-8580 愛知県豊橋市雲雀ヶ丘1-1
†† 名古屋大学大学院 情報科学研究科 〒464-8603 愛知県名古屋市千種区不老町
E-mail: †,††{nobunaga,shin,hiro,tomiyama}@ertl.jp,†††nakasima@tutics.tut.ac.jp

あらまし 本稿では一時的にLinuxの割込み処理を優先することが可能であるハイブリッドOSの設計および実装について述べる．本手法ではLinuxの割込み処理をITRONのタスク例外処理機能として仮想化することで，ITRONのリアルタイム性を保っている．実装を行ったシステムに対し，割込み禁止区間の比較，ハイブリッド化による割込みおよびディスパッチ時間のオーバヘッド，ITRONカーネルの起動時間，Linuxの割込み遅延，カーネル改造量について評価した．

キーワード Linux，ITRON，ハイブリッドOS，タスク例外処理

Design and implementation of a Linux/ITRON hybrid operating system

Nobunaga YASUDA†, Shinichi IIYAMA†, Hiroyuki TOMIYAMA††, Hiroaki TAKADA††, and Hiroshi NAKASHIMA†††

†,††† Department of Information and Computer Sciences, Toyohasi University of Technology
Hibarigaoka 1-1, Tempaku-cho, Toyohashi-shi, Aichi, 441-8580 Japan
†† Graduate School of Information Science, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, Aichi, 464-8603 Japan
E-mail: †,††{nobunaga,shin,hiro,tomiyama}@ertl.jp,†††nakasima@tutics.tut.ac.jp

Abstract This paper describes design and implementation of a hybrid Linux/ITRON operating system. An interesting feature of the hybrid OS is that it can temporarily change priorities of certain Linux interrupts higher than those of ITRON tasks with maintaining the real-time constraints of ITRON. This is done by treating the Linux interrupts as *Task Exception Handling Functions* on ITRON. The hybrid OS has been evaluated in terms of interrupt and dispatch time, interrupt disable region size, and so on.

Key words Linux, ITRON, hybrid OS, *Task Exception Handling Functions*

1. はじめに

近年，組込みシステムは高い付加価値が求められており，多機能化・複雑化している．また，他社よりもいち早く競争力の高い製品を投入するため開発期間の短縮化が求められる一方で，高い信頼性，安定性は必要不可欠である．このような状況において，ソフトウェアの開発効率の向上が求められる．

一般的に，高機能な組込みシステムにはOSが使用される．日本における組込みシステム用OSとしては，μITRON仕様OS（以下，ITRON）が事実上の標準となっている[1]．ITRONは高いリアルタイム性を重視したOSであり，その上で動作するリアルタイム処理を行うソフトウェア資産は豊富に存在する．その反面，ミドルウェアが充実しているとは言い難い．一方，最近では，PC用途として開発され広く普及しているLinuxを組込みシステム用OSとして採用する例も増えている[2]．この主な理由は，Linuxの持つ豊富なミドルウェア資産を利用して，効率的なソフトウェア開発を行うためである．しかし，Linuxは高いスループットを重視したOSであり，組込みシステムで求められるレベルのリアルタイム性を持ち合わせていない．そこで，双方の特長を生かすことのできる，ITRONとLinuxとのハイブリッドOSを実現することで，信頼性の高いシステムを，より効率的に開発することが可能となる．

既存のハイブリッドOSでは，ITRONのリアルタイム性を保つために，ITRONタスク実行中はLinuxへの割込みを全て禁止している．しかしながら，アプリケーションによっては，ITRONタスクよりも優先されるLinuxへの割込みが存在する

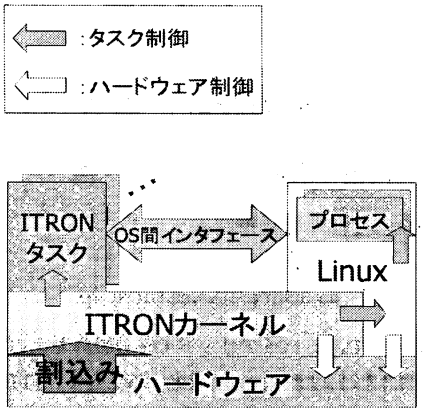

図1　システム構成
Fig. 1　System construction

ことも考えられる．

そこで我々は，ITRON の高いリアルタイム性を保ちながら，一時的にでも，ITRON タスクより Linux への割込み処理を優先して実行可能な構造を持つハイブリッド OS の設計および実装手法を提案する．本手法では，Linux の割込みを ITRON の持つタスク例外処理機能により仮想化する．これにより，システムの要件に合わせてより柔軟に対応可能なハイブリッド OS が実現され，その適用範囲が広がり，ソフトウェア開発の効率をより高めることができる．

本稿では，2 章においてハイブリッド化のアプローチを述べる．3 章において実装手法を述べ，4 章で実現したシステムについて評価を行う．

2.　ハイブリッド化のアプローチ

本章ではハイブリッド化のアプローチを示す．なお，設計および実装を行ううえで，Linux カーネルのバージョンアップに追従するため，Linux カーネルへの改造を最小限に抑えることを目標とする．

2.1　システムの構成

システムの構成を図 1に示す．

本システムにおいて，Linux は ITRON の 1 つのタスクである Linux タスクとして動作し，基本的に最低優先度でスケジューリングされる．これにより，Linux の処理が ITRON のタスクを中断することを防ぎ，リアルタイム性を保つ．また，Linux と ITRON とで，独立したデバイスを制御対象とするシステムモデルを想定することから，OS 間でのデバイスの共有は行わないこととする．本稿では，Linux が制御対象とするデバイスからの割込みを Linux 割込み，ITRON が制御対象とするデバイスからの割込みを ITRON 割込みと呼ぶ．全ての割込みは一旦 ITRON 側で受け取り，Linux 割込みであれば Linux 側へ回送し処理する．システムの起動順序は，Linux が起動し，その初期化処理の中で ITRON を起動することとする．ITRON タスクと Linux プロセス間のデータ通信は OS 間インタフェースにより行う．なお，Linux タスクは，Linux カーネル内のみで ITRON のシステムコールを発行可能とする．

2.2　処理の優先順位

ITRON の処理を Linux の処理より優先して実行することでリアルタイム性を保つことが可能となる．優先順位の実現のためには，ITRON タスク実行中に Linux 割込み処理を行わない構造が必要となる．基本の優先順位は以下に示す順である．

（1）　ITRON 割込み処理
（2）　ITRON タスク処理
（3）　Linux 割込み処理
（4）　Linux プロセス処理

従来のハイブリッド OS は基本の優先順位に固定され，動作していた．しかしながら，この優先順位では，例えリアルタイム制約の厳しくない ITRON タスクが存在する場合でも，Linux 割込み処理を優先することができない．そこで，一部の Linux 割込み処理を優先して実行可能とするための優先順位を以下に示す．ここで，リアルタイム制約の厳しくない ITRON タスクを ITRON 低優先度タスク，それより優先して処理したい Linux 割込みを Linux 高優先度割込み，それ以外の ITRON タスクと Linux 割込みとを，それぞれ ITRON 高優先度タスク，Linux 低優先度割込みとする．

（1）　ITRON 割込み処理
（2）　ITRON 高優先度タスク処理
（3）　Linux 高優先度割込み処理
（4）　ITRON 低優先度タスク処理
（5）　Linux 低優先度割込み処理
（6）　Linux プロセス処理

上記の優先順位は，Linux タスクの動的な優先度変更のみで実現可能である．この優先順位の実現により，従来は不可能であった，Linux 割込み処理の優先が可能となる．

2.3　Linux 割込みの取り扱い

Linux 割込み処理は，ITRON タスク実行中には行わない構造とするため，ITRON のタスク例外処理機能により仮想化し，Linux を ITRON の 1 つのタスクとして実現する．タスク例外処理はタスクに対して例外を発生させる機能である [1] [3]．Linux 割込みの発生により，Linux タスクに対してタスク例外処理を要求し，その処理ルーチンで Linux 割込み処理を行う．これにより，Linux タスクの視点からは割込み処理と同等の機能を提供していると言える．

また，通常，Linux の割込み処理は割込み禁止状態で行われることから，その間は他の割込み処理を行うことができない．これは，割込み即応性を低下させ，リアルタイム性を損なう原因となる．しかし Linux 割込み処理をタスク例外処理として仮想化することで，Linux 割込み禁止/許可を，タスク例外処理禁止/許可に変更可能となり，Linux 割込み処理中であっても ITRON 割込み処理が可能となる．これにより，割込み即応性が向上し，リアルタイム性を保つことができる．

3.　実　　装

本章では，ハイブリッド OS の実装手法について示す．

実装は，ターゲットハードウェアに日立 SH-3 を搭載した Solu-

tion Engine，Linux カーネルに kernel-2.4.18，ITRON カーネルに TOPPERS/JSP Release1.3 を用いて行った [4] [5] [6] [7]．ここで示す実装手法において，割込みおよびディスパッチはプロセッサに依存する処理であるため，SH-3 に限定した処理である．他のプロセッサに適用する場合，少なくとも割込み要因ごとにマスク可能である必要がある．

3.1 OS 間切替え

本節では ITRON と Linux との OS 間切替えについて述べる．

3.1.1 Linux 実行状態の判定

本稿では，システムにおける Linux の実行状態を示すため，Linux 割込み処理および Linux プロセス処理実行時を Linux 実行状態とし，それ以外を Linux 停止状態とする．

3.1.2 割込み発生時の切替え

Linux 実行状態時に割込みが発生すると，ITRON 割込み処理へと移行するため，Linux 停止状態とする．

SH-3 では，割込みハンドラ起動直後の状態において，仮想メモリ空間にアクセスすることができない．TOPPERS/JSP は物理メモリ空間上での動作を前提としており，全てのタスクスタック領域はカーネルコンフィギュレーション時に物理メモリ空間上に確保される．そのため，TOPPERS/JSP は割込みハンドラ起動直後に，実行中タスクのスタックにコンテキストの保存を行うことが可能である．しかし，Linux プロセスは仮想メモリ空間上で動作し，使用するユーザスタックも仮想メモリ空間上に存在する．よって，Linux プロセス実行時に割込みが発生した場合，本システムでは最初に物理メモリ空間上に存在するスタックへ変更する必要がある．このためのスタックとして，ITRON 側で用意した Linux タスク用のスタック（以下，Linux タスクスタック）を使用する．

割込み発生時のスタックポインタとプロセッサモードを表すレジスタは，タスク例外処理による Linux 割込み処理時に必要となる．これらを，Linux 割込み処理用コンテキストと呼ぶ．プロセッサモードを表すレジスタが必要となるのは，Linux の割込みハンドラ内で，Linux ユーザスタックから Linux カーネルスタックへの変更を行うためである．このコンテキストを，Linux タスクスタックへ保存すると，タスク例外処理時にスタック内のどの位置にあるか判別が困難であるため，これとは異なるスタックへ保存する必要がある．これを Linux 割込みスタックと呼ぶ．

以上を元に，割込み発生時における OS 間切替えを図 2に示す．

割込み出口処理において，Linux 停止状態から Linux 実行状態へ移行する場合，保存していたコンテキストを復帰して Linux の処理を再開する．

3.1.3 ディスパッチ時の切替え

ITRON のシステムコールを Linux カーネル内で発行しディスパッチが発生すると，ITRON タスク処理へと移行するため，Linux 停止状態とする．

TOPPERS/JSP ではディスパッチが要求されると，タスクスタックにコンテキストを保存し，そのスタックポインタをタスクのタスクコントロールブロック（以下，TCB）に保存する．Linux カーネル内からディスパッチが発生した場合に Linux カーネルスタックのスタックポインタが Linux タスクの TCB に保存されると，今後，Linux タスクスタックが使用不可能となる．よって，一度 Linux タスクスタックに変更することで，スタックポインタの上書きを防ぐ．

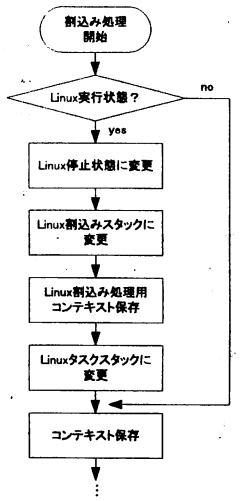

図 2 割込み発生時における OS 間切替え
Fig. 2 OS switch at the occurrence of interrpts

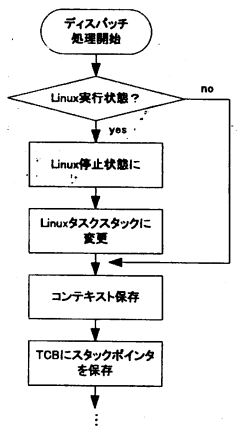

図 3 ディスパッチ時における OS 間切替え
Fig. 3 OS switch at the occurrence of dispatch

以上を元に，ディスパッチ時の OS 間切替えを図 3に示す．

ディスパッチによって Linux 停止状態から Linux 実行状態へ移行する場合，保存していたコンテキストを復帰し，Linux カーネルスタックへ変更して処理を再開する．

3.2 タスク例外処理による Linux 割込みの仮想化

本節ではタスク例外処理による Linux 割込みの仮想化について示す．

3.2.1 割込み発生時

ITRON 仕様では，タスク例外要因パターンはビット毎の論理和で管理されるため，図 4に示すように，1 つの割込み要因と

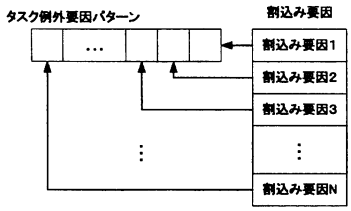

図4　割込み要因とタスク例外要因との対応

Fig. 4　Correspondence between a interrupt factor and a task exception factor

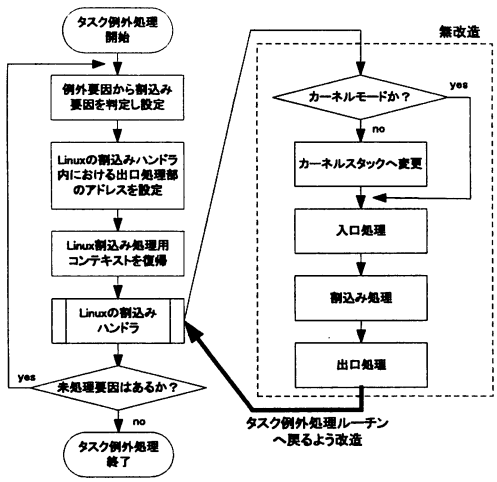

図5　タスク例外処理としてのLinux割込み処理

Fig. 5　Linux interrpt processing as task exception handling

タスク例外要因パターンの1ビットとを対応させる．Linux割込み発生時，割込み要因に対応するタスク例外要因をパラメータとして，タスク例外処理を要求する．

3.2.2　割込み処理

通常，Linuxの割込みハンドラでは，割込み発生直後に，割込み要因と割込み処理ルーチン終了後に行う割込み出口処理への戻り番地を，汎用レジスタに設定する．そこで，タスク例外処理ルーチンにおいてこれらの値を予め設定してから，関数呼出しによってLinuxの割込みハンドラを起動し，処理する．割込み要因は，要求されたタスク例外要因ビットから判定する．

また，通常のLinuxの割込みハンドラの出口は割込み処理終了命令となっていることから，これを関数の呼出し元へのリターン命令とする．これにより，Linuxの割込みハンドラの出口のみの改造で実現可能となる．

Linuxの割込みハンドラ起動前に，Linux割込み発生直後にLinux割込みスタックに保存したコンテキストを復帰し，Linux実行状態としてから，ハンドラを起動する．タスク例外処理としてのLinux割込み処理を図5に示す．

3.2.3　割込み禁止/許可

Linux割込み禁止はcli，許可はstiによって行われる．これらを，タスク例外処理禁止状態とするdis_texおよび許可状態とするena_texへと変更する．

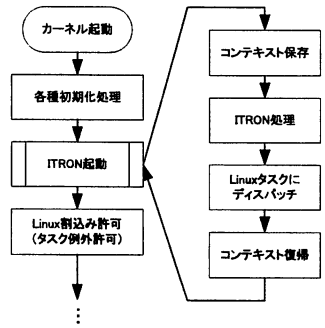

図6　ハイブリッドOSの起動手順

Fig. 6　The starting procedure of hybrid OS

3.3　起動方法

本節ではハイブリッドOSの起動方法について説明する．ハイブリッドOSの起動には，ITRONカーネルの起動とLinuxタスクの起動とが含まれる．

3.3.1　ITRONカーネルの起動

ITRONカーネルの起動方法としては，LinuxからITRONを起動する方法と，ITRONからLinuxを起動する方法の2通りが考えられる．ITRONからLinuxを起動する場合，ITRONで初期化を行ったデバイスに関してはLinuxで初期化を行わないよう，Linuxカーネル内への改造が必要になる．よって，本システムでは，ITRONカーネルはLinuxカーネルの初期化処理の中で起動されることとする．ITRONカーネルが起動されると割込み許可状態となるため，Linuxカーネルが割込み処理に関する初期化処理をすべて終了し割込み受付可能となってから，割込み禁止状態でITRONカーネルを起動すべきである．

3.3.2　Linuxタスクの起動

通常のITRONタスクと異なり，Linuxタスクは起動後，中断されていたLinuxカーネルの初期化処理を再開する．そのため，中断する際のコンテキストを保存し，Linuxタスク起動時にこのコンテキストを復帰することでLinuxタスクの起動を行う．

ITRONカーネルの起動とLinuxタスクの起動とを含めたハイブリッドOSの起動手順を図6に示す．

3.4　OS間インタフェース

本節ではOS間インタフェースの実装について示す．OS間インタフェースは，日本エンベデッドリナックスコンソーシアムによって策定された仕様に従った実装例を元に，FIFO方式の実装を行った[8][9]．OS間インタフェースによる通信は，ITRON側では新規に定義したシステムコールにより行われ，Linux側ではOS間インタフェース用のキャラクタ型デバイスを作成し，そのデバイスドライバにより行われる．

LinuxはITRONの1タスクであるため，Linuxカーネル内でITRONのシステムコールを発行可能である．よって，FIFOバッファの確保にはITRONのメモリプール管理機能に含まれる固定長メモリプールを，データの送受信にはITRONの同期・通信機能に含まれるメールボックスを利用する．実装手法を図

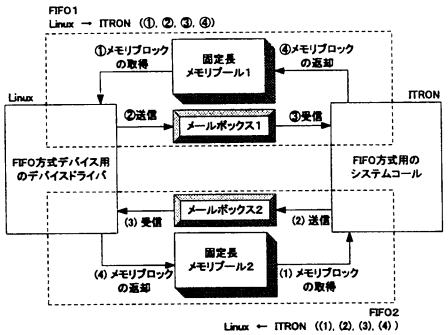

図7 FIFO方式によるOS間インタフェース
Fig. 7 The interface between two OS by the FIFO system

7に示す.

3.4.1 Linuxから ITRONへの送信

LinuxからITRONへの通信を行う場合，Linux側では，まず送信用FIFOデバイスをopenし，そのデバイスに対してwriteシステムコールを発行する．デバイスドライバにおいて，固定長メモリプールから送信バッファ用のメモリブロックを確保し，バッファに送信データを書き込み，そのアドレスをメールボックスへ送信する．

ITRON側では受信用システムコールによって，メールボックスから受信したアドレスを元にデータを読み出し，FIFOバッファとして利用したメモリブロックを固定長メモリプールへ返却する．

3.4.2 ITRONからLinuxへの送信

ITRONからLinuxへの送信を行う場合，ITRON側では送信用システムコールによって，固定長メモリプールから送信バッファ用のメモリブロックを確保し，バッファに送信データを書き込み，そのアドレスをメールボックスへ送信する．

Linux側では，受信用FIFOデバイスをopenし，そのデバイスに対してreadシステムコールを発行する．デバイスドライバにおいて，メールボックスから受信したアドレスを元にデータを読み出し，FIFOバッファとして利用したメモリブロックを固定長メモリプールへ返却する．

4. 評　価

本章では実装を行ったハイブリッドOSについての評価を示す．

4.1 割込み禁止区間の比較

各OSでの割込み禁止区間を命令数により比較した．比較対象とした区間について，TOPPERS/JSPとハイブリッドOSではカーネル内の割込み禁止状態の比較的長い区間とし，Linuxについては一例としてタイマ割込み処理時の割込み禁止区間とした．算出した割込み禁止区間命令数およびハイブリッドOSを基準とした割込み禁止区間の削減率を表1に示す．

結果から，ハイブリッド化によってLinuxの割込み禁止区間を18%にまで削減できていることが分かる．これは，Linuxを使用したシステムでも割込みに対して高い即応性があることを

表1 割込み禁止区間命令数
Table 1 Interrupt disable region size

	命令数 [命令]	削減率 [%]
TOPPERS/JSP	88	113
ハイブリッドOS	99	100
Linux	548	18

表2 割込みハンドラ起動時のオーバヘッド
Table 2 The overhead of interrupt handler starting

	命令数 [命令]	キャッシュ有効時		キャッシュ無効時	
		平均時間 [μsec]	最悪時間 [μsec]	平均時間 [μsec]	最悪時間 [μsec]
TOPPERS/JSP	32	0.94	2.89	6.47	6.72
ハイブリッドOS	54	1.21	10.60	11.34	11.52
オーバヘッド [%]	169	129	367	175	171

表3 タスクディスパッチ時のオーバヘッド
Table 3 The overhead of task dispatch

	命令数 [命令]	キャッシュ有効時		キャッシュ無効時	
		平均時間 [μsec]	最悪時間 [μsec]	平均時間 [μsec]	最悪時間 [μsec]
TOPPERS/JSP	25	0.67	1.92	5.69	6.24
ハイブリッドOS	38	1.59	5.30	6.74	8.64
オーバヘッド [%]	152	237	276	118	138

示している．

4.2 割込みおよびディスパッチオーバヘッドの計測

ハイブリッド化に伴う，割込みハンドラ起動時とタスクディスパッチ時のオーバヘッドを計測した．計測はプロセッサのタイマモジュールを使用し，割込みハンドラ起動後，要因ごとの割込み処理ルーチンの実行が開始するまでの区間およびタスクディスパッチを行う関数起動後，タスクディスパッチが完了するまでの区間で行った．割込みハンドラ起動時のオーバヘッドを表2に、タスクディスパッチ時のオーバヘッドを表3に示す。それぞれ試行数1000回の結果である。

平均時間で比較すると，キャッシュ有効時における割込みハンドラ起動時のオーバヘッドは129%である．仮に毎秒10000回の割込み発生時であっても負荷の増加率は0.3%と低い．また，キャッシュ有効時のタスクディスパッチ時におけるオーバヘッドは237%と高いが，毎秒1000回のタスクディスパッチと仮定すると，負荷の増加率は0.092%と僅かである．このことから，平均性能を重要視するシステムにおいて実用可能であると言える．

一方，最悪時間で比較すると，キャッシュ有効時において，割込みハンドラ起動時で367%，タスクディスパッチ時で276%とオーバヘッドが大きい．キャッシュ無効時の時間の増加率が命令数の増加率と同程度であることから，Linuxタスク実行時にキャッシュのエントリが書き換わることによるキャッシュミスヒットが原因であると考えられる．よって，最悪時間を重要視するシステムにおいては，許容範囲が異なるため一概には言えないことから，使用者の判断に委ねられる．

表4 Linux割込み処理遅延の最短命令数
Table 4 The shortest delay of Linux interrupt

	割込み入口	割込み処理ルーチン	割込み出口	タスク例外処理ルーチン	合計
命令数 [命令]	54	181	35	94	364

表5 ハイブリッド化に伴う改造コード量
Table 5 The number of reconstruction code lines

	プロセッサ非依存部 [行]	プロセッサ依存部 [行]	合計 [行]
TOPPERS/JSP	19	713(423)	732(423)
Linux	3	113(89)	116(89)

4.3 ITRONカーネル起動時間の計測

ITRONカーネルの起動時間を計測した．計測区間は，Linuxカーネルの初期化処理の中でITRONカーネルの初期化ルーチンが呼び出されてから，ITRONにおいて最初のタスクが実行される直前までとする．計測は京都マイクロコンピュータ株式会社のPARTNER-Jに内蔵されたリアルタイムカウンタ機能により行う．ITRONカーネル内で初期化を行うオブジェクトの数により結果は変化するため，ここではITRONが4つのタスクとタイマモジュールを持つこととした．

10回実行した結果は，平均91μsecであった．数秒要するLinuxカーネルの起動時間と比較すると，ITRONカーネルの起動時間はごく僅かであり，ハイブリッド化に伴う起動のオーバヘッドは無視できる．しかし，この結果はあくまでITRONタスクが起動される直前までの計測結果であり，ITRONカーネル後に長時間実行されるITRONタスクが起動されると，Linuxカーネルの初期化処理の完了は遅れるため，注意が必要である．

4.3.1 Linux割込み処理最短遅延命令数の算出

Linux割込みはタスク例外処理によって仮想化されるため，発生から処理の開始までに遅延が生じる．命令数で求めた最短遅延を表4に示している．

結果から，Linux割込み処理は最短でも364命令発行する時間の遅延が生じる．ITRONタスクの実行により，この遅延はさらに長くなることから，優先して処理したい割込みについてはLinux高優先度割込みとするか，もしくは即座に処理したい割込みであればITRON割込みとすべきである．

4.4 改造コード量の算出

ハイブリッド化に要した各カーネルのコード量を求めた．改造コード量は，変更，追加ともに1行とし，C言語ソースコードおよびアセンブラソースコードの行数とした．結果を表5に示す．ここ括弧内は数値に含まれるアセンブラソースコードの行数である．

TOPPERS/JSPには732行の改造を必要とした．これは，割込みハンドラやディスパッチ関数を始め，タスク例外処理機能による割込みの仮想化，OS間インタフェースなど，ハイブリッド化を行うにあたり主要な機能の実装を行ったためである．

一方，Linuxの改造コード量は116行であり，少量であると言える．これは，ハイブリッド化に伴う変更が，Linuxの割込みハンドラの入口部分と出口部分のみでよく，残りの部分に関しては再利用可能であったことが主な原因である．

100行程度の改造でハイブリッド化を実現することが可能であることから，Linuxカーネルの早い周期でのバージョンアップにも，十分に追従することが可能であると言える．

5. 今後の課題

本稿で実装を行ったFIFO方式によるOS間インタフェースは動作が不安定であり，改良が必要である．また，送受信を行う機能のみであり，[8]では他に，ITRON側ではFIFOのリセットや参照を行うシステムコール，Linux側ではclose，select，ioctl，fcntlシステムコールが定義されているため，これらの実装を行う必要がある．加えて，共有メモリ方式によるOS間インタフェースの実装も行う必要がある．

また，本稿では実装を行うターゲットプロセッサとしてSH-3を使用した．汎用PCとは異なり，組込みシステムには多種多様なプロセッサが存在するため，その他のプロセッサにも実装を行うことで，ハイブリッドOSの適用範囲を広げる．

さらに，メモリ資源を効率的に使用するため，ITRONカーネルをローダブルモジュールとして実装する手法についても今後の課題とする．

6. まとめ

本稿では，Linux割込み処理をITRONのタスク例外処理機能によって仮想化し，Linuxを完全にITRONの1タスクとして扱うことで，動的な優先度の変更によって一部のLinux割込み処理を優先して実行可能な，LinuxとITRONとのハイブリッドOSを実現した．評価結果より，平均時間を重要視するシステムにおいては実用可能な範囲であることを示した．また，Linuxカーネルの改造を116行に抑えたことから，Linuxカーネルの改造を最小限に抑えるという目標を達成し，バージョンアップにも十分に追従可能であると言える．

文献

[1] 坂村健監修，高田広章編，“μITRON4.0仕様 Ver.4.00.10”，社団法人トロン協会 (2001)
[2] 社団法人トロン協会，“TRONプロジェクトホームページ (http://www.tron.org/)”
[3] 本田晋也，高田広章，“μITRON4.0仕様の例外処理のための機能とその評価”，情報処理学会論文誌，vol.42，no.6，pp.1514-1524，Jun．2001．
[4] 日立製作所，”SH7709ハードウェアマニュアル”，日立製作所半導体グループ電子統括営業本部，4th edition(1998)．
[5] 日立超LSIシステムズ，”SH7709S Solution Engine (MS7709SSE01) 概説書”，日立超LSIシステムズシステム本部プラットフォーム設計部，4th edition(2002)
[6] “The Linux Kernel Archives(http://www.kernel.org)“
[7] “TOPPERSプロジェクト (http://www.toppers.jp)“
[8] 日本エンベデッドリナックスコンソーシアムハイブリッドアーキテクチャワーキンググループ，“Linuxによる RTOSとのハイブリッド構成に関する仕様（第1版）”，日本エンベデッドリナックスコンソーシアム (2002)
[9] 日本エンベデッドリナックスコンソーシアム監修:”TECHI組込みLinux入門”，CQ出版社 (2003)．