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

2005－ＳＬＤＭ－119（19）
2005／3／18

専用ハードウェアによるART-Linuxの高性能化に向けて

堀 洋平† 中島 俊夫† 片下 敏宏† 関山 守† 戸田 賢二†

† 独立行政法人 産業技術総合研究所
〒305–8568 茨城県つくば市梅園 1–1–1 中央第2事業所
E-mail: †{hori.y,skywalker-nakajima,t-katashita,m.sekiyama,k-toda}@aist.go.jp

あらまし 実時間処理向けに拡張されたLinuxであるART-Linuxの性能を，専用ハードウェアによって改善する試みを紹介する．ART-Linuxは，「実時間ロック」と呼ばれる独自の機構によってプリエンプティブ・カーネルを実現したハードリアルタイムOSである．通常のLinuxのデバイスドライバやアプリケーションを実時間化する機能を有し，既存のソフトウェア資産を有効に活用できるため，大規模なリアルタイム・システムを効率よく開発することが可能である．ART-Linuxは Linuxを実時間拡張したものであるため，通常よりオーバヘッドが大きくなることは避けられない．そこで，割込み処理やスケジューリング等を専用ハードウェアで実行することで，性能の向上を目指す．
キーワード リアルタイムOS, Linux, 専用ハードウェア, ソフトコアMPU

Approaches to Improving Performance of ART-Linux with Dedicated Hardware

Yohei HORI†, Toshio NAKAJIMA†, Toshihiro KATASHITA†,
Mamoru SEKIYAMA†, and Kenji TODA†

† National Institute of Advanced Industrial Science and Technology
Tsukuba Central 2, Umezono 1–1–1, Tsukuba-shi, Ibaraki-ken, 305–8568 Japan
E-mail: †{hori.y,skywalker-nakajima,t-katashita,m.sekiyama,k-toda}@aist.go.jp

Abstract This paper explores approaches to improving performance of *ART-Linux*, a real-time extension of Linux, by hardwarizing some kernel operations. ART-Linux is a hard real-time OS that offers functionalities of priority inheritance and re-usability of existing applications and device drivers. Since ART-Linux is a real-time-patched version of Linux, its kernel overhead is necessarily heavier than that of original Linux. Executing some kernel operations on specialized hardware, the overhead of real-time processing would be reduced.
Key words real-time operating system (RTOS), Linux, specialized hardware, softcore MPU

1. はじめに

入出力に関する時間的制約の厳しい組込みシステムでは，しばしばリアルタイム・オペレーティングシステム（RTOS）が利用される．近年，制御対象の規模が増大するにつれ，OSの実時間処理能力だけでなく，仮想記憶，ファイルシステム，ネットワーク等の汎用機能が重要となっている．また，大規模なシステムでは，アプリケーションやドライバをすべて独自で開発することは困難であり，既存のソフトウェア資産を有効に活用できることが求められる．このような経緯の中で，Linuxを実時間処理用に拡張した *Another Real-Time Linux (ART-Linux)* が，旧電子技術総合研究所の石綿（現，株式会社ムービングアイ）によって開発された[1]～[5].

ART-Linuxは，「実時間ロック」と呼ばれる独自の機構によって，カーネルのプリエンプティブ化と多段階の優先度継承機能[6]を実現したハードリアルタイムOSである．通常のLinuxのユーザプログラムやデバイスドライバを実時間化することができるため，大規模なリアルタイム・システムを効率よく開発することが可能あり[7]，ヒューマノイド・ロボットのOSとしても採用されている[8].

ART-Linuxは，オリジナルのLinuxに対するパッチとして配布されているほか，ART化されたLinuxパッケージが株式会社ムービングアイから提供されている[9].

ART-Linuxは，デスクトップ向けの高性能なCPUを用いた場合には，高負荷な環境においても高い実時間処理性能を示すことが報告されている[4]．しかし，筆者らの組込みシステム開

発環境である REX や REX2 [10] 上のソフトコア M32R (ルネサステクノロジ) に ART-Linux を実装した場合，ユーザ・アプリケーションの実行が体感的に遅くなる．これは，実時間処理を実現するためのカーネルのオーバヘッドが大きいためであると考えられる．そこで，ソフトコア CPU を改良し，カーネルの処理をハードウェア的に支援することで，ART-Linux の性能を向上することについて考える．

RTOS のハードウェア支援については，これまでにも研究例が報告されており，これについて第 2 章で紹介する．第 3 章では，ART-Linux の特徴と実時間処理機能の実装について説明する．第 4 章では，ハードウェアによるカーネル処理の支援について説明する．最後に第 5 章において，本研究のこれまでの考察についてまとめる．

2. 関連研究

専用ハードウェアによって RTOS を高速化する試みについて，近年，数多くの研究報告がなされている．

STRON-I は，フラグ，同期，タイマ，タスク等の管理やスケジューリングを行うコプロセッサである．μITRON を対象としており，いくつかの基本的なシステムコールを CPU 外部のハードウェアで処理することで，カーネルのオーバヘッドを削減している [11].

Context Switching Module (CSM) は，コンテキストを保存する専用の記憶階層を構築することで，コンテキスト・スイッチングを高速化している [12]. この記憶階層は，CPU 内部のレジスタ・バンクと，CPU 内部または外部のコンテキスト・メモリ (SRAM) から構成される．

Real-Time Unit(RTU) [13], [14] は，VME バスベースのシステムにおけるコプロセッサであり，時間管理，割込みや IPC 等の管理，およびスケジューリングを行う．

Real-Time Task Manager(RTM) [15] はオンチップのハードウェア・モジュールであり，時間管理，イベント管理およびスケジューリングを行う．

Mooney らによる System-on-a-Chip Dynamic Memory Management Unit (SoCDMMU) [16], System-on-a-Chip Lock Cache (SoCLC) [17], System-on-a-Chip Deadlock Detection Unit (SoCDDU) [18] および System-on-a-Chip Synchronization Unit (SoCSU) [19] は，いずれも RTOS の高速処理を目的としたオンチップのハードウェア・モジュールである．

Zhenyu らは，Field-Programmable Gate Array (FPGA) 上のソフトコア CPU である NIOS (ALTERA 社) の命令セットアーキテクチャを改良し，処理の高速化を行った．MicroC/OS-II のルーチンの一部をカスタム命令として実装することで，タスクスイッチ，プリエンプション，イベント管理の高速化に成功した [20].

Responsive Multithreaded Processor (RMT Processor) は，分散リアルタイムシステムの実現を目的に山崎らにより開発された [21], [22]. 実時間処理，実時間通信，I/O および周辺制御機能等，分散リアルタイムシステムに必要な機能のほとんどが，ハードウェアレベルで実現され 1 チップに集積されている．

3. ART-Linux

大規模な実時間・システムでは，アプリケーションやデバイスドライバのすべてを独自に開発することは困難である．そこで，既存のソフトウェア資産を利用することが可能な RTOS の作成を目的とし，ART-Linux の開発が行われた．

Linux を実時間処理用に拡張するためには，タイマの高精度化，カーネルの細粒度化，固定優先度に基づくスケジューリングが必要である．また，規模の大きなシステムではタスク数が多くなりがちであるから，優先度逆転の発生を防がなくてはならない．本章では，ART-Linux の RTOS としての特徴や機能と，その実装方法について説明する．また，カーネルのオーバヘッドを考察し，現状の問題点について述べる．

3.1 特　　徴

ART-Linux は，以下の特徴を持つ RTOS である [4].

(1) ハードリアルタイム・スケジューリング機能　独自の「実時間ロック」機構を用いて長すぎるプリエンプション禁止期間を解消し，時間精度の高いタスク管理・イベント管理機能を提供する．

(2) 実時間タスクのメモリ保護　実時間タスクにも固有のアドレス空間が割り当てられ，ユーザ空間での実行が可能である．そのため，カーネルや他のプロセスがバグのある実時間タスクから保護される．

(3) 実時間タスクと通常プロセスの共存　実行する実時間タスクが無い場合には，通常のプロセスが実行される．

(4) 多段階の優先度継承機能　RTOS における典型的な問題である優先度逆転を防ぐため，多段階の優先度継承機能を有している．これにより，優先度の高い実時間タスクがブロックされることを防ぐ．

(5) アプリケーションのバイナリ互換　通常の Linux カーネルが提供するすべてのシステムコールは，ART-Linux の実時間タスクからも呼び出すことが可能である．そのため，ART-Linux 固有のシステムコールを用いて，ユーザアプリケーションをバイナリのまま容易に実時間化することができる．

(6) デバイスドライバのソース互換　Linux には，多くのデバイスドライバが標準で付属している．また，ソースコードの公開されたフリーソフトウェアであり，使用者数も多いため，新しいデバイスのドライバの開発も早い．これら多くの Linux 用デバイスドライバは，ART-Linux のヘッダファイルを加えて際コンパイルすることで，容易にリアルタイム・システムに組み込むことが可能である．

3.2 ART-Linux の実装と問題点

ART-Linux は，ハードリアルタイム・スケジューリングを実現するために，高精度なタイマ，カーネルの細粒度化，割込み処理の実時間タスク化および優先度継承機能の実装を行っている．これらの機能の実装と問題点について，以下で説明する．

3.2.1 タ　イ　マ

Linux におけるタイマ割り込みの頻度はカーネルのコンパイル時に決定され，その値は変数 HZ によって指定されてい

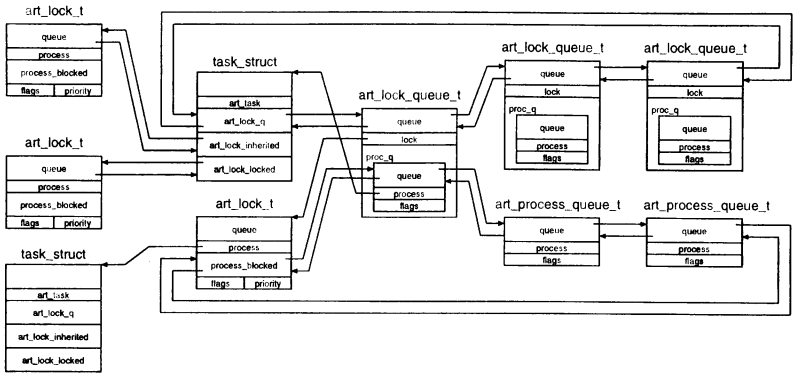

図 1　ART-Linux のデータ構造
Fig. 1　The data structure of ART-Linux.

る(注1). HZ=100 のとき tick は 10ms となり，これより短い時間間隔でのスケジューリングはできないため，リアルタイムシステムの OS として使用するには精度が不十分である．

ART-Linux におけるタイマ割り込みの頻度は，カーネル構築時に指定する変数 CONFIG_ART_TICK 基づいて決定される．CONFIG_ART_TICK の値を充分に小さくすることで，実時間タスクの高精度なスケジューリングが可能である．従来，tick (10^6/HZ [ms]) ごとのタイマ割り込みによって実行されていた処理は，周期 tick で実行される実時間タスクとして実装されている (第 3.2.3 節参照)．

しかし，タイマの高精度化によってタスクの起動時刻の細かな制御が可能になるが，タイマ割込みの頻度が高くなると必然的にオーバヘッドが大きくなる．

3.2.2　実時間ロック

グローバル変数，ファイル，デバイス等の共有資源へのアクセスが発生するクリティカルセクションでは，相互排他のためにプリエンプションを禁止しなければならない．しかし，プリエンプション禁止期間が長すぎると，優先度の高いタスクが実行されず，実時間タスクの時間的制約が満たされなくなる可能性がある．ゆえに RTOS では，カーネルを細粒度化し，クリティカルセクションをできる限り短くする必要がある．

Linux では，クリティカルセクションの相互排他のために割込みを禁止している部分がある．そのため，割込み駆動型の実時間タスクの実行が保留される危険性がある．

ART-Linux は，「実時間ロック」と呼ばれる機構によって，従来割込みが禁止されていた区間におけるプリエンプションを可能にした [4]．あるプロセスが共有資源にアクセスしていても，次に実行されるプロセスがそれにアクセスしない場合は，プリエンプションを禁止する必要がない．ゆえに，Linux の Symmetric Multi-Processing (SMP) におけるスピンロックのようなロック機構によって，共有資源の相互排他が可能である．

また，通常の Linux では，システムコールの実行中に割込みが発生した場合，ハンドラが起動しようとするプロセスはシステムコールが終了するまでスケジューリングされない．そのため，優先度の高いタスクが長時間ブロックされる可能性がある．ART-Linux では，実時間ロック機構によって，システムコールの実行中であってもプリエンプションが可能となっている．

このほか，実時間ロックは優先度継承を実現するための機能も提供している (第 3.2.4 節参照)．

実時間ロック機構を実現するため，ART-Linux のプロセスディスクリプタには以下のメンバが追加されている [3]．

- `art_task`: 実時間タスクを表す構造体．
- `art_lock_q`: 自身をブロックしているロックのリストへのポインタ．
- `art_lock_inherited`: 自身が取得したロックのうち，優先度継承機能を有するもの．優先度の降順にソートされる．
- `art_lock_locked`: 自身が取得したロックのうち，優先度継承機能を持たないもの．優先度の降順にソートされる．

select システムコールによる同期があるため，自身をブロックするロックは複数個ある場合があり，`art_lock_q` はリストでなければならない．`art_lock_inherited` と `art_lock_locked` は，以下のメンバを持つ `art_lock_t` 型の構造体である．

- `process`: そのロックを取得しているプロセスへのポインタ．
- `process_blocked`: そのロックによってブロックされているプロセスのリスト．優先度の降順にソートされる．
- `flags`: ロックの状態
- `priority`: ロックの優先度

実時間ロックの機構は複雑であり，実時間タスクのデータ構造は図 1 のようなネットワークを構成する．このデータ構造に対する操作は，非常に大きなオーバヘッドになっていると考えられる．

3.2.3　割込み処理

割込みは，発生する時刻や頻度を予想することが難しい．そのため，割込みが生じるたびに直ちにハンドラを実行する方法は，実時間タスクの実行を妨げる可能性がある．

ART-Linux では，タイマ以外の割込みハンドラは，周期的に起動される実時間タスクとして実装されている．周辺のデバ

(注1)：カーネル 2.4 でのデフォルトは 100，カーネル 2.6 では 1000

```
init-+-ART-+-PS/2 Mouse
     |     |-eth0
     |     |-ide0
     |     |-ide1
     |     |-keyboard---keyboard_softirq
     |     |-rtc
     |     |-timer-timer_softirq
     |     `-useb-uhci
(省略)
```

図 2　割込み処理用実時間タスクの例

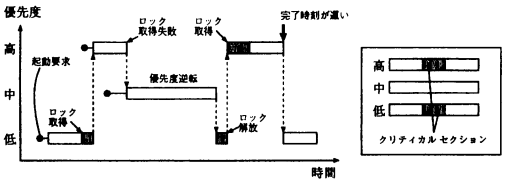

図 3　優先度逆転の例

Fig. 3　An example of *priority inversion*.

イスから割込みハンドラの登録要求があると，ART-Linux はそのハンドラを処理するためのカーネルスレッドを生成する．このスレッドは実時間化され，指定された周期ごとに起動されて割込みの有無を判定し，割込みがあればハンドラを実行する．割込みハンドラの実行をカーネルがスケジューリングすることで，実時間タスクの時間的制約が満たされる．

図 2 は，ART-Linux 上における pstree コマンドの実行結果の例である．マウスやキーボード，IDE 等の典型的なデバイスの他，従来のタイマ処理が実時間タスクとして実装されていることがわかる．

しかし，タスクスイッチは非常にコストのかかる操作であるから，割込みがあまり発生しない環境では，実時間タスクを起動することが大きなオーバヘッドとなる．

3.2.4　優先度継承機能

マルチタスク OS のタスクの同期における典型的な問題の 1 つに，優先度逆転がある．クリティカルセクションの存在によって，高優先度タスクが他のタスクの処理を待ち合わせる必要が生じ，優先度の低いタスクが実行される現象を優先度逆転という．

図 3 に優先度逆転の例を示す．図 3 では，高・中・低の優先度を持つタスクが 1 つずつある．低優先度のタスクが先にロックを取得しクリティカルセクションに入ると，後から実行された高優先度のタスクがクリティカルセクションに入ろうとしてブロックされる．このとき，中優先度のタスクが実行可能状態であれば，低優先度のタスクに先駆けて実行される．中優先度タスクの実行が終わらなければ低優先度タスクはロックを解放することができず，結果的として高優先度タスクの実行は中優先度タスクの実行終了後になる．これにより，高優先度タスクの完了時刻が大きく遅延される．

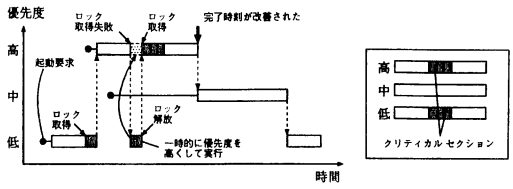

図 4　優先度継承の例

Fig. 4　An example of *priority inheritance*.

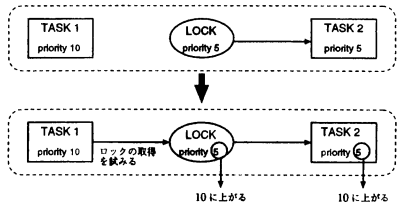

図 5　実時間ロック機構における優先度継承

Fig. 5　An example of priority inheritance with a real-time lock.

優先度逆転を防ぐ手法に，優先度継承 [6] がある．これは，高優先度タスクが低優先度タスクによってブロックされた場合，低い方の優先度を一時的に高い方と等しくする方法である．これにより，低優先度タスクが中優先度タスクに先駆けて実行されるため，クリティカルセクションのロックが直ちに解放される．図 4 に優先度継承の例を示す．図 4 において，高優先度タスクがクリティカルセクションへ入ろうとしてブロックされると，ロックを取得している低優先度タスクの優先度が一時的に高い方と等しくなる．そのため，中優先度タスクより先に低優先度タスクのクリティカルセクションの処理が終了し，その直後に高優先度タスクの処理が実行される．これによって，高優先度タスクの完了時間が改善される．

図 5 に，実時間ロックを用いて優先度が継承される様子を示す．図 5 (上) では，TASK 2 が LOCK を取得している．図 5 (下) において TASK 1 が LOCK の取得を試みるが，TASK 2 によってすでに取得されているため失敗する．ここで，LOCK の優先度が TASK 1 より小さいため，LOCK は低優先度のタスクによって取得されていることがわかる．よって LOCK の優先度は TASK 1 と等しい値に上げられ，さらにこの値が TASK 2 へと継承される．

実時間ロック機構における優先度継承は，リストを辿る操作のオーバヘッドが非常に大きいと思われる．

4.　ハードウェアの設計

本章では，ART-Linux のカーネル処理を支援するハードウェアの設計について考える．今回，特にハードウェア化によって高速化が期待される

- 割込み処理タスクのスケジューリング
- 実時間タスクのスケジューリング
- 実時間ロックと優先度継承

について考察する．考案されたハードウェアの構成を図 6 に示

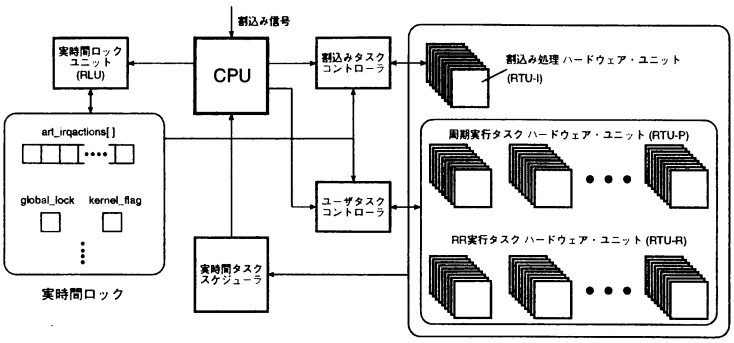

図 6 ハードウェアの構成
Fig. 6 The structure of hardware.

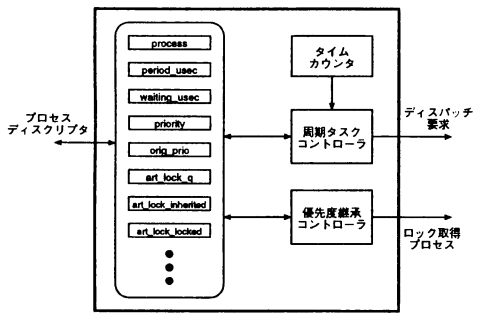

図 7 割込み処理タスクのハードウェア・ユニット (RTU-I)
Fig. 7 The real-time task unit for interrput processing (RTU-I).

す．図 6 が示すように，ハードウェアは主に

- 割込み処理ユニットとそのコントローラ
- ユーザタスク処理ユニットとそのコントローラ
- 実時間タスクスケジューラ
- 実時間ロックユニット

から構成されている．以下に，これらハードウェア・モジュールを用いた高速化手法について説明する．

4.1 割込み処理

第 3.2.3 節で述べたように，ART-Linux における割り込みハンドラは，周期的実時間タスクとして実装されている．これらのタスクが周期的に起動されて割込みの有無を判定するが，割込みが無い場合にもタスクの切替が起きるため，CPU に大きな負荷がかかっていると考えられる．そこで，各割込み処理タスクに対応するハードウェア・ユニットにより割込みを監視し，割込みが発生した場合のみタスク・スイッチが行われるようにする．図 7 に割込み処理を行うハードウェア・ユニット (Real-time Task Unit for Interrupt processing: RTU-I) の構成を示す．

（1） 割込み処理タスクが生成されると，プロセス・ディスクリプタの一部が RTU-I にコピーされる．

（2） RTU-I 内部のタイムカウンタによって時間経過を監視．

（3） 指定周期ごとに割込みの有無を確認し，割込みがあった場合はディスパッチ要求信号を実時間スケジューラに送信する．

（4） 実時間スケジューラがスケジューリング・ポリシーに基づいてディスパッチ要求の 1 つを選択し，CPU に通知する．

従来，実時間タスクを起動して行っていた割込みの監視を RTU-I を用いてハードウェア的に実行し，割込みが発生した場合のみディスパッチを要求することで，ポーリングとタスク切替えのオーバヘッドを削減することが可能である．

4.2 ユーザ実時間タスク

ユーザの実時間タスクも，割込み処理タスクと同様にハードウェア・ユニットによって起動管理を行う．ユーザが作成可能な実時間タスクは，周期実行タスクとラウンドロビン実行タスクの 2 つである．周期実行タスク用のユニットを RTU-P (Real-time Task Unit for Periodic task)，ラウンドロビン実行タスク用のユニットを RTU-R (Real-time Task Unit for Round robin task) と呼ぶ．

ユーザタスクが実時間化されると，プロセス・ディスクリプタの一部がコントローラを通じて RTU-P あるいは RTU-R にコピーされる．RTU-P は，タスクの起動時刻に達したときにディスパッチを要求する．RTU-R は常にディスパッチを要求するが，スケジューラは，割込み処理タスクや周期実行タスクの要求を優先して選択する．

図 6 では，優先度ごとに RTU-P 群と RTU-R 群を用意している．そのため，優先度に基づいたディスパッチ要求の選択は容易であるが，ハードウェア・リソースを大量に必要とする．その他の方法として，RTU-P 群と RTU-R 群をすべての優先度のタスクで共有し，スケジューラに対してディスパッチ要求とともに自身の優先度を通知することが考えられる．この場合，スケジューラがやや複雑になるが，必要なハードウェア・リソースは少ない．

このように，ユーザの実時間タスクのスケジューリングを RTU-P，RTU-R を用いてハードウェア的に行うことで，CPU の負荷を分散することが可能である．

4.3 実時間ロックと優先度継承

第 3.2.2 節で述べたように，ART-Linux の実時間ロック機構ではデータ構造が図 1 のようなネットワークを形成する．このデータ構造に対する操作のオーバヘッドは，極めて大きいと

考えられる．そこで，実時間ロック機構のデータ構造を管理するハードウェア・ユニット (Real-time Lock Unit: RLU) について考える．

あるタスクがロックを取得するとき，RLU 内部の対応するレジスタ (図 6 左) に自身へのポインタを保存する．すでに他のタスクへのポインタが保存されていた場合，取得を試みたタスクはブロックされる．このとき，保存されているポインタに基づき，ロックを取得しているタスクを管理している RTU-I/P/R を探す．見つかった RTU-I/P/R の art_lock_q メンバに，ブロックされるタスクへのポインタを保存する．このとき，ブロックされるプロセスの優先度の方が高い場合は，ロックおよびそれを取得しているタスクの優先度が変更される．

実時間ロックによる優先度継承では，リスト操作がシーケンシャルに行われるため，ハードウェアによる並列処理が困難である．また，1 つのロックによってブロックされるタスク数が不明であるため，充分なハードウェア・リソースを用意する必要がある．実時間ロック機構については，ハードウェア化の効果についてさらに検討してゆく．

5. おわりに

本稿では，ART-Linux の機能とそれらの実装方法について説明し，これらをハードウェア化することで性能を改善することについて考察した．

ART-Linux は，独自の「実時間ロック」と呼ばれる機構によって，カーネルの細粒度化，プリエンプティブ化，優先度継承機能を実現したハードリアルタイム OS である．既存の Linux ドライバやアプリケーションを容易に実時間化することができるため，大規模なリアルタイム・システムの開発効率が大幅に向上する．

ハードウェアによる ART-Linux の高速化の例として，まず，ハードウェア・ユニットによる割込み処理タスクとユーザ実時間タスクの管理について説明した．複数のハードウェア・ユニットによる割込みの監視，時間管理を行い，各ユニットのディスパッチ要求をスケジューラを通じて CPU に伝える方法について説明した．これにより，割込み処理タスクの切替回数と，実時間タスクのスケジューリングのオーバヘッドを低減することができると考える．

さらに，実時間ロックによる優先度継承を，ハードウェアによって支援する方法について考察した．実時間ロックを用いた優先度継承におけるデータ操作は極めて複雑で，ハードウェアによる高速化の効果が大きいと期待される．しかし，リストへのシーケンシャルなアクセスが発生するため並列処理が難しく，ハードウェア・アルゴリズムやアーキテクチャ，性能向上の効果について，今後も検討してゆく．

文　献

[1] 石綿陽一，松井俊宏，“汎用 OS とデバイスドライバを共有できる実時間オペレーティングシステム，” 信学技報 CPSY97-119, pp.41–48, 1998.

[2] 石綿陽一，“ART-Linux 誕生の経緯と使い方，” Interface, pp.109–118, CQ 出版社, Nov. 1999.

[3] 石綿陽一，“リアルタイム処理を実現する ART-Linux の設計と実装，” Interface, pp.119–129, CQ 出版社, Nov. 1999.

[4] 石綿陽一，“Linux カーネル 2.2 シリーズに基づく ART-Linux の開発とそのリアルタイム処理性能，” Interface, pp.130–137, CQ 出版社, June 2002.

[5] 石綿陽一，“SMP カーネルに基づく ART-Linux の安定化と実時間処理性能の測定，” 計測自動制御学会システムインテグレーション部門講演会講演論文集 (II), pp.417–418, 2002.

[6] L. SHA, R. RAJKUMAR, and J.P. LEHOCZKY, “Priority inheritance protocols: An approach to real-time synchronization,” IEEE Trans. Computers, vol.39, no.9, pp.1175–1185, Nov. 1990.

[7] 木野桂司，“MI6 を活用した大幅 TCO 削減をリアルタイム linux で実現，” 株式会社ムービングアイ, Apr. 2003.

[8] K. Yokoi, F. Kanehiro, K. Kaneko, S. Kajita, K. Fujiwara, and H. Hirukawa, “Experimental study of humanoid robot HRP-1S,” Intl. J. Robotics Research, vol.23, no.4–5, pp.351–362, 2004.

[9] 株式会社ムービングアイ, http://www.movingeye.co.jp/.

[10] 戸田賢二，佐谷野健二，“M32R プロセッサソフトマクロの FPGA を用いたマルチプロセッサ実験用プラットフォーム REX への移植，” 信学技報 CPSY2003-51, pp.39–43, 2004.

[11] T. Nakano, A. Utama, M. Itabashi, A. Shiomi, and M. Imai, “Hardware implementation of a real-time operating system,” IEEE Proc. TRON Project Intl. Symp., pp.34–42, 1995.

[12] J. Ito, T. Nakano, Y. Takeuchi, and M. Imai, “Effectiveness of a high speed context switching method using register bank,” IEICE Trans. Fundamentals, vol.E81-A, no.12, pp.2661–2667, Dec. 1998.

[13] L. Lindh, J. Starner, and J. Furünas, “From single to multiprocessor real-time kernels in hardware,” Proc. IEEE Real-Time Technology and Applications Symposium, pp.42–43, 1995.

[14] J. Adomat, J. Furunäs, L. Lindh, and J. Stärner, “Real-time kernel in hardware RTU: A step towards deterministic and high performance real-time systems,” Proc. Eighth Euromicro Workshop on Real-Time Systems, pp.164–168, June 1996.

[15] P. Kohout, B. Ganesh, and B. Jacob, “Hardware support for real-time operating systems,” Proc. Intl. Conf. Hardware/Software Codesign and System Synthesis, pp.45–51, Oct. 2003.

[16] M. Shalan, and V. Mooney, “Hardware support for real-time embedded multiprocessor system-on-a-chip memory management,” Proc. Intl. Symp. Hardware/Software Codesign, pp.79–84, 2002.

[17] B.E.S. Akgul, and V.J. Mooney, “The system-on-a-chip lock cache,” Intl. J. Design Automation for Embedded Systems, vol.7, no.1–2, pp.139–174, Sept. 2002.

[18] P.H. Shiu, Y. Tan, and V.J. Mooney III, “A novel parallel deadlock detection algorithm and architecture,” Proc. Intl. Symp. Hardware/Software Codesign, pp.30–36, 2001.

[19] B.E. Saglam, and V.J. Mooney III, “System-on-a-chip processor synchronization support in hardware,” Proc. Design, Automation and Test in Europe, 2001.

[20] J. Zhenyu, M. Sindhwani, and T. Srikanthan, “RTOS acceleration on soft-core processors using instruction set customization,” Proc. Intl. Conf. Field-Programmable Technology, pp.371–374, Dec. 2004.

[21] 山崎信行，“Responsive multithreaded processor の全体設計，” 信学技報 CPSY2003-46, pp.9–14, 2004.

[22] 薄井弘之，内山真郷，伊藤務，山崎信行，“Responsive multithreaded processor における実時間処理命令供給機構，” 信学技報 CPSY2003-47, pp.15–20, 2004.