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

2004－OS－95　(3)
2004／2／26

リアルタイム組込みウィンドウシステム「EMiRea」の実現と評価

林　寛　　下屋鋪　太一　　並木　美太郎
東京農工大学大学院工学研究科

概要

本稿では，T-Engine でのリアルタイム描画処理の実装を行い，リアルタイムの有効性の評価について述べる．近年の組込みシステムには，オペレーティングシステム (OS) のリアルタイムだけでなく，グラフィカルユーザインタフェース (GUI) へもリアルタイム性が必要となってきている．しかし，リアルタイム OS の活発な開発に比べ，リアルタイム GUI はほとんど存在していないのが現状である．本稿で述べる組込みウィンドウシステム「EMiRea」は，描画要求に対するデッドラインの設定やウィンドウへの優先度の付加により，描画処理におけるリアルタイム性を実現した．また,「EMiRea」を使った，リアルタイム処理の有無による違いを比較する評価実験を行った．その結果，リアルタイム性を持つことで，周期などのデッドラインを持つ描画要求を時間制約を守って処理することが可能であることや優先度に従った描画処理が行われることなど，リアルタイム描画処理の有効性を示す結果を得られた．

Implementation and Evaluation of the Realtime Window System “EMiRea” for Embedded Systems.

Hiroshi Hayashi　Taichi Shimoyashiki　Mitaro Namiki
Graduate School of Engineering, Tokyo University of Agriculture and Technology

Abstract

This paper shows to implementation of realtime drawing processing in T-Engine, and evaluation of the validity of realtime drawing. Realtime operation to draw in window system is needed not only for an operating system (OS) but a graphical user interface (GUI) at Embedded systems in recent years. However most Realtime GUI are not developed although Realtime OS is developed actively, in the present. The realtime window system “EMiRea” for embedded system is implemented realtime drawing processing by setting deadline as drawing demands and setting priority as windows. Moreover, we compared and evaluated the difference to the existence of realtime using “EMiRea”. The results that “EMiRea” was able to keep time restrictions of the drawing processing with deadlines such as a cycle time, and “EMiRea” was able to process drawing according to the priority, were obtained because “EMiRea” has realtime processing. As the result, we were able to show the validity of realtime drawing processing.

1　はじめに

今日，多種多様な機器にマイクロプロセッサが搭載され組込みシステムとして広く利用されている．特にここ数年の著しい技術の発展により組込み機器の高性能化や高機能化，小型化が進み，ハンドヘルド PC や PDA，携帯電話などの情報家電だけでなく，ビデオデッキなどの日常的に利用する家電製品なども，多機能化するにしたがい組込み用のオペレーティングシステム（以下 OS と記す）が搭載されるようになってきた．

多機能化が進む反面，組込み OS を搭載する機器の筐体サイズやリモコンサイズの限界によるボタンなど入力装置の実装個数には制限があると共に，各機能の操作性の良さも求められるため，グラフィカルユーザインタフェース (GUI) が用いられることが多くなった．

また，組込みシステムの使用用途は，遅延などが

許されず確実に処理されなければならないことが多いため，組込み用 OS にはリアルタイム性を持つものが多い．また，組込み GUI についても，操作に対して時間内に確実なフィードバックが必要であるなど，リアルタイム性が必要であることが多い．現在，組込み向け GUI として，MicroWindows プロジェクトの MicroWindows[1] や，QNX Software Systems の Photon microGUI[2]，XiSys Software GmbH の XiBase9[3]，Qt-Embedded[4] など，多くの GUI が開発，製品化されているが，これらの GUI では，RTOS への対応を謳っているものは数多いが，GUI 自体にリアルタイム機能を有しているものは無い．

また，リアルタイム性を持つウィンドウシステムとして，立命館大学ので開発された RTOS「Easel」向けのウィンドウシステム [5] や，X window system をベースとして開発された，コンカレント日本の RealtimeX[6] などがあるが，しかし，これらのウィンドウシステムのターゲットマシンは，デスクトップ PC など高速な CPU を持ち，大容量のメモリを持ったものを想定しているため，組込みシステムへの応用が難しい．

そこで，本研究ではリアルタイム性を持つ組込み向けウィンドウシステム「EMiRea」を開発し，組込み機器への実装を行った．また，リアルタイム性の有効性の評価実験を行った．以降の章で詳細を述べる．

2 「EMiRea」の概要

2.1 目標

本研究では，サーバ･クライアント方式による組込み向けリアルタイム組込みウィンドウシステム「EMiRea」の開発を行い，実際に組込み機器への実装とリアルタイム性の有効性の検証を行うことを目標とする．また，本システムでは，現在携帯電話などで主流となりつつあるムービーの再生やウェブ閲覧など，マルチメディア端末向けのウィンドウシステムをメインの利用方法として開発を行う．

以上の目標を達成するために，本システムでは以下のような目標を設けた．

(1) 描画処理に対してリアルタイム性を持たせる

組込み機器では，時間内に確実に描画させなければならないことは多々ある．特に，ムービー再生などの周期的な描画が必要な場合，周期的な描画要求に対応して，各周期内に描画処理が完了する必要がある．そこで，本システムでは，リアルタイムタスクだけではまかないきれない描画処理に対し，優先度と時間保証によるリアルタイム性を持たせる．

本論文では，描画要求イベントのメッセージコマンドをグラフィックドライバが理解できる形式にデコードする処理のことを描画処理と呼ぶ．

なお，「EMiRea」のリアルタイム処理を行う範囲は，アプリケーション (AP) が描画要求メッセージを送信開始時を始点とし，要求された処理を行った結果を出力装置のドライバへ渡し終えた時点を終点とした．

(2) 資源の消費を抑える

筐体のサイズや値段の関係から，組込み機器に搭載されている ROM や RAM のメモリサイズは，デスクトップ PC などに比べて極めて少ないことが多い．このように少ないメモリサイズで，ウィンドウシステムだけでなく OS や AP プログラムなど複数のプログラムを保存し動作させなければならないため，ウィンドウシステムはできるかぎりメモリの使用量を少なくする必要がある．現在存在する組込み機器のスペックを検証した結果，システム本体のサイズを 500KB 程度，実行時の使用メモリサイズを 1MB 程度に抑えることができれば，多様な組込み機器へで動作すると考え，上記のサイズを本システムでの最終的なサイズの目標とした．また，NEC の VR4181(66MHz) や，SH シリーズの SH3(100MHz)，ARM シリーズなどの中で最低ラインの性能の CPU でも動作させることを考え，「EMiRea」での最低動作クロックを 50MHz 前後とする．

2.2 設計

「EMiRea」は，サーバ・クライアント型のウィンドウシステムとして設計を行った．以降，システム全体を示す場合に「EMiRea」と呼び，ウィンドウシステムの本体部分を「EMiRea」サーバと呼ぶ．

「EMiRea」のリアルタイム描画処理のために，AP からの描画要求にデッドラインを設定し，描画処理に時間を保証することを可能とした．また，ウィンドウごとに優先度を持たせることで，ウィンドウの重要度に応じた描画処理を可能とした．以下に「EMiRea」の全体構成と，デッドラインおよび優先度について詳細を示す．

2.2.1 「EMiRea」の構成

まず,「EMiRea」の全体構成を図 1 に示す．本システムの構成は，ウィンドウシステム本体として AP 管理部，タイマ管理部，グラフィックエンジンがあり，さらに AP とウィンドウシステムを結ぶ通信路，API のライブラリから成る．今回実装を行ったのは，図 1 の黒い太線で囲まれた部分である．

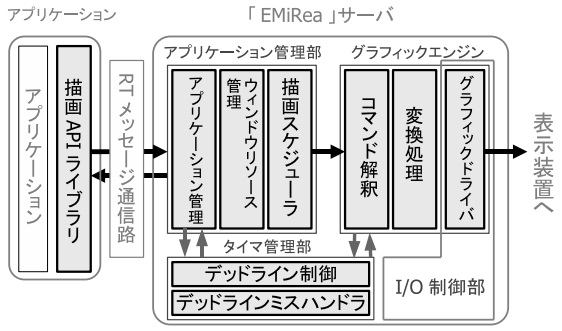

図 1: 「EMiRea」の全体構成

AP 管理部は，AP とのメッセージのやり取りの管理および，ウィンドウの情報管理，AP からの描画要求メッセージのスケジューリングなどを行う．

タイマ管理部は，描画処理のデッドライン管理を行い，デッドラインミスが起こった場合には，AP 管理部への通知やデッドラインミスハンドラの起動などを行う．

グラフィックマネージャは，AP 管理部でスケジューリングされた描画要求を実際にグラフィックドライバに対応する形式への変換を行う．

描画 API ライブラリは，AP に「EMiRea」サーバの間でメッセージのやり取りを行うための API を提供するライブラリである．メッセージには描画要求メッセージとイベントメッセージがあり，描画要求メッセージは，AP が「EMiRea」サーバへ向けて図形などの描画を要求する際に用いるもので，API 関数とほぼ一対一で対応している．また，イベントメッセージは「EMiRea」サーバから AP へ，デッドラインミスや再描画要求，ウィンドウのフォーカスイン，フォーカスアウトなど，ウィンドウの状態通知，入力などの割込みの通知などを行う．描画要求メッセージは,「EMiRea」は省資源化のために，サーバ側にウィンドウの状態を保持するフレームバッファを持たず，再描画を保証しないため，AP へ再描画処理を依頼するために用意されたイベントである．したがって，AP プログラマは再描画要求を受けた際の処理をあらかじめ用意する必要がある．

2.2.2 デッドライン

「EMiRea」では，リアルタイム描画処理を実現するために，描画要求のひとまとまりにデッドラインを設定することで，その要求に対する処理の完了までの時間を制約することができる．

デッドラインは，描画要求の 1 メッセージごとに設定することも，複数のメッセージをまとめて一つ設定することも可能である．

「EMiRea」の目標とするリアルタイム処理はソフトリアルタイムであり，ハードリアルタイムのように制約した時間までに処理が完了すること保証するのではなく，できる限り処理完了までの時間を保証するが，万が一できなかった場合はデッドラインミス処理により復帰できることを前提としている．そのため,「EMiRea」の API には，デッドラインミスが起こった場合に実行されるデッドラインミスハンドラの設定を行う関数が用意され，AP プログラマは作成する AP の用途に応じて，デッドラインミスからの復帰処理を設定することができる．

デッドラインミス処理設定には,「EMiRea」サーバ側と AP 側の二つの設定が必要である.「EMiRea」サーバ側の設定は，以下の三つから選ぶ方式を取る．

(1) NOTICE : ウィンドウを生成した AP にデッドラインミスの通知のみ行う．

(2) REDRAW : 再描画要求のみ発行する．

(3) NO_ACTION : 何も行わない．

NOTICE を選択した場合，AP 側にデッドラインミスハンドラを AP の処理内容に応じて AP プログラマが用意し，デッドラインミス発生メッセージがサーバから来た場合には，このハンドラが実行される．サーバ側に REDRAW を設定した場合，AP へは再描画要求メッセージが通知されるため，AP 側はデッドラインミスハンドラではなく，再描画のために用意したハンドラを利用する．

2.2.3 優先度

リアルタイム描画処理実現のために,「EMiRea」はデッドラインと共にウィンドウに優先度を設定する

機能も持つ．デッドラインのみによる描画順序のスケジューリングに比べ，優先度とデッドラインによる描画順序のスケジューリングは，デッドラインミスの発生を抑えると共に，描画対象への重み付けを行うことが可能となる．

優先度の個数は，最小 2 から最大 255 まで設定することができる．基本的には，タスクの優先度数と合わせるべきではあるが，システムプログラマが用途に応じて任意に設定することが可能である．

また,「EMiRea」ではウィンドウに付加する優先度を固定ではなく，状態に応じて可変させる方式を取っている．優先度が変化する条件は以下の三通りである．

(1) 他ウィンドウとの位置関係による場合

(2) エンドユーザが任意に優先度の変更を行う場合

(3) AP プログラマが必要に応じて優先度を変化させる場合

ただし，AP によっては優先度を変更されては困る場合もある．そのため，本システムは AP プログラマが必要に応じて優先度を固定/可変の変更を行う API も提供する．

3 「EMiRea」の実装

「EMiRea」を実装する上で，ターゲットマシンに必須となるのが LCD などの画面出力デバイスを搭載しているものであることである．また，開発を行う上である程度の開発環境を構築でき，ハードウェアの仕様などが公開されているものが望ましい．この条件に見合うものとして，本システムの実装対象を以下の二機種とした．各マシンのスペックを表 1 に示す．

(1) PIECE(アクアプラス社製ゲーム機：図 2)

図 2: PIECE

(2) T-Engine/SH7727 開発ボード (パーソナルメディア株式会社：図 3)

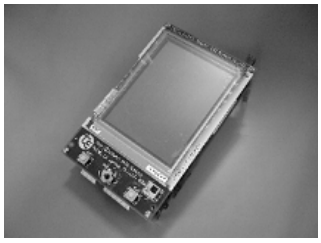

図 3: T-Engine/SH7727 開発ボード

表 1: ターゲットマシンのスペック

	PIECE	T-Engine 開発ボード
CPU	S1C33209 24MHz(EPSON)	SH7727 96MHz(HITACHI)
RAM	256KB	32MB
Flash ROM	512KB	8MB
LCD	128 × 88 FSTN 液晶 (白黒四階調)	240 × 320 TFT 液晶 (16bit カラー)
インタフェース	USB，赤外線入出力	USB，赤外線入出力， PCMCIA スロット シリアル etc.

また,「EMiRea」は単体で動作する訳ではなく組込み RTOS 上で動作することを前提として設計されているため，ターゲット OS が必要となる．そこで，本研究室で開発を進めている組込み RTOS「開聞」[7] をターゲット OS とした.「開聞」には，周期的タスク，デッドライン付きタスク，優先度付きタスク，の三種類のタスク方式を持ちそれぞれの方式ごとに適切なスケジューラを持っている．また，三つのタスク方式には，優先順位があり周期的タスクが最も高優先度でスケジューリングされ，優先度付きタスクが最も低優先度となる．方式の違う複数のタスクが処理待ち状態の場合，この優先順位にしたがってスケジューリングが行われる．

4 「EMiRea」の評価

T-Engine 上で評価を行う上で，まず T-Engine で実装した「開聞」と「EMiRea」の本体サイズおよび実行時のメモリ使用量がどの程度となったのかを表 2 に示す.「EMiRea」は，まだ低レベル API のみの実装であるため，本体サイズおよびメモリ使用量は目標で示した，本体サイズ 500KB 以内，メモリ使用

量 1MB 以内という値に収まっている．今後，高レベル API を実装した場合，本体サイズやメモリ使用量は増加すると考えられるが，そのような機能拡張を行ったとしても，「EMiRea」の目標値以内に収めることができると考えられる．

表 2: 「開聞」と「EMiRea」の実装サイズ

	本体サイズ	メモリ使用量
「開聞」	24.1KB	26.6KB
「EMiRea」	15.5KB	233KB

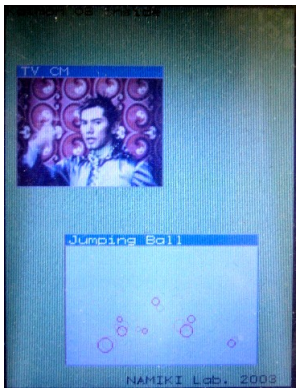

図 4: 実行中の画面

4.1 評価 AP の仕様

今回行った評価では，それぞれ一つのウィンドウを持つ三つの AP を共に周期的タスクとして動作させ，そのうち二つは周期を固定，残りの周期を変動させることで描画処理に与える負荷を変えて，描画処理にリアルタイム性を持つ場合と持たない場合の計測を行った．三つの AP のウィンドウサイズ，AP の内容，動作周期，使用した描画関数の種類および数を表 3 に示す．

表 3: アプリケーションの仕様

	AP1	AP2	AP3
サイズ	160 × 100	120 × 90	217 × 157
AP の内容	図形描画	Movie 再生	ゲーム
動作周期	10〜40ms	40ms	30ms
使用関数	DrawArc(8) ClearWindow(1)	BitBlt(1)	DrawArc(1) DrawBox(1〜36) ClearWindow(0〜1)

また，AP1 と AP2 を実行した際の画面を図 4 に示す．

4.2 WS タスクの処理方式の比較

「EMiRea」サーバを周期的タスクとして動作させた場合と，非周期的タスク (今回は優先度付きタスクを用いた) として動作させた場合の検証を行った．この検証では AP1 を用いて，AP1 の周期を変化させた際の AP1 ウィンドウのデッドラインミスの起こる確率と，「EMiRea」サーバの CPU 利用率を比較することによりリアルタイムウィンドウシステムのタスク方式の得手不得手を考察する．「EMiRea」サーバを周期的タスクとして動作させる際の周期は，AP1 の変化させる周期の最小である 10 ミリ秒とした．また，「EMiRea」サーバを優先度付きタスクとして動作させた場合，AP1 が描画要求を生成するたびに「EMiRea」サーバが起床する形をとった．

図 5 に結果を示す．

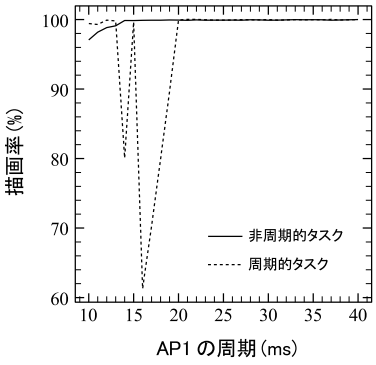

図 5: タスク方式の比較

図 5 の横軸は AP1 の周期，縦軸の描画率は，描画要求回数に対する実際に描画された回数の割合 (1) である．

$$\text{描画率 (\%)} = \frac{\text{描画回数}}{\text{描画要求数}} \times 100 \qquad (1)$$

「EMiRea」サーバを非周期的タスクとした場合，AP1 の周期が 15 ミリ秒程度から描画率が多少落ちてはいるが，周期的タスクよりも安定していることがわかる．「EMiRea」サーバを周期的タスクとして扱った際の描画率の落ち込みは，「EMiRea」サーバと AP2 の起床タイミングの衝突と「EMiRea」サーバの周期が短いことにより「EMiRea」サーバの処理が優先されるためであると考えられる．つまり，同時に起床した場合，上記の理由により AP1 からの描

画要求を受取る前に「EMiRea」サーバが実行され，その後に AP2 が描画要求を発行するために，実際に描画処理が行われるのが「EMiRea」サーバの次の起床時となり，さらに描画要求のデッドラインの短さから，描画処理中または「EMiRea」サーバの起床前にデッドラインミスが発生する．

上記以外にも，他の AP による動作検証も行ったが,「EMiRea」サーバを周期的タスクで動かした場合には，サーバタスクと AP の周期が一致している場合の動作はサーバを非周期的に動作させるよりも安定していたが，サーバの一周期内に描画処理が完了しない場合や，複数の AP が異なった周期で描画要求を発行した場合などには，描画処理にデッドラインミスが多発するなど，描画が不安定になった．

以上の結果より，複数のウィンドウで周期の異なる描画処理を行う場合や，一描画にかかる処理時間が大きなものの場合，タスクの起床時間や動作時間がフレキシブルな，非周期的タスクとして「EMiRea」サーバを動作させた方が効率良く描画処理を行うことができる．したがって，異なる種類の AP，複数ウィンドウでの動作を前提としたシステムを構築するならば,「EMiRea」サーバは非周期的タスクとして動作させた方が良いことがわかった．また，描画処理が軽量で周期的かつ単一ウィンドウ表示であるような AP を扱うのであれば，逆に「EMiRea」サーバを周期的タスクとして，周期をウィンドウの描画周期に合うように調整した方が非周期的タスクとして動作させるよりも描画処理の効率が良い．

以降の評価実験では，周期の異なる複数の描画要求を扱うため,「EMiRea」サーバは非周期的タスクとして動作させた．

4.3 「EMiRea」のリアルタイム描画機能の評価

複数の AP を同時に実行し，描画処理に負荷をかけた場合にリアルタイム処理有無によりどのような違いが出るのか検証を行った．

4.3.1 評価方法

今回の評価方法は，AP1 から AP3 の三つの AP を同時に 30 秒間動作させ，AP1 の周期を変化させていくことで描画要求の発行数を増加させることにより描画処理に負荷をかける．描画処理にかける負荷を変化させながら，

(1) 各ウィンドウの描画率

(2) AP2 のムービーの再生時間

の二つの項目について,「EMiRea」のリアルタイム機能有りの場合と無効化した場合で計測を行い「EMiRea」のリアルタイム機能の考察を行った．(1) の描画率の計測により描画処理に対する優先度の有効性を示し，(2) の再生時間の計測により時間制約の有効性を示す．

今回の評価におけるリアルタイム機能の無効化とは,「EMiRea」の描画機能のうちデッドラインと優先度無くした状態を示す．つまり，図形などの描画処理方式は「EMiRea」とまったく同様であるが，描画要求にデッドラインを設定せず，また優先度を単一とした状態をリアルタイム機能を無効の状態とした．

4.3.2 ウィンドウの描画率による評価

まず，各ウィンドウの描画率について表 4 に示す設定で評価実験を行った．

表 4: 評価実験の設定値

	AP1	AP2	AP3	RT の有無
評価実験 1	2	0	1	有
評価実験 2	0	2	1	有
評価実験 3	-	-	-	無

評価実験 1 から 3 の結果をそれぞれ図 6，図 7，図 8 に示す．なお，事前に各 AP 単体での描画率を計測したが，単体では描画処理およびタスク処理にデッドラインミスは起こらなかった．また，この計測ではどの AP にも周期的タスクにデッドラインミスは起こらず，描画率の低下の原因は全て描画処理のデッドラインミスによるものである．

これらの結果から，リアルタイム有りの場合，高優先度なウィンドウでもデッドラインミスは起こっているものの優先度が高いほど描画率が高く保たれていることが確認できる．リアルタイム無しの図 8 では，負荷の増加に伴って，全てのウィンドウが等しく描画率の低下が起こっているのに対し，リアルタイム描画を行った図 6 や図 7 では，優先度に従ったスケジューリングが行われていることが描画率に現れている．

以上の結果から，優先度による描画スケジューリングが正しく行われていることが確認できた．

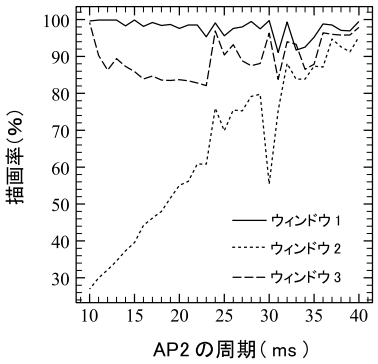

図 6: リアルタイム有りの結果 1

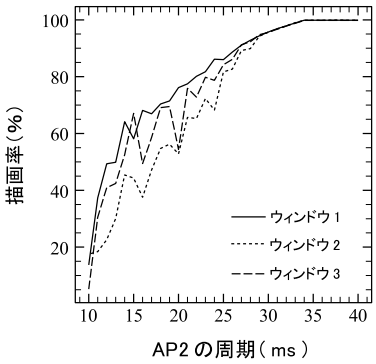

図 8: リアルタイム無しの結果

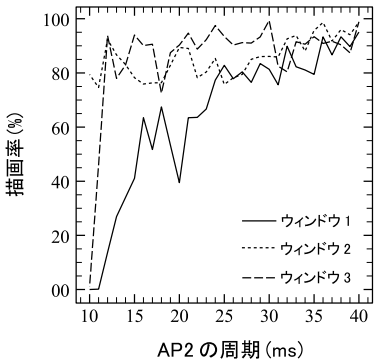

図 7: リアルタイム有りの結果 2

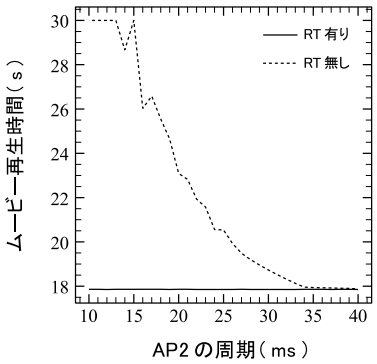

図 9: ムービー再生時間の比較

4.3.3 ムービー再生時間

次に，AP を三つ起動した際の AP2 のムービー一周の再生時間の比較を図 9 に示す．AP2 で再生するムービーは，一周 445 フレームのムービーであり，一周にかかる時間の理論値は，AP2 が 40 ミリ秒の周期であることから「445(フレーム) ÷ 25(フレーム/秒) = 17.8 秒」であるため，計測結果が 17.8 秒ならば時間制約を守っていることになる．なお，リアルタイム処理なしのグラフの 10ms から 13ms にかけて 30 秒で一定となっているのは，評価実験のサンプル取得時間が 30 秒間であるのに対してムービーの再生時間が 30 秒以上かかっていることを示す．

図 9 の比較結果から，リアルタイム処理を行った場合には，負荷を増加させても一周あたりの時間は変化しないのに対し，リアルタイム処理を無効とした場合には，負荷が増えるごとに一周あたりの時間が増加していることがわかる．リアルタイム有りの場合には，優先度を考慮し，デッドラインを守るように描画順序がスケジューリングされるために，各 AP のウィンドウは設定した時間どおりに描画され，また，デッドラインを超えた場合には適切な処理が行われている．したがって，どんなに描画処理に負荷がかかっても，デッドラインミスは起こるもののムービーの再生時間は変化することはない．それに対し，リアルタイム処理を無効化した場合には，優先度が無いために，AP2 が周期的な描画要求を出しても，他のウィンドウへの描画処理が行われていることで，AP2 ウィンドウへの描画処理の開始が遅れ，また，時間制約が無いために，各 AP のウィンドウは前回の描画処理が完了するまで次の描画要求の処理ができない．その結果，負荷が増すほどムービーの再生時間が増加する．

以上の結果より，描画処理にも時間制約を持たせることで，タスクの周期と描画処理の周期のズレや遅延を最小限に留めることが可能であることが確認できた．

4.4 CPU に負荷をかけた場合の検証

前節の評価実験では「EMiRea」サーバを優先度付きタスクとして動作させたが，AP タスクの CPU 利用率が 1%未満と低いため，CPU のほとんどを「EMiRea」サーバが利用することができたため，問題なく動作していた．しかし，ターゲット OS である「開聞」では優先度付きタスクは，周期的タスクよりも優先度が低いため，もし AP タスクやそれ以外のタスクの CPU 利用率を上げた場合に,「EMiRea」サーバは正しく描画処理を行うことができるのかわからない．そこで，本節ではムービー再生 AP である AP2 と CPU に 5%から 90%まで負荷をかけることのできる負荷タスクを同時に起動させ，負荷による AP2 の描画対象であるウィンドウの描画率と「EMiRea」サーバの CPU 利用率を計測し,「EMiRea」サーバの動作検証を行った．

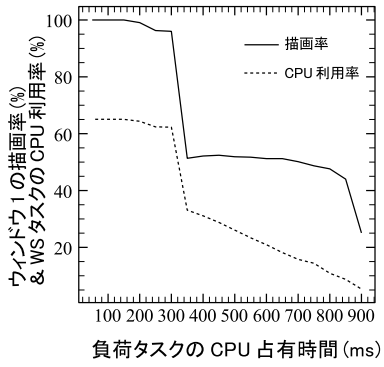

図 10: 負荷をかけた場合の描画率と CPU 利用率

計測結果を図 10 に示す．グラフの横軸は 1 秒あたりの負荷タスクの CPU 占有時間である．

図 10 の結果から，負荷タスクにより CPU の占有時間を上げた場合，負荷タスクよりも優先度の低い「EMiRea」タスクは CPU が割り当てられる時間が減少することがわかる．また,「EMiRea」タスクの CPU 利用率にほぼ比例してウィンドウ 1 の描画率が落ちている．

したがって，非周期的タスクとしてウィンドウシステムを動作させる場合には，AP などの他のタスクにより描画処理を邪魔されない高優先度で動作させるべきであることがわかる．ただし，高優先度でウィンドウシステムのタスクを動作させる場合には，描画処理を行っているウィンドウよりも高優先度なウィンドウを生成した AP が起床した際にウィンドウシステムの処理を一時中断し，AP へ CPU をあけ渡す機能を追加など，描画処理が CPU を占有し続けることで AP へ CPU を割り当てられなくなることを考慮する必要がある．

5 おわりに

本稿では，リアルタイム性を持つ組込みウィンドウシステム「EMiRea」の実現とその評価について示した．

T-Engine に実装した「EMiRea」により，リアルタイム描画処理の有効性の評価を行い,「EMiRea」のリアルタイム描画処理によって時間制約の有効性と，優先度とデッドラインによる描画順序のスケジューリングの有用性を示すことができた．

また，それ以外の課題として，入力関連やリアルタイム通信路の実装，X Window System の Widget のような高レベル API などの AP 開発環境の充実などが挙げられる．

参考文献

[1] *The Microwindows Project*. http://www.microwindows.org/.

[2] *QNX Photon microGUI*. http://www.qnx.com/products/ps_photon/.

[3] *Embedded Graphics XiBase9*. http:://www.xisys.de/.

[4] Knudsen, C.: Troll Tech Announces Embedded GUI Toolkit, *Linux J.*, Vol. 2000, No. 75es, p. 11 (2000).

[5] 御田村晃, 龍本栄二, 芝公仁, 大久保英嗣: リアルタイムオペレーティングシステム Easel におけるリアルタイムウィンドウシステム, 情報処理学会研究報告 2001-OS-87, pp. 153–160 (2001).

[6] RealtimeX グラフィックソフトウェア. http://www.ccur.co.jp/external/TechSup/rtx.html.

[7] 堀口努, 並木美太郎: 組込み用 OS『開聞』におけるリアルタイム機能の開発, 情報処理学会研究報告 2003-OS-92, pp. 91–98 (2003).