2003－OS－93 (17)
2003／5／9

リアルタイム組込みウィンドウシステムの実装

林　　寛　　並木　美太郎
東京農工大学大学院工学研究科

要旨

今日，組込み機器の高性能化や小型化により，携帯電話や家電製品などにオペレーティングシステム（OS）が搭載されるようになった．それに伴い，リアルタイム性を持つグラフィカルユーザインタフェース（GUI）の必要性も増している．そこで，本研究ではリアルタイム性を持った組込みウィンドウシステムの開発を行っている．本稿では，本システムの中心となる描画処理に関して設計と実装について述べる．描画処理に優先度と時間制約を設定することでリアルタイムな描画処理を実現し，また，資源の少ない組込み機器へ実装するため，システム本体サイズおよび実行時の使用メモリを抑えることも考慮した．その結果，描画のリアルタイム処理を実現することができた．

An Implementation of a Real Time Window System for Embedded Systems

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

abstruct

Many devices, such as cellular phones and home electronics, have an operating system(OS), because embedded systems are high-performance and miniaturerized, today. And needs of real time graphical user interface (GUI) are increasing. Thus, we develop a real time window system for embedded systems. In this paper, We describe the design and the implementation of drawing processing that the central part of our system. We realized the real time processing of drawing by priority and guarantee of time settings, and thought to reduce the size of our system and memory with our system use, because our system implement to embedded devices. As a result, we could implement that the drawing processing is real time processing.

１．はじめに

今日，多種多様な機器にマイコン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 だけでなく，携帯電話や e-book[1], さらにはビデオデッキなどの日常的に利用する家電製品などにも組込み用のオペレーティングシステム（以下 OS と記す）が搭載されており，同時に組込みシステムにおけるユーザインタフェースの重要性が増している．特に，携帯電話などは，近年，比較的サイズの大きな液晶を搭載するようになり，インターネットやムービーの再生，ゲームなど，グラフィカルな情報を扱うようになって

きた．情報端末以外にも，電子レンジなどの家電製品にも大型のディスプレイが搭載されるようになり，多彩なメニューを表示したり，料理のレシピをネット上からダウンロードし表示させるなど，グラフィカルインタフェース（GUI）の必要性が高まっている．

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．しかし，GUI に関しては，操作に対して時間内に確実なフィードバックが必要とされることが多いにも関わらず，リアルタイム性を持つものはほとんど無い． そこで，本研究ではリアルタイム性を持つ組込み用 GUI を，リアルタイムウィンドウシステムという形で実現することを目標とする．

２．組込みウィンドウシステムにおけるリアルタイム性

ウィンドウシステムにリアルタイム性を持たせるためには，描画のリアルタイム処理が必要である．通常の OS やウィンドウシステムの描画処理の場合，アプリケーションタスクから発行される描画要求は受け取った順に処理が行われる．そのため，OS がリアルタイム性を持っていたとしても，リクエストキューに処理待ちの描画要求が多数入っていれば，いくら高優先度タスクから発行された描画要求であっても，実際に描画処理が行われるのは後になってしまう．例えば，現在のカーナビはリアルタイム OS を搭載し，画面を二つに分け，経路と各種道路情報などを同時に表示するなど多機能になってきている．今後，カーナビにさらなる付加価値が付き，複数のウィンドウでの多様な情報表示を行うようになってきた場合，経路や位置情報の演算などはリアルタイム OS により優先的に行われるが，実際に画面への描画を行う処理では，他の表示処理が優先されている影響で，一番重要な経路の表示が遅れることが起こりうる．このようなことを防ぐためには，OS だけでなく描画処理に対してもリアルタイム性を持たせる必要がある．

また，いくらリアルタイム性を実現できたとしても，Xwindow などのようにシステムのサイズや実行時に使用するメモリが大きくなってしまっては，資源が制約される組込み機器への実装は行えない．よって，本システムでは，リアルタイム性を実現すると共に本体サイズや実行時の使用メモリサイズはできるかぎり小さくする必要がある．

描画処理以外にも，ウィンドウシステムとアプリケーションの間の通信路や入力などに対するリアルタイム性も必要となるが，今回は，リアルタイムシステムの構築を行うにあたって，基本となるこの描画処理について設計と実装を行った．

３．目標

本研究の目標は，組込みシステム向けのリアルタイム性を持ったウィンドウシステムの実現である．リアルタイム性を持つウィンドウシステムとしては，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[2]や，X window system をベースとして開発された，コンカレント日本株式会社の RealtimeX[3]などがある．しかし，これらのウィンドウシステムは，デスクトップ PC などの高性能で大容量のメモリや二次記憶を持つマシンへの搭載を前提としているため，処理速度が遅くメモリサイズなどが少ない組込み機器への搭載できない．また，組込み機器向けのウィンドウシステムとして，マイクロソフト社の WindowCE[4]や Palm 社の PalmOS[5]のウィンドウシステムがあるが，これらは，OS についてはリアルタイム性を持つが，描画処理については，リアルタイム性を持っていない．また，多様な組込み OS に対応した GUI として PEG[6]などがあるが，やはりリアルタイム OS には対応しているが，PEG 自身はリアルタイム性を持たない．

本システムでは次の目標を設けた．

(1)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．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，リアルタイム性を持たせる．描画処理におけるリアルタイム性としては，ウィンドウを一つの単位として優先度を設定し，ウィンドウの描画処理の時間制約を設定可能とすることで処理の時間保証を行う．

(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 やアプリケーション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．本システムでは，最終的に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 程度とした．また CPU の処理能力についてもデスクトップ PC に比べて低いことが普通であり，処理能力が低くても動作するシステムでなければならない．

4．設計

本システムを実装するにあたり，まず目標の一つである描画処理にリアルタイム性を持たせるために，以下のような特徴を持たせたシステム設計を行った．

(1)各ウィンドウに対し優先度を付加する．

ウィンドウシステムにリアルタイム性を持たせるために，まず描画要求に対して優先度を付ける必要がある．本システムでは，ウィンドウを優先度を付ける単位として処理を行う．ただし，リアルタイム性の実現には優先度だけでなく，次に示す時間保証も必要となる．

(2)描画処理に対して時間保証する．

リアルタイム実現にもう一つ重要なのが描画処理に対する時間保証である．描画要求にデッドラインを設定することで，優先度だけでなく各描画要求の時間制約も考慮したスケジューリングを行うことが可能となる．

(3)上記の特徴を実現するために API を用意する．

優先度やデッドラインの設定を描画要求に対して行うためには，本システム用に API を用意する必要がある．

本章では，まず本システムの全体構成を示し，次に各特徴について詳細を示す．

4.1　システムの全体構成

本システムは，クライアント・サーバモデルである．図 4-1 にシステムの全体構成を示す．

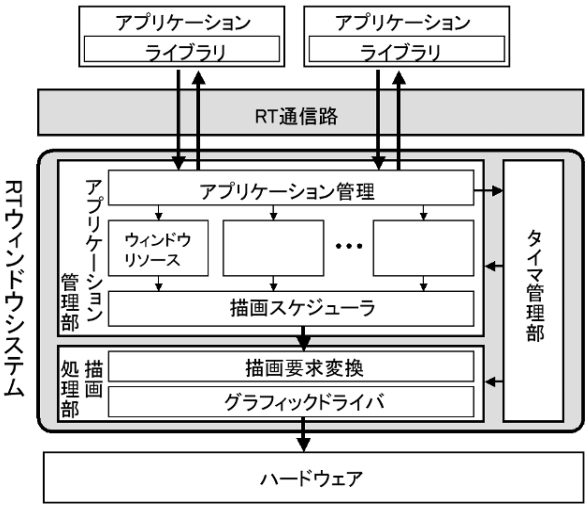

図 4-1　全体構成図

本システムの構成は，ウィンドウシステム本体としてアプリケーション管理部，描画処理部，タイマ管理部があり，さらにアプリケーション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．今回は，ウィンドウシステム本体部分の実装を行った．

4.1.1 アプリケーション管理部

アプリケーション管理部は，アプリケーションとの描画要求およびイベントメッセージのやり取りやウィンドウの管理を行うセクションである．

アプリケーションによりウィンドウが生成されると，空いているウィンドウリソース領域にウィンドウの各種情報を格納し，その領域の ID をアプリケーションへ返す．なお，この時点では，画面上にはウィンドウは描画されない．

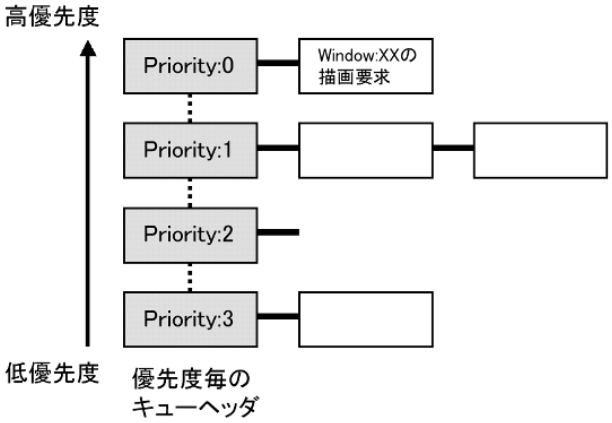

図 4-2　優先度別のリクエストキュー

ウィンドウが生成された状態でアプリケーションからの描画要求を受け取ると，アプリケーション管理部は描画要求に設定されているデッドライン値とウィンドウ ID をタイマ管理部へ渡すと共に，描画要求をウィンドウリソース内に用意されたバッファへ格納する．通常のシステムでは，リクエストキューは一つであるが，本システムではリアルタイムを実現するために，優先度ごとにリクエストキューを用意し，ウィンドウに設定されている優先度のキューへリソースを接続する（図 4-2）．キューへつなぐ際，すでに接続されている要求のデッドラインと比較し，デッドラインが近いものほど先に処理されるように接続を行う．描画スケジューラは，リクエストキューに接続されている要求のうち，高優先度のものから，描画処理部へ渡す．

4.1.2 描画処理部

描画処理部は，アプリケーション管理部から渡された描画要求をグラフィックドライバに対応する形式へ変換し，グラフィックドライバへ渡すセクションである．

4.1.3 タイマ管理部

このセクションでは，リアルタイムの実現のためにもう一つ重要となるデッドラインの管理を行う．アプリケーション管理部から受け取ったデッドラインはウィンドウ ID ごとにタイマへセットし，デッドラインミスが起こると，アプリケーション管理部ならびに描画処理部へ通知する．

4.2　優先度

本システムでは，リアルタイムタスクに付加される優先度とは別に，ウィンドウに独自の優先度を付加し，その優先度に従い描画処理順序を決定する．ただし，基本的にはリアルタイムタスクの優先度とウィンドウの優先度は同じであるのが望ましいが，異なった優先度を設定することもできる．

優先度は 0 から 3 の四段階で，値が小さいほど高優先度となり，基本的にウィンドウ生成時に優先度を設定する. しかし, 優先度は固定ではなく，臨機応変に変化させることが可能である．優先度が変化する条件としては，

(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)アプリケーションプログラマ（以下プログラマと記す）が場合に応じて優先度を変化させる場合

の三つの場合を想定し，設計を行った．

(1)の他ウィンドウとの位置関係とは，対象となるウィンドウが一番手前にあるのかどうかによって，そのウィンドウの優先度が変化する，というものである．つまり，一番手前にあるということは，エンドユーザが必要とするウィンドウであることを意味するため，このウィンドウは一時的に最高優先度のウィンドウとして扱われる．

(2)は，ウィンドウが一番手前にある訳ではないが優先的に描画を行ってほしいものや，逆に一番手前に有るが，描画の優先度が低くてもいいよう

な場合などに，いつでもエンドユーザがそのウィンドウに対して優先度を変更することが可能とするものである．ただし，今回の実装では，入力部分の実装はまだ行っていないため，この機能は現時点では利用できない．

(3)については，プログラマが何らかの理由により，アプリケーションの処理途中にウィンドウの優先度を変更したい場合に，API を用いることにより優先度の変更を行うものである．

4.3 時間保証

描画処理時間を保証するためには，描画要求に対してデッドラインを設定する必要がある．本システムでは，描画要求に対するデッドラインは，一度に描画処理を行いたい描画要求のまとまり，または，一つの要求ごとのどちらでも設定が可能である．例えば，一つ一つの図形の描画に対してデッドラインを設定することや，複数の図形の描画をまとめて一つのデッドラインで設定することもできる．なお，描画処理におけるデッドラインが適用される範囲は，ウィンドウシステムが描画要求を受け取った時点から，グラフィックドライバにより液晶ディスプレイなどのデバイスへ出力されるまでである．

デッドラインミスが起こった場合の処理として，プログラマは，ウィンドウシステム内のデッドラインミスハンドラ（デフォルトハンドラ）を利用するか，またはプログラマが用意したデッドラインミスハンドラを利用するかをウィンドウごとに選ぶことができる．デフォルトハンドラを使用する場合には，下記の(1)〜(4)から処理の内容を選択する．プログラマはウィンドウの生成時や任意の場所でデッドラインミス時の設定や変更を API 関数により行うことが可能である．もし，プログラマが何も設定をしてない状態でデッドラインミスが起こった場合，ウィンドウシステムは，デッドラインミスを起こしたウィンドウの描画処理に対して，以下の四つの処理パターンの中からハンドラが最適と考えられるものを決定し処理を行う．

(1) NOTICE_AND_REDRAW
ウィンドウを生成したアプリケーションに対してデッドラインミスの通知を行い，同時に再描画要求を発行する．

(2) NOTICE
ウィンドウを生成したアプリケーションにデッドラインミスの通知は行うが，その描画は破棄する．

(3) REDRAW
デッドラインミスの通知は行わず，再描画要求のみ発行する．

(4) NO_ACTION
通知を行わず，描画処理を破棄する．

4.4 リアルタイムシステム用 API

リアルタイムシステムを実現するためには，ウィンドウシステム本体だけではなく，アプリケーション側にもそれなりの設定が必要である．つまりプログラマが描画要求などに対し優先度やデッドラインの設定を行う必要があり，それらのプログラミングを考慮した API が必要となる．そこで，本システムは，基本的な図形描画やウィンドウの生成，優先度の変更などの処理を行う API を用意した．

API 関数には，ウィンドウを作成や終了を行うものや，座標の設定や背景や図形の色などを設定する set 関数，実際に図形や文字などの描画行う描画関数，そして，イベントの通知要求などを行う関数がある．

4.4.1 ウィンドウ制御関数

この関数群には，ウィンドウを作成する CreateWindow や終了させる CloseWindow などがある．例えば座標(x,y)に高さ h，幅 w のウィンドウを優先度 p で生成する場合，

```
w_id = CreateWindow(x, y, h, w, p);
```

と記述することにより，ウィンドウが生成され，ウィンドウ ID が返り値として w_id に格納される．

ただし，この時点では，画面上にはまだウィンドウは描画されず，描画関数により描画要求を行うことにより初めて画面上に描かれる．

4.4.2 set 関数

この関数には, 背景色を設定する SetBackColor や，始点座標を設定する SetPoint，デッドラインミス時の処理の設定を行う SetDLM などがあり，引数には，座標や色などの基本情報の他に，ウィンドウ ID を指定する．SetDLM の使用例を以下に示す．

```
SetDLM (w_ID, NOTIC_AND_REDRAW, NULL);
```

この例では，ウィンドウ w_ID に対し，デッドラインミス時の処理としてデフォルトハンドラを利用し通知と再描画要求の発行を行うよう設定している．SetDLM の第二引数は，NOTIC_AND_REDRAW の他に 4.3 で示した NOTICE，REDRAW，NO_ACTION，またはプログラマが用意したハンドラを利用する場合の U_HANDLER のいずれかを入れることにより，プログラマが望むデッドラインミス時の処理を場合に応じて設定を行うことができる. 第三引数は，デフォルトハンドラを用いる場合には NULL を，U_HANDLER を選んだ場合は，プログラマが用意したハンドラへのポインタが入る．任意のハンドラを設定した例を次に示す．この例では，プログラマが作った Dlm_Handler というハンドラをウィンドウ w_ID のデッドラインミスハンドラとして設定している．

```
void Dlm_Handler(){
  /* デッドラインミスハンドラの本体 */
}
      ⋮
SetDLM(w_ID, U_HANDLER, Dlm_Handler);
```

4.4.3 描画関数

描画関数は，リアルタイム性を持たせる上で必要となる描画要求にデッドラインの設定を行う関数である．この関数には，直線や円を描く DrawLine や DrawArc，描画要求を送信する Flush などがある．他のシステムの関数との違いは, 引数として SetPoint で設定された座標に対する終点座標や半径などの図形の情報や描画対象ウィンドウの ID 以外に，デッドラインの値を設定できるところである．例として，直線の描画を行う際のプログラム例を以下に示す．なお，デッドラインを設定する引数を下線で示す．

```
SetPoint(st_x, st_y, w_ID);
DrawLine(ed_x, ed_y, w_ID, dead_line);
```

この例の場合，ウィンドウ w_ID に対して SetPoint で座標(st_x, st_y)に始点を設定し，DrawLine により座標(ed_x, ed_y)までの直線を dead_line ミリ秒までに描画する．

デッドラインの設定は，例のように描画関数内でデッドライン値を設定した場合は，一つの図形の描画要求ごとにデッドラインが設定される．また，複数の図形をまとめて一つのデッドラインに設定したい場合には，各描画関数のデッドライン値に 0 を指定し，まとまりの最後に関数 Flush によりデッドラインを設定する．複数の描画要求によるデッドライン設定を行うプログラム例を以下に示す．DrawLine や DrawArc のデッドライン値の設定は 0 と設定し，最後の Flush 関数の第二引数として，デッドライン値 dead_line を設定することにより直線と円の二つの図形の描画要求をまとめて一つのデッドラインとする．

```
SetPoint(st_x, st_y, w_ID);
DrawLine(ed_x, ed_y, w_ID, 0);
DrawArc(rnd, w_ID, 0);
Flush(w_ID, dead_line);
```

また，以下のようにデッドライン値を 0 と設定している描画関数の間に 0 以上の値を設定した関数があった場合には，デッドライン値が設定され

た関数までに記述されている関数がそのデッドラインの対象となる．以下の例の場合には，③の関数までがひとまとまりとしてデッドライン値 d_line1 が設定され，④の関数が⑤により d_line2 がデッドライン値として設定される．

```
SetPoint(st_x, st_y, w_ID);                    ①
DrawLine(ed_x, ed_y, w_ID, 0);                 ②
DrawLine(ed_xx, ed_yy, w_ID, d_line1);         ③
DawArc(rnd, w_ID, 0);                          ④
Flush(w_ID, d_line2);                          ⑤
```

4.4.4 その他の関数

上記の関数以外に，イベントの通知設定を行う関数 GetEvent などがある．以下に GetEvent の例を示す．この例では，アプリケーションが生成したウィンドウのどれかがアクティブになった場合（FocusIn）と，入力装置の何かしらのキーが押された場合（PressKey）にメッセージを発行する要求を行う．

```
GetEvent(FocusIn | PressKey);
```

5．実装

今回は，ターゲットマシンとして，ロイヤリティフリーの小型携帯ゲーム機であるアクアプラス社の PIECE（図 5-1）を用いて実装を行った．この端末のスペックを表 5-1 に示す．

図 5-1　PIECE（アクアプラス社）

表 5-1　PIECE のスペック

CPU	EPSON S1C33209(24MHz)
RAM	256KB
FlashRAM	512KB
LCD	128x88 FSTN 液晶 モノクロ 4 階調

また，ターゲット OS として，本研究室で開発を行っているリアルタイム組込み OS である「開聞」[7]を使用した．プログラミングは C 言語で行い，一部アセンブラを使用した．図 5-2 に画面表示結果を示す．

図 5-2　実行結果

今回，通常の描画方式と比較するため，描画処理部は同様とする，スケジューラやタイマ処理部を持たない，単一のリクエストキューのシステムを作成し，描画の挙動を比較した．テストアプリケーションとして，三つのアプリケーションを用意し，それぞれ一つのウィンドウを作成し，直線や円を一定間隔で動かすようにプログラミングを行った．三つのウィンドウには，優先度 0～2 を割り当て，共に 20 フレーム／秒(fps)で描画させた．5 秒，10 秒，15 秒，20 秒間描画を行った際の描画された総フレーム数から求めたフレーム数／秒と全体の平均を表 5-2 に示す．各ウィンドウを W1～W3 とし，優先度は W1 が 0，W2 が 1，W3 が 2 とした．

表 5-2　リアルタイム性の有無の比較(fps)

	リアルタイム有			リアルタイム無		
秒	W1	W2	W3	W1	W2	W3
05	20.0	18.2	14.8	18.2	18.3	18.3
10	20.0	18.0	15.2	18.3	18.4	18.4
15	20.0	18.1	15.1	18.3	18.3	18.4
20	20.0	18.1	15.2	18.3	18.4	18.4
平	20.0	18.1	15.1	18.3	18.4	18.4

以上の結果よりリアルタイム性を持たせたシステムは，優先度が低いウィンドウに多少のコマ落ちはあるものの，エンドユーザが必要とする優先度の高いウィンドウにはコマ落ちもなく描画されることが確認できた．それに対し，リアルタイムなしのシステムでは，時間制約がないために三つとも描画フレーム数はほぼ同じであるが，前の描画処理の終了が遅れることや，描画の遅れによって描画要求が溜まるために起こるリクエストキューのオーバーフローなどにより，平均 fps が低くなっている．全体的に見て，リアルタイム性を持たせたシステムの方が画面への表示は安定していた．

6．おわりに

本稿では，組込み機器へのリアルタイムウィンドウシステムの実装について示した．今回の実装により優先度や時間制約による描画処理の制御が可能となった．リアルタイムウィンドウシステムは，描画処理だけでなく，入力や通信路へのリアルタイム性も重要であるため，今後は，描画処理以外に対してもリアルタイム性の実現を課題として取り組む．

参考文献

[1] Larry Press, From P-books to E-books, Communications of the ACM Vol.43 No.5, pp.17-21, May. 2000.

[2] 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, Jun.2001.

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

[4] Windows Embedded home page, “http://www.microsoft.com/windows/embedded/”

[5] Palm.com, “http://www.palm.com/us/”

[6] PEG Portable Embedded GUI Product brief, Swell software, Inc., “http://www.swellsoftware.com/pdfFiles/pegbrief.pdf”

[7] 萱嶋志門, 並木美太郎, 組込み用 OS「開聞」の割込み管理機構, 情報処理学会報告 2000-OS-84, pp.47-54, May.2000.