2005－OS－99（19）
2005／5／27

予測可能リアルタイムカーネルの設計と実装

戴　　毛兵[†]　石川　　裕[††]

従来のリアルタイムシステムでは、割り込み処理と実時間タスクの関係を考慮していないために、割り込みが発生すると、実時間タスクがデッドラインをミスする可能性がある。本研究では、Linuxカーネルを修正して、周辺デバイスからの割り込みを禁止し、デバイスドライバを実時間周期タスクとして実現する。そして、リアルタイムアルゴリズムに基づいた実時間システムを実現する。既存実時間 Linux の一つである ARTLinux と通常 Linux においてタスクデッドラインが守られないような実時間タスクセットにおいて、本研究で実装したシステムでは、デッドラインが守られることを確認する。

The Design and Implementation of a Predictable Real-Time Kernel

Maobing Dai [†] and Yutaka Ishikawa[††]

In a traditional real-time system, when interrupt occurs, there is a possibility of causing a deadline miss because the relation between interrupt processing and real-time tasks is not considered. In this research, the Linux kernel is adapted to a style of prohibiting interrupts from devices and realizing device drivers by periodic tasks. The kernel is implemented based on a real-time algorithm and is confirmed to keep task's deadline time on a real-time task set.

1. はじめに

近年、実時間 Linux カーネルの開発が注目されている。Linux をベースに実時間カーネル化するには、実時間タスクをいかにデッドライン時間内に終了させるかが重要となる。そのため、スケジューラはシステムのすべてのタスクのスタート時間、実行時間、実時間タスクのデッドライン情報を把握しなければならない。これらのタスクの情報の下で、実時間システムはタスクの実行を予測し、正しくスケジューリングして、タスクデッドライン内に実行終了させる。しかし、Linux カーネルでは、タスクの実行は多くのランダムに発生するデバイスからの割り込みに中断される。割り込みが発生すると、デッドライン時間に近付いているタスクがあるにもかかわらず、現在実行中のタスク実行が中断され、割り込み処理が実行される。このため、デッドラインをミスする可能性がある。本研究では、これまでの一般的な実時間 Linux カーネルと違い、デバイスからの割り込みを禁止し、デバイスドライバーを一つの実時間周期タスクにする方法を採用する。このようにすると、カーネル内のランダムな実行時間が短縮され、タスクの実行時間の予測性が保証される。

Linux システムタイマの一つの問題は時間精度である。普通の Linux では 1ms の単位でタスクが実行されている。しかし、ロボットなどのセンサーからの情報を取り込む実時間タスクの要求は 1ms より精度が高いときがある。ARTLinux[8] では、システムを動かす前に、ユーザが前もってシステムの頻度 (Hz) を静的に設定している。本研究では、システムタイマー精度を動的に変更する仕組みを導入する。

Linux カーネルを実時間カーネル化する時、スケジュールを考慮しなければならない。これまで多くのリアルタイムスケジューリングアルゴリズムが提案された。たとえば、タスクの周期時間によって、静的に優先度をつける RM(Rate Monotonic)[1] アルゴリズムと、タスクのデッドライン時間によって、動的に優先度をつける EDF(Early Deadline First)[2] アルゴリズムがある。周期実時間タスクと非周期実時間タスクが混在している環境では、スケジューラがすべてのタスクの要求を満たすことは NP-hard 問題となる。周期タスクの基で非周期タスクを実行するアルゴリズムとして、Jay K.Strosnider の DS(Deferrable Server)[4] ア

† 東京大学情報理工学研究科コンピュータ専攻
Deparment of Computer Science, University of Tokyo
†† 東京大学情報理工学研究科
Graduate School of Information Science and Technology, The University of Tokyo

ルゴリズム、また、Brinkley Sprunt の SS(Sporadic Server)[5] アルゴリズムが提案されている。本研究では、スケジューラは実時間スケジューラと Linux 本来のスケジューラとに分けている。スケジューラは EDF とタスクのスタート時間を考慮して作られている。

本研究で実装したカーネルを用いて以下の実験結果が得られた。1ms タイムより精度が高いタスクがある場合でも、カーネルは正しく時間制約を満たすようにスケジューリングしていることを確認した。三つの実時間タスクにおいて、ART-Linux と一般的なカーネルではデッドラインミスしたことに対して、実装したカーネルでは、すべてデッドライン内に実行が終了した。よって、実装したカーネルは実用的であり、その有効性が確かめられた。

2. 既存リアルタイムアルゴリズムと実時間カーネル

本章では、リアルタイムアルゴリズムと FSMLabs 社の RTLinux[6][7] と ARTLinux について述べる。この他にも実時間化した Linux には KURTLinux[10]、Linux/RK[9]、SRTLinux[11] などがある。

2.1 RM と EDF

RM アルゴリズムは、1973 年 Liu と Layland によって提案された。このアルゴリズムは、タスクの周期時間によって、タスクに優先度を割り当てる。RM アルゴリズムにおいて、全周期タスクの CPU 使用率が最終的に 0.69 まではスケジューラブルであることが知られている[3]。

EDF アルゴリズムは 1974 年、Horn によって提案された。この方法は、デッドラインが早いタスクに高い優先度が割り付けられる。EDF では CPU の使用率が 100 %使用できることも知られている。RM と較べて、EDF は CPU 使用率が良いだけでなく、コンテキストスイッチを減らすことが可能である。本研究では、全ての実時間タスクは周期タスクであることを想定しているが、スケジューリングアルゴリズムとしては EDF アルゴリズムを採用する。

2.2 DS と SS

リアルタイムシステムに混在している周期タスクと非周期タスクを、すべて周期タスクのように見なすアルゴリズムとして、DS と SS がある。

DS アルゴリズムは Jay K.Strosnider によって提案された。周期タスクが非周期タスクのサーバのように動く。非周期タスクがある場合、そのサーバが持つ CPU 時間 (キャパシティー) を非周期タスクに分配する。

SS アルゴリズムは 1990 年 Brinkley Sprunt 氏によって提案された。SS アルゴリズムのサーバの起動時間が非周期タスクの到着時間によって、動的に変えることになっている。このため、非周期タスクのサービス確率が増加し、CPU 使用率が向上する。

本研究では、Linux のスケジューラのほかに、リアルタイムスケジューラを作る。リアルタイムスケジューラは EDF アルゴリズムの下で、非周期タスクも考慮して、スケジューリングしている。

2.3 RTLinux

RTLinux は FSMLabs 社の Victor Yodaiken によって開発された実時間カーネルである。RTLinux の実時間タスクは直接ハードウェアにアクセスできるため、ユーザからハードウェアが保護されていない。また、割り込みも禁止していないため、ネットワークなどの割り込み頻度が高い環境では、実時間タスクがデッドラインをミスする可能性がある。

RTLinux のタイマ精度の処理では、one-shot モードが採用されている。one-shot モードとは、実時間タスクのタイマ精度変更要求があると、それにしたがって、タイマチップをセットセットすることである。RTLinux では、実時間タスクの時間精度を最大限に 1ms まで動的に設定できる。しかし、タイマチップの設定回数が多くなり、システムタイマ割り込み処理のオーバーヘッドが増大する。

このように、RTLinux は実時間タスクの要求をある程度満たす。しかし、ハードウェアデバイスからの割り込みが禁止されているため、実時間タスクがデッドラインをミスする可能性がある。

2.4 ARTLinux

ARTLinux は産業技術総合研究所の石綿氏によって、1995 年に開発されたリアルタイムカーネルである。

ARTLinux では、実時間周期タスクおよび固定優先度スケジューラが実現されている。これらの拡張を用いて RM アルゴリズムを実現できる。しかし、周期タスクセットが与えられた時に、ARTLinux 側でタスクの優先度を決めず、優先度はユーザの設定に任されている。

ARTLinux では、デバイスドライバが周期タスクとして実現されているが、その優先度は全て一番高い優先度として設定されている。また、ハードウェアからの割り込みのタイミングで、周期タスクに割り込みが発生したことを通知している。

ARTLinux のタイマ精度はカーネルをコンパイルするとき、決めなければならない。つまり、RTLinux のタイマ精度は静的であり、実時間タスクの要求に応

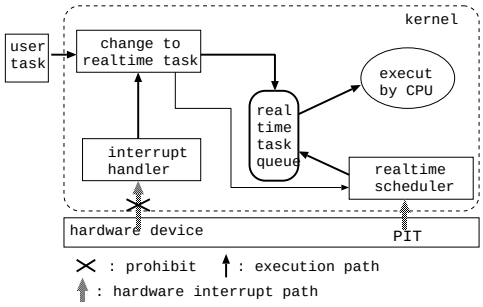

図 1　全体の構造

じて変更できない。

3. 設　　計

第２章では、既存の Linux リアルタイムカーネルと実時間スケジューリングアルゴリズムを見てきた。この章では、本研究での実時間カーネルの設計を述べる。

3.1　全体の設計

前述したように、予測可能なリアルタイムカーネルを実現するには、実時間タスクの実行予測性を把握しなければならない。本研究でのカーネルの設計はこれを中心に割り込み処理、リアルタイムスケジューラ、時間精度、ユーザとのインタフェースなどを設計する。全体の構造は**図 1** のようになっている。

3.1.1　割り込み処理

割り込みは実時間タスクの実行時間の予測に影響を及ぼすため、ハードウェアデバイスからの割り込みを禁止する。ただし、タイマ割り込みはシステムを動かす基礎となるため、タイマ割り込みだけは許可する。タイマ割り込みによる他のタスクの実行に与える影響を最大限に抑えるために、タイマ割り込みルーチンではしすてむの基礎時間 jiffies を増加させる処理だけを行うものとする。大域システムタイマの維持はタイマ割り込みハンドラを周期タスクに変えることによって実現する。**図 2** に割り込み処理を示す。

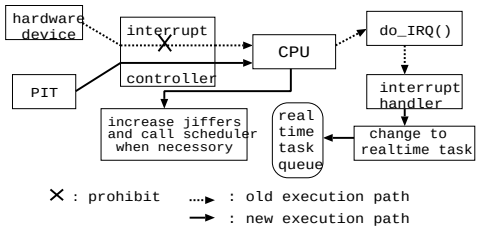

図 2　割り込み処理

3.1.2　スケジューラ

Linux をベースに実時間カーネルを作るには、周期タスク、非周期タスク、一般的な Linux プロセスの三種類に対応しなければならない。そのため、スケジューラを二つに分ける。一つは一般 Linux プロセスを扱う元の Linux スケジューラである。もう一つは実時間タスクを扱う実時間スケジューラである。実時間タスクがある場合、リアルタイムスケジューラが呼び出され、実時間タスクを優先的に実行させる。

- リアルタイムスケジューラの階層化
  リアルタイムスケジューラを二つの階層で実現する。低レベル層では、タスク選択、タスクの次の優先度の設定、エンキュー、タスクスイッチ 等基本機構が定められる。高レベル層では、低レベル層で定められる機構を用いて、様々なスケジューリングポリシーが実現される。高レベル層はユーザがカーネルモジュールとして、定義することができる。
- リアルタイムスケジューラアルゴリズム
  ディフォルトリアルタイムスケジューラのアルゴリズムは EDF を採用する。カーネルは周期タスクの一周期実行終了後、次の周期が始まるまで、実行権を他の実時間タスクに渡すか、元の Linux スケジューラを呼び出して、一般タスクの実行を行う。
- スケジューラ呼び出しタイミング
  リアルタイムスケジューラは三つのタイミングで呼び出される可能性がある。(a) 新しい実時間タスクがカーネルに入るとき、(b) 実時間タスクの一周期実行終了後、(c) タイマ割り込み時に、スタートタイムになった実時間タスクがあるとき、の三つである。

3.1.3　時 間 精 度

実時間タスクの周期が 1ms より小さい場合、システムタイマチップの割り込み頻度を再設定する。カーネルの時間精度を最大 100 マイクロ秒まで設定できるようにした。

カーネルに複数の実時間タスクの時間精度が 1ms より小さい場合は、もっとも時間精度の高いタスクの要求に設定する。一旦設定すれば、カーネルに高精度タスクがなくなるまで、維持される。高精度タスクがすべて終ったとき、タイマチップの時間精度を元とおりの値に設定する。

3.1.4　実時間タスク

実時間タスクの制御は、大きく以下の部分に分かれる。

- 実時間タスクへの変更時
  通常 Linux プロセスは、第 3.2 節で述べる rt_enter() 関数を呼び出すことにより実時間タスク化する。rt_enter() 関数引数で渡されるタスクスタート時間、デッドライン時間、周期時間は、CPU 時間に変換され、該当タスクが実時間ランキューに入る。タスクスタート時間が現在のタイムであれば、直ちにリアルタイムスケジューラが呼び出される。
- 一周期実行終了時
  実時間タスクが一周期実行を終り、周期タスクのカウンターと状態を設定した後、現在実時間タスクの実行要求があるかどうかを検査する。要求がある場合、リアルタイムスケジューラを呼び出す。なければ、Linux のスケジューラを呼び出す。
- 通常タスクへの変更時
  実時間タスクの実行が全て終って、終了段階に入るとき、実時間ラインキューから取り出される。後は非実時間ランキューにエンキューして、Linux のスケジューラを呼び出す。

3.1.5 ユーザとのインタフェース

ユーザタスクを実時間タスクに変えるためのシステムコールが提供される。また、ユーザが実時間タスクの実行がデッドライン内で実行しているかどうかをチェックするための機構として、Linux の proc ファイルシステム上に情報が提供される。ユーザがカーネルスケジューリング動作を定義するための Linux ローダブルモジュール API が提供される。

3.2 定義したシステムコール

rt_enter()：

タスクのスタート時間、デッドライン時間、周期時間を CPU 時間に変えた後、タスクを実時間実行待ち行列に入れる。タスクにスタート時間が 0 である場合、直ちにスケジューラを呼び出す。

rt_next_period()：

タスクが一周期の実行を終了した後、システムの状態を検査して、リアルタイムスケジューラか、Linux のスケジューラを呼び出す。

rt_exit()：

実時間タスクの終了操作をする。主に実時間実行待ち行列からタスクを取り出して、Linux のランキューにエンキューする操作である。

3.3 スケジューラ API

ユーザが独自のスケジューリング動作を定義するには、ローダブルモジュールを利用する。API は次の三つがある。

get_next_task()：

実行待ち行列から、次に実行すべきタスクを取り出す。

set_next_priority()：

タスクに対して、次の優先度を決める

rt_enqueue()：

タスクに対して、実行待ち行列の適当な位置を決め、エンキューする。

4. 実　　装

第 3 章では、Linux をベースにリアルタイムカーネルの設計を述べた。この章では、Intel 社 x86 系の基で、実際どのように実装したのか、具体的に述べる。

4.1 割り込みの禁止

第 3 章で述べたように、実時間タスクに影響を与えないように、タイマ割り込み以外のデバイスドライバーからの割り込みを禁止する。

4.2 タイマ割り込み処理ルーチンの設定

Linux の割り込み処理の最初のルーチンは interrupt[i] である。しかし、本研究では、独自のタイマ割り込み処理ルーチンを定義する。

タイマ割り込み処理ルーチンは 3 章で述べたように、システムの基礎時間 jiffies の増加を維持するので、カーネルの他の実時間タスクの実行を影響を最大限に抑えている。

4.3 デバイス割り込みハンドラの実時間タスク化

デバイス割り込みハンドラを実時間タスクに変換するには、Linux 割り込みハンドラ登録データ構造 irq_desc[]->action を利用する。デバイスがアクティブになるために、デバイス割り込みハンドラを構造体 struct irqaction *action に登録する必要がある。したがって、デバイス割り込みハンドラを実時間タスクに変換するには、ハンドラを action に登録した時点で周期タスクリストに登録する。

4.4 リアルタイムスケジューラの実装

リアルタイムスケジューラは主にタスクのランキュー操作を行う。実時間実行待ち行列に格納されている構造体には、タスクのリンクリストメンバー以外、タスクカウンター、ビットマップ変数、高精度タスクカウンター、次のスタート CPU 時間などの構造体メンバーがある。

リアルタイムスケジューラの実装は大きく分けて、(a) 実行するタスクの選択 get_rt_next()、(b) 選択したタスクの優先度の再設定 set_next_priority()、(c) 選択したタスクのエンキュー rt_enqueue()、(d) タスクスイッチ の四つの部分から構成される。

4.5 実時間タスクの実装

実時間タスクの全ての情報はタスク構造体 struct task_struct のメンバー realtime_task に格納される。

```
struct task_struct {
  ... ...
  realtime_task_t * realtime_task;
  struct list_head realtime_queue;
 }
```

ここでの realtime_queue はリアルタイムランキューのリストヘッドとなっている。

一つの実時間タスクの実装は三つの部分に分ける。タスクのタイム情報を CPU 時間に変換して、ランキューに入る部分 rt_enter()、一周期の実行が終って次の周期を入る部分 rt_next_period()、タスクが終る部分 rt_exit() になっている。全体の構造は**図 3** のようになっている。

```
1  static int realtime_task(u64 start_time,u64 period_time,
                u64 deadline_time, int count)
2  {
3    /* ... ... */
4   rt_enter(start_time,period_time,deadline_time,count);
5   do {
6
7    /* actual task processing; */
8
9    rt_next_period();
10
11    /* judgment of re-execution; */
12
13  } while (1);
14
15   /*  initialization of the variable */
16
17   rt_exit();
18    /* ... ... */
19   return 0;
20 }
```

図 3 realtime_task() 関数

4.6 時間精度の設定

時間精度の設定は PIT(Programmable Interval Timer) チップに対して行う。

5. 実験と考察

本章では、第 5 章で実装したリアルタイムカーネルの実験と性能評価を述べる。実験は、Linux-2.6.5 と ART-Linux と実装したカーネルの 3 種類を用いる。

5.1 割り込みハンドラの遅延

まず割り込みハンドラの遅延を測定する。本研究で実装したカーネルは、割り込みが禁止され、デバイスハンドラは周期タスクに変更して、デバイスからのデータを読み取る。

次に hdparm コマンドを使って、ハードディスクに対して、DMA(Direct Memory Access) 禁止した場合と禁止してない場合における両方のカーネルでの速度を測定した。また、デバイスハンドラの周期を 1ms と設定している。**表 1** にまとめる。

実験は 10 回の平均値である。表 1 から分かるように、実装したカーネルの HDD アクセス速度は普通の Linux カーネルとほとんど同じであることが分かった。

5.2 スケジューラ性能

スケジューラ性能はカーネル CPU クロックカウンタを利用して測定される。方法は次のようになっている。まず、スケジューラの最初に現在の CPU クロックを取得する。次に、スケジューラが実行すべきタスクを選択してからタスクコンテキスト直前にもう一度 CPU クロックを取る。最後に、この両方の時間差を 10000 回測定して、平均値を求める。実験は 8 個の実時間タスクの基で測定した。その結果：Linux-2.6.5 の O(1) スケジューラは平均 159ns の CPU 時間を要することに対して、実装した実時間スケジューラは平均 156ns であった。性能は普通の Linux とほぼ同じであることが確かめられた。したがって、実装したカーネルの実時間スケジューラは実用には問題がないと確認できる。

5.3 時間精度の確認

現在の Linux の時間精度は 1ms となっているが、実時間タスクの時間精度はそれ以下にしなければならない場合がある。実時間タスクの周期は 500 マイクロ秒の場合、タイマーチップのカウンターは 2000 と設定される。

5.4 リアルタイムスケジューリング機能の確認

実装したカーネルの実時間スケジューラがリアルタイムアルゴリズムにしたがって、タスクをスケジューリングしているかどうかを確認する必要がある。実験は**表 2** のタスクセットを用いて行う。

Linux カーネル、ART-Linux カーネル、実装したリアルタイムカーネルでの実行結果を**表 3** に示す。

Linux では、プロセスの実行はタイムシェアリングの方式で実行しているため、タスク 2 がデッドラインミスする。ART-Linux では、周期タスク実行機構があるが、ユーザが陽に明示的に優先度を設定しないとタスク優先度が同じと見なされる。このため、タスク 2 がデッドラインミスする。一方、実装したカーネル

表 1 HDD アクセス速度 (MB/SEC)

kernel	DMA	speed
実装	あり	20.37
実装	なし	2.44
Linux	ある	20.22
Linux	なし	2.42

表 2　実時間タスクセット

タスク	実行時間 (ms)	開始 (ms)	デッドライン (ms)
1	0.64	0	10
2	2.452	0	5
3	3.751	0	8

表 3　タスク実行結果 (単位：ms)

タスク	デッドライン	Linux	ART	実装カーネル
1	10	4.52	4.451	7.172
2	5	5.362	5.212	2.551
3	8	6.722	6.651	6.41

では、EDF アルゴリズムにしたがって、先にタスク 2 が実行される。そのあと、タスク 3 とタスク 1 が実行される。終了時間がすべてデッドライン時間内に収まる。

以上の測定の結果から、本研究で実装したリアルタイムカーネルはデバイスからのデータを読み取る場合、普通の Linux カーネルと比べて、ほとんど差がないことが確認された。実時間タスクに対して正しくスケジューリングされ、要求のデッドライン時間内にタスクの実行が終了した。また、動的に実時間タスクの時間精度要求も満たしていて、効率良く実時間タスクをスケジューリングしていることが確認された。

6. 終 り に

本論文では、Linux をベースに予測可能実時間カーネルを実現するために、ハードウェアデバイスからの割り込みを禁止し、割り込みハンドラを実時間タスクに変換した。さらに、EDF アルゴリズムにしたがった実時間スケジューラを実現し、動的に実時間タスクに対応できるようにした。また、高精度実時間タスクの要求に備えて、カーネルが動的にシステムの時間精度を変えられるようにした。実装したカーネルに対して、性能とスケジューラビリティを確認した。その結果、精度が高いタスクでも、カーネルが正しく設定された。実時間タスクセットに対して、すべてデッドライン内に実行が終了していることが確認された。よって、実装したカーネルの有効性が確かめられた。

今後の課題としては、実時間タスクの数の増加にしたがって、カーネルランキュー操作のオーバーヘッドを抑える必要がある。また、実装したカーネルのマルチスレッドアーキテクチャや、SMP 環境の対応、分散環境の対応も課題として残されている。

謝　　辞

本研究の一部は、科学技術振興機構 (JST) の戦略的創造研究推進事業 (CREST) の支援を受けました。

参 考 文 献

1) C. W. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard real time environment”, Journal of the ACM, 20(1):46-61, 1973
2) W. Horn, “Some simple scheduling algorithms”, Naval Research Logistics Quarterly, 21, 1974
3) J. P. Lehoczky, L. Sha, and Y. Ding ,”The rate monotone scheduling algorithm: Exact characterization and average case behavior”, In Proceedings of the 10th Real-Time Systems Symposium, page 166-171, IEEE Computer Society Press, December 1989
4) J. Strosnider, J. Lehoczky and L. Sha, “The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments”, IEEE Transactions on Computers, Vol.44, No.1, January 1995
5) B. Sprunt, J. Lehoczky and L. Sha, “Aperiodic task scheduling for hard real-time systems”, Real-Time Systems, 1(1):27-60, 1989
6) M. Barabanov and V. Yodaiken, “Introducing Real-Time Linux”, Linux Journal, February 1997.
7) V. Yodaiken, “The RTLinux Manifesto”, In Proc. of the 5th Linux Expo, Raleigh, NC, March 1999
8) 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, Vol.25, No. 11, 1999
9) S. Oikawa and R. Rajkumar “Portable RK: A Portable Resource Kernel for Guaranteed and Enforced Timing Behavior”, In Proceedings of the IEEE Real-Time Technology and Applications Symposium Vancouver, June 1999
10) B. Srinivasan, S. Pather, and D. Niehaus, “A Firm Real-Time System Implementation Using Commercial Off-The-Shelf Hardware and Software”, In Proceedings of IEEE Real-Time Technology and Applications Symposium, June 1998
11) Ingram, D, “Integrated Quality of Service Management”, University of Cambridge Computer Laboratory Technical Report, No. 501, 2000