システムソフトウェアとオペレーティング・システム　86－18
（2001．3．6）

ITRON仕様OSのマルチプロセッサ拡張における デッドロック回避の手法

辰巳　将司　石川　知雄　宮内　新
武蔵工業大学　情報通信研究室
〒158-8557 東京都世田谷区玉堤 1-28-1
TEL 03-3703-3111 内線 2974
FAX 03-5707-2179

高田　広章
豊橋技術科学大学　情報工学系
〒441-8580 愛知県豊橋市天柏町雲雀ヶ丘 1-1
TEL0532-44-6752
FAX 0532-44-6757

E-mail: tatumi@ic.cs.musashi-tech.ac.jp

ITRON仕様OSでは，リアルタイム性を向上させるために，ロックがプロセッサごとにタスク，オブジェクトの2種類に分けられている．多くのシステムコールは，タスクロックのみ，もしくはオブジェクトロック→タスクロックの順でロックをとる（ロックをとる順序が決まっている）のでデッドロックは起こらない．しかし，いくつかの命令では，性質上，上記の通常の命令とは逆の順序でロックをとる．これらの命令をマルチプロセッサに対応させる場合に，シングルプロセッサと同様のアルゴリズムで安易に実装してしまうとデッドロックが生じる可能性があるため，回避方法の検討が必要である．そこで本稿では，このデッドロックが存在するケースに着目し，デッドロックが生じる過程を明らかにし，その対策を示す．

リアルタイムカーネル　デッドロック　マルチプロセッサ

The technique of the avoiding deadlock in multiprocessor extension of ITRON specification OS

Masashi Tatsumi, Tomo Ishikawa, Arata Miyauchi
Information and Communication Labo.
Musashi Institute of Technology
1-28-1 Tamatsutsumi, Setagaya-ku,
Tokyo 158-8557, Japan
TEL +81-03-3703-3111(2974)

Hiroaki Takada
Dept. of Information and
Computer Sciences,
Toyohashi Univ. of Technology
1-1 Hibarigaoka, Tempaku-cho, Toyohashi
Aichi 441-8580, Japan
TEL+81-532-44-6752

E-mail: tatumi@ic.cs.musashi-tech.ac.jp

In ITRON specification OS, in order to raise real-time property, the lock is divided into two kinds, a task and an object, for every processor. And since the order that almost all systemcalls take locks was decided, deadlock does not happen. However, the order that some systemcalls take locks differs. When making these systemcalls correspond to a multiprocessor, the deadlock may happen.

Then, the purpose of this research is clarifying process a deadlock's happening and coping with it so that the deadlock may not happen.

real-time kernel, deadlock, multiprocessor

1 はじめに

近年要望が高まっている高性能なリアルタイムシステムでは，多数の入出力装置等がシステムに接続され，それらからの事象に対して決められた時間内にレスポンスを返す必要がある．このような要求を満たすためのアプローチとしてマルチプロセッサシステムの採用が挙げられる．

マルチプロセッサシステムの特徴は各プロセッサごとに並列に実行できることで，比較的大規模なデータベースサーバ等で多人数が同時に利用する場合や，多数の入出力装置等が接続されたシステム等ではシングルプロセッサでシステムを構築するよりも効率良く処理することができる．

このようなマルチプロセッサシステムにおいて，システムに接続する入出力装置の増設などにより，プロセッサ数を増やす必要がある場合，システムの設計変更を最小限にとどめたいという要求がある．言い換えると，システムがプロセッサ数の増加に対してスケーラビリティ（拡張性）を持つ，ということが求められる．

マルチプロセッサに対応させたITRONである，ITRON-MP の開発において，このリアルタイム性とスケーラビリティの2つの要求を満たすために様々な研究が行われてきた[1], [2]. ITRON-MP は現在，開発段階で様々な研究が行われているが，それらの研究を行う上でもまずは基本的なシステムコールの実装が必須の課題だと思われる．

ITRON ではスケーラビリティ，リアルタイム性を向上させるために，ロックがプロセッサごとにタスク，オブジェクトの2種類に分けられている．このタスク，オブジェクトはプロセッサごとにあり，他のプロセッサからもアクセス可能である．

多くのシステムコールは，タスクロックのみ，もしくはオブジェクトロック → タスクロックの順でロックをとる（ロックをとる順序が決まっている）のでデッドロックは起こらない．

しかし，タイムアウト処理，rel_wai（待ち状態の強制解除命令），chg_pri（タスク優先度の変更）では，命令の性質上，上記の通常の命令とは逆の順序でロックをとる．これをマルチプロセッサに対応させる場合に，シングルプロセッサと同様のアルゴリズムで安易に実装してしまうとデッドロックが生じる可能性があるため，回避方法の検討が必要である．

そこで本研究では，このデッドロックが存在するケースに着目し，デッドロックが生じる過程を明らかにし，その対策を検討することを目的とする．

2 研究環境

本研究では ItIs/MP という ITRON-MP 仕様のマルチプロセッサ用リアルタイム OS を使用する．また，実験には以下のような装置を用いる．

- DVE-68K/40　6枚

  CPU に MC68040 プロセッサ，ローカルメモリを 4MB 搭載した CPU ボードであり，これを6枚用いる．

- DVE-401 1枚

  共有メモリボードで，8MB のメモリを搭載している．

この 6 枚の CPU ボードと共有メモリが VME（Versa Module Europe）バスを通して接続されているマルチプロセッサシステムを用いる．

3 ItIs/MP の基本構造

3.1 TCB（タスクコントロールブロック）の構造

タスクを管理するための主な情報にはタスク ID（タスクの名前），タスクの状態，タスクの優先度などがある．これらの管理情報が TCB（タスクコントロールブロック）に格納される．TCB には，この他，タスクのスタートアドレスやスタックに関する情報の格納領域，タスクの使用するレジスタのセーブ領域などが含まれる．

タスクIDは，システムコールを使ってそのタスクをアクセス（操作）するときに，パラメータとして利用される．また，タスクの状態はタスクの状態遷移に関係し，タスクの優先度はスケジューリングに関係する．このように，タスクの管理はすべてこのTCBの情報に基づいて行われる．TCB内の情報は，タスクのいろいろな動作に深く関係した重要な情報である．

3.2 タスク，セマフォの構造

3.2.1 タスクの構造

タスクのスケジューリングは，タスクの優先度を基準にして行われ，実行可能状態（READY）にあるタスクの中で最も高い優先度を持つタスクが実行状態（RUN）となる．実行可能状態および実行状態のタスクは，タスクの優先度順に行列を作っており，その行列（レディキュー）の先端のタスクが実行状態となる．

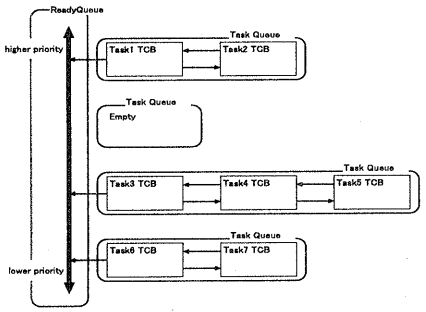

図 1: レディキューの構造

3.2.2 セマフォの構造

セマフォは主に資源の排他利用に使用される．すなわち，非同期に動作する複数のタスク同士が同一の資源を共有する場合にそれを矛盾無く操作するために使用される．

図2で示すように，セマフォの方にはそのセマフォに対する待ちキューが作られ，先頭のタスクがレディキューに繋がっている．そして，そのタスクが処理を終えると待ちキューの次のタスクが先頭に来て待ち状態を解除され，レディキューに繋がる．

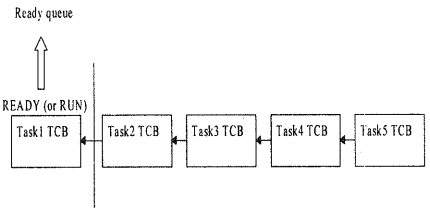

図 2: セマフォキューの構造

4 デッドロック

4.1 ロック単位とデッドロック

ここで，ロックをする単位とデッドロックの関係について説明する．

- すべての資源をロックする場合

  デッドロックは起こらないが未使用の資源もすべてロックしてしまうので効率は悪い

- プロセッサごとにロックをする場合

  上記の場合よりは効率がいいが，図3のような単純な操作でもデッドロックが生じてしまう．また，未使用の資源も一緒にロックしてしまうので効率は良くない

- 各プロセッサでタスク，オブジェクトに分けてロックをする

  命令がロックをかけていく順番（流れ）が決まっていれば，デッドロックは起こらないが，もし，ロックをかけていく順番が異なる命令がある場合，デッドロックが生じる可能性がある

今回の研究で使用するItIs/MPでは，リアルタイム性を向上させるために，各プロセッサでタスクとオブジェクトの2つ分けてにロックをとっている．また，ロックの順序が異なる処理もあるため，デッドロックが生じる可能性がある．

次にそのロックの順序が異なる処理について説明する

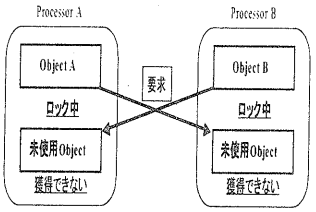

図 3: デッドロックが生じる場合

4.2 通常の命令と特殊な命令のロック順序

4.2.1 通常の命令のロック順序

- 例　sig_sem　　セマフォの資源返却命令

  まずセマフォキューをロックし、そこから指定されたタスクをロックして処理をする.

  このようにセマフォ → タスクの順でロックをとる.

4.3 特殊な命令のロック順序

- 例　rel_wai　　強制的な待ち状態の解除

  タスクをロックしてそのタスクの状態や何を待っているのかを調べ, その調べた先(セマフォキュー等)をロックする. よってロックの順序が通常の場合と入れ替わる. また, 途中でロック先が分かるので必要な資源をあらかじめロックしておくこともできない.

同様にタイムアウト処理や chg_pri (優先度の変更) の場合もまず対象となるタスクをロックしてその内容を見なければならないため, ロックをとる順序が逆になる.

この3つの命令のみがロックをとる順序が逆であり, これらの命令がデッドロックの原因のなっている. 本研究では, これらの3つの命令がいかなる場合に発行されてもデッドロックが生じないように対策するのが目的である.

5 デッドロック

5.1 デッドロックが起こる仕組み

デッドロックが起こるのは, 通常の命令とロックをする順序が逆の命令が衝突を起こした場合である.

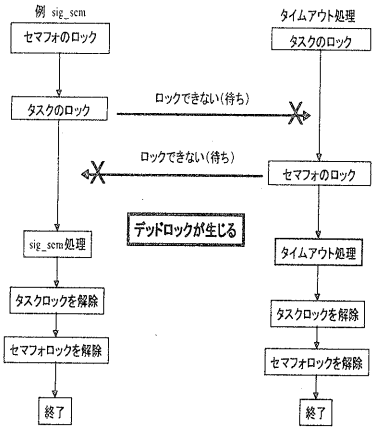

図 4: デッドロックが生じる例

通常の命令 (sig_sem 等) が, semaphore → task の順で2重にロックをするのに対して特殊な命令 (rel_wai, chg_pri, タイムアウト処理) は task → semaphore の順でロックするので, この命令が同時に発行された場合にデッドロックが生じる可能性がある.

5.2 デッドロック回避策

ロックの逆順によって生じるデッドロックの回避策を示す.

図5において, (B) をロックしたあとにタスク状態を見て記憶し, その後アンロック (B) してから (A) をロックする. そして (B) をロックする際, タスク状態が変わっていたらリトライし, 変わっていなければそのままロックし, その後 (B), (A) とアンロックする. このように処理を行えばデッドロックは回避できる.

この場合では2つの命令が同時に発行された場合を示しているが, 基本的には3つ以上の複数の命令が同時に発行された場合でも,

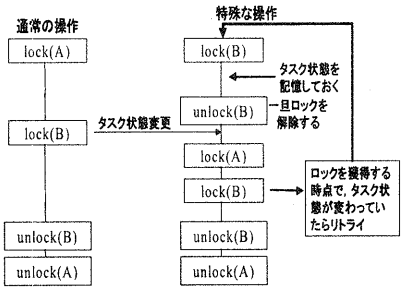

図 5: デッドロック回避策

ロックをとる順がすべて統一されているため，デッドロックが生じるようなことはない．

また，この場合は sig_sem 等の通常の命令を優先させている．rel_wai の様な特殊な命令の方を優先させ速やかに発行させる方が良いという考え方もあるが，ここでは使用頻度の高い通常の命令の方の処理を単純にし，優先させることでスループットの低下を抑える，という考えで通常の命令を優先させる方式を示した．

5.3 リトライしない方法

前節で示したリトライするという方法は，デッドロックは回避できるがリアルタイム性を損なわないとは言い切れない．そこでリトライをせずにデッドロックを回避する方法を検討してみる．

ここではタイムアウト処理，rel_wai，chg_pri の各命令ごとに分けて検討する．

5.3.1 タイムアウト処理の場合

そのデッドロックをリトライすることによって回避するアルゴリズムが図 6 である．sig_sem 命令の方では先ほどと同じくセマフォをロックして，その後タスクをロックしようとする．一方タイムアウト処理の方では，タスクをロックして，その状態を記憶した後，一旦タスクロックを解除する．それにより，sig_sem 命令はタスクをロックすることができ，処理を終えることができる．そして，タイムアウト処理の側はもう一度セマフォ，タスクのロックをとって処理を進めようとするが，sig_sem 命令によって，タスクの状態が以前に記憶した状態とは異なる状態になってしまっているので，改めてはじめから処理をやり直す（リトライする）．

次に，リトライをしないでデッドロックを回避する方法を説明する．

図 7 の様に，はじめにタスクをロックし，そのタスクがセマフォキューに繋がっていた場合には一旦タスクのロックを解除する．そしてセマフォをロックし，タスクをロックする．ここまでは上記の処理と同じである．ここで他の処理が割り込んでいたかを調べるためセマフォキューにまだ繋がっているかをチェックする．まだ繋がっているのならばそのままタイムアウトの処理（待ち解除）してロックを解除する．もしすでにセマフォキューには繋がっていなかった場合は，その命令はすでに他の処理によって割り込まれ，待ち状態を解除されている（割り込まれた処理によって起こされた）ということなのでタイムアウト処理は不要となる．従って，何もせずにそのままロックを解除して処理を終了する．このように処理することにより，デッドロック回避のためのリトライをする必要が無くなる．

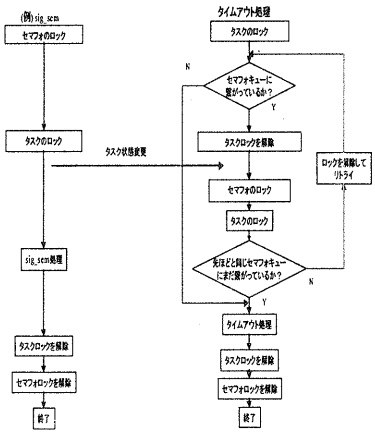

図 6: タイムアウト処理（リトライする方法）

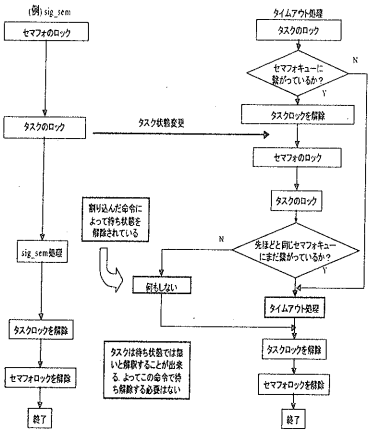

図 7: タイムアウト処理（リトライしない方法）

5.3.2 rel_wai（待ち状態の強制解除命令）の場合

この場合はタイムアウト処理の場合とほぼ同様に処理を進めることができる．

まず，リトライする方法を説明する．タイムアウト処理の場合と同様に `rel_wai` 命令側が一旦タスクロックを解除することにより，`sig_sem` 命令の割り込みを許し，自分はリトライすることによってデッドロックを回避している．図 8 にその様子を示す．

リトライせずにまず，図 9 で示すように，セマフォキューがまだ繋がっているかをチェックし，繋がっていた場合は，同様に処理を進め，繋がっていなかった場合は，そのタスクは他のタスクによってセマフォキューに繋がっている状態（待ち状態）から外れた（待ち解除された）訳なので待ち状態ではないと解釈できる．よってそれ以上処理を進める必要はなく，そのままロックを解除する．このように処理すればデッドロック回避のためのリトライをする必要が無くなる．

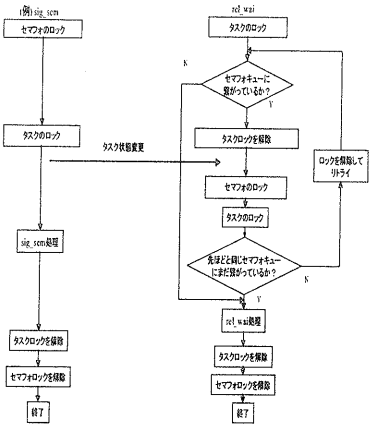

図 8: rel_wai（リトライする方法）

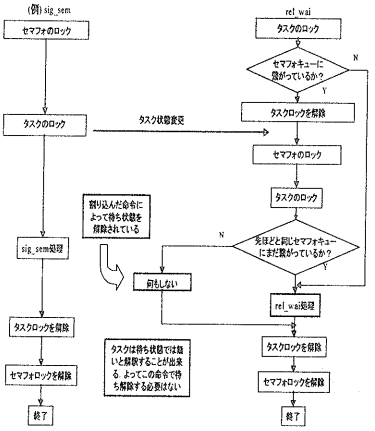

図 9: rel_wai（リトライしない方法）

5.3.3 chg_pri（タスクの優先度変更命令）の場合

タイムアウト処理と rel_wai はどちらの命令も，待ち状態を解除する（タイムアウト処理はタイムアウト待ちしているタスクの待ちを解除する，rel_wai はその命令の通り待ち状態を強制的に解除する）という点が共通しており，対象のタスクが自分ではない他の処理によって待ちが解除される場合を利用してリトライを回避することができたが，優先度の変更はそういうわけにはいかないので，リトライを回避するには他の手段が必要である．

そこで次のような処理を追加する．

- TCB（タスクコントロールブロック，タスク情報が示されている）に ’優先度を xx に変更したい’ というフラグを追加する．これを「優先度変更保留フラグ」と呼ぶことにする．
- 他の命令によって待ち状態を解除する処理を行う場合にも優先度変更の処理をするようにする．

この 2 つを追加することにより，図 10 のように処理をすることができる．

まず，はじめにタスクのロックを解除するときに，優先度変更保留フラグをセットしておき，その後セマフォキューにまだ繋がっているかをチェックしたときに繋がっているのならフラグを戻して優先度変更の処理を進めてロックを解除して終了する．もし繋がっていなかったならば，そのときは他の命令に割り込まれたということとなり，待ち状態を解除する命令の方で優先度が変更されているはずなので，こちら側では何も処理をしないでロックを解除する．一方，割り込んだ側の処理で優先度を変更する．

まず，はじめにタスクのロックを解除するときに，優先度変更保留フラグをセットしておき，その後，セマフォキューにまだ繋がっているかをチェックしたときに，もし繋がっているのならフラグを戻して優先度変更の処理を進めてロックを解除して終了する．繋がっていなかったならば，そのときは他の命令に割り込まれたということとなり，待ち状態を解除する命令の方で優先度が変更されているはずなので，こちら側では何も処理をしないでロックを解除する．一方，割り込んだ側の処理では待ち解除の処理をするときに優先度変更フラグがセットされているかを調べ，もしセットされているならば待ち状態を解除する際に優先度を変更する．

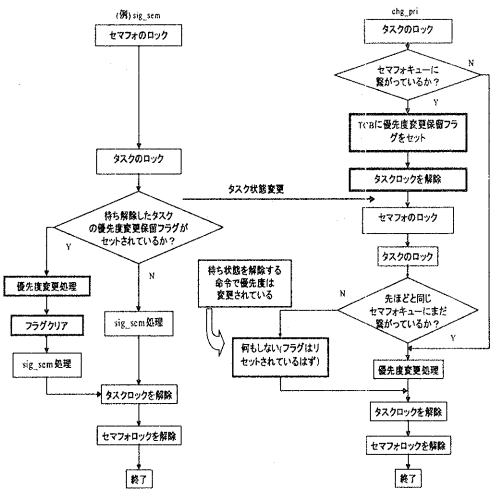

図 10: 優先度変更処理（リトライしない方法）

このように処理を変更することにより，優先度変更命令もリトライすることなくデッドロックを回避することができる．

6 おわりに

本研究では，ロックの逆順によるデッドロックを回避する対策を検討するために，ITRON-MP の基本構造を示し，従来のデッドロックの生じる過程を明らかにしてきた．そして次に，デッドロックが生じる可能性のある場合には，処理を中断してリトライすることによってデッドロックを回避する方法を示し，さらに，リトライをすることなくデッドロックを回避する方法を示した．本手法を用いれば，このようなロックの逆順によって生じるデッドロックはリトライすることなく回避することができる．リトライは，発生した場合のオーバーヘッドが大きいため，リトライをせずに

処理を進めることができるのはリアルタイムシステムを設計，利用する場合には，リアルタイム性の点で利点が大きいといえる．

今後の課題としての評価の方法は次のようなものが考えられる．

- デッドロックが生じないかのチェック

  従来のアルゴリズムではデッドロックが生じてしまうプログラムを作成し，今回提案したアルゴリズムではデッドロックが生じないことを確認する．

- 従来の方法でもデッドロックが生じないケースにおいてのスループットの比較

  従来の方法でもデッドロックが生じないプログラムを作成し，今回提案した方法と比べて，スループットがどのように変化しているかを調べる．

- 従来の方法においてデッドロックが生じるケースにおいて，リトライを必要とするアルゴリズムとリトライを必要としないアルゴリズムのスループットの比較

  リトライするアルゴリズムとリトライを必要としないアルゴリズムではどれくらいの性能差が出るかを比較する．

参考文献

[1] 高田　広章，坂村　健: “非対称マルチプロセッサシステムのためのスケーラブルなリアルタイムカーネルの構想”, 信学技報,vol.94, no.573, pp.1-8, 電子情報通信学会, Mar.1995

[2] 大畠　祐一: “マルチプロセッサ環境におけるマイグレート可能タスクの導入” , 武蔵工業大学修士論文,2000

[3] 坂村　建: “μITRON3.0 標準ハンドブック” , パーソナルメディア株式会社

[4] 坂村　健:“ITRON 標準ガイドブック 2”, パーソナルメディア株式会社