Vol.2010-OS-115 No.23
2010/8/4

周期実行制御における処理終了予定時刻超過の検出と対処

古川友樹[†1]　山内利宏[†1]　谷口秀夫[†1]

ロボットのモータ制御やセンサ制御における処理は周期的に実行される．このように周期的に実行される処理には制限があり，終了予定時刻までに処理を終了する必要がある．周期実行制御を実現する方法として，様々な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ている．しかし，何らかの原因により，終了予定時刻までに処理が終了しない場合への対処は全くなされていない．ロボットなどの組み込みシステムでは，より細かな制御を行うため，周期実行される処理の周期が短くなっている．また，実行する処理が高度化しており，それに伴い機能拡充の要請が多い．ここでは，処理終了予定時刻が近付くと，その旨を周期的に実行される処理に通知し，終了予定時刻までに行える適切な処理を実行できるシステムについて述べ，システムの基本的な処理機構を示す．

Control method to detect and deal with deadline miss for periodic scheduling

Yuuki Furukawa,[†1] Toshihiro Yamauchi[†1]
and Hideo Taniguchi[†1]

The processing to control the motor or the sensor in a robot is executed periodically. Such a processing has a limitation and must be finished before a deadline. Many real-time scheduling algorithms have been proposed to implement periodic scheduling. However, there is no method to deal with the case that the processing is not finished before a deadline. In embedded systems such as robots, the period of the processing executed periodically have shortened to realize a higher control. In addition, the processing is heightened, and there are many requests of the functionality expansion for it. In this paper, we propose the system to inform the processing that a deadline is near and execute a processing finished before the deadline. Also, we show a basic processing mechanism of our system.

1. はじめに

ロボットのモータ制御やセンサ制御における処理は周期的に実行される．このように周期的に実行される処理には制限があり，終了予定時刻までに処理を終了する必要がある．周期実行制御を実現する方法として，様々な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ている．しかし，何らかの原因により，終了予定時刻までに処理が終了しない場合への対処は全くなされていない．例えば，処理の終了予定時刻超過時の対処として，超過を無視した処理の継続，および強制終了がある．あるいは，終了予定時刻超過が全く発生しないように厳格な実行管理を行う．処理を継続した場合，当該処理の次周期実行は正しい周期で実行されない問題がある．また，強制終了した場合，強制終了される処理が関連する処理へ悪影響を与えてしまう．

ロボット[1]などの組み込みシステムでは，より細かな制御を行うため，周期実行される処理の周期が短くなっている．また，実行する処理が高度化[2]しており，それに伴い機能拡充の要請が多い．

ここでは，処理終了予定時刻が近付くと，その旨を周期的に実行される処理に通知し，終了予定時刻までに行える適切な処理を実行できるシステムについて述べる．

2. 従来手法の問題点

周期的に実行される処理（以降，周期処理）は，1 周期に 1 回実行される必要があり，起動した時刻から終了予定時刻までの間に 1 周期分の処理を終了しなければならない．周期処理のようなリアルタイム処理を実現するために，様々な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ている．

Rate Monotonic（RM）[3]は，固定優先度方式のスケジューリングアルゴリズムであり，周期の短い周期処理に高い優先度を割り当てる．つまり，最も短い周期を持つ周期処理が最高優先度の周期処理である．RM は優先度が固定であるため，制御オーバヘッドは小さく，周期処理の状態遷移を予測しやすい．しかし，周期の短い周期処理の実行が優先されるため，全ての周期処理を終了予定時刻（次の起動時刻）までに終了できるようにすると，プロセッサを最大限に利用できない．また，RM は周期処理の実行間隔の変動が小さく，処理終

†1 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

了予定時刻超過が生じる状態でも，周期処理の実行時間が周期より短いならば，最高優先度の周期処理は必ず終了予定時刻までに終了する．しかし，最高優先度以外の周期処理の実行間隔の変動は大きく，処理終了予定時刻超過は防止できない．また，高優先度の周期処理の実行時間が長くなると，低優先度の周期処理の処理終了予定時刻超過を発生しやすくなり，システム利用率が高い場合，低優先度の周期処理が全く実行されない場合も存在する[4]．

Deadline Monotonic（DM）は，起動時刻から終了予定時刻までの時間が短い周期処理に高い優先度を割り当てる固定優先度方式のスケジューリングアルゴリズムであり，周期処理の終了予定時刻が次の起動時刻である場合，DM は RM と同じである．また，固定優先度アルゴリズムであるため，処理終了予定時刻超過における問題は，RM と同じである．

Earliest Deadline First（EDF）は，動的優先度方式のスケジューリングアルゴリズムであり，終了予定時刻の近い周期処理に高い優先度を割り当てる．つまり，制御時において，終了予定時刻に最も近い周期処理が最高優先度の周期処理となる．シングルプロセッサ上で，プロセッサを 100%利用できるため，最適なスケジューリングアルゴリズムであるとされる．EDF は優先度を動的に変更するため，制御オーバヘッドは大きく，周期処理の状態遷移を予測しにくい．また，EDF は周期の短い周期処理の実行間隔を変動させるが，それ以外の周期処理の実行間隔の変動を小さくできる場合がある．しかし，処理終了予定時刻超過が生じる状態である場合，全ての周期処理が処理終了予定時刻超過となることがある[4]．

EDF 以外の動的優先度方式のスケジューリングアルゴリズムとして，Least Laxity First（LLF）がある．LLF は，終了予定時刻までの余裕時間が短い周期処理に高い優先度を割り当てる．余裕時間は，終了予定時刻までの残り時間から周期処理の残りの実行時間を引いた時間である．余裕時間を利用するスケジューリングアルゴリズムとして，RM Zero Laxity（RMZL）[5]，RM Critical Laxity（RMCL）[5]，Earliest Deadline until Zero Laxity（EDZL）[6]，LLREF[7] などがある．

上記のリアルタイムスケジューリングアルゴリズムでは，終了予定時刻超過が全く発生しないように厳格な実行管理が行われている場合において，正しい周期で周期処理を周期実行制御できる．しかし，終了予定時刻を超過する周期処理への対処はなされていない．周期実行制御における処理終了予定時刻超過の問題を**図 1** に示す．図 1 の周期処理 B が終了予定時刻までに終了できない周期処理である．周期実行制御において，終了予定時刻までに周期処理が終了できない場合，この周期処理の予定された処理が実行されない．また，終了予定時刻を超過する周期処理は，正しい周期で周期実行されないにも関わらず，他の周期処理の実行可能な時間を消費し，他の周期処理の実行間隔を変動させる．

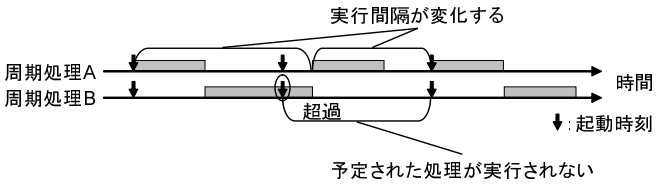

図 1　処理終了予定時刻超過の問題

3. 超過処理防止法

3.1 狙いと課題

処理終了予定時刻が近付くと，その旨を周期処理に通知し，終了予定時刻までに行える適切な処理を実行できる方法（以降，超過処理防止法と名付ける.）を確立し，システムを開発する．この狙いを実現するには，周期処理の実行状況を把握し，終了予定時刻が近付くことを検知し，処理終了予定時刻を超過するか否かを判定し，超過しそうな場合は終了予定時刻までの時間（以降，残り時間）に応じた適切な処理（以降，緊急処理）を呼び出す方法が必要である．このためには，課題として，以下の 5 つの方法を確立する必要がある．

（課題 1）　周期処理分割法
（課題 2）　モジュール間移行検出法
（課題 3）　モジュール実行時間の管理法
（課題 4）　緊急処理呼び出し契機決定法
（課題 5）　緊急処理呼び出し法

周期処理をモジュールに分割する方法を確立する必要がある．処理終了予定時刻の超過を判定し，緊急処理を呼び出すためには，周期処理の実行状況を把握できる必要がある．具体的には，処理終了予定時刻超過の判定処理（以降，判定処理）において，終了予定時刻までに周期処理が終了するか否か判定するためには，周期処理の残りの実行時間（以降，残り実行時間）を把握する必要がある．また，超過する場合において，残り時間に応じた緊急処理を実行するためには，実行中の処理と残りの処理について把握する必要がある．そこで，周期処理の実行状況を把握するため，複数のモジュールに周期処理を分割する．モジュールは，分割された周期処理の一部であり，1 周期において，1 回のみ実行される．周期処理を分割することで，各時刻において，周期処理の実行状況を把握することができる．これを実

現するため，1 回のみ実行されるモジュールに周期処理を分割できるような構造を持たせることが必要になる．

モジュール間の移行を検出する方法を確立する必要がある．周期処理をモジュールに分割し，周期処理の実行状況を把握するためには，1 つのモジュールが終了し，次のモジュールが開始されること（以降，モジュール間移行）を検出する必要がある．以降では，モジュール間移行時における周期処理の位置をモジュール境界とする．

モジュールの実行時間を管理する方法を確立する必要がある．処理終了予定時刻超過を判定するためには，判定時における正確な残り実行時間を把握する必要がある．つまり，各モジュールの開始から終了までに必要な時間を把握できることが必要である．

緊急処理呼び出しの契機を決定する方法を確立する必要がある．終了予定時刻が近付くことを検知するためには，終了予定時刻を把握する必要がある．周期実行制御において，OS は周期処理の処理終了予定時刻が近付くことを把握できる．また，処理終了予定時刻が近付いたとき，周期処理が終了予定時刻を超過するか否か，つまり緊急処理を呼び出すか否かを判定する方法が必要である．

緊急処理を呼び出す方法を確立する必要がある．処理終了予定時刻超過の判定時において，残り時間までに行える緊急処理を実行できるようにするため，緊急処理を呼び出す方法を確立する必要がある．

（課題 1）と（課題 2）に対処することで，周期処理の実行中の処理と残りの処理を把握することができる．これらに加えて，(課題 3）に対処することで，周期処理の残り実行時間を把握できる．また，(課題 4）に対処することで，処理終了予定時刻超過を判定できる．さらに，(課題 5）に対処することで，残り時間に応じた緊急処理を選択し，実行できる．

3.2 対　　処

3.2.1 周期処理分割法

周期処理の階層化プログラミングを規定し，main1，main2，……と呼び出すようなプログラム構造とする．階層化した周期処理の概観を**図 2** に示す．図 2 の main1()，main2()，……が各モジュールを表す．また，main1() と main2() の間がモジュール境界である．

3.2.2 モジュール間移行検出法

モジュール間移行を検出する方法として，以下の案がある．

（案 1） モジュール間移行毎に周期処理が OS に通知する．

（案 2） 周期処理の実行開始前に各モジュール境界を OS に通知する．

（案 3） OS がモジュール間移行を検出する．

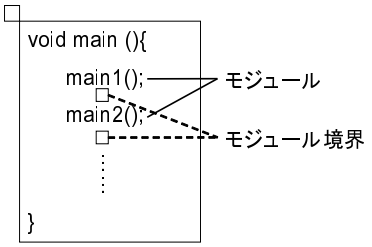

図 2　周期処理の分割

モジュール間移行毎に周期処理が OS に通知する場合，OS が管理しなければならないモジュールの情報が少なく，モジュール間移行を必ず検出できる．しかし，モジュールの数が多い場合，通知の制御オーバヘッドは大きい．

周期処理の実行開始前に各モジュール境界を OS に通知する場合，モジュール間移行を必ず検出できる．しかし，制御のために管理しなければならないモジュールの情報が多い．

OS がモジュール間移行を検出する場合，周期処理が実行状況を OS に通知する必要がない．しかし，制御における OS の負担が大きく，モジュール間移行を全て検出できるとは限らない．

ここでは，OS の管理する情報が少なく，モジュール間移行の検出精度が高いことから，モジュール間移行を検出するために，モジュール間移行毎に周期処理が OS に通知する．モジュール間移行の検出の様子を**図 3** に示す．図 3 の s_i が i 番目に実行されるモジュールを表し，モジュール間移行毎に周期処理は OS に通知を行う．これにより，処理終了予定時刻超過の判定時において，既に終了したモジュールは最後に通知されたモジュール境界より前に存在する全てのモジュールとなり，終了していないモジュールはモジュール境界より後に存在するモジュールとなる．また，周期処理の実行中のモジュールは，最後に通知されたモジュール境界の直後に存在するモジュールである．

3.2.3 モジュール実行時間の管理法

OS がモジュール間移行時の時刻を記録し，モジュール実行時間を計算する．モジュール実行時間を管理する方法として，以下の案がある．

（案 1） 1 周期前の周期処理のモジュール実行時間を保持する．

（案 2） 過去 N 回の周期処理のモジュール実行時間の平均値を保持する．

（案 3） 初回から N 回目までの周期処理のモジュール実行時間の平均値を保持する．

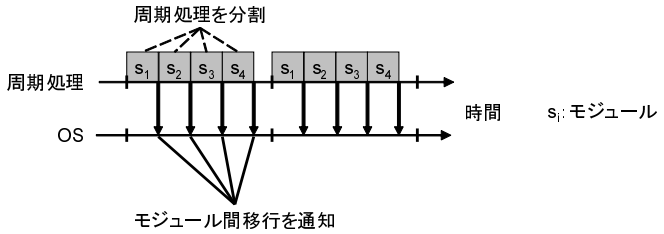

図 3 モジュール間移行の検出

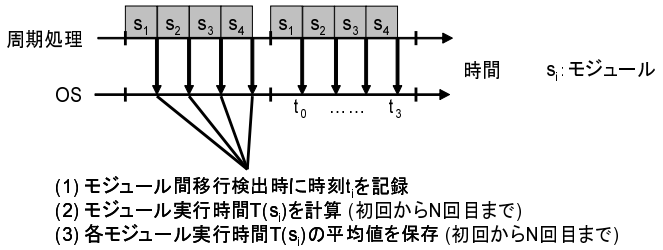

図 4 モジュール実行時間の記録

1 周期前の周期処理のモジュール実行時間を保持する場合，保存しなければならないモジュール実行時間は少なく，モジュール実行時間の計算にかかる制御オーバヘッドも小さい．しかし，モジュール実行時間の変動が大きいとき，制御への影響が大きい．

過去 N 回の周期処理のモジュール実行時間の平均値を保持する場合，何らかの要因で実行時間が変動する場合でも，正確な実行時間を把握できる．しかし，保存しなければならないモジュール実行時間は多く，モジュール実行時間の計算にかかる制御オーバヘッドは大きい．

初回から N 回目までの周期処理のモジュール実行時間の平均値を保持する場合，N 回目以降において，モジュール実行時間の計算にかかる制御オーバヘッドは小さい．しかし，保存しなければならないモジュール実行時間は多く，N 回目までの周期処理において，モジュール実行時間の計算にかかる制御オーバヘッドは大きい．

ここでは，モジュール実行時間の変動が制御に与える影響が比較的小さく，N 回目以降のモジュール実行時間の計算にかかる制御オーバヘッドが小さいことから，初回から N 回目までの周期処理のモジュール実行時間の平均値を保持することにする．モジュール実行時間の記録の様子を**図 4** に示し，以下に説明する．OS は，モジュール間移行の検出時に時刻 t_i を記録する．記録した時刻 t_i からモジュール実行時間 $T(s_i)$ を計算する．また，初回から N 回目までの周期処理のモジュール実行時間 $T(s_i)$ の平均値を計算し，保存する．N 回目以降の周期処理では，モジュール実行時間の平均値を計算しない．

3.2.4 緊急処理呼び出し契機決定法

周期処理の終了予定時刻を把握し，終了予定時刻が近付いたとき，残り時間と残り実行時間を比較することで，処理終了予定時刻超過を判定する．残り時間は，判定時の時刻と周期処理の終了予定時刻から計算できる．また，残り実行時間は終了していないモジュールの実行時間からわかる．処理終了予定時刻超過を判定する方法として，以下の案がある．

（案 1） タイマ割り込み発生時に残り時間と残り実行時間を比較する．

（案 2） モジュール間移行の検出時に残り時間と残り実行時間を比較する．

タイマ割り込み発生時に残り時間と残り実行時間を比較する場合，終了予定時刻の x 秒前に必ず判定処理を実行でき，制御オーバヘッドを小さくできる．しかし，残り実行時間と終了していないモジュールの実行時間の合計に差が生じるため，正確な残り実行時間を把握できない．このため，モジュール実行時間が長いと，処理終了予定時刻超過の検出精度は低くなる．

モジュール間移行の検出時に残り時間と残り実行時間を比較する場合，終了してないモジュール実行時間の合計と残り実行時間が等しくなり，正確な残り実行時間を把握できる．しかし，モジュール間移行の検出毎に判定処理を実行する必要があり，制御オーバヘッドは大きい．また，モジュール実行時間の変動が大きい，または他の周期処理が実行されているとき，処理終了予定時刻を超過する可能性がある．

ここでは，制御オーバヘッドが小さく，モジュール検出の精度が高いことから，タイマ割り込み発生時に残り時間と残り実行時間を比較し，処理終了予定時刻超過を判定する．判定処理の様子を**図 5** に示し，以下に説明する．OS は周期処理の終了予定時刻を把握し，タイマ割り込み発生時において，周期処理の終了予定時刻が近いとき，判定処理を実行する．判定処理において，残り時間 U_i と残り実行時間 V_i を比較し，$U_i \geq V_i$ ならば，周期処理が終了予定時刻までに終了する（正常終了）と判定する．また，$U_i < V_i$ ならば，処理終了予定時刻超過と判定する．

3.2.5 緊急処理呼び出し法

判定処理において，周期処理が終了予定時刻を超過すると判定した場合，OS から周期処

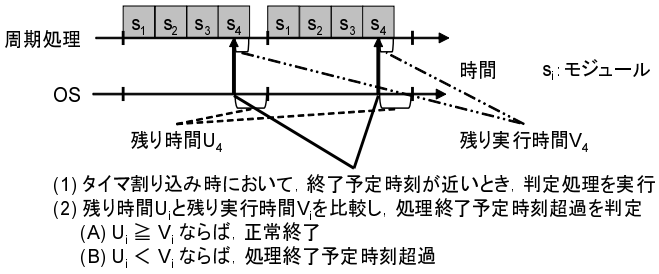

図 5 処理終了予定時刻超過の判定処理

理への upcall により，緊急処理を呼び出す．このとき，OS は実行中のモジュールと残り実行時間を通知する．これにより，周期処理の実行状況と残り時間に応じた緊急処理を実行することができる．

3.3 基本的な処理機構

超過処理防止法の基本的な処理機構を**図 6** に示し，超過制御防止法の 6 つの処理について，以下に説明する．

(1) 周期処理

周期処理は周期的に実行される．超過処理防止法では，周期処理を分割し，周期処理の実行状況を管理する．周期処理は，モジュール間移行時において，OS に通知を行う．周期処理の流れを**図 7** に示す．周期処理は，実行開始時に func1() を呼び出し，OS に周期処理の実行開始を通知する．また，モジュール間移行毎に func2() を呼び出し，OS にモジュール間移行を通知する．周期処理の全てのモジュールが終了したとき，周期処理は func3() を呼び出し，OS に周期処理の実行終了を通知する．

(2) モジュール間移行の検出処理

周期処理の通知毎に，OS は周期処理の実行状況を更新する．また，周期処理からの通知時の時刻を記録する．

(3) モジュール実行時間の記録処理

記録したモジュール間移行時の時刻 t_i から各モジュール s_i のモジュール実行時間 $T(s_i)$ を計算する．また，初回から N 回目までの周期処理におけるモジュール実行時間 $T(s_i)$ の平均値を計算し，保存する．このモジュール実行時間の平均値を処理終了予定時刻超過の判定に用いる．N 回目以降の周期処理では，モジュール実行時間の平均値を計算しない．

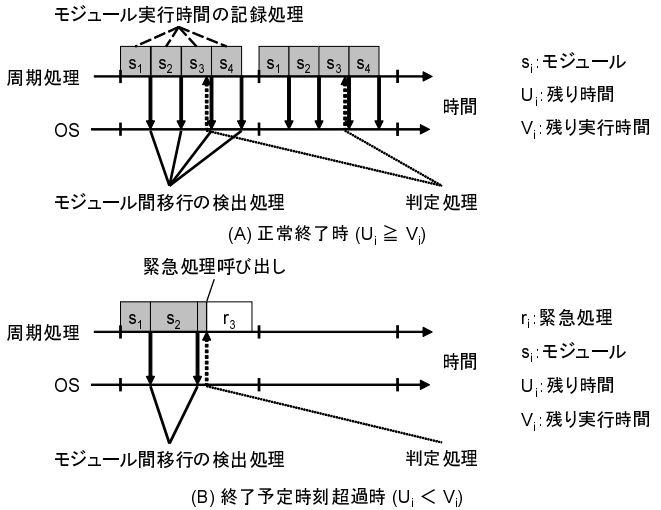

図 6 基本的な処理機構

(4) 判定処理

タイマ割り込み発生時において，周期処理の終了予定時刻 T_i が近いとき，処理終了予定時刻超過を判定する．残り時間 U_i と残り実行時間 V_i を比較し，$U_i \geq V_i$ ならば正常終了，$U_i < V_i$ ならば処理終了予定時刻超過と判定する．残り時間 U_i は判定時の時刻 t_i から終了予定時刻 T_i までの時間であり，残り実行時間 V_i は終了していないモジュール実行時間 $T(s_i)$ の合計 $\sum_{j=i}^{n} T(s_j)$ である．判定処理の流れを**図 8** に示す．タイマ割り込み発生時において，周期処理の終了予定時刻の近いとき，判定処理を実行する．判定処理において，周期処理の終了予定時刻 T を取得し，終了予定時刻 T から判定処理時の時刻 t_i から残り時間 U_i を算出する．また，各モジュールのモジュール実行時間 $T(s_i)$ を取得し，終了していないモジュールの実行時間を合計し，残り時間 V_i を求める．残り時間 U_i が残り実行時間 V_i 以上である場合，正常終了と判定し，判定処理を終了する．残り時間 U_i が残り実行時間 V_i より短い場合，処理終了予定時刻超過と判定し，緊急処理を呼び出す．

(5) 緊急処理の呼び出し

周期処理が終了予定時刻を超過すると判定した場合，OS から周期処理への upcall により，緊急処理を呼び出す．また，実行中のモジュール s_i と残り時間 U_i を通知する．

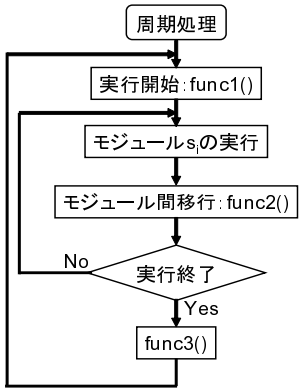

図 7 周期処理の流れ

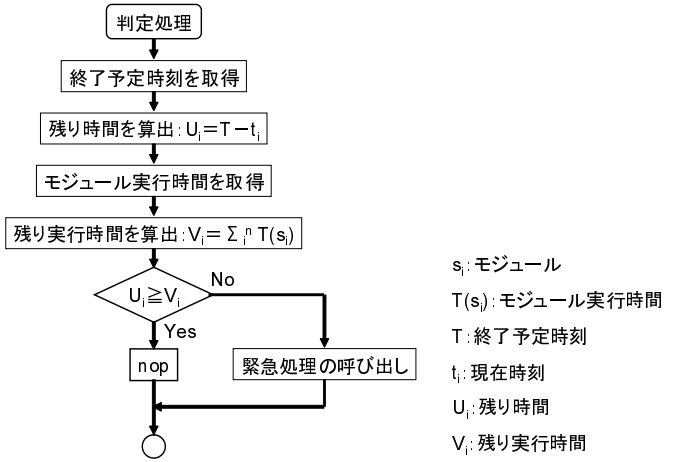

図 8 判定処理の流れ

(6) 緊急処理

緊急処理は，実行中のモジュール s_i と残り時間 U_i から適切な処理 r_i を選択し，実行する．

4. おわりに

周期処理を実現するリアルタイムスケジューリングアルゴリズムと処理終了予定時刻超過の問題について述べた．処理終了予定時刻超過への対処として，処理終了予定時刻が近付くと，その旨を周期処理に通知し，終了予定時刻までに行える適切な処理を実行できる方法を確立した．

ここでは，提案手法の狙いと課題について説明し，各課題への対処について述べた．また，基本的な処理機構を示した．提案手法の特徴は，周期処理をモジュールに分割すること，モジュール実行時間を記録すること，および周期処理が終了予定時刻を超過する場合，残り時間に応じた緊急処理を呼び出すことである．

残された課題として，実現方式の検討がある．

参考文献

1) Yokoi, K., Kanehiro, F., Kaneko, K., Kajita, S., Fujiwara, K. and Hirukawa, H.: Experimental Study of Humanoid Robot HRP-1S, Intl. J. Robotics Research, Vol.23, No.4-5, pp.351–362, 2004.
2) Buttazzo, G.: Research trends in real-time computing for embedded systems, ACM SIGBED Review, Vol.3, Issue 3 pp.1–10, 2006.
3) Liu, C., Layland, J.: Scheduling algorithms for multiprogramming in a hard real-time environment, Journal of the ACM, Vol.20, pp.46-61, 1973.
4) Buttazzo C. G.: Rate monotonic vs. EDF: judgment day, Real-Time Systems, The International Journal of Time-Critical Computing, Vol.29, Issue 1, pp.5-26, 2005.
5) 加藤真平, 山崎信行: Linux カーネル用リアルタイムスケジューリングモジュール, 情報処理学会論文誌: コンピューティングシステム, Vol.2, No.1, pp.75-86, 2009.
6) Cho, S., Lee, S., Han, A., and Lin, K.: Efficient Real-Time Scheduling Algorithms for Multiprocessor Systems, IEICE Transactions on Communications, Vol.E85-B, No.12, pp.2859-2867, 2002.
7) Cho, H., Ravindran, B., Jensen, E. D.: An Optimal Real-Time Scheduling Algorithm for Multiprocessors, Proceedings of the 27th IEEE International Real-Time Systems Symposium, pp.101-110, 2006.