情報処理学会論文誌　コンピューティングシステム　Vol. 4　No. 1　53–68 (Feb. 2011)

RT-Est：準固定優先度スケジューリング向けリアルタイムオペレーティングシステム

千代浩之[†1]　山崎信行[†1]

本論文では，準固定優先度スケジューリング向けリアルタイムオペレーティングシステム RT-Est を提案する．RT-Est は，次の機能を実装する．1）準固定優先度スケジューリングを低オーバヘッドで達成するために Linux カーネル 2.6 における $O(1)$ スケジューラを拡張したハイブリッド $O(1)$ スケジューラ，2）付加デッドライン時刻で付加部分を中断するための高精度タイマ，3）リアルタイムスケジューリングのシミュレーション向けアーキテクチャSIM である．SIM および実機による評価結果では，RT-Est における準固定優先度スケジューリングは自律移動ロボットに適していることを示す．

RT-Est: Real-time Operating System for Semi-fixed-priority Scheduling

Hiroyuki Chishiro[†1] and Nobuyuki Yamasaki[†1]

This paper proposes RT-Est, which is a real-time operating system for semi-fixed-priority scheduling. RT-Est implements the following mechanisms: 1) the hybrid $O(1)$ scheduler, which is an extension of the $O(1)$ scheduler in Linux kernel 2.6, to achieve semi-fixed-priority scheduling with low overhead; 2) the high resolution timer, which performs to terminate optional parts at optional deadlines; 3) SIM, which is an architecture for simulating real-time scheduling. Both evaluation results of SIM and the real machine show that semi-fixed-priority scheduling in RT-Est is well suited to autonomous mobile robots.

†1 慶應義塾大学
Keio University

1. 序　　論

自律移動ロボット[1),2)] のような低ジッタかつ高リアルタイム性を要求するリアルタイムシステムでは，多様な環境で動作することが要求される．リアルタイム性を保証するため，タスクの最悪実行時間を用いたリアルタイムスケジューリングアルゴリズムが多く提案されている[3)–5)]．しかしながら，このようなリアルタイムシステムを実現するためには，経路選択タスクのように，障害物の数に依存して実際実行時間が大きく変動するタスクにも対処しなければならない．現在実社会で利用されている主要な RTOS では，Linux をリアルタイム拡張した ART-Linux[6)]，ITRON 準拠の TOPPERS/ASP[7)] があげられる．ART-Linux と TOPPERS/ASP は RM スケジューリング[3)] を実装しているが，RM スケジューリングは実際実行時間が最悪実行時間より短い場合を考慮していないので，余った CPU 時間は無駄になってしまう．したがって，実際実行時間が最悪実行時間より短い場合，余った CPU 時間を有効利用可能なインプリサイス計算モデル[8)] をサポートとした RTOS が望ましい．

インプリサイス計算モデルは，必須部分（mandatory part）と付加部分（optional part）を持つ．必須部分は許容可能な最低限の品質を持つ結果を生成し，付加部分は必須部分が生成した結果の品質を向上させる．インプリサイス計算モデルを用いる場合，必須部分のリアルタイム性を保証すれば，過負荷状態に陥ったとしても，付加部分を中断することでデッドラインミスを回避することが可能である．従来のインプリサイス計算モデルは付加部分を中断または完了する際に補完的な処理が必要ないことを仮定している．しかしながら，自律移動ロボットの制御には付加部分を中断または完了する際に補完的な処理が必要である．たとえば，画像処理タスクでは画像データを入力（必須部分），演算の精度を向上（付加部分），解析結果を出力（必須部分）のように終端にも第 2 の必須部分が要求される．この**終端部分（wind-up part）**を有するモデルを**拡張インプリサイス計算モデル**[9),10)] と呼ぶ．

拡張インプリサイス計算モデル向け RTOS として RT-Frontier[11)] がある．RT-Frontier は Mandatory-Fist with Wind-up Part（M-FWP）[9),10)] のような拡張インプリサイスタスクを用いた Earliest Deadline First（EDF）[3)] に基づく動的優先度スケジューリングを実装している．しかしながら，動的優先度スケジューリングにおいて，レディキューにタスクを挿入，削除する操作の計算量は，タスク数を n とした場合 $O(log\ n)$ になってしまう．タスクを挿入，削除する計算量が $O(n)$ になるという問題を**表 1** に我々が開発している自律移動ロボット[12),13)] のタスクを用いて説明する．各々のタスクの詳細は省略するが，この自律移動ロボットを実現するためには，これらのタスクのリアルタイム性を保証しなければなら

表 1　自律移動ロボットのタスク例
Table 1　Tasks in an autonomous mobile robot.

タスク	周期 [ms]
モータ制御	1.0
サーボ制御	1.0
状態チェック	1.0
センサー処理	2.0
タスク管理	10.0
データ共有	10.0
自己位置推定	10.0
運動計画	100.0
音声処理	100.0
画像処理	200.0

ない．さらに，より高精度な制御を行うためには，様々なセンサからのデータを処理しなければならず，タスク数はより増加すると考えられる．これに対して，Linux カーネル 2.6 から実装された $O(1)$ スケジューラ[14] では，レディキューにタスクを挿入，削除する操作の計算量は，RM[3] のような固定優先度スケジューリングの場合では $O(1)$ だが，EDF のような動的優先度スケジューリングには適応できない．また，固定優先度スケジューリングで拡張インプリサイスタスクをスケジュールする場合，付加部分のオーバランが原因で終端部分がデッドラインミスを発生してしまう．それゆえ，固定優先度スケジューリングを拡張インプリサイスタスクに適応させた準固定優先度スケジューリング[15] の実装が要求される．

本論文では，準固定優先度スケジューリング向け RTOS である **RT-Est** を提案する．RT-Est は準固定優先度スケジューリングを低オーバヘッドで実現するために，Linux カーネル 2.6 における $O(1)$ スケジューラを拡張したハイブリッド $O(1)$ スケジューラ，付加デッドライン時刻で付加部分を中断するための高精度タイマ，リアルタイムスケジューリングのシミュレーション向けアーキテクチャである SIM を実装する．SIM および実機による評価結果では，RT-Est における準固定優先度スケジューリングは自律移動ロボットに適していることを示す．

本論文の貢献は，準固定優先度スケジューリングを実現する RTOS である RT-Est をスクラッチから実装したことにある．RT-Est は，準固定優先度スケジューリングにより，拡張インプリサイスタスクを低ジッタかつ低オーバヘッドでスケジュール可能なので，自律移動ロボットで高精度な動作を実現する．

本論文の構成を以下に示す．2 章では，本論文で仮定するシステムモデルと用語の定義を述べる．3 章では，準固定優先度スケジューリングについて述べる．4 章では，ハーモニックタスクセットにおける準固定優先度スケジューリングについて述べる．5 章では，RT-Est について述べる．6 章では，RT-Est および準固定優先度スケジューリングの性能評価について述べる．最後に 7 章で結論を述べる．

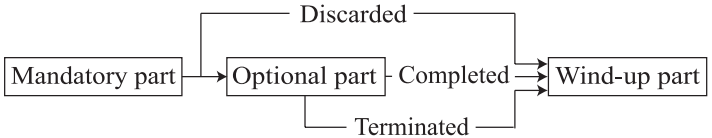


図 1　拡張インプリサイス計算モデル
Fig. 1　Extended imprecise computation model.

2. システムモデル

図 1 に示すように，拡張インプリサイス計算モデル[9),10)] は，従来のインプリサイス計算モデル[8)] に終端部分を追加したモデルである．従来のインプリサイス計算モデルでは，付加部分を中断または完了する際に生成した結果を出力する処理が不必要であるが，自律移動ロボットによるモータ制御タスクを実行する場合，生成した結果をアクチュエータに出力する処理が必要である．この出力処理のリアルタイム性を保証しなければならないが，従来のインプリサイス計算モデルに基づくタスクでは，付加部分の中断または完了後にいかなる処理も実行できない．我々の方式である拡張インプリサイス計算モデル[9),10)] は，従来の Lin らの方式によるインプリサイス計算モデル[8)] の付加部分の中断または完了処理のリアルタイム性を保証可能にするために，従来のインプリサイス計算モデルを改良した．必須部分，付加部分，終端部分からなるリアルタイムタスクを，2 つのリアルタイムタスクと 1 つの非リアルタイムタスクに分割する手法は我々が提案した拡張インプリサイス計算モデル[9),10)] が最初である．また，タスクを分割する手法は，マルチプロセッサにおけるリアルタイムスケジューリングのスケジュール可能性を向上させるために一般的に用いられているが，シングルプロセッサにおけるタスクを分割する手法は，我々の知る限り Periodic Transformation Technique（PTT）[16] のみである．しかしながら，PTT では最悪実行時間を等分割する場合，すなわち必須部分と終端部分の最悪実行時間が等しい場合しか適応できない．これに対して，拡張インプリサイス計算モデルでは，必須部分と終端部分の最悪実行時間が異なる場合にも適応できる．

システムには n 個の拡張インプリサイスタスクから構成されるタスクセット Γ ＝

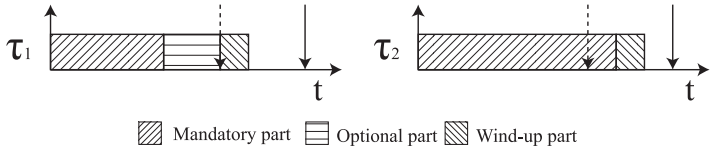


図 2　付加デッドライン
Fig. 2　Optional deadline.

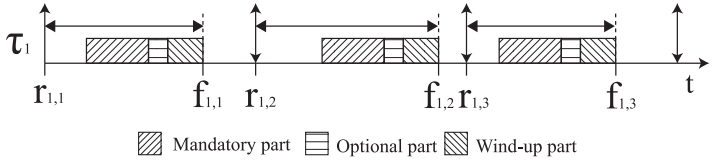


図 3　相対完了ジッタ
Fig. 3　Relative finishing jitter.

$\{\tau_i(T_i, D_i, OD_i, m_i, o_i, w_i), i = 1, 2, ..., n\}$ が与えられる．ここで，T_i は周期，D_i は相対デッドライン，OD_i は相対付加デッドライン，m_i は必須部分の最悪実行時間，o_i は付加部分の要求実行時間，w_i は終端部分の最悪実行時間を表す．表 1 で示したとおり，タスクセット内の全タスクの周期はハーモニック，すなわち互いに整数倍となるので，タスクセットをハーモニックタスクセットに制限することは妥当である．また，タスク τ_i の j 番目のインスタンスのことをジョブ $\tau_{i,j}$ と呼ぶ．付加デッドラインの詳細は後述する．付加部分の要求実行時間はジョブごとに変動し，その上限は未知とする．ただし，RM および EDF のような一般計算モデル用のアルゴリズムで拡張インプリサイスタスクをスケジュールする場合，付加部分のオーバランが原因で終端部分のデッドラインミスが発生することを回避するために，OD_i と o_i をそれぞれ 0 に設定し，必須部分と終端部分を連続して実行する．また，タスクの周期 T_i は相対デッドライン D_i と等しい．τ_i の CPU 利用率 U_i を $(m_i + w_i)/T_i$ とする．このように，U_i には必須部分および終端部分の最悪実行時間のみ含み，付加部分の要求実行時間を含まない．この理由は，付加部分の要求実行時間を実行したか否かがハーモニックタスクセットにおけるスケジュールの成否とは無関係だからである．システム全体の CPU 利用率 $U = \sum_{i=1}^{n} U_i$ とする．タスクは，周期の短い順にソートされているものとする．つまり，$T_1 \leq T_2 \leq ... \leq T_n$ が成立する．スケジュールの開始前にタスクを周期の昇順でソートするため，スケジュール実行時のコストには含まれない．

準固定優先度スケジューリング固有のパラメータである**付加デッドライン**は，付加部分が実行可能な期限および終端部分の実行開始時刻を表す．付加デッドライン以降では付加部分を実行できない．付加デッドラインまでに必須部分が完了していれば，終端部分がデッドラインミスを発生しない時刻に付加デッドラインを設定する．**図 2** に τ_1 と τ_2 の付加デッドラインを例示する．実線の上矢印はリリース，実線の下矢印はデッドライン，破線の下矢印は付加デッドラインを表す．τ_1 は付加デッドライン以降では付加部分を実行せず，終端部分を実行する．これに対して，τ_2 は付加デッドライン以降でも必須部分を実行する．τ_2 のように付加デッドライン以降で必須部分を実行する場合，付加デッドライン時刻ではなく必須部分の完了時に終端部分の実行を開始する．

タスクの**相対完了ジッタ（RFJ: Relative Finishing Jitter）**は 2 つの連続したジョブの完了時刻の最大偏差である．本論文では，RFJ のことをジッタと呼ぶ．ジョブ $\tau_{i,j}$ のリリース時刻 $r_{i,j}$ と完了時刻 $f_{i,j}$ を用いて，RFJ を下式に示す．

$$RFJ_i = \max_j |(f_{i,j+1} - r_{i,j+1}) - (f_{i,j} - r_{i,j})|$$

図 3 に RFJ を示す．この場合，τ_1 の RFJ は $|(f_{1,2} - r_{1,2}) - (f_{1,1} - r_{1,1})|$ と $|(f_{1,3} - r_{1,3}) - (f_{1,2} - r_{1,2})|$ の最大値になる．ジッタを低くすることは RFJ を低くすることを意味する．また，最短周期タスクのみの RFJ を Shortest Period Jitter（SPJ）と定義する．

3. 準固定優先度スケジューリング

準固定優先度スケジューリング[15] は，拡張インプリサイスタスクの各々の部分の優先度は固定だが，必須部分から付加部分または付加部分から終端部分に実行部分を遷移する際に，優先度を変更する．したがって，本スケジューリングを準固定優先度スケジューリングと呼ぶ．準固定優先度スケジューリングは，**図 4** に示すように，1 つの拡張インプリサイスタスク τ_i のリアルタイム性を要求する部分である必須部分および終端部分を 2 つのサブタスク τ_i^m と τ_i^w に分割し，各々のサブタスクを固定優先度でスケジュールする．また，付加部分のオーバランにより必須部分と終端部分のリアルタイム性の低下を防ぐために，必須部分または終端部分が実行可能状態のタスクがない場合のみ，付加部分が実行可能状態のタスクを固定優先度でスケジュールする．したがって，付加部分の優先度は必須部分と終端部分の優先度より必ず低くなる．

図 4 に示すように 1 つの拡張インプリサイスタスク τ_i を 2 つのタスク τ_i^m と τ_i^w に分割する．タスク τ_i^m と τ_i^w の周期はともに T_i，最悪実行時間はそれぞれ m_i と w_i，最初のジョ

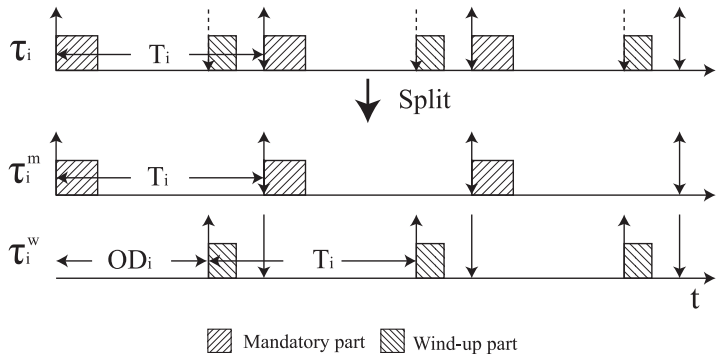


図 4　1 つの拡張インプリサイスタスクを 2 つの一般タスクに分割
Fig. 4　Split one extended imprecise task into two general tasks.

ブの開始時刻はそれぞれ 0 と OD_i，相対デッドラインはそれぞれ D_i と $D_i - OD_i$ である．また，タスク τ_i^m と τ_i^w は同時に実行不可であり，各々固定優先度でスケジュールされる．

図 5 に一般スケジューリングと準固定優先度スケジューリングを示す．一般スケジューリングとは，RM や EDF のような一般タスク用のスケジューリングである．時刻 t における残り実行時間 $R_i(t)$ を，一般スケジューリングは破線，準固定優先度スケジューリングは実線で表す．付加部分の実行は準固定優先度スケジューリングのみなので省略する．一般スケジューリングでは，τ_i のリリース時刻で $R_i(t)$ に $m_i + w_i$ を設定する．$R_i(t)$ は 0 になるまで減少する．これに対して，準固定優先度スケジューリングでは，τ_i のリリース時刻で $R_i(t)$ に m_i を設定する．そして，付加デッドライン時刻 OD_i で $R_i(t)$ に w_i を設定する．必須部分が付加デッドライン時刻以降で実行する場合，付加デッドライン時刻ではなく必須部分の実行が完了した時刻で $R_i(t)$ に w_i を設定する．

準固定優先度スケジューリングアルゴリズム RMWP [15] は，Real-Time Queue（RTQ）と Non-Real-Time Queue（NRTQ）および Sleep Queue（SQ）の 3 つのキューを管理する．**図 6** で示すように，RTQ は必須部分または終端部分，NRTQ は付加部分が実行可能状態のタスクを管理する．RTQ および NRTQ 内にあるタスクはそれぞれ RM でスケジュールされる．ただし RTQ にあるタスクは NRTQ にあるタスクより優先度が高く，RTQ にタスクがない場合のみ NRTQ にあるタスクを実行する．あるタスク τ_i の必須部分または終端部分が両方 RTQ 内に存在する場合はない．**図 7** に RMWP アルゴリズムを示す．RMWP アルゴリズムのイベントは 7 種類あり，各々の条件が成立するときに該当する処理を実行する．ここで，RMWP の最悪干渉時間および付加デッドラインを文献 15) の定理 1 と定理 2 を以下に示す．

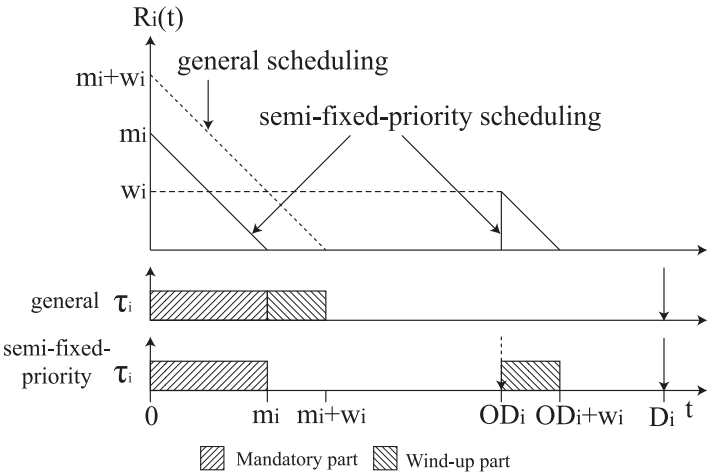


図 5　一般スケジューリングと準固定優先度スケジューリング
Fig. 5　General scheduling and semi-fixed-priority scheduling.

定理 1（RMWP の最悪干渉時間）．タスク τ_k が τ_k より高優先度タスク τ_i（$i < k$）により干渉される最悪干渉時間 I_k^i は大きくとも下式である．

$$I_k^i = \left\lceil \frac{T_k}{T_i} \right\rceil (m_i + w_i)$$

定理 2（RMWP の付加デッドライン）．*RMWP* におけるタスク τ_k の付加デッドライン OD_k は下式の場合，τ_k の必須部分が付加デッドライン OD_k までに完了していれば τ_k の終端部分はデッドラインミスを発生しない．

$$OD_k = D_k - w_k - \sum_{i=1}^{k-1} I_k^i$$

図 8 に RMWP と RM のスケジュール例を示す．ハーモニックタスクセットは $\Gamma = \{\tau_1 = (10, 10, 7, 3, 4, 3),\ \tau_2 = (20, 20, 6, 3, 4, 2)\}$ である．タスク τ_1，τ_2 の付加デッドラインは，定理 2 を用いて算出した．RMWP は，付加デッドライン時刻で終端部分を開始するので，区間 [14,17) でジョブ $\tau_{1,2}$ の付加部分を実行可能であることが RM より優位である．

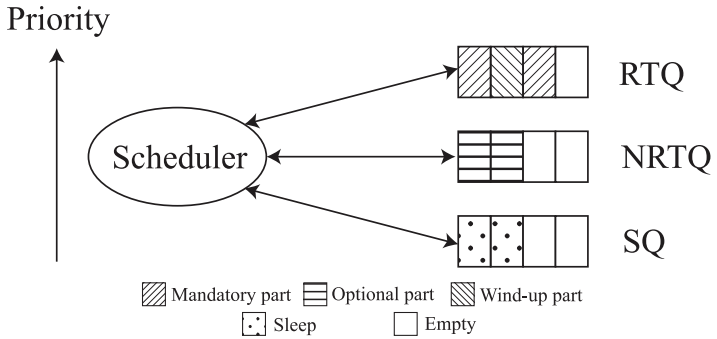


図 6　タスクキュー
Fig. 6　Task queue.

(1)	When τ_i becomes ready, set $R_i(t)$ to m_i, dequeue τ_i from SQ, and enqueue τ_i to RTQ. If τ_i has the highest priority in RTQ, preempt the current task.
(2)	When τ_i completes its mandatory part:
(a)	If OD_i expired, set $R_i(t)$ to w_i.
(b)	Otherwise set $R_i(t)$ to o_i, dequeue τ_i from RTQ and enqueue τ_i to NRTQ. If there are one or multiple tasks in RTQ or NRTQ which have higher priority than τ_i, preempt τ_i.
(3)	When τ_i completes its optional part, dequeue τ_i from NRTQ and enqueue τ_i to SQ.
(4)	When OD_i expires:
(a)	If τ_i is in RTQ and does not complete its mandatory part, do nothing.
(b)	If τ_i is in NRTQ, terminate and dequeue τ_i from NRTQ, set $R_i(t)$ to w_i, and enqueue τ_i to RTQ. If τ_i has the highest priority in RTQ, preempt the current task.
(c)	If τ_i is in SQ, dequeue τ_i from SQ, set $R_i(t)$ to w_i and enqueue τ_i to RTQ.
(5)	When τ_i completes its wind-up part, dequeue τ_i from RTQ and enqueue τ_i to SQ.
(6)	When there are one or multiple tasks in RTQ, perform RM in RTQ.
(7)	When there is no task in RTQ and there are one or multiples tasks in NRTQ, perform RM in NRTQ.

図 7　RMWP アルゴリズム
Fig. 7　RMWP algorithm.

M-FWP では付加部分を実行した場合，ジッタが大幅に高くなってしまうが，RMWP では付加部分の実行に影響されずに安定してジッタを低く維持することが可能である．しかしながら，RMWP は M-FWP より付加部分の実行割合が低くなる．それゆえ，RMWP で付加部分の実行割合を向上させる手法が要求される．

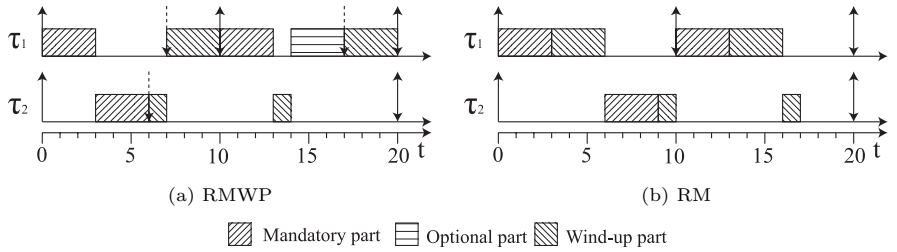


図 8　RMWP と RM のスケジュール例
Fig. 8　Example of schedule generated by RMWP and RM.

4. ハーモニックタスクセットにおける準固定優先度スケジューリング

本章では，ハーモニックタスクセットにおける，RMWP でスケジュールする拡張インプリサイスタスクの付加部分の実行割合を向上させるために，Response Time Analysis（RTA）[5] を拡張した Response Time Analysis for Optimal Optional Deadline with Harmonic Task Sets（RTA-OODH）を提案する．

RTA-OODH では最適な付加デッドラインを設定する．最適な付加デッドラインとは，タスク τ_k の区間 $[OD_k, D_k)$ における τ_k に割当て可能時間が τ_k の終端部分の最悪実行時間 w_k と等しくなる時刻である．すなわち，タスクの実際実行時間が最悪実行時間と等しい場合，付加部分に利用可能な時間があるにもかかわらず，付加部分を中断または破棄しない時刻に，付加デッドラインを設定する．また，RTA-OODH で算出した最適な付加デッドラインは，定理 2 で算出した付加デッドライン以上であることを示す．RTA-OODH を用いることで，RMWP は付加部分の実行割合を向上させることが可能になる．最適な付加デッドラインを算出するために，タスク τ_k に割当て可能な w_k を除いた時間 A_k を求める．

定理 3（ハーモニックタスクセットにおける割当て可能時間）．ハーモニックタスクセットにおけるタスク τ_k に割当て可能な w_k を除いた時間 A_k は，定理 *1* より算出した，タスク τ_k が τ_k より高優先度タスク τ_i に干渉される最悪干渉時間 I_k^i を用いて，下式に示す．

$$A_k = D_k - w_k - \sum_{i=1}^{k-1} I_k^i \tag{1}$$

証明．ハーモニックタスクセットにおける，タスク τ_k と τ_k より高優先度の全タスクの周期の最小公倍数は T_k，すなわち D_k と等しい．また，ハーモニックタスクセットでは，τ_k の各々のジョブの最悪干渉時間は一定である．したがって，タスク τ_k に割当て可能な w_k を除いた時間 A_k は式 (1) となる．　□

次に，ハーモニックタスクセットにおける，区間 $[0, OD_k)$ のタスク τ_k の最悪干渉時間 I_k を求める．

定理 4（ハーモニックタスクセットにおける最悪干渉時間）．ハーモニックタスクセットにおける，区間 $[0, OD_k)$ のタスク τ_k の最悪干渉時間 I_k は下式である．

$$I_k = \sum_{i=1}^{k-1} \left(\left\lceil \frac{OD_k}{T_i} \right\rceil m_i + \left\lceil \frac{OD_k - OD_i}{T_i} \right\rceil w_i \right)$$

証明．図 4 のように 1 つの拡張インプリサイスタスク τ_i を 2 つの一般タスク τ_i^m と τ_i^w に分割して考える．一般タスク τ_i^m と τ_i^w の最初のジョブの開始時刻はそれぞれ 0 と OD_i で，周期は両方とも T_i である．したがって，区間 $[0, OD_k)$ でタスク τ_k は τ_k より高優先度タスク τ_i の必須部分に $\lceil OD_k/T_i \rceil$ 回，終端部分に $\lceil (OD_k - OD_i)/T_i \rceil$ 回干渉される．　□

そして，RTA [5] を応用して，定理 3 と定理 4 より最適な付加デッドラインを RTA-OODH より算出する．

定理 5（RTA-OODH）．ハーモニックタスクセットにおけるタスク τ_k の最適な付加デッドライン OD_k は下式である．

$$OD_k = A_k + I_k \tag{2}$$

証明．タスク τ_k の付加デッドライン OD_k と，τ_k に割当て可能な w_k を除いた時間 A_k は，それぞれ RTA [5] におけるタスク τ_k の最悪応答時間と最悪実行時間に該当する．すなわち，タスクの実際実行時間が最悪実行時間と等しい場合，式 (2) より算出する付加デッドラインは，付加部分に割当て可能時間があるにもかかわらず，付加部分を中断または破棄されない．また，ハーモニックタスクセットにおける，各々のジョブに割当て可能時間は一定なので，区間 $[OD_k, D_k)$ における τ_k に割当て可能時間が w_k と必ず等しくなる．それゆえ，式 (2) より算出する付加デッドライン OD_k は最適である．　□

ここで，定理 2 より算出する一般タスクセットにおけるタスク τ_k の付加デッドライン OD_k は，定理 3 より算出する τ_k に割当て可能な w_k を除いた時間 A_k と等しい．定理 5 より，τ_k の最適な付加デッドライン OD_k は必ず A_k 以上になるので，RTA-OODH で算出する最適な付加デッドラインは定理 2 より算出する付加デッドライン以上になることは明らかである．**図 9** に，RTA-OODH 関数を示す．RTA-OODH 関数では RTA のスケジュール可能性判定と同様の反復を行い，最適な付加デッドラインを算出する．定理 5 を用いて，ハーモニックタスクセットにおける RMWP のスケジュール可能性上限を以下に示す．

```
RTA-OODH(Γ) {
  while (τ_k ∈ Γ) {
    A_k = D_k − w_k − Σ_{i=1}^{k−1} I_k^i;
    I = 0;
    do {
      OD = I + A_k;
      I = Σ_{i=1}^{k−1} ( ⌈OD/T_i⌉ m_i + ⌈(OD−OD_i)/T_i⌉ w_i );
    } while (I + A_k > OD);
    OD_k = OD;
  }
}
```

図 9　RTA-OODH 関数
Fig. 9　RTA-OODH function.

定理 6（ハーモニックタスクセットにおける RMWP のスケジュール可能上限）．ハーモニックタスクセットにおける RMWP のスケジュール可能上限 U_{lub} は 1 である．

証明．定理 5 より算出する最適な付加デッドラインまでにタスクの必須部分が完了していれば，終端部分がデッドラインミスを発生しないことは明らかである．また，文献 15) の定理 3 より RMWP は RM より優位である．それゆえ，ハーモニックタスクセットにおける RMWP のスケジュール可能上限 U_{lub} は RM のスケジュール可能上限と等しく 1 である [3]．　□

図 10 にハーモニックタスクセットにおける，RMWP のスケジュール例を示す．ハーモニックタスクセットは $\Gamma = \{\tau_1 = (5, 5, 4, 1, 0, 1),\ \tau_2 = (10, 10, 8, 2, 0, 1),\ \tau_3 = (20, 20, 14, 2, 2, 2)\}$ である．最適な付加デッドラインは，定理 5 より算出した．RTA-OODH によるタスク τ_3 の最適な付加デッドラインの算出例を述べる．まず，タスク τ_3 に割当て可能な w_k を除いた時間 A_3 は定理 3 より $A_3 = D_3 - w_3 - \sum_{i=1}^{2} I_3^1 = 20 - 2 - [4 * (1 + 1) + 2 * (2 + 1)] = 4$ と算出する．次にタスク τ_3 の最適な付加デッ

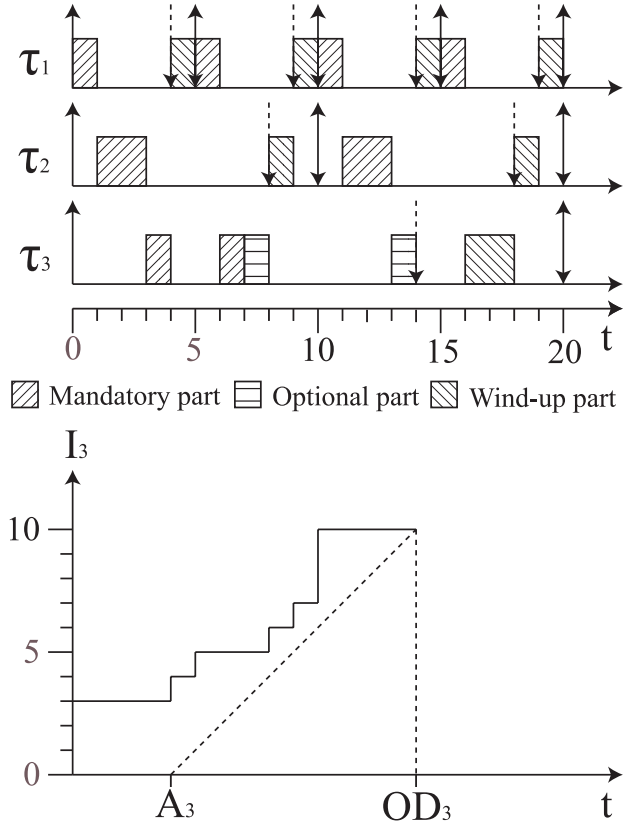


図 10　ハーモニックタスクセットにおける RMWP のスケジュール例
Fig. 10　Example of schedule generated by RMWP with harmonic task sets.

ドライン OD_3 を定理 5 より算出する．図 10 の下部に τ_3 の干渉時間 I_3 の推移を示す．定理 2 より算出したタスク τ_3 の付加デッドラインは 4，必須部分の完了時刻は 7 なので，τ_3 の付加部分を実行できない．これに対して，定理 5 より算出した τ_3 の最適な付加デッドラインは 14 なので，タスク τ_3 のデッドラインミスを発生させることなく τ_3 の付加部分を実行する．

```
part = save_context();
switch (part) {
case MANDATORY:
  exec_mandatory();
  res = end_mandatory();
case OPTIONAL:
  if (res != DISCARD) {
    exec_optional();
    end_optional();
  }
case WINDUP:
  exec_windup();
}
end_task();
```

図 11　拡張インプリサイス計算モデルの疑似コード
Fig. 11　Pseudo code of extended imprecise computation model.

5. RT-Est

本章では，提案する準固定優先度スケジューリング向け RTOS である RT-Est について述べる．RT-Est は準固定優先度スケジューリングを実現し，拡張インプリサイスタスクを低ジッタかつ低オーバヘッドでスケジュールする．まず，拡張インプリサイス計算モデルの実装について述べる．次に，ハイブリッド $O(1)$ スケジューラの実装について述べる．そして，ハイブリッド $O(1)$ スケジューラを用いて準固定優先度スケジューリングを実現するために必要な機能である高精度タイマの実装について述べる．最後に，リアルタイムスケジューリング向けアーキテクチャである SIM について述べる．

5.1 拡張インプリサイス計算モデル

本節では，RT-Est における拡張インプリサイス計算モデルの実装について述べる．**図 11** に拡張インプリサイス計算モデルの疑似コードを示す．まず，`save_context` 関数でタスクのレジスタやプログラムカウンタ等を保存する．`save_context` 関数が，付加部分の中断時のタイマ割込み以外で呼び出された場合，その返戻値は `MANDATORY` となる．ただし，付加部分の中断時のタイマ割込みで呼び出された場合，その返戻値は `WINDUP` となる．`switch` 文には `break` 文が存在せず，必須部分，付加部分，終端部分の順でタスクを実行する．まず，`exec_mandatory` 関数で必須部分を実行する．必須部分を完了した場合，`end_mandatory` 関

数を呼び出す．end_mandatory 関数では，付加部分に割当て可能時間の有無を判定する．付加部分に割当て可能時間がない場合，つまり end_mandatory 関数の返戻値が DISCARD の場合，付加部分を破棄し，exec_windup 関数で終端部分を実行する．付加部分に割当て可能時間がある場合，つまり end_mandatory 関数の返戻値が DISCARD 以外の場合，exec_optional 関数で付加部分を実行する．付加部分を完了した場合，end_optional 関数を呼び出す．end_optional 関数では，付加部分の完了処理を行う．そして，exec_windup 関数で終端部分を実行する．付加部分を中断した場合，中断時刻でタイマ割込みが発生する．スケジューラは save_context 関数で保存したタスクのレジスタやプログラムカウンタ等を復帰させる．付加部分の中断処理から復帰するアドレスは save_context 関数の呼び出し後のプログラムカウンタである．タイマ割込みが発生した場合，save_context 関数の返戻値は WINDUP となるので，exec_windup 関数で終端部分を実行する．終端部分の完了後，end_task 関数を呼び出し，タスクを次のリリース時刻までスリープさせる．

5.2　ハイブリッド $O(1)$ スケジューラ

ハイブリッド $O(1)$ スケジューラは，Linux カーネル 2.6 における $O(1)$ スケジューラ[14]を準固定優先度スケジューリング向けに拡張することにより実現する．ここで，RT-Est はスクラッチから開発された RTOS なので，我々はハイブリッド $O(1)$ スケジューラをスクラッチから開発した．タスクを周期の昇順でソートすることは，スケジュールを開始する前に行われる．したがって，スケジュール実行時には，このコストを無視できるので，$O(1)$ で最高優先度タスクを検索できる．**図 12** にハイブリッド $O(1)$ スケジューラを示す．拡張インプリサイスタスクの必須部分および終端部分の優先度と付加部分の優先度はそれぞれ 256 段階とする．また，Linux カーネル 2.6 と同様に値が小さいほど優先度は高いとする．それゆえ，RTQ に格納するタスクの必須部分および終端部分の優先度の範囲を 0〜255 とし，NRTQ に格納するタスクの付加部分の優先度の範囲は 256〜511 とする．ハイブリッド $O(1)$ スケジューラは $O(1)$ スケジューラと同様に各々の優先度ごとにレディキューを持ち，双方向循環リストでタスクを管理する．

例として，タスク τ_1 の優先度が 0 の場合，τ_1 がハイブリッド $O(1)$ スケジューラにより，どのようにスケジュールされるのかを述べる．まず，τ_1 がリリースした場合，SQ から τ_1 を取り除き，τ_1 の優先度を 0 に設定し，τ_1 の必須部分を実行可能状態として，RTQ に τ_1 を格納する．RTQ にあるタスク τ_1 の必須部分が完了した場合，RTQ から τ_1 を取り除き，τ_1 の優先度を 0 から 256 に変更し，τ_1 の付加部分を実行可能状態として，NRTQ に τ_1 を格納する．τ_1 の付加部分を中断する場合，NRTQ から τ_1 を取り除き，τ_1 の優先度を 256 から 0 に変更し，τ_1 の終端部分を実行可能状態として，RTQ に τ_1 を格納する．τ_1 の付加部分または終端部分が完了した場合，NRTQ または RTQ から τ_1 を取り除き，SQ に τ_1 を格納する．SQ ではタスクを起床時刻の昇順でソートして管理する．ハイブリッド $O(1)$ スケジューラは，優先度を論理和演算および論理積演算で変更可能であり，RTQ および NRTQ にタスクを挿入，削除する操作の計算量は $O(1)$ である．それゆえ，ハイブリッド $O(1)$ スケジューラは $O(1)$ スケジューラと比較してオーバヘッドの増加を抑えつつ，準固定優先度

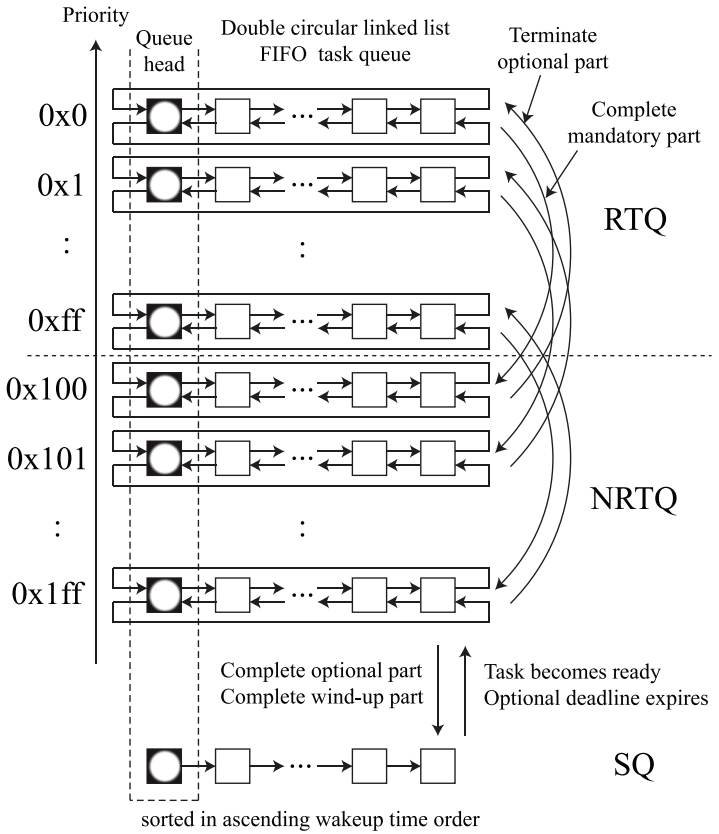


図 12　ハイブリッド $O(1)$ スケジューラ
Fig. 12　Hybrid $O(1)$ scheduler.

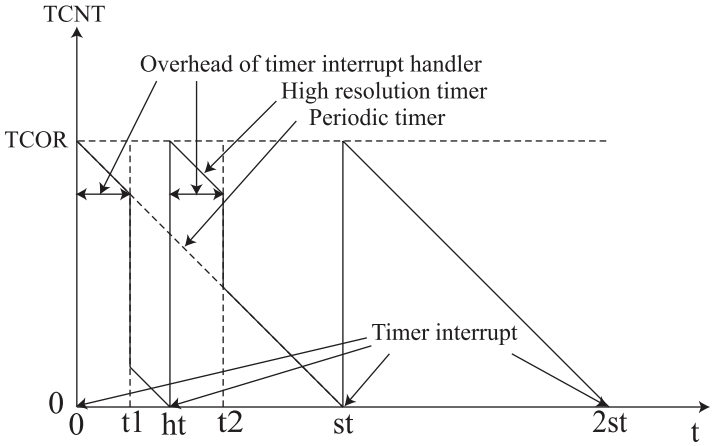


図 13　周期タイマと高精度タイマ
Fig. 13　Periodic timer and high resolution timer.

スケジューリングを実現する．

5.3　高精度タイマ

ハイブリッド $O(1)$ スケジューラを実装するためには，付加デッドライン時刻で付加部分を中断する必要がある．付加デッドライン時刻は，システムが時刻を更新するために周期ごとに発生するタイマ（周期タイマ）の周期より短い間隔で設定されることがあるので，付加デッドライン時刻でタイマ割込みを発生させるため，高精度タイマの実装が要求される．RT-Est では，周期タイマ割込みごとにインクリメントする `sys_jiffies` と，周期タイマ割込みの周期より短い間隔でタイマ割込みが発生した場合を考慮して，`sys_jiffies` からの経過時刻を表す `exec_jiffies` を用いて時間管理を行う．`sys_jiffies` の単位はタスクセット内のタスクの周期の最大公約数，`exec_jiffies` の単位はタスクセット内のタスクの各々の部分の最悪実行時間の最大公約数とする．`sys_jiffies` と `exec_jiffies` は単位が異なるので，`sys_jiffies` の単位から `exec_jiffies` の単位に変換する場合は `SU2EU` マクロ，`exec_jiffies` の単位から `sys_jiffies` の単位に変換する場合は `EU2SU` マクロを用いる．

本論文で RT-Est を実装するプロセッサは SH-4A である．SH-4A プロセッサでは，Time Management Unit（TMU）を用いて時間管理を行う．それゆえ，TMU を用いて高精度タイマを実現する手法について述べる．ここで，TMU の設定レジスタのアドレス `addr` から値を読み出し `data` に格納する場合には `data=read(addr)`，`addr` に `data` を書き込む場合には `write(addr,data)` と表記する．

図 13 に周期タイマと高精度タイマに関する動作例を示す．タイマカウンタレジスタ（TCNT）の推移を，周期タイマは破線，高精度タイマは実線で示す．タイマコンスタントレジスタ（TCOR）には周期タイマの周期 st と等しいタイマカウンタが設定されている．

まず，区間 [0,2 st) における周期タイマの動作について説明する．時刻 0 でタイマを開始すると TCNT の初期値は 0 なので，タイマ割込みが発生し，ハードウェアは TCNT に TCOR を設定する．TCNT は 0 になるまで減少する．時刻 st で TCNT は 0 になり，アンダフロー割込み（タイマ割込み）が発生する．同時に，ハードウェアは TCNT に TCOR を設定し，TCNT は 0 になるまで減少し，時刻 2 st でタイマ割込みが発生する．

次に，周期タイマの周期 st 間の時刻である時刻 ht（0<ht<st）でタイマ割込みを発生させる動作について説明する．時刻 0 でタイマ割込みが発生し，割込みハンドラを実行後の時刻である t1 で，タイマ割込みを時刻 ht で発生させるため，現在の TCNT から区間 [ht,st) と等しいタイマカウンタの値を減算する．時刻 ht でタイマ割込みが発生し，タイマ割込みハンドラを実行後に TCNT を再設定可能になった時刻 t2 で，タイマ割込みを時刻 st で発生させるため，現在の TCNT から区間 [0,ht) と等しいタイマカウンタの値を減算する．時刻 st でタイマ割込みが発生した後，ハードウェアは TCNT に TCOR を設定し，時刻 2 st でタイマ割込みを同様に発生させる．

図 14 に `do_timer` 関数を示す．`do_timer` 関数は，タイマ割込みハンドラ内で呼び出される関数である．`next_timer_interrupt` には，次にスケジューラを起動する時刻からタイマ割込みが発生した時刻を引いた差が `exec_jiffies` 単位で格納されている．つまり，`next_timer_interrupt` は，次の周期タイマ割込みの発生時刻，現在実行状態の拡張インプリサイスタスクの部分（必須部分，付加部分または終端部分）の完了時刻，タスクセット内の全タスクで最も早い付加デッドライン時刻および最も早い次のジョブのリリース時刻の最小値である．`do_timer` 関数では，まず周期タイマの周期 `st` ごとにシステム時刻を更新するため，`exec_jiffies` が `SU2EU(1)` の場合，`sys_jiffies` をインクリメントする．

周期タイマの周期 st より短い間隔でタイマ割込みを発生させる場合，すなわち `next_timer_interrupt` が `SU2EU(1)` 未満の場合について説明する．まず，`tmp` に区間 [0,SU2EU(1)-next_timer_interrupt) と等しいタイマカウンタの値を格納する．タイマが実行状態では，TCNT を変更できないので，TCNT を変更するため，`stop_timer` 関数でタイマを停止する．`read(TCNT)-tmp-TMU_ADJUST` と 0 の最大値を TCNT に設定する．`TMU_ADJUST`

```
do_timer(next_timer_interrupt) {
  if (exec_jiffies == SU2EU(1) {
    sys_jiffies++;
    exec_jiffies = 0;
  }
  if (next_timer_interrupt < SU2EU(1)) {
    tmp = EU2SU((SU2EU(1) - next_timer_interrupt) * read(TCOR));
    stop_timer();
    write(TCNT, max(0, read(TCNT) - tmp - TMU_ADJUST));
    start_timer();
    prev_exec_jiffies = exec_jiffies;
    exec_jiffies += next_timer_interrupt;
  } else {
    exec_jiffies = SU2EU(1);
    prev_exec_jiffies = 0;
  }
}
```

図 14　do_timer 関数
Fig. 14　do_timer function.

は stop_timer 関数を呼び出してから start_timer 関数を呼び出すまでの処理時間を表し，事前評価で計測した値を用いる．TMU_ADJUST を減算することで，TCNT を設定する処理にかかるオーバヘッドを考慮する．TCNT を設定後，start_timer 関数でタイマを開始する．また，周期タイマの周期 st 間でタイマ割込みを発生させる場合，現在実行状態のタスクが次のタイマ割込みまでに利用する時間を算出する必要がある．それゆえ，prev_exec_jiffies に現在の sys_jiffies からの経過時刻である exec_jiffies を，exec_jiffies に次のタイマ割込みが発生する時刻である exec_jiffies+next_timer_interrupt を設定する．したがって，現在実行状態のタスクが次にタイマ割込みが発生するまでに利用する時間は，exec_jiffies から prev_exec_jiffies を引いた差である．

これに対して，周期タイマの周期 st の間隔でタイマ割込みを発生させる場合，TCNT を再設定せず，exec_jiffies に SU2EU(1) を，prev_exec_jiffies に 0 を設定する処理を行うのみである．この場合，現在実行状態のタスクが次にタイマ割込みが発生するまでに利用する時間は SU2EU(1) である．

5.4　SIM アーキテクチャ

RT-Est はリアルタイムスケジューリングのシミュレーションを実行するためのアーキテクチャとして SIM を実装する．**図 15** に RT-Est の設定を示す．RT-Est の設定には，アーキテクチャを選択する Architecture module とアルゴリズムを選択する Algorithm module がある．Architecture module は SIM または SH，Algorithm module は RMWP，M-FWP または RM を選択する．SIM は User-Mode Linux のように，実際にタスクが消費する CPU 時間を計測するのではなく，Real-Time system SIMulator（RTSIM）[17] のようにスケジューリング理論どおりのシミュレーションを行う．また，SIM はアーキテクチャ共通部分のプログラムが SH のような他のアーキテクチャに利用可能であることが，RTSIM と異なる．ここで，SIM におけるアーキテクチャ共通部分の実装は SH のような他のアーキテクチャの実装に利用可能である．たとえば，ハイブリッド $O(1)$ スケジューラの実装はアーキテクチャ共通部分なので，SIM で実装したハイブリッド $O(1)$ スケジューラのソースコードを他のアーキテクチャに利用可能である．また，Algorithm module が SIM を選択した場合で，たとえば Linux における gcc で RT-Est をコンパイルすると，Linux 上で実行可能なバイナリファイルを生成する．それゆえ，RT-Est の SIM では，gdb 等のデバッグツールを利用可能であるので，リアルタイムスケジューリングのシミュレーションだけでなく，OS の開発効率を向上させることに有効である．

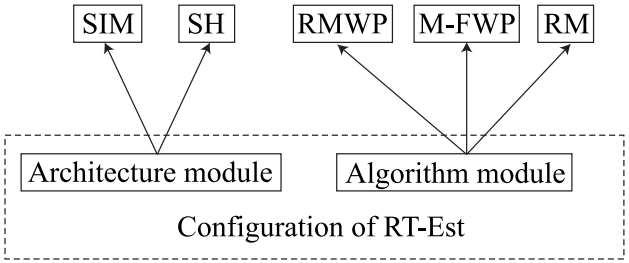


図 15　RT-Est の設定
Fig. 15　Configuration of RT-Est.

6. 評　　価

本章では RT-Est における準固定優先度スケジューリングの有効性をシミュレーションおよび実機の両方の観点から評価する．評価に用いるアルゴリズムは RMWP と M-FWP および RM である．シミュレーション評価は SIM，実機評価は SH で行う．**表 2** に RT-Est のコードサイズを示す．RMWP は付加デッドライン時刻の中断処理，M-FWP は付加部分

表 2　RT-Est のコードサイズ
Table 2　Code sizes of RT-Est.

アルゴリズム	SIM のコードサイズ [byte]	SH のコードサイズ [byte]
RMWP	148,223	181,268
M-FWP	144,314	177,212
RM	136,080	166,564

の割当て可能時間の算出処理を行うため，RM よりコードサイズが大きくなったと考えられる．

6.1　シミュレーション評価

6.1.1　シミュレーションの評価環境

シミュレーションは 5.4 節で述べた RT-Est の SIM アーキテクチャにより行う．評価には 1,000 個のハーモニックタスクセットを用いる．k 番目のハーモニックタスクセットのシミュレーション時間はハーモニックタスクセット内の全タスクの周期の最小公倍数であるハイパーピリオド H_k とする．本論文の対象とする自律移動ロボットでは，表 1 で示したように周期が大きく異なるタスクが存在する．表 1 で示した画像処理タスクの周期は 200 ms とタスクセット内で最も長いが，文献 12)，13) で使用している CPU の周波数は 100 MHz と低性能なため，長い周期を設定している．これに対して，本論文で用いる CPU の詳細は後述するが CPU の周波数は 600 MHz と高性能なため，画像処理タスクの周期をタスクの周期の最大値である 32 ms 程度に設定して，タスクセットを生成することは妥当である．それゆえ，タスク τ_i の周期 T_i は 1 ms から 32 ms まで 2 の乗数おきで設定する．CPU 利用率 U_i は 0.02 から 0.25 まで 0.01 おきで設定する．システム全体の CPU 利用率 U は 0.3 から 1 まで 0.05 おきで測定する．U_i および U は必須部分および終端部分のみを含む．各々の U_i は一様分布により必須部分および終端部分に分割する．これらとは別に，付加部分の要求実行時間 o_i の CPU 利用率 o_i/T_i を 0.1，0.2，0.3 の 3 種類を基準値として設定し，各々の基準値の ±0.05 の範囲で一様分布により生成する．つまり，付加部分の CPU 利用率の範囲はそれぞれ $[0.05, 0.15]$，$[0.15, 0.25]$，$[0.25, 0.35]$ となる．評価結果にはそれぞれ RMWP-10，RMWP-20，RMWP-30 のように表記する．ただし，o_i が 0 の場合，評価結果には RMWP と表記する．自律移動ロボットでは，タスクの実際実行時間が最悪実行時間より短い場合があることを想定して，実際実行時間と最悪実行時間の比 ACET/WCET は $[0.5, 1.0]$，$[0.75, 1.0]$，1.0 の 3 種類で評価を行う．疑似乱数生成器には高い均等分布特性を持つ Mersenne Twister [18] を用いる．

本論文で評価指標として用いる，付加部分の実行割合（Reward Ratio），コンテキストスイッチの頻度（Switch Ratio），RFJ をタスクの周期で正規化した割合（RFJ Ratio），最短周期タスクの RFJ である SPJ をタスクの周期で正規化した割合（SPJ Ratio）をそれぞれ下式に示す．

$$\text{Reward Ratio} = \frac{\sum_k \sum_i \frac{T_i}{H_k} \sum_j \frac{o_{i,j}}{o_i}}{\#\text{ of tasks in successfully scheduled task sets}}$$

$$\text{Switch Ratio} = \frac{\sum_k \frac{\#\text{ of context switches}}{H_k}}{\#\text{ of successfully scheduled task sets}}$$

$$\text{RFJ Ratio} = \frac{\sum_k \sum_i \frac{RFJ_i}{T_i}}{\#\text{ of tasks in successfully scheduled task sets}}$$

$$\text{SPJ Ratio} = \frac{\sum_k \frac{RFJ_1}{T_1}}{\#\text{ of successfully scheduled task sets}}$$

ここで，ハーモニックタスクセットにおける，RMWP と M-FWP および RM のスケジュール可能上限は 1 なので，スケジュール成功率に関する評価は行わない．タスクセット全体のジッタの評価である RFJ Ratio だけでなく SPJ Ratio を評価する理由は，本論文の対象とする自律移動ロボットで高精度な動作を実現するためには，ハーモニックタスクセット内で最短周期かつ許容するジッタが低いタスクであるモータ制御タスクのジッタが重要になるからである．**表 3** にシステム全体の CPU 利用率ごとのタスク数を示す．Reward Ratio と RFJ Ratio の分母は表 3 の値である．各々の CPU 利用率ごとのタスク数は数千個になるので，グラフの信頼性は高いといえる．これに対して，Switch Ratio と SPJ Ratio の分母はシステム全体の CPU 利用率ごとに評価するタスクセット数の 1,000 である．

6.1.2　シミュレーションの評価結果

図 16 に Reward Ratio の評価結果を示す．図 16 (a)，図 16 (b)，図 16 (c) のいずれにおいても，RMWP-10 と RMWP-20 および RMWP-30 の Reward Ratio は，それぞれ M-FWP-10 と M-FWP-20 および M-FWP-30 の Reward Ratio と同程度である．ここで，一般タスクセットにおける RMWP の Reward Ratio は，M-FWP の Reward Ratio より低い[15]．これに対して，M-FWP は動的に付加部分の割当て可能時間を算出するが，RMWP

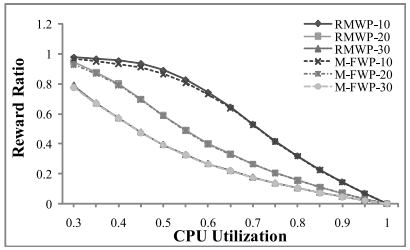
表 3　各々のシステム全体の CPU 利用率ごとのタスク数
Table 3　# of tasks in each system CPU utilization.

CPU 利用率	タスク数
0.3	2,799
0.35	3,155
0.4	3,495
0.45	3,864
0.5	4,262
0.55	4,626
0.6	5,041
0.65	5,339
0.7	5,707
0.75	6,130
0.8	6,425
0.85	6,821
0.9	7,151
0.95	7,579
1.0	8,022

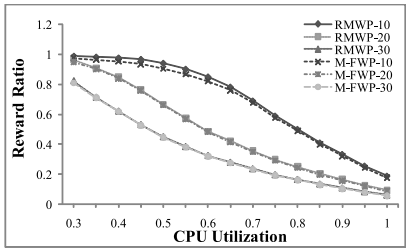
は静的に最適な付加デッドラインを算出するにもかかわらず，ハーモニックタスクセットにおいて，RMWP は M-FWP と同程度の Reward Ratio を発揮することができる．

図 17 に Switch Ratio の評価結果を示す．RMWP-10，RMWP-20，RMWP-30 の順で Switch Ratio が低い理由は，付加部分の CPU 利用率が高いほどコンテキストスイッチを行わずに，タスクの必須部分と付加部分および終端部分を連続して実行する場合が多いからである．M-FWP も RMWP と同様に付加部分の CPU 利用率が高いほど M-FWP の Switch Ratio は低くなる．付加部分を実行した場合，RMWP および M-FWP の Switch Ratio は RM の Switch Ratio より高い．したがって，拡張インプリサイス計算モデル用アルゴリズムは，一般計算モデル用アルゴリズムより Switch Ratio は高くなってしまう．また，RMWP の Switch Ratio が M-FWP の Switch Ratio より高い理由は，M-FWP はタスクの付加部分が完了した場合，続けて終端部分を実行するが，RMWP はタスクの付加部分が付加デッドライン時刻までに完了した場合，付加デッドライン時刻まで終端部分の実行を遅らせるからである．

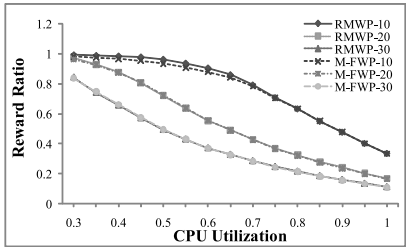
図 18 に RFJ Ratio の評価結果を示す．図 18 (a) において，RMWP と RM および M-FWP の RFJ Ratio はつねに 0 であるが，M-FWP-10 と M-FWP-20 および M-FWP-30 の RFJ Ratio は大幅に高い．また，ACET/WCET の変動の幅が大きくなるほど，RMWP と RM の RFJ Ratio は高くなる．これに対して，M-FWP-10 と M-FWP-20 および M-FWP-30 の RFJ Ratio は ACET/WCET の変動の幅に依存せず，大幅に高い．興味深いことに，図 18 (b) と図 18 (c) に関して，M-FWP の RFJ Ratio は RM の RFJ Ratio より低い．M-FWP は付加部分を実行しない場合，EDF と同じスケジュールを生成する．それゆえ，タスクの実際実行時間が最悪実行時間より短い場合，EDF の RFJ Ratio は RM の RFJ Ratio より低い．



(a) ACET/WCET = 1.0

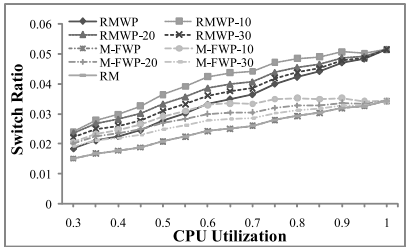


(b) ACET/WCET = [0.75,1.0]

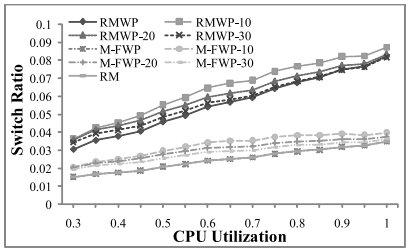


(c) ACET/WCET = [0.50,1.0]

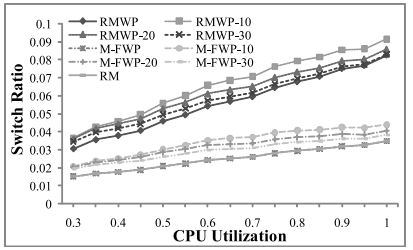
図 16　Reward Ratio
Fig. 16　Reward Ratio.



(a) ACET/WCET = 1.0

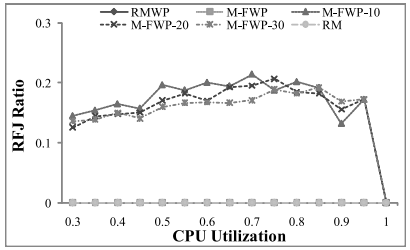


(b) ACET/WCET = [0.75,1.0]

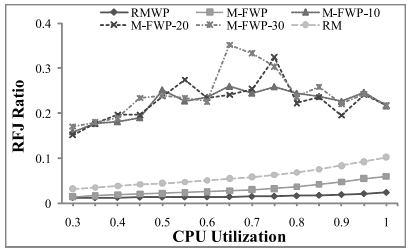


(c) ACET/WCET = [0.50,1.0]

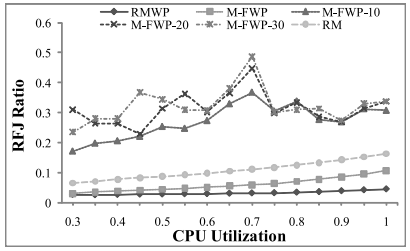
図 17　Switch Ratio
Fig. 17　Switch Ratio.



(a) ACET/WCET = 1.0

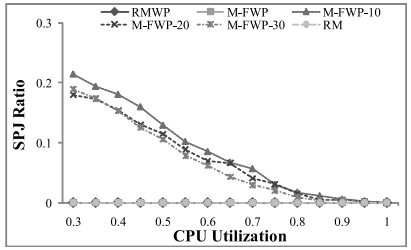


(b) ACET/WCET = [0.75,1.0]

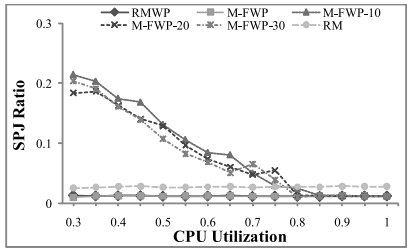


(c) ACET/WCET = [0.50,1.0]

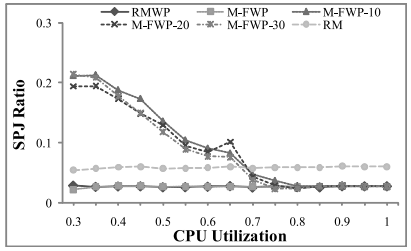
図 18　RFJ Ratio
Fig. 18　RFJ Ratio.



(a) ACET/WCET = 1.0



(b) ACET/WCET = [0.75,1.0]



(c) ACET/WCET = [0.50,1.0]

図 19　SPJ Ratio
Fig. 19　SPJ Ratio.

図 19 に SPJ Ratio の評価結果を示す．図 19 (a) では，RMWP と M-FWP の SPJ Ratio は RM の SPJ Ratio と等しいが，図 19 (b) および図 19 (c) では，RMWP と M-FWP の SPJ Ratio は RM の SPJ Ratio より低い．文献 19) では，最短周期タスクのジッタが低いことに関して，RM は EDF より優位であるとであると主張しているが，これは実際実行時間が最悪実行時間と等しい場合のみ成立し，実際実行時間が最悪実行時間より短い場合，EDF は RM より優位である．また，M-FWP-10 と M-FWP-20 および M-FWP-30 の SPJ Ratio は RFJ Ratio と同様に依然として大幅に高い．また，M-FWP-10 と M-FWP-20 および M-FWP-30 の SPJ Ratio は，CPU 利用率が高くなるほど低くなり，M-FWP の SPJ Ratio と同程度になる CPU 利用率は，図 19 (a) の場合は 1.0，図 19 (b) の場合は 0.85，図 19 (c) の場合は 0.75 である．すなわち，ACET/WCET の変動の幅が大きくなるほど，CPU 利用率が高くなることによる，M-FWP-10 と M-FWP-20 および M-FWP-30 の SPJ Ratio の低下が早くなる．

表 4　LEPRACAUN の仕様
Table 4　Specification of LEPRACAUN.

CPU	SH-7785(R8A77850BA) 600 MHz
FLASH ROM	NOR FLASH ROM 8 MB
SDRAM	DDR2 SDRAM 256 MB
I-Cache	32 KB 4-way set associative
D-Cache	32 KB 4-way set associative
デジタル I/O	各 16 チャネル
寸法	幅 92 mm ×奥行 55 mm ×高さ 20.5 mm

表 5　拡張 I/O ボードの仕様
Table 5　Specification of extended I/O board.

A/D	分解能 12 ビット 8 チャネル
D/A	分解能 12 ビット 8 チャネル
エンコーダ	A/B/Z 相 各 6 チャネル
DI/O	各 16 チャネル
寸法	幅 92 mm ×奥行 55 mm ×高さ 11.1 mm

6.2　実機評価

6.2.1　実機の評価環境

実機評価では，SH-4A プロセッサである SH-7785 に多チャネル I/O を搭載することで，ロボット制御用にカスタマイズした LEPRACAUN [20] を用いる．**表 4** に LEPRACAUN の仕様を示す．LEPRACAUN は名刺と同程度のサイズであり非常に小型なので，ロボットの腕モジュール等に搭載して利用することが可能である．また，**表 5** に示す LEPRACAUN 専用の拡張 I/O ボードを LEPRACAUN に増設することで，ロボット制御に必要な各種 I/O を利用可能になり，多様な環境で動作する自律移動ロボットに適していると考えられる．RT-Est のコンパイルには，SH4-Linux 用のクロスコンパイラである `sh4-linux-gcc`

表 6　アルゴリズムの評価回数
Table 6　# of evaluations in each algorithm.

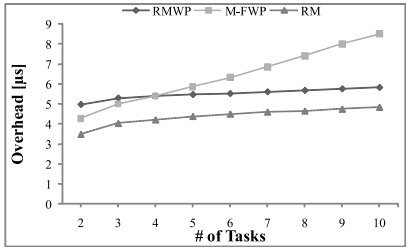
タスク数	RMWP	M-FWP	RM
2	87,888	74,812	64,142
3	369,714	296,372	239,170
4	607,000	464,766	366,380
5	773,816	567,092	454,044
6	845,986	594,736	493,258
7	785,002	532,752	460,038
8	553,376	364,884	327,788
9	284,996	183,578	168,146
10	129,204	81,208	75,818

の version 4.2 を用いる．`sh4-linux-gcc` クロスコンパイラの最適化オプションは`'-O2'`とする．6.1 節で生成したタスクセットを用いる．タスク τ_i の付加部分の CPU 利用率 o_i/T_i の範囲は $[0, 0.3]$ とする．また，タスクの最悪実行時間にオーバヘッドを含む．

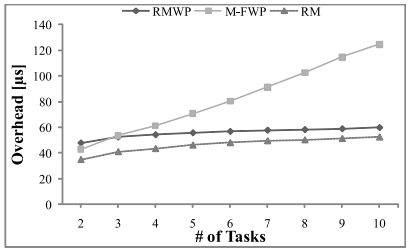
実機の評価指標はアルゴリズムのオーバヘッドである．アルゴリズムのオーバヘッドの評価では，5.3 節で示したタイマ割込みが発生してから，スケジューラを呼び出し，タイマ割込みから復帰するまでの処理時間の平均と最大を計測する．**表 6** にアルゴリズムの平均，最大オーバヘッドのタスクごとの評価回数を示す．評価回数は数万〜数十万となるので，信頼性は高いといえる．アルゴリズムごとにスケジューラの呼び出し回数が異なるので，計測したオーバヘッドの合計を各々の評価回数で除算した値を平均オーバヘッドとする．また，各々のオーバヘッドの評価の最大値を最大オーバヘッドとする．

拡張インプリサイス計算モデルは実際実行時間が最悪実行時間より短い場合を考慮したモデルである．また，現在研究開発されている最悪実行時間解析ツールは命令キャッシュおよびデータキャッシュを考慮した解析を行っていない[21),22)]．それゆえ，最悪実行時間はキャッシュを無効にした状態で見積もる必要がある．キャッシュによる実行時間の短縮により，どれだけ付加部分の実行に利用可能であるかを評価する．キャッシュの評価は，命令キャッシュとデータキャッシュを両方有効（ID-Cache）と両方無効（No-Cache）の 2 種類で行う．ID-Cache では各々のタスクセットのスケジュールを開始する前に命令キャッシュおよびデータキャッシュを初期化する．タスク数を 2 から 10 まで変化させて評価を行う．

事前評価として，図 14 で示した，タイマを停止する `stop_timer` 関数からタイマを開始する `start_timer` 関数までのオーバヘッドである `TMU_ADJUST` を 100 回計測し，平均値を計測した．`TMU_ADJUST` は，ID-Cache では 0.92 μs，No-Cache では 1.14 μs になった．計測したオーバヘッドを，図 14 で示した `TMU_ADJUST` にそれぞれ設定し，ID-Cache および No-Cache の評価を行う．

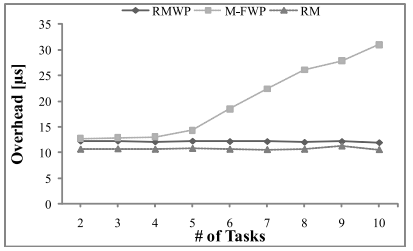


(a) ID-Cache

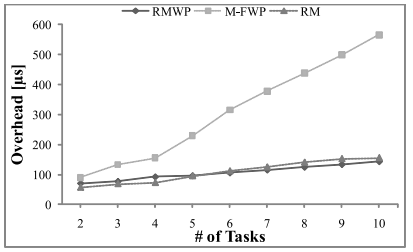


(b) No-Cache

図 20　アルゴリズムの平均オーバヘッド
Fig. 20　Average overhead of algorithm.



(a) ID-Cache



(b) No-Cache

図 21　アルゴリズムの最大オーバヘッド
Fig. 21　Maximum overhead of algorithm.

6.2.2　実機の評価結果

図 20 にアルゴリズムの平均オーバヘッドを示す．図 20 (a) と図 20 (b) の両方の場合，RMWP と RM の平均オーバヘッドはタスク数が増加しても，ほぼ一定を保っている．また，RMWP の平均オーバヘッドが RM の平均オーバヘッドより若干高い理由は，準固定優先度スケジューリング向けハイブリッド $O(1)$ スケジューラは，固定優先度スケジューリング向け $O(1)$ スケジューラと比較して，図 12 で示した必須部分の完了処理および付加部分の中断処理等を行うからである．これに対して，M-FWP の平均オーバヘッドが，図 20 と図 20 の両方の場合でタスク数が増加するほど大幅に高くなる理由は，レディキューにタスクを挿入，削除する操作および付加部分に割当て可能時間を動的に算出する処理の計算量が $O(n)$ になるからである．

図 21 にアルゴリズムの最大オーバヘッドを示す．図 21 (a) と図 21 (b) の両方の場合，RMWP と RM の最大オーバヘッドはタスク数が増加しても，ほぼ一定を保っている．これに対して，M-FWP の最大オーバヘッドは，M-FWP の平均オーバヘッドの評価と同様に，図 21 (a) と図 21 (b) の両方の場合で，タスク数が増加するほど大幅に高くなる．

タスク数が増加してもオーバヘッドがほぼ一定を保っている RMWP および RM は，タスク数が増加するとオーバヘッドが大幅に高くなる M-FWP よりスケーラビリティの観点から優れている．また，ID-Cache の場合のオーバヘッドは，No-Cache の場合のオーバヘッドと比較して大幅に低い．最悪実行時間解析ツールは，命令およびデータキャッシュを考慮していないので，実際実行時間は最悪実行時間より大幅に短くなり，RMWP は大幅に余った時間を付加部分に利用可能なことが，RM より優れている．それゆえ，RMWP は自律移動ロボットに適している．

7. 結　　論

本論文では，準固定優先度スケジューリング向け RTOS である RT-Est を提案した．RT-Est に準固定優先度スケジューリングアルゴリズム RMWP を実装し，シミュレーションおよび実機評価の両方を行った．シミュレーション評価では，RMWP は M-FWP および RM と比較してコンテキストスイッチの頻度が高くなってしまうが，ジッタに関して RMWP は M-FWP および RM より高い性能を示した．さらに，最適な付加デッドラインを算出する RTA-OODH により，付加部分の実行割合に関して RMWP は M-FWP と同程度の性能を示した．実機評価では，M-FWP のオーバヘッドはタスク数が増加するほど大幅に高くなってしまったが，RMWP のオーバヘッドは RM のオーバヘッドと比較して若干高くなるにとどまった．シミュレーションおよび実機評価の両方を考慮した結果，RMWP は自律移動ロボットに適していることを示した．

今後の課題として，RT-Est を自律移動ロボットに利用する．具体的には，表 1 で示したタスクを RMWP でスケジュールし，RT-Est の実用性を実証する．また，RT-Est および RMWP をマルチプロセッサ用に拡張する．Linux カーネル 2.6 における $O(1)$ スケジューラはマルチプロセッサでも高いスケーラビリティを発揮するので[14]，RT-Est におけるハイブリッド $O(1)$ スケジューラも同様の性質があると考えられる．さらに，マルチプロセッサにおける付加デッドラインの算出手法および RMWP のスケジュール可能上限を解析する予定である．

謝辞　本研究の一部は科学技術振興機構 CREST の支援による．本研究の一部は文部科学省グローバル COE プログラム「環境共生・安全システムデザインの先導拠点」によるものである．

参 考 文 献

1) Kanehiro, F., Hirukawa, H. and Kajita, S.: OpenHRP: Open Architecture Humanoid Robotics Platform, *The International Journal of Robotics Research*, Vol.23, No.2, pp.155–165 (2004).
2) Center, F.R.T.: HallucII. http://www.furo.org/ja/robot/halluc2/index.html
3) Liu, C. and Layland, J.: Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, *J. ACM*, Vol.20, pp.46–61 (1973).
4) Leung, J. and Whitehead, J.W.: On the Complexity of Fixed Priority Scheduling of Periodic Real-Time Tasks, *Performance Evaluation*, Vol.2, No.4, pp.237–250 (1982).
5) Audsley, N.C., Burns, A., Richardson, M.F., Tindell, K. and Wellings, A.: Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling, *Software Engineering Journal*, Vol.8, No.5, pp.284–292 (1993).
6) 石綿陽一，松井俊浩，國吉康夫：高度な実時間処理機能を持つ Linux の開発，第 16 回日本ロボット学会学術講演会予稿集，pp.355–356 (1998).
7) TOPPERS プロジェクト：TOPPERS/ASP カーネル．http://www.toppers.jp/asp-kernel.html
8) Lin, K., Natarajan, S. and Liu, J.: Imprecise Results: Utilizing Partial Computations in Real-Time Systems, *Proc. 8th IEEE Real-Time Systems Symposium*, pp.210–217 (1987).
9) Kobayashi, H. and Yamasaki, N.: An Integrated Approach for Implementing Imprecise Computations, *IEICE trans. information and systems*, Vol.86, No.10, pp.2040–2048 (2003).
10) Kobayashi, H., Yamasaki, N. and Anzai, Y.: Scheduling Imprecise Computations with Wind-up Parts, *Proc. 18th International Conference on Computers and Their Applications*, pp.232–235 (2003).
11) Kobayashi, H. and Yamasaki, N.: RT-Frontier: A Real-Time Operating System for Practical Imprecise Computation, *Proc. 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp.255–264 (2004).
12) Taira, T. and Yamasaki, N.: Functionally Distributed Control Architecture for Autonomous Mobile Robots, *Journal of Robotics and Mechatronics*, Vol.16, No.2, pp.217–224 (2004).
13) Taira, T., Kamata, N. and Yamasaki, N.: Design and Implementation of Reconfigurable Modular Robot Architecture, *Proc. 2005 IEEE/RSJ International Con-*

ference on Intelligent Robots and Systems, pp.3566–3571 (2005).
14) Molnar, I.: Ultra-Scalable $O(1)$ SMP and UP Scheduler (2002). http://www.uwsg.iu.edu/hypermail/linux/kernel/0201.0/0810.html
15) Chishiro, H., Takeda, A., Funaoka, K. and Yamasaki, N.: Semi-Fixed-Priority Scheduling: New Priority Assignment Policy for Practical Imprecise Computation, *Proc. 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp.339–348 (2010).
16) Sha, L., Lehoczky, J.P. and Rajkumar, R.: Solutions for Some Practical Problems in Prioritized Preemptive Scheduling, *Proc. 7th IEEE Real-Time Systems Symposium*, pp.181–191 (1986).
17) Palopoli, L., Lipari, G., Lamastra, G., Abeni, L., Bolognini, G. and Ancilotti, P.: An object oriented tool for simulating distributed real-time control systems, *Software–Practice and Experience*, Vol.32, No.9, pp.907–932 (2002).
18) Matsumoto, M. and Nishimura, T.: Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator, *ACM Trans. Modeling and Computer Simulations*, Vol.8, No.1, pp.3–30 (1998).
19) Buttazzo, G.C.: Rate Monotonic vs. EDF: Judgment Day, *Real-Time Systems*, Vol.29, No.1, pp.5–26 (2005).
20) ゼネラルロボティックス株式会社：LEPRACAUN. http://www.generalrobotix.com/product/LEPRACAUN-CPU.htm
21) Yamamoto, K., Ishikawa, Y. and Matsui, T.: Portable Execution Time Analysis Method, *Proc. 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp.267–270 (2006).
22) Ermedahl, A.: A Modular Tool Architecture for Worst-Case Execution Time Analysis, Ph.D. Thesis, Uppsala University (2003).

(平成 22 年 8 月 4 日受付)
(平成 22 年 11 月 30 日採録)

千代　浩之（学生会員）
2008 年慶應義塾大学理工学部情報工学科卒業．2010 年同大学大学院理工学研究科開放環境科学専攻修士課程修了．現在，同博士課程に在籍．リアルタイムシステム，オペレーティングシステム，分散ミドルウェア等の研究に従事．

山﨑　信行（正会員）
1991 年慶應義塾大学理工学部物理学科卒業．1996 年同大学大学院理工学研究科計算機科学専攻博士課程修了．博士（工学）．同年電子技術総合研究所入所．1998 年 10 月慶應義塾大学理工学部情報工学科助手．同専任講師を経て，2004 年 4 月より同助教授（現，准教授）．リアルタイムシステム，プロセッサアーキテクチャ，並列分散処理，システム LSI，ロボティクス等の研究に従事．日本ロボット学会，IEEE 各会員．