ソフトウェア基礎論　37－22
プログラミング言語　27－22
(1990. 12. 14)

分散環境における Ada プログラムの実時間ランデブについて

藤井義巳 荒木啓二郎 平原正樹

九州大学工学部

概要

Ada は実時間組み込みシステムの開発を主たる目的として開発された言語である。しかし、Ada の実時間処理機能はそのままでは実用的な実時間処理の記述には適さない部分が多い。実用的な実時間処理モデルについてはすでに多くの研究がなされていて、いくつかのスケジューリングアルゴリズムが提案されている。本研究ではそういった実用的な実時間処理モデルとスケジューリングアルゴリズムを示し、そのモデルに基づいて Ada の言語仕様に変更を加える。更にそれを分散環境上に実現する方法を提案する。

Real-Time Rendezvous in Ada Programs on Distributed Environment

Yoshimi FUJII†, Keijiro ARAKI† and Masaki HIRABARU‡

† Department of Computer Science and Communication Engineering, Kyushu University
‡ Computation Center, Kyushu University
Hakozaki 6-10-1, Higashi-ku, Fukuoka 812 Japan

Abstract

Programming language Ada was designed to develop embedded real-time systems. However, Ada has several shortcommings for describing practical real-time processing. In this paper, we first study a model for real-time processing and a real-time scheduling algorithm. According to the model and the algorithm, we propose a small modification on the tasking semantics of Ada and present a design for implementing real-time rendezvous on a distributed environment.

1. はじめに

Adaは実時間組み込みシステムの開発を主たる目的として開発された言語であるにもかかわらず、実時間処理の要求を満たしているとは言い難い。実時間処理の研究はこれまでにも数多くなされていて、いくつかの実用的なスケジューリングアルゴリズムも提案されている。しかし現在のAdaはそのままでは多くの問題点をかかえていて、[1] それらのスケジューリングアルゴリズムを直接用いた実時間処理プログラムを開発することが困難である。言語の設計時に実時間処理のモデルを明確に設定しなかったことがその原因の一つだと考えられる。

我々の研究室では並列・分散アプリケーションを記述するプログラミング言語がどのような機能を持つべきか、それをサポートするオペレーティングシステムはどのような機能を用意すべきかを調べるという研究を行なっている。[2] その研究の一環として、Adaのサブセット処理系ParaDisEとその実行を支援するオペレーティングシステムDaOS[3]を分散環境上に実現し、それを用いて分散アプリケーションを実際に記述し、評価を行なっている。現在ParaDisEにはAdaのタスク機能のうち、実時間処理を記述するために用意された三つの機能である遅延選択肢付き選択待機、即時エントリ呼び出し、時限エントリ呼び出しを実現していない。これらの分散環境上での実現を目指して研究を進める過程で、Adaの実時間処理モデルそのものの実用性を見直す必要性が生じた。そこで、分散環境上で実用的な実時間処理システムを記述できるモデルについて考察し、そのモデルを適用できるようにAdaの言語仕様に変更を行なった。

本論文の構成はまず2章で実時間処理のモデルを設定し、スケジューリングアルゴリズムについて検討する。3章ではAdaの言語仕様のうち、実時間処理に関する部分について報告し、2章で検討した実時間処理モデルのAdaへの適用法について説明する。4章では3章で説明した仕様を分散環境上に実現する方法を具体的に示す。

2. 実時間タイム処理

2.1 実時間処理のモデル

実時間処理は次の2種類に分類される。[4]

i) 周期的な処理

ii) イベント駆動の処理

周期的な処理は、タイマ割り込み等により、ある決まった時間間隔で処理の要求が発生し、次の要求が発生するまでにその処理を完了しなければならないような処理である。それに対してイベント駆動の処理では、外界からの要求で任意の時刻にイベントが発生し、対応する処理をイベントが発生してから決められた時間内に完了しなければならない。前者は後者の特殊な場合と考えられる。すなわち、実時間クロックを用いて周期的にイベントを発生させることにより、前者の処理はシミュレートされる。本研究ではより一般的と考えられる後者を研究の対象とする。

実時間処理の単位はタスクと呼ばれ、タスクτの実時間に関する性質は次の三つのパラメタで表現される。

要求の発生時刻	–	S
処理の期限 (deadline)	–	D
実質処理時間	–	C

Sはイベント等、タスクに対して処理の要求が発生した時刻である。その処理は時刻Dまでに完了することが望まれる。CはCPUがその処理を中断なしに行った場合にかかる時間である。実時間処理の例を図示すると、図1のように表せる。直接実時間処理を行なうのがサーバタスクである。イベントハンドラは外界からの割り込みやタイマイベントを受け取り、対応するサーバタスクに期限を指定して処理を依頼する。タイママネージャは周期的な処理のためにタイマイベントを発生したり、処理の期限を守れなかったタスクの処分を行なうタイミングを作り出す。その他に処理の期限が無い($D = \infty$)バックグラウンドタスクが存在する。サーバタスクはバックグラウンドタスクから処理の要求を受けることもある。

一つのCPUが一度に処理するタスクが一つならば簡単である。処理が時刻Dに完了するには、

$$C \leq (D - S)$$

であればよい。しかし、この仮定は現実的でない。実際には処理の途中でより緊急度の高い処理の要求が発生した時のことを考慮しなければならない。そこでタスクのスケジューリングが必要となってくる。

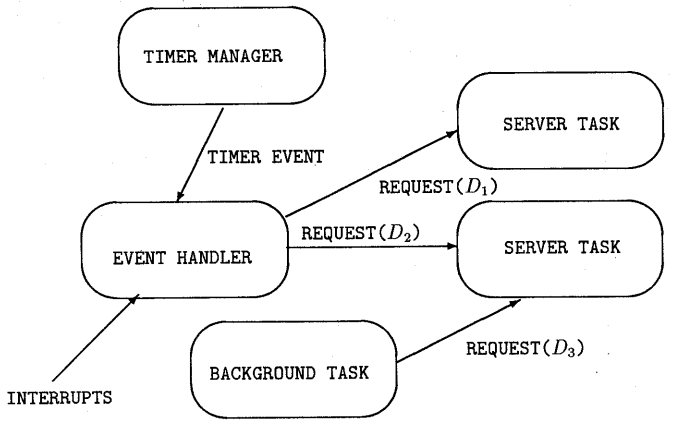

図1 実時間処理の概念図

2.2 スケジューリングアルゴリズム

実時間システムを実現するためにはタスクにその緊急度に応じて優先度を設定し、横取り (preemptive) スケジューリングを行なわなければならない。ある時刻にn個のタスク $\tau_1, \tau_2, ..., \tau_n$ が存在し、その優先度をそれぞれ $P_1, P_2, ..., P_n$ とし、$P_1 \leq P_2 \leq ... \leq P_n$ である時、適切なスケジューリングが行なわれたなら、

$$\sum_{i=1}^{k} C_i \leq D_k$$

がすべての $k(1 \leq k \leq n)$ について満足される時に限り、すべてのタスク τ_i がその処理の期限 D_i までに終ることが保証される。このことをスケジューリングが可能である (schedulable) という。次の二つは適切なスケジューリングアルゴリズムである。[5] [6]

i) D が小さい順 (earliest-deadline-first) スケジューリング

ii) $D-S-C$ (余裕時間) が小さい順スケジューリング

また、周期的なタスクのスケジューリングとしてはRate monotonic scheduling[7] がよく知られている。n個のタスク $\tau_1, \tau_2, ..., \tau_n$ の周期をそれぞれ $T_1, T_2, ..., T_n$ とし、$T_1 \leq T_2 \leq ... \leq T_n$ とする時、タスク τ_i の優先度は T_i が小さいほど高く、

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

が満たされる時に限ってスケジューリングが可能である。

前述のとおり本研究ではイベント駆動の実時間処理を対象とする。スケジューリングアルゴリズムは上述のi)「D が小さい順スケジューリング」を選んだ。その理由はスケジューリングのパラメタとして、S や D と比べて求めるのが困難な C を必要としないからである。スケジューリングの際、処理期限の時刻 D の値をそのまま優先度の値として用いることにする。この時、D の早いタスクほど優先度が高いことになる。以後本論文で $D_1 < D_2$ という表記は時刻 D_1 が時刻 D_2 より早いということではなく、優先度 D_1 が優先度 D_2 より低いことを表す。

2.3 優先度の継承

前節のモデルではサーバ間の同期は考慮しなかったが、現実にはサーバ同士の同期も考慮しなくてはならない。三つのタスク τ_1,τ_2,τ_3 が存在し、その優先度をそれぞれ D_1,D_2,D_3 とし、$D_1 < D_2 < D_3$ と仮定する。ある時、τ_1 が τ_3 にブロックされたとする。タスクが自分より優先度の低いタスクにブロックされる現象は、共有資源のアクセスのための排他制御を行なう時等に発生する。そのため、排他制御の行なわれるきわどい領域の実行は実時間システムではできるだけ短い時間に行なわれなければならない。τ_3 がきわどい領域を実行中、より優先度の高い τ_2 がそのきわどい領域に無関係であればいつでも τ_3 を横取りでき、このとき τ_1 は自分より優先度の低い τ_2 に間接的にブロックされることになる。この現象を優先度の逆転という。一旦これが起こるといつか τ_2 の実行が終るか、τ_2 がブロックされるかして τ_3 に実行が移り、τ_3 がきわどい領域の実行を終らないない限り τ_1 は実行を再開することができない。緊急度の高いタスクがこのような優先度の逆転などによって不当に長い時間ブロックされることは、実時間システムでは起こってはならない。この優先度の逆転を避けるために考えられたのが優先度の継承である。あるタスクが自分より優先度の高いタスクをブロックした場合、そのタスクの優先度を継承し、そのタスクをブロックしている間は継承した高い優先度で実行を行うものである。この考え方を更に進めた優先度シーリング (priority ceiling) の提案と Ada への適用例も報告されている。[8][9] 本研究では優先度の逆転を防止する目的で、優先度の継承を採用する。

次章では、Ada が本来提供する実時間処理機能について考察した後、本章で説明した実時間処理のモデルを Ada の言語仕様に適用する。

3. Ada の実時間モデル

3.1 Ada の本来の実時間処理

Ada は実時間処理のための機能として次の三つの構文を用意している。

i) 遅延選択肢付きの選択待機 (selective wait)

ii) 即時エントリ呼び出し (conditional entry call)

iii) 時限エントリ呼び出し (timed entry call)

これらの詳細は Ada 基準文法書 [10] を参照されたい。Ada のこれらの機能を用いて記述される実時間処理とは、即時に（あるいはある一定時間内）に処理を開始できなかった場合、代替の処理を行なうことができるということである。また、Ada のタスクは整数値の優先度を持つことができるが、優先度を実行中に変えることはできない。また、エントリ待ち行列は優先度順ではなく、先着順 (FIFO) に処理されるため、複数の処理要求があった場合、処理の受け付け順は優先度に関係なく先着順となる。このように Ada の実時間処理の考え方は、前章で示した実時間処理のモデルとはかけはなれている。次節では前章の実時間モデルを Ada の言語仕様に適用する方法を考察する。

3.2 時限エントリ呼び出し

Ada には前節で述べたような実時間処理のタスクモデルを直接に記述する構文は存在しない。しかし、Ada の時限エントリ呼び出しは見方によっては Ada 風の実時間処理の機能を提供していると考えられる。Ada の時限エントリ呼び出しは次のような形をしている。[10]

```
select
    EntryCall(Parameters);
or
    delay d;
end select;
```

このプログラム片の動作は、エントリ呼び出し EntryCall を行い、d 時間だけ待つ。その間にエントリ呼び出しが受け付けられたらランデブを行なう。もし、d 時間待ってもエントリ呼び出しが受け付けられなければエントリ呼び出しをとりやめる。この構文はただ単にランデブのタイムリミットを指定して、間に合わない時に代替の処理を行なうためのものと解釈することもできる。ここで 2 章で述べた実時間スケジューリングとの関連を考察しよう。本研究では、ある実時間処理の要求が時刻 S に発生した時、処理の期限 D を優

先度の値として用い、複数のタスクが処理期限を守れるように適切なスケジューリングを行うことにした。このような観点からAdaの時限エントリ呼び出しを見直すと、呼び出し側のタスクが受け側のタスクに対して処理期限を指定してランデブの要求をしていると考えることができる。つまり、dの情報を積極的に活用して適切なスケジューリングを行えば、より多くの時限エントリ呼び出しがランデブを行えるようになる。時刻Sに行なうエントリ呼び出しの期限がDの時、$d = D - S$として時限エントリ呼び出しを行なう。受け側のタスクは対応するaccept文にたどり着くまでの処理のデッドラインとしてDが設定され、優先度Dの順にスケジューリングされる。言い替えれば、時限エントリ呼び出しは受け側のタスクに優先度$D = S + d$を付けるという意味を持つことになる。

時限エントリ呼び出しでランデブが時間内に成功しなかった時に実行される処理はあくまでも代替処理である。できるだけランデブが行われ、本来の処理が行れるべきであると考えることもできる。そのためには適切なスケジューリングが行われる必要がある。本研究では、時限エントリ呼び出しの時限dの情報を積極的に活用して、全ての時限エントリ呼び出しが成功するように、適切なスケジューリングを行なう。このように実現される時限エントリ呼び出しを実時間エントリ呼び出しと呼ぶことにする。

3.3 Adaの優先度とエントリ待ち行列

前節で説明したように、我々の提案するタスクの優先度はAdaの本来の優先度とは異っている。現行のAdaの優先度はタスク毎に整数の固定値が与えられるため、時限エントリ呼び出しでサーバタスクに呼び出し毎に違う優先度を与えたり、優先度の継承を行うことができない。よって我々はAda本来の優先度は用いないことにする。また、一つのアクセプト文に対して同時に複数のエントリ呼び出しが行なわれた場合、タスクはエントリ待ち行列で待たされる。この時エントリ待ち行列に入る順番は現在のAdaの言語仕様では先着順となっている。本研究で定義した実時間処理を行うためには、現行のAdaの言語仕様のうち、エントリ待ち行列が優先度順待ち行列になるよう変更しなければならない。現在のように先着順の待ち行列では適切なスケジューリングを行うことができない。

4. 分散環境への実現法

本章では、前章で説明したAdaの実時間エントリ呼び出しと優先度の継承を分散環境に実現する方法を説明する。まず最初に分散環境について幾らかの仮定する条件を掲げ、それから実時間性を考慮しない単純な遠隔エントリ呼び出し、実時間遠隔エントリ呼び出しの順で説明する。

4.1 分散環境に対する仮定

分散環境におけるAdaプログラムの動作を定義する前に、まず、分散環境についていくつかの仮定をする。

i) 通信データは必ず到着する。

ii) 送信者と受信者が同一の場合通信データの順序は保存される。

iii) 通信時間の最悪値が既知である。

iv) 各ノードは時間間隔が十分に正確な時計を持っている。

v) 各ノードの時計は一致しているとは限らない。

時計に関する二つの仮定iv),v)は妥当であろう。通信に関する仮定i),ii),iii)は検討の余地がある。分散環境をこのように仮定すると、その上で実行されるAdaプログラムの振舞いは前節で示したものを素直に適用すれば良いことが分かる。では次に、仮定した分散環境でAdaプログラムはどのような振舞いをするかを示す。

4.2 遠隔エントリ呼び出し

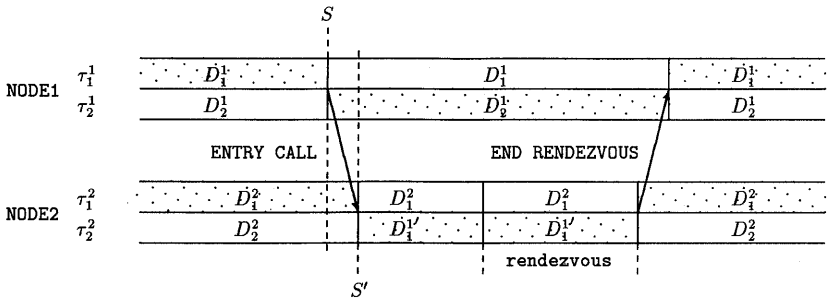

図2 遠隔エントリ呼び出し

図2は遠隔エントリ呼び出しが行なわれる様子である。時間は左から右に向かって経過している。本論文では以後、NODEi上のj番目のタスクをτ_j^iと表す。図ではNODE1に二つのタスクτ_1^1とτ_2^1が、NODE2にも二つのタスクτ_1^2とτ_1^2が存在する。影をつけた部分は、そのタスクが実行中であることを表す。D_1^1, D_2^1, D_1^2,D_2^2, $D_1^{1'}$はタスクの優先度を表している。それぞれの優先度の関係はこの例では$D_2^1 < D_1^1$かつ$D_2^2 < D_1^2$かつ$D_1^2 < D_1^{1'}$であるとする。時刻SにNODE1のタスクτ_1^1からNODE2のタスクτ_2^2に対してエントリ呼び出しが行なわれる。この時、通信時間の最悪値はT_Cであるとする。エントリ呼び出しの時、パラメタといっしょに$D_1^1 - S$の値も送られる。エントリ呼び出しがNODE2に到着した時刻S'にτ_2^2はまだ対応するアクセプト文に到達していなかったとする。この時τ_2^2は自分の優先度D_2^2と、NODE2の時計で求められたτ_1^1の優先度$D_1^{1'}$とを比較し、$D_1^{1'} < D_2^2$であることから$D_1^{1'}$を継承する。$D_1^{1'}$の値は次式で与えられる。

$$D_1^{1'} = S' + (D_1^1 - S) - 2 \times T_C$$

更に$D_1^{1'} < D_1^2$であれば、τ_2^2はτ_1^2を横取り(preempt)する。やがてτ_2^2は対応するアクセプト文にたどり着き、二つのタスクτ_1^1とτ_2^2の優先度(τ_1^1の優先度はNODE2では$D_1^{1'}$である。)のうち高い方の優先度$D_1^{1'}$でランデブを行なう。ランデブが終了すると、τ_2^2はそれをτ_1^1に知らせ、自分は元の優先度に戻る。NODE2の上ではτ_2^2にブロックされていたτ_1^2が実行を開始、NODE1においてはτ_1^1がブロックされていた間実行されていたτ_2^1をτ_1^1が横取りして実行を再開する。以上、普通の遠隔エントリ呼び出しと優先度の継承について説明した。次節では実時間遠隔エントリ呼び出しの説明を行なう。

4.3 実時間遠隔エントリ呼び出し

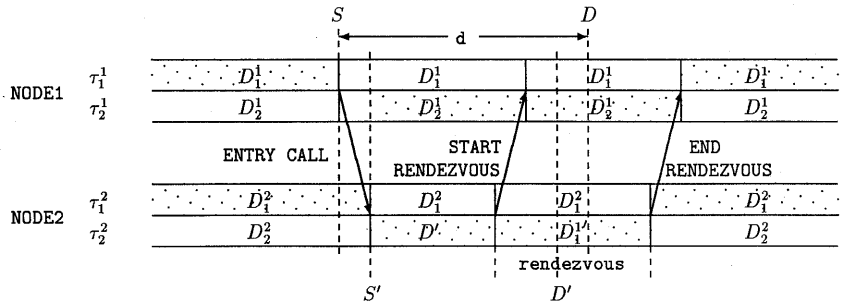

図3 (a) 実時間遠隔時限エントリ呼び出し(ランデブ成功)

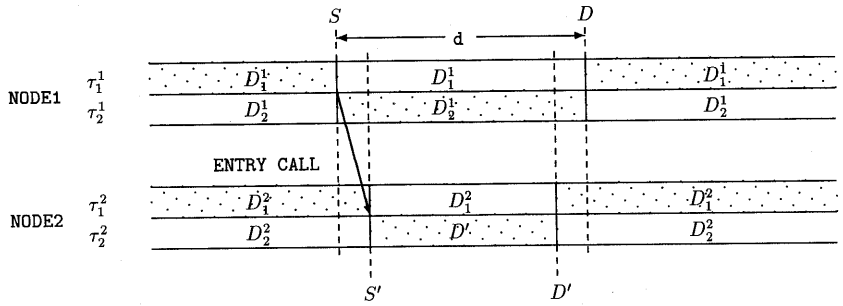

図3 (b) 実時間遠隔時限エントリ呼び出し（ランデブ失敗）

図3 (a),(b) はNODE1のタスク τ_1^1 がNODE 2のタスク τ_2^2 に対して実時間遠隔エントリ呼び出しを行う様子である。図3 (a) はランデブが成功した場合、図3 (b) は失敗した場合を表している。実時間エントリ呼び出しは新たな優先度の値を作り出す。タスクの優先度の関係は、$D_2^1 < D_1^1$ かつ $D_2^2 < D_1^2$ であるとする。時刻 S に τ_1^1 から τ_2^2 に対して実時間エントリ呼び出しが行われる。この時指定される優先度（時限ランデブの期限）は D である。また、この要求がNODE2に到着する時刻を S' とする。通信時間の最悪値を T_C とすると、遠隔時限エントリ呼び出しの通信データに含まれる $D-S$ の値から、NODE2の時計を用いて D' は次のように求められる。

$$D' = S' + (D - S) - 2 \times T_C$$

こうして求められた D' が $D_2^2 < D_1^2 < D'$ であるなら、タスク τ_2^2 の新たな優先度は D' となり、横取りスケジューリングが行われるのである。ただし、

$$D_1^{1\prime} = D_1^1 + (D_1^1 - S) - 2 \times T_C$$

で求められる $D_1^{1\prime}$ と D' との関係が $D' < D_1^{1\prime}$ である時、優先度の継承がおこるため、τ_2^2 の優先度は $D_1^{1\prime}$ となる。

τ_2^2 が時刻 D' までに対応するアクセプト文にたどり着いたなら τ_1^1 に対してそのことを通知し、ランデブを行う。ランデブは二つのタスク τ_1^1 と τ_2^2 の優先度（実際は $D_1^{1\prime}$ と D_2^2）のうち高い方の優先度で行なわれる。このことは、Ada本来のランデブの規則に合致するが、見方を変えれば τ_1^1 から τ_2^2 に対して優先度の継承が行なわれたと考えることもできる。そのため、実時間エントリ呼び出しの時には $D-S$ の他に $D_1^1 - S$ も送られる必要がある。ランデブが終了するとタスク τ_1^1 に知らされる。NODE1ではランデブの終了を知らされた τ_1^1 が τ_2^1 を横取りして実行を再開する。

タスク τ_2^2 が時刻 D' までに対応するアクセプト文にたどり着かなかった場合、τ_2^2 は τ_1^1 からのエントリ呼び出しをエントリ待ち行列から取り除いた後、元の優先度 D_2^2 に戻され、τ_1^2 に横取りされる。タスク τ_1^1 は時刻 D まで待っても返事が来ないため、エントリ呼び出しを中止して τ_2^1 を横取りし、実行を再開する。

前章で定義したAdaの実時間処理モデルを、分散環境上に実現する方法を示した。通信時間が処理時間に対して無視できない分散環境においても、前述の仮定を満たすならば実時間処理を行うことができる。

5. まとめと今後の課題

Adaの実時間処理や分散環境用のAdaの研究は以前から行なわれている。それらの中には非同期のエントリ呼び出しの提案などがある。[11] 分散環境では他のノードのタスクにエントリ呼び出しを行なう際の通信の遅れを無視できない。非同期のエントリ呼び出しを用いると、呼び出し側のタスクはエントリ呼び出しを行なった後、受け手からの返事を待っている必要がなく、応答性が高くなる。また、実時間処理では実行時システムの効率もばかにならない。Adaの実行時システムは複雑で効率が悪いため、効率の良いサブセットの実行時システムを用意し、制限された言語仕様に従って作られたプログラムはそれを利用できるようにするという研究もある。

本研究は実時間処理のモデルを調査することから始め、実時間スケジューリングに着目した。システムの処理効率や応答性も大切であるが、スケジューリングがうまく行われなければそれらは意味をなさないと考えたからである。次に、実時間処理のモデルに基づいて、時限エントリ呼び出しに新しい解釈を与え、実用的でAdaの言語仕様にあった実時間モデルを提案した。Adaの言語仕様には、優先度や先着順のエントリ待ち行列等そのままでは実時間スケジューリングの妨げになる部分が存在したため最小限度の変更を加えた。更にその実時間モデルを分散環境上に実現する方法を示した。今後、本研究で提案した実現法にそって分散環境のための処理系を開発し、それを用いて実際に実時間処理プログラムを開発し、本研究で提案したAdaの実時間モデルが本当に実用的なモデルと言えるのかどうか評価する予定である。

参考文献

[1] Borger, M., Klein, M., Weiderman,N. and Sha,L.:"A Testbed for Investigating Real-Time Ada Issues," ACM Ada LETTERS, Vol.8, No.7, Fall, 1988, pp.7-11.

[2] 平原, 岡村, 縄田, 荒木:"ランデブと共有変数を持つ並列型言語の実行支援系," 情報処理学会, 計算機アーキテクチャ研究会報告, Apr., 1990

[3] 岡村, 縄田, 平原, 荒木:"単一アドレス空間モデルに基づいた分散環境上での並列プログラミング言語処理系の実現," 信学技法, CPSY89-20, 1989, pp.33-38.

[4] Ford,R.:"Concurrent Algorithms for Real-Time Memory Management," IEEE Software, Vol.5, Sep., 1988, pp.10-23.

[5] Ramamritham,K., Stankovic,J.A.:"Dynamic Task Scheduling in Hard Real-Time Distributed Systems," IEEE Software, Vol.1, July, 1984, pp.65-75.

[6] Ramamritham,K., Stankovic,J.A. and Shiah,P.:"Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems," IEEE Transactions on parallel and distributed systems, Vol.1, No.2, Apr., 1990

[7] Liu, C.L.:"Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," Journal of the ACM, Vol.20, No.1, Jan., 1973, pp.46-61.

[8] Goodenough, J.B., Sha,L.:"The priority Ceiling Protocol:A Method for Minimizing the Blocking of High Priority Ada Tasks," ACM Ada LETTERS, Vol.8, No.7, Fall, 1988, pp.20-31.

[9] Sha, L., Goodenough, J.B.:"Real-Time Scheduling Theory and Ada," IEEE Computer, Vol.23, No.4, Apr., 1990, pp.53-62.

[10] "Reference Manual for the Ada Programming Language," ANSI/MIL-STD-1815A, United States Department of Defense, Jan., 1983 (情報処理振興事業協会編 "最新 Ada 基準文法書", bit 別冊, 共立出版, 1984)

[11] Elsom, K.:"Asynchronous Communication in Ada", ACM Ada LETTERS, Vol.10, No.4, Spring, 1990, pp.57-65.