解 説

Real-Time Mach：実時間マイクロカーネル†

徳 田 英 幸††

1. はじめに

Real-Time Mach 3.0は，カーネギーメロン大学で開発されたMach 3.0マイクロカーネル[1),4)]をベースにARTSカーネル[10),24),28)]で開発したリアルタイムスレッド，リアルタイムスケジューラ，同期操作，タイマオブジェクト，クロックオブジェクト，リアルタイムプロトコルモジュールなどを取り入れてリアルタイム処理用に拡張したマイクロカーネルである．従来のリアルタイムエグゼクティブと違い，解析でき，かつ予測可能な分散リアルタイムOSを提供することを目標としている[25)~27)]．また，1992年から慶應義塾大学環境情報学部にてスタートしたマルチメディア統合環境プロジェクト（KEIO-MMP Project）においても，分散マルチメディア環境を支える共通基盤ソフトウェアのためのカーネルとして，Real-Time Machにマルチメディア対応のための機能拡張が行われてきている[20),21),29)]．

本稿では，Real-Time Mach 3.0のマイクロカーネルアーキテクチャと基本的なリアルタイム機能について述べる．また，Real-Time Mach 3.0を組込みシステムや分散マルチメディアシステムに利用するために改良・拡張した機能について概説する．

2. Real-Time Machの概要

Real-Time Mach 3.0の機能の開発に関しては，従来からのMachマイクロカーネルが提供している機能を損ねることなく独立した拡張機能として設計，実装している．ここでは，リアルタイムマイクロカーネルReal-Time Mach 3.0[25),26)]のリアルタイム機能の概要を述べる．

2.1 マイクロカーネルアーキテクチャ

Real-Time Mach 3.0では，Mach 3.0と同じUNIXサーバ[4)]，MS-DOSサーバやX-windowシステムを動作させることができる．つまり，リアルタイム用のプログラムとともに，従来のUNIX用に作成されたプログラムやDOSのプログラムも同時に混在して動作させることができるマルチパーソナリティな実行環境を提供している．また，これからのPDAやモービルホストを利用したユビキタスコンピューティング環境などを実現するために，Real-Time Server（RTS）やNetwork Protocol Server（NPS）[13)]といったリアルタイム用のサーバ群を提供することでUNIXサーバを介さず，分散リアルタイムアプリケーションを直接マイクロカーネル上で動作できるように構成されている．

このようにマルチパーソナリティをもった実行環境は図-1のように表すことができる．

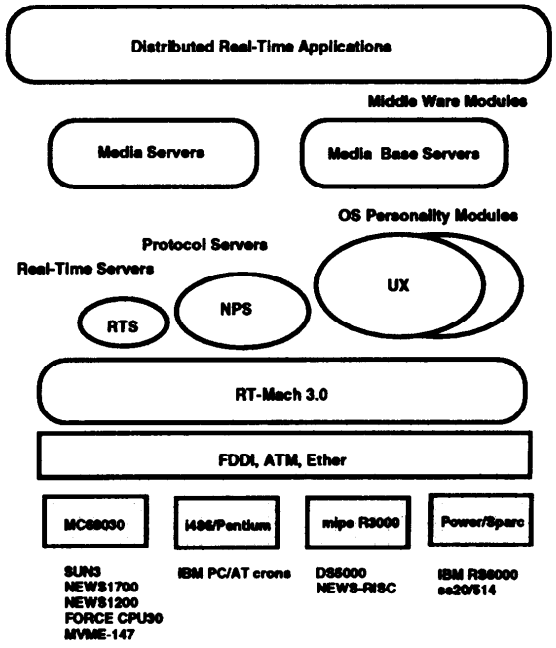

図-1 Multipersonality Environments for Real-Time Mach

† Real-Time Mach: A Real-Time Microkernel by Hideyuki TOKUDA (Faculty of Environmental Information, Keio Univ. / School of Computer Science, Carnegie Mellon Univ.).

†† 慶應義塾大学環境情報学部／カーネギーメロン大学計算機科学科

2.2 リアルタイムスレッド

Real-Time Mach では，ARTS カーネルで開発したリアルタイムスレッドの概念[24]と Mach の“C-Thread”パッケージ[3]を融合させた RT-Thread モデルが提供されている．Real-Time Mach では，次のようなリアルタイムスレッドを制御するシステムプリミティブがマイクロカーネルに実装されている．

カーネルプリミティブ

rt_thread_create(parent_task, thread, thread_attr)
rt_thread_exit(thread)
thread_get_attribute(thread, flavor, old_attr, old_attrCnt)
thread_set_attribute(thread, flavor, new_attr, new_attrCnt)

ライブラリ

rt_thread_attributed_init(prioriy, period_secs, period_nsecs, entry, arg, port, thread_attr)
rt_thread_deadline_handler(thread,thread_attr, entry, arg)

RT-Thread のインタフェースは，基本的には，C-Thread の拡張であるが，リアルタイムタスクにおいて，より正確に周期的スレッドを記述でき，かつデッドラインなどの時間的制約を明示できることが特徴である*．

以下の簡単化されたプログラム例では，関数 f( ) (9 行目) がリアルタイムスレッドとして，時間的な属性 f,Si,Ti,Di をもって生成される例を示してしている．ここで，f は，そのスレッドの関数本体 f( ),Si,Ti,Di は，スレッド f の開始時間，周期，デッドライン情報を表している．

```
1. root( )
2. {
3.   thread_id f_id;
4.
5.    f_attr={f,Si,Ti,Di};
        /* set thread attribute of f */
6.    thread_create(&f_id,f_attr);
        /* creating f( ) as a thread */
7. }
8.
9. f(arg){
10.    f's body
11. }
```

* 従来の方法を，プログラムと時間的制約の implicit binding といい，明示的な方法は，explicit binding という．

一般に，リアルタイムスレッドは，周期的なスレッドと非周期的なスレッドに分類される．周期的なリアルタイムスレッドの場合には，関数 f が終了した場合，自動的にその時間属性にしたがって再起動される．また，これらの時間属性を実行時に動的に変更することができる．

2.3 デッドライン制御

通常のマルチスレッドプログラムにおいて，スレッドが存在しないメモリを参照したり，0 による除算をした場合には，例外処理が発生しオペレーティングシステムが対処している．リアルタイムスレッドの場合は，このような論理的エラーだけではなく，時間的エラーにより，例外処理が発生する．すなわち，時間的エラーとは，リアルタイムスレッドがその属性としてもっている時間的制約（たとえば，デッドライン）を満足できなかった時に発生する場合である．この場合システムは，制御の流れをユーザが指定した例外処理スレッドへとスイッチする．

以下の簡単化されたプログラム例では，関数 rf( ) (13 行目) が関数 f( ) (9 行目) の例外処理スレッドとして定義されている．

```
1. root( )
2. {
3.    f_attr={f,Si,Ti,Di};
        /* set thread attribute of f */
4.   rf_attr={rf,Si,Ti,Di};
        /* set thread attribute of rf */
5.    thread_create(rf_id,rf_attr);
        /* creating rf thread */
6.    thread_create(f_id,f_attr);
        /* creating f thread */
7. }
8.
9. f(arg){
10.    f's body
11. }
12.
13. rf(time,thread,message)
14. {
15.    wait_for_notification( );
        /* waiting for a deadlinemiss
         notice */
16.    rf's body
17. }
```

一般に，*within (t) do s except q* といった構造

をもったリアルタイム処理用言語RTC++[5]では，時間的制約tを満たせない場合は，例外処理手続きqが起動される．しかし，この例外処理において，スレッド本体sの優先順序より，優先順序を高くして実行したいといった要求が多く発生する．そこで，この例に示すように，例外処理手続きrf本体も，独立したリアルタイムスレッドとして定義している．これにより，メインスレッドであるfとは異なった優先順序での処理が行われる．また，例外処理スレッドrfでは，アプリケーションのセマンティクスにあった形で，時間的エラーに対処する．たとえば，メインスレッドを異常終了させたり，バックワードやフォワードなリカバリ機能を提供し，プログラムを正常な状態に戻すことが可能である．

2.4 クロック・オブジェクトとタイマ・オブジェクト

Machでは，一般にシステムから提供されている1つのクロックしか提供されていない．しかし，より精度の高い時間に基づく制御を行うために，Real-Time Machでは，複数のクロックをサポートでき，またそのクロックの精度に基づいたタイマ・オブジェクトを提供している[22]．

クロックは，システムが備えるハードウェア・クロックに関連づけられたオブジェクトである．1つのクロックに対して，複数のタイマを定義できる．タイマとは，ある指示された時刻にアクションを起こすカーネル・オブジェクトである．いずれも，ユーザからはポートを介してIPCでアクセスされる．クロックの作成，設定はdevice操作プリミティブを使い，タイマの操作は以下のプリミティブが用意されている．

```
timer_create(parent_task, new_timer, clock_device)
timer_terminate(timer)
timer_arm(timer, expire_time, interval_time,
        expire_port, flags)
timer_cancel(timer, flags)
timer_sleep(timer, expire_time, flags)
timer_info(timer, lavor, timer_info, timer_infoCnt)
task_timers(task, timer_list, timer_count)
```

2.5 スケジューラ

一般に，リアルタイムOSで採用されているスケジューリング方式は，そのシステムが対象としているリアルタイムタスクの性質，すなわち，静的／動的なタスクセット，周期／非周期的なタスクなどの状況によって大きく左右される．従来からの組込み型システムでは，サイクリック・エグゼクティブモデル（Cyclic Executive Model）[2]が多く使われているが，タスクセットが複雑になったり，タスクの変更が頻繁に起こる場合は，割当て作業を手動で行い，再構成をしなければならないなど多くの問題を含んでいた．

Real-Time Machでは，レート・モノトニック方式や最短デッドライン優先方式などリアルタイムタスクを考慮しつつ，従来のTSS型のスケジューリングポリシィもサポートできるといったスケジューリングポリシィ選択可能なスケジューラが実現されている．たとえば，レート・モノトニック方式は，周期の一番短いタスクに最高の優先順序を与える横取り可能なスケジューリング方式であり，従来のモデルと違いスケジューラビリティの解析システムなどとの整合性が非常に良いのが特徴である[25]．とくに，周期的データストリームを扱う周期的なアクティビティの処理において有効な方式である．現在Real-Time Machでサポートされているスケジューリング・ポリシィは次のとおりである．

Mach Timesharing：Machの時分割スケジューリング

Fixed Priority/RR：固定プライオリティ／ラウンドロビン方式スケジューリング

Fixed Priority/FIFO：固定プライオリティ/FIFO方式スケジューリング

Rate Monotonic：周期の一番短いスレッドに最高の優先順序を与える先取り可能なスケジューリング

Deadline Monotonic：デッドラインの短いスレッドに最高の優先順序を与える先取り可能なスケジューリング

Earliest Deadline First：デッドラインに最も近いスレッドから順に実行権を与える先取り可能なスケジューリング

また，これらのポリシィは，各プロセッサセット（i.e.，プロセッサとスレッドの集合）ごとの属性として設定されており，processor_set_get_attribute( )とprocessor_set_set_attribute( )のプ

リミティブを使って変更することが可能である．

2.6 リアルタイム同期プロトコル

Machでは，タスク内のすべてのスレッドはタスクの資源を共有するため，スレッド間での同期機構が必要であり一般的なモニタ型同期機構を提供している．モニタ型同期機構では，同期待ちにあるタスクすべてを平等に扱い，飢餓状態（starvation）にならないことを保証するためFIFO順にキューイングしている．しかし，この方式では，FIFO順にクリティカル・セクションに入るので，先に，ロックをかけた優先度の低いタスクによって優先度の高いタスクの実行が遅れさせられてまうといったプライオリティ・インバージョンが起きてしまう．Real-Time Machでは，このような状況を回避するために，プライオリティ継承プロトコル[23]などをmutex_variableの生成時に，mutex_variableの属性として指定できるように拡張してある．現在拡張されているモニタ関連のプリミティブ[26]は次のとおりである．

```
rt_mutex_allocate(&mutex, mutex_attr)
rt_mutex_deallocate(mutex)
rt_mutex_lock(mutex, timeout, context)
rt_mutex_unlock(mutex)
rt_mutex_control(mutex, cmd, arg, size)
```

また，実現されている同期プロトコルとしては，次の5つのポリシィがある．

Kernelized Monitor（KM）：クリティカルセクションを実行中のスレッドは，先取りされない．

Basic Priority（BP）：スレッドがクリティカルセクションを実行中でも，先取りを許す．同期待ちキューはスレッドの優先度順である．

Basic Priority Inheritance Protocol（BPI）：あるスレッドがクリティカルセクションを実行中に，そのスレッドより高いプライオリティをもつスレッドが同期待ちになった場合，すでに実行中のスレッドは待ちにあるスレッドの中の最高のプライオリティを継承する．

Priority Ceilling Protocol（PCP）：BPIを拡張したプロトコルである．各ロックに対して，そのロックを獲得する可能性のある全スレッド中の最高プライオリティをそのロックに対するプライオリティの上限値とする．

Restartable Critical Section（RCS）：クリティカルセクションを実行中のスレッドよりプライオリティの高いスレッドが同期待ちになった場合，現在実行中のプライオリティの低いスレッドの処理を中止させ，クリティカル資源の状態をもとに戻し，待ち行列に戻す．

2.7 リアルタイムIPC

MachのIPCは，すべてカーネル内のポートを通して行われる．送信元からのメッセージはポートにFIFO順にキューイングされ，送信先のタスクへ渡される．

Real-Time Machでは，ポートのメッセージキュー待ちにおいて発生するプライオリティ・インバージョンの問題を回避するために，ポートの属性を拡張したRT_IPCを実現している．ポート属性でプライオリティ制御のポリシィ，キューイングのポリシィ，バッファの数や要求ブロックの大きさを指定できる．現在，ポートに対して以下の属性を指定することができる．

Message Queueing：メッセージキューのキューイングに関し，FIFO順か優先度順かを指定する．

Priority Hand-off：受信側スレッドの優先度を指定する．Hand-offが選択されると，受信側スレッドの優先度は，送信側の優先度にセットまたは，指定された優先度にセットできる．

Priority Inheritance：プライオリティ継承プロトコルを起動するかしないかを指定する．

Message Distribution：サーバ側が複数の受信スレッドをもっている場合に，どのスレッドにメッセージを渡すかの選択を優先度で行うか，そうでないかを指定する．

現在，Real-Time MachがRT_IPCプリミティブは以下のとおりである．

```
rt_mach_port_allocate(space, right, port_attr, namep)
rt_mach_port_allocate_name(space, right, port_attr, name)
rt_mach_port_associate(thread, name, priop)
rt_mach_port_not_associate(thread)
rt_mach_msg(msg, option, send_size, rcv_size,
        rcv_name, time_out, notify)
```

2.8 メモリ・オブジェクト管理

Machでは，実際にスレッドがメモリ・オブジェクトにアクセスし，ページフォルトが起こるまでは物理メモリを確保しない遅延評価（lazzy evaluation）方式を採用している．しかし，これではページフォルトが起きた時のOS内での処理時間が予測できないのでリアルタイム処理用の記憶管理には適していない．そこで，RT-Thread内では，親タスクの仮想アドレス空間を以下のような vm_wire プリミティブを用いて，実記憶空間へ固定する機能を提供している．

vm_wire(host_priv, task, address, size, access)

3. Real-Time Machの改良・拡張

Real-Time Machの開発に関しては，マイクロカーネルの機能に関する改良・拡張だけでなく，その上のミドルウェアに関しても開発が続いている．とくに，カーネギーメロン大学のARTグループ，慶應義塾大学のKEIO-MMPプロジェクトグループ，北陸先端大学院大学のグループで活発に研究開発されている．ここでは，いくつかの新しい改良・拡張された機能について述べる．

3.1 RTC-Threads

従来のReal-Time Machで提供されているリアルタイムスレッド（RT-Thread）は，非常に柔軟で，周期スレッドのスケジューリングやタイミングエラーに対する処理を容易にしている反面，すべてがカーネルスレッドとして実現されているため，コンテクストスイッチのオーバヘッドなどにより，十分な性能を出しきれていなかった．そこで，C-Threadと同様に，ユーザレベルのRTC-Threadパッケージを開発した[17),18)]．RTC-Threadは，C-Threadの提供するインタフェースと上位互換性をもち，リアルタイム処理のためのインタフェースが追加されている．カーネルに対する変更は，約1100行のソースコードの追加を行った．そのうち約300行は，スレッドおよびタイマ管理ルーチンをユーザレベルスケジューラへのアップコールに対応させるための，既存のソースコードに対する追加変更分である．残りの約800行は，ユーザレベルリアルタイムスレッドをサポートする新しいプリミティブのためのものである．

表-1 に，66MHzのIntel 80486DX2を使用したIBM PC-ATコンパチブルマシンGateway2000 486DX2-66V上で実測した諸性能値を示す．性能測定には，1μsecの精度をもつSTAT!タイマボードを使用した．

表-1 各スレッドの性能比較

	RTC-Threads	RT Threads
thread create	390 μ sec	1571 μ sec
timer create	12 μ sec	318 μ sec
thread resume by timer	193 μ sec	126 μ sec
invoke deadline handler	188 μ sec	233 μ sec
get thread attribute	7 μ sec	178 μ sec
set thread attribute	5 μ sec	82 μ sec
thread resume	14 μ sec	132 μ sec

表-1から明らかなように，ユーザレベルスレッド化を実現したことにより，ほとんどの機能を高速化することが可能となった．

3.2 RTS/NPS Environment

RTC-Threadパッケージと同様に，ネットワークプロトコルサーバ（NPS）をよりアプリケーションに適した実行環境を実現するために，RTS/NPSの環境を拡張した[13),14)]．

とくに，ユーザレベルでのNPSの性能の向上をめざし，NPSとアプリケーションとの通信をIPC経由ではなく共有変数を使った方式，さらにNPS内をC-Thread化した方式，また，NPSをユーザライブラリ化した方式をそれぞれ実装し，その性能を従来のUNIXサーバ方式と比較評価した．

図-2 は，それぞれの方式において，ユーザプロセスがリモートホスト上のプロセスにデータを送った場合の性能比較をラウンドトリップタイムを計測して求めたものである．

図-2において，Mach3.0/UXは，通常のUNIXサーバを利用した方式Mach 3.0/Library Socketは，UNIXのソケットをライブラリ化した方式[9)]を示す．また，NPS(Share)は，NPSとアプリケーションとの通信をIPC経由ではなく共有変数を使った方式，NPS（Copy）は，通常のIPCによる方式，NPS/Cthread（Share）は，NPS内をC-Thread化した方式，NPS（Library）は，NPSをユーザライブラリ化した方式をそれぞれ示している．

NPS（Copy）とNPS（Share）を比較すること

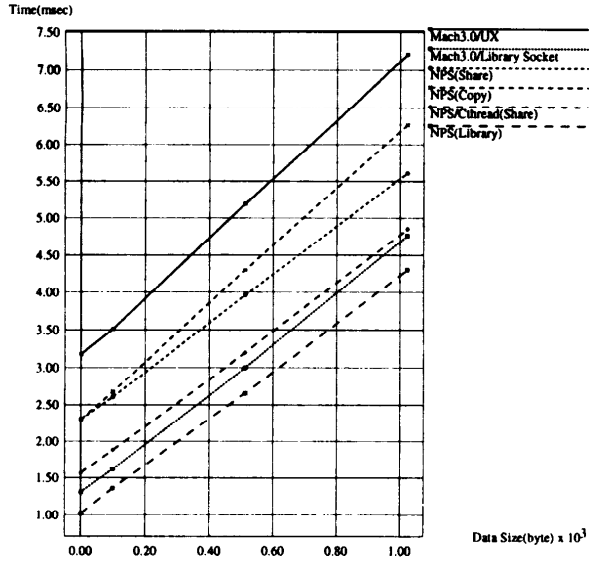

図-2 Comparison of NPS and Mach/Unix server

により，従来の Copy（IPC）方式では，ユーザ空間からデータを NPS へ転送するためのオーバヘッドがあり，共有バッファを使ってこれらを削除し，かなりの性能向上があったことが確認できた．NPS をユーザライブラリ化した方式が，もっとも性能が向上しているが，ユーザプログラムと同一空間上で実装されているので，信頼性に問題が残る．また，UNIX サーバの場合も同様，このような方式の有効性が示されている．

3.3 分散マルチメディアシステム

KEIO-MMP プロジェクトでは，Real-Time Mach を改良・拡張するとともに，さまざまな分散マルチメディアプリケーションを容易に構築できるサーバ群をミドルウェアとして研究開発している．マルチメディアシステムを支援するための基本機能としては，リアルタイム性の優れたマイクロカーネルの機能を提供し，かつ従来の TSS 型オペレーティングシステムがもつインターオペラビリティや相互接続性を向上しなければならない．

KEIO-MMP プロジェクトでは，以下のような機能に対する改良，拡張を行っている．

- Processor Reserves：スレッドごとにプロセッサ資源をどのくらい割り当てるかを制御する[11),12)]．この機能により，マルチメディアアプリケーションが複数同時に実行された場合でも，ほかのプログラムからの干渉を受けずに安定して実行することが可能となる．また，サーバへの Reserve の譲渡や，複数スレッド間での Reserve の分割・統合なども可能である．
- Dynamic QOS Control Schemes：サービスの質（QOS）に関する保証とその動的な制御は，マイクロカーネル内の改良だけでなく，マルチメディアアプリケーションと協調しながら，システムの一時的な過負荷な状況に対処していこうとするアプローチである．たとえば，ビデオホーンシステムにおいて，ビデオセッションを起動する際にあらかじめ，ビデオや音声データのサービスの質の上限，下限を指定し，もし，新たに起動されたセッションが十分なプロセッササイクルやネットワーク資源を得られない場合は，ほかのアクティブでないセッションの QOS を低解像度に動的に変化させ，新しいセッションに資源を提供しようという方式である[6),7),28),29)]．
- Conductor Performer Model：連続メディアデータの処理を実時間性を保証しつつ効率よく行うために，Conductor/Performer モデルを開発している[15),16)]．Real-Time Mach 上にミドルウェアとして実装をし，性能評価を行っている．Conductor は，周期／非周期スレッドを利用して，特定のメディアストリームを処理しているそれぞれの Performer の制御を行っている．
- Multicast Protocols for Continuous Media：分散メディアシステムにおいて，メディアストリームをすべて同一の QOS レベルで転送するのではなく，受信側のネットワーク容量やホストの処理能力に合った QOS レベルに動的に適応させ，かつマルチキャストを行って全体の転送容量を最小に抑える必要がある．現在までに，ST-II プロトコル[30)]などが実際に利用されているが，きめ細かな QOS 制御などが行われていない．ここでは，各ホスト内の資源，中継ノード，ネットワーク処理などを通して統一された QOS の制御を行う必要がある[8),19)]．

4. おわりに

Real-Time Mach は，CMU の ART グループで筆者らが開発したリアルタイム処理技術を Mach 3.0 マイクロカーネルに移植し，マイクロ

カーネル自体のもっている本来の拡張可能性や柔軟性を融合し，新しいリアルタイムアプリケーションに利用することを目的としていた．マイクロカーネルそのものをリアルタイム拡張することにより，非常に幅広いアプリケーションドメインに適応可能となっている．ワークステーションやPCを中心とした分散リアルタイムアプリケーションだけでなく，今後は，モーバイルユニットやNetwork Computerを利用したシステムの重要性も増しており，アプリケーションに最適化された基盤ソフトウェアが必要となってきている．

現在も，Real-Time Machは，CMUおよび慶應義塾大学［http://www.mmp.sfc.keio.ac.jp］から公開ソフトウェアとして配布されている．また，分散マルチディアシステムのためのミドルウェアも慶應義塾大学から公開されている．本稿が，新しいマイクロカーネルやオペレーティングシステムを作っていかれる方々の参考になれば幸いである．

謝辞 Real-Time Machの研究開発は，ARPA，NOSC，IPAをはじめいろいろな研究機関の協力や支援により成り立っている．とくに，カーネギーメロン大学のARTグループ，慶應義塾大学のKEIO-MMPプロジェクトと徳田・村井研究室，北陸先端大学院大学の中島研究室の各メンバに感謝いたします．また，**3.1**，**3.2**節のデータは，追川，中島両氏の実装による．

参考文献

1) Accetta, M. J., Baron, W., Bolosky, R. V., Golub, D. B., Rashid, R. F., Tavanian, A. and Young, M. W.: Mach: A New Kernel Foundation for Unix Development, Proc. of USENIX Summer Conference(1986).
2) Baker, T. P. and Shaw, A.: The Cyclic Executive Model and Ada, Proceedings of the 9th IEEE Real-Time Systems Symposium, pp.120-129(Dec. 1988).
3) Cooper, E. C and Draves, R. P.: C Threads, Technical Report, Computer Science Department, Carnegie Mellon University, CMU-CS-88-154(Mar. 1987).
4) Golub, D., Dean, R., Forin, A. and Rashid, R.: Unix as an Application Program, Proc. USENIX Summer Conference(1990).
5) Ishikawa, Y., Tokuda, H. and Mercer, C. W.: Object-Oriented Real-Time Language Design: Constructs for Timing Constraints, Proceedings of Object-Oriented Programming System, Languages, and Applications, pp.289-298(1990).
6) Kawachiya, K., Ogata, M., Nishio, N. and Tokuda, H.: Evaluation of QOS-Control Servers on Real-Time Mach, Proc. 5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video(May 1995).
7) Kawachiya, K. and Tokuda, H.: Dynamic QOS Control Based on the QOS-Ticket Model, Proc. 3rd IEEE International Conference on Multimedia Computing and Systems(1996).
8) Kihara, S., Onoe, Y., Mitsuzawa, A., Moriai, S., Nambu, A. and Tokuda, H.: An Implementation of ST-II Protocol as a User-Level Server on Real-Time Mach, Collected Abstracts from the 4th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video(Nov. 1993).
9) Maeda, C., Bershad, B. N.: Protocol Service Decomposition for High Performance Networking, Proc. of 14th ACM Symp. on Operating Systems(Nov. 1993).
10) Mercer, C. W. and Tokuda, H.: The ARTS Real-Time Object Model, Proc. of 11th IEEE Real-Time Systems Symposium(Dec. 1990).
11) Mercer, C. W., Savage, S. and Tokuda, H.: Processor Capacity Reserves: Operating System Support for Multimedia Applications, Proc. Intl. Conf. on Multimedia Computing and Systems(May 1994).
12) Mercer, C. W., and Rajkumar, R.: An Interactive Interface and RT-Mach Support for Monitoring and Controlling Resource Management, Proc. of IEEE Real-Time Technology and Applications Symposium(May 1995).
13) Nakajima, T., Kitayama, T. and Tokuda, H.: Experiments with Real-Time Servers in Real-Time Mach, Proc. of 3rd USENIX Mach Symposium(Apr. 1993).
14) Nakajima, T.: NPS: User-Level Real-Time Network Engine on Real-Time Mach, Proc. of First International Workshop on Real-Time Computing Systems and Applications(1994).
15) Nishio, N., Tada, S. and Tokuda, H.: Conductor-Performer: A Middle Ware Architecture for Continuous Media Applications, Proc. of First International Workshop on Real-Time Computing Systems and Applications(1994).
16) 西尾，徳田：連続メディア処理のためのミドルウェアの性能評価，情報処理学会システムソフトウェアとオペレーティングシステム研究報告No. 69-8, pp.55-60(1995).
17) Oikawa, S. and Tokuda, H.: User-Level Real-Time Threads: An Approach Towards High Performancee Multimedia Threads, Proc. of the 4th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video, pp.61-71(1993).
18) Oikawa, S. and Tokuda, H.: User-Level Real-

Time Threads, Proc. of 11th IEEE Workshop on Real-Time Operation Systems and Software(1994).
19) Onoe, Y., Fujii, K. and Tokuda, H.: QOS-based Multicast Communication, Proc. of The 2nd International Workshop of Advanced Teleservices and High-Speed Communication Architectures (1994).
20) 緒方, 人見, 和田, 追川, 徳田: Real-Time Mach 3.0 の評価と改良, Proc. of RTP '93, 情報処理学会／電子情報通信学会(Mar. 1993).
21) 斉藤, 徳田, 萩野, 他: マルチメディア統合環境のテストベッドとその評価, Proc. of RTP '93, 情報処理学会／電子情報通信学会(Mar. 1993).
22) Savage, S. and Tokuda, H.: RT-Mach Timers: Exporting Time to the User, Proc. of 3rd USENIX Mach Symposium(Apr. 1993).
23) Sha, L., Rajkumar, R. and Lehoczky, J.P.: Priority Inheritance Protocols: An Approach to Real-Time Synchronization, IEEE Trans. on Computers(1987).
24) Tokuda, H. and Mercer, C.W.: ARTS Kernel:A Distributed Real-Time Kernel, ACM Operating Systems Review, 23(3) (July 1989).
25) Tokuda, H., Nakajima, T. and Rao, P.: Real-Time Mach: Toward a Predictable Real-Time System, Proceedings of USENIX Mach Workshop(Oct. 1990).
26) Tokuda, H. and Nakajima, T.:Evaluation of Real-Time Synchronization in Real-Time Mach, Procedings of USENIX Mach Symposium, pp.213-221(1991).
27) 徳田英幸: 分散リアルタイム OS の技術動向, コンピュータソフトウェア, Vol.9, No.3, pp.4-14(1992).
28) Tokuda, H., Tobe, Y., Chou, S. T. C. and Moura, J. M. F.: Continuous Media Communication with Dynamic QOS Control Using ARTS with an FDDI Network, Proceedings of ACM SIGCOMM '92(Sep. 1992).
29) 徳田英幸, 斉藤信男: マルチメディア統合環境プロジェクトにおけるリアルタイム処理技術, Proc. of RTP '93, 情報処理学会／電子情報通信学会(Mar. 1993).Tokuda, H. and Kitayama, T.: Dynamic QOS Control based on Real-Time Threads, Proc. 4th Intl. Workshop on Network and Operating System Support for Digital Audio and Video(Nov. 1993).
30) Topolcic, C. et al.: Experimental Internet Stream Protocol, Version 2(ST-II), CIP Working Group, Request for Comments: 1190(Oct.1990).

（平成7年9月11日受付）

徳田 英幸（正会員）

1975年慶應義塾大学工学部管理工学科卒業，1977年同大学院修士課程修了．1983年カナダ国ウォータールー大学計算機科学科博士課程修了，Ph.D. in Computer Science．同年より米国カーネギーメロン大学計算機科学科に勤め，1991年 Senior Research Computer Scientist，1994年 Adjunct Associate Professor．1990年9月より，慶應義塾大学環境情報学部助教授を兼任，1996年同学部教授，現在に至る．この間，分散システム，分散リアルタイムオペレーティングシステム，分散マルチメディアシステムなどの研究に従事．1977年学術振興奨励賞を電子通信学会，1983年第16回 Hawaii International Conf. on System Sciences にて Best Paper Award，1989年 Motorola Foundation Award を受賞．現在，本会システムソフトウェアとオペレーティング・システム研究会主査，日本ソフトウェア科学会理事，ACM，IEEE 各会員．