システムソフトウェアと
オペレーティング・システム 69−11
（１９９５．６．２３）

QOS 管理のための外部資源管理機構について[1]

盛合 敏　木原 誠司　南部 明　徳田 英幸
NTT 情報通信研究所　慶應義塾大学

本報告では，動的な QOS 制御を行うために必要な資源管理機構について考察する．ここでは，QOS の保証という観点から，資源を空間方向と時間方向に分割した資源スロットという単位によって保護，管理することを提案する．そして，RT-Mach 上で QOS に基づいた資源管理のためのマネージャをユーザレベルのサーバとして実現するためのカーネル機構と資源マネージャのインターフェースについて述べる．また，このインターフェースはユーザレベル・デバイスドライバの一般的なインターフェースとしても利用できる．

External Resource Management Facilities for QOS Control

Satoshi Moriai　Seiji Kihara　Akira Nambu
{moriai,kihara,nambu}@isl.ntt.jp
NTT Information and Communication Systems Laboratories

Hideyuki Tokuda
hxt@sfc.wide.ad.jp
Keio University

In this paper, we describe external resource management facilities for dynamic quality-of-service control. A resource in a system is divided into resource slots by spatial and temporal axes and a resource management system provides spatial protection and temporal protection. We also describe the kernel facilities to implement a resource manager for QOS control as a user-level server and the interface of a kernel and a resource manager on Real-Time Mach. This interface can be used for the generic interface to implement a device driver as a user-level server.

[1]この研究の一部は，情報処理振興事業協会 (IPA) が実施している開放型基盤ソフトウェア研究開発評価事業「マルチメディア統合環境基盤ソフトウェア」プロジェクトの研究協力として行われたものです．

1 はじめに

音声や動画といったアナログの連続メディアをデジタル化して扱う場合，時間軸上の制約とサービス品質 (QOS: quality of service) の制約という2種類の条件を守りながら処理を行う必要がある．このためには，連続メディアのセッション毎にシステムの資源が保証されていなければならない．しかし，有限なシステム資源の下でその負荷が動的に変動する状況においては，資源の保証のみでは十分ではなく，各セッションの性質や重要度と資源の利用状況を考慮してQOSの調整を行い，それに基づいた資源の管理が必要となる．このようなQOS制御と資源管理を行うためにはアプリケーションレベルの対応だけではなく，オペレーティングシステムでのサポートも必要である．これまでのオペレーティングシステムにおける資源管理は『公平性(fair share)』に重点が置かれており，資源の保証や利用状況の監視のための機構が欠けているため，連続メディアのサポートのためには，新たな資源管理機構が求められる．

著者らは，連続メディア向けの基盤ソフトウェアの研究として行われているKeio-MMPプロジェクト[12, 13]において，RT-Mach[14]へ各種の連続メディア処理向けの改良や拡張を加えている[9, 7, 10]．本報告では，連続メディアのQOS制御のためのオペレーティングシステムサポートのなかで，QOS制御を目的とした資源管理の機構について述べる．

2 連続メディアのための資源管理

2.1 オペレーティングシステムの機能

オペレーティングシステムカーネルの基本的機能は，計算機システム内のハードウェア資源を複数のプログラムに安全に共有させること (secure resource sharing) にある[4]．カーネルは，様々なハードウェア上で様々なプログラムを効率良く，そして，要求されたサービスの品質を満たすように実行させなければならない．例えば，テキストの編集や整形などのプログラム，電子メイルやファイルを転送するプログラム，ユーザインターフェースを提供するプログラム，そして，連続メディアを扱うプログラムは，それぞれのデータの最適な管理のポリシや資源が不足したときのdegradationのポリシも異なり，それらを支えるハードウェアに対する要求も異なっている．

このような様々な要求に応えるためには，カーネルはメカニズムのみを提供し，固定的なポリシを提供すべきではない．例えばアプリケーションレベルで実現されたスレッドパッケージや仮想メモリシステム[3]の存在は，カーネルが特定のモデルを提供したとしても不十分であり，結局はアプリケーションが必要とするポリシはアプリケーションで実現せざるを得ないことの傍証となっている．すなわち，アプリケーションが望む管理ポリシはアプリケーションが提供すべきであろう．これまでのカーネルが提供している機能は，管理ポリシが固定的であり，それをアプリケーションから制御することは難しく，そして，QOSの枠組みがないという欠点があり，マルチメディアアプリケーションを構築するためには不十分である．動的なQOS制御を行うためには，資源予約機能，資源利用の監視機能，および，資源利用情報の提供機能が必要となる[5]．

表 1. カーネルが管理する資源の例.

資源	管理単位
CPU	タイムスライス (周期 + タイムスライス幅)
TLB	TLB エントリ
Cache	キャッシュエントリ
メモリ	物理ページ
DMA	タイムスライス (周期 + タイムスライス幅)
ディスク	ディスクブロック + 帯域幅 + 遅延時間
ネットワーク	ポート + 帯域幅 + 遅延時間
トラップ	vector table entry

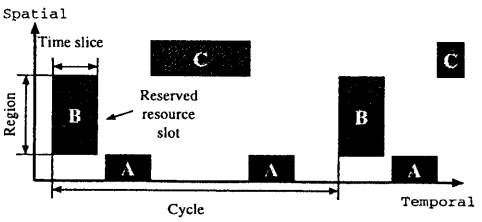

図 1. 資源の管理単位.

2.2 QOS制御のための資源管理機能

これまでのカーネルはアプリケーションプログラムに対して抽象レベルの高い資源を提供してきたが，そのような資源には管理ポリシが含まれてしまうため，カーネルが本来提供すべき機能ではない．そこで，よりハードウェアレベルに近い資源をカーネルが提供することを考える．種々のマイクロカーネル[1, 11, 2]はこのような資源を提供するものである．しかし，QOSの保証という観点では，資源を空間方向に分割した管理や保護 (spatial protection) だけではなく，時間方向に分割した管理や保護 (temporal protection) を行う必要がある[8]．

従来のカーネルでは，表1.に示すような資源に対してfare-shareな資源割り当てポリシによって資源を

割り当てる．例えば，CPU資源についてはタイムスライスの幅を一定に保ち，ラウンドロビンに要求プログラムに割り当て，メモリ，ディスク，およびネットワーク等については on-demand に割り当てている．時間軸上の制約と QOS の制約がないアプリケーションの場合には，使用可能な資源が少ない場合は，それぞれの処理時間を伸ばすことで全体の調整が自然と行われる．

しかし，このような資源割当ポリシを連続メディア処理に適用した場合，時間軸上の制約と QOS の制約が守ることができなくなる．各連続メディアアプリケーションはそれぞれの制約に基づいて必要な資源を確保し，カーネルはその利用を保証する必要がある．すなわち on-demand, fare-share な割当ポリシではなく，予約に基づいて資源を割り当てる必要がある．また，使用可能な資源が少ない場合には，時間軸上の制約は維持しつつ，アプリケーションが許容する QOS に基づいて，各アプリケーションへの割り当て資源量を抑制する必要があり，高度なポリシに基づいた調停によって全体の調整を行い，それによって資源の管理を行うことになる．

以上の考察から，オペレーティングシステムは資源を時間方向と空間方向の 2 次元に分割した資源スロット (**図 1.**) を単位として管理し，アプリケーションは予め必要な量のスロットを予約し，予約した資源についてはアプリケーションが自由に利用できるものとする．各資源の具体的な管理単位の例を**表 1.** に示してある．なお，これまでは特権がない限りアクセスできなかった資源についても，ある決められた範囲であればアクセスさせても差し支えない場合がある．そのような資源についても資源スロットを導入することで，特権を持たないアプリケーションまたは限定された権限をもったアプリケーションに対してアクセスを許すようなことも可能となる．

2.3 外部資源マネージャ

カーネルは資源管理のためのメカニズムのみを提供し，個々の資源管理ポリシはカーネル外に置いた資源マネージャによって実現する．例えば，最も簡単な資源マネージャはアプリケーションからの仮想ページ割当や物理ページへの貼り付けの要求の許可や禁止を行う．言い換えると，資源要求に対するアドミッション制御を行う．より高機能な資源マネージャは，資源が不足したとき，より優先度の高いアプリケーションのために，より優先度の低いアプリケーションに資源の解放を依頼する．これもアドミッション制御の一種と言える．

図 2. に外部資源マネージャを用いたシステムの構成例を示す．資源マネージャは起動時に様々なシステムのパラメータを読み込み，アプリケーションの種類

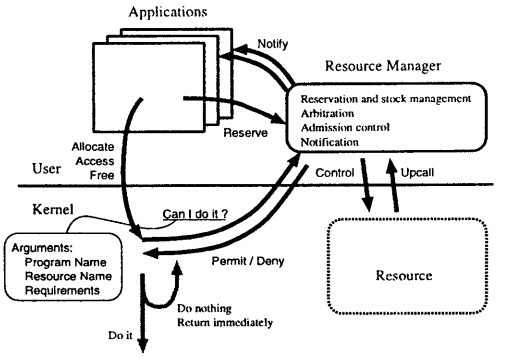

図 2. 資源マネージャ．

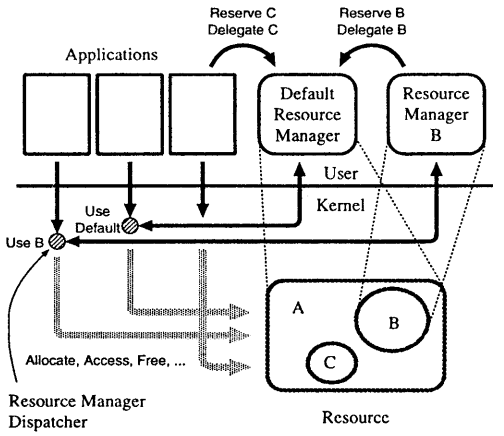

図 3. 複数の資源マネージャ．

別やその権限の種類別に割り当てることの出来る資源量を設定する．アプリケーションは資源の割り当てが必要になったときカーネルへ割当要求をだす．カーネルはプログラム名，リソース名，要求量を資源マネージャへ転送する．資源マネージャは現在の資源利用量をもとに，この要求を許可するかしないかを判断し，その結果をカーネルに通知する．カーネルはその結果をもとに資源の割り当てを続行したり，割り当て失敗の応答を返したりする．カーネルと資源マネージャは，プロセス間通信機能や共有メモリを利用して資源管理の情報を交換する．もし資源に余裕がないなら，資源マネージャは自身のポリシに従って，各アプリケーションに資源の開放を要求したり，強制的に開放した場合はその旨を通知する．

図 3. のように，ある資源に対する資源マネージャは複数おくことも可能である．この場合，資源の全体を管理するものや，資源一部を資源として委譲しても

らいその部分を管理するものなどを考えることができる．また，あるアプリケーションが資源の利用に関して信頼できるならば，または，資源マネージャの機能を合わせもつのであれば，カーネルへの要求を逐次資源マネージャに転送することなくそのまま実行することも許され得る．ある資源には必ず資源マネージャが存在し，どの資源マネージャを選択するかはタスクが決定する．

3 RT-Mach 上の外部資源マネージャの実現

3.1 カーネルインターフェース

資源マネージャのインターフェースは，ユーザレベルで動作する資源マネージャがシステムの資源管理方策を制御したり，アプリケーションが資源管理状況を調査することを可能とする．

資源マネージャの存在を無視したアプリケーションがあったとしてもシステムを正しく動作させるためには，アプリケーションからのカーネルへの要求を資源マネージャへ転送し，その指示にカーネルが従うような機構が必要である．資源には，仮想メモリやCPUサイクルのような抽象的な資源と，より具象的な資源であるデバイスに分けられる．いずれの資源に対する要求も，通常はRT-Machカーネルによって解釈がなされるが，デバイスの場合は，デバイスごとにカーネルで管理された名前を持ち，この名前から **device_open** により個々のデバイス専用のポートを作り出し，それに対して要求を出すようにインターフェースがつくられている．そこで，デバイスとして扱える資源の場合は，デバイス名とRT-Machのポートの組をカーネルへ登録することで，カーネルを経由した要求の転送によらずに直接的に資源マネージャへ要求を送ることができる．

資源マネージャをユーザレベルで実現するために，**表2.** に示すインターフェースを用意する．資源マネージャには，

- デフォールト資源マネージャ (RM_DEFAULT)
- 資源管理サブマネージャ (RM_SUBMANAGER)
- デバイス型資源マネージャ (RM_DEVICE)

の3つのクラスがある．

ある資源に対する資源マネージャはホスト内に複数存在し得るが，デフォールト資源マネージャはホスト内にただひとつ存在し，その資源全体を管理する．それ以外の資源管理サブマネージャは，別の資源マネージャに対して資源を予約した後，その予約した資源内の資源の管理を行うものである．デバイス型については後述する．資源マネージャをカーネルに登録する **host_set_resource_manager** の呼び出しには特権が必要である．

タスクごとの資源マネージャを指定する **task_set_resource_manager** の呼出しはデフォールト資源マネージャに伝えられ，そこで指定を許可するかしないかが判断される．タスクで資源マネージャを指定しない場合には，デフォールト資源マネージャが使われる．設定済のタスク資源マネージャの指定を削除した場合には，デフォールトが有効となる．

デバイスを資源マネージャによって管理することも可能である．これを行うのがデバイス型資源マネージャであり，デバイスに対する全てのカーネルコールを横取りする．デバイス型資源マネージャでは，既に *resource_name* が定義されているなら，古い定義が隠されて新しい定義が有効となるが，この呼出しが終了する前に **device_open** によって得られたポートは有効のままである．もし定義を削除したときに同じ名前で古い定義があるならば，それが有効となる．

カーネルが資源マネージャを呼び出すインターフェースは以下の通りである．なお，引数を省略してある．

- **rm_init** – 資源マネージャを初期化する．資源管理情報がマネージャに伝えられる．
- **rm_terminate** – 資源マネージャを終了する．
- **rm_resource_request** – カーネルに対する資源の割り当てや開放の要求について，資源マネージャに許可を求める．
- **rm_status_changed** – カーネル内の資源管理情報が **rm_request_permitted** によらずに変化したことを通知する．
- **rm_status_reply** – カーネル内の資源管理情報を通知する．

資源マネージャがカーネルへ応答するインターフェースは以下の通りである．

- **rm_change_completed** – **rm_status_changed** や **rm_init** に対する同期が完了したことをカーネルに通知する．
- **rm_request_permitted** – **rm_resource_request** を許可するか否かを通知する．
- **rm_report_status** – カーネルへ資源管理情報の通知を依頼する．

なお，カーネル空間とユーザ空間の共有メモリ機能 [9] を用いて，資源マネージャとカーネルの間で資源管理情報を共有することができる．

3.2 資源マネージャインターフェース

アプリケーションと資源マネージャ間のインターフェースは資源マネージャごとに決めることができるが，本報告ではジェネリックなインターフェースを持つようにしている．主要なプリミティブを**表3.** に示す．*resource_spec* は，資源名，空間方向および時間方向の資源の量で表される資源スロットの大きさ，アプリケーションの属性，および予約の優先度からなる．予約した資源スロットは新たなRT-Machのポー

表 2. 資源マネージャの登録，削除および検索のプリミティブ．
1. 指定できるのは，すでにカーネルに登録された資源マネージャのうち，デバイス型資源マネージャ以外のものである．
2. それぞれの資源マネージャの登録を削除するためには *resource_manager_port* にナルを指定して呼び出す．

機能	関数形式
資源マネージャの登録[2]	**host_set_resource_manager** (*host_priv_port*, *resource_name*, *resource_manager_class*, *resource_manager_port*)
資源マネージャの検索	**host_get_resource_manager** (*host*, *resource_name*, *port*)
タスク資源マネージャの指定[1,2]	**task_set_resource_manager** (*task*, *resource_name*, *resource_manager_port*)
タスク資源マネージャの検索	**task_get_resource_manager** (*task*, *resource_name*, *port*)

表 3. 資源予約関連の主なプリミティブ．

機能	関数形式
資源の予約	**rm_reserve** (*resource_manager_port*, *task_port*, *resource_port*, *resource_spec*)
権限の委譲	**rm_delegate** (*resource_manager_port*, *task_port*, *resource_port*)
資源予約の解除	**rm_free** (*resource_manager_port*, *resource_port*)
notify の設定	**do_rm_notify_resource_shortage** (*notify_port*, *resource_port*)
	do_rm_notify_resource_stolen (*notify_port*, *resource_port*)

ト(オブジェクト)として定義され，通常の RT-Mach のオブジェクトとして同等に扱うことができる．特に，デバイス型の資源マネージャでは，デバイスに対するインターフェース (**device_read**, **device_write**, **device_get_status**, **device_set_status** 等) をそのまま用いることができる．

多くのオペレーティングシステムでは，システムにとって危険の可能性がある資源の割り当てやアクセスを制限するために，特権が必要なシステムコールやケーパビリティを用いて安全性を高めるようになっている．資源マネージャでは，マネージャへのアクセスに制限はないが，予め決められた量以内であれば予約要求を受け付けることで制限を加えることになる．しかし，何に対して予約の上限を決め，予約の主体 (アプリケーションの種類やユーザ等) をどうやって識別し認証するかの問題を解決しなければ，アプリケーションが嘘をつくことで容易に他に対して割り当てるべき資源を横取りされてしまう．このため，資源マネージャは認証結果に基づき，予約主体を識別する機能が必要となる．そこで，*resource_spec* 中のアプリケーションの属性では，アプリケーションのチケットを指定するようになっている．

4 資源マネージャの例

4.1 メモリマネージャ

カーネル内のメモリ資源の管理では，仮想ページ，物理ページへくくりつけられた仮想ページ (wired page)，カーネルとユーザ間の共有ページ (projected buffer) の 3 種類のメモリページの割り付けと解放についてメモリマネージャへ許可を求めるようになっている．予約やアドミッション制御は，あらかじめアプリケーションごとの最大利用量を決定しておき，それを越えたら拒絶することによって行う．資源が不足してきた場合は，優先度の低い順に notify を行う．これにより，特権を持たないアプリケーションでも，予め定められた範囲内で物理メモリの割当が可能となっている．

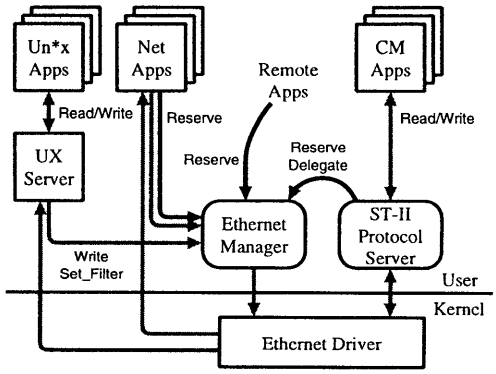

図 4. イーサネットマネージャ．

4.2 イーサネットマネージャ

イーサネットマネージャは帯域幅の制御機能の無いイーサネットでその機能をエミュレートする．これを中心としたプロトコル処理部の全体を図 4. に示す．イーサネットマネージャをデバイス型の資源マネージャとしてカーネルに登録した後，イーサネットデバイスを **device_open** すると，通常はマネージャのポートが返される．このため，Unix のエミュレーションを行う UX Server やネットワークデバイスを直接アクセスするネットワークアプリケーションで

は，ネットワークデバイスへの要求は全てマネージャへ直接送られる．

イーサネットマネージャは送り元と宛先のIPアドレスとポートの対，帯域幅，平均パケット長の3つ組を資源スロットとして管理し，アプリケーションからの要求によって資源スロットの予約，割り当て，解放を行う．そして，アプリケーションからのパケット送信要求を資源スロットによってスケジューリングし，イーサネットドライバに送る．この処理にはCPU資源が必要となるため，イーサネットマネージャはCPUを管理するマネージャに対し，CPU上のタイムスロットの予約を行う．イーサネットには帯域幅等のQOSを制御する機能はないため，イーサネットセグメントに接続された全てのホストが同様な方法でネットワークをアクセスする場合にのみ近似的に帯域幅制御ができるだけである．パケットの受信の場合には，送り元がマネージャに対して予約を行う．

ST-IIプロトコルサーバ[7]は，イーサネットマネージャに対して資源スロットを予約し，その管理の権限を委譲してもらい，プロトコルサーバ自身でパケットの送信をスケジューリングする．つまり，このサーバからのネットワーデバイスへの要求は，マネージャには伝わらずに，ネットワークデバイスに直接送られる．これによって，オーバヘッドを小さくしている．

5 おわりに

本報告では，QOSの保証という観点から，資源を空間方向と時間方向に分割した資源スロットという単位によって保護，管理することを提案した．RT-Machにこれを管理するインターフェースを定義し，資源管理のためのマネージャをユーザレベルのサーバとして実現する方法について述べた．資源マネージャをおくことで，QOSの制御ができるようになることはもとより，一般には特権が必要なカーネルへの要求について，ある状況下では一般のアプリケーションもそのような要求ができるようになるという効果もある．さらに，デバイス型インターフェースを利用することで，カーネル外のサーバがデバイスとして見えるようになるため，ユーザレベル・デバイスドライバの一般的なインターフェースをも提供している．今後は，資源の予約，割り当て，調停のアルゴリズムの検討および評価とQOSチケットモデル[6]との統合を行う予定である．

謝辞

貴重なご討論を頂いている慶應義塾大学環境情報学部斎藤信男教授をはじめとする慶應大学マルチメディア統合環境基盤ソフトウェア(Keio-MMP)プロジェクトの皆様に感謝いたします．

参考文献

[1] Accetta, M., Baron, R., Bolosky, W. J., Golub, D., Rashid, R., Tevanian, A., and Young, M.: Mach: A New Kernel Foundation for UNIX Development, in *Proceedings of the Summer 1986 USENIX Conference*, 93–112 (1986).

[2] Cheriton, D. R.: The V Distributed system, *Communications of the ACM*, **31**, 3, 314–333 (1988).

[3] Engler, D. R., Gupta, S. K., and Kaashoek, M. F.: AVM: Application-Level Virtual Memory, in *Proceedings of the 5th Hot Topics in Operating Systems* (1995).

[4] Engler, D. R. and Kaashoek, M. F.: Exterminate All Operating System Abstractions, in *Proceedings of the 5th Hot Topics in Operating Systems* (1995).

[5] 河内谷, 緒方, 西尾, 徳田: 連続メディア処理のQOS制御のためのOSサポート, 第6回コンピュータシステム・シンポジウム論文集, 119–126 (1994).

[6] 河内谷, 緒方, 西尾, 徳田: 連続メディアのQOS制御のための「QOSチケット」モデル, 第50回情処全大論文集 (1N-6) (1995).

[7] 木原誠司, 盛合敏, 南部明: ST-IIプロトコルサーバのReal-Time Machへの実装と評価, 情報研報, **94**, OS-65, 81–88 (1994).

[8] Mercer, C., Rajkumar, R., and Zelenka, J.: Temporal Protection in Real-Time Operating Systems, in *Proceedings of the 11th IEEE Workshop on Real-Time Operating Systems and Software* (1994).

[9] 盛合敏, 木原誠司, 南部明: 連続メディアオブジェクトに対応したメモリ管理機構について, 情処研報, **94**, OS-64, DPS-65, 73–78 (1994).

[10] Moriai, S., Kihara, S., and Nambu, A.: A Memory Management Mechanism and A External Resource Manager Interface for Continuous Media Objects, in *Abstracts of Mach/Chorus Workshop, USENIX First Symposium on Operating Systems Design and Implementation*, URL:<http://www.cs.utah.edu/~lepreau/osdi/mach.html> (1994).

[11] Mullender, S. J., Rossum, van G., Tanenbaum, A. S., Renesse, van R., and Staveren, van H.: Amoeba : A Distributed Operating System for 1990s, *IEEE Computer*, **23**, 5, 44–53 (1990).

[12] 斎藤他: マルチメディア統合環境のテストベッドとその評価, 情処研報 93-ARC-99, 17–24 (1993).

[13] 徳田, 斎藤: 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, 情処研報 93-ARC-99, 9–15 (1993).

[14] Tokuda, H., Nakajima, T., and Rao, P.: Real-Time Mach: Towards Predictable Real-Time Systems, in *Proceedings of the USENIX 1990 Mach Workshop* (1990).