システムソフトウェアと
オペレーティング・システム 84−30
（２０００．５．２６）

wrapper型資源予約機構の実装と評価

梶原 史雄　盛合 敏
NTT 情報流通プラットフォーム研究所

アプリケーションのバグや DoS 攻撃による資源の浪費に対して，必要な資源を予約する資源予約機構が有効である．しかし，従来の予約ドメインを利用する統合的資源予約機構は新規にアプリケーションを作成することを前提としているため，既に普及しているアプリケーションに対して資源予約機構を適用することができない．本論文では，予約ドメインを既存のアプリケーションに対して適用できるようにアプリケーション単位で資源の予約が可能な wrapper 型資源予約システムを提案する．この wrapper 型資源予約システムを RT-Mach と Lites サーバに実装し，代表的なアプリケーションである Apache サーバを用いて評価を行い，本システムの有効性を示す．

A wrapping approach to resource reservation for legacy applications

Masao Kajihara　Satoshi Moriai
NTT Information Sharing Platform Laboratories

Resource reservation is usefull for preserving resource monopolization caused by a program bug in an application and a denial-of-service attack. However, every conventional resource reservation mechanism is not applicable for a legacy application. Since, it assumes that an application is albe to use a framework provided by the reservation mechanism in order to reserve resources for itself. We propose a wrapping approach to resource reservation for legacy applications and implement it on the Lites server that is a FreeBSD-compatible server on Real-Time Mach.

1　はじめに

現在広く普及しているオペレーティングシステム (以下, OS) では，タイムシェアリングなどの手法で，マルチプロセス環境が実現されており，複数のプロセスがメモリ，ネットワーク帯域，プロセッサ時間などの資源を必要に応じて動的に取得し，利用することができる．このような OS では，特定のプロセスがプログラムのバグなどの原因により資源を浪費し，他のプロセスが実行されるために必要な資源が減少することで実行が困難または不可能となる場合がある．また，ネットワーク経由のサービスを行なうサーバアプリケーションを実行している場合では，DoS 攻撃 (Denial of Service Attack)[?][?] やサービス要求の集中などによってもサーバアプリケーションのプロセスが資源を浪費してしまい，他のプロセスの実行を阻害することがある．

このような特定プロセスによる資源の浪費を解決する方法として，プロセスに対して動的に割り当てる資源の上限を予約する資源予約が研究されている．資源予約はプロセッサ時間の予約，ネットワーク帯域の予約，ディスク帯域の予約など個々の資源についての予約手法が研究されてきたが，統合的に資源予約を行なう機構はまだ少ない．研究が行われている統合的資源予約機構では従来の資源割り当て単位であるプロセスとは直交した予約ドメイン (reservation domain) を利用する仕組みとなっている場合が多い [?] [?]．予約ドメインを利用する資源予約では１つのプロセスに対して複数のドメインを定義することで，細粒度の資源予約を可能とする．

しかしながら，予約ドメインによる資源予約を現在

既に普及しているアプリケーションに対して適用する際にはアプリケーションのプログラムソースを変更する必要がある. 現在, 普及しているアプリケーションにはバイナリ形式で配布されているものも多く, このようなアプリケーションに対して資源予約を適用することができず, 資源の浪費という問題を解決できない.

本論文では以上のような問題を踏まえて, アプリケーション外部から資源予約を行うことができるwrapper型資源予約機構を提案する. また, wrapper型資源予約機構をReal-Time Mach[?](以下, RT-Mach) の予約機構を利用してFreeBSD互換の機能を提供するLitesサーバ[?] 上に実装し, WWWサーバのapacheサーバを用いて評価を行う.

2 関連研究

2.1 Resource Kernel

Resource Kernel[?] [?] はリアルタイムOSの枠組を利用して資源の予約を行うことができる. 精度の高いプロセッサ時間予約ができるhard reservationとプロセッサ時間予約によるディスク帯域制御などを特徴とする.

Resource Kernelでは統合された予約ドメインはないので, 個々のプロセスからプロセッサ時間, ディスク帯域の予約を行う必要があり, 既存のアプリケーションを利用する際にはプログラムソースの変更が必要である.

2.2 resource container

resource container[?]はスレッドやファイル, ソケットを最小単位とした細粒度の資源予約が可能な予約ドメインであり, 階層的な資源予約が行える. さらにLRP(Lazy Receiver Processing)[?] によりカーネル内でのネットワーク処理に費されるプロセッサ時間が正しくresource containerにチャージされるように工夫がなされている.

しかしながら, 予約ドメインであるresource containerの導入のため, 既存のアプリケーションを利用するにはプログラムソースに大幅な変更が必要となる.

2.3 Eclipse/BSD

Eclipse/BSD[?] はFreeBSDを元に階層的予約を可能とする予約ドメインと予約ドメインを表現する/reserveファイルシステムを組み込んである. プロセスやファイルディスクリプタと予約ドメインをタグ付けにより結びつける柔軟な予約システムを特徴とする.

Elipse/BSDはFreeBSDを元にしているので, 既存のアプリケーションを利用する際にはプログラムソースの変更は最低限で良いが, バイナリ形式で配布されるアプリケーションに対する資源予約が行えない.

3 既存のアプリケーションに対する資源の予約

3.1 細粒度予約の問題点

従来の予約ドメインを用いた統合的資源予約機構では細粒度の資源予約を行うことができる.

例えば, 1つのWWWサーバアプリケーションで複数の仮想サイトを作成したり, 複数のユーザで利用したりする際に仮想サイトごと, あるいはユーザごとに予約ドメインを設定して, 個々の予約ドメインに対して資源予約を行うことができる.

但し, このような統合的資源予約機構では上記のようなアプリケーションを新規に作成することを前提としており, 既存のアプリケーションに適用することを考慮されてはいない.

統合的資源予約機構ではプロセス(あるいはスレッド)のプロセッサ時間予約およびディスクやネットワークなどのデバイスへの入出力帯域幅予約を行なっているので, アプリケーションに資源予約の枠組を適用するにはアプリケーションの変更が必要となる. アプリケーションの変更は主にデバイスに関する入出力部分とプログラムを実行するプロセス(あるいはスレッド)自体を適切な予約ドメインに関連づけて資源予約を行うことになる.

このような既存のアプリケーションの変更を行なう際には以下のような問題がある.

1. バイナリ形式のアプリケーション
   プログラムソースが公開されているアプリケーションであればプログラムの変更は可能ではあるものの, バイナリ形式で配布されるアプリケーションは変更できない. Linuxを始めとするUNIX系OSでもバイナリ形式で配布されるアプリケーションが増えつつある事実は無視できない.

2. 予約ドメインの適用と資源の予約

プログラムソースが公開されているアプリケーションでもアプリケーションを幾つの予約ドメインに分割するかという問題がある. 細分化しすぎると, アプリケーション全体で予約する資源の全体像が把握しづらくなる.

アプリケーションの変更を伴なわない上記のような問題に対して本論文ではアプリケーションの変更を伴わない資源の予約方法を模索した.

アプリケーションの変更なしに細粒度の資源の予約を行うにはアプリケーションからOSに対して行われるプロセスの作成, ファイルに対するファイル・ディスクリプタの作成, ネットワークソケットの作成などに対して意味付けをして, 各々をどの予約ドメインに属させるかを自動的に行う必要がある. しかし, 通常のアプリケーションではファイル・ディスクリプタやソケットの生成は頻繁に行われる. これらについて意味付けを行うのは困難である.

3.2 プロセス群による予約

本論文ではアプリケーション全体を一つの予約ドメインとして扱うことで, プログラムの変更なしに資源の予約を行うことを目指す.

アプリケーション全体は通常単一の主プロセスあるいは単一の主プロセスから作成される複数の副プロセスで構成される. そこで, 予約を行うアプリケーションの主プロセスに対して予約ドメインを定義し, 主プロセスから作成される副プロセスはすべて同じドメインに属するようにする. つまり, ある予約ドメインに属するプロセスから作成されるプロセスも親プロセスの予約ドメインに属するように工夫する.

また, 特定の予約ドメインに属さない (予約を行うアプリケーションを構成するプロセスではない) プロセスは予約ドメインに対して予約された資源の残りを使って動作するようにする.

OSの起動時に初期化プロセスが“デフォルト”の予約ドメインを生成し, この初期化プロセスからforkされて生成されるプロセス (通常の状態での全プロセス) はこの“デフォルト”の予約ドメインを継承する. 初期化プロセスの起動時には“デフォルト”の予約ドメインはすべての資源を利用できるように資源の予約を行なう. すなわち, 明示的に予約ドメインを作成していない状態ではどのプロセスでも資源を制限なく動的に取得して利用できる.

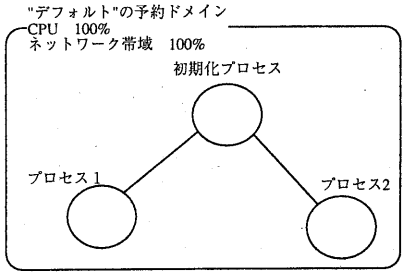

図 1: “デフォルト”の予約ドメイン

例えば, 新規に予約ドメインを取得していない状態で初期化プロセスとプロセス 1, 2, 3 が存在する場合には (図??参照), 初期化プロセスとプロセス 1, 2, 3 との資源取得の合計が資源全体に達するまで資源を動的に取得して利用できる.

新規に予約ドメインを作成する場合には予約する資源と量を指定して予約ドメインを作成する. この際, “デフォルト”の予約ドメインからは新規予約ドメインが予約した分の資源の量が減少する.

図??の例において, あるアプリケーションに対して新規に“予約ドメイン 1”をプロセッサ時間を 20%, ネットワーク帯域を 50%というような資源の予約で作成して, プロセス 3 を結びつけると“デフォルト”の予約ドメイン内のプロセス 1, 2 は合計でプロセッサ時間が 80%, ネットワーク帯域が 50%までしか使えない. 逆に作成された予約ドメイン 1 内のプロセス 3, 4 は合計でプロセッサ時間を 20%, ネットワーク帯域を 50%まで利用できる. さらにプロセス 3 がプロセス 4 を作成すると, プロセス 4 はプロセス 3 の予約ドメインを継承して“予約ドメイン 1”に結びつけられる. (図??参照).

本論文ではこのようなプロセス群による予約ドメインにより従来のアプリケーションに対して資源を予約できる wrapper 型資源予約機構を提案する.

4 wrapper 型資源予約機構

wrapper 型資源予約機構は従来の統合型資源予約機構あるいは資源予約機構を補完して, 既存のアプリケーションに対して資源を予約する枠組を提供する.

wrapper 型資源予約機構では前述したようにプロ

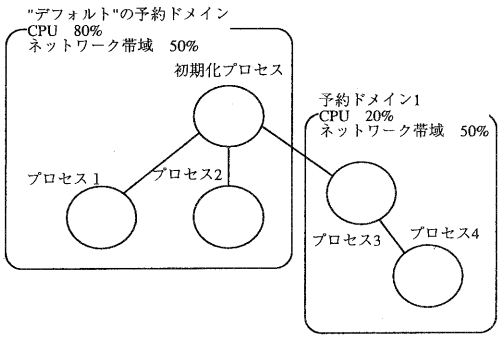

図 2: 予約ドメイン 1 の作成後

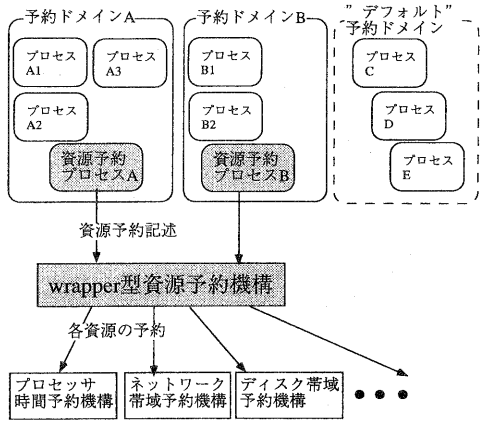

図 3: wrapper 型資源予約機構の構成

セス群を予約ドメインとして定義するが, UNIX 系 OS では process group というプロセス群を管理する枠組が存在する. しかしながら, process group は特権がなくとも setpgrp システムコールにより容易に変更できるため, setpgrp システムコールを使うアプリケーションではアプリケーションを構成するプロセスの一部が予約ドメインから外れてしまい, 意図した資源の予約が行えないことがある.

このような理由により wrapper 型資源予約機構では独自にプロセス群を管理する機構を提供する. 図?? は予約ドメインを持たない従来の資源予約機構を利用する wrapper 型資源予約機構の構成を示す. 以下ではこの wrapper 型資源予約機構の構成要素の機能を説明する.

- 資源予約プロセス
  まず, wrapper 型資源予約機構の予約ドメインであるプロセス群を構成するルートプロセスとしての機能を持つ. 第二に予約ドメインに対する資源を資源予約記述により予約する機能を持つ.
- 資源予約記述
  資源予約記述は予約を行う資源の種類と全資源量に対して予約する資源の割合を記述する.
- wrapper 型資源予約機構
  予約ドメインであるプロセス群の管理と資源予約プロセスから渡された資源予約記述に基づいて各資源予約機構に対する資源の予約を行う.

5 wrapper 型資源予約機構の実装

5.1 プラットホームの選択

wrapper 型資源予約機構は従来の資源予約機構を補完し利用するための枠組であり, 資源予約機構を実装した既存の OS 上に実装できる.

資源予約機構を持つ OS にはネットワーク帯域制御が可能な FreeBSD と Linux, プロセッサ時間の予約とネットワーク帯域の制御が可能な Eclpse/BSD[?] , Linux/RK[?] がある.

これらの OS を選択するにあたり, wrapper 型資源予約機構を実装する OS の条件として以下の点を考慮した.

1. OS のソースが公開されている
   予約ドメインをプロセスと関連づけるので, OS のソースが公開されているもの, 少くとも資源予約機構の API が公開されているものが望ましい.
2. いくつかの資源予約機構が実装されている
   プロセッサ時間, ネットワーク帯域など重要な資源の資源予約機構が実装されているものが望ましい.
3. 一般的なアプリケーションが動作する
   Linux あるいは FreeBSD などの PC/AT 上で動作する UNIX 系 OS か, Solaris や Tru64 UNIX

のようなWS上で動作するUNIX系OSなど商用アプリケーションが動作するOSが望ましい.

表 1: OSによる条件の比較

OS	Linux	FreeBSD	Eclipse/BSD	Linux/RK	RT-Mach
プロセッサ時間予約可能	×	×	○	○	○
ネットワーク帯域予約可能	○	○	○	○	○
実用的アプリケーション利用可能	○	○	○	○	○
カーネルソース入手可能	○	○	×	×	○

しかし, 表??に示すようにwrapper型資源予約機構を実装する際には上記の3条件を満たすOSは存在しなかった. そこで, ソースが公開されており, プロセッサ時間予約機構が実装されており, Litesサーバ上でFreeBSDのアプリケーションが動作するRT-Mach NR2[?]上のLitesサーバに実装を行うことにした.

5.2 ALTQの移植とCBQの改良

RT-Mach NR2ではネットワーク帯域をスレッドやタスクに対して予約する機能がないのでFreeBSDのALTQをLitesサーバ上に移植した.

ALTQ[?]はFreeBSDのパケット送出キューのスケジューリングアルゴリズムを実装できる汎用的な枠組を提供している. ALTQではCBQ(Class Based Queueing)[?] などの代表的なキューイングアルゴリズムがあらかじめ実装してある.

CBQは送出先アドレスと送出先ポート, プロトコル種別などでパケットをフィルタリングするフィルタを定義し, そのフィルタに対応するクラスに対して階層的に帯域を割り当てることでネットワーク帯域を制御できる.

本論文ではCBQによるネットワーク帯域予約を利用するが, オリジナルのCBQでは予約ドメインであるプロセス群とフィルタの対応付けがポートによるものでしか定義できない. この方法では複数のポートを使うアプリケーションではソケットを作る度にフィルタを追加する必要がある.

そこで, CBQのフィルタを改良してパケットの送出元の予約ドメインのIDによるフィルタリングが可能となるように改良した.

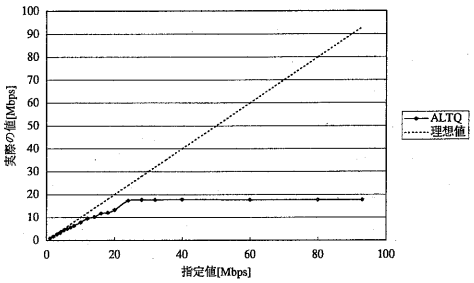

図 4: Liesサーバ上のALTQの性能

5.3 Litesサーバへの実装

wrapper型資源予約機構では予約ドメインをプロセス群として扱うため, UNIXプロセスのプロセス表を参照する必要がある. Litesサーバでは UNIXプロセスの管理をするプロセス表があるため, Litesサーバ内にwrapper型資源予約機構を実装した(図??参照).

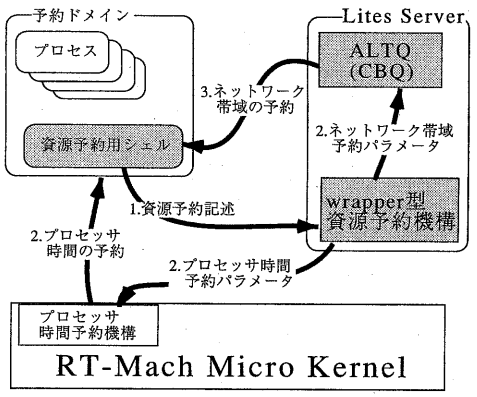

図 5: Litesサーバへの実装

wrapper型資源予約機構では予約ドメイン管理用の予約ドメイン管理表により予約ドメインを管理し, プロセス表にエントリを加えて予約ドメインへのリンクという形でプロセスから予約ドメインへの対応を表すように実装を行った.

本来は各予約ドメインに予約された残りを“デフォルト”の予約ドメインに割り当てるべきだが, Litesサーバ自身も複数のスレッドから構成され, 資源を必要と

することから各予約ドメインに予約された残りの資源のうちの半分を Lites サーバ内のスレッドに割り当てるようにした．さらに半分を “デフォルト” の予約ドメインに割り当てるようにした．

また, 資源予約プロセスとして環境変数を用いて資源の予約が可能な資源予約シェルを実装した．この資源予約シェルから起動されたプロセスはすべて資源予約シェルが定義した予約ドメインに属するようになる．

6 wrapper 型資源予約機構の評価

6.1 評価環境

RT-Mach 上に実装した wrapper 型資源予約機構を代表的サーバアプリケーションである Apache サーバを用いて以下のようなサーバホスト 1 台とクライエントホスト 1 台を FastEtherSwitch で接続した実験環境により評価する．

1. サーバホスト 1 台
   - CPU Celeron 300Mhz
   - Memory 256Mbytes
   - FastEtherNIC(fxp)
   - OS : FreeBSD-3.4R
     FreeBSD-2.2.8R
     Linux-2.2.13
     RT-Mach+Lites
     (+wrapper 型資源予約システム)
2. クライエントホスト 1 台
   - CPU Celeron 300Mhz
   - Memory 256Mbytes
   - FastEtherNIC(fxp)
   - OS : WindowsNT-4.0

この環境においてサーバホスト上で動作している apache-1.3.11 に対してクライエントホストから WebBench3.0[?] というベンチマークアプリケーションを用いてクライエント側から見たスループット [req/sec] を測定し, 評価の目安とする．

6.2 プロセッサ時間予約の評価

Lites 上に実装した wrapper 型資源予約機構と資源予約シェルにより Apache サーバに対してプロセッサ時間を 40%, 20%, 10%と予約したときの WebBench の結果を図??に示す．

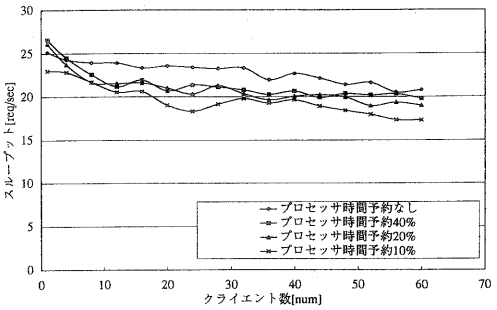

図 6: プロセッサ時間予約の結果

予約するプロセッサ時間を 40%, 20%, 10%と減少させるとスループットが低下するが, わずかである．これより Apache サーバ自体はプロセッサ時間を浪費してはいないことが推測できる．

6.3 ネットワーク帯域予約の評価

Lites 上に実装した wrapper 型資源予約機構と資源予約シェルにより Apache サーバに対してネットワーク帯域を 20, 10, 8, 6, 4, 2, 1Mbps に予約したときの結果を図??に示す．

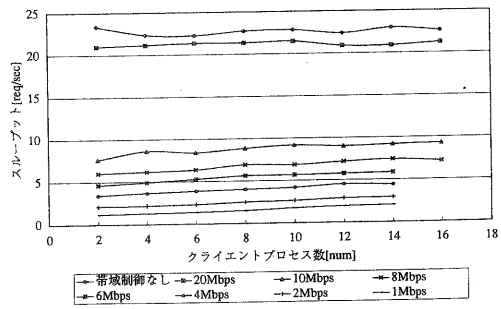

図 7: ネットワーク帯域予約の結果

予約のない状態 (帯域は 100Mbps) から予約するネットワーク帯域を 20Mbps と減らしてもスループットにはほぼ変化がないが, 10Mbps に減少させると極端なスループットの低下が見られる．また, 8, 6, 4, 2, 1Mbps と予約帯域を減少させるとほぼ比例するようにスループットが減少する．

これよりWebBenchによるApacheサーバのスループットが10Mbps以上出ていることがわかる.

6.4 資源浪費プロセスに対する耐性

バグやDoS攻撃などによるプログラムの資源浪費からアプリケーションをwrapper型資源予約機構が守ることができるかを検証する.

WWWサーバホスト上でプロセッサ時間を浪費するプログラムとして空ループを廻すだけの妨害プロセスを複数個作って, これらの妨害プロセスにnice値-20を与えたときのApacheサーバのスループットを測定する.

OSとしてはRT-Mach+Lites以外にFreeBSD-2.2.8R, FreeBSD-3.4R, Linux-2.2.13など代表的なUNIX系OSを用いて比較を行った. なお, wrapper型資源予約機構ではApacheサーバに対してプロセッサ時間を20%, ネットワーク帯域を30%予約した.

この結果を図??に示す.

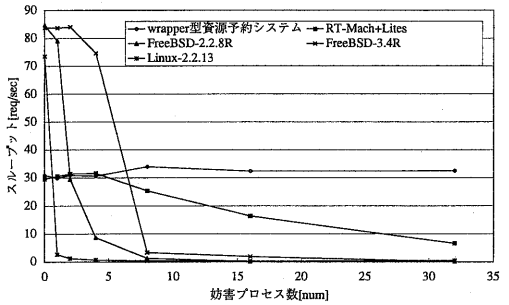

図 8: OSによる妨害プロセスに対する耐性

図??よりwrapper型資源予約機構を実装したRT-Mach+Litesでは妨害プロセス数を32としてもApacheサーバのスループットはほぼ低下しておらず, wrapper型資源予約機構によるApacheサーバに対する資源予約は有効に働いていることがわかる. 一方他のOSでは少からず, 妨害プロセスの影響を受け, LinuxやFreeBSDでは妨害プロセス数を8つに増やすとほぼスループットが0となっている.

また, この図??のスループットの変化傾向より同じUNIX系OSでもスケジューリングポリシが若干異なることが推測される. Linux-2.2.13は妨害プロセスが1つでApacheサーバのスループットがほぼ0となるが, FreeBSD-2.2.8Rは妨害プロセス2つでスループットへの影響があり, FreeBSD-3.4Rは妨害プロセス数が4つでスループットへ影響が出る. また, RT-Mach+Litesは他のUNIX系OSと比較しても妨害プロセス数の増加によるスループットへの影響が少いことがわかる.

7 おわりに

本論文では資源予約機構に対応していない既存のアプリケーションに対して資源予約を行うために従来の予約ドメインをプロセス群に適用するようにするwrapper型資源予約機構を設計し, RT-Mach上のLitesサーバをプラットホームとして選択し, 実装を行った.

実装したwrapper型資源予約機構の有効性を評価するために代表的なサーバアプリケーションであるApacheサーバを用いてwrapper型資源予約機構がバグやDoS攻撃などの原因により資源を浪費するプログラムに対して有効であることの検証を行った. 検証の結果, 資源を浪費する妨害プロセスが32個になると他のUNIX系OSではapacheサーバのスループットがほぼ0となるのに対して, wrapper型資源予約機構を実装したLitesサーバではスループットの低下はまったくみられず, 資源浪費に対して効果があることを実証できた.

今後の課題としてメモリやディスク帯域などの他の物理資源やファイルディスクリプタ, ソケットなどの論理資源についての資源予約機構の統合がある.

参考文献

[1] G.Banga, P.Druschel, and J.Mogul, "Resource containers: A new facility for resource management in server systems.", In Proceedings of the USENIX 3rd Symposium on Operating System Design and Implementation New Orleans, LA, October 1999, Feburary 1999.

[2] G.Banga and P.Druschel, "Lazy receiver processing (LRP): a network subsystem architecture for server systems.", In Proceedings of the second USENIX symposium on Operating systems design and

implementation,Seattle, WA, October 29 - November 1, 1996.

[3] K. Cho, "A framework for alternate queueing: Towards traffic management by pc-unix based routers.", In Proceedings of the USENIX 1998 Annual Technical Conference, New Orleans, Lousiana, June 1998.

[4] R.Rajkumar, K.Juuva, A. Molano, and S. Oikawa, "Resource Kernels: A resource-centric approach to real-time systems.", In Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking, January 1998.

[5] J. Blanquer, J. Bruno, E. Gabber, M. Mcshea, B. œ,AVzden, and A. Silberschatz, "Resource Management for QoS in Eclipse/BSD.", In Proceedings of the FreeBSD 1999 Conference, Berkeley, California, October 1999.

[6] J. Bruno, E. Gabber, B. œ,AVzden, and A. Silberschatz, "The Eclipse Operating System: Providing Quality of Service via Reservation Domains.", In Proceedings of the USENIX 1998 Annual Technical Conference, New Orleans, Louisiana, June 1998.

[7] S. Oikawa and R.Rajkumar, "Linux/RK: A Portable Resource Kernel in Linux.", 1998 RTSS Work-in-progress Session, Madrid, December 1998.

[8] Roger M.Needham, "Denial of Service.", In Proceedings of the 1st ACM Conference on Computer and Communication security, pp.151, 1993.

[9] "CERT/CC Denial of Service.", http://www.cert.org/tech_tips/denial_of_service.html

[10] Floyd. S and Jacobson. V, "Link-Sharing and Resource Management Models for Packet Networks", IEEE/ACM Transactions on Networking, Vol.3, No.4(1995), pp.365-386.

[11] H. Tokuda, T. Nakajima and P. Rao, "Real-Time Mach: Towards a Predictable Real-Time System. ", In Proceedings of USENIX Mach Workshop, Otctober 1990. http://www.cs.cmu.edu/afs/cs/project/rtmach/public/papers/rtmach90.ps

[12] "Real-Time Mach NTT Release", http://info.isl.ntt.co.jp/rtmach/

[13] "UNIX under Mach The LITES Server", http://www.cs.hut.fi/ jvh/lites.MASTERS.ps

[14] "WebBench 3.0" http://www.zdnet.com/zdbop/webbench/