システムソフトウェアと
オペレーティング・システム 89－１４
シ　ス　テ　ム　評　価 2－１４
（２００２．２．１５）

スケジューラの動的な変更を可能とする オペレーティングシステムの構成手法

笠井 秀一† 毛利 公一†† 平瀬 吉也††† 森 英悟 †††

† 東京農工大学大学院工学研究科
†† 東京農工大学工学部
††† ノキア・ジャパン 株式会社

本論文では，スケジューラの動的な変更を可能とするオペレーティングシステムの構成手法について述べる．本システムでは，メモリ管理，スケジューラ，ネットワーク処理などの，オペレーティングシステムの主要となる機能をモジュール化し，動的にそれら主要機能を追加，削除，変更を可能にするシステムをユーザに提供する．ユーザは，本システムが提供するフレームワークに沿ってモジュールを構成することで，任意の処理をカーネルモジュールとして組み込むことができる．本論文では，スケジューラの動的な変更を可能とするインタフェースの実装と性能評価について述べ，その有効性について議論する．

A Design of Dynamic Configurable Scheduler for an Operating System

Syuichi Kasai† Koichi Mouri†† Yoshiya Hirase††† Eigo Mori†††

†Graduate School of Engineering, Tokyo University of Agriculture and Technology
††Faculty of Engineering, Tokyo University of Agriculture and Technology
†††Nokia Japan Co., Ltd.

This paper presents a design of an operating system, which enables dynamic changes of its main functions, such as network system, memory management and process scheduler. These functions are implemented as modules in the system, thus can be loaded and unloaded when it is needed. The design makes it possible to implement most appropriate algorithms for specific applications dynamically and improve its performance significantly without reconstructing operating system. In this paper, an implementation of a process scheduler using the design is shown. We also conducted a performance evaluation on that and show the effectiveness of the design.

1 はじめに

現在，ネットワークインフラの整備が進み，大容量のデータ通信が個人のコンピュータで扱えるようになったことにより，リアルタイムでデータを転送する動画配信サービスなどの，マルチメディアコンテンツサービスが活発になっている．また，最近ではデスクトップ型PCだけでなく，携帯電話や PDA(Personal Digital Assistans) などの携帯情報端末でも，ブロードバンドネットワークを活用したサービスが展開されている．コンサートやスポーツのライブ中継や，映画などが個人のストレージに保存してから見るのではなく，ネットワークを使ってリアルタイムに送受信が可能となり，そのクォリティも向上している．また，それらサービスを受けるためのソフトウェアアプリケーションも，ユーザは利用できるようになっている．Microsoft 社の Mediaplayer[1] やリアルネットワークス社の Realplayer[2] などがユーザに広く普及しているマルチメディアアプリケーションである．

ネットワークインフラ，そしてアプリケーションが整備されていく中，それらの間に介在し，相互の同期などの処理を行うオペレーティングシステム (以下 OS) にも様々なサービスや，アプリケーションの要求を処理できる機能が必要となっている．現在の OS では，リアルタイム性を持たせるならばリアルタイム OS を，スループットを向上させるならメモリ管理機能などの性能を向上させた OS というように，用途に合わせた OS を採用するすることで，様々なサービスに対応できるようにしている．

しかし，ユーザやアプリケーション開発者にとって，機能に特化した OS をサービスに合わせて使用することは，デメリットが多く存在する．ユーザにとっては，あるアプリケーションを動かしたいが，その性能を引き出すためには専用 OS に乗り換えねばならず，時間とコストがかかる．また，アプリケーション開発者にとっても，開発したアプリケーションに必要な OS の機能が提供されていないかもしれない．そのため，独自に OS の開発やアプリケーションの変更を行うことを余儀なくされる．

また，必要な機能をすべて OS に組み込んでしまうと，カーネルが肥大化してしまい，PDA のようなメモリなどの資源の乏しい機器では問題である．またその管理機能が複雑となり，オーバヘッドの原因となることがある．これらのことから，必要な機能を必要な時にだけ組み込むことが重要となる．

これらの問題を解決するために，本論文では Dynamic Kernel Configuration System(以下 DKCS) による，OS の主要機能を動的に変更可能にするシステムを提案し，その実装結果について報告する．ここで述べる OS の主要な機能とは以下の 3 つである．

- スケジューラ
- メモリ管理
- ネットワーク

これらの主要な機能を動的に追加，変更できるようになることで，以下のようなメリットが生まれる．

- 拡張性の向上
- バージョンアップの容易性
- アプリケーション開発者における，テストデバックの容易性の向上

DKCS を使用することにより，OS の仕様に合わせてアプリケーションを変更，開発するのではなく，アプリケーションの要求に合わせて，OS を任意に変更，拡張することができる．また，携帯端末の持つシステム資源の制約に柔軟に対応することができる．アプリケーション開発者にとって，OS の環境を，自分達の要求するような環境に簡単に変更，追加できることは，アプリケーション開発におけるコストの削減に大きくつながる．また，機能をバージョンアップする場合でも，OS の再インストールや再起動をするのではなく，必要な機能を，動的にバージョンアップすることが可能となる．これにより，アプリケーションとバージョンアップキットをペアでユーザにダウンロードしてもらい，アプリケーションのインストールとともに，OS 自身もアプリケーションにあわせた環境変更を行うことが可能となる．

本論文では，始めにスケジューラをターゲットとして開発を進める．スケジューラをモジュール化し，動的に組み込むシステムを開発し，実装することで，DKCS の仕様確定と，有用性を検証する．

本論文では，2 章で動的組み込み方式について述べ，3 章で提案する DKCS の特徴と全体構成を述べる．4 章では，その実装方式について述べ，また，5 章でオーバヘッドに関する検証結果を述べる．

2 動的組み込み方式

現状の OS では，カーネルの機能を動的に拡張する機構として Loadable Kernel Module[3](以下 LKM) がある．LKM は，Linux などに実装されている，動的にカーネルの機能を拡張する機構である．機能を拡張できるものとして，現在ではデバイスドライバ，ファイルシステム，ネットワークプロトコルなどがモジュールとして動的にカーネルにロードすることができる．LKM では，モジュール内で定義した関数に対して，モジュールをロードする際にメモリの確保，シンボルテーブルの追加などを行い，カーネルの機能として登録する．

OS の主要な機能を拡張するための機構として，Mach[4] の外部ページャや SPIN[5] の spindle などが挙げられる．マイクロカーネルは，OS の構成技法のひとつで，Linux などのモノリシックカーネルと対極をなし，OS の API やスケジューリングなどのサービスの機能をサブシステムとしてカーネルの外に出すことで，それらの変更や複数の実装を可能としているシステムである．カーネルには割込み処理，プロセススケジューリング，プロセス間通信などのプロセス管理だけをもたせ，ファイル管理やウィンドウ管理など他の主要な機能をユーザプロセス (モジュール) として実現している．拡張性の他にも，プロセスのある場所を意識せずアクセスできる位置透過性の特徴も持ち，分散システムや並列処理システムにも使用される．

DKCS では，動的に主要な機能 (メモリ管理，スケジューラ，ネットワーク) を追加，変更するために，モジュールとカーネル間のインタフェースを提供する．インタフェースはそれぞれのモジュールの種類によって異なるものを提供し，モジュール内で定義されるサービスの種類に対しては，それぞれのモジュールの種類に共通のインターフェイスを提供する．動的なメモリの確保や，シンボルテーブルの追加は LKM 機能を利用し，DKCS インタフェースが提供するフレームワークに沿ってモジュールを構成する．

3 特徴と構成

3.1 特徴

DKCS の特徴として，以下の二つが挙げられる．

- カーネルの主要機能のモジュール化
- 各モジュール毎のインタフェースを定義

DKCS は，カーネルの主要な機能をモジュール化するためのフレームワークをユーザに提供する．ユーザはそのフレームワークに沿って，任意の処理をプログラムでき，カーネルの主要な機能として登録できるようになる．以下に，ユーザが任意に作成できるモジュールの種類と，そのサービスについてのフレームワークを示す．

- ネットワーク
  - パケットの帯域制御
    ネットワークモジュールでは，帯域制御機構に着目して，パケットの send，receive などのインタフェースに，帯域制御機構システムを，ユーザの望むシステムで動的に組み込めるようにする
  - パケットスケジューリングアルゴリズム
- メモリ
  - ページングアルゴリズム
    マルチメディア向けのメモリ管理機構に着目し，ページングアルゴリズムにユーザカスタマイズ性を持たせる
- スケジューラ
  - スケジューリングアルゴリズム
    スケジューリングアルゴリズムをターゲットとし，ユーザに任意にスケジューリングアルゴリズムを構成し，動的に組み込める機構を提供する

たとえば，スケジューラであれば，リアルタイム性のない OS にリアルタイムスケジューリングアルゴリズムを導入することが可能であり，その着脱の容易性により，複数のリアルタイムスケジューリングアルゴリズムの性能を評価する場合に使用できる．メモリ管理機能ならば，連続メディアアプリケーション向けに先読み機構を向上させたページングアルゴリズムを組み込み，その性能を評価することができる．また，ネットワーク機能では帯域制御機構を備えたソケットインタフェースを動的に組み込むことも可能となる．

3.2 全体構成

DKCS は大きく以下の 2 つの部分に機能を分けることができ，その全体構成は図 1 になる．

- DKCS インタフェース
- DKCS Core

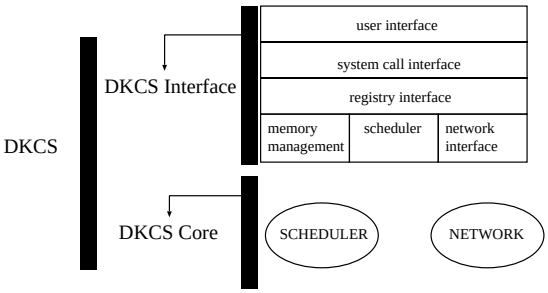

図 1　DKCS の全体構成

以下に DKCS インタフェースと DKCS Core に関してその構成について説明し，次にスケジューラをターゲットとした場合の DKCS インタフェースと DKCS Core について説明する．

3.2.1　DKCS インタフェース

DKCS インタフェースは，メモリ，ネットワークなどのモジュールの種類を，そのモジュールが提供するサービスごとにカーネルに登録するインタフェースである．DKCS はその処理を大きく分けると以下の 4 つに分かれる．

- レジストリインタフェース
- 共通処理関数群
- システムコールインタフェース
- ユーザインタフェース

レジストリインタフェースでは，各モジュールの種類と，そのサービスごとに，カーネルの機能として使用するためにレジストリを行うインタフェースである．各モジュールの種類と，そのサービスは，モジュール内で定義される module_ID と service_ID を使用して識別し，レジストリされる．module_ID はネットワーク，メモリ，スケジューラの 3 つであり，service_ID はモジュール開発者が任意に決定することができる．この module_ID と service_ID は，後述するシステムコールインタフェースにおいて，アプリケーション開発者が使用したいモジュールとサービスを指定して使用する場合にも使用される．レジストリインタフェースにより，使用したいモジュールとサービスのレジストリが，各モジュールとサービスごとに ID を用いて，分別してレジストリができるようになる．

共通処理関数群は，各モジュールの種類ごとに，そのサービスに関係なく使用される処理群が登録されている．ユーザは上記のサービス毎のモジュールと，この共通処理関数群を使用することによって，モジュールのサービスを特定して使用できる．

システムコールインタフェースは，モジュールのサービスをユーザに使用できるようにした，カーネルとユーザ間のインタフェースである．ユーザは与えられたシステムコールインタフェースと，module_ID，service_ID を使用して，使用したいサービスを選択することができる．

ユーザインタフェースでは，システムコールインタフェースをライブラリとしてユーザに提供し，プログラムで実際に上記のモジュールのサービスを使用できるようにしたライブラリ群である．

DKCS の特徴として，各モジュールの種類毎に，カーネルとモジュール間のインターフェイスを構成していることがあげられるが，そのインタフェース部分は，レジストリインタフェースと共通関数群の二つを組み合わせることで，各モジュールの種類に対応してインタフェースを構築している．レジストリインタフェースでは各モジュールの種類毎にレジストリする領域を用意し，共通関数群ではそれぞれのモジュールに共通の処理を定義しておくことで各モジュールの種類を分別している．

3.2.2　DKCS Core

モジュールとなる部分は，それぞれのモジュールの種類によってモジュール化できる部分がユーザに外部仕様として定義されている．ユーザはそれら外部仕様に合わせてモジュールを作成することで，任意のカーネルの主要機能を構築することができる．現段階ではスケジューラの，スケジューリングアルゴリズムをモジュール化し，動的に追加，変更できるように設計している．ユーザは，DKCS システムが提供するインタフェースに沿ってモジュールを構築することで，容易にカーネルへの動的組み込みが可能となる．

3.2.3　スケジューラの DKCS 化

本論文では，まずスケジューラをターゲットとし，スケジューリングアルゴリズムの動的な追加，削除，変更を可能とする DKCS スケジューラ機

構を開発した．その設計について述べる．

DKCS スケジューラは，以下の特徴がある．

- スケジューリングアルゴリズムの動的な追加，変更，削除
- 複数スケジューラの共存

DKCS スケジューラでは，動的にカーネルを変更することで，ユーザが任意のスケジューリングアルゴリズムを使用することができる．また，複数のスケジューリングアルゴリズムが混在するスケジューラを構成することができる．以下にその詳細設計を述べる．

DKCS スケジューラインタフェース　DKCS スケジューラにおける DKCS インタフェースは，スケジューラの，スケジューリングアルゴリズムをモジュール化することをターゲットとしている．そのため，共通処理関数は，タスクを扱う関数群で構成される．モジュールのメインとなる部分はスケジューリングアルゴリズムと，スケジューリングで必要となる関数から構成される．

ユーザインタフェース　ユーザは以下のような手順で，スケジューラモジュールを使用することができる．これらはすべてそのアプリケーションのコード内でおこなう．

(1) スケジューリングアルゴリズムを適用したい部分を特定する
(2) 特定のスケジューリングアルゴリズムを使用する宣言 (システムコール) を行い，通常スケジューラから指定したスケジューラへとプロセスを移行させる
(3) メインの処理をプログラムする
(4) 処理がおわったら，スケジューリングアルゴリズム処理終了の宣言 (システムコール) を行い，指定したスケジューラから通常スケジューラへと移行する

ユーザは，複数のリアルタイムスケジューリングアルゴリズムと，非リアルタイムスケジューリングアルゴリズムを使用でき，それらはユーザが任意に取外しできるよう実装している．リアルタイムスケジューラは，そのアルゴリズムの制限から，リアルタイムスケジューリングアルゴリズム一つと，複数の非リアルタイムスケジューラを登録することができる (図 2 参照)．ユーザは，リアルタイム性能を持たせたいタスクに上記で述べた手順にしたがってプログラミングすることで，そのタスクにリアルタイム性能を持たせることが可能である．

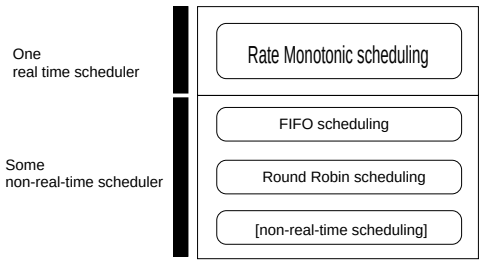

図 2　DKCS スケジューラ

スケジューラの DKCS Core　スケジューラの DCSK Core は，以下のことを定義しておく．

- 次に CPU を割り与えるタスクを決定するスケジューリングアルゴリズムの定義
- ユーザに提供するインタフェースの定義
- スケジューリングアルゴリズムで使用される処理の定義

上記のフレームワークにしたがって，ユーザは任意のスケジューリングアルゴリズムを作成することが可能となる．

本論文では，リアルタイムスケジューリングアルゴリズムとして代表的な RateMonotonic scheduling[6](以下 RM) と Earliest Deadline First[7](以下 EDF) を実装した．RM では，デッドラインミスの影響が他のタスクに及ぶことを最小限にできるため，今回の評価方法の都合から適当と判断した．

4　実装

DKCS は，Linux2.2.14 カーネルをターゲットとして実装を行った．DKCS の詳細な構成は図 3 のようになる．

4.1　システムコールインタフェース

モジュールを使用するときには dkcs_call() システムコールインタフェースを使用する．dkcs_call() 関数は，全モジュールに共通したシステムコール

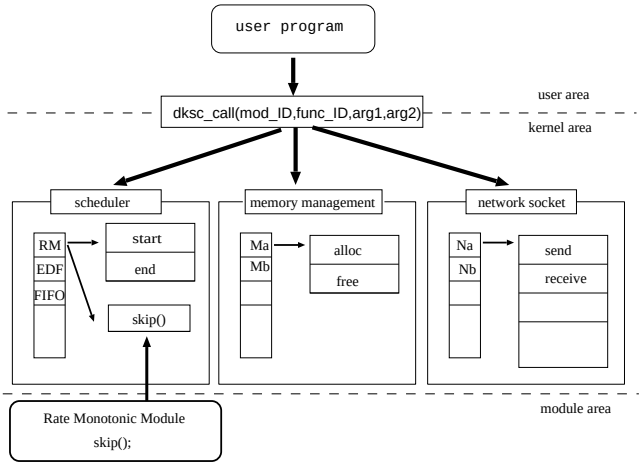

図 3　DKCS の詳細

インタフェースで，module_ID，service_ID によって使用したいモジュールとサービスの種類を特定し，各サービスに必要なパラメータを parameter 構造体に設定する．RM スケジューラモジュールでは，module_ID,service_ID に SCHED,RM を設定する．

4.2　レジストリインタフェース

レジストリインタフェースでは，モジュール内で定義された関数を，カーネル内に登録するためのインタフェースを提供する．それぞれの module_ID と service_ID 毎に配列を用意し，それら関数のポインタを格納する．関数ポインタを格納するタイミングは LKM で init_module を行う際，グローバル宣言された 2 次元配列に格納する．配列の index にはユーザがモジュールを指定するときと同じ index を宣言して用いる．モジュールの関数のカーネルメモリの確保，シンボルテーブルへの追加は LKM を利用する．

4.3　共通処理関数群

共通処理関数では各モジュールで共通の関数を定義している．スケジューラでは以下の関数を定義している．プロセスは，システムコールインタフェースを通して以下の関数を利用し，自らをターゲットのスケジューラのプロセスとして登録する．

- dkcs_sched_start()　プロセスをモジュールスケジューラの Ready キューに移行させる．また，ユーザから受け取ったパラメータをタスク構造体に新たに付加する．
- dkcs_sched_end()　プロセスをターゲットスケジューラの Ready キューから通常のスケジューラへと移行させ，ターゲットスケジューラのプロセスとしての動作を終了し，通常プロセスへと復帰させる．

DKCS では，各スケジューラ毎に Ready キューが用意され，プロセスは起動後に自ら各スケジューラのキューへ移動することで，任意のスケジューリングアルゴリズムで自らを動作できるようにする．dkcs_sched_start() 関数では，プロセスを周期的なプロセスに移行するため，add_to_dkcs_runqueue() 関数を使用して，ターゲットスケジューラへとプロセスを移行し，del_from_dkcs_runqueue() 関数でターゲットスケジューラの Ready キューから通常スケジューラへと移行させる．これらはすべてリスト操作で行われる．

4.4　モジュール

スケジューラモジュールは，スケジューラ関数と，その中で使用される関数を定義する．

- rm_schedule()　RM スケジューリングアルゴリズムにしたがって，次に CPU を割り与えるタスクを決定する．
- skip_next_period()　ユーザに提供する関数で，次の周期までタスクをスリープさせる
- do_RM_process()　タイマ割り込み毎に起動周期時刻をチェックする

module_ID と service_ID をモジュール内で定義し，ユーザはそれらをシステムコールを通して指定することで，これらサービスを使用できるようになる．

5　評価

本研究の評価として，以下の実験を行った．

- 組み込んだスケジューラの動作
- 通常スケジューラのオーバヘッド

5.1 組み込んだスケジューラの動作

本論文では，スケジューラを DKCS Core のターゲットとして用いて，それに対するインタフェースを構築した．実装したスケジューラモジュールは RM アルゴリズムを用いたリアルタイムスケジューリングアルゴリズムである．実験では，動画再生アプリケーションを動作させながら，CPU に負荷を掛ける．そのときの動画再生アプリケーションのフレームレートの低下を，RM スケジューラを用いた場合と，通常スケジューラでプロセスをスケジューリングした場合で測定し，評価を行う．使用した動画再生アプリケーションには，MPEG-PLAY ver.2.3 を使用した．その実験結果を図 4 に示す．

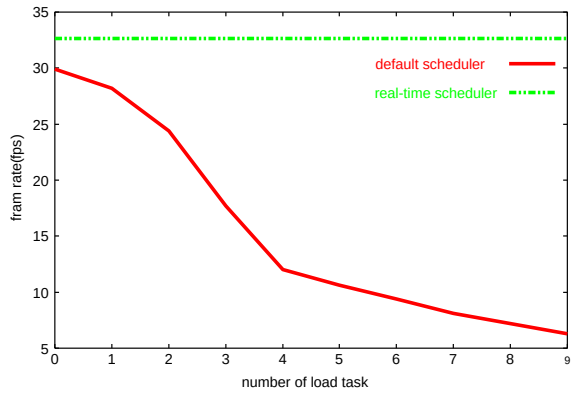

図 4　MPEG_PLAY フレームレート測定

図 4 からわかるように，通常スケジューラで実行した mpeg_play は，負荷生成プロセスの数が増えるにしたがって，フレームレートが落ちているのが分かる．リアルタイムスケジューリングされる mpeg_play は，負荷生成プロセスを多数起動させても，フレームレートの低下は見られなかった．

通常の mpeg_play のフレームレートが著しく低下したのは，Linux 付属のスケジューラにより，mpeg_play プロセスも負荷ぷうろ背巣も公平に CPU 時間が割り当てられるからである．しかし，リアルタイムスケジューリングされる mpeg_play は，リアルタイムプロセスが常に他の通常プロセスよりも優先して処理され，且つ通常プロセスの処理時間，実行数などの影響はいっさい受けないため，フレームレートの低下が見られない．

上記より，DKCS Core として追加した RM スケジューラモジュールが，その機能を果たしていることが確認できた．

5.2 通常スケジューラのオーバヘッド

DKCS をカーネルに実装したことによる，オーバヘッドの増大が懸念されるため，DKCS を実装していない通常スケジューラと，DKCS を実装した通常スケジューラの，起動時間のオーバヘッドを測定した．実験結果は表 5.2 に示す．時間を計測した範囲は，スケジューラ関数が呼び出されて終了するまでの時間を，タスクスイッチが起こらない場合に限って計測した．

表 1　通常スケジューラのオーバヘッド

スケジューラ	起動時間の平均
Default スケジューラ	$6.54\mu sec$
DKCS スケジューラ	$6.89\mu sec$

実験の結果から，DKCS を組み込んだことによる処理時間の増加は $0.35\mu sec$ であり，その差はほとんどないことが分かった．これは，DKCS スケジューラが，関数ポインタで呼び出される用になった以外，スケジューラ関数には大きな変更が加えられていないことによるためである．わずかに増えているオーバヘッドは，以下のことが考えられる．

- スケジューラ別キューの参照
- 関数呼び出し

DKCS では，スケジューラ別に Ready キューを用意しているため，各キューを操作するときは，必ずキューの参照が必要となる．また，スケジューラ関数をポインタとして配列から関数呼び出ししているため，その分のオーバヘッドが通常カーネルのスケジューラよりも大きい．しかし，上記の値は，システムに深刻な影響を与えるほどのオーバヘッドではなく，スケジューリングにも影響はない．

以上のことから，DKCS を組み込んだことによるオーバヘッドは，通常カーネルのスケジューラとほぼ同じ程度に押さえることができた．

6　まとめ

本論文では，スケジューラの動的な変更を可能とするオペレーティングシステムの構成手法について述べた．DKCS の構成では，DKCS インタフェース，DKCS Core について述べ，各モジュールの種類とサービス毎に動的にカーネルにレジス

トされるインタフェースを，ユーザに提供する．ユーザは，DKCSが提供するフレームワークに沿ってDCSK Coreを作成し，任意のスケジューリングアルゴリズムを構築して，動的に追加変更ができる．

また，実際にLinuxシステムにDKCSを組み込むことで，実装可能なDKCSスケジューラのフレームワークを示し，その評価も行った．評価として，リアルタイムスケジューラを組み込んだことで，アプリケーションにリアルタイム性を持たせることができた．また，DKCSを組み込むことにより発生するオーバヘッドは，通常カーネルのスケジューラとほぼ変わらない速度で処理をすることができた．以上より，本論文で提案したDKCSの有効性を示すことができた．

参考文献

[1] microsoft Corporation, *"mediaplayer, "* http://www.microsoft.com/japan/windows, 2002

[2] realnetworks Corporation, *"realplayer, "* http://www.realplay.com/, 2002

[3] Lauri Tischler, *"Loadable Kernel Module HOWTO, "* http://www.linuxdoc.org/HOWTO/Module-HOWTO/

[4] Accetta, M., Baron, R., Golub, D., Rashid, R., Tevanian, A.and Young, M. *"A New Kernel Foundation fro Unix Development, "* 1986 Summer USENIX Conference,pp.93-113(1986)

[5] Bershad, B.N., Chambers, C., Eggers, S., Maeda, C., McNamee, D., Pardyak,P., Savage, S. and Sirer, E.G. *"An Extensible Microkernel for Application-specific Operationg System Service, "* Department of Computer Science and Engineering, University of Washington(1994)

[6] Giorgio C. Buttazzo *"HARD REAL-TIME COMPUTING SYSTEM, "* Kluwer Academic Publishers(1997)

[7] Giorgio C. Buttazzo *"HARD REAL-TIME COMPUTING SYSTEM, "* Kluwer Academic Publishers(1997)