Vol.2012-OS-122 No.18
2012/8/2

コア別ページ回収によるマルチコアシステムの安定性改善

青木英郎[†1]　長井昭裕[†1]　関山友輝[†1]　大島訓[†1]

マルチコアのサーバにおいて，メモリ回収専用コアの割当てにより，リアルタイム性を必要とするアプリケーションがオーバヘッドの低い安定的なメモリ確保を可能とする手法を述べる．計算機の低価格化にともない，高信頼性、安定性が求められる制御システムでも，PC アーキテクチャを基にした計算機と Linux などの汎用 OS が利用されるようになってきた．汎用 OS のメモリ管理では、プロセスやディスク I/O のページキャッシュに、可能な限りのページを割り当てる．このため、メモリ管理機構は，新たなページ割当て要求に対して、オーバヘッドの高いページ回収処理を実行しないと、要求されたメモリが割り当てられない状況が発生する。本研究報告では，ページ回収の契機となる閾値をコアごとに設定できるようにし，コアごとに異なるポリシでページを回収する．これを利用することで，アプリケーションの実行コアに回収が発生しない低い閾値，ページ回収コアに高い閾値を設定するシステムが設計できる．評価により，コア別のページ回収は，システムのメモリ負荷が高くなった場合にでも，アプリケーションの実行を保証する仕組みとして有効であることを確認した．

An improvement of multi-core system stability using page reclaim on each core

Hideo AOKI[†1]　Akihiro NAGAI[†1]　Tomoki SEKIYAMA[†1]　Satoshi OSHIMA[†1]

This technical report presents a method to ensure that real time applications allocate memory with low overhead memory by assigning memory reclaiming core in multicore server. Nowadays, PC based computers and general purpose operating systems are popularly used for control systems. In general purpose operating systems like Linux, memory manager allocates pages as much as possible for processes and disk I/O cache. Thus, in many cases, new page allocation would cause page reclaiming, which is very high overhead procedure. The method that we propose enables memory manager to have different page reclaiming thresholds for each core. To use this method, system administer can set page reclaiming policy to cores. If the threshold is low, the core is able to use for running real time application. Moreover, page reclaiming core has high threshold. In our evaluation, the method is feasible to insure execution of application against high memory pressure.

1. はじめに

計算機技術の発達とそれにともなう低価格化により，社会インフラシステムに利用する制御用計算機や工場の稼働に利用される産業用計算機の構成が変化している．これらの計算機は，従来，高い信頼性を実現するために専用設計されたハードウェアで構成されていた．近年では，低価格かつ高性能な PC ベースの計算機が利用されるようになってきている．オペレーティングシステム(以下，OS と記す)についても，専用のリアルタイム OS ではなく，オープンソースで開発され高機能化が進む Linux®[a]のような汎用 OS が利用されつつある．現在では，数百ミリから 1 秒程度の時間で周期的に動作するリアルタイム性であれば，PC ベースのハードウェアと Linux で制御システムを構成することが多い．

標準の Linux は，低価格な上，グラフィカルユーザインタフェースや開発環境が入手しやすい．しかし，カーネルの内部は，完全にはリアルタイム制御に対応していない．Linux の適用が増える中，制御システムは，スマートグリッドやスマートシティのように機能の高度化やシステム同士の連携が求められており，扱うデータ量，計算量が増している．加えて，Linux をより厳しいリアルタイム制約が求められるシステムでも適用する需要がある．このため，カーネルの各機能についてリアルタイム性や低レイテンシの実現を考慮する必要がある．

本研究報告は，カーネルの中でもメモリ管理のリアルタイム性・低レイテンシの改善について述べる．我々が提案する改善手法では，2000 年代の中頃から一般的に利用されているマルチコアを活用し，メモリ管理のページ回収機能を担当する専用コアを設ける．これにより，安定性・高信頼性が求められる制御システムにおいて，通常プロセスによる大容量のファイルデータ送受信があっても，一定周期で起動するリアルタイムタスクの動作を保証する，といったことが可能となる．

以下，本研究報告では，2 章で研究の背景を示し，3 章で課題を解決するためのアプローチを述べる．4 章で提案するメモリ管理の実装を述べる．5 章では，アプローチと実装の評価を行う．6 章で関連研究，7 章で今後の課題を示し，最後に 8 章で結論を述べる．

†1 (株)日立製作所
Hitachi, Ltd.

a) Linus Torvalds の米国およびその他の国における商標あるいは登録商標．

2. 背景

2.1 汎用 OS における重要アプリケーションの実行保証

高性能化および高機能化が進む汎用 OS において，重要なアプリケーションのリアルタイム性の保証や処理遅延を回避する方法として，従来は，アプリケーションに以下のカスタマイズを行ってきた．

カスタマイズ 1: プロセスの実行優先度を上げる

重要なアプリケーションは，通常のプロセスよりも高い優先度を設定し， CPU が多く割り当てられるようにする．Linux の場合，リアルタイム制御が必要なプロセスは，スケジューリングクラスとして SCHED_FIFO, SCHED_RR リアルタイムプロセス用の高い優先度を使用する．

カスタマイズ 2: 使用メモリのスワップアウト回避

汎用の計算機は，スワップ領域を有効にして稼働させることが多い．重要なアプリケーションが使用するメモリは，スワップアウトの対象になると，アプリケーションの実行速度低下につながり，目標とするリアルタイム性，レイテンシを満たせない．これを回避するため，アプリケーション側で，自身が利用するメモリをスワップアウトの対象から外すよう設定する。Linux の場合，mlockall システムコールを発行する．

2.2 汎用 OS のメモリ管理

汎用 OS のメモリ管理では，システムに搭載されたメモリを最大限に利用するようにメモリ管理機構が設計される．結果として，メモリ管理機構は、割当て要求に対して，可能な限りページを割り当てる．割当ての多くは，ディスク I/O の結果をメモリ上に保持したページキャッシュとして利用される．

Linux のメモリ管理機構は，多数のプロセッサアーキテクチャおよび UMA (uniform memory access)，NUMA (non-uniform memory access)といったメモリの構成が異なる計算機に対応している．このため，物理メモリを NUMA 計算機用のメモリノードと，メモリノード内の複数のゾーンに分けて管理する．

例えば，Intel®[b] i386 アーキテクチャでは，図 1 のように DMA，NORMAL，HIGHMEM の 3 つのゾーンでメモリを管理する．図 1 は，UMA 計算機のものであるため，複数のゾーンは存在しない．物理メモリの管理単位は，4KiB のページである．

メモリ管理機構は，内部でページ管理に用いる閾値を持っている． Linux では，ゾーンに十分な空きページが存在するかどうかを判定するために使用される 3 つの閾値が存在する．各閾値の意味は，以下の通りである．

b) Intel Corporation の米国またはその他の国における商標あるいは登録商標.

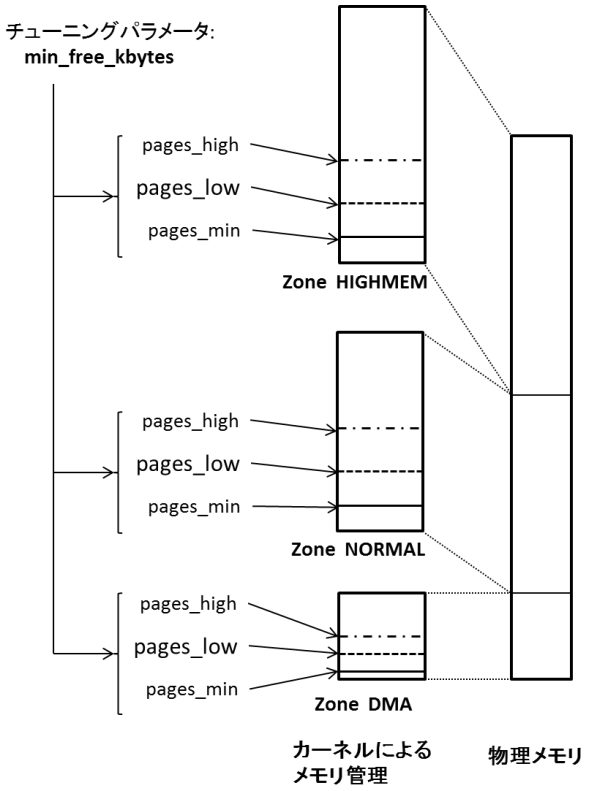

図 1 Linux のメモリ管理概要
Figure 1 Overview of memory management on Linux

(1) **pages_min**
ページ確保の延長上でページ回収を開始する．

(2) **pages_low**
バックグラウンドでのページ回収を開始する．

(3) **pages_high**
開始したページ回収を停止可能とする．

これら 3 つの閾値は，Linux のチューニングパラメータのひとつである min_free_kbytes と搭載メモリ量から，メモリゾーンごとに算出する．

空きページの数が閾値を下回る場合は，ページ回収処理を実行する．ページ回収処理は，割当て済みのページの中から，再利用可能なページ検索し，検索により見つかったページに回収処理を実行し空きページを作り出す．

Linux では，ページ確保要求時に空きページ数が pages_low を下回っていると，kswapd カーネルスレッドを起動し，バックグラウンドでページを回収する．ページキャッシュは，データがディスクに反映されていないダーティな状態の場合，ライトバックを実行してデータを掃き出さなければならない．近年，システムに搭載されるメモリ量が増加しているため，ページの検索をはじめとするページ回収のオーバヘッドは増加傾向にある．

kswapd によるバックグラウンドでの回収が間に合わない場合は，ページ確保処理の延長上でオーバヘッドの高い

回収を実行しなければならない．Linux では，プロセスやページキャッシュに可能な限りページを割り付ける．このため，計算機が長時間動作すると，常に回収をしながら新しいページ確保要求に応える状態となる．

2.3 課題

リアルタイムで動作するアプリケーションの中には，非常にタイミングがシビアなものがある．ネットワーク経由でのデータ送受信のような機能は，実現にあたりカーネル内でページ確保が実行される．このページ確保では，物理連続なページが必要となる場合もある．しかし，Linux は，リアルタイムアプリケーションの実行が保証できるようには，メモリ管理機構が実装されていない．タイミングなシビアなアプリケーションの実行を妨げない仕組みが必要となる。

3. コア別ページ回収によるページ割当て保証

現在，普及している計算機では，マルチコアを有するプロセッサが搭載されている．我々は，このマルチコアを利用し，コア別にページを回収して課題を解決する．図 2 にその概念を示す．

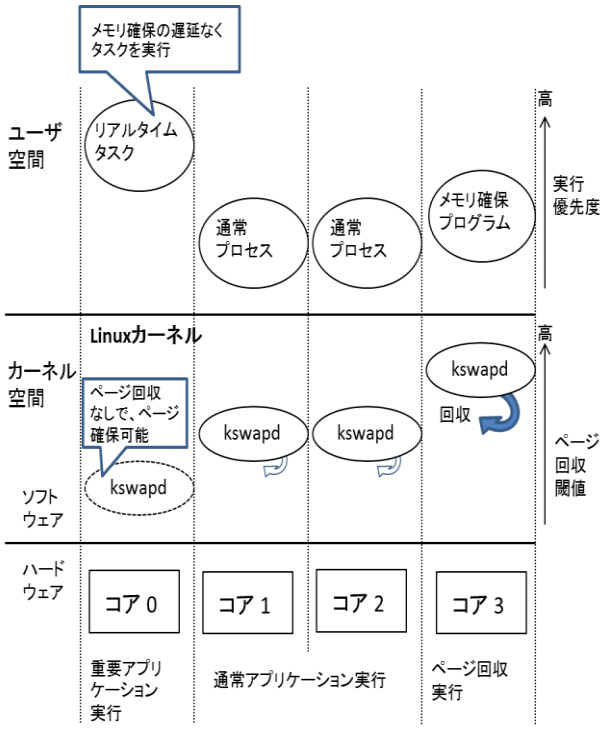

図 2 コアごとのページ回収の概要
Figure 2 Overview of page reclaiming on each core

本解決方法では，OS，システム，アプリケーションのそれぞれで次の対応を行う．

3.1 OS の対応

OS では，カーネルのメモリ管理機構に，マルチコアに対応したページ管理を加える．一般には，メモリ管理機構の内部で一意に保持しているページ回収閾値を，コアごとに分けて管理する．コアごとに分けることで，プロセスの実行のためにページ回収を行わないコア，ページ回収を行う専用のコアといった使い分けが可能となる．

3.2 システム側の対応

前節で示したメモリ管理機構を使うために，システムとして，コアの使用ポリシを定める．まず，複数あるコアの中から，重要アプリケーションの実行コアと，ページ回収の実行コアを決める．つぎに，リアルタイムアプリケーションを実行するコアについては，ページ回収閾値を，回収が起きない値に設定する．ページ回収の実行コアにおいては，ページ回収閾値を，頻繁に回収が起こる値に設定する．図 2 の例では，重要アプリケーションの実行をコア 0，ページ回収の実行をコア 3 で行う．さらに，ページ回収の実行コアでは，ページ回収を誘発する事象を定期的に起こすアプリケーションを動作させる．

3.3 アプリケーション側の対応

リアルタイム性や低レイテンシが必要となるアプリケーションは，2.1 節で述べたカスタマイズに加えて，以下のカスタマイズを実行する．

カスタマイズ 3: アプリケーションの実行コアを固定する
OS が提供するアフィニティ機能などを利用して，アプリケーションが動作するコアを，重要アプリケーション実行コアに固定する．これを実現するために，システム設計として，アプリケーションの開発者，実行者には，アプリケーションが利用可能なコア ID が事前に通知されることを前提とする．

本解決方法では，上に示した OS のメモリ管理機構，システム側の設定およびアプリケーションのカスタマイズの組合せにより，汎用の OS において，アプリケーションの実行性能を保証する．

4. 実装

我々は，前章で述べた解決策を Linux に実装した．本章では，3.1 節に示したコアごとのページ回収を実現するために，カーネルに加えた変更について述べる．なお，実装のベースとした Linux カーネルは，2.6.12 である．古いカーネルであるが，同様の実装は最新のカーネルでも可能で

ある．

4.1 閾値の管理インタフェース追加

メモリ管理に関するチューニングパラメータが配置される擬似ファイルシステム/proc/sys/vm 以下に, コア別にページ回収の閾値を設定するインタフェースを追加した．まず，ディレクトリとして precpu_free_mem を作成した．ディレクトリの下には，コアごとの回収閾値を格納する cpu?ファイル(“?”の部分は，コアの番号) を作成した．

コアごとのファイルには，以下に示す 3 つのパラメータが格納される．

(a) min_free_kbytes
当該コアに設定する pages_min の値を指定する．単位は，キロバイト．

(b) pages_low_ratio
当該コアに設定する pages_low を，pages_min に対する百分率で指定する．

(c) pages_high_ratio
当該コアに設定する pages_high を，pages_min に対する百分率で指定する．

既存のチューニングパラメータ min_free_kbytes(図 1 参照)と新規に追加した percpu_free_mem との関係は，次のように定義した．min_free_kbytes が変更された場合は，すべてのコアに同一の変更を設定したとみなす．(a)には変更された値が入り，(b)(c)には，元々のカーネルで使用されている 125, 150 がセットされる．percpu_free_mem が変更された場合は，各コアの pages_min の和を min_free_kbytes に代入する．

4.2 kswapd の変更

標準の Linux カーネル 2.6.12 は，kswapd をメモリノードごとに生成する．本研究では，これをコアごとに生成できるように変更した．

kswapd の動作に関連するデータ構造は，ノードの構造体 pglist_data に，タスク情報へのポインタや wait queue が格納されている．従来は，1 つの kswapd を動作させるために 1 つしか保存できない．我々は，これらのデータ構造をシステムに搭載されるコア数分の領域が確保できるように拡張した．

kswapd は，メモリゾーンの初期化時に，コアごとに生成する．はじめに，wait queue を初期化する．次に，kswapd カーネルスレッドの生成する．生成に成功すると，カーネルスレッドは，実行コアを固定する．

kswapd の呼出しには，標準の Linux カーネルと同様に，wakeup_kswapd 関数を使用する．関数内では，ページ回収の閾値 pages_low を参照する．このとき，4.1 節に述べたインタフェースから算出された，実行コア番号に対応する pages_low を用いる．実装では，このために，実行コアごとの閾値を取得する関数を定義した．空きページ数が pages_low を下回っている場合は, 実行コアに対応する wait queue を用いて，実行コア用の kswapd を起こす．

kswapd が起床すると, 動作しているコアに設定された閾値と空きページ数を比較する．空きページ数が閾値に満たない場合は，ページを回収する．回収のアルゴリズムは，通常の Linux と同様のものを用いた．

4.3 統計情報およびトレースポイントの追加

システム管理者がコアごとの回収実績を容易に確認できるよう, /proc ファイルシステムに統計情報を格納するファイルを追加した．これは，既存の統計情報ファイル /proc/vmstat の内容を部分的にコアごとに出力したものである．

また，プロトタイプ実装では，デバッグや詳細な動作把握を目的として，Linux 向けのトレーサ実装のひとつである LKST[1]用のトレースポイントを追加した．

5. 評価

本研究報告のメモリ管理方式の有効性を確認するために，評価を実施した．メモリ確保の遅延が許されないアプリケーションが，システムのメモリ負荷が上昇している最中にメモリ確保を実行する状況を模擬し, 効果を測定した．

評価に利用した計算機は，プロセッサとして Intel Core®[c] 2 Duo E8400 (2 コア)，メモリを 4GiB 搭載する．ベース OS として，i386 アーキテクチャ向け Red Hat®[d] Enterprise Linux 4.8 をインストールした．ただし，カーネルは，4 章で述べたメモリ管理方式を実装した Linux カーネル 2.6.12 に入れ替えた．このカーネルは，メモリ管理方式の実装にあたり，閾値の設定を含め 853 行の変更を加えている．

次に評価の方法について述べる．本評価では，コア 1 を重要アプリケーションが動作するコア、コア 0 をメモリ管理用のコアとする. メモリ負荷が高い状況を作り出すため, 測定前に，以下に示す 2 つの負荷をかけた．

- 256MiB のファイル I/O を発行する負荷生成プロセスを実行
- 測定対象とするメモリゾーン(zone NORMAL)の空きページ数が，コア 0 のページ回収開始閾値 pages_low+96 となるまで, ページを確保するカーネルモジュールをロード

測定前の負荷生成後，それぞれのコアに測定時メモリ負荷を生成するカーネルモジュールをロードする．このカーネルモジュールは，実行コア上で 512 ページ確保する．測

c) Intel Corporation の米国またはその他の国における商標あるいは登録商標.
d) Red Hat の米国またはその他の国における商標あるいは登録商標.

定結果の確認として，統計情報ファイルにより，kswapd の起動回数、kswapd がページ管理リストを走査したページ数，走査の結果，実際に回収したページ数を抽出する．

評価は，上に述べた測定を，メモリ管理方式を利用しない場合(percpu_free_mem チューニングなし)と，利用した場合(perfcpu_free_mem チューニングあり)で比較することで行った．各評価における percpu_free_mem の値を表 1 に示す．

表 1 各評価におけるチューニングパラメータの値
Table 1 Values of tuning parameters on each evaluation

チューニングパラメータの値 (percpu_free_mem)	利用なし		利用あり	
	コア 0	コア 1	コア 0	コア 1
pages_min の値 (min_free_kbytes)	3831	3831	12288	256
pages_low の比率 (page_low_ratio)	125	125	240	3300
pages_high の比率 (pages_high_ratio)	150	150	400	4000

測定結果を表 2 に示す. 測定時にロードしたカーネルモジュールがコア 1 で確保するページは，重要アプリケーションの動作に必要なメモリ確保に相当する．結果より，コア別のページ回収を利用した場合，コア 1 では，kswapd が動作していないだけでなく、ページ回収も実行されていない．このため，重要アプリケーションは，メモリ負荷の影響を受けることなく動作したといえる．一方で，コア別のページ回収を利用しない設定で動作させた場合は，重要アプリケーションが動作するコアで kswapd によるメモリ回収が発生している．

以上の評価により，本研究報告で提案するコア別ページ回収を用いたページ割当ては，メモリ負荷が発生した場合にでも，アプリケーションの実行を保証する仕組みとして有効であるといえる．

表 2 測定結果
Table 2 A test result

項目	利用なし		利用あり	
	コア 0	コア 1	コア 0	コア 1
kswapd 起動回数	11	19	87	0
ページ走査回数	330	561	3267	0
回収したページ数	319	544	2847	0

6. 関連研究

アプリケーションの実行に関して性能を保証する仕組みは，これまでも検討されている。しかし，汎用の OS において性能を保証するためには，本研究で示したような仕組みが必要となる．

応答性能が求められるアプリケーションでは，メモリを常駐させる対策が取られる．Linux では、これを実現するシステムコールとして mlockall が用意されており，リアルタイムシステム向けのプログラミングで利用されている[2]．しかし，mlockall システムコールだけでは，アプリケーションが利用する OS サービスで必要となるメモリまでは予約されない．このため，アプリケーションの動作性能を完全に保証することはできない．

Mach[3][4]のようなマイクロカーネル方式の OS では，アプリケーション専用のメモリを管理する外部ページャをサーバとして設けることができる．これにより，リアルタイムタスクが利用するメモリ資源はディスクに退避されない．また，I/O に利用するページキャッシュを周期的に先読みする I/O サーバが存在し，OS のサービスに利用されるメモリ確保を保証できる．しかし，外部ページャや I/O サーバを用意する計算機環境および開発予算を確保できるケースは、限られる．

OS におけるマルチコア活用は，アプリケーションがコアの増加に即した性能向上を得ための OS の構成に関して研究が行われている．マルチコアでは，コア間でのキャッシュの参照，一貫性の制御にコストがかかる．このため，コアが増えると共有データを大量に持つカーネルのオーバヘッドが増える．Corey[5]は，アプリケーションがコア間でのデータ共有を制御できるプリミティブによりスケーラビリティを確保する．McRT[6]では，OS は一部のコアのみを使用し，残りのコアには軽量なスレッドスケジューラが動作する．アプリケーションは，スレッドスケジューラを直接利用することで OS のオーバヘッドを回避する．Barrelfish[7]は，すべてのコアに軽量なカーネルを配置する分散マイクロカーネルの構成をとり，データ共有をメッセージパッシングとすることで，スケーラビリティを出している．Linux においても，カーネルのデータ構造を見直すことで，48 コアまでスケーラビリティが確保できるという報告もある[8]．本研究報告では，既存の研究のアプローチとは異なり，アプリケーションのリアルタイム性，低レイテンシを，汎用な OS 上で保証するために，コアを利用している．

マルチコアを活用して，アプリケーションの実行効率を向上させる別の研究として，FlexSC[9]がある．FlexSC では，システムコールをまとめて発行し，一度に処理させることでシステムコールの実行オーバヘッドを削減する．また，システムコールを専用に実行するコアを定義して，システムコールの処理効率を上げることができる．システムコールの一括実行は，ウェブサーバなどのサーバアプリケーションでは有効だと考えられる．しかし、リアルタイム処理

が要求されるアプリケーションの場合、一括で実行するとリアルタイム制約が守れない可能性がある．リアルタイム処理を目的とした場合は，本研究報告で示したコアの活用は，有効な手段のひとつであると考える．

7. 今後の課題

本研究報告で述べた改善方式には，以下の課題がある．

(1) メニーコア化へ対応したインタフェース

マイクロプロセッサのマルチコア化は，年々進んでいる．現在、数十コアを搭載したプロセッサが試作されている．今後は，数百コアを搭載するメニープロセッサが登場する可能性がある．このようなメニーコアプロセッサに対し，本研究で提案する機構を適用するために，閾値が簡単に設定できるインタフェースを用意する必要がある．

(2) kswapd 起動の自動化

システム設計を容易にするひとつの手段として，ページを回収するコアでの回収処理を自動化する．このためには，カーネル内で kswapd を定期的に起床させる，などの改良を行う．

(3) ホットプラグ，ホットリムーブへの対応

本研究報告の評価として試作したプロトタイプは，CPU のホットプラグとホットリムーブ機能を無効にして実装されている．エンタープライズの用途では，稼働時に致命的なエラーがあったコアを切り離すホットリムーブ，負荷の増大や部品交換のためにコアを追加するホットプラグを使用することがある。本研究で提案した機構を Linux の標準機能とするためには，ホットプラグおよびホットリムーブへの対応が必要であると考える．

8. おわりに

本研究報告では，マルチコアのサーバにおいて，メモリ回収専用コアを割り当てることで，メモリ確保要求に対してオーバヘッドがない安定なシステムを構築可能にする技術を提案した．

参考文献

1) Linux Kernel State Tracer (LKST), http://lkst.sourceforge.jp/
2) Love, R.: Linux System Programming, O’Reilly (2007).
3) Rashid, R., Julin, D. et al.: Mach: A System Software kernel, Proceedings of the 34th Computer Society International Conference (1989).
4) Tokuda, H., Nakajima T. et al.: Real-Time Marh: Towards Predictable Real-Time Systems, Proceedings of the USENIX 1990 Mach Workshop (1990).
5) Boyd-Wickizer, S., Chen, H. et al.: Corey: an operating system for many cores, Proceedings of the 8th USENIX conference an Operating systems design and implementation (2008).
6) Saha, B., Adl-Tabatabai, A. et al.: Enabling scalability and performance in a large scale CMP environment, Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems (2007).
7) Baumann, A., Barham, P. et al.: The multikernel: a new OS architecture for scalable multicore systems, Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (2009).
8) Boyd-Wickizer, S., Clements, A. el al.: An Analysis of Linux Scalability to Many Cores, Proceedings of the 9th USENIX conference on Operating Systems design and implementation (2010).
9) Soares, L. and Stumm, M.: FexSC: flexible system call scheduling with exception-less system calls, Proceedings of the 9th USENIX conference on Operating Systems design and implementation (2010).