Vol.2015-OS-135 No.15
Vol.2015-EMB-39 No.15
2015/11/24

OS資源をコア毎に管理する*Tender*の性能評価

堀井 基史[1]　山内 利宏[1]　谷口 秀夫[1]

概要：排他制御オーバヘッドは，コア数の増加にともない増加する．これにより，性能向上率がコア数の増加にともない低下する問題が生じる．そこで，我々は，OS資源をコア毎に管理するOS構造を提案した．このOS構造において，各コアは，自身のコアにより管理されるOS資源を操作し，他のコアにより管理されるOS資源を遠隔手続呼び出しにより操作する．これにより，コア間でのOS資源の共有を防ぎ，排他制御オーバヘッドを削減する．このため，提案したOS構造は，コア数の増加にともなう性能向上率の低下を抑制できる．本稿では，まず提案手法の実現方式について述べる．また，マイクロベンチマークプログラムを用いて提案手法を実現した***Tender*** を評価した結果を述べる．評価では，メモリの確保と解放処理，およびプロセスの生成と削除処理のスループットと性能向上率により提案手法を実現した***Tender***を評価した．

1. はじめに

マルチコアプロセッサに搭載されるコア数の増加にともない，処理の並列性を向上させる必要性が高まっている．アプリケーションプログラム（以降，AP）は，オペレーティングシステム（以降，OS）にOS資源の操作を依頼し，OSがOS資源の操作を行う．このため，処理の並列性を向上させるためには，OSをマルチコアプロセッサに対応させ，OSにおける処理の並列性を向上させることが重要になる．

OSがOS資源の操作を行う際，複数のコアが共有データを同時に参照し，更新しないよう制御（以降，排他制御）する必要がある．これは，複数のコアが共有データを同時に参照し，更新することにより共有データに不整合が生じ，システムに問題が生じることを防ぐためである．排他制御を実現する手法として，共有データへのロックの取得と解放が広く用いられている．しかし，この手法には，コア数の増加にともない排他制御オーバヘッドが増加し，性能向上率が低下する問題がある [1]．このため，コア数の増加にともなう性能向上率の低下を抑制する手法が研究されている [2][3][4].

我々は，分散指向永続オペレーティングシステム***Tender***[5]（以降，***Tender***）において，OS資源をコア毎に管理するOS構造を提案した [6]．提案手法では，OS資源は，コア毎に管理されるため，コア間で共有されない．このため，OS資源のロックを撤廃することが可能になり，排他制御オーバヘッドを削減できる．これにより，コア数の増加にともなう性能向上率の低下を抑制できる．ただし，OS資源をコア毎に管理すると，各コアは，計算機全体の資源を効率的に利用できなくなる．そこで，コア間の遠隔手続呼び出しによるコア間の連携を提案した．

本稿では，OS資源をコア毎に管理するOS構造について述べる．また，メモリの確保と解放処理，およびプロセスの生成と削除処理の性能を評価し，提案手法は，コア数の増加にともなう性能向上率の低下を抑制できることを示す．

2. *Tender*オペレーティングシステム

2.1 資源の分離と独立化

Tender は，OSが制御し，管理する対象を資源と呼び，資源を分離し，独立化している．たとえば，既存OSは，プロセスに識別子を与え，プロセスの状態，プログラム，およびメモリといったプロセスの構成資源をまとめて管理している．一方，***Tender*** は，既存OSのプロセスを資源「プロセス」,「プログラム」,「仮想ユーザ空間」,「仮想空間」,「仮想領域」および「実メモリ」の6つの資源に分離し，これらの各構成資源に資源名と資源識別子を与えることにより，独立した資源として管理している．また，資源名と資源識別子は，資源名管理木により管理される．資源名と資源識別子を**図 1** に示す．資源名は，場所名，種類名，および固有名からなる文字列であり，資源識別子は，資源の場所，種類，および同一種類内の通番を情報として有する数値である．

Tender は，プログラムの構造上，資源の種類ごとの管

[1] 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

ⓒ 2015 Information Processing Society of Japan

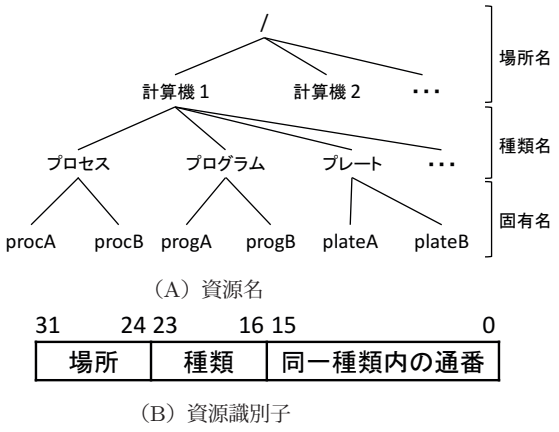

図 1 資源名と資源識別子

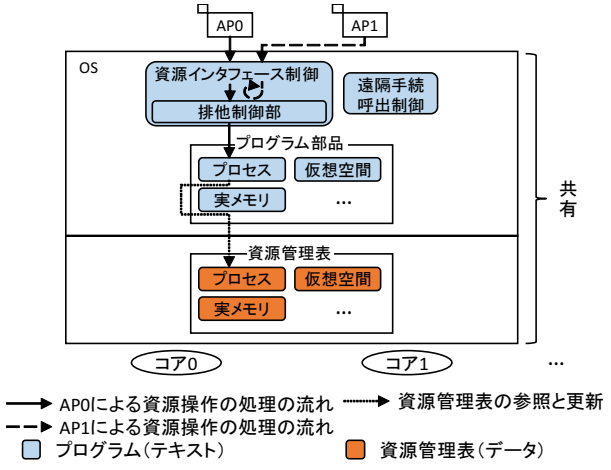

図 2 共有型 ***Tender*** において資源を操作する様子

理表（以降，資源管理表）を個別に用意し，他の種類の資源管理表へのポインタを禁止している．また，資源の種類ごとに管理するプログラム（以降，プログラム部品）を個別に用意し，共通プログラムを排除する．プログラム部品は，特定の処理プログラムである資源インタフェース制御を介して呼び出される．

Tender では，資源の独立化機構に着目したマルチコア対応が実現されている [7]．**図 2** に文献 [7] における ***Tender***（以降，共有型 ***Tender***）において資源を操作する様子を示す．共有型 ***Tender*** では，資源操作のプログラム部品の呼び出しを制御する資源インタフェース制御において操作対象となる資源ごとに排他制御することにより，マルチコア対応における OS 機能の開発工数を削減し，かつ処理の並列性を向上させている．しかし，このマルチコア対応は，資源を管理するための一部の共有データに競合が発生し，コア数の増加にともない性能向上率が低下する問題がある．

2.2 遠隔手続呼出制御

複数の計算機を結んだ分散環境において，複数の計算機

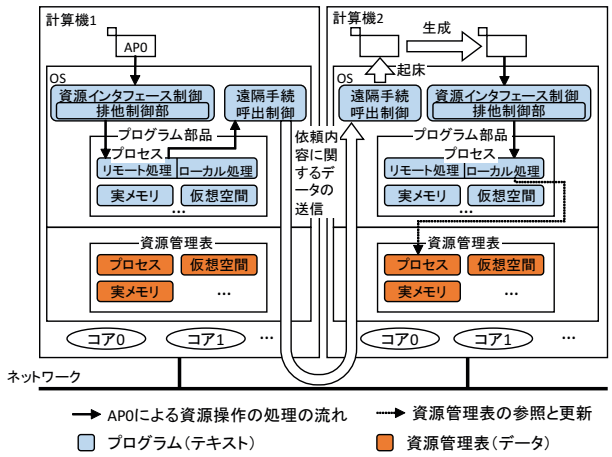

図 3 遠隔手続呼出制御によりリモート計算機の資源を操作する様子

資源を利用した負荷分散を行うために，遠隔手続呼出制御を利用し，各計算機上の計算資源を位置透過に，かつ効率よく操作できる資源操作方式が ***Tender*** に実現されている [8]．これにより，利用者は，資源操作を依頼する計算機（以降，ローカル計算機）により管理される資源と，他の計算機（以降，リモート計算機）により管理される資源を同一のインタフェースで操作できる．

遠隔手続呼出制御によりリモート計算機の資源を操作する様子を**図 3** に示し，以下で説明する．まず，AP0 がリモート計算機の資源の操作を資源インタフェース制御に依頼する．資源インタフェース制御は，操作対象資源に付与された資源名の場所名，あるいは資源識別子の場所の情報を用いて操作対象資源を管理する計算機を判別し，操作対象の資源を管理する計算機がリモート計算機であるか否かを判別する．ここで，操作対象の資源がリモート計算機により管理されている場合，リモート計算機により管理されている資源を操作するためのプログラム部品であるリモート処理プログラムを呼び出す．また，操作対象の資源がローカル計算機により管理されている場合，ローカル計算機により管理されている資源を操作するためのプログラム部品であるローカル処理プログラムを呼び出す．

次に，リモート処理プログラムは，遠隔手続呼出制御にリモート計算機への資源操作を依頼し，遠隔手続呼出制御は，依頼内容に関するデータをリモート計算機に送信する．依頼内容に関するデータを受信したリモート計算機は，依頼を実行するプロセスを生成し，生成されたプロセスが依頼された処理の操作を資源インタフェース制御に依頼する．最後に，資源インタフェース制御はローカル処理プログラムを呼び出して資源を操作した後，操作した結果を依頼元の計算機に返却する．

3. OS 資源をコア毎に管理する OS 構造

3.1 目的

OS のマルチコアプロセッサへの対応において，共有デー

タの不整合によりシステムに問題が生じることを防ぐために，ロックの取得と解放により排他制御する手法が広く利用されている．しかし，この手法には，コア数の増加にともない排他制御オーバヘッドが増加し，これにより性能向上率が低下する問題がある．

そこで，我々は，コア数の増加にともなう性能向上率の低下の抑制を目的とした OS 構造として，OS 資源をコア毎に管理する OS 構造を提案した [6]．提案手法では，OS 資源をコア毎に管理するため，OS 資源はコア間で共有されない．このため，OS 資源のロックの撤廃が可能になり，これにより排他制御オーバヘッドを削減できる．排他制御オーバヘッドを削減することにより，コア数の増加にともなう性能向上率の低下を抑制できる．

3.2 課題

3.2.1 OS 資源のコア毎の管理

OS 資源をコア毎に管理するためには，以下の課題への対処が必要である．

（課題 1）OS 資源を管理するデータ構造のコア毎の確保

OS 資源をコア毎に管理するためには，まず，OS 資源を管理するデータ構造をコア毎に確保する必要がある．たとえば，各コアは，コア数分に分割された物理メモリ空間のうちの 1 つを管理するためのデータ構造を管理する．

（課題 2）他のコアが管理するデータ構造の参照と更新の禁止

次に，OS 資源を操作するプログラムの呼び出しを制御する必要がある．つまり，他のコアが管理するデータ構造の参照と更新を禁止し，自身のコアが管理するデータ構造のみを参照し，更新するよう制御する必要がある．

3.2.2 コア間の連携

OS 資源をコア毎に管理すると，各コアにより操作される OS 資源は，1 つのコアにより管理されている資源に制限されるため，各コアは，計算機全体の資源を効率的に利用できない．このため，以下の課題への対処が必要である．

（課題 3）コア間の連携

各コアが計算機全体の資源を効率的に利用するためには，コア間で資源操作を依頼し，連携する機能が必要である．

3.3 *Tender* における実現方式

（課題 1）と（課題 2）に対して，***Tender*** の特徴である資源の分離と独立化に着目して対処する．

（対処 1）資源管理表のコア毎の確保

Tender では，資源の種類ごとに資源管理表が存在する．さらに，資源管理表は，資源インタフェース制御により管理されている．このため，資源管理表をコア毎に確保し，資源インタフェース制御に登録することにより，（課題 1）「OS 資源を管理するデータ構造のコア毎の確保」に対処できる．

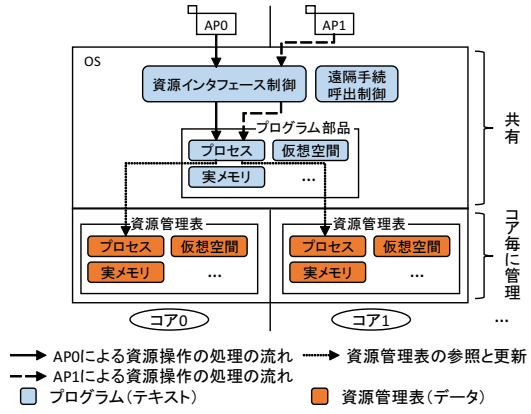

図 4 個別型 ***Tender*** において資源を操作する様子

（対処 2）資源インタフェース制御への資源管理表の登録

既存 OS では，OS 資源を管理するデータ構造は，様々なプログラムから参照され，更新される．このため，自身のコアにより管理されるデータ構造のみ参照し，更新するための修正に要する工数が大きい．一方，***Tender*** では，資源インタフェース制御が資源を操作するプログラムの呼び出しを管理している．このため，資源インタフェース制御がプログラム部品を呼び出す際，資源の操作を要求したコアが確保した資源管理表を引数として渡すことで，（課題 2）「 他のコアが管理するデータ構造の参照と更新の禁止」に対処できる．***Tender*** では，資源インタフェース制御を修正すれば，資源管理表の参照と更新を行うプログラム部品の呼び出しを制御できるため，修正に要する工数が少ない．

さらに，分散指向 OS である ***Tender*** には，計算機間の遠隔手続呼出制御が実現されている．これを用いて以下の対処を行う．

（対処 3）計算機間の遠隔手続呼出制御をコア間に拡張

これにより，（課題 3）「コア間の連携」に対処できる．各コアは，コア間の遠隔手続呼び出しを利用して連携することにより，計算機全体の資源を操作できる．さらに，遠隔手続呼び出しにより，コア毎に管理される OS 資源を同じインタフェースで操作することが可能になる．このため，アプリケーションプログラムの開発者は，複数のコアに分散している OS 資源をそれが複数のコアに分散しているということを意識せずに操作できる．これにより，アプリケーションプログラムの開発者への負担を軽減できる．コア間の遠隔手続呼び出しの設計と実装は，残された課題とする．

図 4 に上記の対処を行った OS 資源をコア毎に管理する ***Tender***（以降，個別型 ***Tender***）において資源を操作する様子を示し，以下で説明する．各コアは，プロセス，仮想空間，実メモリなどの資源を管理するための資源管理表を管理する．コア 0 上で走行する AP0 は，資源インタフェース制御にプロセスの操作を依頼する．この際，参照

ⓒ 2015 Information Processing Society of Japan

Vol.2015-OS-135 No.15
Vol.2015-EMB-39 No.15
2015/11/24

され，更新される資源管理表は，コア 0 により管理されるプロセスの資源管理表である．一方，コア 1 上で走行する AP1 が資源インタフェース制御にプロセスの操作を依頼する際に参照され，更新される資源管理表は，コア 1 により管理されるプロセスの資源管理表である．

4. 評価

4.1 評価方法

マイクロベンチマークプログラムを使用し，個別型 ***Tender*** と共有型 ***Tender*** の処理の並列性を評価した．評価環境を**表 1** に示す．

マイクロベンチマークプログラムは，評価対象の処理を行うプロセスを各コアに 1 つずつ配置し，配置されたプロセスが任意の回数だけ評価対象の処理を行う．

評価観点は，コア数を 1 コアから 16 コアまで変化させた場合のスループットと 1 つのコアあたりの性能向上比であり，個別型 ***Tender*** と共有型 ***Tender*** についてこれらを比較した．スループットは，単位時間（ms）あたりの評価対象の処理の実行回数を示す．性能向上比は，1 コアを基準としてコア数を変化させた場合の性能向上率を示す．

以下に評価対象の処理について述べる．

(1)メモリの確保と解放処理

4KB のメモリの確保と解放の処理を繰り返す評価用プロセスを各コアに配置し，実行する．***Tender*** は，資源「仮想領域」と「仮想ユーザ空間」のカーネルコールにより，メモリの確保と解放を実現する．資源「仮想領域」は，実メモリあるいは外部記憶装置のデータ格納域情報を仮想化した資源である．また，資源「仮想ユーザ空間」は，仮想アドレスの空間である資源「仮想空間」に資源「仮想領域」を貼り付けることにより生成される．ここで，「貼り付ける」とは，仮想アドレスと実アドレスに対応付けることである．これにより，メモリの確保処理を行う．また，仮想アドレスと実アドレスの対応付けを解除することにより，メモリの解放を行う．

評価では，メモリの確保と解放の処理を 3,000 回繰り返して行い，開始から 1,000 回目までの処理と 2,001 回目から 3,000 回目までの処理を除いた 1,000 回の処理により，スループットと性能向上比を評価した．

(2)プロセスの生成と削除処理

子プロセスの生成と削除の処理を繰り返す評価用プロセスを各コアに配置し，実行する．***Tender*** は，資源「プロセス」のカーネルコールにより，子プロセスの生成を実現する．生成する子プロセスは，テキスト部が 1,000B，データ部が 220B，BSS 部が 8B のプログラムである．なお，ディスクへのアクセスによる評価への影響を防ぐため，子プロセスのロードモジュールをカーネルに組み込んでいる．子プロセスの生成の際は，ディスクにアクセスせず，カーネルに組み込んだロードモジュールからプロセスを生成する．

表 1 評価環境

OS	個別型 ***Tender***，共有型 ***Tender***
CPU	Intel Xeon CPU E5-2665 @ 2.4GHz (物理 8 コア，2 ソケット)
RAM	65,536MB（64GB）

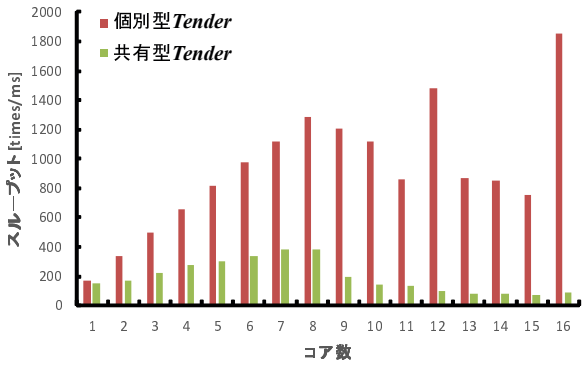

図 5 メモリの確保と解放処理におけるスループット

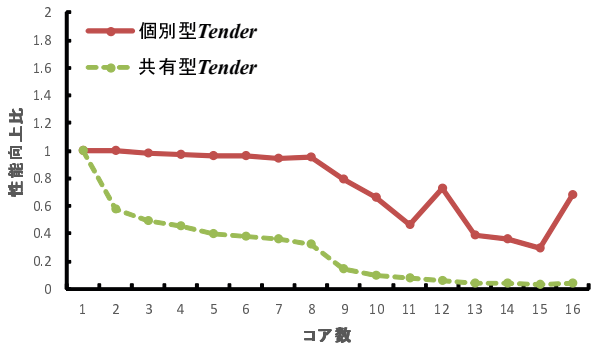

図 6 メモリの確保と解放処理における 1 つのコアあたりの性能向上比

評価では，子プロセスの生成と削除の処理を 300 回繰り返して行い，開始から 100 回目までの処理と 201 回目から 300 回目までの処理を除いた 100 回の処理により，スループットと性能向上比を評価した．

4.2 メモリの確保と解放処理

メモリの確保と解放処理におけるスループットを**図 5** に示す．個別型 ***Tender*** と共有型 ***Tender*** のスループットを比較すると，すべてのコア数の場合において個別型 ***Tender*** のスループットの方が高い．これは，個別型 ***Tender*** は，資源のロックの取得と解放を行わないため，排他制御オーバヘッドが削減されているためである．

また，個別型 ***Tender*** と共有型 ***Tender*** のどちらもコア数が 8 より多い場合のスループットは，減少傾向にある．今回評価に用いた計算機は，8 コアを搭載したプロセッサを 2 基搭載しており，プロセッサ間のキャッシュの一貫性制御によりスループットが低下していると考えている．

ⓒ 2015 Information Processing Society of Japan

メモリの確保と解放処理における 1 つのコアあたりの性能向上比を**図 6** に示す．共有型 ***Tender*** の性能向上比は，コア数の増加にともない低下している．共有型 ***Tender*** では資源の生成と削除時の資源名管理木の一部の処理において，資源の種類ごとのロックの取得と解放を要する箇所がある．このため，同一種類の資源の生成と削除が多発した場合，資源名管理木において競合が多発し，性能向上率が低下する．ここで，メモリの確保と解放処理の評価では，資源「仮想領域」と「仮想ユーザ空間」の 2 種類の資源の生成と削除を繰り返し行う．このため，資源「仮想領域」と「仮想ユーザ空間」の生成と削除処理において競合が多発し，性能向上率が低下した．

一方，個別型 ***Tender*** において，コア数が 8 の場合までの性能向上比は，ほぼ低下しておらず，理想に近い性能向上率を達成している．つまり，個別型 ***Tender*** は，コア数の増加にともなう性能向上率の低下を抑制できていることがわかる．これは，個別型 ***Tender*** は，メモリ関連の資源の資源管理表に加え，資源名管理木もコア毎に管理するため，共有型 ***Tender*** において生じていた競合が発生しないためである．

4.3 プロセスの生成と削除処理

プロセスの生成と削除処理におけるスループットを**図 7** に示す．プロセスの生成と削除処理においてもメモリの確保と解放処理と同様に，個別型 ***Tender*** のスループットの方が共有型 ***Tender*** のスループットより高い．この理由も同様に，個別型 ***Tender*** は，資源のロックの取得と解放を行わないため，排他制御オーバヘッドが削減されているためである．

プロセスの生成と削除処理における 1 つのコアあたりの性能向上比を**図 8** に示す．コア数が 2 の場合から 9 の場合の 1 つのコアあたりの性能向上比は，1 を超えている．この原因として，すべてのコアで共有しているキャッシュによる影響を考えている．

また，共有型 ***Tender*** の 1 つのコアあたりの性能向上比は，コア数の増加にともない低下していることがわかる．一方，個別型 ***Tender*** の 1 つのコアあたりの性能向上比は，コア数が 11 の場合までほとんど低下していない．このため，共有型 ***Tender*** と比べて，個別型 ***Tender*** は，性能向上率の低下を抑制できていることがわかる．

5. おわりに

まず，コア数の増加にともなう性能向上率の低下の抑制を目的とした提案手法である OS 資源をコア毎に管理する OS 構造を実現するための課題について述べた．次に，***Tender*** における課題への対処を述べた．

評価では，個別型 ***Tender*** と共有型 ***Tender*** の性能を比較した．メモリの確保と解放処理の評価結果では，個別型 ***Tender*** の性能向上比は，コア数が 8 の場合までほぼ低下しておらず，理想に近い性能向上率を達成していることを示した．プロセスの生成と削除処理の評価では，個別型 ***Tender*** の性能向上比は，コア数が 11 の場合までほぼ低下していないことを示した．以上により，提案方式では，コア数の増加にともなう性能向上率の低下を抑制できることを示した．残された課題として，コア間の遠隔手続呼び出しによる連携機能の実現と評価がある．

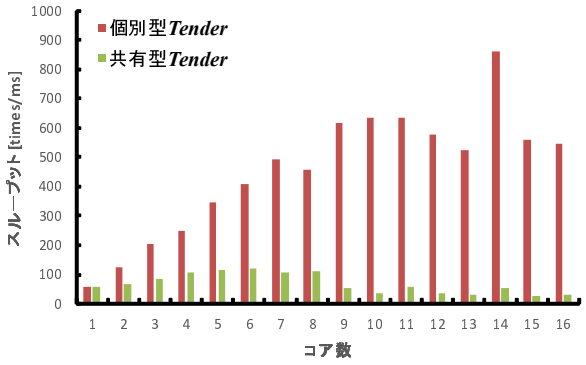

図 7 プロセスの生成と削除処理におけるスループット

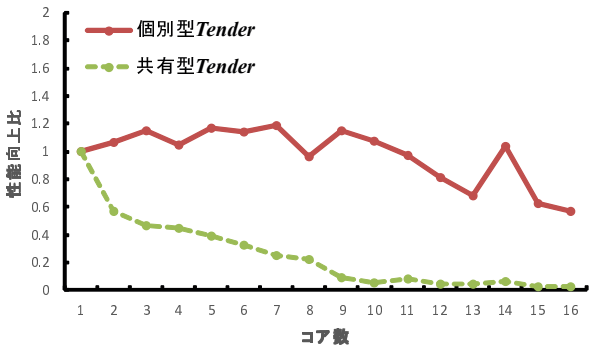

図 8 プロセスの生成と削除処理における 1 つのコアあたりの性能向上比

謝辞 本研究の一部は，科学研究費補助金若手研究（B）（課題番号：25730046）による．

参考文献

[1] Boyd-Wickizer, S., Kaashoek, M. F., Morris, R. and Zeldovich, N.: Non-scalable locks are dangerous, *Proceedings of the Linux Symposium* (2012).

[2] Baumann, A., Barham, P., Dagand, P.-E., Harris, T., Isaacs, R., Peter, S., Roscoe, T., Schüpbach, A. and Singhania, A.: The Multikernel: A New OS Architecture for Scalable Multicore Systems, *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pp. 29–44 (2009).

[3] Boyd-Wickizer, S., Chen, H., Chen, R., Mao, Y., Kaashoek, F., Morris, R., Pesterev, A., Stein, L., Wu, M., Dai, Y., Zhang, Y. and Zhang, Z.: Corey: An Operating System for Many Cores, *8th USENIX Symposium on Operating Systems Design and Implementation*, Vol. 8, pp. 43–57 (2008).

ⓒ 2015 Information Processing Society of Japan

[4] Song, X., Chen, H., Chen, R., Wang, Y. and Zang, B.: A Case for Scaling Applications to Many-core with OS Clustering, *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pp. 61–76 (2011).
[5] 谷口秀夫，青木義則，後藤真孝，村上大介，田端利宏：資源の独立化機構による ***Tender*** オペレーティングシステム，情報処理学会論文誌， Vol. 41, No. 12, pp. 3363–3374 (2000).
[6] 堀井基史，山内利宏，谷口秀夫：***Tender*** におけるコアごとに資源を用意し個別に管理する OS 構造の設計，情報処理学会研究報告，Vol. 2014-OS-131, No. 7, pp. 1–8 (2014).
[7] 山本貴大，山内利宏，谷口秀夫：資源の独立化機構により排他制御を局所化するマルチコア向け ***Tender*** の実現，情報処理学会論文誌 コンピューティングシステム (ACS)，Vol. 7, No. 3, pp. 25–36 (2014).
[8] 石井陽介，谷口秀夫：位置透過な資源操作方式によるプロセス生成機構，情報処理学会論文誌コンピューティングシステム (ACS)， Vol. 44, No. 10, pp. 62–75 (2003).

ⓒ 2015 Information Processing Society of Japan