Vol.2014-DBS-160 No.7
Vol.2014-OS-131 No.7
Vol.2014-EMB-35 No.7
2014/11/18

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

堀井 基史[1]　山内 利宏[1]　谷口 秀夫[1]

概要：マルチコアプロセッサ環境で走行するオペレーティングシステム（以降，OS）は，共有データの不整合を防ぐために排他制御を利用している．しかし，プロセッサに搭載されるコア数の増加にともない，排他制御による待ち時間が増加し，性能向上率が低下する問題がある．そこで，共有データを削減するために，OS が制御し管理する対象（以降，資源）をコアごとに用意し，個別に管理する OS 構造を提案する．これにより，アプリケーションプログラム（以降，AP）による資源操作における排他制御箇所を削減できる．なお，提案方式では，AP は，走行するコアの資源のみを利用することになり，計算機全体の資源を効率的に利用できない．そこで，資源操作をコア間で連携させ，遠隔手続呼出制御により処理を連携できる機能を実現する．本稿では，分散指向永続オペレーティングシステム ***Tender*** に提案方式を実現するための課題と対処を述べる．

1. はじめに

プロセッサの性能向上を図るために，プロセッサに搭載されるコア数は，年々増加している．プロセッサの処理性能を活かすためには，アプリケーションプログラム（以降，AP）だけではなく，オペレーティングシステム（以降，OS）をマルチコアプロセッサに対応させ，処理の並列性を向上させる必要がある．

OS のマルチコアプロセッサへの対応（以降，マルチコア対応）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，排他制御が利用されている．しかし，コア数の増加にともない，排他制御による待ち時間が増加し，性能向上率が低下する問題がある [1]．この問題に対処するには，AP だけではなく，OS 処理の分散が重要な鍵となる．

分散指向永続オペレーティングシステム ***Tender***[2] では，資源の独立化機構に着目したマルチコア対応を行っている [3]．このマルチコア対応は，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，マルチコア環境における 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，性能向上率が低下する問題がある．たとえば，資源を識別するためにすべての資源に付与される文字列と識別子を管理するためのデータと，個々の資源に関するデータを管理するテーブル内の共有データの競合がメモリ処理において多発することが確認されている．

そこで，コアごとに資源を用意し個別に管理する OS 構造を提案する．提案方式は，コアごとに資源を用意し，各コアに割り当てる．また，AP により操作される資源を自プロセスが走行しているコアにより管理される資源に制限し，コア間での資源の共有を防ぐ．これにより，資源を管理するための共有データを削減し，資源操作における排他制御箇所を削減できる．

しかし，この制限により，AP 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，コア間の連携操作を実現し，遠隔手続呼出制御 [2] により処理を連携できる機能を実現する．

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」，資源「プログラム」，資源「仮想ユーザ空間」，資源「仮想空間」，資源「仮想領域」，および資源「実メモリ」の 6 つの資源に分離し，これらの各構成資源に識別子

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

Vol.2014-DBS-160 No.7
Vol.2014-OS-131 No.7
Vol.2014-EMB-35 No.7
2014/11/18

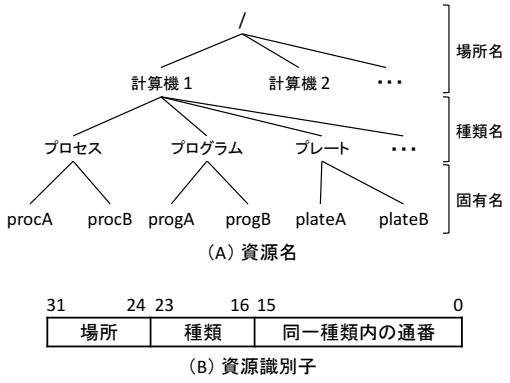


図 1 資源名と資源識別子

を与えることにより，独立した資源として管理している．

Tender は，生成した個々の資源に対して，文字列による名前（以降，資源名）と数字による識別子（以降， 資源識別子）を付与し，資源名管理木により，管理している．**図 1** に資源名と資源識別子を示す．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．資源名は，AP により指定され，資源識別子は，OS により資源の生成時に決定される．

Tender は，資源の分離と独立化を実現するための機構として，表プログラム構造を有する．**図 2** に表プログラム構造を示す．表プログラム構造は，資源の種類ごとの操作インタフェース（以降，プログラム部品）と，個々の資源に関するデータを管理するテーブル（以降，資源管理表）を管理するプログラムポインタ表から構成される．プログラムポインタ表の行要素と列要素は，それぞれ操作する資源の種類と操作内容の種類に対応している．資源の操作内容の種類は，資源の生成（open 系），削除（close 系），入力（read 系），出力（write 系），および制御（control 系）に分類される．

すべてのプログラム部品は，資源インタフェース制御を介して呼び出される．資源インタフェース制御は，プログラムポインタ表における資源の各操作内容の種類に対応する資源操作インタフェースとして，open_rsc()，close_rsc()，read_rsc()，write_rsc()，および control_rsc() を提供している．

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，各計算機上の計算機資源を位置透過に，かつ効率良く操作できる資源操作方式が ***Tender*** に実現されている [4]．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，別の計算機（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．

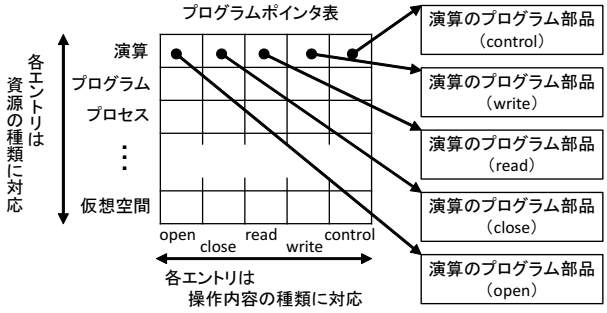


図 2 表プログラム構造

場所	場所名	状態
0	tender	LOCAL_MACHINE
1	tender1	LOCAL_MACHINE
2	tender2	REMOTE_MACHINE
⋮	⋮	⋮

図 3 遠隔手続呼出制御により利用される通信表の例

遠隔手続呼出制御は，場所，場所名，および状態の 3 つの情報の対応を管理する通信表を利用し，操作対象資源を管理する計算機がローカル計算機であるか否かを判別する．遠隔手続呼出制御により利用される通信表の例を**図 3** に示す．通信表における場所と場所名は，それぞれ資源識別子に含まれる場所と，資源名に含まれる場所名を示す．状態の LOCAL_MACHINE は，当該エントリの場所と場所名を付与された計算機が，ローカル計算機であることを示す．また，REMOTE_MACHINE は，当該エントリの場所と場所名を付与された計算機が，リモート計算機であることを示す．

たとえば，資源名/tender2/process/procA の場所名は，tender2 である．図 3 に示した通信表の例では，場所名の欄が tender2 であるエントリは，通信表の上から 3 行目にある．また，このエントリの状態は，REMOTE_MACHINE であるため，この資源名を付与された資源は，リモート計算機により管理されていると判別できる．

なお，場所として 0，または場所名として tender を指定することにより，ローカル計算機の操作対象資源を陽に指定できる．陽にローカル計算機を指定していない場合のみ，操作対象資源の場所の判別処理を行う．

2.3 すべてのコアで資源を共有する OS 構造

マルチコアプロセッサの処理性能を活かすために，OS のマルチコア対応が必要である．OS のマルチコア対応において，開発工数を削減するため，排他制御を OS 機能の処理の入り口に局所化すると，単一の大域的なロックによる排他制御とならざるをえず，処理の並列性が低下する．一方，細粒度な排他制御により，並列性を向上させようとすると，排他制御箇所が増加し，開発工数が増加する．

Vol.2014-DBS-160 No.7
Vol.2014-OS-131 No.7
Vol.2014-EMB-35 No.7
2014/11/18

そこで，***Tender*** において，資源の独立化機構に着目したマルチコア対応が行われている [3]．文献 [3] によるマルチコア対応は，すべてのコアで資源を共有し，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，マルチコア環境での 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．

3. マルチコア向けの OS 構造

3.1 分類の観点

コア数の増加にともなう性能向上率の低下を抑制するマルチコア向け OS 構造を検討するために，***Tender*** を事例とし，マルチコア向けの OS 構造を 4 つの方式に分類する．分類の観点を以下で説明する．

（観点 1）各種類の資源を分割するか否か

各種類の資源をコア数分に分割するか否かで分類する．

（観点 2）資源種別ごとの配置

すべての資源を一箇所に配置するか，資源種別ごとに排他的に配置するか，および分割した各種類の資源を各コアに配置するかで分類する．

（観点 3）資源の操作方式

資源を操作するために必要な操作により分類する．

すべての資源を一箇所に配置する場合，各コアによる資源操作により各種類の資源を操作する場合がある．ただし，この場合，すべての資源は共有されるため，排他制御が必要となる．また，単一のコアによる資源操作と，資源を管理する単一のコアへの資源操作依頼により各種類の資源を操作する場合がある．

資源種別ごとに排他的に配置する場合，各コアは，各コアに配置された資源の操作に加え，操作対象資源を管理するコアに資源操作を依頼することにより，各種類の資源を操作する．

分割した各種類の資源を各コアに配置する場合，各コアは，各コアに配置された分割された資源を操作することにより，各種類の資源を操作する．なお，この場合，資源は共有されないため，排他制御は必要ない．

3.2 分類結果

3.1 節で述べた観点により分類した結果を**表 1** に示し，各方式の利点と欠点を**表 2** に示す．また，各方式によるマルチコア向け ***Tender***（以降，マルチコア ***Tender***）を**図 4** に示す．なお，図 4 は，コア数が 2 個の場合の各方式によるマルチコア ***Tender*** を示している．また，各方式と，各方式の利点と欠点を以下に説明する．

表 1 マルチコア向け OS 構造の分類

分類	各種類の資源を分割するか否か	資源種別ごとの配置	資源の操作方式
共有方式	分割しない	すべての資源を一箇所に配置	各コアによる資源操作（排他制御あり）
集約方式			単一のコアによる資源操作＆資源操作依頼
分散方式		資源種別ごとに排他的に配置	各コアによる資源操作＆資源操作依頼
個別方式	分割する	分割した各種類の資源を各コアに配置	各コアによる資源操作（排他制御なし）

表 2 各方式の利点と欠点

分類	利点	欠点
共有方式	資源操作依頼によるオーバヘッドなし	排他制御によるオーバヘッドあり
集約方式	AP は効率よく走行可能	資源操作依頼が集中する場合あり
分散方式	資源操作依頼を分散可能	資源操作依頼が集中する場合あり
個別方式	排他制御によるオーバヘッドを削減可能	資源操作依頼によるオーバヘッドあり

（方式 1）すべてのコアで資源を共有する OS 構造（以降，共有方式）

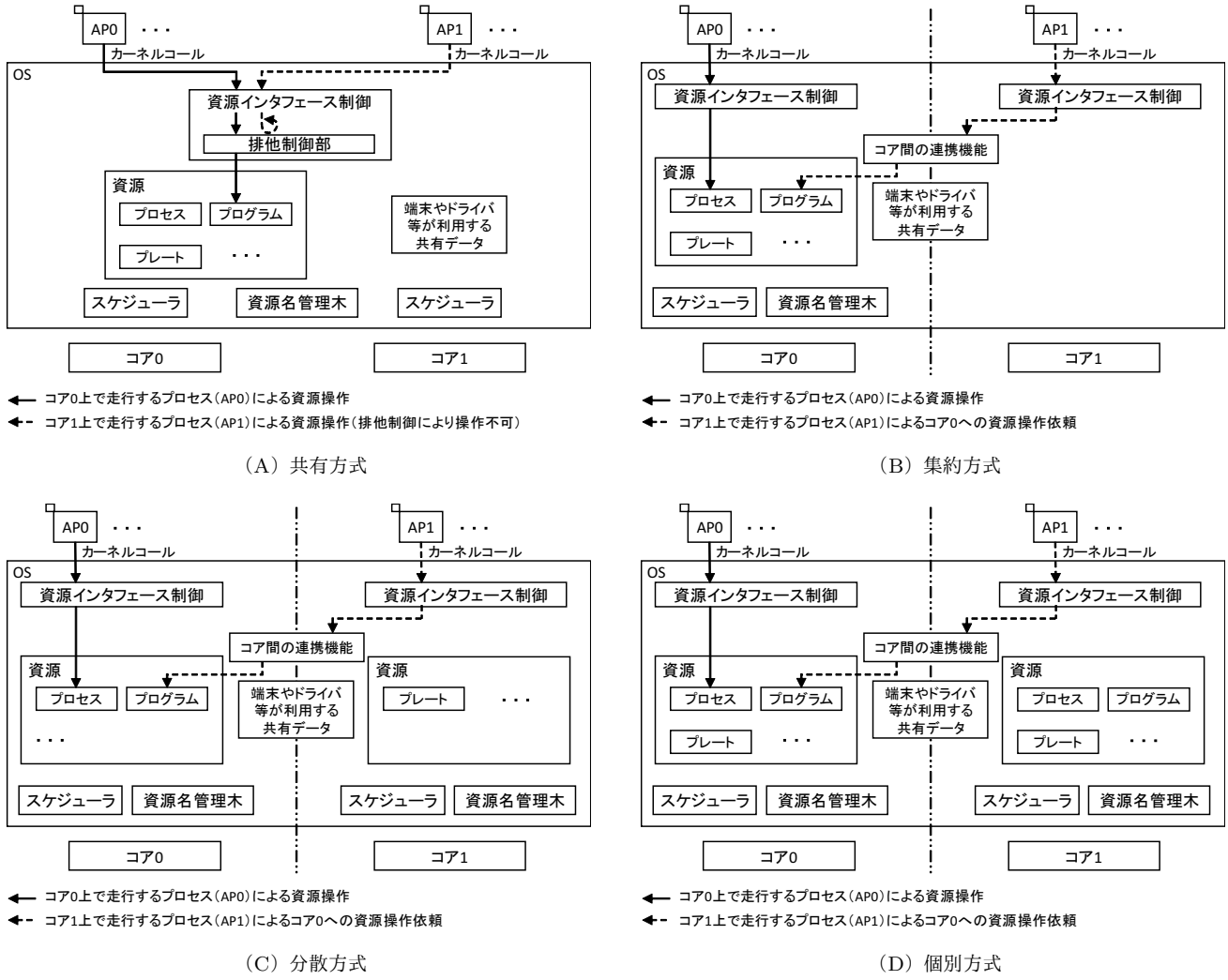
図 4 の（A）に示すように，共有方式は，資源を一箇所に配置し，単一の資源インタフェース制御により呼び出しを管理する．また，すべての資源を共有するため，資源名管理木は，1 つのみ存在する．また，各コア上で走行するプロセスをスケジューリングするために，コアごとにスケジューラが存在する．また，以降の各方式では，資源インタフェース制御により呼び出しを管理されない端末やドライバなどにより利用されるデータは，すべてのコアで共有される．

共有方式は，すべてのコアで資源を共有するため，資源操作を排他制御する必要がある．このため，コア数の増加にともない，排他制御による待ち時間が増加し，オーバヘッドが増加する．一方，資源操作を要求するプロセスが走行するコア（以降，自コア）は，自コア以外のコア（以降，他コア）に資源操作を依頼する必要がない．このため，資源操作依頼によるオーバヘッドは生じない．

文献 [3] は，共有方式によるマルチコア ***Tender*** の実現方式と問題点を示している．具体的には，資源名管理木は，資源の種類ごとに排他制御する必要があり，同一種類の資源の生成と削除が頻繁に行われる場合，資源名管理木の競合が発生し，性能向上率が低下する問題がある．また，資源管理表内の共有データの排他制御により競合が発生し，性能向上率が低下する問題がある．

（方式 2）各種類の資源を単一のコアが管理する OS 構造（以降，集約方式）

図 4 の（B）に示すように，集約方式は，すべての資源を単一のコアが管理する．このため，資源名管理木とスケ



（A）共有方式　（B）集約方式

（C）分散方式　（D）個別方式

図 4　各方式によるマルチコア ***Tender***

ジューラは，すべての資源を管理する単一のコアにのみ存在する．また，資源を管理しないコアは，遠隔手続呼出制御によりコア間の連携操作を実現する機能（以降，コア間の連携機能）を利用し，資源を管理する単一のコアに資源操作を依頼する．コア間の連携機能は，資源インタフェース制御により呼び出されるため，コアごとに資源インタフェース制御が存在する．

集約方式において，資源を管理していないコアは，他コアに資源操作を依頼するため，自コアで資源操作を行わない．このため，資源を管理していないコア上で走行するAP は，カーネル処理により処理を妨害されにくく，キャッシュヒット率の向上や割り込みの削減により処理時間を削減できる．しかし，集約方式は，コア数が増加すると，資源を管理する単一のコアに資源操作依頼が集中するため，資源を管理する単一のコアがボトルネックになる．このため，コア数が増加すると，カーネル処理の並列性が低下し，性能向上率が低下する．

（方式 3）コアごとに異なる種類の資源を管理する OS 構造（以降，分散方式）

図 4 の（C）に示すように，分散方式は，コアごとに異なる種類の資源を管理する．このため，資源名管理木は，コアごとに存在する．また，各コア上で走行するプロセスをスケジューリングするために，コアごとにスケジューラが存在する．各コアは，自コアが管理していない資源を操作するために，コア間の連携機能を利用して他コアに資源操作を依頼する．このため，資源インタフェース制御は，コアごとに存在する．

分散方式は，AP が特定の種類の資源を頻繁に操作しない場合，コアごとに管理している資源の種類が異なるため，資源操作の依頼を分散できる．これにより，カーネル処理の並列性が向上し，性能向上率の低下を抑制できる．一方，AP が特定の種類の資源を頻繁に利用する場合，特定の種類の資源を管理するコアに資源操作の依頼が集中する．このため，カーネル処理の並列性が低下し，性能向上率が低下する．

（方式 4）コアごとに資源を用意し個別に管理する OS 構造（以降，個別方式）

個別方式は，図 4 の（D）に示すように，各種類の分割

された資源を各コアに配置し，管理する．コアごとに資源を管理するため，資源インタフェース制御，資源名管理木，およびスケジューラは，コアごとに存在する．また，計算機全体の資源を効率的に利用するために，コア間の連携機能を利用して他コアに資源操作を依頼する．

個別方式は，他コアにより管理される資源の操作を禁止することにより，コア間での資源の共有を防止し，資源操作における排他制御箇所を削減する．したがって，排他制御によるオーバヘッドを削減し，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．一方，コア間の連携機能により，他コアへの資源操作依頼によるオーバヘッドが生じる．

4. コアごとに資源を用意し個別に管理するOS構造の設計

4.1 設計方針

コア数の増加にともなう性能向上率の低下を抑制するために，カーネル処理の並列性を向上し，かつ特定のコアへの資源操作依頼の集中を防ぐ必要がある．共有方式は，資源操作を排他制御しているため，コア数の増加にともないカーネル処理の並列性が低下する．一方，集約方式，分散方式，および個別方式は，資源操作を排他制御しないため，カーネル処理の並列性の向上が期待できる．

また，集約方式と分散方式は，各種類の資源を操作するために資源操作依頼が必須である．このため，コア数の増加にともない特定のコアに資源操作依頼が集中する．一方，個別方式において，各コアは，各種類の資源を管理しているため，資源操作依頼は必須ではなく，コア間で連携する場合のみ必要となる．このため，個別方式において資源操作依頼が要求される頻度は，集約方式と分散方式と比較すると，低いと考えられる．したがって，資源操作依頼によるオーバヘッドは小さく，また，資源操作依頼の集中を軽減できると考えられる．

以上より，個別方式は，コア数の増加にともなう性能向上率の低下を抑制するOS構造として，各方式の中で最も適していると考えるため，個別方式によるマルチコア ***Tender*** を実現する．

個別方式の設計方針を以下に示し，説明する．

（方針1）資源操作における排他制御箇所の削減

OSのマルチコア対応において，資源の競合を防ぐために排他制御を利用すると，コア数の増加にともない性能向上率が低下する．このため，資源操作における排他制御箇所を削減する必要がある．

（方針2）遠隔手続呼出制御を利用したコア間の連携

個別方式において，APにより操作される資源は，自プロセスが走行しているコアにより管理される資源に制限される．このため，APは，計算機全体の資源を効率的に利用できない．APが計算機全体の資源を操作するためには，他コアに資源操作を依頼する機能が必要である．

そこで，分散OSである ***Tender*** において実現されている遠隔手続呼出制御を利用し，コア間の連携機能を実現することにより，APによる計算機全体の資源の操作を可能にする．

4.2 提案方式

個別方式は，資源インタフェース制御を各コアに配置し，資源インタフェース制御において自コアにより管理される資源のみ呼び出すよう制御する．また，各コアは，自コアで管理する資源の資源名と資源識別子のみを管理するため，他コアとは独立した資源名管理木を保持する．また，各コアは，他コアとは独立したスケジューラを保持し，自コアで管理する資源「プロセス」と関連付いた資源「演算」のみスケジューリングする．なお，資源「演算」とは，プロセッサを割り当てられる程度を資源化したものである．

各コア上で走行するAPは，自コアにより管理される資源を操作し，コア間の連携機能を利用して他コアに資源操作を依頼する．つまり，各コアで資源を共有しない独立したOSが動作しているとみなし，各コア上で動作するOSが連携する．

4.3 課題

（方針1）と（方針2）を満足するための課題を以下で説明する．

（課題1）コアごとに資源を用意し個別に管理するOS構造の実現

（方針1）を満足するためには，コアごとに資源を用意する必要がある．このため，メモリ空間やデバイスなどの資源を分割し，各コアに割り当てる必要がある．また，各コアは，他コアにより管理される資源の操作を禁止されているため，自コアにより管理されるプロセスのみをスケジューリングするようスケジューラを独立化する必要がある．つまり，（方針1）を満足するための課題として，以下の課題が考えられる．

（A）メモリ領域の競合を防止するメモリ管理方式

各コアが同じメモリ領域を同時に参照し，更新すると，競合が生じ，データの不整合が生じるため，システムに問題が起きる．これを防ぐため，メモリ領域の競合を防止する必要がある．ただし，（方針1）を満足するためには，できるだけ排他制御を利用せずに，メモリ領域の競合を防止する必要がある．

（B）デバイスの分割

資源操作における排他制御箇所を削減するためには，外部記憶装置やネットワークインタフェースカード（以降，NIC）などのデバイスの操作を排他制御なしに行う必要がある．デバイスの分割については，今後の課題とする．

（C）スケジューラの独立化

Vol.2014-DBS-160 No.7
Vol.2014-OS-131 No.7
Vol.2014-EMB-35 No.7
2014/11/18

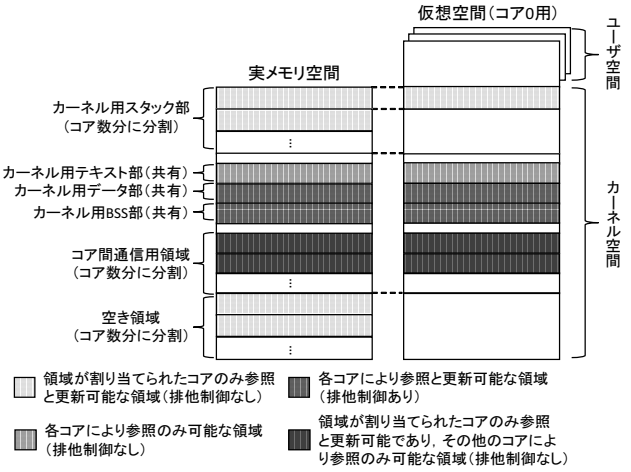


図 5 個別方式によるマルチコア ***Tender*** の仮想空間の構成

個別方式において，各コアは，自コアにより管理される資源のみ操作できる．このため，各コアは，他コアとは独立したスケジューラを保持し，自コアにより管理されるプロセスのみスケジューリングする必要がある．

（課題 2）コア間の連携機能の実現

（方針 2）を満足するためには，操作対象資源を管理するコアを特定し，特定したコアと依頼内容に関する情報を送受信する必要がある．また，他コアにより管理される資源の操作は禁止されているため，資源操作を依頼されたコアが資源を操作する必要がある．つまり，（方針 2）を満足するための課題として以下の課題が考えられる．

（A）操作対象資源を管理するコアの特定

他コアへの資源操作依頼によりコア間で連携するために，操作対象資源を管理するコアを特定する必要がある．

（B）データの送受信方式

コア間で連携するためには，依頼内容に関する情報をコア間で送受信する必要がある．データの送受信方式については，今後の課題とする．

（C）代行処理の実行方式

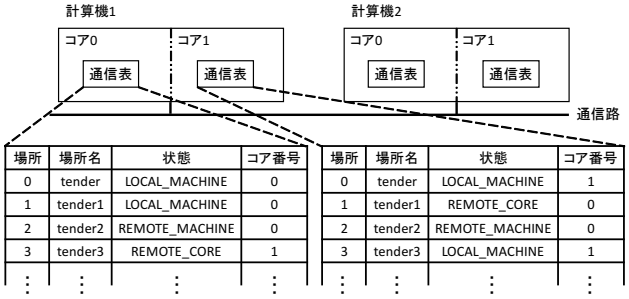
他コアにより管理される資源の操作は禁止されているため，操作対象資源を管理するコアに資源操作の代行を依頼し，代行依頼を受けたコアが代行して資源を操作する必要がある．

4.4 対処

4.3 節で示した課題に対して，以下の対処を行う．

（対処 1-A）メモリ領域の競合を防止するメモリ管理方式

排他制御を利用せずにメモリ領域の競合を防止するために，実メモリ空間をコア数分に分割し，各コアに割り当てる．また，各コアは，自コアに割り当てられた領域からのみメモリ領域を確保するよう制御する．これにより，各コアにより利用できるメモリ領域が重複しないよう制限されるため，メモリ領域の競合を防止できる．



場所	場所名	状態	コア番号
0	tender	LOCAL_MACHINE	0
1	tender1	LOCAL_MACHINE	0
2	tender2	REMOTE_MACHINE	0
3	tender3	REMOTE_CORE	1

場所	場所名	状態	コア番号
0	tender	LOCAL_MACHINE	1
1	tender1	REMOTE_CORE	0
2	tender2	REMOTE_MACHINE	0
3	tender3	LOCAL_MACHINE	1

図 6 拡張した通信表をコアごとに管理する様子

個別方式によるマルチコア ***Tender*** の仮想空間の構成を**図 5** に示す．なお，図 5 では，コア 0 用の仮想空間を示している．図 5 に示すように，カーネル用スタック部と，空き領域は，コア数分に分割され，各コアに割り当てられる．また，各コアは，自コア用の領域のみ参照し，更新する．各種類の資源の資源管理表や資源名管理木などは，各コア用の空き領域から確保される．このため，各種類の資源の資源管理表や資源名管理木などは，他コアにより操作されない．したがって，資源管理表や資源名管理木の操作における排他制御を撤廃可能となり，3 章で述べた共有方式によるマルチコア ***Tender*** の問題点を解決できる．

ただし，カーネル用テキスト部は，一度書き込まれると，更新されることはない．このため，カーネル用テキスト部は，各コアで共有し，各コアは，この領域への参照のみ可能とする．また，カーネル用データ部とカーネル用 BSS 部は，コア数分に分割することが困難であるため，各コアで共有し，排他制御により競合を防止することで，参照と更新を可能にする．

また，コア間の連携機能で利用されるコア間通信用領域は，各コアごとに存在する．各コア用のコア間通信用領域は，自コアのみ参照と更新を可能にし，他コア用のコア間通信用領域は，参照のみ可能とする．

（対処 1-C）スケジューラの独立化

Tender は，資源「演算」と資源「プロセス」を関連付けることにより，資源「演算」に割り当てられたプロセッサ時間を利用して当該演算に割り当てられた資源「プロセス」が走行する．各プロセスは，関連付けられた資源「演算」に割り当てられたプロセッサ時間に基づいてスケジューリングされる．

個別方式は，各コアが自コアに対応する資源「演算」を管理する．そこで，自コアにより管理される資源「演算」は，自コアにより管理される資源「プロセス」にのみ関連付け可能とする．これにより，各コアは，自コアにより管理される資源「プロセス」が関連付けられた資源「演算」のみスケジューリングできる．

（対処 2-A）操作対象資源を管理するコアの特定

操作対象資源を管理するコアを特定するために，各コアは，拡張した通信表を保持する．**図 6** に，各コアが拡張

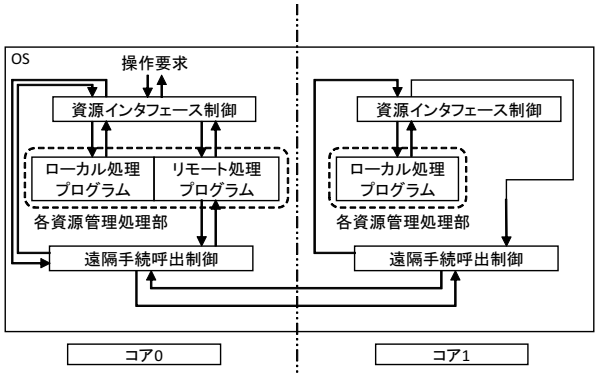


図 7 個別方式おける資源操作処理の様子

した通信表を保持する様子を示す．拡張内容として，まず，通信表の状態にローカル計算機の他コアを示す REMOTE_CORE を追加する．また，通信表のそれぞれのエントリに 1 台の計算機中の各コアに固有の番号を示す「コア番号」を追加する．

他コアに資源操作を依頼する際，拡張した通信表から操作対象資源の場所，または場所名に対応するエントリを探す．エントリが見つかった場合，操作対象資源を管理するコアは，一意に特定できるため，特定したコアに資源操作依頼を行う．

たとえば，図 6 の場合にコア 0 が資源名/tender3/process/procB が付与された資源を操作する場合を考える．資源名/tender3/process/procB の場所名は，tender3 である．このため，コア 0 が管理する通信表の上から 4 番目のエントリが該当する．当該エントリの状態は，REMOTE_CORE であり，コア番号は，1 である．これにより，この資源は，ローカル計算機のコア 1 が管理していると判別できる．

なお，個別方式では，場所として 0，または場所名として tender を指定することで，操作対象資源が自コアにより管理されていることを陽に指定できる．

（対処 2-C）代行処理の実行方式

図 7 に個別方式における資源操作処理の様子を示し，以下で説明する．なお，資源管理処理部とは，資源の各操作内容に対応する各プログラム部品を示す．また，ローカル処理プログラムは，ローカル計算機により管理される資源を操作するためのプログラム部品であり，リモート処理プログラムは，リモート計算機，または他コアにより管理される資源を操作するためのプログラム部品である．

すべての資源操作は，資源インタフェース制御に依頼される．資源インタフェース制御は，陽に自コアを指定していない場合のみ，引数の資源名，または資源識別子を調べて，当該の要求が陽に自コアを指定しているのか否かの判別を行う．一方，陽に自コアを指定している場合，場所の判別処理が不要になり，自コア内における資源操作の処理効率を改善できる．

陽に自コアを指定していない場合は，要求された処理を行う資源管理処理部のリモート処理プログラムを呼び出す．一方，陽に自コアを指定している場合，要求された処理を行う資源管理処理部のローカル処理プログラムを呼び出し，その戻り値を返す．

リモート処理プログラムは，他コアに対して処理を依頼するため，遠隔手続呼出制御を呼び出す．遠隔手続呼出制御は，（対処 2-A）において述べた通信表を利用し，依頼先のコアを特定する．この際，依頼先が自コアである場合は，その処理要求を陽に自コアに指定し直して，再度自コアの資源操作インタフェース制御に処理の依頼を行う．一方，依頼先が他コアである場合は，依頼内容に関するデータを依頼先のコアに送信することで，他コアに対して処理の依頼をする．

他コアは，依頼された処理要求を自コアを陽に指定するように変更し，自コアの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において，当該の要求は，陽に自コアを指定していると判別し，要求された処理を行う資源管理処理部のローカル処理プログラムを呼び出し，その戻り値を返す．この戻り値は，処理依頼とは逆の手順で，依頼元のコアに返される．

4.5 期待される効果

4.1 節の「（方針 1）資源操作における排他制御箇所の削減」により，排他制御による待ち時間を削減できるため，コア数の増加にともなう性能向上率の低下の抑制が期待できる．また，「（対処 1-A）メモリ領域の競合を防止するメモリ管理方式」により，各コアにより利用されるデータを局所化できる．これにより，キャッシュヒット率の向上が期待できる．

また，「（方針 2）遠隔手続呼出制御を利用したコア間の連携」により，ローカル計算機の各コアにより管理される資源と，リモート計算機の各コアにより管理される資源を位置透過に操作できるため，利用者は，操作する資源の場所に応じてインタフェースを変更する必要がない．これにより，利用者への負担の軽減が期待できる．

5. 関連研究

大部分の OS 機能を単一のコアでのみ提供する OS として，AUTOSAR[5] がある．AUTOSAR は，単一のコアでのみ OS 機能を提供するため，資源の競合が発生しない．

特定の OS 機能を実行する複数の専用のコアにより OS 機能を提供し，資源の競合を削減する OS として，Corey[6]，Factored Operating System[7]，および GenerOS[8] がある．たとえば，Corey は，資源の共有を AP により制御できるため，不必要な資源の共有を排除できる．これにより，特定の OS 機能を実行する複数の専用のコアを用意し，資源の競合を削減できる．

各コアが資源の複製を保持する OS として，Barrelfish[9] がある．Barrelfish において，各コアは，各コアが保持する資源の複製を操作するため，資源の競合を防止できる．また，各コアが保持する資源の状態の一貫性は，明示的なコア間通信により保たれる．

分割した資源を複数の AP からなるサービスに割り当てる OS として，Tessellation[10] がある．Tessellation は，資源のサービスへの割り当てと，サービス内の AP への資源の割り当て分離している．サービス内の AP への資源の割り当ては，他のサービスを意識しなくてよいため，AP に特化した資源の割り当てが可能である．

6. おわりに

Tender におけるコアごとに資源を用意し個別に管理する OS 構造の設計について述べた．この方式は，他コアにより管理される資源の操作を禁止することにより，コア間での資源の共有を防止する．これにより，資源操作における排他制御箇所を削減できるため，コア数の増加にともなう性能向上率の低下の抑制が期待できる．また，AP による計算機全体の資源操作を可能にするために，遠隔手続呼出制御を利用したコア間の連携機能を実現する．遠隔手続呼出制御を利用するため，利用者は，操作する資源の場所に応じてインタフェースを変更する必要がない．このため，利用者への負担の軽減が期待できる．

コアごとに資源を用意し個別に管理する OS 構造を実現するために，実メモリ空間をコア数分に分割し，各コアに割り当てる．これにより，各コアにより利用されるデータを局所化し，キャッシュヒット率の向上が期待できる．また，各コアは，各コアにより管理される資源「演算」のみスケジューリングするよう制御する．

また，コア間の連携機能を実現するために，通信表を拡張し，資源名，または資源識別子により操作対象資源を管理するコアのコア番号を特定できるようにする．また，代行処理を実現するために，他コアにおいて，依頼された処理要求を自コアを陽に指定するよう変更する．これにより，他コアの資源インタフェース制御に資源操作を要求し，他コアにおける資源操作の代行処理を実現する．

今後の課題として，デバイスの分割とデータ送受信方式がある．

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），および科学研究費補助金基盤研究（B）（課題番号：24300008）による．

参考文献

[1] Boyd-Wickizer, S., Clements, A. T., Mao, Y., Pesterev, A., Kaashoek, M. F., Morris, R. and Zeldovich, N.: An Analysis of Linux Scalability to Many Cores, *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation*, pp. 1–16 (2010).

[2] 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).

[3] 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).

[4] 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).

[5] Böhm, N., Lohmann, D., Schröder-Preikschat, W. and Erlangen-Nuremberg, F.: A Comparison of Pragmatic Multi-Core Adaptations of the AUTOSAR System, *7th annual workshop on Operating Systems Platforms for Embedded Real-Time applications*, pp. 16–22 (2011).

[6] 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).

[7] Wentzlaff, D. and Agarwal, A.: Factored Operating Systems (fos): The Case for a Scalable Operating System for Multicores, *ACM SIGOPS Operating Systems Review*, Vol. 43, No. 2, pp. 76–85 (2009).

[8] Yuan, Q., Zhao, J., Chen, M. and Sun, N.: GenerOS: An Asymmetric Operating System Kernel for Multi-core Systems, *Parallel & Distributed Processing, 2010 IEEE International Symposium on*, pp. 1–10 (2010).

[9] 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).

[10] Colmenares, J., Eads, G., Hofmeyr, S., Bird, S., Moreto, M., Chou, D., Gluzman, B., Roman, E., Bartolini, D., Mor, N., Asanovic, K. and Kubiatowicz, J.: Tessellation: Refactoring the OS around explicit resource containers with continuous adaptation, *Proceedings of the 50th Annual Design Automation Conference*, pp. 1–10 (2013).