計算機アーキテクチャ 65－2
（１９８７．３．１３）

実時間多重処理環境における共通資源保護方式

青木 道宏　　上森 明　　新谷 廣
ＮＴＴ電気通信研究所

一般に共通資源をアクセスするプロセスは動的に変化するため，ソフトウェアの不法アクセスに起因するエラーの解析は困難である．このため共通資源アクセスにおける保護の導入による，エラーの事前防止，早期検出が必要となる．

本報告では，実時間性を考慮した共通資源へのアクセスをプロセス単位で実現した保護方式を提案する．本方式では，保護に係わるオーバヘッドの増加を①資源管理部における保護情報の管理を含む高速確保／解放機構，②連想メモリ等の専用機構によるアクセス制御機構によって最小限に抑えている．

更に，実時間多重処理環境での評価例により，約５％の性能低下で実現できる見通しを得，本方式の有効性を確認した．

A PROTECTION METHOD FOR A COMMON RESOURCE IN REAL-TIME MULTIPROCESSING ENVIRONMENTS

Michihiro AOKI　Akira UEMORI　Hiroshi SHINTANI
NTT Electrical Communications Laboratories
9-11 Midori-Cho 3 Chome Musasino-Shi,Tokyo 180 Japan

As a common resource can be dynamically allocated to many user processes, it is very difficult to analyze the location of a fault which is caused by illegal access. It is necessary to introduce a protection method when acceding into the common resource .

In this paper, a protection method for common resources that are useful in real-time multiprosessing environments is proposed. It can reduce run-time overhead to maintain the information with regard to a protection and check the accessibility of each process.

In addition, this method is applied to a real-time multiprocessing systems. The result shows that this method can be easily realized and the system performance whould not be degraded by less than 5 persent .

1．まえがき

一般に複数のプロセスから共通的に確保，解放，参照，更新される共通資源に対するアクセスは，時間と共に使用するプロセスが動的に変化するため，ソフトウェアにおける不法アクセスに起因するエラーの検出・解析は困難である．従って，共通資源アクセスにおけるソフトウェアの原因によるエラー防止のためには，アクセス保護の導入が必要となる．

一方，厳しい実時間多重処理性が要求されるプログラム走行環境では，資源の確保／解放が頻繁に発生する．従って，保護のための情報の設定／解除のオーバヘッドおよびアクセス時の保護情報の検索・参照によるオーバヘッドの増加を最小限に抑えることが必要となる．

本報告では，実時間性を考慮した共通資源へのアクセス保護方式を提案する．同時に，実時間多重処理環境での１つの評価例により，本方式の有効性を確認する．

2　プロセス単位での保護の必要性

共通資源に対する不法アクセスが発生した場合の例を図１に示す．図１－（ａ）は，プロセスＡが確保した共通資源のブロック１に対し，アクセス権の無いプロセスＢが不法にアクセスを行った例である．この場合には図１－（ｂ）に示すように，原因発生（プロセスＢのブロック１へのアクセス）からエラー現象発生（プロセスＡがブロック１をアクセス）までに時間がかかる．また，この例ではエラー現象発生時にはプロセスＢは消滅している．このように，エラー現象発生時には共通資源の状態が大きく変化しているため，原因究明が困難となる．このため，エラーの事前防止およびエラー箇所の早期検出が必要となる．

保護のために監視すべき項目は，以下に述べる４種に大別できる〈１〉，〈２〉，〈３〉．

①アクセス範囲：対象とした資源の範囲（上限／下限）を超えてアクセスを行っていないことの監視．

②アクセス方法：対象とした資源へのアクセス方法（読み／書き／実行等）が正当であることの監視．

③属性毎のアクセス権：アクセス者の属性（ユーザ／システム等）が，対象とした資源へのアクセスを許可されていることの監視．

④プロセス毎のアクセス権：アクセ

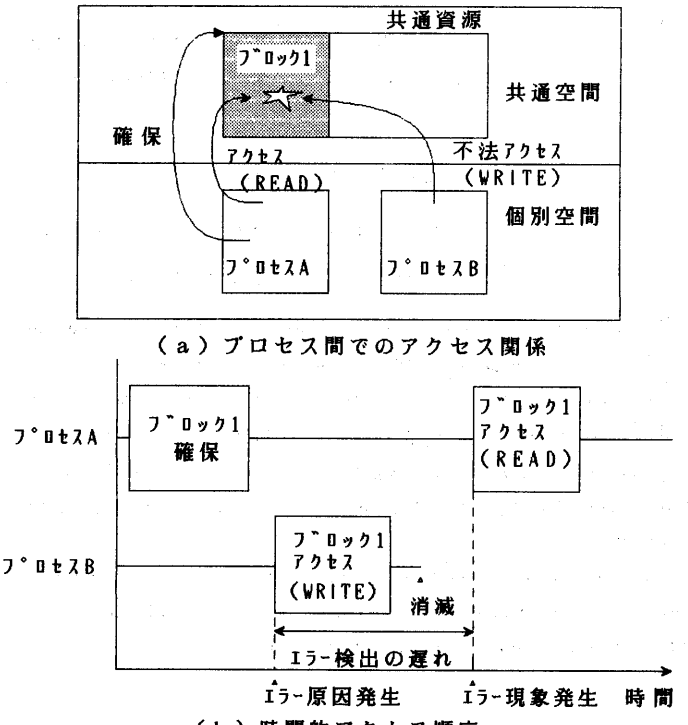


図１．共通資源への不法アクセス

スを行ったプロセスが，対象とした資源へのアクセスを許可されていることの監視．

以上の４種の監視項目に対する保護の効果を表１に示す．

共通資源においては，同一の仕事を並列に実行するような場合が多い．この様な場合には，アクセスされる資源の規模，用途が同一なため，間違った資源にアクセスした場合でも，①アクセス範囲，②アクセス方法，③属性毎のアクセス権は同一となり，①，②，③の監視項目による保護では，不法アクセスを防止することはできない．そこで，共通資源保護においては，④のアクセス権利を持ったプロセスからのアクセスか否かの判定によるプロセス単位のきめ細かな保護が必要となる．

３　従来の共通資源管理方式

共通資源の管理方法には以下に示す２種がある[4]．

ムーブ方式：共通資源をユーザ毎の空間にコピーして使用する．

ロケート方式：共通資源は共通空間に置いたまま，資源へのポインタを用いてアクセスを行う．

両者の保護方式の比較を表２に示す．

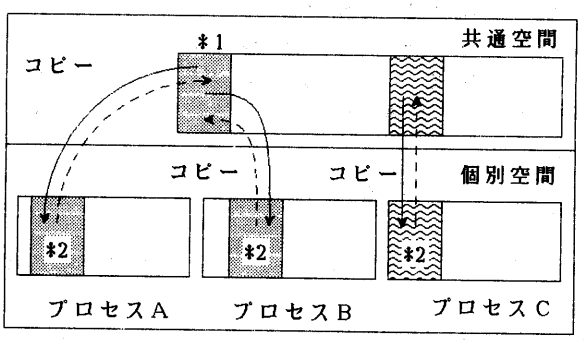
前者では，資源の確保／解放時に図２で示すように共通空間から個別空間へ

表１．共通資源における保護監視項目と効果

監視項目	保護効果	例
①アクセス範囲	・他ﾌﾟﾛｼｰｼﾞｬ/ﾌﾟﾛｸﾞﾗﾑへのｱｸｾｽ防止 ・ｽﾀｯｸ/ﾋｰﾌﾟ等のﾃﾞｰﾀｴﾘｱのｵｰﾊﾞﾌﾛｰ防止	上限／下限
②アクセス方法	・コードの書換え防止 ・定数データの書換え防止	読み／書き／実行
③アクセス権（属性毎）	・許可された属性以外からの不法ｱｸｾｽ防止	ｼｽﾃﾑ/ﾕｰｻﾞ/ﾕｰｻﾞｸﾞﾙｰﾌﾟ
④アクセス権（プロセス毎）	・他ﾌﾟﾛｾｽからのｱｸｾｽ防止	ﾌﾟﾛｾｽ

表２．共通資源管理方式の比較

項目＼方式	ムーブ方式	ロケート方式
メモリ使用効率	× 個別空間へのコピーが必要	○ ポインタの授受
通信オーバヘッド	× 間接通信	○ 直接通信
保護	○ 他空間からのアクセス防止	× 範囲チェックが必要



＊1：通信用媒体の確保　＊2：写し ⟶ 確保時
⇢ 解放時

図２．ムーブ方式

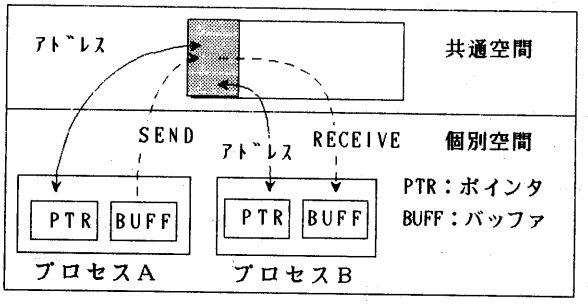


図３．ロケート方式

資源のコピーを行う必要があり，資源の確保／解放要求時における応答性が損なわれる．また，複数プロセス間で同一資源ブロックを共通的に使用する場合には図２のプロセスA，プロセスBの例で示すように共通空間を経由した通信が必要となり性能が低下する．

後者では，図３に示すように全てのプロセスが自由にアクセス可能な共通空間に資源があるため，前者のようなコピーによるオーバヘッドは防げるが，逆に不法アクセスに対する保護はソフトウェアで実現する必要が生じる．

以上のように，ムーブ方式では保護は強力であるが性能面で不十分であり，ロケート方式では性能は得られるが保護の面で不十分となる．従って，共通資源の管理には，両者の利点を兼ね備えた方式が必要となる．

4．共通資源における保護方式の提案

4．1　方式と構成概要

本章では，実時間多重処理環境においてプロセス単位での保護を実現するための方式[5]を提案する．

本方式では，

（Ⅰ）アクセス範囲チェック

（Ⅱ）アクセス方法チェック

（Ⅲ）プロセス単位でのアクセス権チェック

により前記保護を実現する．

本方式で提案する共通資源の保護・管理機構は図４に示すように，主に①共通資源管理部（ＲＳＭ），②アクセスディスクリプタテーブル（ＡＤＴ），③保護情報検査部（ＰＩＣ），④制御部（ＣＮＴ）で構成する．

ここでは，共通資源を管理する基本単位をブロックと呼ぶ．

本機構の機能は，

（イ）共通資源へのアクセス

（ロ）共通資源の確保

（ハ）共通資源の解放

の３種である．

（イ）では，プロセスからブロックを識別するための資源識別子（ＲＩＤ）とブロック内オフセット（ＢＯＦＦ）を用い，ＣＮＴにアクセス要求を出す．この時，ＰＩＣではＡＤＴ中の該当ＲＩＤに対応する保護情報を検索し，これを基に（Ⅰ），（Ⅱ），（Ⅲ）のチェックを行う．ＡＤＴ中に該当するＲＩＤが存在しないか，（Ⅰ），（Ⅱ），（Ⅲ）を満足しない場合はアクセスは禁止される．アクセス条件が満足されると，ＡＤＴ中のブロック先頭アドレス（ＢＴＯＰ）とＢＯＦＦの和を実アドレスとして共通資源をアクセスする．

（ロ）では，ＣＮＴへ確保要求を出すことによって，ＲＳＭから共通資源の１ブロックを得，要求プロセスにＲＩＤを通知する．この時，（イ）で用

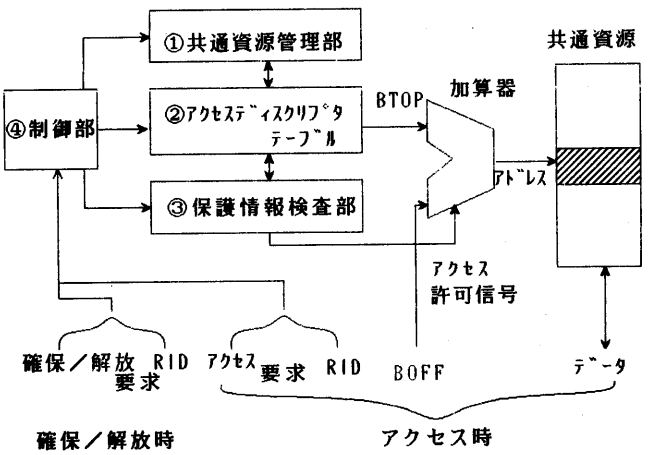


図４．共通資源保護・管理機構の構成

いた保護情報をＡＤＴに登録する．

（ハ）では，ＣＮＴへＲＩＤと解放要求を出すことによって，該当ＲＩＤの全情報をＡＤＴから削除し，ＲＳＭへ資源の返還を行う．

４．２　共通資源管理方式

共通資源管理部における確保／解放の効率化は，高速応答性を要求される環境では重要である．本方式では，資源の確保／解放とＡＤＴへの登録／削除処理を同時に行う必要があり，両者を効率よく実現する必要がある．

このため，図５に示すような方式を採用した．ここでは，予め共通資源を用途毎のプールに分割し，更にこれをプール内で同一サイズのブロックに分割する．この後，空きブロックを資源管理用のキューに登録する．キューの各要素には確保時のＡＤＴへの登録を効率よく行うため，ＲＩＤ，ＢＴＯＰ，サイズ（ＳＩＺＥ）の情報を保持する．

ＲＩＤの付与方法は，動的付与と静的付与の２種がある．

前者では資源を確保時にＲＩＤを決定し，後者では資源の初期設定時に決定する．ここでは，高速化に適している後者を採用する．

以上によって資源確保／解放はキューへの登録／削除の単純操作で容易に行え，ＡＤＴへの登録も情報が予め用意されているため高速に行える．

４．３　ＡＤＴの構成

本方式では，資源へのアクセスの度にＡＤＴを検索し，保護情報の検査を行う必要が有るため，テーブルの高速参照が要求される．

ＡＤＴを検索する方式には，多くのセグメントディスクリプタ方式で行っているように，ディスクリプタの登録番号で行う登録番号方式（図６（ａ），（ｂ））と，資源に固有の識別子を与え，この識別子で行う識別子方式（図６（ｃ））がある．

登録番号方式では複数プロセスから同一資源がアクセスされる場合，図６（ａ）のようにアクセス可能プロセス名の情報をリストチェーンで持つチェーン方式，図６（ｂ）のようにプロセス毎にテーブルを持つテーブル分離方式等の採用が必要となる．チェーン方式の場合には，アクセス時のプロセス名探索が必要となり，処理能力が低下する．テーブル分離方式の場合には，図６（ｂ）に示すようにテーブルに歯抜け状態が生じ，メモリの使用効率が低下する．

識別子方式では，メモリ使用効率の低下を防げるが，テーブル探索回数が多重度（同時共有プロセス数），使用資源数に比例して増加するため処理能力が低下する．

また，資源の解放時にはチェーン方式ではリストを検索しながら容易に削除できるのに対し，テーブル分離方式と識別子方式では全テーブルを検索する必要があり，エントリーの削除に時間がかかる．

しかし，識別子方式で連想メモリを

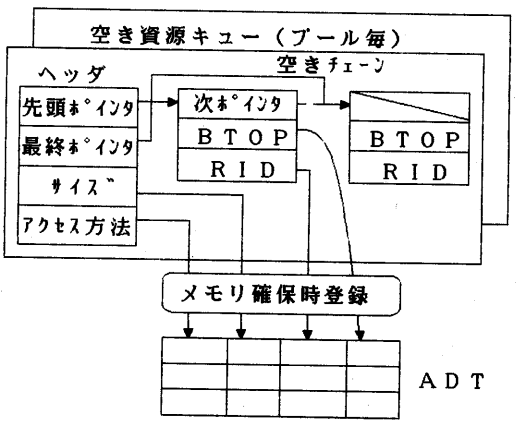


図５．共通資源管理方式

採用すれば，資源識別子をキーとした１回の検索によってＡＤＴを検索できるため，ＡＤＴへの登録／削除／検索処理の単純化・高速化が図れる．

５．本方式の適用例とその評価

本方式を実時間多重処理環境に適用した場合の性能見積りを行った．

条件としては，同時に走行するプロセスの多重度を１００，個々のプロセスが使用する共通資源を１０と仮定している．

５．１　処理性能

共通資源へのアクセス時の保護情報の検索・比較処理によるオーバヘッドを，

①本保護方式を導入しない場合

②アクセスディスクリプタテーブルの参照・更新，保護監視項目のチェックをソフトウェアで実現した場合

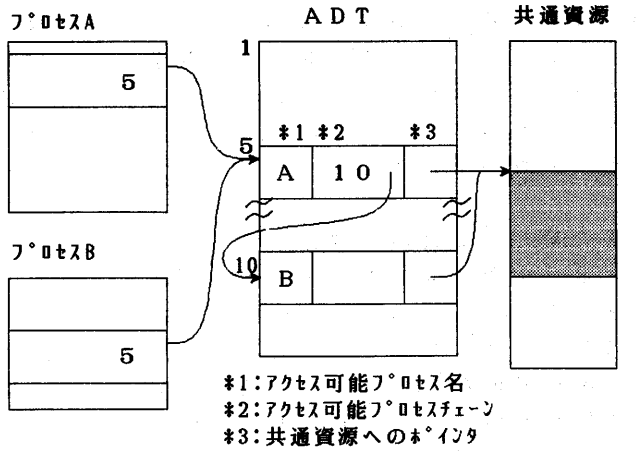
③②をハードウェアで実現した場合

について①を基準に比較した結果を図７に示す．

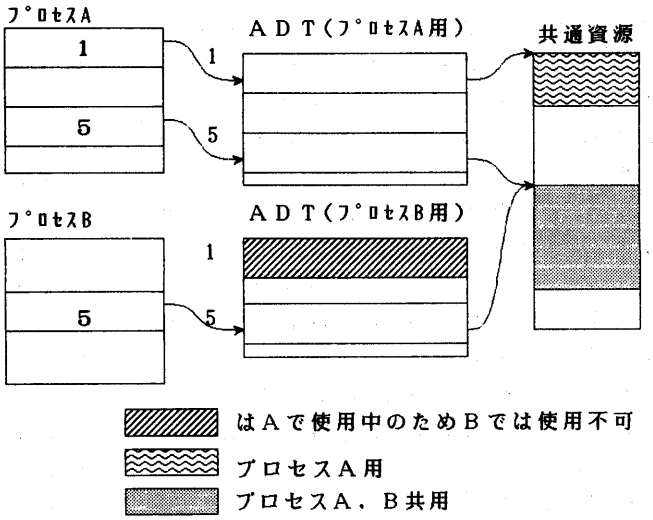
ここでは以下を前提に，処理ステップ比（ＤＳ増加率）を計算している．

（１）共通資源へのアクセス頻度

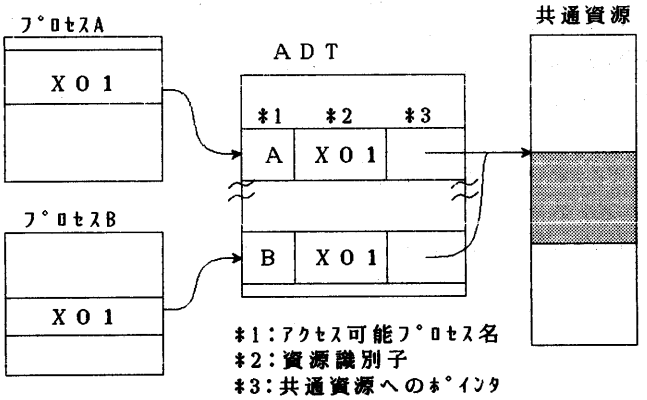
＝１つのトランザクション(*)を処理するための全ダイナミックステップ（ＤＳ）の約１０％



（ａ）登録番号方式（リストチェーン）



（ｂ）登録番号方式（テーブル分離）



（Ｃ）識別子方式

図６．ＡＤＴの構成

・メモリロード／ストア命令の使用頻度を全ＤＳの２０％として，共通資源へのアクセスをその５０％と仮定

＊：トランザクションとは交換処理では１つの電話呼，情報処理では１つのジョブまたはユーザプロセス，通信処理では１つの電文のことを指す．

（２）ソフトウェアで実現した場合の１回のアクセスにおける処理ステップ数の増加

＝１５．５ステップ

・ＲＩＤをキーとするインデックステーブルを経由して，ＡＤＴへのアクセスを最大２回行う（共有プロセス数分）と仮定

・保護情報のチェック及びＢＴＯＰのアドレス演算を含むことを想定

（３）ハードウェア化した場合の増加

＝０．５ステップ

・連想メモリのキーの設定，検索，読み出しのサイクル数／平均命令実行サイクル数により算出

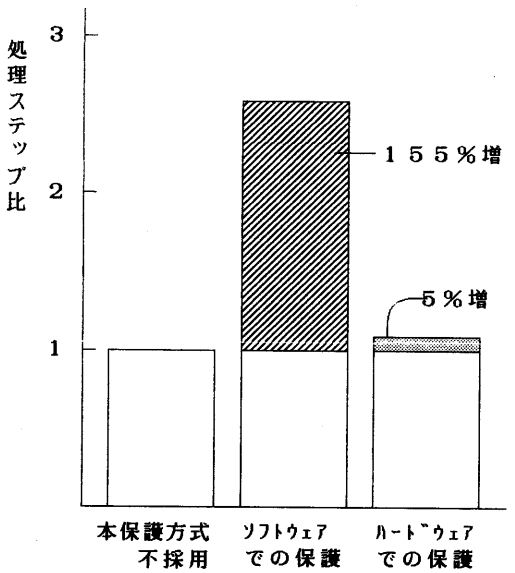


図７．処理性能比

５．２　ハードウェア量

本方式をハードウェアによって実現する際に増加するハードウェアは，共通資源管理機構，アクセスディスクリプタテーブル，テーブル制御機構，保護情報検査機構，アドレス演算機構であるが，増加の支配要因はアクセスディスクリプタテーブルと制御機構を有する連想メモリである．この規模は，以下によって約２ＫＷ×８４ｂとなる．

（１）エントリー数：プロセス多重度＝１００，共通資源数＝１０／プロセス，資源共有率（同一資源を共有するプロセス数）を２と仮定すると，２０００となる．

（２）エントリーサイズ：アドレスを３２ｂ，サイズを３２ｂと仮定すると，ＲＩＤが１０ｂ，プロセス名が７ｂ，ＢＴＯＰが３２ｂ，ＳＩＺＥが３２ｂ，保護情報が３ｂの計８４ｂとなる．

５．３　考察

本報告で提案した保護方式の適用に当たっては，ハードウェアの専用機構の導入が極めて有効である．これによって，実時間多重処理の１つの適用例においては，保護を約５パーセント程度の性能低下で実現できることが分かった．このことから，厳しい実時間多重処理性が要求されるプログラム走行環境において，本方式の適用が可能と考えられる．

ハードウェア量の増加はアクセスディスクリプタテーブルに要する連想メモリ量が主要因であるが，ＶＬＳＩ技術の適用で容易に実現できる規模であり，実現性の高いものと思われる．

また本方式は，ソフトウェア開発におけるデバッグ時の生産性向上にも有効である．

６．むすび

本検討では，実時間多重処理環境に

おける共通資源保護方式を提案し，実時間多重処理環境での適用例における実現性を確認した．

本検討では共通資源の管理を固定長としているため，可変長方式に比べ柔軟性に欠け，資源の使用効率が低下している．

今後は，

①可変長の共通資源を用いた資源の管理・保護方式

②実時間多重処理向きガーベッジコレクション方式

の検討を進め，より適用範囲が広く，効率の良い共通資源保護方式を検討する．

最後に，本報告にあたって有益な御意見，御助言を頂いた通信網第一研究所，基幹交換研究部，処理方式研究室の浜田泰昭室長ならびに室員各位に深謝致します．

［参考文献］

<1>ADVANCE INFORMATION 80386,intel Corporation,1985

<2>Intel Introduction to the iAPX432 Architecture,intel Corporation,1981

<3>G.J.Myers,"Advances in Computer Architecture",IBM System research Institute,1978

<4>香西，久保田，小菊，”高水準言語をサポートするプロセス間通信インタフェースの検討”，信学技報SE86-63,1986

<5>青木，上森，新谷，”実時間多重処理環境における共通資源保護機構の一提案”，信学会総合全国大会,No1659,1987