ユーザスクリプトをカーネル内実行する低オーバヘッドの外部ページャ機構

中村隆幸[†] 猪原茂和[†,☆] 益田隆司[†]

効率の良い分散協調アプリケーション等の構築には，分散共有メモリ等のメモリシステムの利用が効果的である．マイクロカーネル技術のひとつである外部ページャ機構を用いることで，アプリケーションの特性に応じたメモリシステムをユーザレベルで構築することができる．しかし外部ページャ機構では，ページフォールト等のイベントが発生するごとにユーザレベルで動作する外部ページャにイベント処理を依頼するため，コンテクスト切り替えや無駄なメモリオブジェクトの操作等に関するオーバヘッドが大きいことが問題であった．本論文では，カーネル内に組み込んだインタプリタにより，イベント処理を行うユーザ定義スクリプトを低オーバヘッドで動作させる機構を提案する．メモリに関するイベント処理の多くは，単純で典型的な処理列である．このことをふまえ，カーネル内インタプリタとユーザ空間で動作する外部ページャとに，イベント処理を条件に応じて振り分ける．振り分けられた多数の単純な処理列をカーネル内で実行することにより，従来問題とされていたコンテクスト切り替え等のオーバヘッドを除去する．本機構を用いることで，ユーザ定義メモリシステムの総オーバヘッドを削減することができ，効率の良いアプリケーションの構築が可能になる．

An External Pager Mechanism of a Low Overhead with User Scripts in the Kernel

TAKAYUKI NAKAMURA,[†] SHIGEKAZU INOHARA[†,☆] and TAKASHI MASUDA[†]

A memory system such as a distributed shared memory makes it efficient to build distributed cooperative applications. With an external pager mechanism, which is one of the microkernel technology, a user can implement virtual memory functions appropriate for a characteristic of an applicaion at a user level. This flexibility, however, introduces additional overheads of pager-kernel interaction; whenever an event on a memory object such as a page fault occurs, a kernel requests the pager to process it. This interaction leads to a large number of context switching, and wasteful construction of memory objects. This paper proposes the mechanism to reduce the overheads by execution of user-defined scripts to handle events with an interpreter incorporated into the kernel. An event on a memory object often requires only a sequence of simple and typical operations. Based on this property, the mechanism sorts out such an event and handles it by interpreting a brief program in the kernel to remove overheads such as context switching. The mechanism reduces the amount of overheads of a user-defined memory system.

1. はじめに

分散環境下でユーザの協調作業を支援するような新しいタイプのアプリケーションを構築するにあたって，分散共有メモリ[1]（DSM）等のメモリシステムを用いて，ホスト間の通信を記述することの有用性が指摘されている[2]．通信にメモリシステムを用いることにより，ポインタを含むデータ構造を直接通信することができ，アプリケーションの動作を効率化できる．特にメモリの一貫性制御等のアルゴリズムをアプリケーションの特性に応じて自由に記述・変更することで，より効率的な動作が可能になる．

メモリシステムの動作をユーザレベルで記述するための機構として，マイクロカーネルOSであるMachでは外部ページャ機構[3]が準備されている．外部ページャ機構では，指定したメモリオブジェクトに対して発

† 東京大学大学院理学系研究科情報科学専攻
Department of Infomation Science, Graduate School of Science, University of Tokyo
☆ 現在，日立製作所中央研究所
Presently with Central Research Laboratory, Hitachi Ltd.

生したページフォールト等のイベントの処理を，すべてユーザ空間で動作する外部ページャに依頼する．アプリケーションは適切なイベントハンドラを外部ページャとして準備することにより，DSM 等のメモリシステムの動作を，そのアプリケーションに特化させることができる．外部ページャ機構の使用にはスーパーユーザ権限は必要なく，外部ページャ機構を誤って使用した場合にも，他のプロセスやシステムは不正なアクセスから保護される．しかし外部ページャ機構を使用すると，イベントが発生するたび，ユーザ・カーネル間のコンテクスト切り替えが発生し，オーバヘッドが増加してしまう．また他のオーバヘッドの原因としては，ページのマッピング等のメモリに関する操作をユーザ空間で動作する外部ページャから安全に行わせるため，外部ページャ機構が物理ページを抽象化した高レベルのメモリオブジェクトを通じてそれらの操作を提供することがあげられる．このメモリオブジェクトの構築や更新といったデータ構造の操作が頻繁に行われるため，さらにオーバヘッドが増加する．このように従来の外部ページャ機構には，その柔軟性と安全性の代償として無視できないオーバヘッドが存在していた．

本論文では，この問題を解決する **Reactive Event Processor**（REP）機構を提案する．我々は DSM 等の機能を提供する典型的な外部ページャにおいて，イベントの多くは，短く単純な処理列から成るイベントハンドラで処理されるという点に着目した．そこで，カーネル内にこのような単純なイベント処理に特化したインタプリタを組み込み，イベントの処理を行うためにユーザから提供された簡潔なプログラムをカーネル内で解釈実行する．これにより，コンテクスト切り替えにともなうオーバヘッドを除去する．またカーネル内部で動作するインタプリタからは，適切な実行時検査を行うことで，余分なメモリ資源の抽象化なしに安全に低レベルのメモリ操作を行うことができる．これにより，抽象化のためのデータ構造の操作にともなうオーバヘッドを除去する．インタプリタの扱うプログラムは短く簡潔であるため，解釈実行にともなうオーバヘッドは十分小さい．発生したイベントのうち，単純なイベントの場合にはカーネル内のインタプリタを起動し，それ以外の複雑なイベントの処理は，従来どおりユーザ空間で動作する外部ページャに依頼する．このような一連の動作を行う REP 機構を用いることで，多くのイベントが単純なイベントハンドラで処理される典型的 DSM を，一般ユーザ権限で安全にかつ小さなオーバヘッドで実現することができる．

以下では，2 章で REP の概略について例を交じえながら示し，3 章で REP の詳細な構成を説明する．4 章で保護についての問題と解決を述べる．5 章で実装と性能評価について述べ，6 章で関連研究について触れる．最後にまとめを述べる．

2. 本システムの概略と適用例

Reactive Event Processor（REP）は，イベントフィルタとカーネル内インタプリタから構成される（図 1）．イベントフィルタは発生したイベントの処理を，与えられた条件に従ってカーネル内インタプリタないしユーザ空間で動作する外部ページャに振り分ける．カーネル内インタプリタは，ユーザから提供されたプログラムを解釈実行することで，イベントフィルタから受け取ったイベントを処理する．

カーネル内インタプリタがイベントフィルタから起動されると，ユーザプログラムによってあらかじめ登録されたイベント処理用のプログラムを解釈実行して，発生したイベントを処理する．この場合，オーバヘッドをともなうユーザプロセスへのコンテクスト切り替えは発生しない．またカーネル内インタプリタには，メモリページを直接扱うような変数やプリミティブが準備されており，オーバヘッドをともなう高レベルの抽象化を行うことなく，メモリに関する低レベルの操作を直接行うことができる．したがって，カーネル内インタプリタによってイベント処理を行う場合，これらのオーバヘッド削減効果が得られる．逆に，追加のオーバヘッドは命令解釈に関するものであり，これはプログラム実行ステップに比例して増加する．つまり，ある長さ以下のイベントハンドラについてカーネル内インタプリタを用いてプログラムを記述することにより，イベント処理の総オーバヘッドを削減する

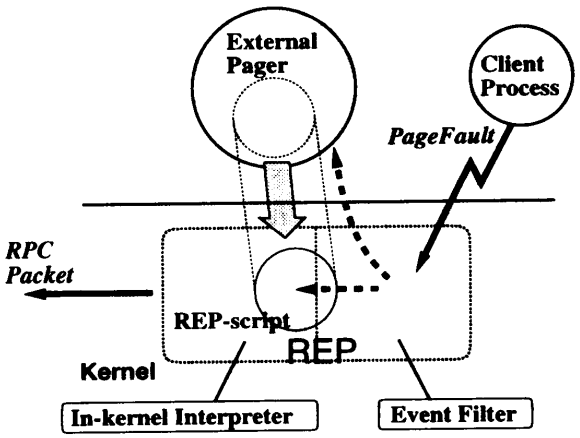

図 1 REP を用いた DSM サーバの動作例
Fig. 1 Action of a DSM server with REP.

ことができる．これらのイベント群をイベントフィルタによってカーネル内インタプリタへと振り分けることで，ユーザ定義のメモリシステム全体のオーバヘッドを削減することが可能になる．

ユーザプログラムはあらかじめイベントフィルタに対し，最適なイベント処理ができるように，短く単純なイベントハンドラで処理されるイベントについてはカーネル内インタプリタを起動するように指示し，複雑な処理を要するイベントはユーザレベルで動作する外部ページャに処理を依頼するように指示しておく．イベントフィルタは，イベント振り分け条件とそのイベントを処理するためのイベントハンドラの所在を，あらかじめユーザプログラムから受け取って登録しておく．イベントフィルタはイベント発生時に，それが登録された条件に適合するかどうかを検査し，適合したイベントについては対応するイベントハンドラへ処理を依頼する．

2.1 REP の基本的な動作

従来の外部ページャ機構を用いてDSMサーバを構成した場合と，REPを用いて構成した場合について比較してみる．ここでは一例として，Invalidateプロトコルを用いたセントラルサーバ方式のDSM[4]を採りあげる．DSMサーバは，ページを集中管理するセントラルサーバとマシンごとに準備されるローカルサーバとに分かれており，遠隔手続き呼び出し（RPC）によってサーバ間の制御やデータ転送を行う．ローカルサーバにおいて扱うイベントには，ページフォールト等のメモリに関するイベントとネットワークからのメッセージ到着イベントとがある．ここでは，メモリ読み出しでのページフォールト処理について見ていく．

外部ページャの管理するメモリオブジェクトをマップされたクライアントがInvalid状態のページに対してメモリ読み出しを行うと，ページフォールトが発生し，カーネル内の割り込みハンドラが起動される．従来の外部ページャ機構を用いた場合には，カーネルはユーザ空間で動作している外部ページャに対し，ページフォールトの発生を通知する．通知を受けた外部ページャが行う処理は，以下のように簡潔なものである．前半と後半は，それぞれ別のイベントの処理と考えられる．

- ページ読み出し要求をセントラルサーバに送る．
- ページ内容の到着を待つ．
  （この間他のイベント処理を行う．）
- ページ内容をクライアントプロセスに提供するようカーネルに依頼する．
- ページ状態を保持する変数を「無効」から「共有」に変更する．

図1は，REPを用いて構成した場合の動作を示している．イベントフィルタはページフォールトの発生を検知し，登録されている**イベント振り分け条件**と照合して，そのイベントを処理するためのカーネル内インタプリタ用プログラム（**REP script**）を起動する．カーネル内インタプリタは，REP scriptの中に書かれたパケット送出命令文を解釈実行することで，ページ読み出し要求RPCメッセージをセントラルサーバに対してカーネル内から直接送信する．その後，セントラルサーバからメッセージを受け取って正しく処理できるように，適切なイベント振り分け条件とイベントハンドラとをイベントフィルタに登録するREP script命令を実行することで，ページ内容到着の待機を実現する．

なお，ユーザがREP機構を用いて記述した外部ページャのうち，カーネル内で動作するREP scriptに対して，ユーザ空間で動作する部分を**ユーザレベルページャ**と呼ぶ．

2.2 イベントフィルタ

REPが監視対象とするカーネル内の各機能部には，そこでのイベント発生を監視するイベントフィルタが組み込まれる．イベントフィルタにはあらかじめイベント振り分け条件が与えられており，発生したイベントが条件に適合するかどうかを判断し，適合した場合は対応するイベントハンドラを起動する．具体的にはイベントフィルタは，仮想記憶管理部やネットワークパケット受信処理部に組み込まれ，ページフォールトやRPCメッセージ受信を監視し，条件を検査する．

ユーザプログラムから与えられたイベント振り分け条件は，集中管理される大きな表の形態で保持される．その表は，**メモリオブジェクトID・ページID・イベント種別**の三つ組である**イベントID**から，イベントを処理するREP scriptを得るような構成になっている．イベントフィルタはこの表を検索することにより，発生したイベントをどのように処理すればよいかを決定する．イベントを処理するためのREP scriptのエントリポイントが表に登録されていた場合，イベントフィルタはカーネル内インタプリタを起動してイベント処理を依頼する．対応するイベントが表に登録されていなかった場合，そのイベントはイベントフィルタを素通りし，ユーザレベルページャへと直接転送される．

以上のように，イベントの振り分け処理はイベントフィルタの表を引くだけの処理で済むため，短時間で終了する．イベントが表に登録されておらずユーザレ

ベルページャに転送される場合も，その処理に必要なオーバヘッドは小さい．

2.3 カーネル内インタプリタ

カーネル内インタプリタは，イベントフィルタによって指定されたREP scriptを解釈実行する．カーネル内インタプリタの受け付ける命令セットは，基本的な算術演算や条件分岐に加え，イベント処理に必要なメモリページの取り扱いやネットワークへのRPCといった高レベルプリミティブを持つ（表1）．インタプリタ命令としてレジスタアーキテクチャを採用する．インタプリタ命令は，通常のCPUの機械語命令に類似した中間コードに変換してREPに登録される．

メモリページを扱うプリミティブとして，物理ページの確保や解放，仮想アドレス空間へのマップやアンマップ，アクセス許可ビットの変更等が提供される．またネットワークへのメッセージ送受信プリミティブとして，RPCパケットのmarshalおよびunmarshal操作命令，パケット送信命令が提供される．これらの高レベルプリミティブは1命令あたりの実行時間が比較的長いため，このような命令を多く含むREP scriptを実行する場合，インタプリタの命令解釈オーバヘッドは相対的に小さくなる．多くのイベントは，少数の高レベルプリミティブのみを主に使用する単純なREP scriptによって処理されるため，全体としてインタプリタの命令解釈オーバヘッドは小さくて済む．なお，RPCパケットがネットワークから受信されると，イベントフィルタはその内容を引数バッファに入れてREP scriptを起動し，そこでパケット受信イベントの処理が行われる．そのため，カーネル内インタプリタにはパケット受信命令は存在しない．

表1 カーネル内インタプリタの命令セット（抜粋）
Table 1 An extract from an instruction set of the in-kernel interpreter.

命令	引数	機能説明
add	R1, R2, R3	加算
mul	R1, R2, R3	乗算
and	R1, R2, R3	論理積
je	R1, R2, offset	条件分岐（一致）
jae	R1, R2, offset	条件分岐（以上）
let	R1, R2	代入
return		インタプリタ終了
alloc	PR	メモリページ新規確保
map	R1, R2, PR, m	メモリページの仮想空間へのマップ
marshal	R1, fmt, args...	RPCパケット組立
sendmsg	dest	RPCパケット送出
unmarshal	R1, fmt, args...	RPCパケットからの引数の取出
filterset	R1, R2, conds...	イベント振り分け条件設定

REP scriptからアクセス可能な変数として，数個のローカルレジスタやグローバル変数のほか，各メモリオブジェクトやページごとに状態を保持しておくための変数や，メモリの物理ページを一階のオブジェクトとして直接保持する変数等が提供される．また，REP scriptがユーザレベルページャの管理するデータ構造に従って動作できるようにするため，カーネル内インタプリタからユーザレベルページャのアドレス空間を直接読み書きする機能が提供される．

外部ページャ機構の大きな目標は，一般ユーザ権限で安全にメモリシステムの機能を拡張できるようにする点にある．カーネル内インタプリタは，物理ページの低レベル操作やRPCパケットの送信，ユーザのアドレス空間の読み書き等，保護上の問題点となりうるような操作を，不正な使用に対しても安全性を保つような形で提供する．

2.4 DSMサーバへのREPの適用例

ここでは，2.1節で述べたDSMローカルサーバのイベント処理ルーチンを，実際にREP scriptで記述した例を示し，特徴的な部分について説明する．

この処理を行うREP scriptは，図2のようになる．

ローカルサーバとセントラルサーバとの間の通信にはポートを用いる．ポートとはメッセージの受け口であり，Machの基本機能としてサポートされている．Machでは一般的にタスク間通信の手段として，ポートを介した通信が用いられる．

sys_myport等のsys_で始まる変数はシステム変数

```
read-invalid:
  marshal read_req, "%p%p%i", \
      sys_myport, sys_objectid, sys_pageid

  sendmsg central_server

  filterset sys_objectid, sys_pageid, \
      (page_contents, read-invalid-wait)

  return

read-invalid-wait:
  unmarshal page_contents, \
      "%*p%*i%0Ac", pagekeep

  map [dealloc] sys_objectid, sys_pageid, \
      pagekeep, VM_PROT_READ

  let pr[0], SHARED

  filterset sys_objectid, sys_pageid, \
      (data_return, pageout-shared), \
      (lock_request, write-shared) \
      (do_not_read, do-not-read)

  return
```

図2 DSMサーバのREP script例
Fig. 2 An example of an REP script for a DSM server.

であり，自動的に設定される．central_server はグローバル変数であり，セントラルサーバへのポートを持つ．pr[0] は各ページごとに準備される変数であり，ページ状態の遷移の保持に用いる．pagekeep はページ保持変数であり，余分なデータ構造を介さずメモリページそのものを直接保持する．read_req，page_contents，data_return，lock_request，do_not_read は遠隔手続きの名前であり，固有の遠隔手続き ID を持つ．

2.1 節で述べたイベント処理の 4 つの段階について，順に見ていく．

RPC の実行は marshal と sendmsg の 2 つの段階に分けられている．marshal 命令の 2 番目の引数は，遠隔手続きの引数並びの型を示しており，REP script を中間コードに変換するときに適切な命令に変換される．

filterset 命令はイベントフィルタの表を書き換える．指定されたメモリオブジェクト ID・ページ ID・イベント種別に対して，実行すべき REP script のエントリポイントを登録している．イベントフィルタの表の書き換えによって，指定されたメッセージの到着を待つという動作を実現している．

ネットワークからの page_contents メッセージの到着によって起動される read-invalid-wait では，まず unmarshal 命令によって RPC パケットから引数を取り出す．ここでは，out-of-line で転送されてきた配列データを，ページ内容として直接ページ保持変数に格納している．

取り出されたページを map 命令によってクライアントのアドレス空間に提供し，その後 let 命令で状態変数を更新している．

以上のように REP script の各命令は，2.1 節で述べたような行うべき処理の内容をよく反映している．それぞれのイベント処理に必要な命令数は数命令と少なく，またほとんどが高レベルプリミティブであるため，インタプリタの命令解釈オーバヘッドは小さい．これらの処理はすべてカーネル内で行われ，コンテクスト切り替えは必要ない．map 操作は，従来の外部ページャ機構においてはカーネル内部で高レベルのデータ構造を余分に操作する必要があったが，REP script からはページ保持変数を用いてより直接的に行える．

3. REP 設計上の詳細

この章では，REP を構成するにあたっての詳細について，メモリ管理，イベントフィルタ，カーネル内インタプリタの順に解説する．

3.1 メモリ管理の詳細 ～REP とユーザレベルページャとの通信

REP はユーザレベルページャと様々な情報の交換を行う．REP はイベント振り分け条件や REP script を受け取り，行ったイベント処理の結果を何らかの形でユーザレベルページャに反映する．たとえば，ユーザレベルページャでイベント処理が行われページ状態が変化したときには，対応するページのイベント振り分け条件を更新しなければならない．また，REP がイベント処理を行ったときは，ユーザレベルページャが管理している対応するページの状態変数を更新しなければならない．REP の導入目的はコンテクスト切り替えのオーバヘッドを削減することにあるので，上のような状態の更新のたびにコンテクスト切り替えが発生するような通信法は採用できない．そこで REP とユーザレベルページャは，共有メモリを介して情報の交換を行う．

イベントフィルタの表や REP script 等を保持する REP の作業領域は共有メモリ上に配置され，ユーザレベルページャは提供されるライブラリ関数を用いて作業領域をアクセスする．ライブラリを通すことで，作業領域の詳細な内部構造を，善意のユーザに対しては隠蔽することができる．作業領域が不正な操作によって破壊された場合にも，REP は保護境界を越えた不当なアクセスをしないよう，適切にアクセス権管理を行う．

REP の作業領域は参照される頻度が非常に高いため，可能な限りスワップアウトされないように管理されなければならない．また，カーネル内の割り込みハンドラがユーザ空間の任意のメモリページをアクセスするためには，多少の処理を必要とする．これらの事情から，REP の作業領域は特殊なメモリオブジェクトとして確保され，専用のサブページャによって特別に管理される．

またこの節の始めに述べたように，REP がイベント処理を行った場合等で，REP script からユーザ空間のデータ構造をアクセスする必要がしばしば生じる．カーネル空間で動作する REP からは，ユーザ空間のアクセスのために，TLB 等のハードウェアのメモリ管理機構を使用することはできない．このため，(1) 仮想アドレスから物理アドレスへの変換ができない，(2) 保護をかけることができない，という問題が発生する．そこで REP は，そのつどページテーブルを調べ，これらの問題を解決する．ページテーブルの検索結果をしばらくの間 REP 内にキャッシュすれば，短い期間で同じページのアクセスの必要が生じた場合の

性能低下を抑えられる．

3.2 イベントフィルタの詳細
～メッセージ到着イベントの振り分け処理

REPが扱うイベントは，メモリに関するイベントとメッセージ到着イベントに大別される．それに対応して，イベントフィルタは仮想記憶管理部やネットワークパケット受信処理部に組み込まれる．

イベントフィルタの表を引くためには，メモリオブジェクトID・ページID・イベント種別の三つ組であるイベントIDを知る必要がある．メモリに関するイベントが発生した場合，イベントフィルタは直接イベントIDを得られる．しかしメッセージ到着イベントの場合，イベントフィルタはRPCパケットを解読して，イベントIDを求めなければならない．ネットワークパケット受信処理部に組み込まれたイベントフィルタには，**ヘッダ解析部**（HPM）が付随し，イベントIDの算出を受け持つ．

RPCパケットからイベントIDを算出する方法は，それぞれの遠隔手続きごとに異なる．ユーザレベルページャはあらかじめ，RPCパケットからイベントIDを算出する方法をHPMに登録しておく．HPMは，その情報とパケットが到着したポートの情報とを用いて，実際にイベントIDを算出する．

イベント種別の算出

イベント種別とは，具体的にはここでは遠隔手続きの種類のことである．RPCパケット内の定位置には，RPCプロトコルの遠隔手続きIDが書かれている．ただし遠隔手続きIDが同じでも，実際の遠隔手続きの種類が同じとは限らない．なぜなら，異なる通信路では異なる遠隔手続きの族を使用できるため，遠隔手続きIDが衝突していても構わないからである．つまり，遠隔手続きIDが同じでも，別の族に属する遠隔手続きである可能性がある．この場合，通信路と遠隔手続きIDの両方が与えられれば，遠隔手続きの種類が定まる．すなわち，イベント種別は，RPCパケットが到着したポートとそのパケットの内容とが与えられれば，そこから一意に求まる（**図3**）．

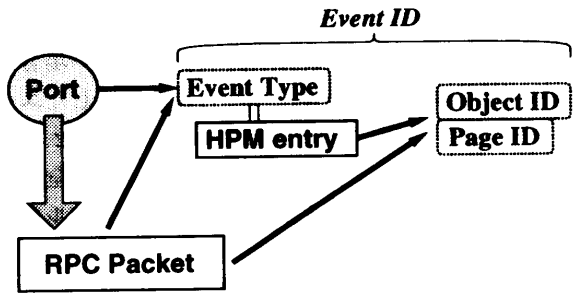

図3 イベントIDの算出
Fig. 3 Calculate an event ID.

メモリオブジェクトID・ページIDの算出

ユーザレベルページャは，RPCパケットからメモリオブジェクトIDとページIDを算出する方法について記述した**HPMエントリ**を，遠隔手続きの種類ごとに準備する．イベント種別とHPMエントリとは，1対1の関係にある．

図3は，到着したRPCパケットからイベントIDを求めるまでの手順をまとめてある．RPCパケットが到着したポートとそのパケットの内容とから，イベント種別を求める．次に，求まったイベント種別とRPCパケットの内容とから，イベント種別と1対1に対応するHPMエントリを用いて，メモリオブジェクトIDとページIDを求める．このようにして求まったメモリオブジェクトIDとページIDとイベント種別との三つ組が，イベントIDである．

以下では，HPMエントリに記述する内容について述べる．

外部ページャどうしの通信に用いる典型的なプロトコルにおいてメモリオブジェクトIDを伝えるには，次のような方法が考えられる．

(1) メモリオブジェクトIDのポートをそのまま通信に用いる☆．
(2) メモリオブジェクトごとに新たに通信用ポートを作成しそれを用いる．
(3) RPCの引数のひとつとしてメモリオブジェクトIDのポートを渡す．
(4) RPCの引数のひとつとしてメモリオブジェクトごとに新たにポートを作成しそれを渡す．
(5) それ以外．

そこでHPMエントリには，(a) 上の(1)～(5)のどの方式をとっているか，(b) どのRPCの引数にポートが渡されるか，を記述する．

また(2)，(4)のために，ポートからメモリオブジェクトIDへの変換表をHPMはユーザレベルページャから受け取り，必要なポートの変換を行う．(5)の場合に対応するために，RPCパケットからメモリオブジェクトIDを算出するための特別なREP script（**HPM script**）が登録できる．HPMは必要に応じてHPM scriptの実行をカーネル内インタプリタに依頼し，メモリオブジェクトIDおよびページIDを得る．

☆ Mach 3.0において，外部ページャ機構でのメモリオブジェクトIDとは，イベント発生時にメッセージをカーネルが外部ページャに送信するためのポートにほかならない．

ページIDの算出も同様に行われる．典型的なプロトコルにおいてページIDを伝えるには，そのページが先頭から何ページ目かを渡したり，そのページのオフセットを渡したりする方法がある．この値は，スケールファクタ *scale* とオフセット *ofs* が与えられれば，N 番目のページに対して $N * scale + ofs$ の形で一般的に表現できる．そこで，(a) *scale* および *ofs* の値，(b) どのRPCの引数に対して計算を行うか，をHPMエントリに記述する．この方法で対応できない特殊なプロトコルの場合は，HPM scriptを解釈実行してページIDを得るようにすればよい．

3.3 カーネル内インタプリタの詳細

カーネル内インタプリタに関する詳細として，この節では特にメッセージ送信と実行スケジューリングの2点について述べる．

3.3.1 RPCパケットの送信

カーネル内インタプリタでは，RPCの発行に関する操作が提供される．REP scriptからRPCを行うには，(1) RPCパケットのmarshal，(2) RPCパケット送信，という手順を踏む．送信操作を2段階に分割することにより，メッセージのマルチキャストや受信したメッセージの他のポートへの転送等の場合に，marshal操作を省略できる．

RPCパケットのmarshalのために，カーネル内インタプリタは遠隔手続きの引数型についての知識が必要である．システム標準のスタブジェネレータの生成する情報をカーネル内インタプリタが利用することはできないので，REP script中で引数型についての十分な情報を与えてやらなければならない．そこでRPCパケットのmarshal命令には，引数の値自体の記述に加え，引数型についての記述も行うことにする．

標準のスタブジェネレータが扱える手続きの型は多岐にわたる．カーネル内インタプリタが扱う遠隔手続きの型は，整数やポート，固定長・可変長配列といった基本的な型の組み合わせに限ることにする．外部ページャ間の通信プロトコルに特殊な型の組み合わせを用いることは少ない．そのうえ，そのような複雑な引数を持つ手続きは処理も複雑であると考えられるため，単純で短い処理を行うために導入したREP scriptがそのようなRPCパケットを扱うことは考慮しなくてよい．

3.3.2 スクリプト実行のスケジューリング

短く単純な処理を行うために導入されたREP scriptは，通常は長い命令列を実行することはない．イベント発生を受けて起動されるカーネル内インタプリタは，通常のプロセスとは別の，割り込み処理ルーチンのコンテクストで実行される．しかし，カーネル内インタプリタはループを記述する能力を持っているため，REP scriptが長い期間走行し続ける場合がある．そこで，カーネル内インタプリタは実行命令数をカウントし，プリエンプションを行う．プリエンプトされているREP scriptが存在する場合，タスクスケジューラは対応するユーザレベルページャの実行を行う代わりに，REP scriptの続きを実行する．

ユーザレベルページャがあるページに対して処理を行っているとき，そのページに対して新しいイベントが発生すると，処理中のページに対してREP scriptのイベントハンドラが起動される可能性がある．これを回避するため，共有作業領域にロック変数を設け，それを用いてユーザレベルページャとREP scriptとの間の排他制御を行う．

4. 保護に関する議論

REPの設計にあたっては，保護境界を侵す不正な資源アクセスへの対処が重要な点となる．外部ページャ機構では，ユーザに対してメモリシステムの拡張を安全な形で行えるようにするのが，もともとの目標であった．REPの導入によって，外部ページャ機構の安全性が損なわれてはならない．この章では，保護上の問題となりうる点について，その対処法をまとめる．

作業領域をユーザ空間に置くことについて

通信オーバヘッドを最小にするため，REPの作業領域は共有メモリ上に配置され，ユーザレベルページャは提供されるライブラリ関数を用いて作業領域をアクセスすることになっている．ユーザが不正なアクセスによって作業領域を破壊した場合に備え，REPは共有作業領域には，破壊されても破綻しないような形式で情報を置く．具体的には，メモリページや通信路等の資源は小さい整数で表される記述子の形で変数に格納し，記述子から実際の資源の低レベル内部表現への変換表は，共有されない安全な作業領域に置く．

REPからユーザ空間を操作できることについて

ユーザレベルページャのデータ構造を参照または更新しながら動作するREP scriptを使用したい場合がある．REP scriptからは，それ自身を登録したユーザレベルページャのアドレス空間以外へのアクセスは，許可されない．カーネル内インタプリタは，安全にメモリアクセスを行うために，適宜ページテーブルを参照して妥当性検査を行う．

メモリの低レベル操作について

REP scriptからは，メモリページのマップ操作等，メモリに関する低レベルのプリミティブ群が利用可能

であり，これらの使用によって処理のオーバヘッドを減らしている．このようなプリミティブの実行を安全に行うために，カーネル内インタプリタはメモリページを保持する専用の変数群を準備しており，低レベル操作を行うプリミティブはこの変数群を用いて資源を扱う．またカーネル内インタプリタは，これらの変数を上書きや消去する場合に変数の内容を適切に検査することによって，資源の授受を正しく追跡する．以上のような手法を用いることで，変数を用いた低レベル資源の安全な操作を実現している．

送信できる通信路について

REP script からは，ネットワークに対する RPC パケット送信プリミティブが利用可能である．この実行を安全に行うために，REP が扱うことのできる通信路は，対応するユーザレベルページャが確立した通信路に限ることにする．これによって，アクセス権を侵すような不正なパケットの送出を防ぐ．

5. 実装と性能評価

SPARCstation ELC で動作する NetBSD-1.0 上に，REP を持つ外部ページャ機構を実装し，REP を使用した DSM サーバを実装する．NetBSD カーネル内の仮想記憶管理部に外部ページャ機構相当の機能を追加したうえで，仮想記憶管理部およびネットワークパケット受信処理部にイベントフィルタを埋め込み，またカーネル内インタプリタやシステムコール処理ルーチン等の追加を行った．現在のところ，扱えるメモリオブジェクトの数を限定した REP の試験的な実装が動作しており，その上で DSM サーバを実装中である．

REP の能力を確認し，REP を使用することで性能を改善できるイベントハンドラの長さの上限を概算するため，カーネル内インタプリタの命令実行速度およびページフォールト処理に要する時間についてのマイクロベンチマークを行った．測定には SPARCstation ELC を使用した．

カーネル内インタプリタの命令実行速度

基本算術 4 命令と条件分岐 1 命令から成るループを 20 回（すなわち 100 命令分）繰り返す REP script の実行に要する時間と，対照用にループを 1 度も回らず終了する REP script の実行時間とを測定した（**表 2**，単位は μ秒）．表中の値は 100 回測定の平均値である．この表から，1 命令あたりの実行時間は約 6.4 μ秒であることが分かる．

ページフォールト処理に要する時間

イベント処理を REP で行う例として，ページフォールトの処理について実験した．ユーザプロセスのメモリアクセスによって生じたページフォールトに対して，あらかじめ準備しておいたページを供給し，ユーザプロセスが再開するまでに要する時間を測定した（**表 3**，単位は μ秒）．表中の値は 20 回測定の平均値である．測定値のうち，user はユーザ空間で動作する外部ページャを用いて処理した場合，REP はイベントハンドラの構築に REP を用いた場合，kernel はそれらと同等の処理をカーネル内に直接埋め込んだ場合である．

表 2 カーネル内インタプリタの命令実行速度
Table 2 Interpretation speed of the in-kernel interpreter.

loop（100 命令） + return	676.6 μ秒
return	40.4 μ秒
1 命令あたり	6.36 μ秒

表 3 ページフォールト処理に要する時間
Table 3 Elapsed time for page-fault handling.

処理方式	時間（μ秒）
user	1036
REP	353
kernel	280

kernel の数値は，イベント処理の応答速度の上限を表していると考えられる．REP を用いた場合の性能は上限に比較的近く，ユーザが供給したイベントハンドラが低いオーバヘッドで動作していることが分かる．

また，user と REP との差は約 680 μ秒であり，前節の実験から 1 命令あたりの実行時間は約 6.4 μ秒である．したがって，およそ 100 命令以内の長さのイベントハンドラについては，REP を用いて記述することで性能が改善できることが分かった．多くの典型的な DSM サーバでのイベント処理において，REP script で 100 命令というのは十分な長さである．一部のイベントハンドラがそれ以上の長さになる場合には，その部分の処理だけを，ユーザレベルページャに転送すればよい．

6. 関連研究

Packet filter[5)~7)]は，ネットワークのトラフィックモニタ等のアプリケーションをユーザレベルで実装するための機構である．カーネル内に組み込んだインタプリタがユーザから供給されたプログラムを実行することで，そのアプリケーションが着目すべきパケットだけを選り分け，ユーザ空間で動作するアプリケーションへと転送する．文献 8) は同様のパケット選り分けを，表によるマッチングで行う．これらは，カーネル内にインタプリタを組み込む点や，パケットを選り分けて一部をユーザプロセスに送る点等で REP と

関連性がある．

HiPEC[9]は，カーネル内にインタプリタを埋め込み，外部ページャ機構にページ置き換えポリシーのカスタマイズ機能を追加する．コンテクスト切り替え回数の増加に対処するため，カーネル内のインタプリタが，ポリシーを記述したプログラムを解釈実行する．DSM のオーバヘッドの削減が目的の REP とは，外部ページャ機構にインタプリタを埋め込む点は共通だが，その目的がページ置き換えポリシーのカスタマイズであることが大きく異なる．

文献 10)~12) は，一般に拡張可能 OS と呼ばれているシステムである．中でも SPIN[10]は，ユーザが供給するプログラムをカーネル内で動作させる大規模な機構を提供している．型安全な言語で記述されたプログラムは，カーネルに供給されると動的にコンパイルされ，カーネル内で安全に実行される．これにより，ネットワークやメモリ管理等の動作をユーザが安全に記述可能である．これに対して我々は対象をメモリシステムに絞り，短く単純なイベント処理が多いためにインタプリタの主要なオーバヘッドである命令解釈オーバヘッドを十分小さくできるため，システムをより小規模にできると考える．

Application-specific Safe Handlers (ASHs)[11]は，アプリケーションレベルのメッセージハンドラであり，kernel 内にバイナリコードが送り込まれ実行される．ASHs は exokernel の枠組みの下で，REP と類似の機能を提供する．ASHs は扱えるプロトコルの制約がない代わりに，使用するには普通はスーパーユーザ権限が必要になる．REP は対象領域を限定し，扱うプロトコルを RPC に絞ることで，一般ユーザ権限でも安全に使用できる．

7. まとめ

本論文では，Reactive Event Processor (REP) を持つ外部ページャ機構を提案した．REP を用いることにより，イベント処理の多くを占める単純で短いイベント処理を，カーネル内で安全に行うことができ，コンテクスト切り替え等のオーバヘッドを削減することができる．

今後の課題としては，REP の実装を進め，実際に DSM サーバを構築した際の性能を評価するとともに，より複雑なプロトコルを用いる DSM にも対応可能であることを実装によって検証することが考えられる．また，本論文と同様の手法を他のアプリケーションにも適用し，一般的な枠組みを抽出することも今後の課題である．

参考文献

1) Li, K. and Hudak, P.: Memory Coherence in Shared Virtual Memory Systems, *ACM Trans. Computer Systems*, Vol.7, No.4, pp.321–359 (1989).
2) Inohara, S. and Masuda, T.: Using Huge Address Spaces to Construct Cooperative Systems, *Proc. International Symposium on Autonomous Decentralized Systems (ISADS) 93*, pp.85–92, IEEE Computer Society (1993).
3) Young, M.W.: Exporting a User to Memory Management from a Communication-oriented Operating System, Technical Report, DEPTCS., Carnegie-Mellon University (1989).
4) Nitzberg, B. and Lo, V.: Distributed Shared Memory: A Survey of Issues and Algorithms, *IEEE Computer*, Vol.24, No.8, pp.52–60 (1991).
5) Mogul, J.C., Rashid, R.F. and Accetta, M.J.: The Packet Filter: An Efficient Mechanism for User-level Network Code, *Proc. 11th Symposium on Operating System Principles (SOSP)*, pp.39–51 (1987).
6) McCanne, S. and Jacobson, V.: The BSD Packet Filter: A New Architecture for User-Level Packet Capture, *Proc. 1993 Winter USENIX Technical Conference*, pp.259–269 (1993).
7) Yuhara, M., Bershad, B.N., Maeda, C. and Moss, J.E.B.: Efficient Packet Demultiplexing for Multiple Endpoints and Large Messages, *Proc. 1994 Winter USENIX Technical Conference*, San Francisco, CA, pp.153–165 (1994).
8) Bailey, M.L., Gopal, B., Pagels, M.A., Peterson, L.L. and Sarkar, P.: PATHFINDER: A Pattern-based Packet Classifier, *Proc. First Symposium on Operating Systems Design and Implementation*, pp.115–123 (1994).
9) Lee, C.H., Chen, M.C. and Chang, R.C.: HiPEC: High Performance External Virtual Memory Caching, *Proc. First Symposium on Operating System Design and Implementation*, pp.153–164 (1994).
10) Bershad, B.N., Savage, S., Pardyn, P., Sirer, E.G., Fiuczynski, M.E., Becker, D., Chambers, C. and Eggers, S.: Extensibility, Safety and Performance in the SPIN Operating System, *Proc. 15th Symposium on Operating System Principles* (1995).
11) Engler, D.R., Kaashoek, M.F. and O'Toole Jr., J.: Exokernel: An Operating System Architecture for Application-level Resource Management, *Proc. 15th Symposium on Operating*

System Principles (1995).
12) Cheriton, D.R. and Duda, K.J.: A Caching Model of Operating System Kernel Functionality, *Proc. First Symp. on Op. Sys. Design and Impl.* (*OSDI*), pp.179–193 (1994).
13) 中村隆幸，猪原茂和，益田隆司：カーネル内のユーザ定義イベントハンドラを用いた外部ページャ機構，1995年並列/分散/協調処理に関する『別府』サマー・ワークショップ (SWoPP 別府'95) 予稿集，Vol.95-OS-70, pp.73–80, 情報処理学会 (1995).

(平成8年4月18日受付)
(平成8年9月12日採録)

中村 隆幸

1971年生．1995年東京大学理学部情報科学科卒業．現在，同大学大学院修士課程在学中．拡張可能なOS構築法の研究に従事．

猪原 茂和（正会員）

1967年生．1993年東京大学大学院理学系研究科情報科学専攻博士課程中退，同専攻助手に就任．1995年同専攻より学位取得．1996年より現在まで（株）日立製作所中央研究所．分散・並列オペレーティングシステムおよびミドルウェア，永続・並列データベース管理システム，トランザクション処理等のシステムソフトウェアに興味を持つ．理学博士．情報処理学会，ACM，IEEE，USENIX各会員．

益田 隆司（正会員）

1939年生．1963年東京大学工学部応用物理学科卒業．1965年同大学大学院修士課程修了．同年（株）日立製作所入社．1977年から筑波大学，1988年3月から東京大学に勤務．現在，同大学大学院理学系研究科情報科学専攻教授．専門はオペレーティングシステム．