システムソフトウェアと
オペレーティング・システム 70−10
（１９９５．８．２３）

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

中村 隆幸　猪原 茂和　益田 隆司

東京大学 大学院理学系研究科 情報科学専攻

要旨

近年のマイクロカーネル技術の発展により登場した外部ページャ機構を用いると、それぞれのアプリケーションの特性に応じて分散共有メモリ等の機能を柔軟に実現することができ、分散協調アプリケーションなどの効率的な構築に有効である。しかしページフォールト等のイベントが発生するたび、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。このためコンテクスト切り替えの回数が多く、無駄なメモリオブジェクトの構築を必要とするなど、オーバヘッドが大きい。

本稿では、カーネル内に組み込んだインタプリタにより、柔軟性を損なわずに外部ページャ機構の総オーバヘッドを削減する手法を提案する。メモリオブジェクト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。この事実に着目し、外部ページャが状態に応じて供給した簡潔なプログラムをカーネル内インタプリタが実行することで、イベントの多くを外部ページャに転送することなくカーネル内で処理することができ、オーバヘッドの削減が可能となる。

An External Pager Mechanism with User-defined Event Handlers in the Kernel

Takayuki Nakamura, Shigekazu Inohara and Takashi Masuda

Department of Information Science, Graduate School of Science, University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113

Abstract

An external pager mechanism, which has appeared through the recent development of microkernel technology, provides the ability to enhance the virtual memory mechanisms at the user level. With this mechanism, users can implement virtual memory functions for new kinds of applications, such as distributed shared memory for distributed cooperative applications. 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 that reduces the overheads of pager-kernel interaction without spoiling the flexibility. Events often requires only the sequence of simple and typical operations. Based on this property, an interpreter incorporated into the kernel handles most events without forwarding them to an external pager by interpreting brief programs, which are sent from external pagers depending on the status of memory objects.

1 はじめに

分散コンピューティング環境でのアプリケーションでは、ホスト間の通信が本質的に重要な意味を持っている。特に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構築においては、分散共有メモリ (DSM) 等のメモリシステムを用いて通信を記述することの有用性が指摘されている。

マイクロカーネルOSであるMachでは、外部ページャ機構が導入された。これはページフォールト等のメモリに関するイベント処理を、ユーザ空間で動作する外部ページャに依頼する機構である。外部ページャ機構により、DSM等のメモリシステムを、ユーザが使用するアプリケーションに応じて柔軟に提供することが可能となった。このように動作がカスタマイズ可能なメモリシステムを使用することにより、アプリケーションの特性に応じたDSMプロトコルを使用することができ、実行効率を改善することが可能である。

しかしながら外部ページャ機構には、その柔軟性の代償として無視できないオーバヘッドが存在する。外部ページャの管理するメモリオブジェクトに関して発生したイベントは、すべてユーザ空間で動作する外部ページャに通知される。その度にユーザとカーネルの保護境界をまたいで処理が移されることとなり、コンテクスト切り替えの回数が増大し、これがオーバヘッドの増加を招く。また外部ページャ機構は、ユーザ空間で動作する外部ページャに対して物理ページのクライアントプロセス空間へのマッピング等の操作を許しながら、なおかつシステムの安全性を保つ必要がある。そのため外部ページャ機構は、物理ページを抽象化する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ことになるため、さらにオーバヘッドが増加する。

本稿では、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、イベント処理ルーチンの多くが短く単純な処理列である点に着目した。そこで、カーネル内にこのような単純なイベントの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必要に応じて解釈実行する。またそれ以外の複雑な処理は、従来通りユーザ空間で動作する外部ページャに依頼する。以上のような動作を行うREP機構をカーネル内に組み込むことによって、単純なイベント処理はカーネル内で、複雑なイベント処理はユーザ空間で動作する外部ページャで行うことが可能になり、上で述べたオーバヘッドの大部分を削減することが可能となる。

本稿の残りの部分の構成は以下の通りである。2章ではREPの概略について、例を交えながら示す。3章ではREPの構成について詳細に述べる。4章で実装と性能評価に関して述べ、5章で関連研究について触れる。最後にまとめを述べる。

2 システムの概略

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。イベントフィルタは、イベントフィルタリング条件とそのイベントを処理するための動作の記述をあらかじめユーザプログラムから受け取って登録し、発生したイベントの中から与えられた条件に適合するものだけをふるい出す。ふるい出されたイベントはカーネル内インタプリタに送られ、登録されたプログラムを解釈実行することでそのイベントを処理する。

2.1 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。

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はシステムコールを用いて、セントラルサーバにページ内容を要求する遠隔手続き呼び出し (RPC) メッセージを送出する。

図1は、REPを用いて構成した場合を示している。発生したページフォールトはイベントフィルタに送られ、登録されているイベントフィルタリング条件と照合される。そしてイベントが登録された条件に適合していることが分かると、イベントフィルタはそのイベントをカーネル内インタプリタに送り、指定されたイ

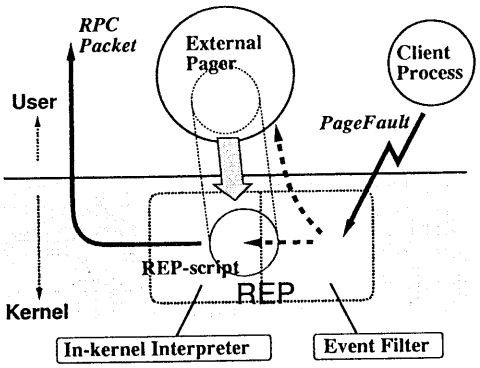

図 1: REP を用いた DSM サーバの動作例

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、ページ内容を要求する 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出する。このような手順により、ページフォールト発生というイベントの処理を、ユーザ空間で動作する外部ページャ (ユーザレベルページャ) に依頼することなく、カーネル内で行うことができる。

またそのページのイベントフィルタリング条件として読み出しでのページフォールトが登録されていない場合、そのイベントはイベントフィルタを通って従来通りユーザレベルページャへと転送される。このため、ページの状態により複雑なイベント処理を必要とする場合にも、効率の低下を起こすことなく従来通りの処理を行うことができる。

2.2 REP の 2 つの構成要素

イベントフィルタ 注目するメモリオブジェクトに関するページフォールトやスワップアウト等のイベントが発生したり、指定されたポートを通じてネットワークからメッセージが RPC パケットの形で到着すると、それらはイベントフィルタに送られる。

イベントフィルタの中心は、メモリオブジェクト ID・ページ ID・イベント ID からイベントを処理する REP-script を求めるための、大きな表である。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イベントフィルタの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。

メモリオブジェクトに関するイベント発生時には、表を引くために必要なメモリオブジェクト ID・ページ ID・イベント ID はすべて自明である[1]。ネットワークからのメッセージ到着の場合、受け取った RPC パケットからメモリオブジェクト ID・ページ ID・イベント ID を算出する。

以上のように、イベントのフィルタリング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する場合も、オーバヘッドは小さい。

カーネル内インタプリタ イベントフィルタによってふるい出されたイベントを処理する REP-script は、カーネル内インタプリタによって解釈実行される。カーネル内インタプリタの受け付ける命令セットは、通常の CPU が持つような基本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。インタプリタ命令は汎用レジスタアーキテクチャを採用し、通常の CPU の機械語命令に類似したエンコーディングを施される。

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・解放、仮想アドレス空間へのマップ・アンマップ、アクセス許可ビットの変更、などが提供される。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は、RPC パケットの marshall・unmarshall、パケットの送信命令が提供される。なおネットワークから受信したパケットは、イベントフィルタに転送され必要に応じて適切な REP-script が起動されてそこで処理されることになるため、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。

REP-script からアクセス可能な変数として、数個のグローバルレジスタの他、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が提供される。また REP-script がユーザレベルページャの

[1] Mach の外部ページャ機構では、メモリオブジェクトに関するイベント発生は、カーネルから外部ページャに対する RPC の発行という形で通知される。

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。

2.3 分散共有メモリサーバへの適用例

この節では、典型的なInvalidateプロトコルDSMサーバをREPを用いて構成した例のうち、読み出しのページフォールトを処理するローカルサーバのイベント処理ルーチンを示す。

クライアントがInvalidのページに対して読み出しを行うと、読み出しのページフォールトが発生する。外部ページャはセントラルサーバに対して読み出し要求を発行し、リプライを待つ。セントラルサーバからページ内容が到着すると、そのページをクライアントに供給し、状態機械を "Shared" に設定する。

この処理を行うREP-scriptは次のようになる。

```
read-invalid:
  marshall 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:
  unmarshall page_contents, \
    "%*p%*i%OAc", 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, pageout-write) \
    (do_not_read, do-not-read)
  return
```

sys_myport, sys_objectid, sys_pageid, sys_pagerはシステム変数であり、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、状態機械の保持に用いる。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。

RPCの実行はmarshallとsendmsgの2つの段階に分けられている。marshall命令の2番目の引数は、遠隔手続きの型を示しており、REP-scriptのアセンブル時に適切な命令に変換される。

filtersetまた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ID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イベントの到着によって起動されるread-invalid-waitでは、まずunmarshall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、本節の始めに述べたような行うべき処理の内容をそのまま反映している。それぞれのイベント処理に必要な命令数は4～5命令と少なく、インタプリタの解釈オーバヘッドは小さい。これらの処理は全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。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操作する必要のあったmap操作は、ページ保持変数を用いてより直接的に行われる。

3 各部の構成

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・デバッグ環境の順に解説する。

3.1 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。イベントフィルタリング条件やREP-scriptを受け取り、イベント処理を自動的に行ったことを何らかの形で通知する。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フィルタリング条件を更新し

なければならない。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機械を更新しなければならない。

コンテクスト切り替えのオーバヘッドを削減するために REP を導入したのであるから、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。従って、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必要がある。ユーザレベルページャは、REP システムの提供するライブラリ関数を用いて、共有メモリである REP 作業領域をアクセスする。ライブラリを通すことで、REP 作業領域の詳細な内部構造を、善意のユーザに対して隠ぺいすることができる。

REP はイベントが発生するごとに、REP 作業領域を読む必要がある。この頻度は非常に高いため、作業領域は可能な限りスワップアウトされないことが強く要求される。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 は作業領域を特殊なメモリオブジェクトとして確保し、それを管理するための専用のサブページャを準備する。

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。ユーザレベルページャの管理する状態機械を更新したり、ユーザレベルページャの管理するディレクトリに従ってマルチキャストを行う場合がそれに当たる。カーネルアドレス空間で動作する REP からは、ハードウェアのメモリ管理機構は使用できない。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。REP-script からユーザ空間をアクセスするときには、REP がその都度ページテーブルを調べ、これらの問題を解決する必要がある。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。単純なイベント処理を行うために導入された REP-script からは、アクセスすべきユーザ空間のデータ量は限られているため、キャッシュサイズは数ページ程度でよい。

3.2 イベントフィルタ

2.2節で述べたように、ネットワークからのメッセージ到着イベントのフィルタリングでは、まず RPC パケットからメモリオブジェクト ID・ページ ID・イベント ID を算出する。しかし、受け取ったメッセージから遠隔手続き ID (≒イベント ID) は自明であるが、そのメッセージがどのメモリオブジェクトのどのページに対してのものであるかはすぐにはわからない。なぜなら、遠隔手続きの種類によって引数のパターンが異なるのが普通であり、どの引数にどのような形でメモリオブジェクト ID やページ ID が含まれているかが一意ではないためである。

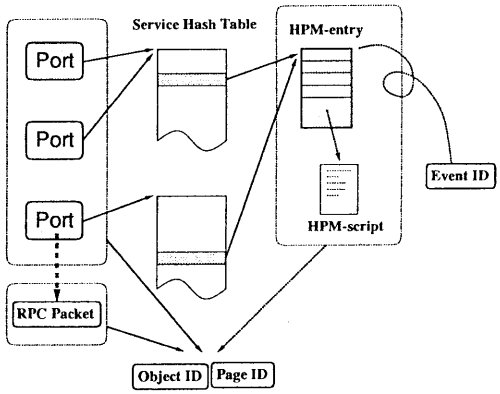

図 2: メモリオブジェクト ID・ページ ID の算出

そこでユーザレベルページャはあらかじめ、RPC パケットからメモリオブジェクト ID とページ ID を算出する方法を遠隔手続きの種類ごとに登録しておく。ネットワークから到着したメッセージを受け取ったイベントフィルタは、ポートと遠隔手続き ID から得られた算出法に従ってメモリオブジェクト ID とページ ID を算出し、それを用いてイベントを処理するための REP-script のエントリポイントを表から検索する。

3.2.1 ヘッダ解析モジュール

RPC パケットからのメモリオブジェクト ID とページ ID の算出は、イベントフィルタ中のヘッダ解析モジュール (HPM) が担当する。算出法は遠隔手続きの種類ごとに異なっているため、算出法をそれぞれの手続きごとに準備された **HPM** エントリに記述しておく。図 2は、メモリオブジェクト ID とページ ID の算出の手順を示している。

メッセージの監視対象となる各ポートには、サービスハッシュ表が対応づけられている。複数のポートが

同じサービスハッシュ表を指していてもよい。サービスハッシュ表は、遠隔手続きIDからHPMエントリの位置を求める表である。遠隔手続きIDは、システム全体で一意とは限らない。異なるポートを通じてのRPCであれば、別の型の遠隔手続きが同じIDを持つこともありうる。従って、異なるポートは別のサービスハッシュ表と関連づけできる必要がある。またRPCパケットのヘッダに書かれている遠隔手続きIDは、上の意味でシステム全体で一意なイベントIDとはなり得ない。そこでイベントIDとして、上のように求められたHPMエントリの位置を用いる。

外部ページャ同士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、(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]、(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1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1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、といった選択肢が考えられる。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（ポート変換テーブル）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の解釈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（及びページID）を得る。

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すか、その数を何倍かしたものを渡す（特にページサイズ倍したものを渡すことが多い）か、さらにそれにある定数を加算したものを渡すか、である。そこで、(a)このような計算のための定数に何を用いるか、(b)どのRPCの引数に対して計算を行うか、をHPMエントリに記述しておけば、多くのプロトコルでページIDの算出が可能である。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。

[2]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他ならない。

3.2.2 イベントフィルタリング条件の検査

どのイベントに対してどのREP-scriptを起動すればよいかを示すイベントフィルタリング条件は、各ページごとに異なっている。REPが管理する全てのページには、現在のイベントフィルタリング条件の記述へのポインタが格納されている。イベントフィルタリング条件の記述とは、イベントIDからREP-scriptの実行開始位置を求める表である。

ページフォールト等のメモリに関するイベントの場合には即座に、またネットワークからのメッセージ到着の場合にはHPMによって、メモリオブジェクトID・ページID・イベントIDが得られる。前者2つを用いて、対象ページのイベントフィルタリング条件を記述した表を得る。この表をイベントIDで引くことで、イベント処理を行うためのREP-scriptの実行開始位置を得ることができる。表に当該イベントIDの項目が無い場合、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転送される。

3.3 カーネル内インタプリタ

カーネル内インタプリタはREP-scriptを解釈実行するインタプリタである。カーネル内インタプリタは、(1)発生したイベントの処理をイベントフィルタに依頼される、(2)マルチキャスト等カーネル内でより効率的に行える処理をユーザレベルページャに依頼される、(3)HPM-scriptの実行をイベントフィルタに依頼される、といった方法で起動される。

3.3.1 RPCパケットの取り扱い

カーネル内インタプリタには、RPCの発行に関する操作が提供されている。REP-scriptからRPCを行うには、(1)RPCパケットのmarshall、(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、marshall操作を省略できる。

RPCパケットのmarshallのために、カーネル内インタプリタは遠隔手続きの型についての知識が必要である。標準のスタブジェネレータの生成する情報を利

用することはできないので、REP-script中で手続きの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。そこでパケットのmarshallを行うための可変長命令には、引数自体の記述に加え、遠隔手続きの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。その上、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、簡単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。しかしカーネル内インタプリタは後方への条件分岐命令をサポートしているため、無限ループを記述することができる。これはユーザレベルのデータ構造に従ってマルチキャストを行う場合などには、重要な機能である。従って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わなければならない。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かわりに、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、REP作業領域にロック変数を設け、REPはそれを監視してイベント処理を一時停止する。ユーザレベルページャはロック変数を適切に制御する必要がある。

3.4 デバッグサポート

REPを用いたアプリケーションの開発にあたって、カーネル内で動作するイベントフィルタリング条件やREP-scriptの動作テストが問題になる。これらがカーネル内に組み込まれたREPによって処理される限り、ユーザプロセスのデバッグを対象とする従来のデバッグツールを用いたデバッグは困難である。REPの各部にデバッグ用のフック地点を準備しておくという手法も考えられるが、この手法は柔軟性に乏しく、平常時の実行効率やセキュリティの観点からも疑問が残る。

そこで、柔軟で強力なデバッグ環境を提供するために、REPと同等のシステムをユーザ空間で動作するライブラリとして実現する方法が考えられる。REPの作業領域はほぼ全てユーザ空間に置かれており、ユーザレベルページャの視点からREPと等価に見えるシステムを、ユーザ空間で構築することは可能である。カーネル内に組み込まれた真のREPには、あらかじめイベントフィルタの設定を「全てのイベントを指定のポート経由で外部ページャに送信する」ようにしておく。外部ページャにリンクされたデバッグ用疑似REPは、管理用に作成したポートから受け取ったイベントを、真のREPが行うのと同様にフィルタリングし、必要に応じて疑似カーネル内インタプリタを起動する。疑似REPには、イベントフィルタリング条件を監視したりREP-script実行をトレースするような機能を組み込む。

これにより、真のREPには何等特別な仕組みを準備することなく、柔軟なデバッグ環境を実現することができる。新たなデバッグ用のフック地点を設けたい場合には、ユーザレベルのライブラリを修正するだけでよい。

4 実装

SPARCstation/ELCで動作するNetBSD-1.0/sparc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。現在のところ、REP及びDSMサーバが一部動作している。

REPの能力を確認するため、動作している部分に関して、いくつかのマイクロベンチマークを行った。

カーネル内インタプリタの命令実行速度 基本算術と条件分岐から成るループを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の実行時間を測定した(表1, 単位はμ秒)。この表から、1命令当たりの実行時間は約6.4μ秒であることがわかる。

ページフォールト処理に要する時間 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

loop (100 命令) + return	676.6 μ秒
return	40.4 μ秒
1 命令	6.362 μ秒

表 1: カーネル内インタプリタの命令実行速度

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 2, 単位は μ 秒)。表中の 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。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比較的上限に近く、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わかる。

処理方式	user	REP	kernel
時間 (μ秒)	1035.5	353.4	280.3

表 2: ページフォールト処理に要する時間

5 関連研究

Packet filter[5] 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より分け、ユーザ空間で動作するアプリケーションへと転送する。

HiPEC[3] は、ページ置き換えポリシーのカスタマイズ機能を外部ページャ機構に追加する。コンテクスト切り替え回数の増加に対処するため、カーネル内に埋め込んだインタプリタが、ページ置き換えポリシーを記述したプログラムを解釈実行する。

拡張可能 OS である SPIN[2] 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。この枠組みにより、メモリ・CPU 時間・ネットワーク等の資源の管理をユーザがカスタマイズ可能である。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と考える。

6 まとめ

本稿では、Reactive Event Processor 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検証することが考えられる。

参考文献

[1] Mary L. Bailey, Burra Gopal, Michael A. Pagels, Larry L. Peterson, and Prasenjit Sarkar. PATHFINDER: A pattern-based packet classifier. In *Proceedings of the First Symposium on Operating Systems Design and Implementation*, 1994.

[2] Brian N. Bershad, Stefan Savage, Przemyslaw Pardyak, Emin Gün Sirer, Marc Fiuczynski, David Becker, Susan Eggers, and Craig Chambers. Extensibility, safety and performance in the SPIN operating system. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, 1995. To appear.

[3] Chao Hsien Lee, Meng Chang Chen, and Ruei Chuan Chang. HiPEC: High performance external virtual memory caching. In *Proceedings of the First Symposium on Operating System Design and Implementation*, pages 153–164, November 1994.

[4] Steven McCanne and Van Jacobson. The BSD packet filter: A new architecture for user-level packet capture. In *Proceedings of the 1993 Winter USENIX Technical Conference*, pages 259–269, Winter 1993.

[5] Jeffrey C. Mogul, Richard F. Rashid, and Michal J. Accetta. The Packet Filter: an Efficient Mechanism for User-level Network Code. In *Proceedings of the 11th Symposium on Operating System Principles*, November 1987.