システムソフトウェアと
オペレーティング・システム 85－2
（２０００．８．４）

UDPを用いた高速通信ライブラリWindの実装

林 章仁† 齋藤 彰一†† 上原 哲太郎†† 國枝 義敏††

†和歌山大学大学院システム工学研究科
††和歌山大学工学部情報通信システム学科

和歌山市栄谷930番地

我々は，分散共有メモリシステムに適した高速通信ライブラリ“Wind”を開発した．WindはLinuxカーネルに変更を加えることで実現されている．Windは，受信したUDPパケットのデータ部分を，直接ユーザプロセスの仮想アドレス空間にコピーすることで，高速化を計っている．これにより，既存のUDPでは受信の時に必要だったシステムコールと，ユーザプロセスにスケジュールを切替えるコンテキストスイッチが不要になる．その結果，Windでは高負荷環境で動いているマシン上においても，パケットの受信を失敗する回数が減少する．本稿では，Windの実装方法と性能評価について述べる．

Wind: a High Performance Communication Module for Software Distributed Shared Memory Systems

Akihito Hayashi† Shoichi Saito†† Tetsutaro Uehara†† Yoshitoshi Kunieda††

†Graduate School of Systems Engineering, Wakayama University
††Faculty of Systems Engineering, Wakayama University
930 Sakaedani, Wakayama 640-8510 Japan

We are developing a high performance communication module named “Wind” for software DSM(Distributed Shared Memory) systems. Wind is built in a Linux kernel, and can copy directly each packet from the buffer for UDP (User Datagram Protocol) into the specified location of virtual address space which assigned for a user process without any context switches. Therefore Wind makes it possible that user processes even on heavy load can receive every packet without loss. This paper describes the implementation and the performance evaluations of Wind.

1 はじめに

近年，1台あたりのPCの高性能化が進んでいる．このようなPCを複数台使用し大規模演算などの1台のマシンでは長時間を要する処理を，高速に実行するシステムとして，PCクラスタが注目されている．PCクラスタにおいて，異なるPC間でデータの共有に用いられる方法として，分散共有メモリがある．大規模演算を対象とする分散共有メモリシステムでは，負荷の高い状態でのネットワークを用いる機会が多い．そこで，効率よく高速に通信することができる通信システムが求められている．

分散共有メモリシステムは，メモリの一貫性を保つためにネットワークを用いた通信を行う．現在，PCクラスタを構築するために用いられるイーサネットは，CPU，主記憶，二次記憶などに比べて速度が遅いことが知られている．よって，分散共有メモリシステムにおける，全実行時間の占める通信時間の割合は少なくない．そこで，我々は分散共有メモリシステムに対して高速なネットワーク処理が行えるライブラリを開発した．このライブラリを以降“Wind”[1]と呼ぶ．WindはLinuxカーネルの内部のUDP(User Datagram Protocol)処理部に変更を加えて実現した．まずWindは，UDPのデータとして送られて来たパケットの中身を参照する．次に、当該パケットの先頭に置かれているWind用のヘッダを参照し，受信プロセスの仮想メモリ空間へ，直接受信したデータをマッピングする．これによって，受信プロセスへのコンテキストスイッチと，システムコールの処理をなくすことができる．Windは，既存のUDPプロトコルよりも受信処理を軽くすることにより，効率よくデータを受け取ることを可能にしている．分散共有メモリシステムにおいて，メモリ内容を転送する処理について考える．分散共有メモリシステムが受け取ったデータはrecv_from()などのシステムコールによって，ユーザプロセスの受信用バッファに格納される．その後，分散共有メモリシステムが適切なアドレスにコピーする．この過程をみると，メッセージを受信した時と，データを分散共有メモリにコピーする時の2回のコピーが発生している．Windを用いた場合，UDPで受け取ったデータを，受信プロセスの仮想メモリ空間へ直接マッピングする．このことにより，2度のコピーを1度のコピーに減らしている．これもWindの利点である．本研究では，我々が開発中の分散共有メモリシステムであるFagus[2]にWindを実装し，並列アプリケーションに応用することが目標である．

本稿では，第2章でWindにおける各メッセージの種類と，各メッセージに対する処理について述べる．第3章では，WindのLinuxカーネル上の実装について述べる．第4章では，Windのインタフェースについて述べる．第5章ではWindを使用して2種類の実験を行い，結果と考察を述べる．そして，最後にまとめについて述べる．

2 Windメッセージタイプ

Windでは既存のUDPのデータ部の先頭に，Wind用のヘッダを埋め込んでいる．Windは，そのヘッダにメッセージタイプを記述し，データの処理方式を切り替えている．本章では，Windのメッセージタイプについて述べる．

Windには5種類のメッセージタイプがある．これらのメッセージタイプの詳細について以下に述べる．なお，Direct Diff DecodeとHeader Diff Decodeは現段階では未実装である．

Direct Map Windは受信したUDPのバッファ内に置かれているデータを，直接受信プロセスの仮想メモリ空間にマッピングする．その後，Wind用のヘッダは破棄する．このため，Direct Map方式では，受信プロセスは，ページを受信したことを知ることができない．

Header Through 基本的な動作はDirect Map方式と同じである．唯一の違いは，Wind用のヘッダがカーネルによって破棄されることなく，ヘッダのみがユーザプロセスに渡されることである．これによって，ユーザプロセスはパケットの受信を知ることができる．

Message Through 既存のUDPプロトコルと同等の処理を行う．この方式では，データとヘッダの両方ともユーザプロセスへ渡される．

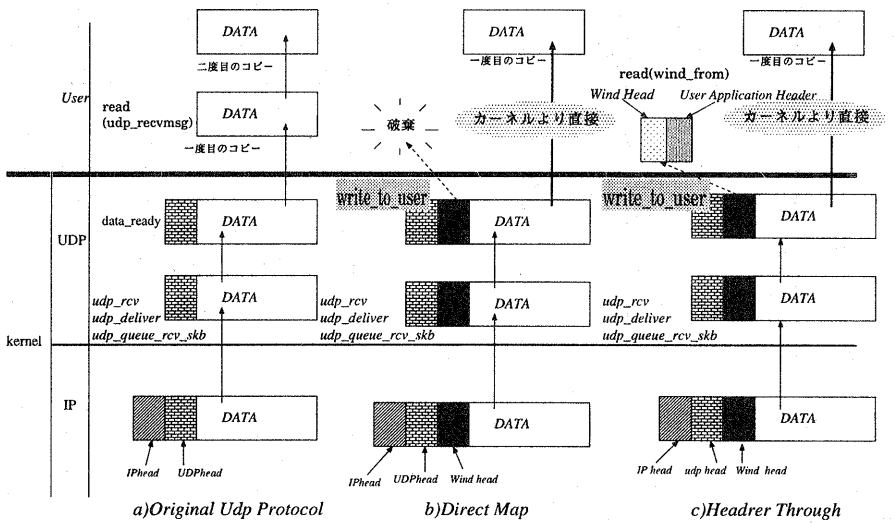

図 1　Windの受信プロセス

Direct Diff Decode このメッセージタイプは，分散共有メモリシステムを特に意識した方式である．Direct Diff Decedeは，Direct Mapがページ単位で転送を行っていたのに対して，ページの更新された部分のみを転送するという特徴をもつ．他の方法は，Direct Mapと同じである．

Header Through Diff Decode Direct Diff Decodeと同様に，ネットワークの転送単位はページではなくページの更新された場所のみである．Direct Diff Decodeとの違いは，ヘッダをユーザプログラムに渡すという点である．

3　Windプロトコル

本章では，Windにおける送受信の処理方式について述べる．Windにおける通信プロトコルは，Windプロトコルと呼ばれるプロトコルで定義されている．

Windプロトコルは 2 つのサブプロトコルで構成されている．1 つは初期化時に使われるPCP(Preparatory Connection Protocol)である．もう 1つは PCPで得たデータを用いてデータを送受信する PTP(Page Transfer Protocol)である．

3.1　Preparatory Connection Protocol (PCP)

PCPは，受信側プロセスから送信側プロセスに対して，受信側プロセスの PIDと分散共有メモリの仮想アドレスを送るためのプロトコルである．これらの情報は，受信プロセスの仮想メモリ空間に対して，Windが受信したページを直接マッピングをするときに必要である．実際に，マッピング処理において必要な情報は，物理アドレスである．マッピングの対象となる仮想アドレスに対応する物理アドレスを求めるためには，プロセス IDと仮想アドレスが必要である [3][4]．Windでは，PCPを用いてこれらの情報を受信側プロセスから送信側プロセスに伝える。PCPは Windの初期化時に

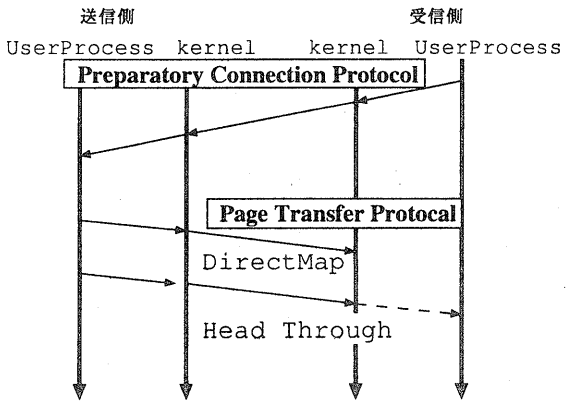

図 2 Windプロトコルの流れ

一度だけ実行すればよい．初期化後は，PCPで受け取った情報を使用できるためである．

3.2 Page Transfer Protocol (PTP)

PTPは，DSM用のページを転送するためのプロトコルである．以下，送信側の処理方式と受信側の処理方式について述べる．

送信処理

主な送信処理は，Windのヘッダの構築である．ヘッダの構成の詳細については，4.2節で述べる．ヘッダを構築するのに必要なプロセスIDと仮想アドレスは，PCPによって受信プロセスから通知される．メッセージタイプは，ユーザが直接選択する．Windヘッダを UDPの先頭に構築することを除いては，既存の UDPと同じ処理方式でデータを送信することができる．

受信処理

ハードウェアによる受信処理から IP層の受信処理までは，既存の UDPを利用した場合と同様である．UDP層において Wind独自の処理が加わる．Windは，Wind用のヘッダを参照し，ヘッダのメッセージタイプ用の処理を行う．Direct Mapの場合は，Windは**write_to_user()**を呼び出す．この関数は，我々が独自に組みこんだ Wind用の関数である．**write_to_user()**はプロセス ID，マッピング対象先の仮想アドレス，受信データへのポインタとデータのサイズを引数にとる．本関数は，プロセス IDと仮想アドレスを用いて，直接，その仮想アドレスに受信データ (UDPヘッダと Windヘッダを除いたもの) をマッピングする．この際，ユーザプロセスへのコンテキストスイッチは発生しない．その後，カーネルが，Wind用のヘッダを破棄して，受信処理を終了する．

次に Header Through方式の場合でも，Direct Map方式同様に，**write_to_user()**を呼び出し，受信データを受信プロセスの仮想アドレスに対してマッピングする．Direct Mapとの違いは，ヘッダをユーザプロセスに受信させることである．このために，ヘッダのみを通常の UDPの処理プロセスに戻し，処理を終える．

Message Through方式は既存の UDPと全くおなじ方法である．そのため Wind用のヘッダと受信データの両方が，ユーザプロセスに渡される．受信データがカーネルによって直接マッピングされることはない．

4 ユーザプログラムインタフェース

Windでは二つのシステムコールと二つのライブラリ関数をユーザに提供している (表 1参照)．ユーザはこれらのシステムコールとライブラリ関数を用いることによって，Windを利用することができる．本章では，これらについて解説する．さらに，Wind用のパケットヘッダの詳細についても説明する．

4.1 システムコールとライブラリ関数

wind_init()は，Windを使用する場合，受信側プロセスにおいて，初めに呼ばれなくてはならないシステムコールである．本システムコールは，受信データを受信プロセスの中でマッピングできる範囲を制限するためのものである．具体的には，Wind宛に送られて来たパケットを処理できるポート番号，プロセス ID，仮想アドレスの開始アドレ

表 1 ユーザインタフェース

ライブラリ関数	使用時のシステム状態
wind_to	メッセージ送信時
wind_from	メッセージ受信時
システムコール	**使用時のシステム状態**
wind_init	Wind 初期化時
wind_close	Wind 終了時

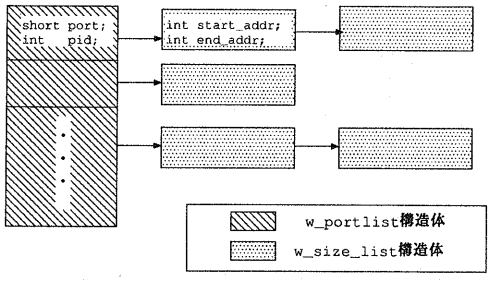

図 3 wind_to の実装におけるデータ構造

ス，終了アドレスが登録される (これらは構造体 w_portlist と w_size_list に格納される．図 3 参照)．登録はするためには，以下の 4 つの条件を満たす必要がある．

(1) 1 つのポートに対して一つのプロセス ID が登録でき，複数のプロセス ID は登録できない．

(2) 1 つのプロセスに対して複数のポートを登録することができる．

(3) マッピング対象となる仮想アドレスの範囲指定の領域は，複数のアドレス空間が指定することが可能である

(4) 登録された条件外のパケットが Wind に対して送られて来た場合は，Wind はそのパケットを処理することなく破棄する．

wind_close() は Wind の使用を終える場合に，呼び出さなくてはならないシステムコールである．

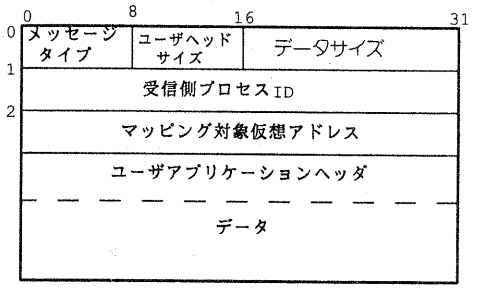

図 4 Wind Header 構造

このシステムコールは，wind_init() において登録された情報をカーネルから削除する．wind_close() はカーネルに登録されている wind に関する情報のうち，呼び出したプロセスのプロセス ID と一致する全ての情報を削除する．

wind_to() は，Wind パケットを送るときに使用するライブラリ関数である．wind_to() では，引数に標準のシステムコールとして提供されている sendto() で使用する引数，メッセージタイプ，受信側のプロセス ID，開始仮想アドレス，Wind 用のヘッダのサイズを引数にとる．wind_to() は送信をするユーザとのインタフェースである

wind_from() は，Wind を用いてデータを受信するときに使用するライブラリ関数である．wind_from() は引数として，recvfrom() の全ての引数と，メッセージタイプが必要である．wind_from() は受信を行うユーザとのインタフェースである．

4.2 Wind ヘッダ構造

Wind を使用する場合，ユーザは Wind 用のヘッダを構築し，パケットの先頭に配置しなければならない．本節では Wind ヘッダの詳細について述べる．図 4 に，ヘッダ構造を示す．以下に，ヘッダの各データについて説明する．

メッセージタイプ

メッセージの型はユーザが自由に選ぶことができる．メッセージタイプの詳細は第 2 章で述べた．

ユーザアプリケーションヘッダサイズ

表 2　実験環境

CPU	Celeron400MHz
Main Memory	128M
OS	Linux Kernel 2.2.16(Wind 用に変更)
Network	100Mbps Ethernet with Switching Hub
Network Interface Card	EtherExpressPro

表 3　1 台のマシンを用いての実験

メッセージタイプ	送信パケット数				紛失したパケット数
	4096	8192	16384	32768	
Direct Map	8.76	17.53	35.10	70.28	0
Header Through	9.75	19.58	39.29	78.66	39
Message Through	11.10	22.41	44.98	90.07	126

ユーザアプリケーションヘッダサイズは，Wind ヘッダの次に配置されるユーザアプリケーションが使用するヘッダのサイズである．ユーザアプリケーションヘッダとは，ユーザが独自に使用するヘッダである．このヘッダは，Header Through 方式においてユーザプログラムに渡されるヘッダである．Wind は，このヘッダに対する制限は行っていない．

データコピーサイズ
仮想メモリ空間にマッピングするデータの大きさ．

受信側プロセス ID
(PCP で得られる情報の 1 つである)
物理アドレスを求めるのに用いる．詳細は第 3 で述べた．

マッピング対象仮想アドレス
(PCP で得られる情報の 1 つである)
物理アドレスを求めるのに用いる．詳細は第 3 で述べた．

5　実験

本章では，Wind の性能を測定する実験を行い，その結果と結果について述べる．なお，実験環境を表 2 に示す．我々が行った実験は，全部で 2 種類である．2 つの実験内容を以下に示す．なお，複数台での通信の場合，受信するマシンは受信のみを行い，送信することはない．

実験 1　1 台と 2 台のマシンでそれぞれ通信を行い，8192 個のパケット (1 パケット 60KByte) を送り終えるまでの時間と，受信に失敗をしたパケット数を測定する．

実験 2　複数台のマシン (最大 9 台) を用い，高負荷の場合と低負荷の場合とで，パケットの受信性能について測定する．

実験 1 に関して表 3 を見ると，Direct Map と Header Through の両方の方式の結果は，Message Through に比べて高速であることが分かる．さらに，パケットの受信失敗もないことがわかる．一方，表 4 において 2 台の PC で通信を行った場合は，Header Through の性能はほとんど Message

表 4　2台のマシンを用いての実験

メッセージタイプ	送信パケット数				紛失したパケット数
	4096	8192	16384	32768	
Direct Map	20.84	41.77	83.67	167.00	0
Header Through	20.94	41.86	83.77	167.56	2
Message Through	20.94	41.88	83.77	167.74	2

表 5　受け取ったパケット数

メッセージタイプ	負荷	送信マシン数			
		2	4	6	8
Message Through	light	10867	8712	8614	8876
	heavy	8729	7421	7276	5588
Direct Map	light	10515	11960	8671	8765
	heavy	10992	10576	7649	5590

Through と同じである．実験 1 の 1 台の場合は，システムコールの処理を必要とせず，さらにコンテキストスイッチの切替えと，受信プロセス内でのコピーが不要なことによって，効率の良い高速なネットワーク転送が実現されている．一方，実験 1 の二台での場合は，2 台の PC での通信が発生し，通信にかかる時間が大きくなる．そのため，システムコールの処理，コンテキストスイッチの切替え，ユーザプロセス内でのコピーが不要といった処理の割合が小さくなってしまい，性能の向上がみられなかった．

実験 2 では，低負荷環境と高負荷環境の 2 種類の環境において実験を行った．これは，分散共有メモリシステムにおける処理では，各 PC は常に，高負荷な環境で動いていると考えられる．そのような環境を想定し，高負荷環境における実験を行った．ここでの負荷は Linux Kernel を実際に構築して与えている．表 5 によると，Direct Map は，既存の UDP と比べると，負荷が大きい状態において既存の UDP よりもより多くのパケットを受信できている．2 台と 4 台の PC が送信する場合は，処理が軽くなり Wind の性能が発揮されているといえる．しかし，6 台と 8 台の場合には，既存の UDP と Wind を使用した時の差がほとんどなくなっている．これは，NIC(Network Interface Card) の受信性能限界に達し，Wind の性 能を生かしきれていないためだと思われる．しかし，さらなる分析が必要だと考えられる．

6　おわりに

本稿では，高速通信ライブラリ Wind について述べた．Wind は既存の UDP と比べ，より効率よくデータを運ぶことができる．特に，受信側のマシンの負荷が大きくなった時は既存の UDP と比べて高性能となることがわかった．これは，システムコールを呼び出す必要がないため，Direct Map がユーザプロセスとカーネル間でのスケジューリングの切替えを必要とせず，ソフトウェア割り込みが削除できるためである．さらに，データのコピー回数が 1 回減ることも，Wind の効率の良い通信を実現している．この結果，Wind を用いることにより分散許勇メモリシステムの実行速度の向上が見込まれる．

我々は，この Wind を，我々の研究グループが開発中の分散共有メモリシステム Fagus に実装する予定である．さらに，Wind の拡張として，Wind

単体で ACK を返し，メッセージの到達保証性を確保する機能を実現する予定である．さらに，デバイスドライバなどを改良することで，さらなる高速化が期待できると思われる．本研究グループではこれらを対象に研究を進めていきたい．

参考文献

[1] Shoichi Saito, Akihito Hayashi, Tetsutaro Uehara, Kazuki Joe and Yoshitoshi Kunieda, a Low-cost Comminication Module for Software DSM Systems, In Proceeding of the International Conference on PDPTA '2000, pp. 721-727 (2000).

[2] 横手 聡，齋藤 彰一，上原 哲太郎，國枝 義敏，コンパイラによる制御可能な DSM システム「Fagus」の実現，情報処理学会，システムソフトウェアとオペレーティングシステムの研究会 SWoPP2000 発表予定．

[3] R. Card, E. Dumas and F. Mevel, Linux2.0 カーネルブック，オーム社出版局 (1999).

[4] Michel Beck, Harald Bohme, Mirko Dzidzka, Robert Magnus, Ulrich Kunitzand and Dirk Verwoner, Linux カーネルインターナル，株式会社アスキー (1999).