社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－OS－101 (2)
2006／2／16

User Mode Linux 上における高速通信機構

仲亀 渉† 石川 裕†

本論文では User Mode Linux(UML) の通信性能を改善する手法 Hnet system を提案し、評価する。Hnet system とは、TCP/IP や UDP/IP プロトコルハンドリングをゲスト OS 上ではなく、ホスト OS 上で行なう手法である。さらに、Hnet system の性能改善のため 2 つの手法 Short-term Busy Wait(SBW) と Direct Memory access between Host and User(DMHU) を提案し評価する。SBW は、プロトコルハンドリング完了通知メッセージを減らすため、ゲスト OS がホスト OS をポーリングする手法である。DMHU は、UML ユーザプロセスが UML カーネルに対してシステムコールを発行する際に発生するコピーを減らす手法である。

提案手法の評価のため、2 台のマシンの間における通信遅延と通信バンド幅を測定した。提案手法を実装した UML の通信性能は、オリジナルの UML の通信性能と比較して、ピーク通信バンド幅は 3.77 倍、ラウンドトリップタイム (RTT) は 48%である。また、SBW の評価結果から、ポーリング期間は 1 jiffies が適切である。同時に、SBW においてポーリング期間が 1 jiffies であれば UML 上の他のユーザプロセスの性能を劣化させないことを示す。

The High Performance Communication Mechanism in User Mode Linux

WATARU NAKAGAME† and YUTAKA ISHIKAWA†

The *Hnet system* is proposed and evaluated in order to improve the performance of communication in the User Mode Linux(UML). In the Hnet system, TCP/IP and UDP/IP protocol handling is performed in the host OS instead of the guest OS. In addition, two methods, *Short-term Busy Wait(SBW)* and *Direct Memory access between Host and User(DMHU)*, are proposed and evaluated in order to improve the Hnet system. In the SBW, the guest OS polls the incoming data buffer from the host OS in order to reduce the notification messages. In the DMHU, extra copies of memory are reduced. These copies are caused when user process in UML calls system call such as write system call and read system call.

To evaluate the efficiency of the proposed system, the performance of communication between two machines is evaluated. The peak bandwidth of the proposed mechanisms outperforms 3.77 times faster than that of the original UML. The round trip time between the proposed mechanisms outperforms 48% less than that of original UML. The polling span in the SBW is evaluated and it is shown that 1 jiffies is the best span. Furthermore, it is shown that the SBW does not deteriorate the performance of other process in the UML.

1. はじめに

計算機の仮想化技術は過去 30 年に渡って研究されてきた。仮想化には様々な方式があるが、User Mode Linux(UML) [1,2] は仮想 OS 型である。仮想 OS 型の他の例としては、Virtuozzo [6] が挙げられる。仮想 OS 型とは異なる仮想化技術の例として、Xen [3,4] と VMware [5] がある。VMware は、ゲスト OS が必要とするデバイスをホスト OS 上に仮想デバイスとして実装する方式である。Xen は、コンピュータリソースを仮想化して管理する方式である。仮想化されたリソースを利用出来るようにゲスト OS の API を変更し、ゲスト OS へ適切にリソースを配分する。

仮想 OS 型の特長は、リソースやハードウェアを仮想化するのではないという点である。ユーザプロセスはディスク等のデバイスにアクセスする際には、必ずシステムコールを経由するという性質がある。仮想 OS 型は、ページフォルト等の例外処理やシステムコールをトラップし適切に処理することで、ユーザプロセスがあたかもホスト OS とは別の OS 上で動作しているように見せかける技術である。UML では、これらのトラップには ptrace システムコールを用いている。

UML は通信性能が低いことが知られている [3]。本論文ではこの原因を調査し通信性能を向上させる手法を提案し、評価する。まず最初に、オリジナルの UML を調査し、頻繁にシグナルハンドラが呼ばれることがボトルネックとなっていることを明らかにする。次に、

† 東京大学情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo

シグナルハンドラの呼び出し回数を減らすための手法を検討し、以下の手法を提案する。

(1) TCP/IP や UDP/IP プロトコルハンドリングをゲスト OS 上ではなく、ホスト OS 上で行なう

(2) プロトコルハンドリング完了通知メッセージを減らすため、ゲスト OS がホスト OS をポーリングする

(3) システムコール時のコピー回数を減らす

最後に、提案手法を設計・実装し、評価する。

2. 考　　察

2.1 オリジナル UML の問題点

オリジナルの UML の通信バンド幅を一方向バースト通信テストで評価した。本テストは、ある送信プロセスがある受信プロセスへバーストでデータ転送を行なうテストである。図 1 は、送受信プロセスがオリジナルの UML 上にある場合と、送受信プロセスがホスト OS 上にある場合の通信バンド幅を測定したものである。**bufsize** とは、送受信プロセスが read システムコールや write システムコールに渡すバッファのサイズである。

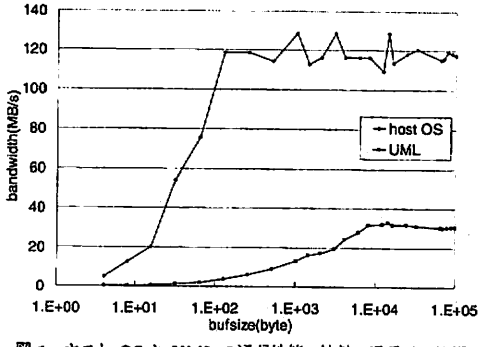

図 1　ホスト OS と UML の通信性能の比較：通信バンド幅

ホスト OS のピーク通信バンド幅は 117MB/s であるのに対し、オリジナル UML では 31MB/s である。

次に、バースト ping-pong テストで Round Trip Time(RTT) を測定する。バースト ping-pong テストとは、インターバルを挟まず可能な限り早く ping-pong を行なうテストである。本テストの結果を表 1 に示す。

表 1　ホスト OS と UML の通信性能の比較：通信遅延

UML (usec)	ホスト OS (usec)
427	143

UML のネットワーク性能は低いが、その原因を調査するため、通信においてボトルネックとなる場所を調査した。ネットワーク上にチェックポイントを設定し、そのポイントでの時刻を記録することでオーバーヘッドを測定した。時刻には、1 クロック毎にカウントアップする 64 ビットの CPU カウンタの値を使用した。調査結果を図 2 に示す。

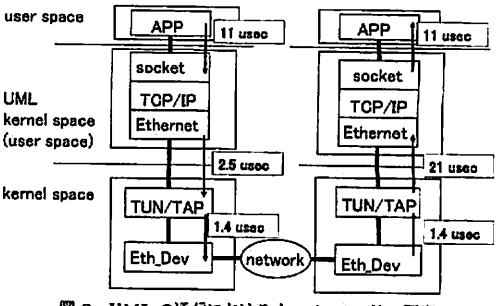

図 2　UML の通信におけるオーバーヘッドの調査

図 2 における TUN/TAP とは、UML 上の NIC の実体である。TUN/TAP はホスト OS 上ではイーサネットデバイスとして扱われている。TUN/TAP デバイスに届いたパケットを UML カーネルが読み出すオーバーヘッドは 21usec であり、最も大きなオーバーヘッドである。本論文では、このオーバーヘッドをシグナルハンドラオーバーヘッドと呼ぶ。TUN/TAP デバイスは自身にパケットが到着するとすぐに SIGIO を UML カーネルに投げ、UML カーネルのシグナルハンドラ内でパケットを読み出す。シグナルハンドラオーバーヘッドは単独でも大きなオーバーヘッドである上に、SIGIO はパケット到着の度に投げられる。通信時には頻繁にシグナルが発生することになり、結果として通信性能が低下する。

2.2 改善手法の検討

本節では UML 上の通信性能を向上させるための方法を検討する。ホスト OS のカーネルを変更する手法には、ホスト OS のカーネルがアップデートされる度に対応しなければならないという問題がある。よって、本論文ではホスト OS のカーネルは変更せず、ホスト OS のローダブルモジュールで対応可能な範囲に限定して解決方法を検討する。

前節で述べたように、ネットワーク性能を向上させるためには、シグナルハンドラオーバーヘッドを改善しなければならない。これを解決するために 2 つの方法が考えられる。

(1) ホスト OS のスケジューラを改良して、シグナルハンドラの応答性を上げる

(2) シグナルハンドラの呼び出しを最小限にする

しかし、方法 1 を実現するためにはホスト OS を変更することになるため、最初に述べた前提を考慮すると適切な方法とは言えない。よって、方法 1 は本論文では採用しない。

方法2は、UMLカーネルの機能の一部をホストOS上で行なうことで実現可能である。UMLカーネルがプロトコルハンドリングを終えたデータだけを受け取るようにすればよい。すなわち、TCP/IPやUDP/IPプロトコルハンドリングをゲストOS上ではなくホストOS上で行なう。ホストOSでは、ハードウェア割り込みとタスクレットにより、パケットはUML上よりも効率的に処理することが可能である。UMLカーネルはホストOSから見れば1プロセスに過ぎないので、ユーザプロセスが利用できるホストOSの機能はシステムコールを通じて全て利用可能である。つまり、ホストOSを一切変更しないでTCP/IPやUDP/IPを利用可能である。本論文では方法2を選択し、設計、実装、評価を行なった。

3. 設計と実装

第2章で述べたように、UMLのネットワーク機能の一部をホスト側で扱い、UMLの通信性能を向上させる手法を設計し実装する。本論文では本手法をHnet systemと呼ぶ。Hnet systemは2つの特長がある。

- TCP/IPやUDP/IPといったプロトコルハンドリングをホストOS上で行ない、ゲストOSはハンドリングが終ったデータをゲストOS上のユーザプロセスに渡すための仲介をする
- Hnet systemはホストOSの変更の必要はなく、オリジナルのUMLが動く環境であればHnet systemは適用可能である

さらに本論文ではShort-term Busy Wait(SBW)とDirect Memory access between Host and User process(DMHU)という2つの手法を提案する。SBWは、プロトコルハンドリング完了通知が大量に発生するというHnet systemの問題を改善するための手法である。DMHUはHnet systemにより発生するメモリコピーを減らすための手法である。

3.1 Hnet systemの基本設計の概要

図3はTCP/IPのプロトコルスタックを示している。左側がHnet systemであり、右側がオリジナルのUMLのプロトコルスタックである。

UMLユーザプロセスがあるソケットをUML上に生成する際に、対応するもうひとつのソケットがホストOS上に作られる。本論文では、UMLユーザプロセスとUMLカーネル間のインターフェースとしてのソケットをUsocketと呼び、UMLカーネルプロセスとホストOSの間のインターフェースとしてのソケットをHsocketと呼ぶ。Usocketの実体はUMLカーネル空間内にあり、Hsocketの実体はホストOSのカーネル空間内にある。

UMLカーネルはUMLユーザプロセスとホストOSの仲介役のような存在であり、UMLユーザプロセスが呼んだシステムコールに対応して、UMLカーネルがホストOSのシステムコールを呼ぶ。例えばwriteシステムコールの場合は、UMLユーザプロセスはUsocketを用いてUMLカーネルに対してwriteシステムコールを発行する。さらに、UMLカーネルプロセスは適切なHsocketを用いてホストOSに対してwriteシステムコールを発行する。本論文ではこれら2つのシステムコールを区別するため、UMLユーザプロセスが呼ぶシステムコールにプレフィックスとしてU_をつけることにする。本論文では、U_wirteシステムコール、U_readシステムコールの実装についてのみ概要を説明する。

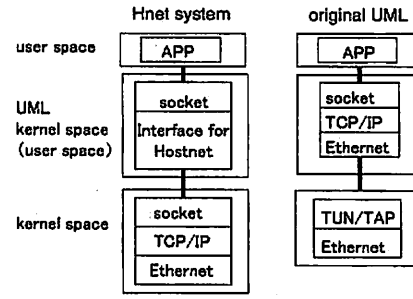

図3 プロトコルスタック

Hsocketに対する操作はノンブロッキングでなければならない。UMLカーネルプロセスは内部でホストのシステムコールを呼び出すことが可能であるが、その際はノンブロッキングでないと、ブロックが解除されるまでUMLカーネル全体が止まってしまう。Usocketに対する操作がブロッキングであり、かつ対応するHsocketが利用可能でなかった場合、UMLカーネルはシステムコール呼び出し元のプロセス(UML上のユーザプロセス)をシステムコール内でスリープさせる。ホストOSは、Hsocketが利用可能になったことをSIGHNETシグナルによってUMLカーネルに通知する。SIGHNETシグナルはHnet system専用に使われるシグナルで、UMLカーネルが使用していないシグナルであればよい。UMLカーネルには、自身が受け取ったシグナルをIRQ信号として扱う機構が存在する。SIGHNETシグナルを受信した際には、UMLカーネルはHNETIRQ番のIRQ信号として扱うことにする。HNETIRQ番に対応するIRQハンドラを用意し、スリープしているプロセスはこのハンドラが起床させる。

3.1.1 U_write

U_writeシステムコールの処理の概要は、以下のようになる。Hsocketはすでにconnect/accept済みであるものとする。Hsocketに対してwriteシステムコールを発行した際の返り値がEAGAINだった場合、Hsocketのバッファが一杯で書き込み不能状態であることを示す。

(1) カーネル空間内にバッファkbufを用意し、UMLユーザ空間のバッファubufのデータをkbufに

コピーする

(2) write システムコールを用いて kbuf を送信するループ
- もし write システムコールが正の値 n を返したら、ssize にその値を加える
- もし write システムコールが EAGAIN を返し、Usocket がブロッキングだったら、呼び出し元のプロセスをスリープさせる
- もし Usocket がノンブロッキングで、write システムコールの返り値 n が負の値だったら、n を返り値として呼び出し元に戻る
- ssize が ubuf のサイズになるまで、ループをくり返す

(3) バッファを解放し、ssize を返す

3.1.2 U_read

U_read システムコールの処理の概要は、以下のようになる。Hsocket はすでに connect/accept 済みであるものとする。Hsocket に対して read システムコールを発行した際の返り値が EAGAIN だった場合は、Hsocket のバッファに読み出せるデータが存在せず、読み込み不能状態であることを示す。

(1) カーネル空間内にバッファ kbuf を用意する

(2) read システムコールを用いて、kbuf にデータを読み込むループ
- もし read システムコールの返り値が正の値 n を返したら、rsize に n を加え、kbuf の内容を ubuf の適切な場所にコピーする
- もし read システムコールの返り値 n が EAGAIN で Usocket がノンブロッキングだったら、そのまま n を返り値として呼び出し元に戻る
- もし read システムコールの返り値 n が EAGAIN であり Usocket がブロッキングであったら
  - rsize が 0 だったら、呼び出し元のプロセスをスリープさせる
  - rsize が正の値だったら、その値 rsize を返り値として呼び出し元に戻る
- rsize が ubuf のサイズになるまで、ループをくり返す

(3) バッファを解放し、rsize を返り値として呼び出し元に戻る

TCP/IP のデータはストリームであるので、受信側はコネクション切断以外の方法ではデータの切れ目を知ることが出来ない。よって U_write システムコールとは異なり、U_read システムコールは 1 バイトでも読み出したら、バッファサイズ分に満たない量の受信しか出来なかった場合でも、受信の完了を待たずに呼び出し元に戻る。

3.2 Short-term Busy-Wait(SBW)

I/O 待ちの場合、一般的にスリープせずにポーリングしたほうが高速化できる反面、ポーリング中は他のプロセスを動かすことが出来ない。逆に割り込み駆動型は、I/O 待ちをしているプロセスがスリープするため、その空き時間で他のプロセスが仕事をすることが可能であるという利点があるが、性能ではポーリングに劣る。

Hnet system だけでは受信の通信性能は改善しきれていない。バースト通信時には、スリープしないでも少しの間ポーリングで待てばすぐにデータが届くはずである。スリープしていない時は SIGHNET シグナルを止め、さらにスリープする条件が揃ってもすぐにはスリープせず、一定期間はポーリングで待つ方法が有効である。ただし、長い期間ポーリングすると他のプロセスに対しての影響が大きいので、最小限の時間だけポーリングするべきである。以上の方針に基づいて短期間ポーリングを行なう手法を Short-term Busy Wait(SBW) と呼ぶ。SBW は割り込み駆動型とポーリングの利点を組み合わせた手法であると言える。

次に、SBW の処理の流れを説明する。U_read システムコールや U_write システムコールのループに入る前に、その時点での UML カーネルにおける jiffies を保存する (その値を s_ji とする)。jiffies とは、Linux がカーネル内部で利用する時間の単位であり、起動直後からタイマ割り込み毎にカウントアップされる値である。本論文で扱う jiffies は全て UML カーネル内の値としての jiffies であり、ホスト OS の jiffies ではない。また、U_read システムコールや U_write システムコールのスリープする部分の処理を以下の処理に置き換える。

- もし jiffies - s_ji が一定期間より小さければ、schedule 関数を呼び出して呼び出し元プロセス (U_read や U_write を呼んだ UML 上のユーザプロセス) の CPU 使用権を放棄し、再び UML カーネルのスケジューラによって CPU の使用権を割り当てられたら処理を再開する
- もし jiffies - s_ji が一定期間より大きければ、以下の手順でスリープする
  - Hsocket から SIGHNET シグナル通知がされるようにする
  - 呼び出し元プロセス (UML 上のユーザプロセス) をスリープさせる
  - Hsocket からの SIGHNET シグナル通知を無効化する

単にポーリングするのではなく schedule を呼び出して一旦 CPU を放棄することで、ポーリングしながら他のプロセスにも CPU を渡すことが可能である。

3.3 Direct Memory Access between Host and User process(DMHU)

Direct Memory access between Host and User process(DMHU) はメモリコピーを減らす手法である。U_read システムコールや U_write システムコールでは、ホスト OS と UML ユーザプロセスは直接データをコピーできず、UML カーネルプロセスへのコピーが必要であった。DMHU は、UML ユーザプロセスのメモリが UML カーネルプロセスのヒープ領域にある

という性質を利用し、直接ホスト OS と UML ユーザプロセスが直接データ交換をできるようにするための手法である。本論文では U_write システムコールについての DMHU のみ説明する。U_read システムコールは U_write システムコールと同様の手法で実現可能である。図 4 は通常の U_write システムコールにおけるデータの流れを表している。

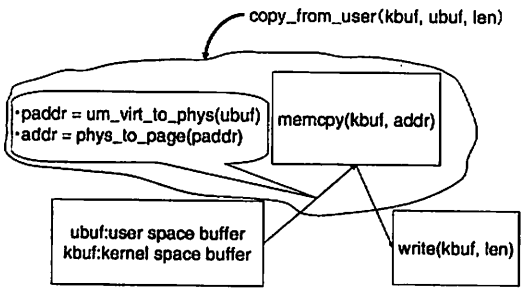

図 4 Hnet system の write

copy_from_user は、ユーザ空間のアドレスをチェックをしながらメモリをコピーする関数である。メモリコピー部分をホストへのシステムコールで置き換えればよい。図 5 が置き換えた後のデータの流れを表している。

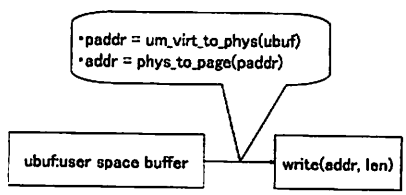

図 5 DMHU による Hnet system の write

ユーザ空間のメモリはページ単位で割り当てられているので、システムコールに渡すことの出来るデータサイズは 1 ページ分 (4KB) に限られる。バッファサイズが 4KB 以上の場合、4KB 毎に分割してシステムコールに渡す必要があるため、システムコール呼び出し回数は増える。しかし、第 2.1 節の図 1 の結果から、バッファサイズが 4KB であれば通信バンド幅はピークになるため、4KB 毎に分割することに問題はない。

4. 評　　価

4.1 評価環境

Gigabit の NIC を持つ 2 台のマシンをスイッチで繋ぎ、通信遅延と通信バンド幅の測定をした。マシンの仕様を表 2 に示す。

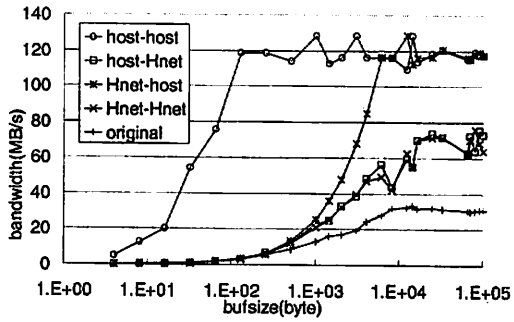

図 6 Hnet system の通信バンド幅

表 2 マシンの仕様

Processor	Dual Xeon 2.80GHz
Memory	Xeon 1GByte
Network	Intel Pro1000 Gigabit Ethernet × 2
OS	Linux 2.6.11-skas3-v8

4.2 Hnet system の基本実装の評価

最初に、第 3.1 節で説明した Hnet system の U_read システムコールと U_write システムコールの通信バンド幅を、一方向バースト通信テストで測定した。本テストは、ある送信プロセスがある受信プロセスへバーストでデータ転送を行なうテストである。送信プロセスと受信プロセスを動作させる環境を様々に変えて測定した結果が図 6 である。例えば図中の host-Hnet は、送信プロセスがホスト OS 上にあり受信プロセスが Hnet system の UML 上にある場合を表しており、他も同様である。また、original はオリジナルの UML 間の通信バンド幅である。

ホスト OS が受信側になる場合は、ピーク通信バンド幅は 117MB/s であるが、Hnet が受信側になる場合はピーク通信バンド幅は 71MB/s である。Hnet system は受信において問題がある。

次に、第 3.1 節で説明した Hnet system の U_read システムコールと U_write システムコールの RTT を、バースト ping-pong テストで測定した。送信プロセスと受信プロセスを動作させる環境を様々に変えて測定した結果が表 3 である。ping 側と pong 側を入れ換えても結果は当然同じである。例えば Hnet-host の結果は host-Hnet の値と同じとなる。

表 3 Hnet system の RTT

host-host	Hnet-Hnet	org-org
146 usec	264 usec	427 usec

Hnet system による通信遅延は、オリジナルの UML の通信遅延と比較して 62%(= 264/427 * 100) に改善

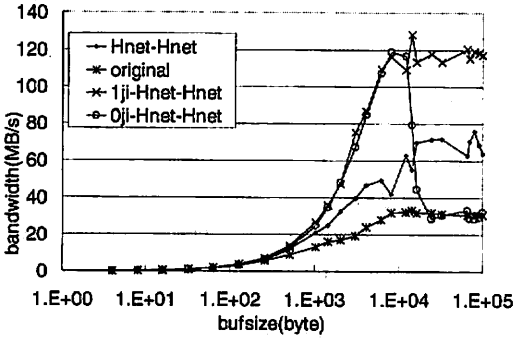

図 7 SBW を用いた場合の通信バンド幅

されている。

4.3 SBW の評価と考察

第 3.2 節で提案した SBW の通信バンド幅を評価した結果が図 7 である。測定条件は第 4.2 節の一方向バースト通信テストと同様である。

1ji とは、1jiffies の期間ポーリングするという意味である。また、0ji というのは、ポーリングせずにすぐにスリープする場合である。

ピーク通信バンド幅については、0ji、1ji 共に 117MB/s であり、受信プロセスが host である場合と同じである。これはスリープの回数を十分に減らすことが出来たためである。0ji 以外ではバッファサイズが 12000byte を越えても通信バンド幅は低下しないが、0ji ではピーク通信バンド幅となるバッファサイズは 6000byte から 12000byte であり、これを過ぎると通信バンド幅は逆に低下する。原因は、バッファサイズが大きすぎると一度では通信が完了出来ずにスリープする場合が多くなるからである。スリープする準備にコストがかかる上に、スリープしたプロセスを起こすために SIGHNET シグナルが必要である。これらのコストのため、頻繁にスリープすると通信性能が低下する。

SBW の通信遅延を評価した結果が表 4 である。測定条件は第 4.2 節と同様である。また、ping-pong テストの試行回数は 20 万回である。SIGHNET の回数とは HNETIRQ ハンドラが呼び出された回数に等しく、これは ping(或いは pong) プロセスがテスト中にスリープした回数とも等しい。表 4 は ping プロセスにおいての結果である。

ping-pong テストでは、1ji から 10ji までは差はほとんどないが、0ji は他に比べて 64usec 劣っている。ping-pong テストでは送るデータサイズは 4byte と小さいので、write システムコールでブロックすることはない。ping プロセスは pong プロセスの応答を待つ際にブロッキング read システムコールで待ち、pong

表 4 SBW を用いた場合の RTT

ポーリング期間 (jiffies)	RTT	SIGHNET の回数
0	292 usec	199983
1	226 usec	565
2	226 usec	554
3	228 usec	468
4	229 usec	229
5	228 usec	71
6	229 usec	23
7	234 usec	16
8	225 usec	5
9	225 usec	2
10	228 usec	1

プロセスは ping プロセスのメッセージをブロッキング read システムコールで待つ。よって一往復あたり 2 回のブロッキング read が起こっている。SBW を有効にしている場合は直にはスリープせずにポーリングしている。1ji から 10ji における RTT の平均は 228usec であるので、スリープ一回のオーバーヘッドは 32(= (292 − 228)/2) usec である。

SBW が他のプロセスに対してどの程度影響するかを調べるテストを行なった。受信プロセスのある UML 上に、浮動小数点計算のみを行なう CPU バーストプロセスを同時に走らせる。送信側のプロセスが転送量を絞って同じ通信バンド幅になるようにして通信を行ない、その際に CPU バーストプロセスがどれだけ実行されたかを調べた。本テストを負荷テストと呼ぶ。オリジナルの UML、SBW なし、SBW ありの 3 つについて調べた結果が表 5 である。それぞれの通信バンド幅を同じにするため、第 4.3 節の評価から、バッファサイズは 8192byte とし、通信バンド幅は 10MB/s となるように調整した。CPU バーストのみを動作させた場合は 554MFlops である。よって、通信による負荷は 554 から CPU バーストプロセスの性能を引いた値である。

表 5 SBW を用いた場合の負荷テスト

ポーリング期間 (jiffies)	CPU バーストプロセスの性能
0	491 Mflops
1	489 Mflops
10	475 Mflops

また、オリジナルの UML について同様のテストを行なった結果、CPU バーストプロセスの性能は 380 Mflops であった。負荷テストの結果から 2 つのことが言える。1 つめは、Hnet system はオリジナルの UML と比較して負荷が軽くなっている。負荷は 1ji の場合 36%(= (554 − 489)/(554 − 380) * 100) に低下している。10ji のポーリングでも、オリジナルの UML よりは負荷は小さい。2 つめは、ポーリング期間が長くにつれて負荷が大きくなるが、1jiffies であれば問題になるほど大きな負荷ではないということである。0ji と 1ji の通信による負荷の差は 3%(= (1 − (554 − 491)/(554 − 489)) * 100) である。

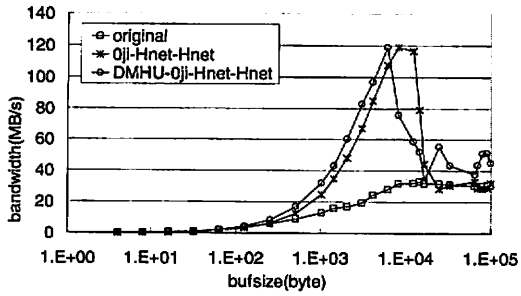

図 8 0ji において DMHU を用いた場合の通信バンド幅

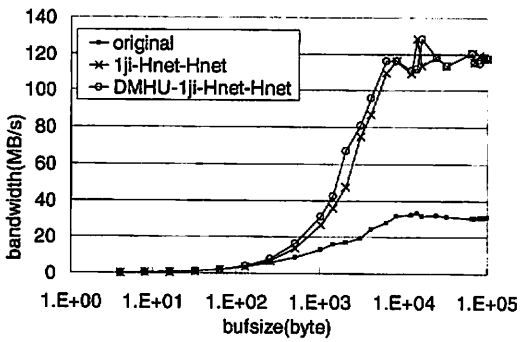

図 9 1ji において DMHU を用いた場合の通信バンド幅

以上から、SBW はピーク通信バンド幅を改善するのに有効な手法である。また、ポーリングの期間は出来るだけ短い方がよく、1jiffies が適切である。

4.4 DMHU の評価と考察

第 3.3 節で説明した DMHU の通信バンド幅を測定した結果が図 8 と図 9 である。測定条件は SBW の評価時と同様である。また、図 10 はバッファサイズが 512byte 以下の場合の結果を抜き出したものである。ピークの通信バンド幅は DMHU の有無に関わらず 117MB/s であるが、バッファサイズが小さい場合に DMHU の効果が確認できる。例えばバッファサイズが 512byte の場合、DMHU により通信バンド幅は 18%(= (1 − 16.5/14) * 100) 改善される。

DMHU の通信遅延を測定した結果が表 6 である。DMHU によって、一度の一方向通信につき 2 回メモリコピーを減らすことが出来る。一方は U_write システムコールにおける UML ユーザプロセスから UML カーネルへのコピーであり、多方は read システムコールにおけるホストから UML カーネルへのコピーである。DMHU を使うことで RTT は平均 19 usec 小さくなっている。オリジナルの UML の RTT は 427usec

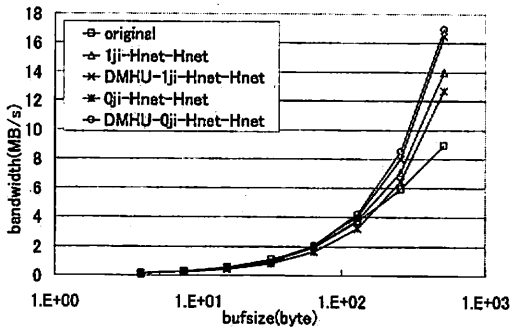

図 10 図 8 と図 9 のバッファサイズが 512 以下の部分を拡大したグラフ

であるので、オリジナルの UML と比較して RTT は 48%(= 205/427 * 100) に改善された。

表 6 DMHU の通信遅延

ポーリング (jiffies)	DMHU RTT	non-DMHU RTT
0	280 usec	292 usec
1	212 usec	226 usec
2	204 usec	226 usec
3	201 usec	228 usec
4	212 usec	229 usec
5	216 usec	228 usec
6	201 usec	229 usec
7	209 usec	234 usec
8	201 usec	225 usec
9	214 usec	225 usec
10	208 usec	228 usec

5. 結　　論

本論文は UML の通信性能を向上させることを目的とし、TCP/IP や UDP/IP プロトコルハンドリングをゲスト OS 上ではなく、ホスト OS 上で行なう手法 Hnet system を提案し実装した。ホスト OS では、ネットワークパケットはハードウェア割り込みとタスクレットの機構により効率的にハンドリングされる。さらに、Hnet system の性能改善のため 2 つの手法 SBW と DMHU を提案し実装した。SBW は、プロトコルハンドリング完了通知メッセージを減らすため、ゲスト OS がホスト OS をポーリングする手法である。ポーリングは CPU リソースを消費するので、ポーリング期間は適切な長さを選択する必要がある。SBW の評価から、ポーリング期間は 1jiffies が適切である。DMHU は、UML ユーザプロセスが UML カーネルに対してシステムコールを発行する際に発生するコピーを減らす手法である。

提案手法の評価のため、2 台の PC 間における通信

遅延と通信バンド幅を測定した。提案手法を実装したUMLの通信性能は、オリジナルのUMLの通信性能と比較して、ピーク通信バンド幅は3.77倍、ラウンドトリップタイム(RTT)は48%であった。また、SBWの評価結果から、ポーリング期間は1 jiffiesが適切である。同時に、SBWにおいてポーリング期間は1 jiffiesであればシステム全体の性能が劣化しないことを示した。

謝辞 本研究の一部は文部科学省「eSociety基盤ソフトウェアの総合開発」の委託による。本研究を進めるにあたって、忙しい中貴重な時間を割いて協力してくれた石川研究室の皆様への感謝の意をここに表します。

参考文献

1) Jeff Dike. skas mode. http://user-mode-linux.sourceforge.net/skas.html.
2) Jeff Dike. A user-mode port of the linux kernel. In *In Proceedings of the 4th Annual Linux Showcase and Conference*, 2000.
3) B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the art of virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles*, October 2003.
4) K.Fraser, S.Hand, R.Neugebauer, I.Pratt, A.Wargield, and M.Williamson. Safe hardware access with the xen virtual machine monitor.
5) S.Devine, E.Bugnion, and M.Rosenblum. Virtualization system including a virtual machine monitor for a computer with a segmented architecture. *US Patent, 6397242*, Oct 1998.
6) SWsoft. *Product Overview: Virtuozzo*, 2005. http://www.swsoft.com/virtuozzo/.