Linuxにおけるプロセス優先度に基づく受信処理の実現

尾崎亮太[†] 中山泰一[††]

インターネットの発展とユーザの増加にともない，サーバシステムに対して高い処理性能や安定性などの要求が高まっている．最近ではサービスの重要度に従ったサーバの資源割当ての制御がサーバシステムに対する新たな要求となっている．Unix 系 OS では，優先度を用いて個々のプロセスに割り当てる資源を調節している．しかしネットワークの受信処理において優先度が反映されていないという問題が存在する．本研究では上記問題に対する解決法として，パケットの選択破棄と割込みの抑制という 2 つの手法を導入する．本論文では，これらの手法を導入したシステムの設計および Linux における実現法について述べる．また SMP 型計算機上で試作システムを用いて行った評価実験の結果について報告する．実験によりネットワーク負荷が高いとき，実現したシステムが優先度に従い CPU 資源を正しく割り当てていることを確認した．

Priority-based Receiver Processing in the Linux Kernel

Ryota Ozaki[†] and Yasuichi Nakayama[††]

The explosive growth of the Internet and the number of its users places interesting new demand for the server systems. Service managements according to contents of service become new requirements for server systems. Linux operating systems provide service managements to individual processes using priority. On network receive processing, however, priority does not work adequately. To overcome this problem, we propose two techniques, *dropping lower-priority packets* and *suppression of interrupts*. We have designed and implemented proposed system on Linux, and have evaluated it by comparing with original network processing system. Experimental results show that our system have an advantage over original network processing system.

1. はじめに

インターネットの発展とユーザの増加にともない，ネットワーク上で種々のサービスを提供するサーバシステムへの要求が高まっている．サーバシステムには多数のクライアントの要求に応える処理性能やシステムの安定性が必要である．そのような要求に対し，SMP 型計算機などの導入による処理性能の強化や，オペレーティングシステム（以下 OS）の改善による対応がなされている[1),3)~8)]．またプロセスへの資源割当てを適切に行うことで，サービスを安定して供給する手法に関する研究が行われてきた．

Unix 系 OS では，優先度を用いて個々のプロセスに割り当てる資源を調節している．通常は CPU 資源の利用時間に差を持たせ，高い優先度を持つプロセスには他のプロセスより多くの CPU 資源を割り当てる．ところが，Unix 系 OS にはネットワーク受信処理において優先度が反映されず，適切な資源割当てが行われないという問題が存在する．OS 内で消費される CPU 時間に比例し不適切さが大きくなるため，システムへのネットワーク負荷が高いときに影響が大きい．また，OS 内のネットワーク処理はそれに関係するプロセスには依存しない．そのため，SMP 型計算機上においては OS 内での CPU 消費の割合が高くなり，資源割当てに対する影響が大きくなる．

本研究の目的は OS のネットワーク受信処理において適切な資源割当てを実現することである．高負荷時に破棄されるパケットが優先度を反映していないことに着目し，パケットの選択破棄という手法を導入する．優先度の高いパケットを優先的に残すことで，対応する優先度の高いプロセスに対し適切に CPU 資源が消費される．また優先度が高く多くの CPU 資源を割り当てられるプロセスほど，割込みを多く受けるという問題に対し，SMP 型計算機の割込みコントローラを

† 総合研究大学院大学数物科学研究科情報学専攻
Department of Informatics, The Graduate University of Advanced Studies
†† 電気通信大学電気通信学部情報工学科
Department of Computer Science, The University of Electro-Communications

利用して，優先度の高いプロセスを実行しているプロセッサに対する割込みを抑制することにより，プロセスのパケット受信処理が割込みによって阻害されることを防ぐ．

本研究では，これらの手法を導入したシステムを設計し，Unix 系 OS の 1 つである Linux 上に実現した[9),10)]．実現した試作システムを用い SMP 型並列計算機上で性能評価実験を行った．ネットワーク負荷が高いときオリジナルの Linux と比較して，優先度に従い CPU 資源を正しく割り当てていることが確認された．

以下本論文では，2 章で Unix 系 OS におけるネットワーク受信処理について述べたのち，その問題点について言及する．3 章では 2 章の問題点を克服する手法を提案し，システムの設計を行う．4 章では実現したシステムの評価実験について報告する．最後に 5 章で関連研究について述べ，6 章でまとめを行う．

2. ネットワークサブシステム

2.1 パケット受信処理

ここでは Unix 系 OS におけるネットワークサブシステムの受信時におけるパケットの処理内容について述べる．また Linux におけるネットワークサブシステムの特徴についても，同時に言及する．

外部から送られてきたパケットは割込みハンドラ，ソフトウェア割込み☆で必要な処理を施されプロセスへ渡される（**図 1**）．ネットワークインタフェース（以下 NI）にパケットが到着すると割込みハンドラが起動し，パケットに対する処理を行う．パケットは一時的にバックログへ格納される．しかる後，OS の遅延処理機構であるソフトウェア割込みによりプロトコル処理を施されソケットバッファへ格納される．

2.1.1 バックログ

バックログとは，パケットを一時的に保存しておくために用意されたバッファである．バックログには格納できるパケット数に限界値が設定され，その数を超えるパケットは破棄されるのが一般的である．

割込みハンドラは送られてきたパケットをいったんバックログに格納し処理を終える．バックログに格納されたパケットは，後でソフトウェア割込みによって取り出され処理される．

Linux ではプロセッサごとにバックログを持ち，キューアクセスにおけるプロセッサ間の競合をなくしている．バックログにおけるパケット最大格納数は決められており，無尽蔵にパケットを蓄積することはできない．バックログのサイズはカーネルインタフェース，たとえば proc ファイルシステムなどを利用して変更することは可能であるが，動的に変化するものではない．

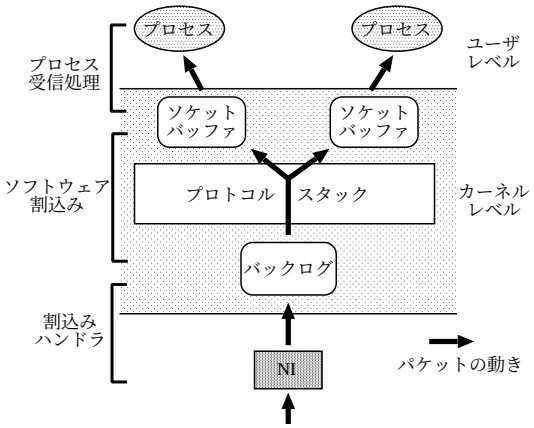

図 1 パケット受信処理
Fig. 1 The process of receiving packets.

Linux ではバックログ内のパケット数を基に，高負荷状態の判定を行っている．ネットワークの負荷を平静状態と高負荷状態の 2 つに分け処理内容を変更する．到着パケット数が増え，バックログの最大格納数を超えた場合，高負荷状態であると判断する．そして平静状態に戻るまでそれ以降到着するパケットはすべて破棄する．バックログのパケットをすべて処理し終えると，高負荷状態から平静状態へ移行し再び到着するパケットを受け入れる．

2.1.2 ソケットバッファ

ソフトウェア割込みではパケットに対しプロトコル処理をはじめ様々な処理を行う．パケットの宛先が自ホストであることが分かるとそのパケットは対応するソケットに用意されたバッファに格納され，プロセスの受信処理を待つ．バックログと同様，バッファのサイズの最大値を超える場合新たなパケットは破棄される．

ソケットバッファもバックログと同様最大格納数が決まっており，それを超えて到着したパケットは破棄される．最大格納数は proc ファイルシステムなどのカーネルインタフェースで変更可能である．

2.2 問 題 点

Linux では重要度の違うプロセスに優先度を付け資源割当てに差を持たせている．しかしネットワークサブシステムにおいては，すべてのパケットは同等に扱われるため資源割当てが適切に機能していない．

☆ システムコール呼び出しなどを利用する場合に，ソフトウェア的に発生させる割込み（同じくソフトウェア割込みと呼ぶ）とは異なる概念であるので注意されたい．

2.2.1 パケット破棄ポリシ

ネットワークサブシステムでは外部から送られてくるパケットをすべて同等に扱うため，優先度の高いプロセスへのパケットが優先度の低いプロセスへのパケットに CPU 時間を不当に取られるという問題がある．負荷が高くなくプロセッサの処理能力に余裕がある場合，処理量の不適切さは問題ではない．しかしネットワークの負荷が上がりネットワーク処理の占める割合が増えた場合，この不適切さがプロセスと対応するパケットに対する CPU 時間のバランスに大きな影響を及ぼす．

割込みハンドラでは処理時間の短縮のため到着したパケットをバックログに入れる処理のみ行う．もしバックログが満杯であった場合，パケットは即破棄される．たとえ到着したパケットの優先度がバックログにあるパケットのものより高い場合でも，到着したパケットが破棄される．

2.2.2 割込みによるプロセスの処理の阻害

優先度の高いプロセスは優先度の低いプロセスに比べ CPU が割り当てられる頻度が高く割当て時間が多くなる．しかし CPU 割当て時間の長いプロセスは，それだけ実行中に割り込まれる回数が増える．そのためパケットがソケットバッファにたどり着いたとしても，プロセスが受信処理を行う前にソケットバッファが満杯になりパケット破棄が起き，その結果プロセスのパケット受信量は低下する．

3. 提案するネットワークサブシステム

3.1 設計方針

前章で提起した問題点を受け，プロセスの優先度を適切に反映するネットワークサブシステムを提案する．本研究ではパケットの選択破棄と割込みの抑制という手法を導入する．バックログとソケットバッファにおいて，破棄されるパケットを調整することで優先度に従ったパケット制御を実現する．

3.2 パケットの選択破棄

通常パケット受信処理ではプロトコル処理が終わらなければそのパケットが送られるべきプロセスが分からない．すでに CPU 時間を大量に消費してしまっているため，この段階でパケットを破棄することは無意味である．

そこで NI からパケットを受け取った段階で送信先プロセスを判別しそのパケットの優先度を割り出す．CPU 時間を消費していない段階で，割り出した優先度に基づき破棄するパケットを決定する．

本システムではこの機構を実現するため，バックログに新たなキュー構造を導入する（**図 2**）．以下では導入するキュー構造を PBQs（Priority Based Queues）と呼ぶ．PBQs は優先度に従いソートされた複数のキューを持ち，パケットは対応するキューに振り分けられる．それぞれのキューにあるパケットを調べることにより，破棄すべきパケットを決定する．本システムでは負荷が高くなったときに PBQs を利用する．負荷の低いときは従来のバックログキューを利用し，バックログにおけるパケット振分けのコストがかからないようにする．

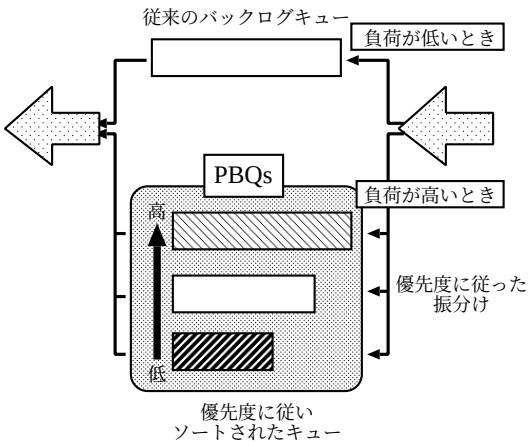

図 2 提案するバックログの構造
Fig. 2 Structure of proposed backlog.

以下では，バックログに対するパケットの格納と取り出し処理について述べる．

3.2.1 バックログに対する処理

図 3 でバックログに対するパケットの格納処理，**図 4** でバックログに対するパケットの取り出し処理について，疑似コードで示す．以下は疑似コードのための諸定義である．

- $\boldsymbol{P}_{<id>}$：パケットを表す．$_{<id>}$ は識別名である．
- $\boldsymbol{Q}_{<id>}$：パケットを格納するキューを表す．$_{<id>}$ は識別名である．特に従来のバックログキューを Q_{orig} と表記する．
- $\boldsymbol{P} \leftarrow_{head} \boldsymbol{Q}$：$Q$ の先頭から P を取り出すことを表す．
- $\boldsymbol{P} \leftarrow_{tail} \boldsymbol{Q}$：$Q$ の末尾から P を取り出すことを表す．
- $\boldsymbol{Q} \leftarrow_{tail} \boldsymbol{P}$：$Q$ の末尾にパケット P を格納することを表す．

本システムではパケットから高速に優先度を割り出すために，ポート番号をキーとし，そこから対応する優先度を得るハッシュを用意する．サーバプロセスが bind システムコールを呼び出したとき，ポート番号と

```
 1: function Backlog_enqueue(P_in)
 2:   if（負荷が低い）then
 3:     Q_orig ←_tail P_in
 4:   else
 5:     PBQ_enqueue(P_in)
 6:   endif
 7: end

 8: function PBQ_enqueue(P_in)
 9:   // P_in の優先度 prio を割り出す
10:   prio := determine_priority(P_in)
11:   // prio を基にキュー Q_in を決定する
12:   Q_in := determine_queue(prio)
13:   if（PBQs が満杯）then
14:     // prio より優先度の低いキューの中から
15:     // ランダムにキューを決定し Q_sac とする
16:     Q_sac := determine_sacrifice(prio)
17:     if（Q_sac に一定以上のパケットがない）then
18:       P_in を破棄する
19:       return
20:     endif
21:     P_sac ←_tail Q_sac
22:     P_sac を破棄する
23:   endif
24:   Q_in ←_tail P_in
25: end
```

図 3 バックログへのパケットの格納（割込みハンドラ）
Fig. 3 Store a packet into backlog (Interrupt handler).

```
 1: function Backlog_dequeue()
 2:   if（Q_orig にパケットがある）then
 3:     P_out ←_head Q_orig
 4:     return P_out
 5:   else
 6:     return PBQ_dequeue()
 7:   endif
 8: end

 9: function PBQ_dequeue()
10:   if（PBQs にパケットがある）then
11:     // パケットを持つキューのうち
12:     // 最も優先度の高いキュー Q_out とする
13:     Q_out := determine_max_priority_queue()
14:     P_out ←_head Q_out
15:     return P_out
16:   else
17:     return NULL
18:   endif
19: end
```

図 4 バックログからのパケット取出し（ソフトウェア割込み）
Fig. 4 Pick up a packet from backlog (Software interrupt).

ソケットおよびサーバプロセスの優先度を取得する．*determine_priority*（）ではパケットに対しプロトコルスタックで行われる処理☆を施しポート番号を取得．それを基にハッシュより優先度を得る．

パケットの格納処理では，優先度を割り出し対応するキューにパケットを格納する．PBQs が満杯であった場合，パケットの優先度より低い優先度のキューから 1 つ選んでその中のパケットを破棄する．ただし低い優先度のキューにあるパケットがある一定の数（最小パケット数）以下の場合は，新しいパケットの方を破棄する．これにより，ある優先度の低いパケットを過剰に破棄して性能が低下することを防ぐ．このキューの最小パケット数は優先度に従って決められ，優先度が低いキューほどサイズが小さい．

3.3 割込みの抑制

ネットワークの負荷が高くなり，割込みの発生が多くなると実行中のプロセスへの割込みの回数が増加する．割り込まれる回数はプロセスの実行頻度に比例するため，優先度の高いプロセスほど割り込まれる回数も多くなる．

本研究では割込みの発生回数が増加したとき，プロセスの優先度に従い割込みを抑制する処理を導入する．パケットを待つプロセスに処理が移行したときプロセスの優先度に従い割込みを禁止にする．

これにより優先度の高いプロセスに多く CPU 時間が割り当てられ，より多くのパケットを受信することが可能となる．

3.3.1 割込み禁止判定

コンテクスト切替え時に割込み禁止判定を行う．本研究では以下の条件がすべて満たされたとき割込みを禁止する．

- 高負荷状態である．
- ソケットバッファのパケット数が閾値を超えている．
- プロセスの優先度に従った判定に成功した．

本システムでは負荷が高い状態のときのみ，割込みを禁止する（負荷の判定方法については 2.1.1 項参照）．これによりプロセスのパケット受信を促進させる．またソケットバッファに格納されているパケット数がある閾値を超えているとき割込みを禁止する．ソケットバッファにパケットが少ない場合は，プロセスが受信処理を行っても取得できるパケットは少ないと考えられる．閾値は経験的に決まる値で，本システムでは閾値をソケットバッファの最大格納数の半分にしている．そしてプロセスの優先度に従った確率による判定を行う．高い優先度のプロセスほど割込みを禁止する回数が多くなるようにすることで，優先度を反映した処理が可能となる．

☆ パケットの種類の判定などパケットヘッダの検査．

3.4 実　　装

本研究では Linux カーネルバージョン 2.4.18-18.8.0☆1を利用し試作システムを実現した．コードの改変はネットワークサブシステムやスケジューラなどに対して行っており，追加コードを含めると 3,000 行弱の変更となった．

3.4.1 優先度管理

bind システムコール（カーネル内では inet_bind 関数☆2）が呼ばれるのを契機に，ソケットとプロセス優先度とポートを関連付け保存する．ハッシュによりそれぞれの要素を高速に取得できるようにする．同様に close システムコール（カーネル内では inet_release 関数☆2）を監視し不要になった要素を削除する．

本システムでは nice システムコール（カーネル内では set_user_nice 関数☆4）を監視して，優先度の変化に対応している．プロセスの優先度が変化した場合，新たな優先度を元に PBQs をソートし直す．

本システムはソケットを 1 つの制御単位として実装をしており，基本的に複数プロセスで協調して動作するプログラムには対応していない．複数プロセスでソケットを共有する場合，プロセスの中で一番高い優先度が反映される．

3.4.2 バックログ操作

Linux では到着したパケットは割込みハンドラ内で呼び出される netif_rx 関数☆3内でバックログへ格納される．本システムではこれを図 3 における *Backlog_enqueue* に置き換えている．バックログからのパケット取出し操作はソフトウェア割込み内で，process_backlog 関数☆3によって実行される．本システムではこれを図 4 における *Backlog_dequeue* に置き換えている．

パケット取出し処理では優先度の高いキューから先にパケット取り出す．この処理は Linux の $O(1)$ スケジューラと同様の手法を用いており，計算量は $O(1)$ である．PBQs のそれぞれのキューでパケット数を監視して，ビット列を管理する．Linux が持つ find_first_bit 関数☆4を用いてパケットの存在するキューを見つけ出す．

3.4.3 割　込　み

schedule 関数☆5の中で，割込み抑制を行う．Linux カーネルの IRQ-affinity を用いて，特定の NI の割込みを禁止する．Linux では balance_irq 関数☆6を使い IO-APIC で割込みをかけるプロセッサを分散している．IO-APIC に対して ACK を返すときに，次に割込みをかけるプロセッサをプロセッサ番号の隣になるプロセッサのどちらかからランダムで選ぶ．

本システムでは，Linux の IRQ-affinity 機能を利用し NI の割込み禁止を実現している．IRQ-affinity は IO-APIC を利用することで特定 IRQ から特定プロセッサへの割込みを禁止することを可能としている．ただしこの場合，特定 IRQ からの割込みをすべてのプロセッサが禁止することは不可能である．つまり最低 1 つのプロセッサは割込みを許可している必要がある．

3.5 本システムの適応範囲や前提条件

以下では本システムの適応範囲や前提条件について，各要因ごとに述べる．

バックログの有無

本システムはバックログを持つネットワークサブシステムに対して適応可能である．

SMP 型計算機

IRQ-affinity はシングルプロセッサのシステムでは利用できない．そのため本システムは SMP 型計算機においてのみ有効である．Local-APIC を利用した割込み禁止手法を利用すれば，シングルプロセッサ計算機であっても同様の効果が得られる．しかしその場合，文献 7) のようなポーリングによるパケット受信機構を導入しなければ性能劣化が起きる．

サーバプロセス

本システムでは，bind システムコールでポートを開いたサーバプロセスのみ管理する．少しの改良でクライアントプロセスに対しても適応させることは可能であるが，本研究では対象外としている．

4. 性能評価実験

前章の設計方針を基にシステムを実現し性能評価実験を行った．

4.1 実 験 環 境

実験環境として，ネットワークで接続された 1 台のサーバマシンと 1 台の計測用クライアントマシンおよび 4 台の負荷生成用クライアントマシンからなるネットワーク環境を構築した．実験で使用するハードウェアおよび OS について**表 1** にまとめた．

サーバマシンとしては 4 台のプロセッサと 2 枚の

☆1 RedHat Linux バージョン 8 において提供されている Linux カーネルである．
☆2 Linux ソース `linux/net/ipv4/af_inet.c` 参照．
☆3 Linux ソース `linux/net/core/dev.c` 参照．
☆4 Linux ソース `linux/asm/bitops.h` 参照．
☆5 Linux ソース `linux/kernel/sched.c` 参照．
☆6 Linux ソース `linux/arch/i386/kernel/io_apic.c` 参照．

表 1 ハードウェアの仕様
Table 1 Hardware configuration of experimental environments.

ネットワーク	
スイッチングハブ	CentreCOM FS909GT V1 1000 BASE-T ポート ×1 100 BASE-TX ポート ×8

クライアント	
プロセッサ	Intel Celeron 800 MHz
メモリ	256 MBytes
OS	RedHat Linux 7.1（カーネル 2.4.7）
NI カード	DEC 21140 chip 100 Mbps

サーバ	
プロセッサ	Intel PentiumIII Xeon 500 MHz （2 次キャッシュ512 KB）×4
メモリ	256 MBytes
OS	RedHat Linux 8.0（カーネル 2.4.18-18.8.0）
NI カード	Intel PRO/1000XT PWLA8490XT 1 Gbps ×2

Gigabit Ethernet カードを持つ SMP 型計算機を用いた．クライアントマシンとしては Fast Ethernet カードを持つ PC を利用した．サーバマシンとクライアントマシンはスイッチングハブを介して接続されており，サーバマシンの一方の NI に計測用マシン，もう一方の NI に 4 台の負荷生成用クライアントマシンが接続される．サーバ・スイッチングハブ間は Gigabit Ethernet で接続され，クライアント・スイッチングハブ間は Fast Ethernet で接続されている．サーバの NI カードである Intel PRO/1000XT のドライバソフトウェアには Intel 社が提供している e1000-4.4.19☆を用いた．

4.2 マイクロベンチマークによる評価

この実験では TCP 通信における通信遅延とスループット，UDP 通信における通信遅延について計測した．ベンチマークプログラムには LMbench（バージョン 2.0.3）☆☆を利用した．サーバマシンとクライアントマシンはスイッチングハブを挟んで，1 対 1 の構成で実験を行った．それぞれの実験におけるデータサイズなどのパラメータは LMbench の標準値を用いた．TCP スループットでは 64 KBytes のデータ，TCP と UDP の通信遅延はそれぞれ 1 Byte，4 Bytes のデータを転送する．TCP，UDP ともに往復通信遅延を計測している．

本システムと手を加えていない Linux カーネルについての計測結果を**表 2** に示す．TCP 通信のスループットには差がほとんど見られず，通信遅延の差も小さい．PBQs と従来のバックログキュー操作に加えられた分岐処理のオーバヘッドの影響が両者の差として現れている．しかし本システムの従来のバックログキューを併用しているため，そのオーバヘッドは最小限に抑えられている．

☆ http://www.intel.com/support/network/adapter/1000/e1000.htm で入手可能.

☆☆ http://sourceforge.net/projects/lmbench で入手可能.

表 2 マイクロベンチマーク結果
Table 2 Results of micro-benchmarks.

カーネル	TCP スループット（Mbps）	TCP 通信遅延（μsecs）	UDP 通信遅延（μsecs）
オリジナル	94.96	102.50	81.90
本システム	94.91	103.76	85.41

4.3 パケット受信数による評価

サーバに負荷がかけられている状態でのサーバのパケット受信数を評価する実験を行った．計測プログラムとしては，クライアントから送られてくる UDP パケットを数え上げるプログラム（以下 bw）を用いた．bw クライアントは一定のスループット（この実験では 34,000 pkts/sec）で UDP パケットを bw サーバに送信する．bw サーバは単に到着したパケットを数え上げる．

ネットワーク負荷生成のために UDP パケットをサーバに対して送信するプログラム（以下 flood）を利用した．flood サーバは到着した UDP パケットを即座に破棄する．flood クライアントは負荷生成クライアントマシン上で動作している．これにより bw クライアントのパケットがネットワーク上で妨害されることを防ぐ．flood クライアントを 4 つの計算機に 1 つずつ動かし，サーバ上の 4 つの flood サーバプログラムに対してデータを送信する．この実験では bw サーバの優先度を最大（nice 値では −19），flood サーバの優先度を最小（nice 値では 20）に設定している．この設定により flood サーバはほとんど実行されない．しかしネットワークサブシステムでのパケット受信処理は通常どおり行われる．

実験結果を**図 5** に示す．横軸は flood クライアントが送信するパケット量を表しており，縦軸は bw サーバのパケット受信数を表している．ネットワーク負荷が高くなると，サーバが処理する flood プログラムのパケットが増え，逆に bw プログラムのパケットの処理量が減少する．バックログでパケットが破棄されるようになると，本システムの選択破棄の機能が有効に働き，優先度の高い bw プログラムのパケットの代わりに flood プログラムのパケットが破棄されるようになる．図 5 において flood クライアントのパケット送信率が 190,000 pkts/sec 以上のとき，従来のシステム

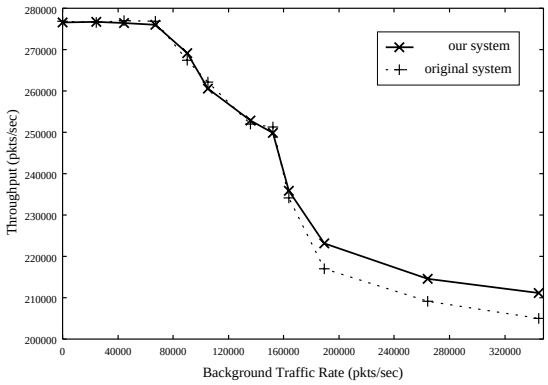

図 5 パケット受信数
Fig. 5 Throughput with concurrent load.

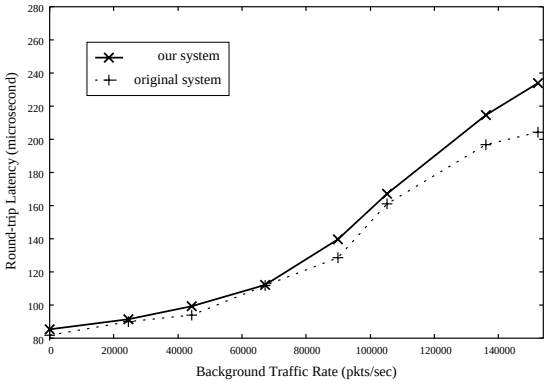

図 6 通信遅延
Fig. 6 Latency with concurrent load.

より本システムのパケット受信数が上回る．つまり優先度の高い bw サーバに対して CPU 資源が，多く割り当てられていることになる．flood クライアントのパケット送信率が 400,000 pkts/sec 超えたとき本システム・従来のシステムともに livelock [7] が発生し，計測が不可能となった．

4.4 通信遅延による評価

サーバに負荷がかかっている状態での通信遅延を計測する実験を行った．クライアントとしては LMbench の UDP における通信遅延を計測するプログラム（以下 lat_udp）を利用した．4.3 節の実験と同様 flood プログラムを用いサーバに負荷をかける．ネットワーク環境および優先度の設定も 4.3 節の実験と同様である．

実験結果を**図 6** に示す．横軸は flood クライアントが送信するパケット量を表しており，縦軸は lat_udp によって計測した通信遅延を表している．ネットワーク負荷が高くなるとバックログに多くのパケットが溜まるため，lat_udp クライアントのパケットが処理されるまでの時間が長くなる．そのため負荷の上昇に従い通信遅延が大きくなる．負荷が低いときには本システムと従来のシステムともに同じバックログが使われるため性能に差はない．負荷が高くなり PBQs にパケットが格納されるようになると，優先度の高い lat_udp クライアントのパケットが優先的に処理されるようになる．そのため 1 つのキューしか持たない従来のバックログに比べ，lat_udp クライアントのパケットが素早く処理されることが期待される．しかし実験結果を見るとネットワーク負荷が高くなり PBQs が利用されるようになると，従来のシステムと本システムの通信遅延の差が大きくなっている．これは負荷が高くなり PBQs に対する処理量が増えたため，PBQs に対する処理のオーバヘッドが顕著に現れたためと考えられる．パケット送信率が 150,000 pkts/sec を超えると lat_udp サーバに到着する前に破棄されるパケットが多くなり正常な計測ができなかった．

5. 関 連 研 究

LRP [4] ではパケットを待つプロセスが実行中である場合のみ対応するパケットの受信を許すことで，プロセスの優先度を間接的にパケット受信処理に反映させている．この手法によりあるプロセスに対するネットワーク負荷が高くなっても他のプロセスのネットワーク処理に影響を与えない．しかし本研究と異なり SMP 型計算機に対する考慮はなされていない．

また Scout OS [8] では Path と呼ばれる通信路を概念化した構造を導入している．Path を用い静的にルーティングを決定しておくことで通信の最適化を図っている．しかし本研究の手法のように他の優先度のパケットを落とすなど，互いの通信路が影響を及ぼすような処理は行っていない．

Banga らは Resource Container [1] と呼ばれる実行主体と資源とを分離するための概念を導入している．これにより資源割当てを正しく行うことを目的としている．しかし本研究のように割込みに対する考慮はなされていない．

Sequent 社の Balance という SMP 型計算機は，SLIC [2] と呼ばれる割込みを分散させることができるハードウェアを持つ．SLIC は OS がプロセッサごとに設定する優先度を利用して，割込みをかけるプロセッサを選ぶ．SLIC を用いることで，優先度の高いプロセスの動くプロセッサへ割込みを行わないことにより，優先度の高いプロセスに適切に CPU が割り当てられる．しかしネットワークサブシステム内におけるパケットの処理を制御できないため，ネットワーク

が高負荷状態になった場合などでは，優先度を適切に反映させることはできない．

6. おわりに

本研究では優先度を適切に反映するネットワークサブシステムを設計し Linux 上に実現した．ネットワークサブシステムについて述べ，その問題点を言及した．パケットの選択破棄と割込みの抑制という手法を導入し，問題点を克服するネットワークサブシステムの試作システムを Linux に実現し，その評価を行った．

Unix 系 OS ではネットワーク負荷が高い状況において，ネットワークサブシステムの受信時にプロセス優先度が正しく反映されないという問題が存在する．パケットの受信処理においては，その優先度は反映されておらずすべてのパケットが同等に処理される．そのためバックログが満杯のときに発生するパケット破棄では，優先度の高いパケットが不当に破棄されるという問題が起こる．本研究ではバックログにおいてパケットの選択破棄を行うことでその問題に対応した．またパケット到着などのイベント時に起こる割込みが優先度の高いプロセスの処理を阻害する問題がある．本研究では割込み発生を制御することでこの問題に対応した．

また本研究では上記手法を Unix 系 OS である Linux に対し適用し，システムを設計した．試作システムを実現し，それを基に性能評価実験を行った．評価実験ではマイクロベンチマークおよびネットワーク負荷が与えられたときのスループット・通信遅延について評価を行った．マイクロベンチマークを用いた実験によりシステムの基礎性能は従来のシステムとほぼ同等であることを示した．またネットワーク負荷が高いときに，優先度を正しく反映させていることを実験により確認した．

今後の課題としては，システムのオーバヘッド削減，TCP/IP を用いる応用プログラムでの実験，LRP などの他のシステムとの比較などによる詳しいシステムの評価などがあげられる．またシステムにポーリングによるパケット受信処理を導入することが考えられる．これにより livelock を回避することができる[5]．またポーリングと割込みの併用により，割込み禁止時でも性能の劣化を防ぐことができるため，本システムのシングルプロセッサへの適応が可能となる．

参考文献

1) Banga, G., Druschel, P. and Mogul, J.C.: Resource Containers: A New Facility for Resource Management in Server Systems, *Proc. ACM Symposium on Operating Systems Design and Implementation*, pp.45–58 (1999).
2) Beck, B., Kasten, B. and Thakkar, S.: VLSI Assist For A Multiprocessor, *Proc. 2nd International Conference on Architectural Support for Programming Languages and Operating Systems* (*ASPLOS II*), pp.10–20 (1987).
3) Brustoloni, J., Gabber, E., Silberschatz, A. and Singh, A.: Signaled Receiver Processing, *Proc. USENIX 2000 Annual Technical Conference*, pp.211–223 (2000).
4) Druschel, P. and Banga, G.: Lazy Receiver Processing (LRP): A Network Subsystem Architecture for Server Systems, *Proc. 2nd USENIX Symposium on Operating Systems Design and Implementation* (*OSDI '96*), pp.261–276 (1996).
5) Hansen, J.S. and Jul, E.: Latency Reduction Using a Polling Scheduler, *Proc. 2nd Workshop on Cluster-Based Computing*, pp.27–31 (2000).
6) Kitayama, T., Nakajima, T., Okikawa, D. and Tokuda, H.: Network Subsystem Architecture Alternatives for Distributed Real-time System, *Trans.Information Processing Society of Japan*, Vol.40, No.1, pp.23–33 (1999).
7) Mogul, J.C. and Ramakrishnan, K.K.: Eliminating Receive Livelock in an Interrupt-Driven Kernel, *ACM Trans. Comp. Syst.*, pp.217–252 (1997).
8) Mosberger, D. and Peterson, L.L.: Making Paths Explicit in the Scout Operating System, *USENIX Symposium on Operating Systems Design and Implementation*, pp.28–31 (1996).
9) 尾崎亮太：並列計算機のためのネットワーク処理機構，電気通信大学大学院電気通信学研究科情報工学専攻修士論文 (2003).
10) 尾崎亮太，中山泰一：プロセスの優先度に基づいてサービス品質を制御するネットワーク処理機構，情報処理学会ハイパフォーマンスコンピューティング研究会，2003-HPC-93, pp.71–76 (2003).

(平成 15 年 5 月 26 日受付)
(平成 15 年 12 月 2 日採録)

尾崎 亮太（学生会員）

1979 年生．2003 年電気通信大学大学院電気通信学研究科情報工学専攻博士前期課程修了．現在，総合研究大学院大学数物科学研究科情報学専攻博士後期課程在学中．並列・分散処理，オペレーティング・システムに興味を持つ．

中山 泰一（正会員）

1965 年生．1988 年東京大学工学部計数工学科卒業．1993 年同大学大学院工学系研究科情報工学専攻博士課程修了．工学博士．同年電気通信大学情報工学科助手．現在，同学科助教授．オペレーティング・システム，並列・分散処理，ゲーム・プログラミングに興味を持つ．日本ソフトウェア科学会，電子情報通信学会，IEEE CS，CSA，ICCA 等の会員．