資源濫用攻撃に耐性のある資源管理方式

河野健二[†] 金子 済[††] 清水謙多郎[††]

不正なプログラムによる攻撃の一形態に，計算資源を意図的に占有することによって，他のプログラムの応答性を著しく低下させる攻撃が知られている．このような形態の攻撃を資源濫用攻撃と呼ぶ．本論文では，資源濫用攻撃に耐性のある資源管理方式として，優先度付き資源管理方式を提案する．優先度付き資源管理方式では，各プロセスに資源優先度と呼ぶ優先度を与え，その優先度に応じた資源割当てを行う．優先度の高いプロセスは，優先度の低いプロセスの資源を横取りすることができ，資源を濫用するプロセスから資源を奪い，それらの資源を他のプロセスに割り当てることが可能となっている．そのため，資源濫用攻撃を未然に防いだり資源濫用攻撃から回復することができる．実際，優先度付き資源管理方式を Linux 2.4.7 に実装し，資源濫用攻撃を防ぐことができることを確認した．

A Resource Management Scheme Resilient to Resource-monopolizing DoS

Kenji Kono,[†] Wataru Kaneko[††] and Kentaro Shimizu[††]

Resource-monopolizing Denial-of-Service (DoS) is one form of malicious code attack. An attacking code exclusively uses shared resources and attempts to drop the responsiveness of the machine so low that it is practically useless. Resource-monopolizing DoS is difficult to prevent in commodity operating systems, because they allow every process to compete for shared resources in an uncontrolled manner. This paper presents preemptive resource management, a scheme we developed to defend against resource-monopolizing DoS. Preemptive resource management extensively applies *priority* and *preemption* to every type of resource. It controls resource allocation based on priority, and preempts resources allocated to lower-priority processes based on the availability of shared resources. The experimental results we obtained suggest that preemptive resource management prevents resource-monopolizing DoS from impinging on the responsiveness of other innocent processes.

1. はじめに

インターネットが広く普及し，プログラムの配布手段としてインターネットを用いることが一般的になりつつある．しかし，インターネットの持つ高い開放性ゆえ，あらかじめ，悪意のあるプログラムが配布される可能性を完全に排除することは難しい．そのため，インターネットからダウンロードしたプログラムが不正攻撃を行い，ユーザの計算機システムに損害を与える危険性がある．

不正なプログラムによる攻撃の一形態に，計算資源を意図的に占有することによって，他のプログラムの応答性を著しく低下させる攻撃が知られている．このような形態の攻撃を**資源濫用攻撃**と呼ぶ．資源濫用攻撃は，サービス拒否攻撃（Denial-of-Service attack）の 1 つであり，攻撃を受けた計算機は，資源不足のためにユーザにサービスを提供し続けることが困難になる．最悪の場合，計算機がフリーズしたのと同じ状況に陥ってしまう．資源濫用攻撃は現実の脅威であり，2001 年には，i-mode サービスにおける資源濫用攻撃として，CPU 時間を濫用するメールの例が報告されている[1)]．これと類似の攻撃は，一般のクライアント計算機に対しても可能であり，資源濫用攻撃の危険性を示唆している．

本論文では，資源濫用攻撃に耐性のある資源管理方式として，**優先度付き資源管理方式**と呼ぶ仕組みを提案する．優先度付き資源管理方式では，各プロセスに**資源優先度**と呼ぶ優先度を与え，その優先度に応じた資源割当てを行う．すなわち，資源優先度の高いプロセスには優先的に資源を割り当て，資源優先度の低い

† 電気通信大学情報工学科
Department of Computer Science, University of Electro-Communications
†† 東京大学大学院理学系研究科情報科学専攻
Department of Information Science, Faculty of Science, The University of Tokyo

プロセスにはゆとりのある場合にのみ資源を割り当てる．優先度の高いプロセスは，優先度の低いプロセスの資源を横取り（preempt）することができ，優先度の高いプロセスの実行前に占有されてしまった資源でも，優先度の低いプロセスがそのまま占有し続けることはできない．この仕組みを用いると，資源濫用攻撃を未然に防いだり，資源濫用攻撃から回復したりすることができる．優先度付き資源管理方式は，次の点において，資源濫用攻撃に耐性のある資源管理方式といえる．

- **資源濫用攻撃の未然防止が可能**
  優先度付き資源管理方式を用いることによって，資源濫用攻撃を未然に防ぐことができる．インターネットなどから入手した信頼性の低いプログラムを実行する場合，そのプログラムの資源優先度をあらかじめ低く設定しておく．これによって，そのプログラムが資源濫用攻撃を行ったとしても，資源優先度が相対的に高い他のプロセスが攻撃の影響を受けることがない．
- **資源濫用攻撃からの回復が可能**
  優先度付き資源管理方式を用いると，資源濫用攻撃を受けてシステムの応答性が低下した場合でも，その状態から回復することができる．攻撃を仕掛けているプロセスの資源優先度を下げれば，他のプロセスの資源優先度が相対的に高くなり，それらのプロセスは占有された資源を横取りすることができる．その結果，資源不足が解消され，資源濫用攻撃から回復することができる．
- **簡明な資源割当て制御**
  優先度付き資源管理方式では，資源優先度の変更のみで資源割当てを制御できる．資源割当てを制御するために，プロセスが使用できるCPU時間，メモリ，I/Oのバンド幅などを物理量で指定する必要はない．本論文で述べる手法は，一般のエンド・ユーザが利用することを想定しており，資源割当ての制御が簡潔に行えることが望ましい．実時間処理やメインフレーム用のオペレーティングシステムでは，柔軟な資源割当て制御が行えるようになっているものも多いが，これらのシステムでは複雑で綿密な設定を正しく行う必要がある．

優先度付き資源管理方式の有効性を示すため，CPU時間，物理メモリ，ディスクI/Oのバンド幅を対象に，優先度付き資源管理方式の実現を行った．Linux 2.4.7カーネルに提案方式を組み込み，資源濫用攻撃に対する耐性を有することを確認した．資源濫用攻撃を行うと，提案方式を組み込まないLinuxでは，その応答性が100分の1程度に低下するのに対し，優先度付き資源管理方式を組み込んだLinuxでは，応答性の低下はほとんどみられなかった．

以下，2章では資源濫用攻撃による脅威を示し，既存の資源管理方式の問題点を明らかにする．3章では優先度付き資源管理方式について述べる．4章ではLinuxでの実装を示し，6章で実験結果について報告する．7章で関連研究を述べ，8章で本論文をまとめる．

2. 資源濫用攻撃

2.1 資源濫用攻撃の影響

資源濫用攻撃とは，CPU時間やメモリなどの共有資源を意図的に占有し，他のプロセスの応答性を著しく低下させる攻撃である．共有資源を占有するプロセスのことを**占有プロセス**と呼ぶ．バッファ溢れなどの他の不正攻撃に比べ，資源を濫用する攻撃用プログラムを作成することは比較的容易である．たとえば，単にメモリを消費するプログラムを作成すればよい．また，PostscriptやPDFなどのドキュメント・ファイルに細工をし，`ghostscript`などの閲覧プログラムに資源を濫用させることも可能である．そのような細工を施したドキュメント・ファイルを閲覧した場合，信頼性の高い閲覧プログラムであっても占有プロセスとなりうる．

資源濫用攻撃を受けた対話的なプロセスの応答性の低下を示すため，Linux 2.4.7の稼働しているPC/AT互換機（PentiumIII 733 MHz，メモリ128 MB）上で資源濫用攻撃を行った．25 MBのメモリ領域に繰り返しアクセスする占有プロセスを用意し，この占有プロセスを同時に5個実行した．これらの占有プロセスは，物理メモリを占有することによってスラッシング（thrashing）を引き起こし，その結果として，スワップ処理のためにCPU時間とディスクI/Oのバンド幅も同時に占有してしまう．

対話的なプロセスの応答性が低下していることを示すために，対話的なプロセスの振舞いをエミュレートするプロセスを用意した．このプロセスは，4 MBのデータ領域にアクセスし，5秒間スリープするという動作を繰り返す．スリープ時間はユーザからの入力を待つ時間に相当し，4 MBのデータ領域にアクセスする時間はユーザからの入力に応じた処理を行う時間に相当する．したがって，このプロセスの応答時間として，4 MBのデータ領域のアクセスに要する時間を計測した．

この実験では，最初に対話的なプロセスのみを実行

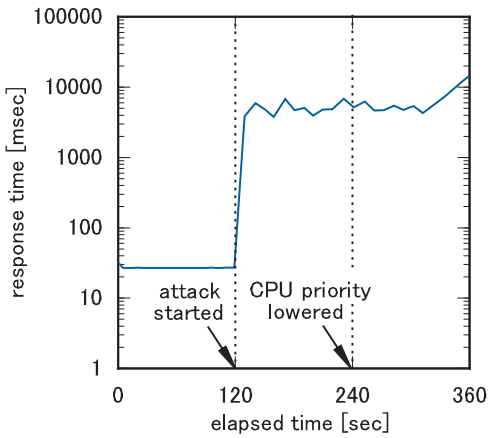


図 1 資源濫用攻撃による応答性の低下
Fig. 1 Impact of resource-monopolizing DoS.

し，120 秒後に 5 つの占有プロセスを実行した．**図 1** に実験結果を示す．横軸は対話的なプロセスを実行してからの経過時間を表し，縦軸は対話的なプロセスの応答時間を示す．なお，縦軸は対数表示となっている．図 1 に示したように，占有プロセスを実行する前の応答時間は 27 msec 前後でほぼ一定であるのに対し，占有プロセスを実行した後の応答時間は 10,000 msec 程度にまで増加しており，フリーズしたのに近い状態となっている．対話的なプロセスを実行してから 240 秒経過した時点で，占有プロセスの CPU の優先度を低下させたが，対話的なプロセスの応答性は改善していない．これは，CPU 時間以外の資源が占有されているためであり，CPU 時間の優先度だけでは資源濫用攻撃を回避することができないことを示している．

2.2 資源濫用攻撃に対する既存の防御策

一見したところ，既存のオペレーティングシステムであっても資源濫用攻撃からの回復は容易であるように思われる．資源濫用攻撃を受けた場合，攻撃を仕掛けている占有プロセスを特定し，その占有プロセスを強制終了すればよいからである．しかし，前節で示したように，計算機の応答性が著しく低下した状況では，UNIX における`ps` や`kill` などのプロセス制御コマンドの実行にも長い時間を要するようになり，結局は計算機をリセットせざるをえない状況となってしまう．計算機をリセットした場合，揮発性のデータがすべて失われてしまうため，エディタなどで編集作業を行っていたドキュメントなどがすべて失われてしまう．また，占有プロセスを強制終了する方式では，資源濫用攻撃を未然に防ぐことはできない．

UNIX などの既存のオペレーティングシステムで広く採用されている資源管理方式では，資源割当てを公平に行うことを主眼としているため，プロセスの信頼度に応じて資源割当てを制御する仕組みを十分には備えていない．CPU 時間の割当てを制御する`nice` 値など，部分的に資源割当てを制御する機構を備えているものの，2.1 節で示したように CPU 時間の割当て制御だけでは十分ではない．

既存のオペレーティングシステムでは，プロセスの使用できる資源の上限（クォータ）を設定できるものも多い．たとえば，UNIX では`setrlimit()` システムコールによって，プロセスが利用できる CPU 時間やメモリ量などを制限する仕組みを備えている．しかし，ディスク I/O やネットワークの帯域に関しては設定を行うことができないため，資源濫用攻撃を防止するには不十分である．ディスク I/O やネットワークの帯域にもクォータが設定できるように拡張したとしても，競合するプロセスが存在しない場合には，帯域を十分に活用できず資源の利用効率が低下する．100 Mbps の帯域を持つネットワークを備えていても，各プロセスが利用できる帯域を 10 Mbps に制限したとすると，90 Mbps の帯域が利用されることがない．

実時間処理システム向けに開発された資源管理方式を利用し，資源濫用攻撃を防ぐ方法が考えられる．実時間処理システムでは，各アプリケーションがその処理に必要となる資源量やデッドラインをあらかじめ申告し，資源の使用状況に応じて入場制限（admission control）を行う[2),3)]．これらのシステムでは，実時間処理向けに設計されたアプリケーションを対象としており，必要とする資源量を見積もることができると仮定している．実時間処理向けでない通常のアプリケーションでは，処理内容に応じて必要な資源量が大きく変動し，処理に必要な資源量をあらかじめ見積もることは容易ではない．したがって，実時間処理向けの資源管理方式をそのまま適用することは難しい．

3. 優先度付き資源管理方式

前章で述べたとおり，既存の資源管理方式では資源濫用攻撃に対処することは難しい．本章では，資源濫用攻撃に耐性を有する資源管理方式として，**優先度付き資源管理方式**を提案する．3.1 節では優先度付き資源管理方式の概要を述べ，3.2 節では優先度付き資源管理方式の運用例を示す．

3.1 資源優先度と横取り

優先度付き資源管理方式とは，他のプロセスに割り当てられた資源を**横取り**（preempt）することを可能にした資源管理方式である．プロセス間で共有しているなんらかの資源（CPU 時間や物理メモリなど）に

空きがない場合，すでに他のプロセスに割り当てられている資源を “奪って”，別のプロセスに割り当てることができる．各プロセスは**資源優先度**（resource priority）と呼ぶ属性を持ち，資源優先度の高いプロセスは資源優先度の低いプロセスに割り当てられた資源を “奪う” ことができる．

この資源管理方式を用いると，資源濫用攻撃から回復したり資源濫用攻撃を未然に防ぐことができる．占有プロセスが共有資源を濫用しても，占有プロセスの資源優先度を下げれば，他のプロセスは占有プロセスから資源を横取りしながらその処理を継続することができる．

優先度付き資源管理方式では，資源に空きがある場合には，資源優先度にかかわらず対等に資源の割当てが行われる．そのため，既存のオペレーティングシステムに優先度付き資源管理方式を組み込んだ場合でも，資源に空きがあれば，組込み前と同じ方式を用いて資源の割当てが行われ，システムの応答性やスループットを低下させることはない．資源が不足した状態であっても，同じ資源優先度を持つプロセス同士では，従来と同じ方式で資源の割当てが行われる．したがって，資源優先度の低いプロセス以外のプロセスは，従来とほぼ同様に資源割当てを受けることができ，提案方式を組み込んでも以前の応答性やスループットを損なうことはない．

資源優先度は，どのプロセスがどのプロセスから資源を奪ってよいかを定めたものであり，通常のスケジューリングで用いられる優先度とは若干異なる意味合いを持つ．直感的には “プロセスの信頼度” を表し，信頼度の高いプロセスは信頼度の低いプロセスから資源を奪ってよいということを表す．また，資源優先度は親プロセスの資源優先度を引き継ぐようになっており，資源優先度の低いプロセスから生成されたプロセスは低い資源優先度を持つ．さらに，あるプロセスの資源優先度を変更すると，その子孫の資源優先度も同じように変更されるようになっている．

3.2 運 用 例

本節では優先度付き資源管理方式を用いたシステムの運用例を示す．本論文で示す優先度付き資源管理方式では，任意段階の資源優先度を設定することが可能であるが，ここでは 3 段階の資源優先度を用いた運用例を示す．優先度の高い方から，`admin`，`normal`，`suspect` という 3 段階の資源優先度を設ける．`admin` は，UNIX における`ps` や`kill` などのプロセス制御用コマンドに割り当て，`normal` は通常のプロセスに割り当てる．`suspect` は，通常のプロセスより低い優先度が必要となったときに利用する．プロセスの資源優先度を変更するコマンドを用意し，そのコマンドにも`admin` 優先度を割り当てる．なお，`admin` 権限を持つプロセスは一部の特権プロセスのみであり，これらのプロセスが多くの資源を消費することはない．

3.2.1 信頼性の低いプログラムの実行

信頼性の低いプログラムを実行する場合，あらかじめそのプログラムの資源優先度を`suspect` に設定してから実行する．実際に，このプログラムが資源濫用攻撃を行ったとしても，そのプロセスの資源優先度`suspect` は他のプロセスの資源優先度`normal` よりも低いため，それらのプロセスの資源を奪うことはできない．したがって資源濫用攻撃は成立しない．

信頼性が低く資源優先度を`suspect` に設定されたプロセスであっても，資源濫用攻撃を行わなければ，資源優先度高い他のプロセスと対等に資源が割り当てられる．これは，システムの資源が不足しない限り，資源の横取りは行われないからである．したがって，信頼性は低いが実際には攻撃は行わないプロセスであれば，他のプロセスと対等に実行することができる．

3.2.2 信頼性の高いプログラムによる攻撃

2.1 節で述べたように，信頼性の高いプログラムであっても，悪質なデータを入力された場合には，資源濫用攻撃を引き起こしてしまう場合がある．このような場合であっても，上記の 3 段階の優先度を用いると，資源濫用攻撃を受けた状態から回復することができる．

あるプロセス X が資源濫用攻撃を行った場合，ユーザは`ps` コマンドを用いてプロセス X を特定し，そのプロセスの資源優先度を`suspect` に下げることができる．プロセス X が資源濫用攻撃を行っている間であっても，`ps` などのプロセス制御用コマンドは，資源濫用攻撃の影響なく迅速に実行することができる．なぜなら，これらのコマンドには`admin` 優先度が割り当てられており，プロセス X に割り当てられている`normal` 優先度よりも高い資源優先度を持つからである．プロセス X の資源優先度を`suspect` に下げた後には，他のプロセス（資源優先度`normal` を持つ）はプロセス X から資源を奪いながら処理を継続することができ，結果として資源濫用攻撃から回復することができる．

優先度付き資源管理機構を持たなければ，プロセス制御用のコマンドの実行に長い時間を必要とし，資源濫用攻撃からの回復は容易ではない．優先度付き資源管理方式を用いることによって，プロセス制御用のコマンドが迅速に起動できるため，占有プロセスの優先度を下げたり，場合によっては占有プロセスを強制終了させたりすることが可能となっている．

このように，資源優先度を変更するだけで資源割当てを制御できるため，優先度付き資源管理方式はその制御が簡明である．実時間システムやメインフレーム用のオペレーティングシステムが備えている資源割当ての制御機構は，アプリケーションの処理内容に応じた煩雑な設定を必要とし，各資源の物理量（CPU 時間の割合やメモリ使用量など）を逐一指定しなければならない．優先度付き資源管理方式では，そのような煩雑な設定を必要としないため，一般のエンド・ユーザであっても理解しやすいものとなっている．

4. 実　　装

優先度付き資源管理における「資源の横取り」を実現するには，次の点を考慮しなければならない．第 1 に，資源の横取りは迅速に行わなければならない．資源の横取りに長い時間を要すると，占有プロセスから資源を奪うのに長い時間がかかりプロセスの応答性が低下する．第 2 に，資源の横取りに多くの資源を必要としてはならない．資源の横取りに多くの資源を要すると，連鎖的に資源不足を引き起こし，資源濫用攻撃から回復できない．

本章では，物理メモリ，ディスク I/O のバンド幅，CPU 時間という 3 種の資源を対象に，優先度付き資源管理方式の実現手法を述べる．Linux 2.4.7 カーネルをベースに実装を行ったため，ここで述べる実現手法は Linux 2.4.7 カーネルの実装に依存したものとなっている．しかし，優先度付き資源管理方式は，他のオペレーティングシステムであっても同様に実現可能である．

4.1 物理メモリ

優先度付き資源管理方式をメモリ管理に適用すると，資源優先度の高いプロセスは資源優先度の低いプロセスの物理ページ（ページ・フレーム）を横取りできるようになる．資源優先度を考慮したメモリ管理機構では，資源優先度を考慮して犠牲ページ（victim page）を選択しなければならない．また，物理ページの横取りは，スワップ領域への書出しやスワップ領域からの読込みが必要となるため，ディスク I/O にともなう遅延が生じる．したがって，この遅延を隠蔽する実装上の工夫が必要となる．

4.1.1 犠牲ページの選択アルゴリズム

通常のメモリ管理機構では，LRU の近似アルゴリズムを用いて犠牲ページの選択を行う．優先度付き資源管理機構では，各物理ページに対する参照の履歴だけではなく，資源優先度を考慮した犠牲ページの選択を行う．あるプロセスが物理ページを要求してくると，そのプロセスより低い資源優先度を持つプロセスの使用している物理ページの中から犠牲ページを選択し，それらの中から最も長い間参照されていないページを犠牲ページとする．

物理ページを要求してきたプロセスより資源優先度の低いプロセスがなかったり，資源優先度の低いプロセスに割り当てられた物理ページがなかった場合には，同じ優先度を持つプロセスに割り当てられたページの中から，最も長い間参照されていないページを犠牲ページとして選択する．

上記の方針に従って犠牲ページを選択すると，資源優先度の高いプロセスに割り当てられた物理ページは，長い間参照されていなくとも，優先度の低いプロセスに割り当てられることはない．したがって，長い間参照されていないメモリ・ページがスワップ・アウトされず，システム全体の物理ページの利用効率が低下する．この問題を回避するため，資源優先度の高いプロセスに割り当てられた物理ページであっても，長い間参照されていないページは犠牲ページの選択対象とするようにする．Linux における実現方法の詳細については次項で述べる．

なお，犠牲ページを選択する際，物理ページを使用しているプロセスを特定し，その資源優先度を知る必要がある．物理ページが 1 つのアドレス空間にマップされている場合，そのアドレス空間に対応するプロセスがその物理ページを使用しているプロセスである．しかし，ファイルキャッシュのように複数のプロセスが共有している物理ページでは，その物理ページを使用しているプロセスを特定するのが難しい．

この問題を回避するため，優先度付き資源管理方式では，最初に物理ページを割り当てたプロセスをその所有者と見なすようにした．物理メモリを占有するには，多数の物理メモリ割当て要求を出す必要があるため，このように見なしても問題は生じない．

4.1.2 Linux 上での実装

前項で述べた犠牲ページの選択アルゴリズムを Linux 2.4.7 カーネルを対象に実装を行った．Linux では，Mach で用いられているセカンド・チャンス付き FIFO [4] の一種を用いてメモリ管理を行っている．

この方式では，各物理ページは，*active*，*inactive-dirty*，*inactive-clean* の 3 つの FIFO キューのいずれかに属する．最近参照されたページは active キューに入り，しばらく参照されていないページは inactive-dirty または inactive-clean キューに入る．物理ページの空きがある閾値より少なくなると，ページデーモンが起動し，active キューにあるページのうち参照さ

れていないページを inactive-dirty キューに移す．その後，inactive-dirty キューにあるいくつかのページを inactive-clean キューに移す．このとき，参照ビットが立っていれば，active キューに戻し，更新ビット（dirty bit）が立っていればディスクへの書出しを行う．犠牲ページは inactive-clean キューから選択する．

優先度付き資源管理方式を組み込んだ Linux では，inactive-dirty または inactive-clean キューが空でないときは，従来の Linux と同じ方式で犠牲ページを選択する．これによって，優先度の高いプロセスに割り当てられた物理ページうち，長い間参照されていないページは優先度の低いプロセスによって再利用できる．

inactive-dirty，inactive-clean キューがともに空の場合，active キューから資源優先度の低い物理ページを選択し，最近，参照されているかどうかにかかわらず，そのページを inactive-dirty キューに移す．これによって物理ページの横取りが実現できる．

4.1.3 横取り遅延の隠蔽

物理ページの横取りに要する遅延時間を隠蔽するため，2つの最適化を行っている．第1に，あらかじめ一定数の物理ページを予約しておき，物理ページの横取りが起きるときは，この予約されたページから割当てを行う．これによって，犠牲ページの内容をディスクに書き出す待ち時間が隠蔽される．第2に，物理ページの横取りを貪欲（eager）に行う．物理ページを横取りする際，貪欲に数ページの物理ページを横取りしておく．物理ページを横取りした優先度の高いプロセスは，続けて数ページの物理ページを横取りする場合が多い．そのため，貪欲に横取りを行うことによって，予約した物理ページの不足を避けることができる．

4.2 ディスク I/O

優先度付き資源管理方式では，ディスク I/O のスケジューリングも資源優先度を考慮して行われる．優先度の高いプロセスが発行したディスク I/O 要求は，優先度の低いプロセスの発行したそれを “追い越す” ことができる．

通常の Linux では，各プロセスの発行したディスク I/O 要求はキューに入れられ，エレベータ・アルゴリズムを用いてスケジューリングされる．優先度付き資源管理方式を導入した Linux では，キューに入ったディスク I/O 要求は，そのディスク I/O を発行したプロセスの資源優先度順に並べ替えられ，資源優先度の高いものから実行される．同じ資源優先度を持つディスク I/O 要求は，通常の Linux と同様に，エレベータ・アルゴリズムに従って並べ替えられる．

ディスク I/O のスケジューリングでは，ページデーモンの発行したディスク I/O の扱いに注意する必要がある．ページデーモンの発行したディスク I/O は，ページデーモンの資源優先度ではなく，ページデーモンにページ・イン，ページ・アウトを要求したプロセスの資源優先度に応じてスケジューリングされるべきである．さもなくば，資源優先度の低いプロセスが頻繁にページングを行うことによって，資源優先度の高いプロセスのディスク I/O を妨げることができてしまう．

この問題を回避するため，ページデーモンはディスク I/O を発行する前に，自身の資源優先度を変更し，ページ・イン，ページ・アウトを要求してきたプロセスのそれに設定する．これによって，ページデーモンの発行したディスク I/O 要求は，ページ・イン，ページ・アウトを要求してきたプロセスの資源優先度に設定される．

4.3 CPU 時間

CPU 時間のスケジューリングに関しては，多くのオペレーティングシステムにおいてすでに優先度付き資源管理が行われている．Linux においても優先度付き資源管理が行われている．一般のエンド・ユーザは各プロセスのナイス値（nice value）を変更することによって，各プロセスの CPU 時間優先度を制御することができる．ナイス値を大きく設定すると，プロセスの CPU 時間優先度は低くなり，ナイス値を小さく設定すると，プロセスの CPU 時間優先度は高くなる．

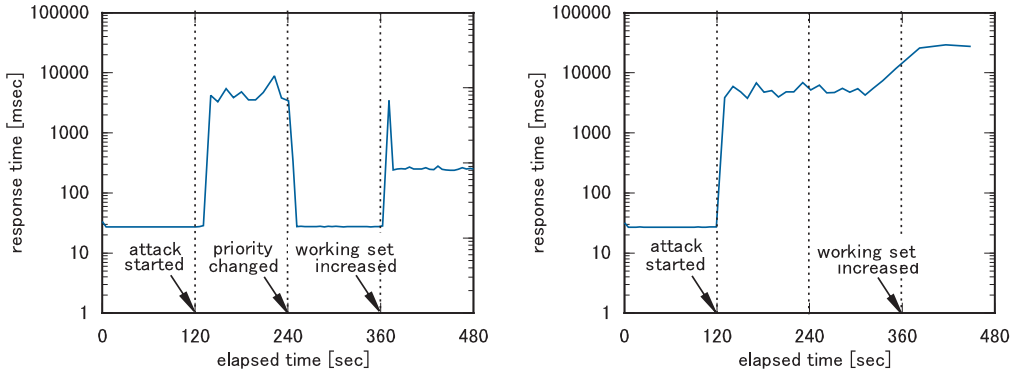
現在のプロトタイプ・システムでは，ナイス値を用いて CPU 時間の資源優先度を制御している．資源優先度を高く設定するとナイス値を低く設定し，資源優先度を低く設定するとナイス値を高く設定する．これによって，資源優先度の高いプロセスに優先的に CPU 時間が割り当てられるようになる．

しかし，ナイス値を用いた CPU 時間の制御では，厳密に優先度付き資源管理を実現したことにはならない．なぜなら，資源優先度の低い（すなわちナイス値の大きい）プロセスを複数実行すれば，CPU 時間の多くを占有することができ，資源濫用攻撃を行うことができるからである．CPU 割当ての制御方法については，すでに多くの研究が知られており，たとえば Performance Isolation [5] で用いられている手法が利用できる．

5. 議　　論

5.1 資源割当ての公平性

優先度付き資源管理方式を用いた場合であっても，`normal` 優先度を持つ通常のプロセスに関しては，資



(a) Preemptive Resource Management (b) Linux 2.4.7

図 2 対話的なプロセスの応答時間
Fig. 2 Response time of interactive process.

源割当ての公平性が保たれるようになっている．これは，同一の資源優先度を持つプロセス間では，通常の資源割当てが行なわれるようになっているからである．

また，`normal` 優先度より高い`admin` 優先度を持つプロセスは，一部の特権プログラム（`kill` など）に限定されており，それらの特権プロセスが長期間にわたって動作したり，大量の資源を消費することはない．したがって，これらのプロセスが`normal` 優先度のプロセスの資源を長期間横取りし，`normal` 優先度のプロセスが資源不足に陥ることはないようになっている．

資源濫用攻撃は，従来の一般的なオペレーティングシステムが提供する資源割当ての公平性を悪用した攻撃であるといえる．優先度付き資源管理方式の特徴は，資源不足に陥った状況に限り，選択的に資源割当ての公平性を放棄することができるという点にある．ユーザが明示的に`suspect` と指定したプロセスに対してのみ，一時的に資源割当ての公平性を放棄することによって，資源濫用攻撃に対する耐性を高めている．

6. 実　　験

優先度付き資源管理方式が資源濫用攻撃に対して耐性を有することを示す実験を行った．この実験では，資源濫用攻撃を行っても，優先度付き資源管理のもとでは，対話的なプロセスの応答性が低下しないことを示す．実験は，Pentium III（733 MHz）のプロセッサ，SDRAM 128 MB，20 GB の UltraDMA/66 HDD を搭載した PC/AT 互換機上で行った．プロトタイプ・システムの実装は Linux 2.4.7 をベースに行った．

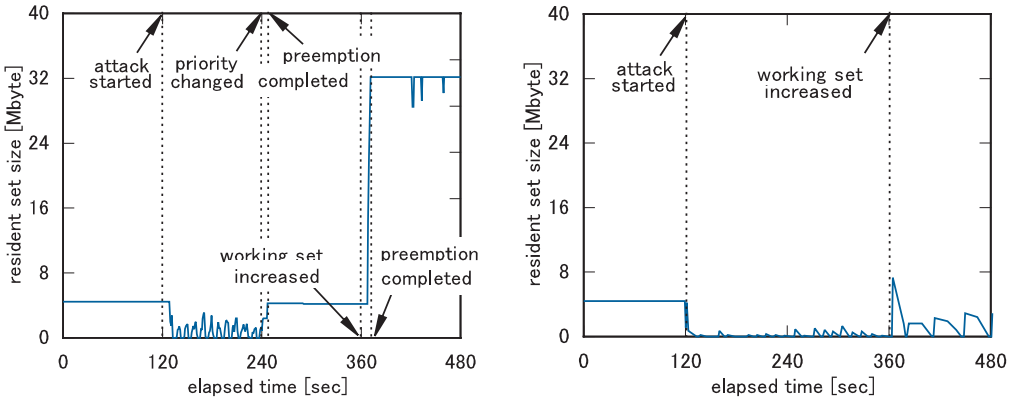
実験に用いた対話的なプロセスと資源濫用攻撃を行うプロセスは次のとおりである．これらのプロセスは 2.1 節で用いたプロセスとほぼ同じである．

対話的なプロセス： 対話的なプロセスの振舞いをエミュレートするプログラムであり，データ配列を先頭から末尾までアクセスし，5 秒間スリープするという動作を繰り返す．スリープ時間はユーザからの入力を待つ時間に相当し，データ配列をアクセスする時間はユーザからの入力に応じた処理を行う時間に相当する．アクセスする配列のサイズは最初は 4 MB に設定し，実験の途中で 32 MB に設定する．

占有プロセス： CPU 時間，物理メモリ，ディスク I/O を同時に占有するプロセスを 5 つ同時に動作させる．それぞれの占有プロセスは，25 MB のメモリ領域に繰り返しアクセスするという動作を行う．5 つのプロセスで計 125 MB のメモリ領域に頻繁にアクセスを行うため，スラッシングを引き起こし，ページ・イン，ページ・アウトのために CPU 時間とディスク I/O も占有する．

図 2 に対話的なプロセスの応答時間を示す．図 2 (a) は優先度付き資源管理方式を用いた場合の応答時間であり，図 2 (b) は通常の Linux 2.4.7 における応答時間である．また，物理メモリの割当て状況を調べるため，対話的プロセスの RSS（resident set size）の計測を行った．**図 3** にその結果を示す．図 3 (a) は優先度付き資源管理方式を用いた場合であり，図 3 (b) は通常の Linux 2.4.7 の場合である．

実験開始後の 120 秒間は，占有プロセスは実行していない．図 2 に示したように，優先度付き資源管理を行った場合もそうでない場合も，対話的なプロセスの応答時間はほぼ 27 msec で一定である．図 3 から分かるように，対話的プロセスは 4 MB のメモリ領域を割り当てられている．これは，対話的プロセスがアク



(a) Preemptive Resource Management (b) Linux 2.4.7

図 3 対話的プロセスの Resident Set Size (RSS)
Fig. 3 Resident set size (RSS) of the interactive process.

セスするデータ配列が 4 MB に設定されているためである．

実験開始後 120 秒経過した時点で，占有プロセスの実行を開始する．この時点では，占有プロセスの資源優先度は対話的プロセスのそれと同じに設定されている．そのため，対話的プロセスの応答時間は 10,000 msec にまで増加している．図 3 から分かるように，対話的プロセスにはほとんど物理メモリが割り当てられていない．

実験開始後 240 秒経過した時点で，占有プロセスの資源優先度を下げる．図 2 (a) から分かるように，対話的プロセスの応答時間は急速に回復し，再び 27 msec でほぼ一定となっている．これは，占有プロセスの資源優先度を下げたため，対話的プロセスの資源優先度が相対的に高くなり，占有プロセスの物理メモリを横取りしたためである．図 3 (a) が示すように，対話的なプロセスの RSS は 4 MB にまで回復している．

実験開始後 360 秒経過した時点で，対話的なプロセスのアクセスするデータ配列を 4 MB から 32 MB に増加させる．図 2 (a) に示したように，対話的なプロセスの応答性は一時的に 10,000 msec まで低下するが，すぐに 250 msec にまで回復する．一時的に応答性が低下したのは，物理メモリの横取りが完了していなかったためである．図 3 (a) から分かるように，その後，すぐに物理メモリの横取りが完了し，対話的なプロセスに 32 MB のメモリが割り当てられ，応答性が回復する．対話的なプロセスの応答時間が 27 msec から 250 msec に増加したのは，アクセスするデータ配列の大きさが 4 MB から 32 MB にまで増加したためである．

7. 関 連 研 究

従来のオペレーティングシステムにおける資源管理方式は，プロセス間の公平性を達成することを目的とすることが多かった．近年になって，さまざまな状況において，優先的な資源割当ての必要性が認識され，さまざまな資源管理方式が提案されている．

Stealth [6] は，ワークステーション・クラスタ上での負荷分散を目的としたスケジューラである．Stealth は 2 段階の資源優先度を持ち，優先度の高いプロセスは優先度の低いプロセスの資源を横取りすることができる．我々の提案する優先度付き資源管理方式と Stealth は，その目的が異なるため，多くの点で異なる特徴を持つ．優先度付き資源管理方式では，多段階の資源優先度を用いることができ，動的に資源優先度を変更できる．それに対し，Stealth の優先度は 2 段階に固定されており，各プロセスの資源優先度を動的に変更することはできない．また，Stealth では資源の横取りを迅速に行うための工夫はなされていない．

文献 7) において，Liedtke らは物理メモリの濫用攻撃を防ぐ手法を提案している．Liedtke らの手法はマイクロ・カーネルを対象としており，物理メモリを要求した時点でメモリに空きがなければ，自分の持つ物理メモリをマイクロ・カーネルに返還しなければならない．この方式では，占有プロセスが占有した物理メモリは他のプロセスに返還されることがない．そのため，物理メモリの濫用攻撃から回復することは難しい．

Performance isolation [5] は，共有メモリ型のマルチプロセッサ用に設計された資源管理方式である．SPU と呼ぶプロセス・グループに対してあらかじめ定められた量の資源が割り当てられ，SPU 間では資源の貸し

借りが可能となっている．資源を貸し出したSPUは必要に応じてその資源を取り返すことができる．資源を取り返すことができるという点で，我々の方式と類似しているが，ファイルキャッシュのような共有資源に対しては資源を取り返すことができない．したがって，ファイルキャッシュを占有する資源濫用攻撃が依然として可能である．

文献8)では，lotteryスケジューリング[9]を拡張し，チケットのやりとりによってプロセスに割り当てられる資源量を調整する機構が提案されている．チケットをやりとりするポリシーを変更することによって，さまざまなスケジューリングが可能となっている．しかし，資源濫用攻撃に耐性を有するようなチケットのやりとりの手法は知られていない．

MS Manners[10]は，ユーザ・プロセスとの協調によって資源割当てを制御する機構である．悪意ある占有プロセスは協調動作をしないため，MS Mannersを資源濫用攻撃の防御に用いることはできない．文献11)では，コンパイラによる静的解析を用いて物理メモリの占有を防止する方式が示されている．本論文で対象とする占有プロセスの場合，ソース・コードが入手できるとは仮定できないため，この方式を用いることはできない．

本論文で示した優先度付き資源管理方式は，実時間システムのために開発された資源管理方式とは異なった特徴をもつ．実時間システム向けの資源管理方式は，資源割当てを保証することを目的としており，アプリケーションごとに必要な資源量を指定し，入場制限（admission control）を行う．優先度付き資源管理方式は，資源割当てを保証することは目的としてはおらず，むしろ，資源が不足した時に選択的にプロセスの公平性を放棄できるようにしたものである．したがって，実時間処理向けの資源管理方式とも組み合わせて用いることが可能である．たとえば，入場制限によって信頼性の高いプロセスの実行が拒否された場合，資源優先度の低いプロセスから資源を横取りし，そのプロセスの入場を許可することができる．

8. まとめ

本論文では，資源濫用攻撃に耐性を有する資源管理方式として，優先度付き資源管理方式の提案を行った．資源濫用攻撃とは，意図的に計算資源を濫用し，他のプロセスの応答性を著しく低下させる攻撃である．優先度付き資源管理方式では，資源を濫用するプロセスから資源を奪い，それらの資源を他のプロセスに割り当てることが可能となっている．そのため，資源濫用攻撃を未然に防いだり資源濫用攻撃から回復することができる．実際，優先度付き資源管理方式をLinux 2.4.7に実装し，資源濫用攻撃を防ぐことができることを確認した．

本論文で示した優先度付き資源管理方式では，資源優先度の設定は手動で行わなければならない．侵入検知システムで用いられている技術を援用すれば，資源濫用攻撃を行っているプロセスを自動的に判別し，資源優先度の設定が自動化できるだろう．また，本論文で示した手法では，Xサーバのように複数のプロセスが共有するサーバ・プロセスに対しては適用が難しい．これはプロセスより細かい粒度での資源横取りが難しいためである．Resource container[12]やScout[13),14)]で用いられているパスなど，プロセスより細かい粒度での資源管理を可能とする仕組みと組み合わせれば，共有サーバに対する資源濫用攻撃も防止できると期待される．

参考文献

1) NTT DoCoMo: Booby Trap Mail. http://www.nttdocomo.co.jp/itazura.html
2) Rajkumar, R., Juvva, K., Molano, A. and Oikawa, S.: Resource kernels: A resource-centric approach to real-time and multimedia systems, *SPIE*, Vol.3310 (1997).
3) Bruno, J., Gabber, E., Özden, B. and Silberschatz, A.: The Eclipse Operating System: Providing Quality of Service via Reservation Domains, *Proc. USENIX Annual Technical Conference*, pp.235–246 (1998).
4) Draves, R.P.: Page Replacement and Reference Bit Emulation in Mach, *Proc. Second USENIX Mach Symposium*, pp.201–212 (1991).
5) Verghese, B., Gupta, A. and Rosenblum, M.: Performance Isolation: Sharing and Isolation in Shared-Memory Multiprocessors, *Proc. 8th International Conference on Architectural Support for Programming Languages and Operating Systems* (1998).
6) Krueger, P. and Chawla, R.: The Stealth Distributed Scheduler, *Proc. 11th IEEE International Conference on Distributed Computing Systems*, pp.336–343 (1991).
7) Liedtke, J., Islam, N. and Jaeger, T.: Preventing denial-of-service attacks on a μ-kernel for WebOSes, *Proc. 6th IEEE Workshop on Hot Topics in Operating Systems* (*HotOS*), Chatham (Cape Code), MA (1997).
8) Sullivan, D.G. and Seltzer, M.I.: Isolation with Flexibility: A Resource Management Frame-

work for Central Servers, *Proc. 2000 USENIX Annual Technical Conference* (2000).
9) Waldspurger, C.A. and Weihl, W.E.: Lottery Scheduling: Flexible Proportional-Share Resource Management, *Proc. USENIX Symposium on Operating Systems Design and Implementation* (1994).
10) Douceur, J.R. and Bolosky, W.J.: Progress-based regulation of low-importance processes, *Proc. 17th ACM Symposium on Operating Systems Principles* (*SOSP'99*) (1999).
11) Brown, A.D. and Mowry, T.C.: Taming the Memory Hogs: Using Compiler-Inserted Releases to Manage Physical Memory Intelligently, *Proc. 4th USENIX Symposium on Operating Systems Design and Implementation* (*OSDI2000*) (2000).
12) Banga, G., Druschel, P. and Mogul, J.C.: Resource Containers: A New Facility for Resource Management in Server Systems, *Proc. USENIX Symposium. on Operating Systems Design and Implementation*, pp.45–58 (1999).
13) Mosberger, D. and Peterson, L.L.: Making Paths Explicit in the Scout Operating System, *Proc. USENIX Symposium on Operating Systems Design and Implementation*, pp.153–167 (1996).
14) Spatscheck, O. and Peterson, L.L.: Defending Against Denial of Service Attacks in Scout, *Proc. USENIX Symposium on Operating Systems Design and Implementation*, pp.59–72 (1999).

(平成 15 年 2 月 3 日受付)
(平成 15 年 4 月 26 日採録)

河野 健二（正会員）

昭和 45 年生．平成 9 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．理学博士．平成 12 年 1 月より電気通信大学情報工学科に勤務．現在，同講師．オペレーティングシステムおよびミドルウェア等のシステムソフトウェア，分散および並列処理に興味を持つ．IEEE/CS，ACM 各会員．平成 11 年度情報処理学会論文賞受賞．

金子 済

昭和 51 年生．平成 12 年東京大学理学部情報科学科卒業．平成 14 年同大学大学院理学系研究科情報科学専攻修了．現在，同大学大学院情報理工学系研究科コンピュータ科学専攻博士課程に在学中．

清水謙多郎（正会員）

昭和 32 年生．昭和 60 年東京大学大学院理学系研究科情報科学専攻博士課程修了．理学博士．平成 10 年より同大学大学院農学生命科学研究科教授．生命情報学，並列・分散処理，オペレーティングシステムの研究に従事．ACM，ACS，IEEE Computer Society，電子情報通信学会，日本バイオインフォマティクス学会，日本農芸化学会各会員．