システムソフトウェアと
オペレーティング・システム 87－15
（２００１．６．２９）

メモリ占有DoS攻撃の防止: 優先度付きメモリ管理

金子 済[1]　河野 健二[2,3]　清水 謙多郎[4,1]

[1] 東京大学 大学院理学系研究科 情報科学専攻
[2] 電気通信大学 情報工学科
[3] 科学技術振興事業団 さきがけ研究 21
[4] 東京大学 大学院農学生命科学研究科 応用生命工学専攻
wataruk@is.s.u-tokyo.ac.jp, kono@cs.uec.ac.jp, shimizu@bi.a.u-tokyo.ac.jp

要旨

計算機システムを攻撃し損害を与えるプログラムをインターネットから入手・実行する危険性が高まりつつある。アクセス制御を中心とした従来のOSの保護機構では，計算機資源を濫用し計算機のサービスを停止させるサービス拒否攻撃に対処することは難しい。本論文では，物理メモリを濫用するサービス拒否攻撃を防ぐことを目的とした，優先度付き物理メモリ管理方式を提案する。提案方式を用いることにより，資源を濫用するサービス拒否攻撃があってもその影響をサンドボックスすることができ，他のプロセスは資源濫用の影響を受けることなく処理を継続できる。優先度付きメモリ管理方式では，メモリ優先度の高いプロセスがメモリ優先度の低いプロセスから物理メモリを奪取でき，攻撃プロセスのメモリ優先度を低下させるだけで，サービス拒否攻撃の影響を避けることができる。

Defending against Memory-Monopolizing DoS Attacks: Prioritized Memory Management

Wataru Kaneko[1]　Kono Kenji[2,3]　Kentaro Shimizu[4,1]

[1]Department of Information Science, Faculty of Science, University of Tokyo
[2]Department of Computer Science, University of Electro-Communications
[3]PRESTO, Japan Science and Technology Corporation
[4]Department of Biotechnology, Faculty of Agriculture, University of Tokyo

Abstract

Downloading executable codes from the Internet increases the risk of executing malicious codes. Commodity operating systems can control access rights of downloaded code, but can do little to prevent resource-monopolizing Denial-of-Service (DoS) attacks. In this paper, we focus on memory-monopolizing DoS attacks, and propose the prioritized memory management to defend the attacks. Using this system, the user can sandbox the memory-monopolizing attacks, and the attacker process cannot affect other processes. On our system, each process has memory-priority. Higher-prioritized process can steal physical memory from lower-prioritized process, but the reverse is forbidden. As a result, by prioritizing the attacker process lower, the memory-monopolizing attacks are sandboxed.

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が損害を被る危険性がある。

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し，他のプログラムが処理を継続できないようにする攻撃が知られている。このような形態の攻撃を「資源濫用による**サービス拒否攻撃**(Denial-of-Service attack)」という。資源濫用によるサービス拒否攻撃は現実の脅威であり，たとえば，大量の資源を消費するアプレットを作成し，それをウェブ経由で配布することが簡単にできる。

従来のオペレーティングシステム (OS) が提供してきた保護機構では，資源濫用によるサービス拒否攻撃を防ぐのは難しい。なぜなら，従来の OS が提供してきた保護機構はアクセス制御を主としており，プロセスによる資源濫用を防ぐことを目的としたものではないからである。

本論文では，物理メモリを占有するサービス拒否攻撃を対象に，その防御方式を提案する。不正なプロセスが物理メモリを占有すると，他のプロセスに割り当てる物理メモリが不足するだけでなく，カーネル内で使用するファイルキャッシュやバッファ等のメモリ資源も不足し，システム全体の処理能力が著しく低下してしまう。実際，物理メモリが占有されてしまうと，計算機がフリーズしたのと同じ状態になり，攻撃を仕掛けているプロセスの終了を待つか，システムの再起動を行わなければならない。

資源濫用攻撃の特徴は，システムの処理能力不足による過負荷状態と資源濫用攻撃による過負荷状態とを区別することが本質的に難しいという点にある。たとえ多くの資源を使用しているプロセスであっても，ユーザにとって有用な処理を行っているのであれば資源濫用攻撃とはならない。すなわち，サービス拒否攻撃であるか否かはユーザの主体的判断によって決まるものであって，システムによって機械的に判断できるものではない。

そのため，資源濫用攻撃を防ぐには，ユーザの主体的判断によってプロセスの資源割り当てを制御できる必要である。このような制御機構には次の三点が備わっている必要がある。

- **柔軟な資源割り当て**: 多くの資源を必要とするプロセスであっても，ユーザが資源を濫用していると判断しない限り，必要に応じて資源割り当てを受けることができる。
- **資源濫用のサンドボックス化**: 資源を濫用しているとユーザが判断した場合，その影響が他のプロセスに及ばないようにできる。すなわち，資源を濫用しているプロセスだけが資源不足による性能低下を被り，他のプロセスは必要に応じて資源を確保することができる。複数のプロセスが連携して資源濫用攻撃を仕掛ける場合であっても，それらプロセスをまとめてサンドボックスに入れ，他のプロセスに影響を及ぼさないようにすることができる。
- **簡明な資源割り当てポリシー**: ユーザによる資源割り当て制御を簡明にするため，プロセスの定性的な属性を制御するだけで資源割り当て制御ができる。言い換えると，資源の物理的容量を定量的に指定することなく，資源割り当てのポリシーを変更できなければならない。

上記の三点を実現する資源割り当て制御機構として，本論文では**優先度付き物理メモリ管理機構**を提案する。このメモリ管理機構を用いると，各プロセスのメモリ優先度を制御するだけで，資源濫用の危険性のあるプロセスをサンドボックスに入れ，その影響を押さえることができる。たとえば，アプレットを実行する Java 仮想機械をサンドボックスに入れておけば，大量の資源を消費するアプレットを起動してもウェブブラウザはその影響を受けることがない。また，資源不足により計算機がフリーズした場合でも，UNIX における ps や kill コマンド，windows におけるタスクマネージャなどのプロセス制御プログラムを迅速に起動し，攻撃を仕掛けているプロセスを停止させることができる。

以下，第 2 節では，既存の資源管理機構では資源を濫用するサービス拒否攻撃に対処しにくいことを示す。第 3 節では，優先度付き物理メモリ管理機構を提案し，第 4 節でその実装方式を示す。第 5 節では，実験によって，優先度付きメモリ管理機構を用いて資源濫用攻撃が防げることを示す。第 6 節は関連研究を示す。第 7 節で本論文の結論を述べる。

2　既存の資源管理機構による防御

本節では，資源濫用攻撃の防御策として既存の資源管理方式を用いた場合の限界について論じる。既存の資源管理方式は，上限値方式，資源予約方式，プロセッサ優先度方式の三つに大別できる。以下，各資源管理機構の限界を示す。

2.1　上限値方式

上限値方式とは，各プロセスが利用できる資源の量に上限を設ける方式である。信頼性の低いプロセスが使用できるメモリ量に上限を設ければ，このプロセスによるメモリの占有を未然に防ぐことができる。しかしこの方式には次の二つの問題がある。

- **上限値設定が困難**: 上限値方式では，使用してよい資源の量を適切に定めるのが難しい。物理メモリの使用量を制限する場合，指定した物理メモリ量が少なすぎると，資源を濫用していないプロセスであっても処理を継続できない。逆に指定した物理メモリ量が多すぎると，資源濫用攻撃が可能になってしまう。特に，インターネットからダウンロードしたプログラムの場合，そのプログラムを実行することなく適切な上限値を定めなければならず，上限値の設定が一層難しい。

- **複数のプロセスによる攻撃に脆弱**: プロセスごとに上限値を定める方式では，複数のプロセスが連携して資源濫用攻撃を仕掛けてきた場合に対処できない。たとえば，あるウェブページに置かれた複数のアプレットが連携して攻撃を仕掛けるような場合，Java 仮想機械の消費する資源に上限値を定めても，複数の Java 仮想機械の消費する資源の総和が上限値を超えてしまい，結果として資源濫用攻撃となってしまう。

2.2　資源予約方式

資源予約方式とは，あるプロセスが必要とする資源をあらかじめ予約しておくことのできる方式である。この方式を用いると，あらかじめ必要な資源を予約しておき，資源濫用攻撃から保護すべきプロセスが資源濫用攻撃の影響を受けないようにできる。

- **予約量の設定が困難**: 元来，資源予約機構はリアルタイム・システムにおけるサービス保証 (Quality-of-Service) を実現するために開発されたものであり，サービス保証に必要な資源量をあらかじめ知ることのできるプログラムを対象としている。そのため，資源の使用量をあらかじめ知ることのできないプログラムには適用しにくい。通常のプログラムの多くは，ユーザの入力に応じて必要な資源の量が大きく変化することが多く，あらかじめ一定量の資源を予約しておくことが難しい。

- **未使用の予約資源**: ユーザの入力によって必要な資源量が変わるプログラムでは，ユーザの入力次第では予約した資源の一部しか使用されず，残りの予約資源が無駄になってしまう。

2.3　プロセッサ優先度方式

プロセッサ優先度方式とは，多くの資源を使っているプロセスのプロセッサ優先度を下げ，それ以上資源を消費しないようにする方式である。物理メモリの場合，プロセッサ優先度を下げれば，そのプロセスが使用していたページフレームがスワップアウトされ，他のプロセスがそのページフレームを再利用できると期待される。

しかしプロセッサ優先度方式は，資源濫用攻撃を仕掛けてきたプロセスによる物理メモリの使用を直接的に制限できない。そのため，このプロセスは，プロセッサ優先度を最小にまで下げられても，物理メモリを占有するに十分なプロセッサの割り当てを受けることができる場合がある。したがって，プロセッサ優先度方式は必ずしも攻撃を防ぐことができるとは限らない。

3　優先度付き物理メモリ管理

本節では，物理メモリ占有攻撃を防ぐための機構として，優先度付き物理メモリ管理方式を提案する。

優先度付き物理メモリ管理機構では，すべてのプロセスは**物理メモリ優先度**と呼ぶ属性をもつ。メモリ優先度とは，各プロセスにどの程度優先的に物理メモリを割り当てるかを決める属性であり，優先度の高い方から順に`admin`，`normal`，`danger` の三つの属性を用意している。原理上は任意段階の優先度を

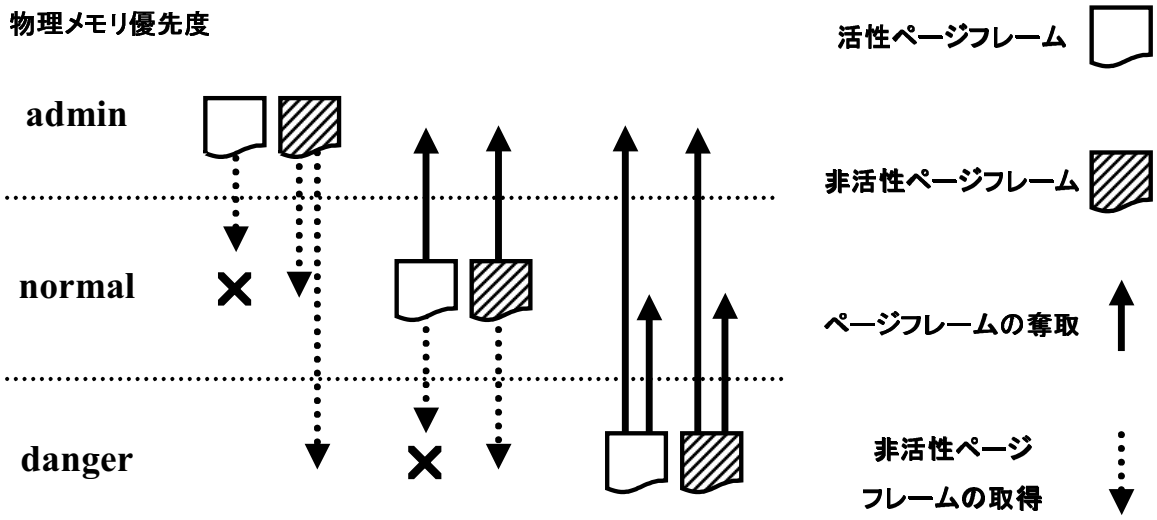

図 1: ページフレーム再配置戦略

設けることができるが，実用上はこの三つの属性で十分であろうと考えている。

- admin: もっとも高い優先度であり，資源濫用攻撃の下でも迅速に起動・実行する必要のあるプロセス制御プログラムに与える。
- normal: デフォルトで与えられる優先度。
- danger: もっとも低い優先度であり，normal 優先度を与えると物理メモリを占有する可能性のあるプロセスに与える。

上記の三段階の優先度を用いると，次のようにして資源の濫用によるサービス拒否攻撃を防ぐことができる。Java アプレットが物理メモリを占有し，ウェブブラウザや他のアプリケーションがフリーズ状態に陥ったとしよう。admin 優先度が与えられたプロセス制御プログラムは資源濫用攻撃の下でも迅速に起動・実行されるため，プロセス制御プログラムを起動し，著しくメモリ使用量の多い攻撃を仕掛けているプロセスを特定することができる。Java アプレットが攻撃を仕掛けていることが特定できたら，そのアプレットを実行している Java 仮想機械のメモリ優先度をdanger に変更し，アプレットによるメモリ濫用攻撃の影響が他のプロセスに及ばないようにすることができる。

このように，優先度付きメモリ管理方式を用いると，第 1 節で述べた (1) 柔軟な資源割り当て，(2) 資源濫用のサンドボックス化，(3) 簡明な資源割り当てポリシーの 3 点が達成できる。物理メモリ占有攻撃を受けない限り，全てのプロセスは同じnormal 優先度で動作し，従来の OS によるメモリ管理機構と同様，物理メモリは柔軟に各プロセスに割り当てられる。また，ユーザの主体的判断で，資源を濫用するプロセスのメモリ優先度をdanger に下げれば，その影響はnoraml およびadmin 優先度のプロセスには及ばず，資源濫用のサンドボックス化が実現できる。さらに，ユーザはメモリ優先度admin，normal，danger の三つを指示するだけでよく，各プロセスに割り当てるべき物理メモリ量を定量的に指定する必要がない。

3.1 ページフレーム再配置戦略

優先度付き物理メモリ管理では，各プロセスのメモリ優先度に従って，メモリ優先度の高いプロセスに対して優先的にページフレームを割り当てるようになっている。LRU による従来のページフレーム再配置戦略では，メモリ優先度という属性はなく，各ページフレームが**活性**であるか**非活性**であるかによって，ページフレームの再配置を行っている。活性ページフレームとは頻繁にアクセスされているページフレームのことであり，非活性ページフレームとはしばらくの間アクセスされていないページフレームのことである。

優先度付きメモリ管理方式では，ページフレームの活性・非活性にかかわらず，メモリ優先度の高いプロセスはメモリ優先度の低いプロセスからページフ

レームを奪取することができる。以下，優先度の低いプロセスからページフレームを奪うことをページフレームの**奪取**と呼ぶ。逆に，メモリ優先度の低いプロセスは，非活性ページフレームに限ってメモリ優先度の高いプロセスから取得することができる。以下，優先度の高いプロセスから非活性ページフレームを取得することをページフレームの**取得**と呼ぶ。これによって，メモリ優先度の低いプロセスが物理メモリの占有を試みても，その影響はメモリ優先度の高いプロセスにまで及びにくい。また，優先度の高いプロセスに割り当てられたページフレームであっても，非活性なページフレームは有効に活用されるようになっている。

この様子を図 1 に示す。admin 優先度のプロセスは，ページフレームを活性・非活性にかかわらず，より優先度の低いnormal およびdanger のプロセスからページフレームを奪取できる。同様に，normal 優先度のプロセスはより優先度の低いdanger のプロセスからページフレームを奪取することができる。非活性なページフレームに関しては，たとえadmin 優先度のプロセスのページフレームであっても，より優先度の低いnormal やdanger のプロセスが取得できる。同様にnormal 優先度のプロセスのページフレームはより優先度の低いdanger のプロセスでも取得できる。

4 実装

本節では，Linux 2.2.16 カーネルをベースに優先度付き物理メモリ管理を実現する方式を述べる。ここで述べる実装は Linux に依存したものであるが，優先度付きメモリ管理方式は特定の OS に依存する方式ではなく，Linux とは異なるメモリ管理方式を持つ OS 上でも実装できるものである。

4.1 ページフレームの所有者プロセス

メモリ優先度はプロセスに与えられた属性であるため，物理メモリを優先度によって管理するには，プロセスとページフレームとの対応付けをする必要がある。ユーザ空間にマップされたページフレームの場合，プロセスと一対一に対応しておりその対応付けは自明である。しかし，カーネルが内部に持つファイル・キャッシュやバッファ等のメモリ資源は，複数のプロセスが共有しているため，プロセスとの対応関係は自明ではない。もしこれら共有メモリ資源が優先度管理されていないと，こうした共有メモリ資源の占有から回復することができない。

そのため，カーネル内の共有メモリ資源についても優先度付きメモリ管理を行うため，次のような近似的手法を用いている。あるページフレームの割り当てを最初に要求したプロセスを，そのページフレームの**所有者プロセス**とし，共有メモリに割り当てられたページフレームはその所有者プロセスに対応付けている。すなわち，複数のプロセスによって共有されたページフレームは，そのページフレームを最初に要求したプロセスのメモリ優先度を使って管理される。物理メモリ占有攻撃を行うプロセスは，多量のページフレーム割り当て要求を行うので，この方法でも十分な近似になっていると期待できる。

4.2 犠牲者ページの選択方法

優先度付きメモリ管理方式では，ページフレームの割り当て要求に対し割り当てられる空きフレームがない場合，メモリ優先度を考慮して犠牲者 (victim) ページを選択する必要がある。優先度付きメモリ管理方式を実現するために，Linux の用いているページフレーム再配置アルゴリズムの変更を行った。

Linux では，セカンドチャンス法を用いて犠牲者ページを決定している。セカンドチャンス法では，全てのページフレームを環状の待ち行列に入れておき，各ページフレームの参照ビットを順に調べ，参照ビットがオフになっているページフレームを犠牲者として選択する。

このセカンドチャンス法を拡張し，優先度付き物理メモリ管理を行うように変更した。各ページフレームは，参照ビットに加え，所有者プロセスとアクセスタイムスタンプの情報を持つ。アクセスタイムスタンプは，参照ビットがオフであった期間を計測するのに使用する。

一周目のスキャンでは非活性なページフレームを探す。優先度の高いプロセスがページフレームの割り当てを要求してきた場合，優先度の低いプロセスのページフレームのうち参照ビットがオフになっているものか，メモリ優先度にかかわらず 30 秒以上参照ビットがオフのままであったページフレームを選択する。優先度の低いプロセスがページフレームの割

り当てを要求してきた場合，メモリ優先度にかかわらず 30 秒以上参照ビットがオフのままであったページフレームのみが選択される。

一周目のスキャンで犠牲者ページを選択できなかった場合，二周目のスキャンを行う。二周目のスキャンでは，非活性なページフレームは存在しないので，できる限り優先度の低いプロセスのページフレームを選択する。所有者プロセスの優先度がページフレームを要求しているプロセスの優先度より低い場合，参照ビットがオンでも犠牲者ページに選択する。所有者プロセスの優先度がページフレームを要求しているプロセスの優先度と等しい場合，参照ビットがオフである場合に限り犠牲者ページに選択する。

4.3 ページフレーム奪取の高速化

優先度の高いプロセスが優先度の低いプロセスからページフレームを奪取する場合，ページフレームの内容をバッキングストアに書き込む必要がある。この書き込みに長い時間を要すると，優先度の低いプロセスは優先度の高いプロセスの実行性能を低下させることができ，優先度の高いプロセスも資源濫用の影響を受けてしまう。

こうした事態を避けるため，次のようにしてページフレーム奪取の高速化を行った。優先度の高いプロセスのために未使用ページを常に用意しておく。優先度の高いプロセスからページフレームの割り当て要求が来ると，用意しておいたページフレームを直ちに割り当てる。これによって，バッキングストアへの書き込みによる遅延を隠蔽し，ページフレーム奪取による性能低下を押さえている。

用意しておいた未使用ページの量が低下すると，ページアウトデーモンをバックグラウンドで実行し，優先度の低いプロセスからページフレームを奪取し未使用ページの量を回復させている。

5 実験

物理メモリ占有攻撃に対し，優先度付き物理メモリ管理方式が有効であることを示すため，いくつかのベンチマークを行った。実験に使用した PC は，Pentium III 733MHz，SDRAM 128MB，UltraDMA/66 HDD 20GB から構成される。

5.1 マイクロベンチマーク

メモリ優先度の高いプロセスが，メモリ優先度の低いプロセスから，物理メモリを奪えることを確認する実験を行った。このマイクロベンチマークでは，測定開始後直ちに攻撃プロセスを起動し，100MB のメモリを繰り返しアクセスさせる。測定開始後 30 秒経過した時点で，被攻撃プロセスを起動し，64MB のメモリを確保して繰り返しアクセスさせる。測定開始から 60 秒経過した時点で，被攻撃側プロセスはアクセスするメモリ量を 32MB に減らし，測定開始から 120 秒後に実行を終了する。

それぞれのプロセスのアドレス空間にマップされた物理メモリ量 (Resident Set Size, RSS) の時間変化を記録した。比較のため，(1)Linux 2.2.16，(2) 攻撃側プロセスのプロセッサ優先度を下げた場合，(3) 優先度付きメモリ管理方式の三つの方式で測定を行った。なお，(2) の方式では`nice`値を 19 に設定し，プロセッサ優先度を低下させた。図 2，3，4 に結果を示す。

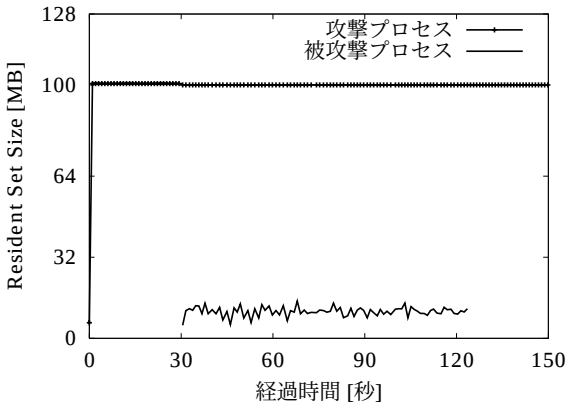

図 2: Linux の場合

Linux の場合，後から起動された被攻撃プロセスは，10MB 未満の不十分な物理メモリしか確保することはできていない。また，攻撃プロセスのプロセッサ優先度を下げても，攻撃プロセスは依然として 100MB の物理メモリを占有するに十分なプロセッサ割り当てを得ており，物理メモリの占有を解消することはできていない。この結果から，プロセッサ優先度の変更では物理メモリ占有攻撃を防ぐことができないと言える。

図 4 に優先度付き物理メモリ管理を行った場合の結果を示す。このベンチマークでは，攻撃プロセス

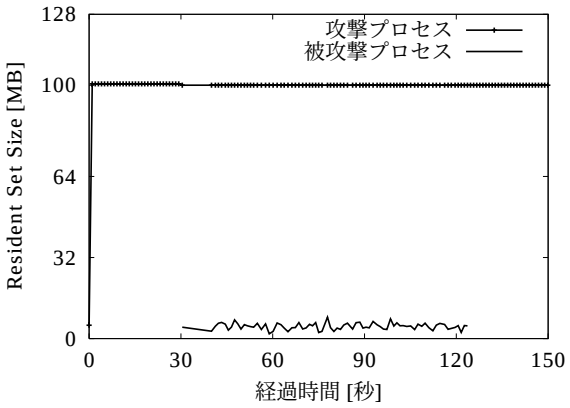

図 3: 攻撃プロセスのプロセッサ優先度を下げた場合

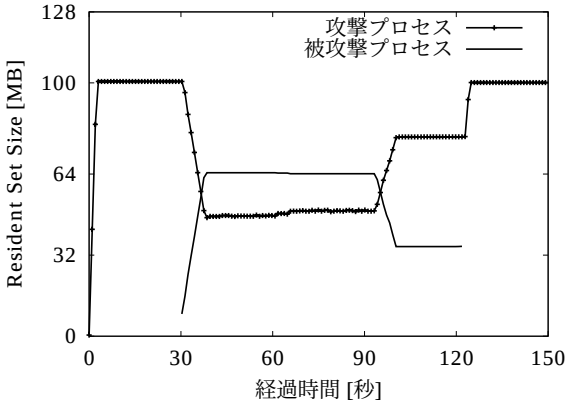

図 4: 優先度付き物理メモリ管理の場合

のメモリ優先度を`danger` に設定し，被攻撃プロセスのメモリ優先度を`normal` に設定した。被攻撃プロセスは攻撃プロセスの物理メモリを奪取して，9 秒間以内に 64MB の物理メモリの確保に成功している。測定開始後 90 秒経過した時点では，被攻撃プロセスが 30 秒間アクセスしなかった物理メモリが非活性な物理メモリと判断されて，攻撃プロセスに割り当てられ有効に利用されている。測定開始後 120 秒経過した時点で，被攻撃プロセスが実行を終了し，攻撃プロセスが再び物理メモリを占有している。以上の結果から，我々の実装した優先度付き物理メモリ管理が機能していることが確認できる。

5.2 マクロベンチマーク

マイクロベンチマークの被攻撃プロセスはメモリアクセスしか行わないものを用いた。現実のプロセスではメモリアクセス以外に，ディスク I/O などの様々な処理を行う。そうした現実のプロセスに対しても優先度付き物理メモリ管理方式が有効であることを確認するために，Linux カーネルのコンパイル時間と，Netscape6 の起動時間の測定を行った。攻撃プロセスと比較対象は第 5.1 節の実験で用いたものと同じものを使用した。

5.2.1 Linux カーネルのコンパイル

Linux カーネルのコンパイルでは，コンフィグレーションを全てデフォルトとし，コンパイラには egcs 2.91.66 を使用した。

	攻撃なし	攻撃あり	比
Linux 2.2.16	168 秒	3923 秒	19.6 倍
プロセッサ優先度低下		3126 秒	18.6 倍
メモリ優先度低下	168 秒	290 秒	1.7 倍

表 1: Linux カーネルのコンパイル時間

表 1 に結果を示す。Linux 上では，物理メモリ占有攻撃ありの場合，攻撃なしの場合よりもコンパイル時間が 19.6 倍に増加する。攻撃プロセスのプロセッサ割り当て優先度を下げた場合，攻撃なしの場合の 18.6 倍の増加である。それに対し，優先度付き物理メモリ管理機構の下では，コンパイル時間の増加を攻撃なしの場合の 1.7 倍に抑えることに成功している。

5.2.2 Netscape6 の起動

	攻撃なし	攻撃あり	比
Linux 2.2.16	15.5 秒	132.4 秒	8.5 倍
プロセッサ優先度低下		105.5 秒	6.8 倍
メモリ優先度低下	15.5 秒	102.0 秒	6.6 倍

表 2: Netscape6 の起動時間

表 2 に結果を示す。物理メモリ占有攻撃の下では，Linux の場合では Netscape6 の起動時間は攻撃なしの場合の 8.5 倍に増加する。攻撃側プロセスの`nice` 値を 19 に変更して，プロセッサ割り当て優先度を低下させた場合は，攻撃なしの場合の 6.8 倍に増加する。優先度付き物理メモリ管理の場合でも，攻撃なしの場合の 6.6 倍に増加している。

5.2.3 考察

Linux カーネルのコンパイルは，プロセッサ，メモリ，ファイル I/O を総合的に行う処理であり，PC の性能評価に広く用いられている。一方，Netscape6 の起動はディスク I/O インテンシブの処理である。実際，起動には 15〜16 秒を要するが，その間のプロセッサ利用率は 20%未満である。

優先度付きメモリ管理方式では，優先度の低いプロセスはメモリ不足に陥ってスラッシングを起こし，ディスク帯域を占有してしまう。そのため，優先度の高いプロセスがディスク I/O インテンシブである場合，優先度付き物理メモリ管理方式だけでは，資源濫用の影響を十分に防ぐことができない。ディスク I/O も優先度管理すれば，ディスク I/O インテンシブなアプリケーションに対してもメモリ占有攻撃を防ぐことができると期待している。

6 関連研究

Liedtke らは文献 [1] において，物理メモリ占有攻撃に対処する一方式を提案している。その方式では，未使用のページフレームが尽きた場合，ページフレームを要求しているプロセスが，自身の所有しているページフレームをスワップアウトするか，ページフレームを譲ってくれるプロセスを探すという方式である。Liedtke らの手法では，サービス拒否攻撃から守るべきプロセスよりも先に攻撃プロセスが物理メモリを占有した場合，攻撃プロセスからメモリを奪い返す手段を提供していない。

Stealth Distributed Scheduler [2] は，プロセス移送を用いて計算機クラスタの負荷分散を行う方式である。計算機が高負荷状態になると，ローカルユーザのプロセスは，他の計算機から移送されてきたプロセスから物理メモリを奪うことができる。Stealth の設計は資源濫用攻撃を想定していないため，優先度の低いプロセスから優先度の高いプロセスに迅速にメモリ割り当てを変更できる仕組みを持たない。そのため，ひとたび物理メモリを占有されてしまうと，占有状態からの回復に長く時間がかかってしまう。

Scout [3]，SBOX [4]，Secure Java [5] は，信頼できないプログラムが利用できるメモリ量に上限を設け，物理メモリ占有攻撃に対処する方式を採用している。しかしすでに第 2.1 節で述べたように，信頼できないプログラムを実行する前に適切な上限値を定めることは難しい。

Lottery スケジューリング [6, 7] は，proportional-share を実現するスケジューリング方式のひとつであり，計算機資源の配分量を割合で指定することができる。各プロセスは *lottery* と呼ぶトークンを持ち，これをプロセス間で取引することによって，資源配分量を動的に変更できる。しかし，物理メモリ占有攻撃に対処できる lottery の取引戦略は知られていない。

7 まとめ

本論文では，信頼性の低いプログラムによる物理メモリ占有攻撃を防ぐ方法として，優先度付き物理メモリ管理を提案した。この方式では，占有攻撃の影響を優先度の低いプロセスにサンドボックスすることができる。提案方式では，メモリ優先度の高いプロセスはメモリ優先度の低いプロセスから物理メモリを奪取することができ，物理メモリ占有攻撃の影響は優先度の高いプロセスには影響しない。実際，物理メモリ占有攻撃を行った場合でも，Linux カーネルのコンパイル時間の増加を 1.7 倍程度に抑制することができた。今後，物理メモリに限らず計算機資源全般に優先度を導入し，任意の資源濫用攻撃に対処できるようにすることを考えている。

参考文献

[1] J. Liedtke, N. Islam, and T. Jaeger, “Preventing denial-of-service attacks on a μ-kernel for WebOSes,” in *Proceeding of the 6th IEEE Workshop on Hot Topics in Operating Systems (HotOS)*, (Chatham (Cape Code), MA), May 5–6 1997.

[2] P. Krueger and R. Chawla, “The Stealth distributed scheduler,” in *Proceedings of the 11th IEEE International Conference on Distributed Computing Systems*, pp. 336–343, 1991.

[3] O. Spatscheck and L. L. Peterson, “Defending against denial of service attacks in Scout,” in *Proceedings of the Third USENIX Symposimum on Operating Systems Design and Implementation*, pp. 59–72, February 1999.

[4] L. D. Stein, “SBOX: Put CGI scripts in a box,” in *Proceedings of the 1999 USENIX Annual Technical Conference*, pp. 145 – 155, June 1999.

[5] L. van Doorn, “A secure Java virtual machine,” in *Proceedings of the 9th USENIX Security Symposium*, August 2000.

[6] C. A. Waldspurger and W. E. Weihl, “Lottery scheduling: Flexible proportional-share resource management,” in *Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation*, November 1994.

[7] D. G. Sullivan and M. I. Seltzer, “Isolation with flexibility: A resource management framework for central servers,” in *Proceedings of 2000 USENIX Annual Technical Conference*, June 2000.