動的なアクセス権限変更のための アクセス制限の安全な解除機構

光来健一* 千葉滋**

*東京大学

**筑波大学・さきがけ研究21

プロセスのアクセス制限

- 信頼できないプログラムの実行に有用
  - ウェブサーバのCGIプログラムなど
- アクセス制限の解除は難しい
  - 安易な解除機構はセキュリティ上危険
    - setuidプログラム

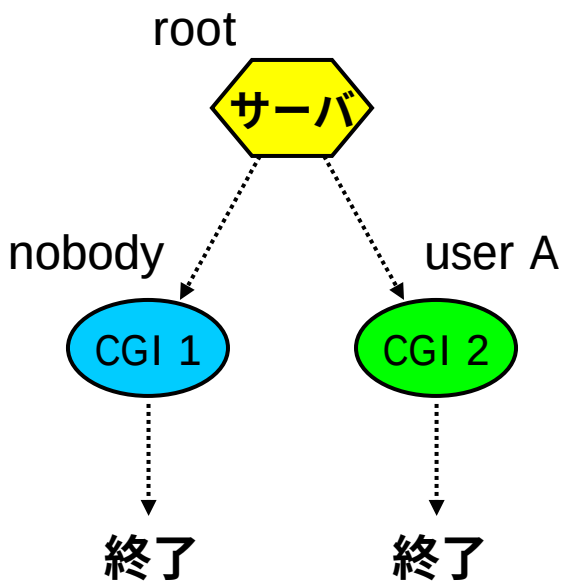
安全にアクセス制限を解除する機構を提案

Compactoによるアクセス制限

- Compacto：きめ細かいアクセス制限を可能にするOS
  - システムコールの利用制限
    - システムコールの引数等によるきめ細かい制限
  - ユーザ・グループの制限
    - setuid/setgidシステムコールによる権限の制限
  - ディレクトリの制限
    - chrootシステムコールによるルートディレクトリの変更

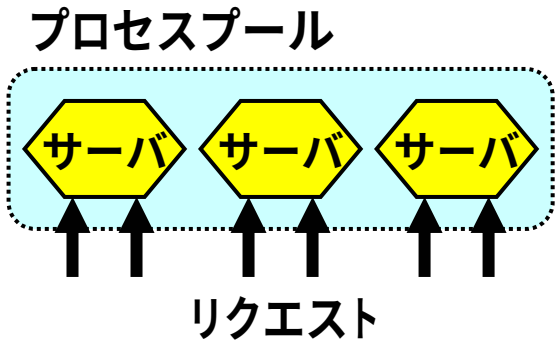
子プロセスによる実行

- 信頼できないプログラムは子プロセスで実行する
  - 子プロセスにアクセス制限をかける
- アクセス制限の解除は必要ない
  - 処理が終われば子プロセスは終了する



プロセスプール

- 実行性能改善のテクニック
  - あらかじめいくつかのプロセスを作っておく
  - それらのプロセスを再利用する
- アクセス制限の解除が必要
  - 一つのプロセスが様々なアクセス制限の下で動く必要がある



サーバのプログラム例

```
while () {
  s = accept();
    <アクセス制限の追加>
    <リクエストの処理>
  close(s);

    <アクセス制限の解除>
}
```

アクセス制限を解除する危険性

- 信頼できないプログラムに勝手に解除されるかもしれない
  - 悪意をもったプログラム
  - バッファオーバフロー攻撃を受けたサーバ
- 解除した後に影響が出るかもしれない
  - 書き換えられた環境変数の参照
  - 変更されたシグナルハンドラの呼び出し

プロセス･クリーニング

- アクセス制限を安全に解除する機構
  - アクセス制限を追加する前にプロセスの状態を保存しておく
    - レジスタやメモリイメージ等
  - アクセス制限を解除する時には状態を復元する
    - プログラムの実行による影響を取り除く

サーバのプログラム例

```
/* プロセスの状態を保存(1) */
save_safe_states();

s = accept();
  <アクセス制限の追加>
  <リクエストの処理>
close(s);

/* プロセスを(1)の状態に戻す */
/* アクセス制限を解除し(1)へ */
restore_safe_states();
```

復元される資源

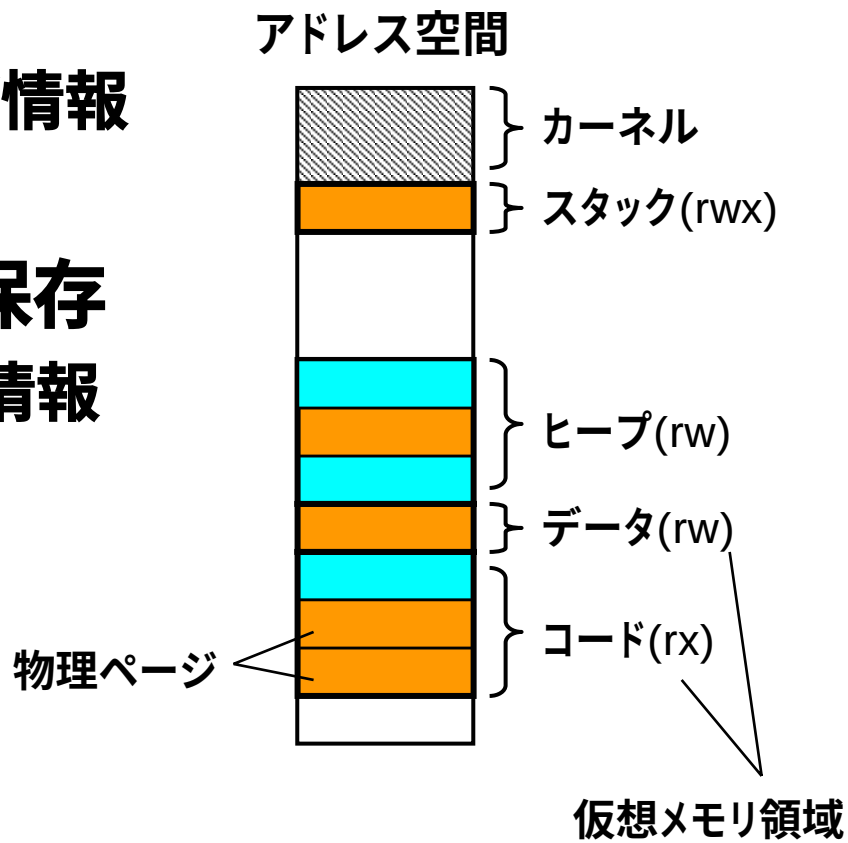
- レジスタ (プログラムカウンタなど)
  - プログラムの制御を取り戻せる
- メモリ (スタック、環境変数など)
  - スタック等に置かれた不正コードを除去できる
- シグナル
  - シグナルによるトロイの木馬攻撃を防げる
- ファイル・ソケット
  - ファイル・ソケットに関するDoS攻撃を防げる

レジスタの保存･復元

- 保存
  - 全てのレジスタの内容を保存(setjmp)
- 復元
  - 保存しておいたレジスタの内容を元に戻す(longjmp)
    - スタックポインタが指すスタックフレームはメモリの復元により正常なものになる
      - 復元時には破壊されてしまっているかもしれない

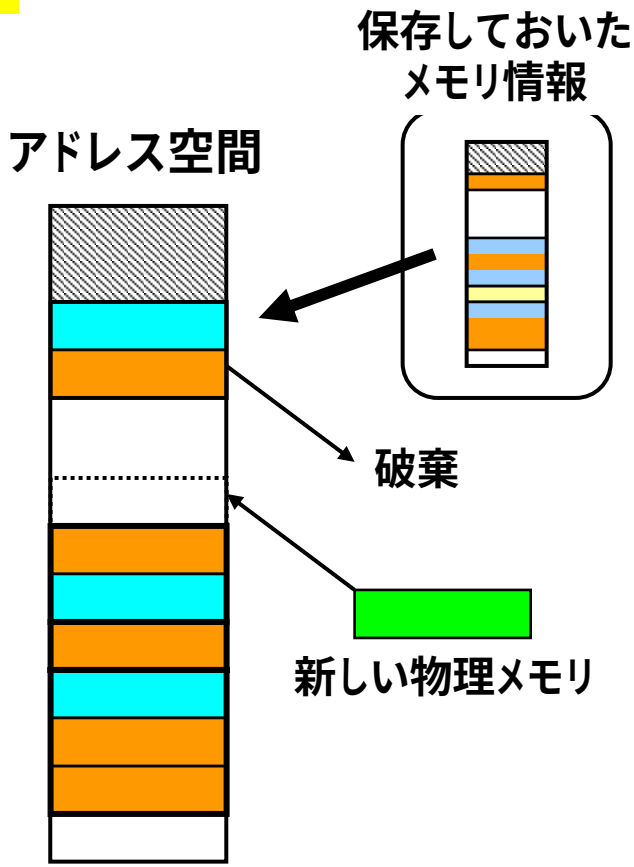
メモリの保存

- メモリイメージを保存
  - 物理ページのマッピング情報
  - 物理ページの内容
- 仮想メモリ管理情報を保存
  - 仮想メモリのマッピング情報
  - メモリ保護情報
  - データセグメント・サイズ



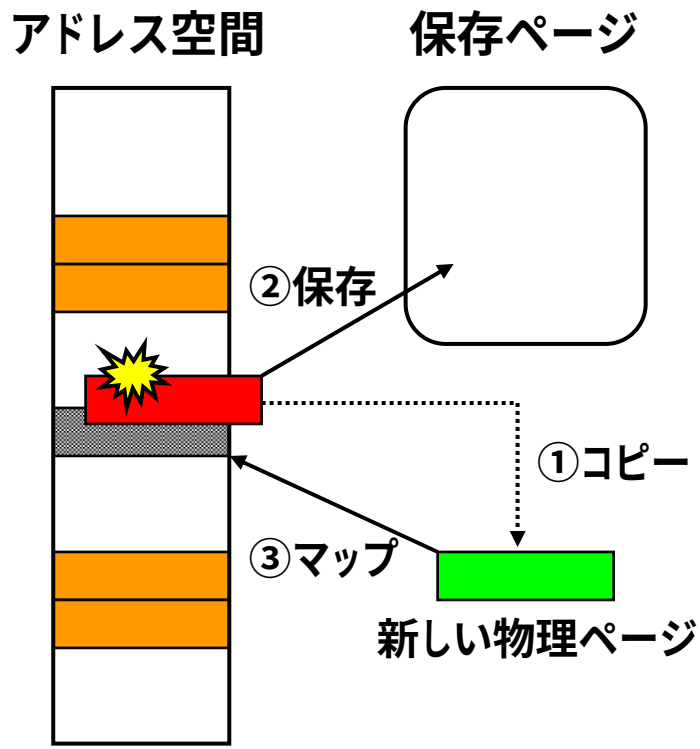
メモリの復元

- メモリイメージを元に戻す
  - 保存前からあるページは元の内容をコピー
    - なくなっていれば新しい物理ページを割り当てる
  - 保存前になかったページは破棄
- 仮想メモリ管理情報を元に戻す
  - 保存後にmmap/munmap/mprotectされた領域



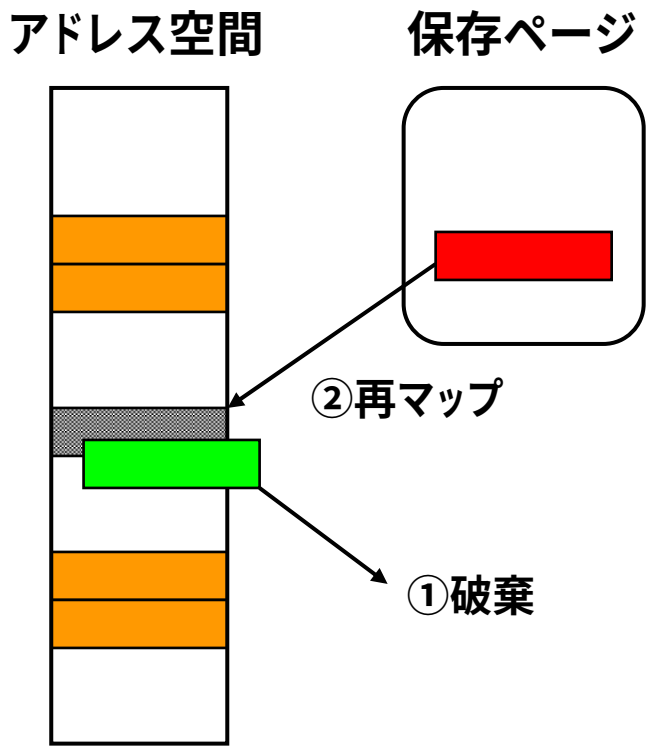
*copy-on-write*を使った最適化(1)

- 内容が変更されたページだけをコピーする
  - 保存時に書き込み可能なページを書き込み禁止にしておく
    - 書き込みが起こるとページフォールトが発生する
  - フォールト時に元の内容を新しいページにコピーする
    - 元のページは保存しておく
    - 新しいページを書き込み可能にしてマップする



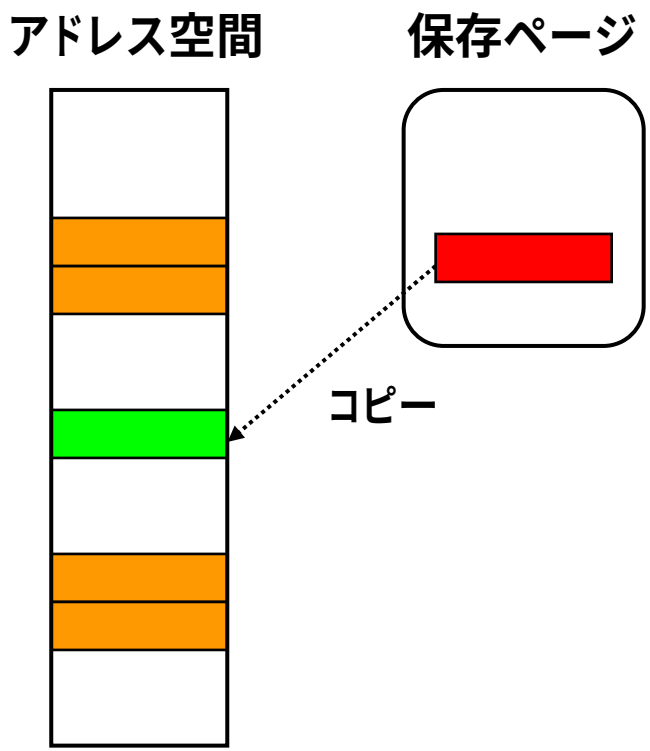
*copy-on-write*を使った最適化(2)

- 復元時には内容が変更されたページだけを元に戻す
  - 内容が変更されたページを破棄する
  - 保存しておいたページをマップする
    - そのページは書き込み禁止にしておく



予測ページコピーによる最適化

- 再び書き込まれそうなページを投機的にコピーする
  - 予測は比較的容易
    - リクエスト処理は毎回アクセスパターンが似ている
  - 復元時に元の内容をコピーする
    - ページフォールトを減らせる
    - 予測が外れると無駄になる



シグナルの保存･復元

- 保存
  - シグナルハンドラ等を保存
- 復元
  - 未処理のシグナルを処理する
    - ブロックされていなければシグナルハンドラを実行
    - ブロックされている場合
      - 元の状態でブロックされている場合はそのまま
      - ブロックされていない場合は破棄
  - 変更されたシグナルハンドラ等を元に戻す

ファイル･ソケットの保存･復元

- 保存
  - オープンされているファイル･ソケットを記録
    - 参照回数を1増加させておく
- 復元
  - 保存後にオープンされたファイル･ソケットはクローズする
  - 保存後にクローズされたファイル･ソケットは再びオープン状態にする

実験：マイクロベンチマーク

- プロセス・クリーニングのオーバヘッドを測定した
  - 書き換えられるメモリはスタック1ページ
  - シグナル、ファイル・ソケットの状態は変化なし
- 比較のため、子プロセスを作って処理させる場合についても測定した
- 実験環境
  - Pentium II 400MHz、メモリ128MB
  - プロセス・クリーニングを実装したCompacto（Linux 2.2.16ベース）

結果：マイクロベンチマーク

- メモリを元に戻すのに大きなオーバヘッドがかかる
  - 予測ページコピーにより大幅に性能改善できる
    - 予測は100%成功
- 子プロセスを作る場合のオーバヘッドは100μs

プロセス・クリーニングのオーバヘッドの内訳

内容	時間（μs）
システムコール	0.79
レジスタ	0.25
メモリ	3.36 (6.20)
シグナル	0.13
ファイル・ソケット	0.09
その他	0.10
計	4.62 (7.52)

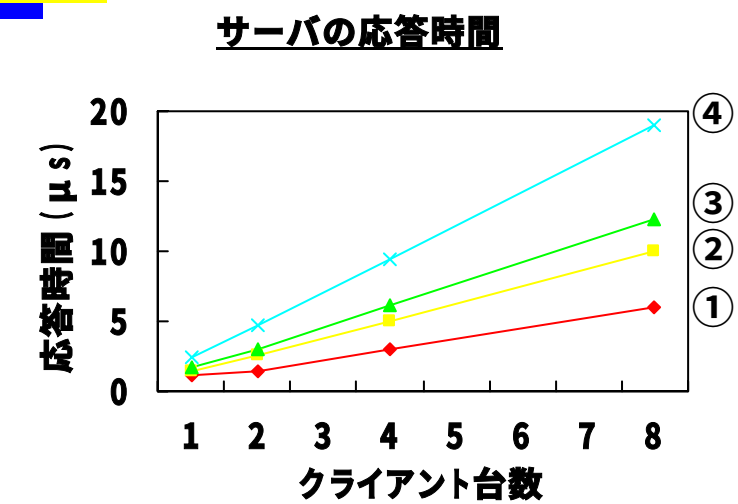
（カッコ内の数値は予測ページコピーを行わなかった場合）

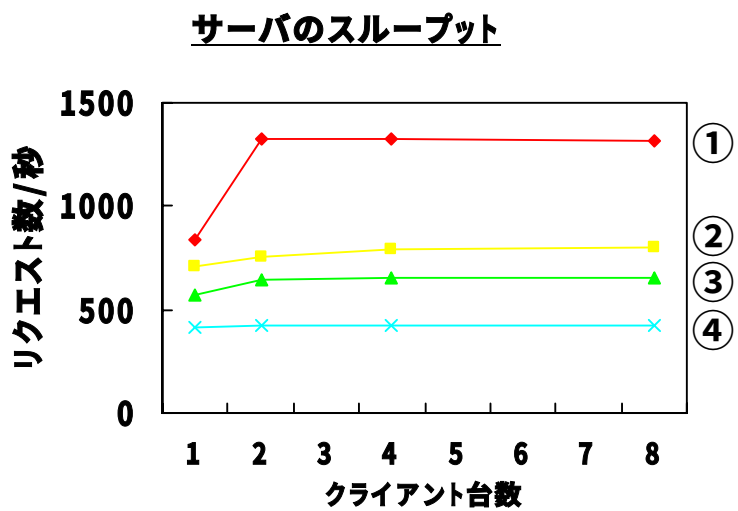
実験：Apacheウェブサーバ

- WebStoneを用いてApacheの性能を測定した
  - クライアントは0 byteのHTMLファイルをリクエスト
  - 実験したサーバ
    - プロセス・クリーニングを行うサーバ
      - 予測ページコピーあり・なし
    - Apache標準のサーバ
    - リクエスト毎に子プロセスを作るサーバ
- 実験環境
  - PentiumII 400MHz、メモリ128MB、Compacto(サーバ)
  - Celeron 300MHz、メモリ64MB、FreeBSD(クライアント)
  - 100Mbpsのスイッチングハブで接続

結果：Apacheウェブサーバ

- プロセス・クリーニング版 (②)
  - Apache標準 (①) と比べて
    - 応答時間 1.2〜1.7倍
    - スループット 16〜40%低下
  - 子プロセス版 (④) と比べて
    - 応答時間 0.5倍
    - スループット 90%向上
  - 予測ページコピーなし (③) と比べて
    - 応答時間 0.8倍
    - スループット 24%向上





関連研究

- チェックポインティング
  - プロセス・クリーニングはロールバックにあたる
- OSによる細粒度保護ドメイン[高橋ら'99]
  Palladium[Chiueh et al.'99]
  - プロセス内でモジュールを安全に実行できる
  - モジュール内の悪影響を取り除く必要がある
- StackGuard[Cowan et al.'98]
  - バッファオーバフローを検出する

まとめ

- プロセスのアクセス制限を安全に解除する機構を提案した
  - アクセス制限を解除する時にはプロセスを元の状態に戻す
- プロセス・クリーニングのオーバヘッドは実用に耐える程度であることを示した
  - プロセス・クリーニングを行ってもプロセスプールする方がサーバの性能が良くなる

今後の課題

- 予測ページコピーの予測の精度を上げる
  - ページのダーティビットを利用
- バッファリングを行っているライブラリ関数に対応する
  - 状態を復元する前にバッファをフラッシュする
- 状態を復元してもグローバルな情報を残せるようにする
  - 安全性を考慮する必要がある