A Framework for Easily and Efficiently Extending Operating Systems
OSの機能拡張を容易にかつ効率よく行なうための枠組

益田研究室　修士２年
光来健一

- ＯＳの拡張に対する要求が高まっている
  - 機能拡張
    - 新しいネットワークプロトコルを追加する
      例：ＩＰｖ６
  - 性能向上
    - データベース専用のファイルシステムを使う
      例：ファイルキャッシュのポリシにＬＲＵを使わない
- 拡張可能ＯＳが研究されている
  - ＯＳを拡張するためのプログラム（拡張モジュール）を動的に追加できる

fail‐safe機構

- 拡張モジュールのエラーからＯＳを守る必要がある
  - 拡張モジュールは意図しない動作をするかもしれない
- fail-safe機構の役割
  - 拡張モジュールのエラーを検出する
  - エラーの検出された拡張モジュールをＯＳから安全に切り離す

従来のアプローチの問題点

- fail-safe機構は性能を犠牲にするので、両者のトレードオフを取るのは難しい
  - 拡張モジュールをloadable kernel moduleとして作る
  - 拡張モジュールをユーザプロセスとして作る (Mach[Accetta et al.86])
  - 拡張モジュールにload/store/jumpのアドレスを検査するコードを挿入する (VINO[Selzer et al.96])

fail-safe機構に関する考察

- 常に完全なfail-safe機構が必要なわけではない
  - 保護レベルは段階的に変えられた方がよい

例1．拡張モジュールの使用

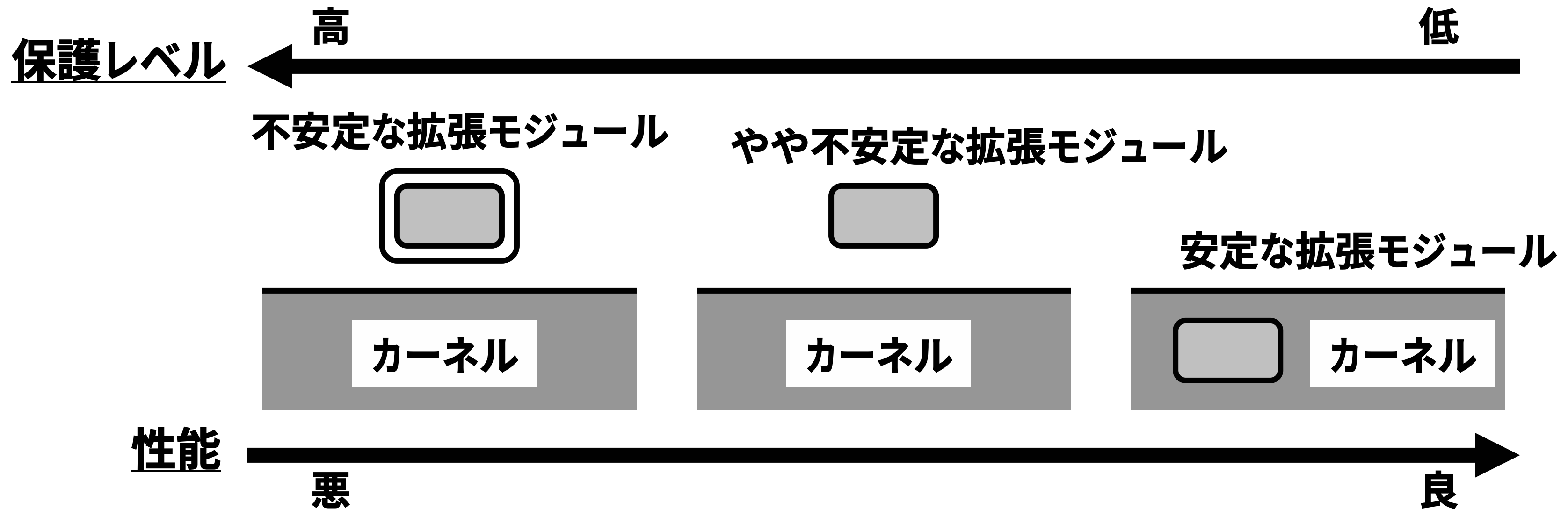
- 不安定な場合は必要な保護をして使う
- 安定している場合は保護をせずに使う

例2．拡張モジュールの開発

1. デバッグ時には十分に保護する
2. βテスト時には保護を弱めて性能を良くする
3. リリース時には保護なしで動かせるようにする

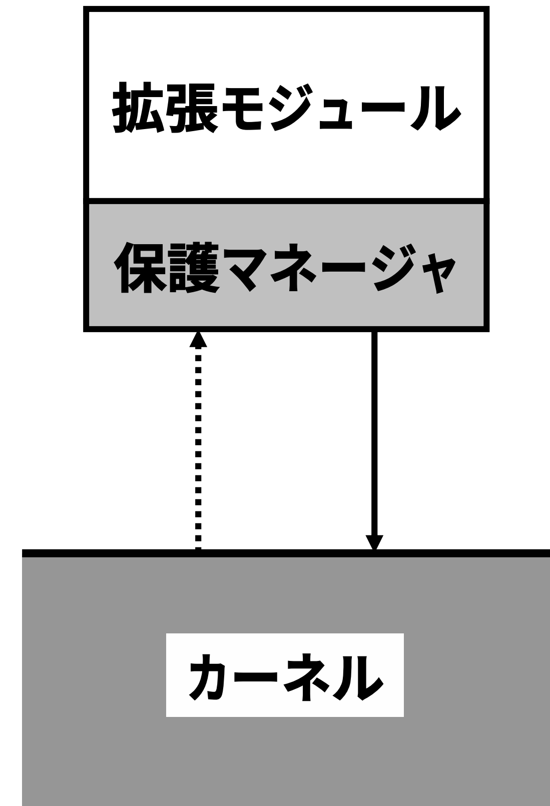
多段階保護機構の提案

- 拡張モジュールを様々な保護レベルでＯＳに組み込むことができる
  - 安定度に応じて性能とfail-safeの能力のトレードオフを取ることができる



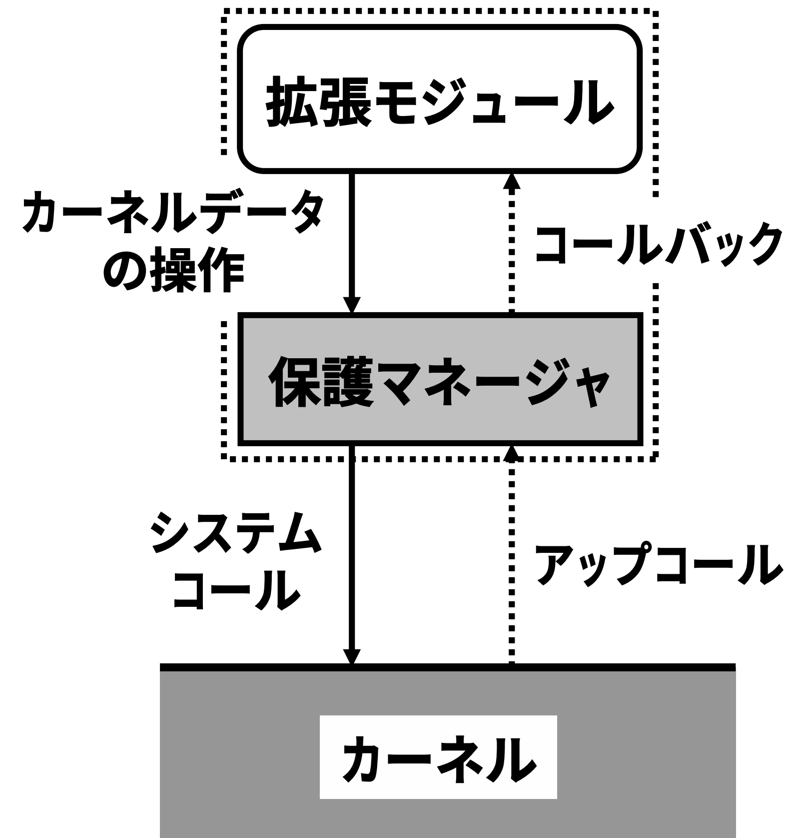
システム構成

- 多段階保護機構を実現するシステム
  - 保護マネージャ
    - fail-safe機構を実現する
    - 拡張モジュールとリンクされるライブラリ
    - 複数用意されている
    - それぞれが様々な保護レベルを提供する
  - 適切な保護マネージャを選ぶことができる



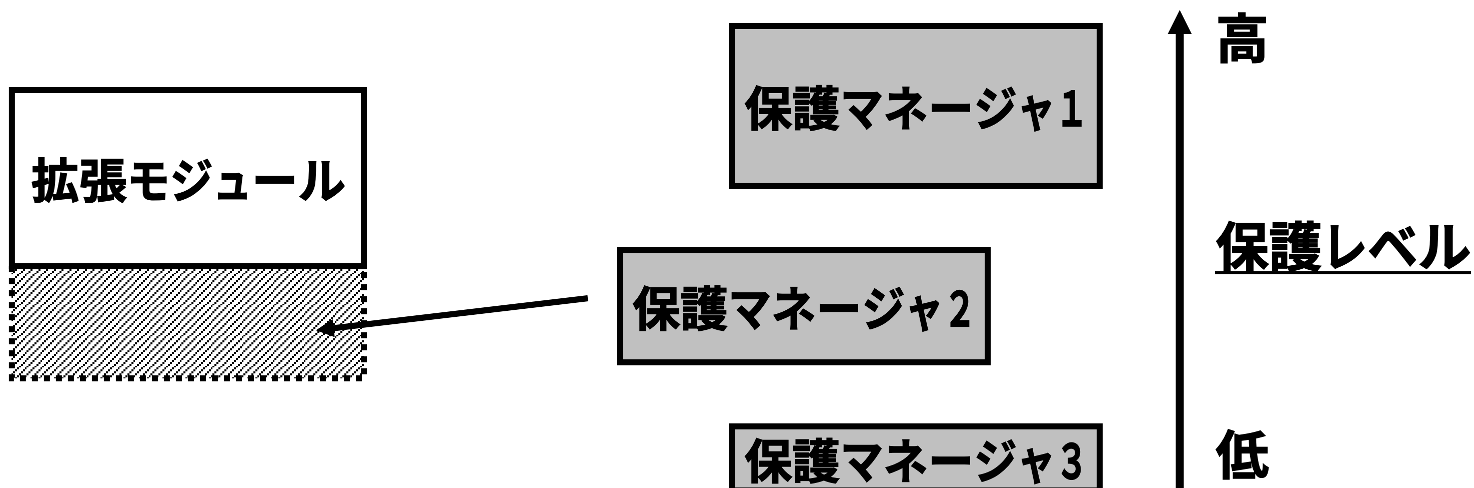
保護マネージャ

- 拡張モジュールとカーネルの間のゲートウェイの役割を果たす
  - カーネルからのアップコールを中継する
  - 拡張モジュールにカーネルデータを安全に操作させる



保護レベルの変更

- 保護レベルの変更は保護マネージャの交換によって行なう
  – 拡張モジュールのソースコードを変更したり、再コンパイルしたりする必要はない



共通のＡＰＩ

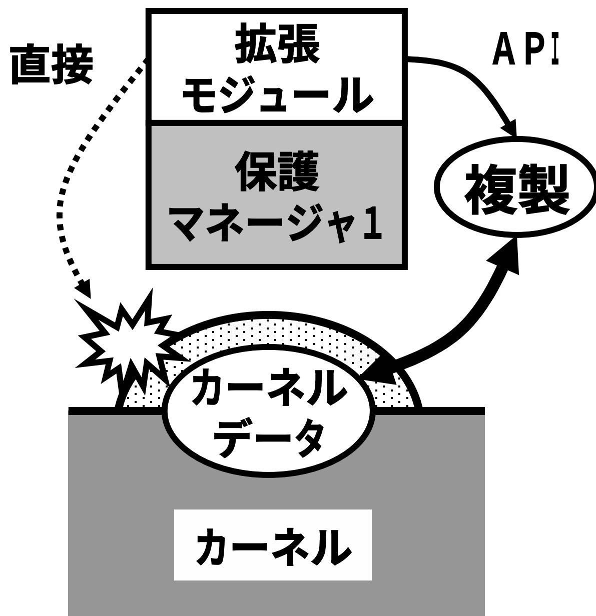
- すべての保護マネージャが共通のＡＰＩを提供することで交換を容易にする
  - それぞれの保護マネージャの実装の違いを吸収する
- ＡＰＩの種類
  - コールバックのためのＡＰＩ
    - カーネルからのアップコールによって呼び出される
  - カーネルデータをアクセスするためのＡＰＩ
    - 抽象度の高いデータ構造を見せる

様々な保護レベルの実現

- 保護マネージャは以下の保護技術を組み合わせて様々な保護レベルを提供する
  - 不正メモリアクセスの検出
    - 拡張モジュールをユーザ空間に配置するかどうか
    - カーネルデータのあるメモリを保護するかどうか
    - カーネルデータを複製・検査するかどうか
  - デッドロックの検出
    - どのくらいの頻度でデッドロックの検査をするか

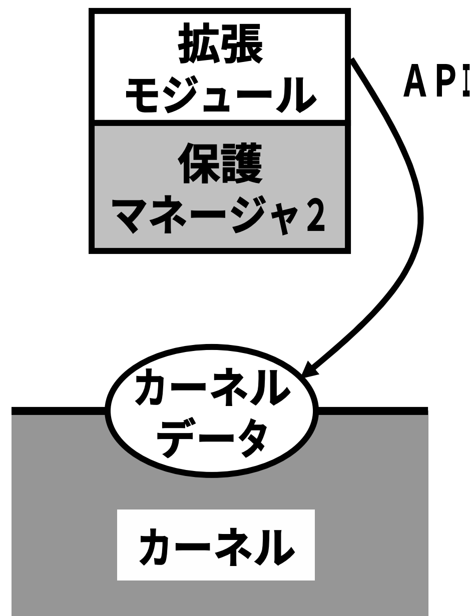
保護技術の組合せの例

(a) 最大の保護レベル



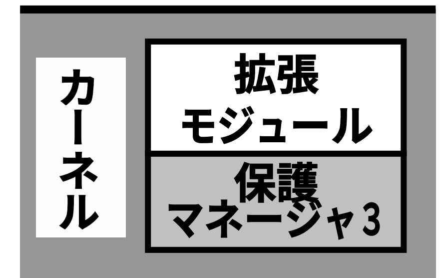
- •ユーザ空間
- •メモリ保護
- •カーネルデータ複製

(b) 中間の保護レベル



- •ユーザ空間

(c) 最小の保護レベル



- •カーネル空間

カーネルによるエラー回復

- 拡張モジュールのエラーを検出したら、カーネルからエラーの影響を取り除く
  - 不正メモリアクセスを検出した場合
    - 以後そのモジュールを参照しないようにする
    - ログを元に安定な状態に戻す
      - カーネルの状態を不安定にする操作についてはログをとっておく
  - デッドロックを検出した場合
    - wait-for-graphのループを破壊する

- 多段階保護機構を実現するシステムCAPELAをNetBSDをベースにして実装した
- 実験の目的
  - 保護レベルに応じて性能が良くなることの確認
  - 最大の保護レベルのオーバヘッドの測定
  - ＡＰＩを共通にすることによるオーバヘッドの測定
- 実験環境
  - ＰＣ (PentiumII 400MHz) ２台
  - 10Mbpsイーサネット

実験対象

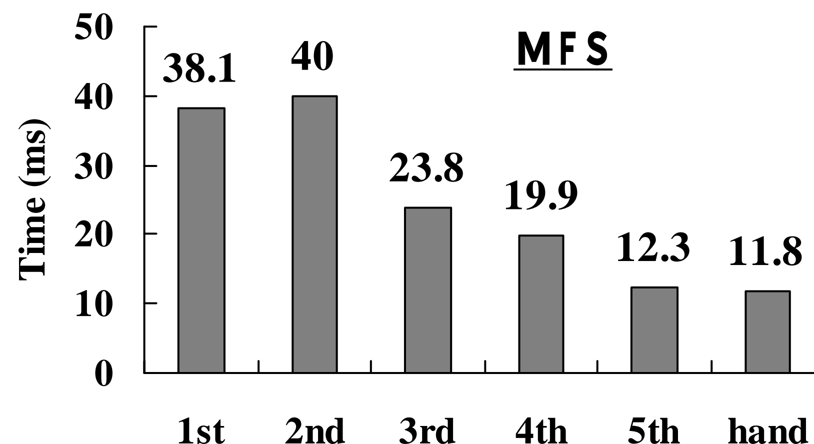
- 対象
  - ファイルシステム：MFS（Memory File System）、NFS
    - 64KBのファイルのコピーに要する時間を測定
  - ネットワーク・サブシステム：UDP、TCP
    - 往復のレイテンシとスループットを測定
- 表の5つの保護レベルとカーネルに直接作り込んだものについて実験した

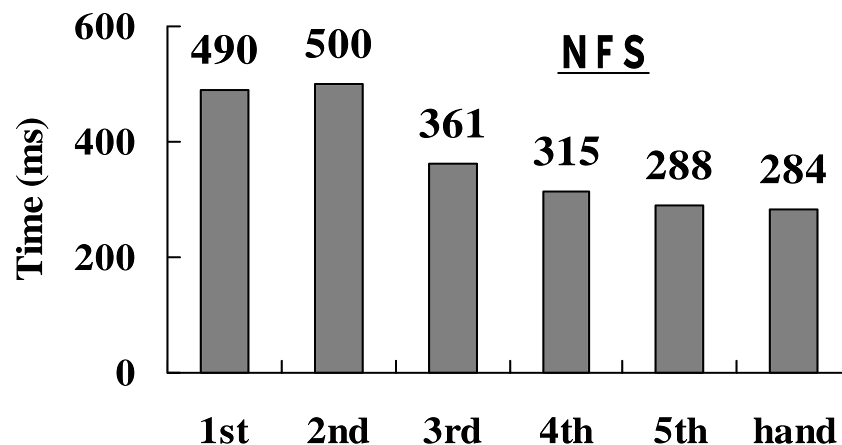
	1st	2nd	3rd	4th	5th
メモリ保護	○	△			
カーネルデータの複製	○	○	○		
アドレス空間の切替え	○	○	○	○	

△：読み出し専用にして保護

結果：ファイルシステム（コピーに要する時間）

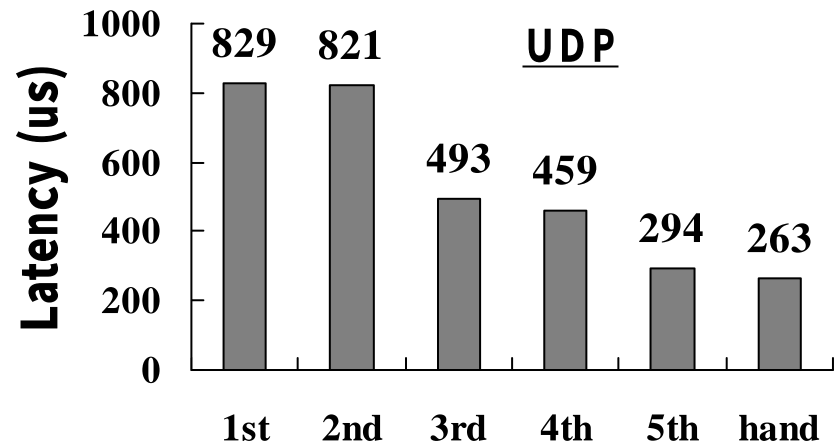
- 最大の保護レベルのオーバヘッド
  - ＭＦＳ：210%
  - ＮＦＳ：70%
- ＡＰＩを共通にすることによるオーバヘッド
  - ＭＦＳ：3.7%
  - ＮＦＳ：1.4%

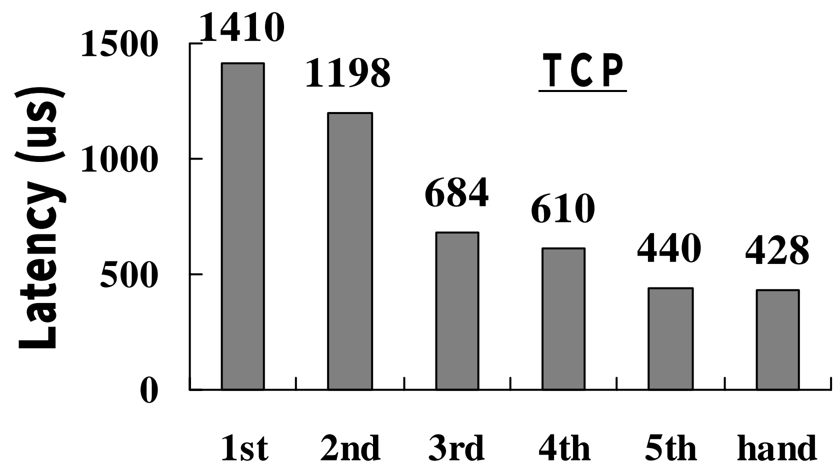




結果: ネットワーク・サブシステム（往復のレイテンシ）

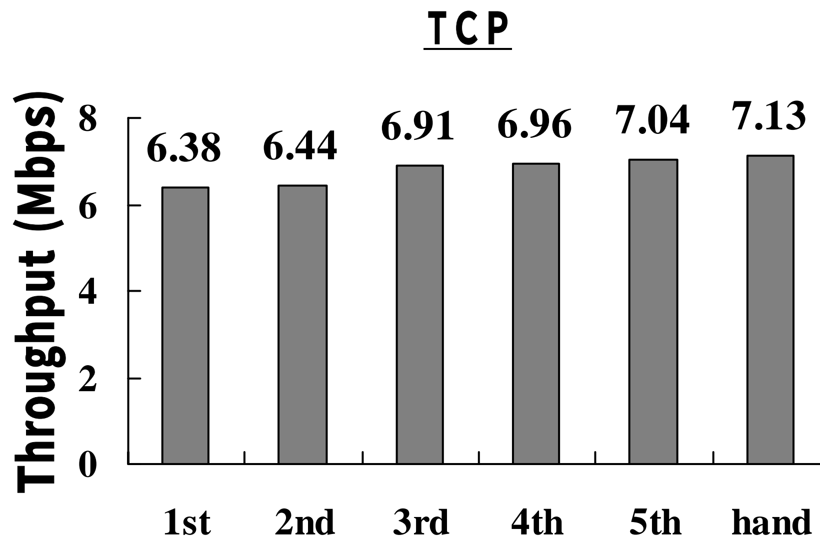
- 最大の保護レベルのオーバヘッド
  - UDP: 180%
  - TCP: 220%
- APIを共通にすることによるオーバヘッド
  - UDP: 12%
  - TCP: 2.8%





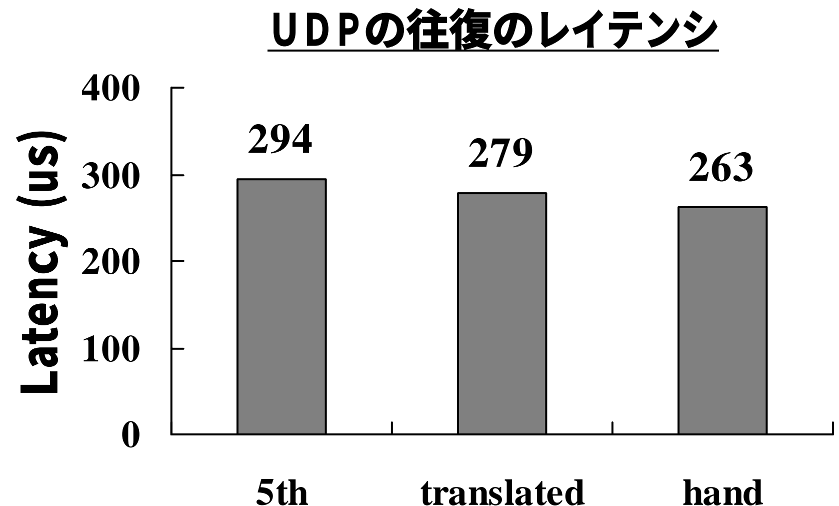
結果: ネットワーク・サブシステム (スループット)

- 最大の保護レベルのオーバヘッド
  - TCP: 10%
- APIを共通にすることによるオーバヘッド
  - TCP: 1.3%



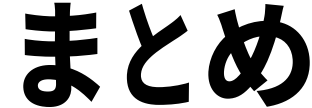
ソースコード変換による性能向上

- OpenC++[Chiba95]を使いソースコード変換をした
  - APIを共通にするためのオーバヘッドを減らす
    - カーネルが使うデータ構造と拡張モジュールが使うデータ構造の変換など
- UDPでは変換した後のオーバヘッドは6.1%に抑えられる
  - 変換前の約半分



関連研究

- Chorus[Rozier et al.88]
  - ユーザレベルサーバをカーネルに入れることができる
  - カーネルに入れた時の性能が不十分
    - ファイルの読み出しで80%のオーバヘッド
- SPIN[Bershad et al.95]
  - 型安全な言語Modula-3を使って拡張モジュールを記述する



まとめ

- 多段階保護機構を提案した
  – 拡張モジュールを変更なしに様々な保護レベルでＯＳに組み込める
- CAPELAを実装し、実験によって多段階保護機構の有用性を確かめた
  – 不安定な時は許容範囲内のオーバヘッドで十分な保護ができる
  – 安定している時はカーネルに作り込んだものに十分近い性能が得られる

今後の課題

- カーネルレベルでも様々な保護レベルを提供する
  – タイミング依存のエラーの検出など
- いつ保護レベルを変えるべきかの指針を示す
  – モジュールの安定度に応じて自動的に保護レベルを変える
- 拡張モジュール間のやりとりを効率よくできるようにする
  – ＯＳの拡張モジュールという特徴を活かす