Capability-Based Microkernelによる セキュアな ユーザーレベルメモリ管理システム

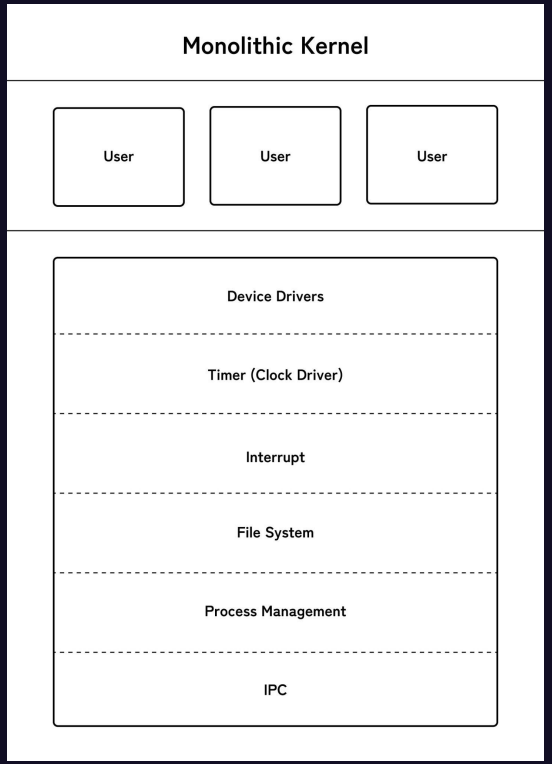
伊組烈火

Index

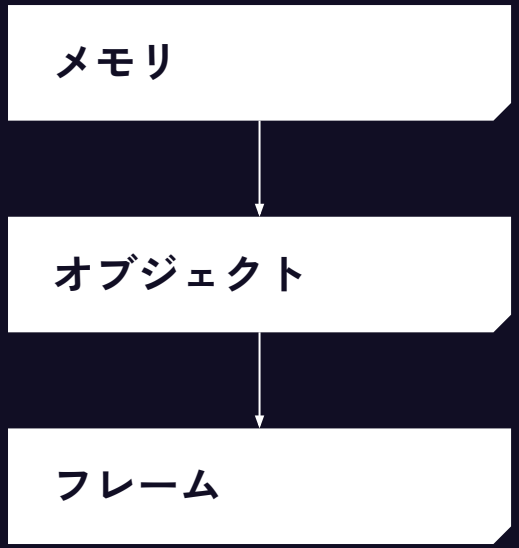
Index

典型例：Linuxカーネルの構造

Linuxカーネルは**モノリシック**なアプローチを取っている
基本的なシステムの機能を**カーネル自体が持つ**



典型例：Linuxカーネルのメモリ管理システム



kmalloc

カーネル内であるサイズを割り当てる
あらかじめスラブ・キャッシュを持つ

SLAB/SLUB/SLOB Allocator

同一サイズのカーネルオブジェクトを割り当てる
内部フラグメンテーションに強い

Buddy System

物理メモリフレームを割り当てる
外部フラグメンテーションに強い

全てカーネルが提供する

モノリシックなカーネルの課題点

カーネルの提供する方針にシステムが固定されてしまう

- ユースケースに最適化できない
- 新しいアルゴリズムを試すことができない
  - カーネルの再ビルドが必要になる

カーネルが不安定なものに依存してしまう

- 不安定なドライバは問題を引き起こす
  - カーネルごとクラッシュする

うれしくない……

Not 柔軟

マイクロカーネルによるモジュラー化

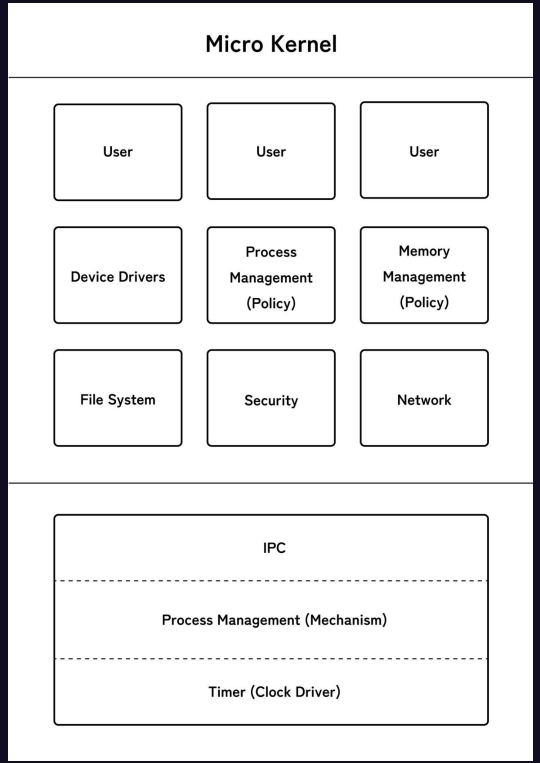
カーネルを小さくし, 機能をユーザーへ切り分ける

方針をユーザーが決定する

- ユースケースに**最適化できる**
- カーネルの変更なしに方針を変更できる

カーネルが安定する

- 不安定なドライバはユーザー空間上で動作する
  - 境界が適切に設定される



マイクロカーネルのメモリ管理システム

しかし, 殆どのマイクロカーネルはカーネル内でメモリ管理方針を決定する

課題1 : カーネル内のメモリ確保制限

- カーネル自身が使用するデータの動的な割り当てができない
- 静的なヒープを持った場合, 拡張ができない

課題2 : カーネル情報の流出

上記の問題に対処するため, 管理をユーザーへ委譲することになるが.....

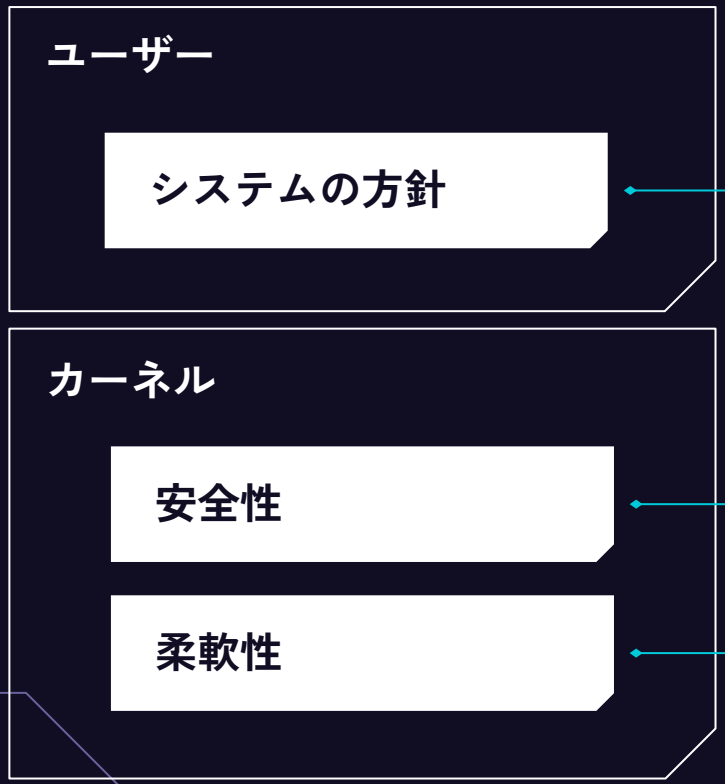
- ユーザーレベルのメモリ管理サーバーが権限を持ちすぎてしまう
  - ユーザーがカーネルの割り当て情報を知ることが可能となる
  - 不正なメモリアクセスを引き起こす

うれしくない.....

Not Secure

Index

柔軟かつセキュアなシステムの実現



ユーザーレベルサーバー

カーネルが担っていた処理をユーザー側で行う
OSとアプリケーションが水平になる

Object-Capability Model

セキュアに特権操作を行うための仕組み
操作の対象を制限する

マイクロカーネル

処理をユーザーに委譲
最小限の機能を持つ

Object-Capability Model

権限ベースのセキュリティ

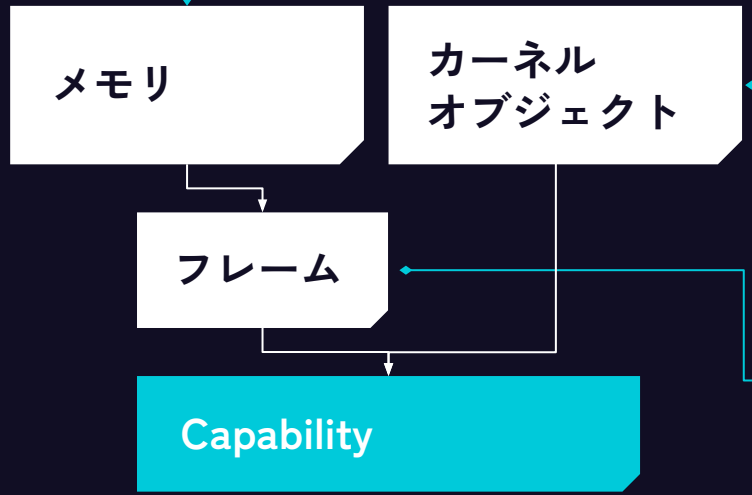
オブジェクトに対する操作権限

- メモリ操作のような特権的操作は**Capability**を通してのみ実行可能である
- Capabilityは**Descriptor**により**間接的**に指定される
- Capabilityは**譲渡**することができる

セキュアなユーザーレベル操作

- Capabilityは偽造することができない
- 自分の持たないCapabilityに対する操作は不可能である
    - 各コンポーネントは**最小の特権**のみを有する

セキュアなユーザーレベルメモリ管理システム



libc malloc

あるサイズを割り当てる

SLAB/SLUB/SLOB Allocator

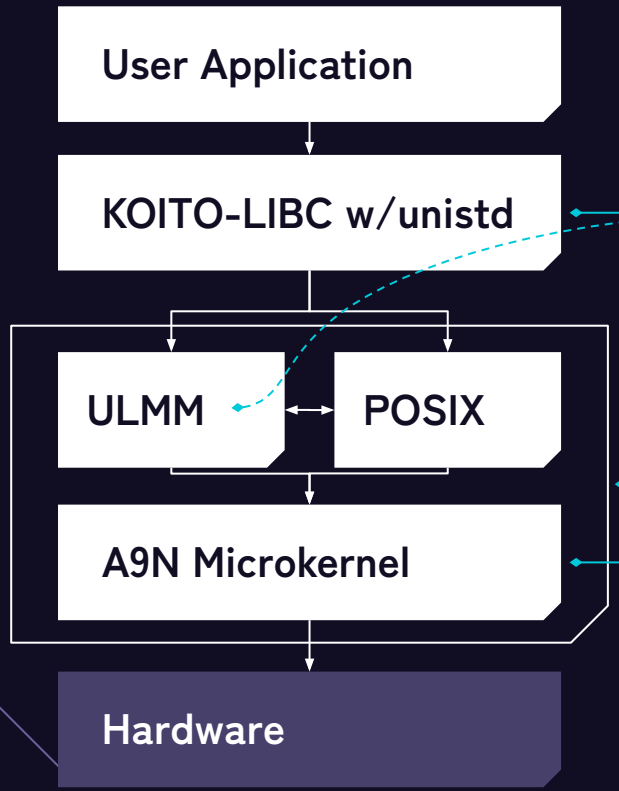
カーネルオブジェクトを割り当てる
Generic Capabilityから**変換**して作成する

Buddy System

物理メモリフレームを割り当てる
Frame Capabilityを介してメモリを操作する

ユーザーレベルで実現！

提案システムの全体図



KOITO libc

User Applicationが使用するための標準ライブラリ
libc内のシステムコール関数は, POSIXサーバーへのIPCによって実行される

User-Level Memory Management Server

ユーザーレベルでBuddy System, SLAB Allocatorを実装する

KOITO

ULMMによりユーザーレベルでメモリ管理を行うOS
POSIX互換性を備える (おおむね準拠)

A9N Microkernel

HALを備えたCapability-Based Microkernel
アーキテクチャ非依存のAPI

現代の仮想化技術に対する親和性

コンテナ

POSIXサーバーを新規に実行し，システムコールとしてのIPC対象を切り替えるだけで隔離可能

Hypervisor

Hypervisorによって実行されるシステムに対し，**個別に最適化**されたオーケストレーションが可能
VMMはNOVAのようにユーザーレベルで動作

WASM

Hypervisorと同様に，ユースケースに合わせて**個別に最適化**が可能

セキュリティと柔軟な最適化の両立を実現

Index

先行事例：seL4

Capability-Based Microkernel

✅ 形式検証
✅ 9kLoC：十分に小さなコードベース
✅ 高速
☒ アーキテクチャ依存のAPI

Memory Management

✅ カーネル内ヒープを持たない
✅ Capabilityを介して操作
☒ アーキテクチャ依存のページテーブル Capability

Userland

☒ 未成熟なユーザーランド
☒ Capabilityを直接操作する低レベルなライブラリ

先行事例 : Zircon

Capability-Based Microkernel

☑️ Rightsによるセキュリティ
☒ 50kLoC : マイクロカーネルとしては大きい

Memory Management

☑️ Capabilityを介して操作
☒ カーネル内メモリ管理 (ページテーブル)

Userland

☑️ Googleによる成熟したユーザーランド (e.g., Fuchsia)

本提案：KOITO on A9N Microkernel

Capability-Based Microkernel

✅ Capabilityによるセキュリティ
✅ 10kLoC
✅ **HALによる移植容易性**
✅ **ハードウェア非依存のAPI**

Memory Management

✅ カーネル内ヒープを持たない
✅ Capabilityを介して操作
✅ **ULMMによる高レベルメモリ管理API**

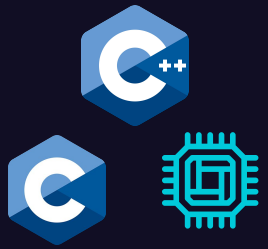
Userland

✅ **POSIX互換による既存資産の活用**
✅ **低レベルのCapability操作はライブラリに隠蔽**

組み込み/デスクトップ/サーバーに囚われないシステム

Index

どう打ち出すか?



実装言語/アーキテクチャ

- ASM, C, C++
- 期間中はx86_64に絞る

OSS

- GitHub上で公開
- MITライセンス

マニュアル

- 組版エンジンとしてTypstを用いる
- Typstコードを公開する

- セキュアを謳うクローズドなシステムは透明性が欠けている
- オープンであっても使用方法が理解できなければならない

課題を解決し, 長期的なコミュニティを形成

Index

現在の実装進捗度

85%

A9N Microkernel

- ☑ UEFI ブートローダー
- ☑ マイクロカーネルの基礎
- ☑ x86_64 ハードウェア依存部分
- ☐ Capability
  - ☑ Capability Node
  - ☑ Generic Capability
  - ☐ IPC Capability (実装中)
  - ☐ Frame Capability (実装中)
  - ☐ PageTable Capability (実装中)
  - ☐ Process Control Capability (実装中)

☑ほぼ実装済み

2023年度未踏ジュニアの
支援を受け実装

10 %

KOITO, ULMM, KOITO-libc

ソフトウェアの設計中
期間中に実装

Index

Q&A

マイクロカーネルの速度は?

注意深くIPCを設計することにより, 十分な速度を達成できる
[Liedtke, J.: “Improving IPC by Kernel Design” Proc. 14th Symp. on Operating Systems Principles, ACM, pp. 237-150, 1995.]

ターゲットは?

IoT時代の高度なオペレーションを必要とするシステム
既存カーネルの置き換えによる柔軟な最適化を狙う

仮想化への対応は?

1 : カーネルにHypervisor Capabilityを実装する予定がある
2 : ユーザーレベルでWASM Runtimeを実装する予定がある

いずれも期間中に実装する予定はないが, スケジュール次第

なぜ未踏?

未踏コミュニティによる人的ネットワーク構築