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

── 最速の 3rd-Generation Microkernel とその応用 ──

1. 背景

近年，計算機の民主化による **Internet of Things**（**IoT**）の拡大は大きなムーブメントを生み出している．Moore's Law で予見されたように，半導体集積回路に対する面積当たりのトランジスタ数は指数関数的に増加した． このような Hardware の技術革新により，かつては軍事や研究機関に限られていた高度な計算資源が一般にも普及し利用されるようになった． そして，これらは相互に Network を形成し，**Ubiquitous Computing** の実現に向け進化を遂げつつある． これにより，Hardware と User を繋ぐ基盤となる **Kernel** や **OS** といった Software の役割もますます重要視されるように変化している．

しかし，Linux のように **Monolithic** な構造を持つ Kernel はもはや現代の要求に適合しなくなりつつある． このような Architecture は Project 成長に伴う Code Base の肥大化が避けられず，**Trusted Computing Base**（TCB）の拡大による Security 上のリスクが増大する． また，人的リソースの限界による開発の困難化や，利害関係者の増加に伴う意思決定スピードの低下も問題となる．

Kernel を柔軟かつ安定したものとするため，**Microkernel** という Architecture が存在する．Microkernel は Monolithic Kernel とは異なり，提供する機構を最小化することで User へ最小限の **Application Programming Interface**（**API**）のみを提供するような設計手法である．したがって，ほぼ全ての System Component は User-Level で実行され．安定性と柔軟性を両立させることができる． また，User-Level で実行されるそれぞれの Component は互いに **Inter Process Communication**（**IPC**）し，ある種の Network を形成する．そのため，Ubiquitous Computing のような相互通信を基本とする System にも適する．

ただし，Microkernel は性能面の問題を抱えている．Microkernel はその特性上，IPC の実行回数が Monolithic Kernel と比較しても増加し Overhead となる． また，多くの Microkernel は Security 上の問題から Memory Management を Kernel-Level で実行する傾向にあり，Microkernel の持つ利点が損なわれてしまう問題がある．

2. 目的

Monolithic な Kernel や Microkernel の問題を解決するためには，柔軟性と安定性，安全性，そして性能を両立させる必要がある．そのため，本 Project ではこれらの要求を満たす新たな Kernel と，その Kernel を Base とした System の開発を実施した．

3. 開発の内容

本 Project では，**Object-Capability Model** による **3rd-Generation Capability-Based Microkernel** である **A9N Microkernel** を開発した．また，この Kernel を Base として OS を構築するための Framework である **Nun** と，それを用いて実装される，User-

Level で Memory Management を実行する OS である **KOITO** を開発した．加えて，A9N Microkernel を Modern に開発するための Library である **`liba9n`** も開発した．

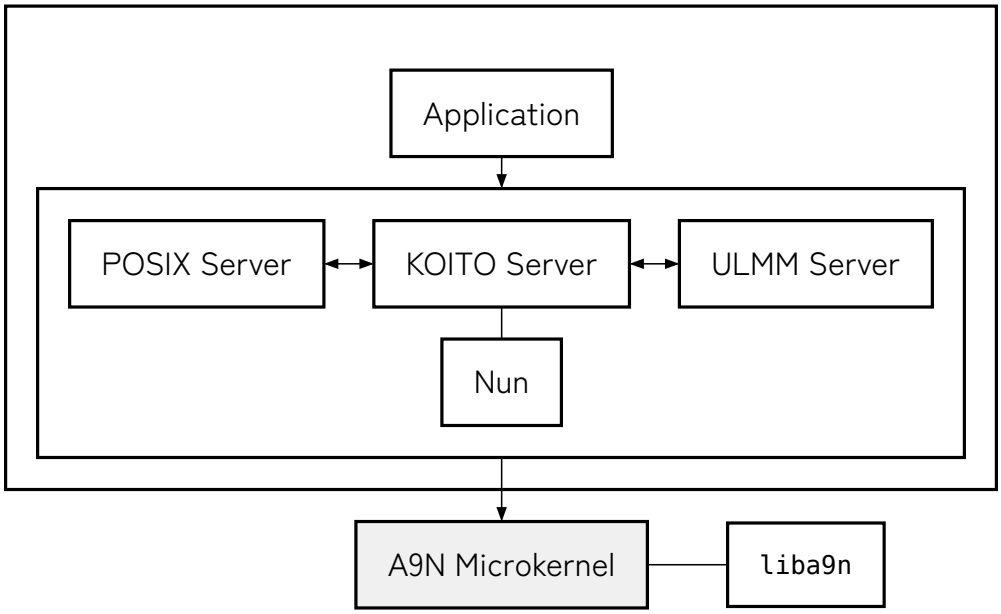

Figure 1: Project Architecture

A9N Microkernel は Capability という概念を用いることで，柔軟かつ安全に System を構築できる．Capability はある Object に対する操作権限を表す譲渡可能かつ偽造不可能な Token であり，User はこれを用いることでのみ特権的な Object に対する操作を実行できる．従来の Access Control List（ACL）とは異なり，Capability そのものが操作に必要な鍵といえる． これにより，A9N Microkernel が持つ Kernel-Level API は極めて簡素化され，種類としては Capability に対して操作を実行する **Capability Call** と，実行中 Context が Time Slice を譲る **Yield Call**，そして Debug に使用される **Debug Call** の 3 つのみになる．

それぞれの Capability は **Radix-Based Capability Node** によって管理され，Addressing される．そのため，探索にかかる時間はわずか$\mathrm{O}(\log \mathrm{n})$であり，極めて高速である．

Memory に関連する特権的操作として，**Generic Capability**，**Address Space Capability**，**Page Table Capability**，そして **Frame Capability** などが提供される．これにより A9N Microkernel は Kernel 内に Heap を持たず，完全に User-Level で Memory Management を実現する．また，これらの操作は Architecture-Independent であり，User は実装した Memory Management Server を容易に Porting できる．

また，A9N Microkernel は高速な IPC 機構を持つ．**Virtual Message Register** によって Hardware Register を効率的に利用し，Low-Latency な IPC を実現する．さらに，Client-Server Model に最適化された **Call** と **Reply Receive** が **Send** や **Receive** といった基本的な機構に置き換わり使用される（cf., Figure 2）．この機構により，IPC のさらなる高速化を実現する．

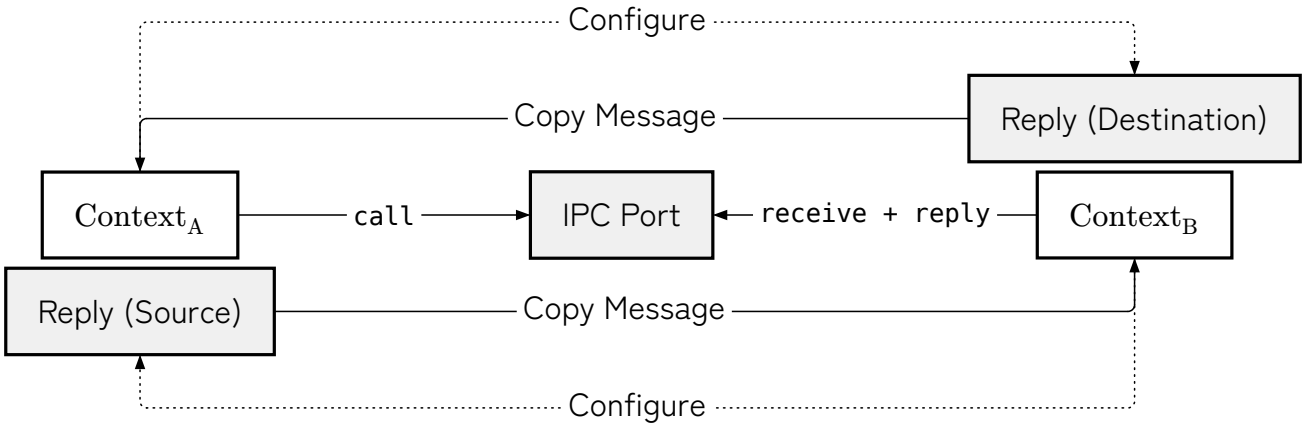

Figure 2: Call/Reply Mechanism

Nun は A9N Microkernel 上に OS を構築するための Rust 製 Framework である．Kernel は Init Protocol をもって Initial Server を起動するが，この Framework はその Protocol に沿う OS を構築するために最小限の抽象化を提供する．Rust を用いることで Type Safety や Memory Safety を保証し，また Cargo による優れた Build System を享受できる． 通常，Microkernel-Based System の Startup は複雑な処理 Flow を持つが，Nun はこれを簡略化し，User が OS を開発する際の負担を軽減する．

KOITO は Nun によって実装される，A9N Microkernel 上で動作する OS である．A9N によって提供される Mechanism を用い，それに対応する Policy を実装する Layer といえる．実態としては複数の Server で実現される抽象概念であり，Initial Server, POSIX Compatible Server, User-Level Memory Management Server によって構成される． KOITO を構成する各 Server はすべて User-Level で動作するため，極端な Usecase から普遍的な Usecase まで動的かつ柔軟に対応することが可能となる．

`liba9n` は A9N Microkernel のために設計された C++20 用の Template Library であり，Monadic Operation を用いた Modern な Error Handling 機構を提供する．. Standard Library に一切依存しないため，Embedded な Freestanding 環境や A9N Microkernel 以外の Project でも使用できる．

4. 従来の技術との相違

既存の Capability-Based Microkernel は Architecture-Dependent な API を持つことが多く，User-Level Server の Porting が困難であることが多い． 一方，A9N Microkernel は Hardware Abstraction Layer を備えることで Kernel-Level の Portability を実現し，さらに Architecture-Independent な API を提供することによる User-Level の Portability も実現している．また，A9N Microkernel は極めて高速な IPC 機構を持ち，NICTA の開発した seL4，Kernkonzept の開発した Fiasco.OC, Google の開発した Zircon といった競合となる 3rd-Generation Microkernel と比較した Benchmark（cf., Table 1）の結果，590Cycles/91.4ns という x86_64 Architecture においてほぼ世界最速といえる値を達成した．そのため，この Kernel は柔軟性，安定性，安全性，性能，さらに Portability を兼ね備えたものといえる．また，Rust-Based な OS 構築を支援する Framework の実装も他にはあまり見られない．

Kernel	Architecture	Cycles
seL4 (Fastpath)	Intel® Core™ i7-6700	786
Fiasco.OC (Fastpath)	Intel® Core™ i7-6700K	2717
Zircon	Intel® Core™ i7-6700K	8157
A9N	Intel® N150	590

Table 1: Single-Core IPC Round-Trip Latency cf., [1], [2]

5. 期待される効果

本 Project はあらゆる Usecase に対応することが可能な抽象性を持っている．そのため，Embedded Device から Server，HPC，そして一般に使用する Desktop 環境まで幅広く利用できる．あらゆる Device が Network を形成していく中で，A9N Microkernel による Ecosystem はその基盤となることが期待される．

6. 普及の見通し

現在途中まで完成している A9N Microkernel の Document に引き続き，Nun や KOITO，`liba9n` の Document を作成する．これにより新規 User を獲得し普及させていく予定である．また，複数人程度の Contributor を獲得し，継続して開発を実施する．

7. クリエータ名

伊組 烈火（フリーランス）

8. 関連 URL

https://github.com/horizon2038/A9N

A9N Microkernel

https://github.com/horizon2038/A9NLoader

A9N Boot Protocol(x86_64) に従った Reference Bootloader 実装

https://github.com/horizon2038/Nun

A9N Microkernel 上で動作する OS を開発するための Rust 製 Framework

参考文献

[1] Z. Mi, D. Li, Z. Yang, X. Wang, and H. Chen, “SkyBridge: Fast and Secure Inter-Process Communication for Microkernels,” in Proceedings of the Fourteenth EuroSys Conference 2019, in EuroSys '19. Dresden, Germany: Association for Computing Machinery, 2019. doi: 10.1145/3302424.3303946.

[2] “seL4 Performance.” [Online]. Available: https://sel4.systems/About/Performance/