A9N Manual

Version 1.0.0

written by horizon2k38

1

Table of Contents

2

Introduction

2.1. A9N Microkernel Overview

A9N Microkernel は 3rd-Generation Capability-Based Microkernel です.
最適化された IPC 機構と Object-Capability Model によるセキュリティ機構を持ち, 低 Latency かつ高信頼性なシステムを実現します. また, 高度に設計された HAL によって, さまざまなハードウェアプラットフォームに Porting が可能です.
このマニュアルは, A9N Microkernel の概要とその設計について解説するものです.

2.2. Capability

特権的操作に対する安全性は, Capability と呼ばれる偽造不可能かつ譲渡可能な Token によって保たれます. Capability は従来の ACL 的なアプローチとは異なり, その所有者がその権限を持つことを示すものです. これにより, より高い粒度の権限管理を実現します.

2.3. IPC

3

Kernel Call

3.1. Introduction

A9N Microkernel は 3 タイプの Kernel Call を提供します. 言い換えれば, たった 3 つの Kernel Call を使用することで System 全体を制御することが可能です. 呼び出しには Architecture 固有の ABI (cf., Section 20) を使用します.

3.2. Capability Call

第一引数として Capability Descriptor を受け取り, その Capability に対して操作を実行する Kernel Call です. すべての特権的操作は Capability Call を介して行われます.

Capability Call は以下のような Interface(Pseudo Code)によって実行されます :

```
capability_call(
    target_descriptor,
    operation_type,
    args ...
)
```

Capability への操作は低レベル呼び出しの Wrapper として実装されます.
`operation_type` は Library 関数自体が設定するため, 呼び出し時に省略されます.

3.2.1. Virtual Message Register

Capability Call は Virtual Message Register を介して行われます. 一定数の Message は Architecture 固有の Register に Mapping され, その数を超える Message は実行中の

Process Control Block に割り当てられた IPC Buffer[1] に格納されます. Map 可能な Register 数を超えない限り, Capability Call はコピーなしに実行可能です.

3.3. Yield

Yield は実行中の Context を Block し, Time Slice を他の Context へ移譲するための Kernel Call です. Capabiltiy Call とは異なり, すべての実行中 Context で使用可能です.

3.4. Debug Call

Debug Call は Debug に使用するための Kernel Call です. Capability を必要としませんが, Production Project における使用は推奨されません.

[1] L4 Microkernel Family における UTCB のようなもの

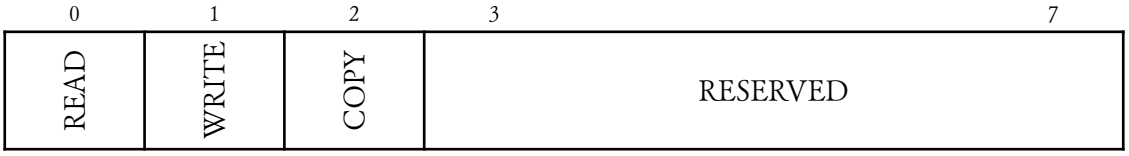
4

Capability

4.1. Introduction

4.2. Rights

4.2.1. `capability_rights`



4.3. Dependency Node

5

Capability Node

5.1. Introduction

Capability Node は, Capability を格納するためのコンテナとして使用される Capability です.
この Node は2^{radix}個の Slot を持つ Radix Tree です.
子として Node が保持可能であり, 複数階層の Capability Tree を作成できます.

5.2. Addressing

Node 内の Capability は Capability Descriptor を用いて Addressing されます :

1. Descriptor の先頭 8bit を取り出し, `depth_bits` とします
   - `depth_bits` は探索可能 bit 数の最大値を表します
   - 例えば, 64bit Computer では64 − 8の 56bit が標準の `depth_bits` となります
2. Node 内の `radix_bits` から Index に使用する Bit を決定します
3. Descriptor から 2 で得た Index 分の bit を取り出し, 子を取得します
4. 子に対して, Descriptor を使い切るか終端に到達するまで再帰的に探索を行います

Node と Descriptor は Page Table と Virtual Address のような構造をしています.

5.3. Node API

5.3.1. `copy`

Capability の Copy を行います. Rights はそのまま Copy されます.

type	name	description
`capability_descriptor`	`node_descriptor`	対象 Capability Node への Descriptor
`word`	`destination_index`	Destination となる Capability を格納している Node 内 Index
`capability_descriptor`	`source_descriptor`	Source となる Node の Descriptor
`word`	`source_index`	Source となる Node の Capability を格納している Index

5.3.2. `move`

Capability の Move を行います. Rights はそのまま Move されます.

type	name	description
`capability_descriptor`	`node_descriptor`	対象 Capability Node への Descriptor
`word`	`destination_index`	Destination となる Capability を格納している Node 内 Index
`capability_descriptor`	`source_descriptor`	Source となる Node の Descriptor

type	name	description
`word`	`source_index`	Source となる Node の Capability を格納している Index

5.3.3. `mint`

Capability の Mint を行います. 新しい Rights は元となる Rights の Subset である必要があります. 従って, より高い Rights を持つ Capability を作成することはできません.

type	name	description
`capability_descriptor`	`node_descriptor`	対象 Capability Node への Descriptor
`word`	`destination_index`	Destination となる Capability を格納している Node 内 Index
`capability_descriptor`	`source_descriptor`	Source となる Node の Descriptor
`word`	`source_index`	Source となる Node の Capability を格納している Index
`capability_rights`	`new_rights`	新しい Rights

5.3.4. `demote`

Capability の Rights を降格します. この操作は不可逆であり, 一度降格した Rights を昇格することはできません.

type	name	description
`capability_descriptor`	`node_descriptor`	対象 Capability Node への Descriptor
`word`	`target_index`	対象の Capability を格納している Node 内 Index
`capability_rights`	`new_rights`	新しい Rights

5.3.5. `remove`

Capability の Remove を行います. Dependency Node に Sibling が存在しない場合, Revoke も実行し完全に削除されます.

type	name	description
`capability_descriptor`	`node_descriptor`	対象 Capability Node への Descriptor
`word`	`target_index`	削除対象の Capability を格納している Node 内 Index

5.3.6. `revoke`

Capability の Revoke を行います.

type	name	description
`capability_descriptor`	`node_descriptor`	対象 Capability Node への Descriptor

type	name	description
`word`	`target_index`	削除対象の Capability を格納している Node 内 Index

6

Generic

6.1. Introduction

Generic はメモリを抽象化する Capability です.
A9N Microkernel はヒープを持ちません. 従って, 必要な Kernel Object を User が明示的に割り当てる必要があります.
User-Level の Kernel Object 割当ては, Generic による `convert()`メカニズムを用いて安全に実現されます. `convert()`によって作成したオブジェクトは親となる Generic の Dependency Node に登録され, 初期化処理などに使用されます.

6.2. Generic API

6.2.1. `convert`

type	name	description
`capability_descriptor`	`generic_descriptor`	対象 Generic への Descriptor
`capability_type`	`type`	作成する Capability の Type

type	name	description
`word`	`specific_bits`	Capability 作成時に使用する固有 Bits cf., Section 6.2.1.1
`word`	`count`	作成する Capability の個数
`capability_descriptor`	`node_descriptor`	格納先 Node への Descriptor
`word`	`node_index`	格納先 Node の Index

6.2.1.1. **`specific_bits`**

name	description
`Capability Node`	Node の Slot 数を表す Radix (count = $2^{\text{specific\_bits}}$)
`Generic`	Generic の Size を表す Radix (size = $2^{\text{specific\_bits}}$)
`Process Control Block`	-
`IPC Port`	-
`Interrupt Port`	-
`Page Table`	depth
`Frame`	-
`Virtual CPU`	-
`Virtual Page Table`	-

7

Process Control Block

7.1. Introduction

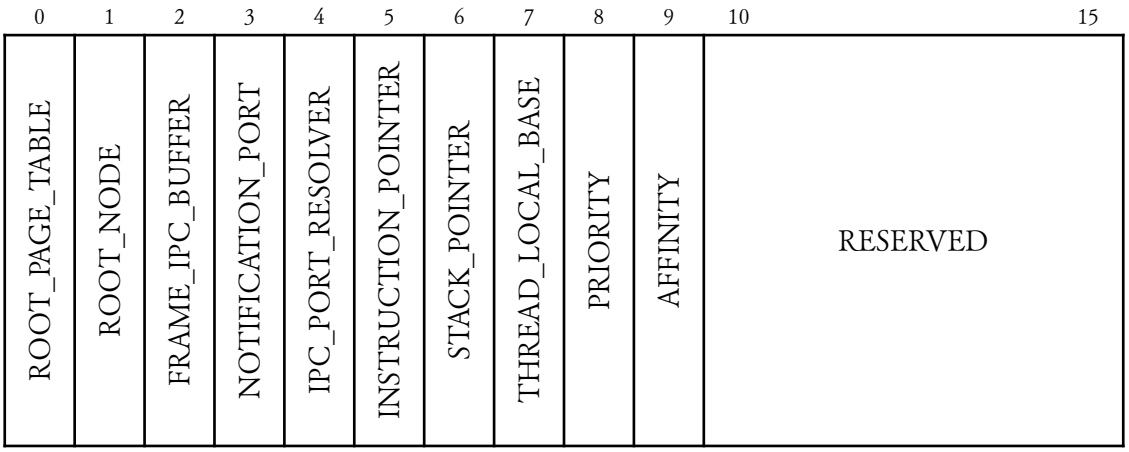
Process Control Block は実行可能な Context を表す Capability です. 他 Kernel における Thread に相当します.
Hardware Context (Register Set)と Time Quantum, Priority, 付随する Capability などが 1 つにパッケージされています.

7.2. Process Control Block API

7.2.1. `configure`

Process Control Block の設定を行います. 不必要な Copy を避けるため, Configuration Info によって設定項目を指定します.

7.2.1.1. **`configuration_info`**



type	name	description
`capability_descriptor`	`process_control_block`	対象 Process Control Block への Descriptor
`configuration_info`	`info`	設定する項目
`capability_descriptor`	`root_page_table`	Root Page Table への Descriptor
`capability_descriptor`	`root_node`	Root Node への Descriptor
`capability_descriptor`	`frame_ipc_buffer`	IPC Buffer としたい Frame への Descriptor
`capability_descriptor`	`notification_port`	Notification Port への Descriptor
`capability_descriptor`	`ipc_port_resolver`	Resolver としたい IPC Port の Descriptor
`virtual_address`	`instruction_pointer`	Instruction Pointer
`virtual_address`	`stack_pointer`	Stack Pointer
`virtual_address`	`thread_local_base`	Thread Local Base
`word`	`priority`	優先度
`word`	`affinity`	SMP 環境における Affinity (CPU Core の Index)

7.2.2. read_register

Register を読み出します.

type	name	description
descriptor	process_control_block	対象 Process Control Block への Descriptor
word	register_count	読み出す Register の数

7.2.3. write_register

Register を書き込みます.

type	name	description
descriptor	process_control_block	対象 Process Control Block への Descriptor
word	register_count	書き込む Register の数

7.2.4. resume

Process Control Block を Scheduler Queue に Push し, 実行可能状態にします.

type	name	description
descriptor	process_control_block	対象 Process Control Block への Descriptor

7.2.5. suspend

Process Control Block を実行不可能状態にします.

type	name	description
`descriptor`	`process_control_block`	対象 Process Control Block への Descriptor

8

IPC Port

8.1. Introduction

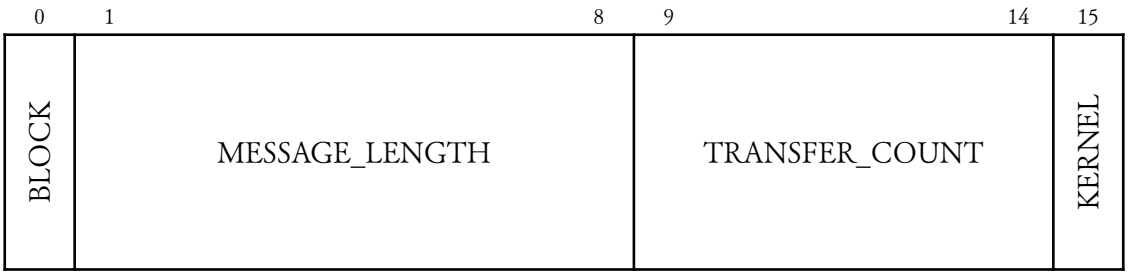
IPC Port は IPC を行うための Capability です. Sender と Receiver の Queue を持ち, 1:n もしくは n:1 の通信を可能とします.
`send` や `receive` などの一般的な操作に加え, Usecase に最適化された `call`, `reply`, `reply_receive` などの API も提供されます.

8.1.1. Reply Mechanism

TODO: -

8.2. IPC Port Data Structure

8.2.1. `message_info`



name	description
`BLOCK`	設定されている場合, IPC 操作は Block されます

name	description
`MESSAGE_LENGTH`	メッセージの長さ (WORD_BITS 単位)
`TRANSFER_COUNT`	Transfer する Capability の数
`KERNEL`	設定されていた場合, Message は Kernel からのものであることを示します. User からこの Flag を設定しても無視されます

8.3. IPC Port API

8.3.1. `send`

Message を送信します. IPC Port が送信待ち状態(Receiver Queue が構成されている)の場合, Queue から Process を 1 つ取り出し Message を送信します.

8.3.1.1. Arguments

type	name	description
`descriptor`	`ipc_port_descriptor`	対象 IPC Port への Descriptor
`message_info`	`info`	送信する Message の情報

8.3.1.2. Return Value

type	name	description
`ipc_port_result`	`result`	Message の送信結果

8.3.2. `receive`

Message を受信します. IPC Port が受信待ち状態(Sender Queue が構成されている)の場合, Queue から Process を 1 つ取り出し Message を受信します.

type	name	description
`descriptor`	`ipc_port_descriptor`	対象 IPC Port への Descriptor
`word&`	`identifer`	送信元の Identifier

8.3.3. `call`

Message を送信し Reply を受信する過程を 1 つの Atomic な操作で実行します. Reply Mechanism により, 送信した Message を受信した Process からの Reply を受信することが保証されます. Message Info に `BLOCK` Flag が設定されていない場合でも, 対象に Reply Object が設定されていない場合は Block されます.

type	name	description
`descriptor`	`ipc_port_descriptor`	対象 IPC Port への Descriptor
`message_info`	`info`	送信する Message の情報

8.3.4. `reply`

Reply Object が設定されている場合, その Object を介して該当 Process に Message を送信します.

type	name	description
`descriptor`	`ipc_port_descriptor`	対象 IPC Port への Descriptor[2]

8.3.5. `reply_receive`

Reply を実行してから Message を受信する過程を 1 つの Atomic な操作で実行します. 典型的にはスタートアップとして Receive を実行し, Reply Object を設定してからループ内で Reply Receive を実行することで高速に通信を行うことが可能です.

[2] 宛先の Descriptor は本来必要としないが, 今後の拡張を考慮し引数として受け取る

type	name	description
`descriptor`	`ipc_port_descriptor`	対象 IPC Port への Descriptor
`word&`	`identifer`	送信元の Identifier

8.3.6. `identify`

IPC Port に Identifer を付与します. この Identifer は Capability Slot Local であり, 実体として同じ IPC Port を指していても異なる Identifer を持つことが可能です.

type	name	description
`descriptor`	`ipc_port_descriptor`	対象 IPC Port への Descriptor
`word`	`identifier`	IPC Port に付与する Identifier

9

Notification Port

10

Interrupt Region

11

Interrupt Port

12

IO Port

13

Address Space

14

Page Table

14.1. Introduction

14.2. Page Table API

14.2.1. `map`

type	name	description
`descriptor`	`page_table_descriptor`	対象 Page Table への Descriptor
`descriptor`	`target_descriptor`	対象に Map する Descriptor (Page Table or Frame)
`virtual_address`	`address`	Map する仮想アドレス
`flags`	`flags`	Map に使用する Flag

14.2.2. unmap

type	name	description
`descriptor`	`page_table_descriptor`	対象 Page Table への Descriptor
`descriptor`	`target_descriptor`	Unmap する Page Table への Descriptor
`virtual_address`	`address`	Unmap する仮想アドレス

15

Frame

16

Virtual CPU

17

Virtual Address Space

18

Virtual Page Table

19

Protocol

19.1. Boot Protocol

19.1.1. x86_64

19.2. Init Protocol

19.2.1. x86_64

20

ABI (Application Binary Interface)

20.1. x86_64

21

Porting