IPA

# 2023 年度 未踏 IT 人材発掘・育成事業 採択案件評価書

## 1. 担当 PM

田中 邦裕（さくらインターネット株式会社 代表取締役社長）

## 2. クリエータ氏名

今村 翔太（慶應義塾大学 環境情報学部 環境情報学科）

## 3. 委託金支払額

2,736,000 円

## 4. テーマ名

Capability を主軸とするマイクロカーネル

## 5. 関連 Web サイト

ソースコード：https://github.com/caprese-project

## 6. テーマ概要

本プロジェクトは、従来のオペレーティングシステム（OS）の設計に対する新たなアプローチを提案し、Capability-based Security を核とするマイクロカーネル「Caprese」の開発を行うものである。

Capability-based Security とは、UNIX 系の OS で行われるアクセス制御リスト（ACL）やユーザとグループに基づくパーミッションと根本的に異なり、リソースへのアクセス権を表す Capability というトークンが使用される。例えば、Web サーバソフトウェアのプロセスが 80 番ポートで LISTEN するにあたって、多くの UNIX 系の OS では 1024 番未満のポートを開くために特権が必要というだけのために、プロセスへ root 権限を過剰付与しているが、Capability-based Security であれば 80 番ポートで LISTEN するためだけの Capability をプロセスへ付与することで必要最小限の権限付与で済む。いわば、ホテルの鍵をトークンだとすると、ホテルの自室の鍵を紛失した宿泊客へ、マスターキーを一時的に貸与するのが UNIX の権限管理であり、自室の古い鍵を失効させて新しい鍵を支給するのが Capability-based Security と言えるだろう。

これらのように、従来の OS が持つセキュリティに関する課題を克服し、セキュリティと効率性の向上を実現することがプロジェクトの目的である。なお、マイクロカーネル Caprese に加えて、Caprese 上で動作する OS のサンプルと、汎用的な C/C++標準ライブラリの実装も行い、Capability-based Security の有効性を示すとともに、実際のシステムへの応用可能性を探る。

本プロジェクトの成果は、セキュリティ強化と効率的な権限管理の実現により、将来のソフトウェア開発に新たな方向性を示すものであり、この分野の進展に大きく貢献することが期待される。

## 7. 採択理由

本プロジェクトでは、セキュリティを重視した新たな OS を作るもので、SELinux に代表される既存の OS カーネルでは不十分だった利便性や機能性を、一から設計しようとする野心的なものである。

既存の SELinux ではアクセス制御リスト（ACL）ベースで強固なセキュリティを実現しているものの、プロセスを跨いでセキュリティの欠陥を突くような攻撃手法には対応できないことがあり、Capability-based と言われるユーザの役割や資格に応じてアクセス制限を行う手法が有用である。そのため、今回のプロジェクトにおいては、capability を実装した新たなマイクロカーネルを開発し、それをベースとしたセキュリティ重視の汎用 OS を開発し、GitHub 上で OSS として公開しようとしている。

多くのプロジェクトでは既存のカーネルや OS をベースに開発することが多いが、今回はカーネルから開発を行う。目的とする高いセキュリティを実現するために、多くのハードルを超えて開発を進めること自体に未踏性があると考えて採択した。

## 8. 開発目標

本プロジェクトの達成にあたって、Capability-based Security を実装したマイクロカーネル「Caprese」を開発する。

Capability-based Security を採用することで、従来のオペレーティングシステムに比べて、より安全なシステムの構築を目指しており、このセキュリティモデルは、権限の細かい管理を可能にし、不正なアクセスや権限の乱用を防ぐことが期待される。加えて、マイクロカーネルアーキテクチャを採用することで、システムのモジュール性を高め、保守性と効率性を向上させることを目指すとともに、必要最小限の機能のみをカーネル空間に持ち、その他の機能はユーザ空間で実行させることとして、カーネル内のバグや脆弱性が相対的に低くなるように設計し、システム全体の堅牢性を高める。従来のモノリシックカーネルでは OS の主要機能がカーネル内に組み込まれているため、機能の変更や拡張が困難だったが、マイクロカーネルではメモリ管理、スケジューリング、ファイルシス

テム、ネットワークスタックなどをユーザプロセスとして実装できるため、目的に応じた機能の自由な実装や差し替えが可能になる。

Caprese 内部に実装するのは、Capability の発行と管理、プロセスの管理、メモリページテーブルの管理、プロセス間通信の中継、割り込みハンドラの中継といった必要最低限な機能だけである。ユーザプロセスとして実装予定なのは、メモリ管理サーバ、スケジューラサーバ、ファイルシステムサーバ、TCP/IP スタックサーバとし、プロセス間通信を介してカーネルやプロセス間でやりとりを行う。カーネルのブート部分はアセンブラで記述し、その他の部分は C++で実装を行い、RISC-V 64bit アーキテクチャ（rv64gc）をターゲットとし、QEMU と VisionFive2 実機ボードの両方で稼働することを目指す。

Capability については、上述の通り、Capability はカーネルが一元的に発行・管理し、全てのリソースへのアクセスは、対応する Capability を介してのみ行える。最初に起動するルートサーバがカーネルから全 Capability を受け取り、ルートサーバが必要なサーバ群を起動し、適切に Capability を移譲する（図 1）。Capability は移譲や借用が可能で、機能の制限版を作ることもでき、ファイル、メモリページ、割り込みハンドラ、プロセス間通信など全てが Capability で保護される。

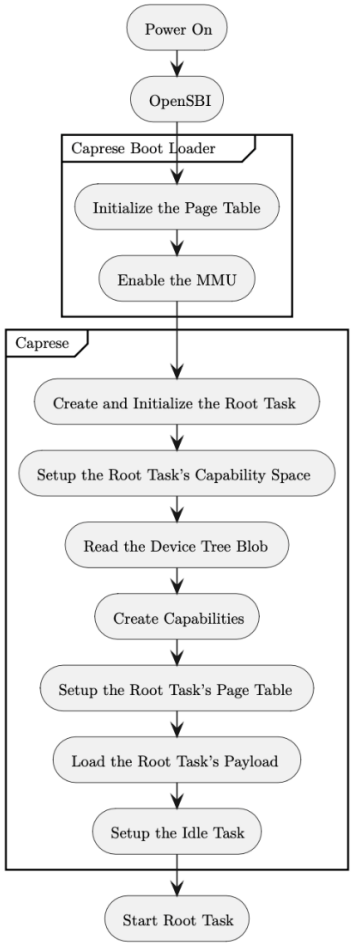


**図 1：カーネルの起動**

なお、Caprese はカーネルであり、ファイルシステムやデバイスドライバなどの基本的な OS 機能を提供するサンプル OS についても実装を行う。また、OS 上でアプリケーションを動作させるための汎用的な C/C++標準ライブラリの実装も行い、最低限のシェルやコマンドなどの開発も併せて行う。

これらの目標を達成することにより、本プロジェクトはセキュリティの向上と効率的な権限管理を実現し、将来のソフトウェア開発に新たな方向性を示すことを目指す。

## 9. 進捗概要

本プロジェクトにおいては、まず Capability-based Security を実装したマイクロカーネル「Caprese」を開発した。Caprese は開発言語として C++を、ビルドのために CMake を採用しており、ボードのファームウェアとして OpenSBI を利用している。ログを出力するためには UART などのシリアル通信ドライバをカーネルに組み込む必要があったが、ドライバを組み込むとカーネルの複雑性が増してしまうことから、OpenSBI をファームウェアとし、OpenSBI が提供するコンソール出力を利用することで、Caprese 自体はデバイスドライバを持たずに済むようにした。

Caprese が提供する Capability が保護するリソースはメモリ領域などのハードウェアリソースおよびプロセスなどのカーネルオブジェクトである。Capability の種類により複製できるものとできないものがある。Caprese が提供する Capability については 7 種類あり、その具体的な種類については以下の通りである。

- Memory Capability

  Memory Capability は複製不可能であり、重複しない物理メモリの範囲を表す。この Capability 単独では特に何かできるわけではないが、領域の一部をほかの Capability に変換することが可能であり、変換した Capability の種類に応じてその領域に対する操作が可能になる。例えば、後述する Task Capability に変換した場合、その領域はカーネル内のプロセス管理用の領域として利用され、そのプロセスに対する操作が可能になる。

  Memory Capability によりメモリ管理をユーザランドに委譲することで、カーネル自身がヒープを持たずメモリ管理を行うことがなくなり、カーネルの複雑性が低減される。
- Task Capability

  Task Capability は複製可能であり、特定のプロセスに対する操作を提供する。この Capability を利用することによって、ユーザアプリケーション側でプロセスを管理する機構を実装したり、別のプロセスへ Capability を委譲/転送したりすることができる。

- Endpoint Capability
  Endpoint Capability は複製可能であり、プロセス間通信におけるチャネルの役割を担う。
- Page Table Capability
  Page Table Capability は複製不可能であり、プロセスの仮想アドレス空間を構成するページテーブルを表す。この Capability と後述する Virt Page Capability を利用することで、プロセスの仮想アドレス空間をユーザアプリケーション側から構成することができる。
- Virt Page Capability
  Virt Page Capability は複製不可能であり、特定のページを表す。この Capability と前述の Page Table Capability を利用することで、プロセスの仮想アドレス空間をユーザアプリケーション側から構成することができる。
- Cap Space Capability
  Cap Space Capability は複製不可能であり、後述する Capability 領域を表す。この Capability は特に機能を持たず、カーネルにメモリ領域を割り当てるために使用される。これにより、プロセスが持てる Capability の上限数を増やすことができる。
- ID Capability
  ID Capability は複製可能であり、システム全体で一意となる値を表す。この Capability はユーザアプリケーションが提供するリソースに対して一意な ID を割り当てるために使用される。

Capability はプロセスごとに存在する Capability 領域に配列上に格納されている。配列であるため、インデックスによって Capability を特定することができる。プロセスはこのインデックスを引数としてシステムコールを呼び出すことで Capability を利用する。この仕組みによって、ユーザアプリケーションに対して Capability の実体を秘匿しつつ、Capability を行使することができる。また、Capability 領域はプロセスごとに固有であるため、異なるプロセスが同じインデックスを指定しても異なる Capability を指すこととなり、Capability を奪取したり偽造したりすることはできない

Caprese におけるプロセスとは、実行コンテキストの単位である。カーネルが保持する内容は、プロセスの状態、レジスタ、待ちキュー、ルートのページテーブル、所持している Capability のリストである。デバッグ用にプロセス ID のような概念を持つが、これを用いてプロセスを制御することはできず、Capability を介してのみ制御をおこなうことができる。プロセス間通信については、Endpoint Capability を介して提供している。全ての API が同期的であり、非同期的な通信は提供していない。これは、非同期的な通信を提供することでプロセス間の状態を管理するための仕組みやカーネルによるメモリの管理が必要

となり、カーネルの複雑性が増してしまうためである。パフォーマンス向上のため、短いメッセージのための API と長いメッセージのための API を分けて提供している。短いメッセージのための API は、メッセージの内容をレジスタに乗せて送信するため、カーネル内でのメモリのコピーを最小限に抑えることができる。一方で、長いメッセージのための API は、メッセージの内容をメモリに置いて送信するため、カーネル内でのメモリのコピーが発生する。また、長いメッセージのための API は Capability を送信することもでき、これにより各々のプロセス間通信のフォーマットに Capability の送受信を組み込むことができる。

Caprese は OS やドライバ、ユーザアプリケーションが Capability を利用するためのインタフェースとしていくつかのシステムコールを提供している。Capability に対する操作のほか、次のシステムコールが Capability を不要とする操作として提供されている。

- sys_system_null
  何もしない。
- sys_system_core_id
  現在実行されている CPU コアの ID を取得する。
- sys_system_page_size
  このアーキテクチャにおける最小ページのサイズを取得する。RISC-V であれば 4096 が返る。
- sys_system_user_space_start
  ユーザ空間の開始アドレスを取得する。Page Table Capability はこのアドレス以降にのみマップできる。
- sys_system_user_space_end
  ユーザ空間の終了アドレスを取得する。Page Table Capability はこのアドレス以前にのみマップできる。
- sys_system_caps_per_cap_space
  一つの Capability 領域が持てる Capability の数を取得する。
- sys_system_yield
  プロセスを実行可能状態にし、別のプロセスへ遷移する。遷移先のプロセスはカーネルの待ちキューの先頭プロセスとなる。
- sys_system_cap_size
  特定の種類の Capability を作成するために必要なメモリサイズを取得する。
- sys_system_cap_align
  特定の種類の Capability を作成するために必要なメモリアライメントを取得する。

なお、開発期間の全体を通して、Caprese は何度か再設計と再実装を行っている。これは、実装中により良い仕様を検討し、それに基づいて再設計を行うことで、より良いカーネルを実現するためである。例えば初期の段階のカーネルではカーネル自身がメモリを管理しており、Linux で言うところの kmalloc に当たるような機能を持っていた。しかし、これは Memory Capability の機能と重複しているため、カーネルの複雑性が増してしまうという点と、カーネルにヒープを持たせる必要があるような操作は全て Memory Capability を用いることでユーザランド側に代替できるであろうと考えた点から、カーネル自身は全くメモリを管理しないような設計へと変更した。

次に Caprese 上で動作する OS のサンプル実装について説明する（図 2）。この OS は、Caprese による Capability の機構が実際に OS やドライバ開発に適合することを示すために開発されたもので、OS 実装の一例であり、Caprese の上で動作する OS は必ずしも本 OS と同様の実装方法である必要はない。

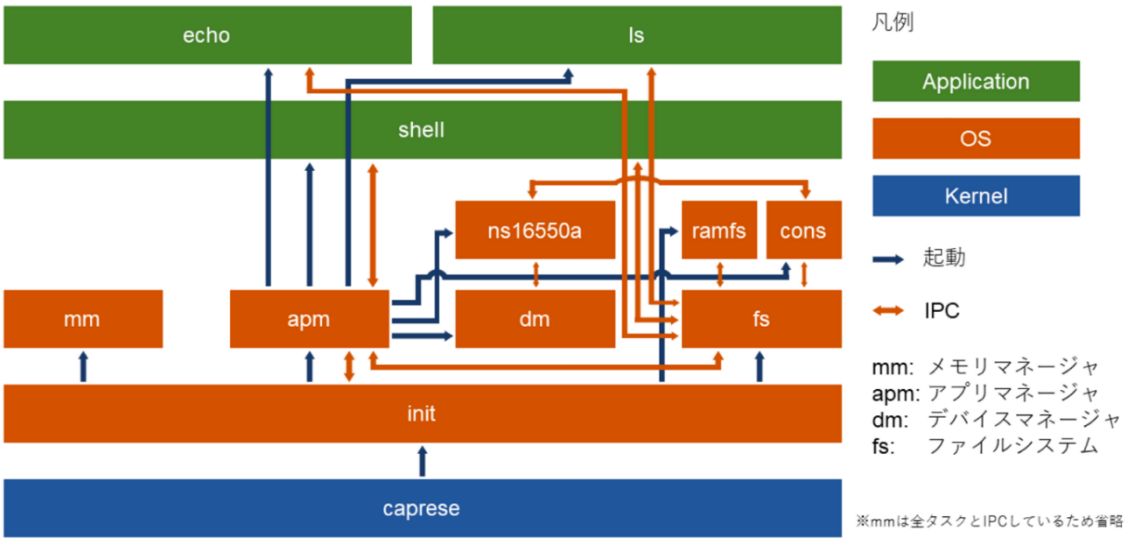


**図 2：サンプル OS の構成図**

具体的に現時点で提供されているプロセスと、そのプロセスでどのように Capability を利用しているかを説明する。

- init

  Caprese のブートが終わった後に最初に起動されるプロセス。このプロセスは Caprese から全ての Capability を受け取っており、それらを利用して他のプロセスを起動する。
- mm

  メモリ管理を行うプロセス。このプロセスは Memory Capability や Page Table Capability、Virt Page Capability を利用してメモリを管理する。このプロセスにより、他のプロセスはヒープを利用できる。

- apm
  プロセスを管理するためのプロセス。このプロセスは Task Capability を利用してプロセスを起動したり、終了したりするなどの操作をする。
- dm
  デバイスドライバを管理するためのプロセス。このプロセスは Device Tree Blob を読み込み、適切なデバイスドライバを起動する。
- ns16550a
  UART を利用するためのデバイスドライバプロセス。このプロセスにより、コンソールに文字を出力することができる。
- fs
  ファイルシステムを利用するためのプロセス。このプロセスは Linux で言うところの vfs に相当するものであり、仮想ファイルシステムを提供する。このプロセス自身はファイルシステムを持たず、配下のプロセスを抽象化しつつプロセス間通信の中継を行うことで、どのようなファイルシステムが利用されているかを意識せずにファイルシステムを利用できるようになっている。また、本 OS ではファイルディスクリプタとして ID Capability を用いている。
- ramfs
  オンメモリのファイルシステム。fs とプロセス間通信をしてファイルシステムを構成する。内部表現は CPIO である。
- cons
  コンソールへの入出力を行うファイルシステム。このプロセスは fs とプロセス間通信をしてファイルシステムを構成する。Linux で言うところの tty に相当するものであり、行単位での入出力を行う cooked モードと文字単位での入出力を行う raw モードを提供している。
- shell
  ユーザとの対話を行うためのプロセス。このプロセスは標準入力からコマンドを受け取り、apm へプロセス間通信をしてプロセスの起動を行う。
- echo
  標準出力に実行時引数で受け取った文字列を出力する。
- clear
  標準出力をクリアする。
- ls
  実行時引数で指定したディレクトリに含まれるファイルを表示する。
- printenv
  環境変数を表示する。
- pwd
  現在のディレクトリ（PWD 環境変数）を表示する。

- touch
  ファイルを作成する。

Caprese および OS のサンプルに加えて、その上でアプリケーションを開発するための C/C++の標準ライブラリについても開発を行った。Caprese や OS のサンプルは POSIX や WASI などの既存の API とは違うインタフェースを持つため、独自の標準ライブラリを提供する必要があった。

Caprese はファイルシステムなどの OS として必要とされる機能を持たず、上に乗る OS のインタフェースを POSIX などの既存のものに制限しない。そのため、OS 機能に依存する部分をスタブとして提供するような C/C++標準ライブラリがあれば、どのようなインタフェースをもつ OS を開発しても、その OS に向けた C/C++標準ライブラリの実装が容易になると考え、本ライブラリの開発に至った。これにより OS 開発者は OS 機能の実装のみに集中できるため、OS 開発の生産性向上に役立つと考えられる。

## 10. プロジェクト評価

本プロジェクトにおいては、これまでの UNIX 系 OS におけるセキュリティアプローチとは全く異なる Capability-based Security を実装するために、独自のカーネルや OS コマンド群、ライブラリなどを一から開発し、約 5 万行という大規模な成果を生み出した。当初より目標としていた Capability-based Security を実装するマイクロカーネルを開発し、サンプル OS や標準ライブラリを実装し、実機で稼働させることまで成功しており、ネットワーク機能以外の部分について、全て目標を達成している。

元々は、マイクロカーネルを作りたいという好奇心から始まったプロジェクトではあるが、それにセキュリティ向上という意味を持たせて、今後の発展にもつながるアプローチを完遂させたことは素晴らしい。

## 11. 今後の課題

本プロジェクトの今後の課題には、機能の更なる拡充、実用性の検証、ドキュメントとサポートを充実させコミュニティを構築することなどが挙げられる。

まず機能の拡充でいうと、Caprese および Caprese 上で動作する OS には、まだ実装されていない改善すべき点や機能の拡張が存在する。より高度なスケジューリングアルゴリズムの実装、セキュリティの強化、パフォーマンスの最適化などが挙げられるほか、ネットワーク機能の実装などの機能面の充実が求められる。また Caprese の実用性をさらに検証し、実際のアプリケーションやシステムでの適用可能性を探ることが重要であり、特に当初の提案書で述べられていた IoT 機器や分散コンピューティング環境での利用に向けた検証が深められることを期待する。

ドキュメントやサポートという点では、開発者やユーザが Caprese や関連するライブラリを容易に利用できるようなチュートリアルや、丁寧なドキュメントの作成に加え、オープンソースプロジェクトとして、開発者やユーザのコミュニティを構築し、活発な情報交換や協力体制を築くことが重要である。特に、コミュニティの支援によって、プロジェクトの発展や問題解決が加速されることが期待される。