MirageOS における IoT 向け入出力制御の実装と評価

2019SE002 青山飛鳥
指導教員：宮澤元

1 はじめに

近年，Internet of Things(IoT) の技術を用いた IoT デバイスが急速に普及している．しかし，IoT デバイスは開発コストの制約から，デバイスに潤沢な計算資源を搭載することが難しく，計算資源の不足が問題視されている．また，計算資源の不足から，十分なセキュリティ対策を行うことが難しい点も問題となっている．

これらを解決するために，Unikernel を IoT デバイスに導入する方法が検討されている [1]．Unikernel は，ライブラリ OS を利用して構築された単一アドレス空間のマシンイメージを提供する技術である．代表的な Unikernel に Mirage Unikernel(MirageOS) がある．MirageOS 上のアプリケーションは型安全な言語を用いて記述されるので，一般の Unikernel 上のアプリケーションより高いセキュリティを持つ．

本研究の目的は，IoT デバイスにおいてよりセキュアなアプリケーション実行環境を実現するために Unikernel 上で OCaml から利用できる IoT 向け入出力インタフェースを提供することである．Unikernel はもともとクラウド環境で利用されることを想定した技術であり，IoT デバイスでの利用は想定されていない．そこで，IoT デバイスが持つ特有の入出力装置に Unikernel 上の OCaml アプリケーションから簡単にアクセスする方法が必要である．

本稿では, IoT デバイス上の Unikernel において OCaml アプリケーションから利用できる入出力装置の制御を行うインタフェースとその実現について述べる．Unikernel 用のサンドボックス環境に実装するハイパーバイザーコールを OCaml から呼び出すインタフェースを介して，入出力制御を行う．

研究課題は以下の 2 点である．

1. OCaml 向け GPIO インタフェースの実装
2. 実装したインタフェースを用いたアプリケーションの実行性能の評価

2 研究の背景

本節では, 関連研究として Unikernel と代表的な Unikernel である MirageOS について述べる．また，Unikernel における入出力制御に関する先行研究も紹介する．

2.1 Unikernel

Unikernel は単一の目的を果たすことに特化しており，目的外の機能が取り除かれている小さな OS なので，従来の OS を用いる場合よりも必要となる計算資源の削減ができる．また，イメージサイズが小さいので，外部からの攻撃対象領域も小さくなり，セキュリティ面での問題の改善にも役立てることができる [2].

MirageOS は Unikernel の 1 つであり，安全で高性能なネットワークアプリケーション用の Unikernel を構築することができる [2]．MirageOS が外部からの攻撃に対するセキュリティが高い理由として，型安全な言語の使用が挙げられる．1 つの言語のみを使用することで，ソースレベルの下位互換性を失う代わりに，外部向けのクラウドサービスのセキュリティを大幅に上昇させている．

hvt/Solo5 は Unikernel を動作させるために必要なソフトウェアである．hardware virtualized tender(hvt) は仮想マシン環境を作り出すことができ，Unikernel を動作させるための仮想マシンレイヤである．hvt は Unikernel バイナリのロードと，これを元にした仮想マシンの作成と運用を行う．対して，Solo5 は様々な Unikernel ベースのアプリケーションをサンドボックス環境で動作させるためのソフトウェアである．Solo5 は複数のプラットフォーム上で，Unikernel アプリケーションを動作させるための抽象化レイヤを担う．

2.2 先行研究

赤羽根らは，IoT デバイス上で動作する Unikernel から General-Purpose Input/Output(GPIO) の制御を行う共通のインタフェースを実現するためのシステムを試作した [3]．GPIO とは IoT デバイスを含む様々なコンピュータに実装されている汎用入出力端子である．この GPIO に接続された入出力装置を制御対象としたインタフェースが実現されている．

実現されたインタフェースでは C で記述されたアプリケーションのみが実行可能であるが，C アプリケーションはメモリアクセスなどの処理の記述が容易だが，メモリ管理機能を持たないためメモリリークやバッファオーバーフローなどが発生しやすいという問題がある．

3 MirageOS による入出力装置の制御

本節では，MirageOS 及び hvt/Solo5 に追加するインタフェースについて述べる．また，実装を行うシステムの構成と処理の順序についても述べる．

3.1 実装するインタフェース

GPIO の各端子は有効・無効や入出力をユーザ側で自由に設定することができる．つまり，GPIO の制御を行うためには最低限これらの設定や入出力の操作ができなければならない．これらを踏まえて，GPIO を制御するために必要となるインタフェースを以下のように決定した．

開始　GPIO 端子を有効にして制御を開始する
設定　GPIO 端子の入出力の設定を行う

入力　GPIO 端子に接続された装置から入力を行う
出力　GPIO 端子に接続された装置へ出力を行う
終了　GPIO 端子を無効にして制御を終了にする

3.2　システムの構成と処理の順序

MirageOS 及び hvt/Solo5 に IoT デバイスが持つ入出力装置を制御するためのインタフェースを追加する．図 1 は，実装を行うシステムの構成と処理の順序を示しており，図 1 中の矢印は，GPIO を制御する際の処理の順序を示している．基本的な処理の順序としては，アプリケーションからの GPIO 制御命令を MirageOS，Solo5，hvt の順で受け取り，GPIO に接続された入出力装置の操作を行う．操作後は hvt，Solo5，MirageOS の順で GPIO の操作結果が返ってくる．

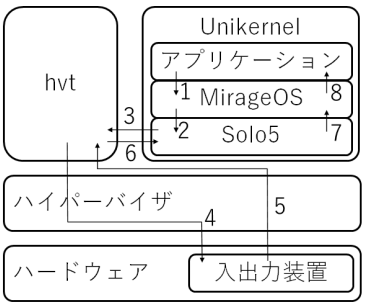

図 1　システムの処理順序

4　実装

本節では，hvt/Solo5 及び MirageOS に実装したインタフェースについて述べる．本実装では IoT デバイスとして Raspberry Pi 4 Model B(Raspberry Pi 4) を用いる．

4.1　hvt/Solo5 のインタフェース

hvt/Solo5 に GPIO を制御するインタフェースを以下のように実装した．

xx_gpio_set　GPIO を有効にして入出力の設定を行う
xx_gpio_input　GPIO に接続された装置から入力を行う
xx_gpio_output　GPIO に接続された装置へ出力を行う
xx_gpio_clear　GPIO を無効にして制御を終了にする

上記のインタフェース名の xx は，hvt ならば hypercall，Solo5 ならば solo5 となっている．これらのインタフェースを実装することで，hvt から GPIO の制御を行うハイパーバイザーコールが提供される．

4.2　OCaml のインタフェース

OCaml から GPIO を制御するためのインタフェースを以下のように実装した．

ml_gpio_set　xx_gpio_set を呼び出す
ml_gpio_input　xx_gpio_input を呼び出す
ml_gpio_output　xx_gpio_output を呼び出す
ml_gpio_clear　xx_gpio_clear を呼び出す

上記のインタフェースを実装することで，MirageOS に GPIO の制御を行う機能が hvt/Solo5 から提供されるので，アプリケーションはこれらの関数を利用して GPIO に対する制御が可能となる．

5　評価実験

実装したインタフェースを利用するアプリケーションの実行性能を確認するために，C 言語と OCaml，C 言語のみのインタフェースをそれぞれ用いるアプリケーションの実行にかかる CPU 時間と実行時間を計測した．また，実行時間は hvt から GPIO を操作する際の sysfs インタフェースの都合で 6 秒の sleep 時間を含む時間である．計測には perf コマンドを利用し，hvt/Solo5 の起動からアプリケーションが終了するまでにかかる時間を計測した．実行するアプリケーションは，GPIO に接続された LED を点灯/消灯させる処理を 10 回繰り返すものである．

表 1　実行にかかった CPU 時間と実行時間の平均

	C 言語のみ	C 言語と OCaml
CPU 時間（ミリ秒）	10.27	24.21
実行時間（秒）	6.024	6.039

表 1 は実験で計測した CPU 時間と実行時間の平均値である．C 言語と OCaml の方が C 言語のみよりも CPU 時間及び実行時間がかかることが分かった．この時間差は，OCaml での実行オーバーヘッドによるものと考えられる．

6　おわりに

本稿では，Raspberry Pi 4 の GPIO を制御するための OCaml インタフェースの実現について述べた．また，今回の実験では MirageOS にインタフェースを実装することによって，アプリケーション実行時の CPU 時間は C 言語のみの場合と比べて約 2.4 倍となることが分かった．

今後の課題としては，リアルタイム制御が挙げられる．IoT デバイスはリアルタイム性が求められるが，今回は割り込み制御を KVM に任せているためリアルタイム性について深く考慮していない．将来的にリアルタイム制御が求められる場合，KVM だけでは不十分なので，割り込み制御も考慮したインタフェースが必要になると考えられる．

参考文献

[1] T. Imada, “MirageOS Unikernel with Network Acceleration for IoT Cloud Environments,” *in IC-CBDC’18:Proceedings of the 2018 2nd International Conference on Cloud and Big Data Computing*, pp. 1-5, 2018.

[2] A. Madhavapeddy et al., “Unikernels: Library Operating Systems for the Cloud”, *in the Proceedings of the 18th international conference on ASPLOS’13*, pp. 461-472, 2013.

[3] 赤羽根 光平, 他, “IoT デバイスにおける軽量 VM 向け入出力制御機構の試作,” 南山大学理工学部 2019 年度卒業論文，2020.