IoT デバイスにおける軽量 VM 向け入出力制御機構の試作

2016SE001 赤羽根 光平　2016SE046 黒田 竜太郎
2016SE056 水野 彰大
指導教員：宮澤 元

1　はじめに

Internet of Things (IoT) の普及に伴い, IoT デバイスの計算リソースが少ないことによる問題点が意識されるようになっている. 特に IoT デバイスはインターネットに接続されるので, 計算リソースの不足によって十分なセキュリティ対策を取ることができない問題は深刻である．IoT デバイスでは CPU やメモリが限られており, PC などで用いられるウィルススキャンや高度な暗号化といったセキュリティ対策手法を利用しにくい.

IoT の問題点を改善するために, IoT デバイスで動作するソフトウェアを Unikernel として動作させる方法が検討されている [2]． Unikernel は軽量仮想マシン (VM) とも呼ばれ, VM の限られたリソースを効率的に利用するために提案された技術である [3]. Unikernel とは, VM 上で直接動作できるようにライブラリ OS を使用して単一バイナリイメージとして構築されたソフトウェアである．VM 上で既存の OS を動作させ, その上でアプリケーションを動作させる従来の手法と比較して必要な計算リソースが少ない. また, ソフトウェアのイメージサイズが小さく外部からの攻撃にさらされる面が小さいので, セキュリティの改善も期待できる [1, 3] .

Unikernel はもともとクラウド向けの技術であり， PC やサーバで標準的に利用される仮想ネットワークや仮想ディスクといった仮想入出力装置のみを持つ VM 上で動作することを想定している．一方，IoT デバイスには様々なものがあり，それぞれ IoT デバイス特有の入出力装置としてスイッチやセンサ，アクチュエータといった固有の入出力装置を持つ．IoT デバイス上で動作する Unikernel からこれらの入出力装置を制御するためには，各 IoT デバイスごとに個別の入出力装置に対応するようにハイパーバイザや Unikernel 内のライブラリ OS を変更する必要がある．

本研究の目的は，IoT デバイス上で動作する Unikernel からセンサやスイッチといった VM の仮想入出力装置としては通常提供されないような入出力装置を制御する枠組みを実現することである．以下の 2 点を研究課題とする．

1. Unikernel から IoT デバイスの入出力装置を制御するためのインタフェースの定義
2. 1. のインタフェースを実現するためのシステムの試作

本稿では IoT デバイス上で動作する既存の Unikernel システムに対して追加する入出力装置制御インタフェースの設計と実装について述べる．IoT デバイスとして Raspberry Pi3(RPi3) を用い，GPIO に取り付けた外部入出力装置を既存の Unikernel システムである hvt/solo5 を介して操作するシステムを試作する．

2　研究の背景

本節では IoT が抱える問題点と，それを解決するために研究されている Unikernel ， hvt/solo5 について述べる．

2.1　IoT

近年 IoT 技術が普及し，これまでインターネットに接続されなかったモノまでインターネットに接続されるようになった．中にはディスプレイを持たないモノもあり，外部からの攻撃を察知しづらいモノもある．また，計算リソースに乏しいために，セキュリティ対策を満足に行えないモノも少なくない．IoT の脆弱性として，Web インターフェースが安全でない，認証や承認が不十分である，ネットワークサービスが安全でないなど様々な問題を抱えている [2].

2.2　Unikernel

Unikernel を IoT デバイスに組み込むことでセキュリティ上の問題点を解決する試みがなされている．ここで定義される IoT デバイスとは，IoT におけるモノにあたり，具体的にはスマートフォンや最小限の OS であるが故に高い機密性を備えた Unikernel を利用することで，少ないリソースの中で十分なセキュリティを構築することが期待できる．

Unikernel とはクラウドまたは組み込み環境に展開できる，ハイパーバイザ上で動作する単一アプリケーション用の実行環境である．仮想化を行わない通常のアプリケーション実行環境では，ハードウェア上に OS が存在し，その上で各アプリケーションが実行される．ハイパーバイザ型の仮想化の場合，ハードウェア上にハイパーバイザが存在し，その上に 1 つ以上の OS があり，各 OS の上でそれらに紐づいた複数のアプリケーションが実行される． Unikernel もハイパーバイザ型の仮想化と大まかな仕組みは変わらないが， Unikernel の最大の特徴は， 1 つの OS の上で，単一のアプリケーションのみが実行されることである (図 1)．すなわち Unikernel では，アプリケーション 1 つを動作させるためだけの機能を持った OS が，アプリケーションの数だけ存在することになる．このために Unikernel は， Linux OS 等を用いた通常の仮想化と比べても，OS のイメージサイズを小さく済ませることができる．Unikernel は，一般的な OS が提供するサービスをライブラリ形式で提供する，ライブラリ OS を用

いて構築される． Unikernel を実装するには，ベアメタル環境や， Xen や KVM のようなハイパーバイザ型の仮想化環境を用意したうえで，IncludeOS や MirageOS ，Rumprun といったライブラリ OS を用いたシステムが必要となる．これらのシステムは，使用するにあたってのメリット・デメリットをそれぞれ持っており，用途別に使い分ける必要がある．Unikernel を用いた際のメリットとして， Unikernel Systems 社 (現在では Docker 社に買収) では，セキュリティの向上，少ないメモリ使用量，高度な最適化，高速な起動の 4 つが挙げられている [4]．またこれらに関して，多くの論文上でどのように実現されるかの説明，あるいはどのようなデメリットが付随するかの指摘がなされている．特にセキュリティに関しては， Unikernel では不要なサービスとコードを削除され，必要なカーネルの依存関係のみが残るため，攻撃対象領域が減少するメリットが存在する．しかし， Unikernel はハイパーバイザ型の OS が持つような高度なセキュリティメカニズムをすべて削除しているために，攻撃者が Unikernel アプリケーションコードに脆弱性を発見した場合，ゼロデイ攻撃が行えてしまうという指摘もある [5, 6]． Unikernel を用いる事のデメリットとしては，このようなセキュリティ上の問題のほか，開発されてから日が浅いためにナレッジの絶対量が少ないという根本的な問題もあり，Unikernel は未だ発展途上の技術であると言える．

図 1 通常，ハイパーバイザ型，Unikernel の実行環境の比較

2.3 hvt/solo5

本研究で用いる Unikernel には， C 言語で書かれたオープンソースソフトウェアである， Solo5 を使用した．Solo5 は VM 上で Monitor として動作し，アプリケーションの動作に必要のないインターフェースを省くことでインターフェースの特殊化を担い， Unikernel と hvt 間の通信を支援する．Solo5 は VM が提供する仮想入出力装置を操作することができる．VM 上で行われた仮想入出力装置に対する操作を受け取るためには，hardware-virtualized tender(hvt) と呼ばれるプログラムを使用する． hvt は， Unikernel を動作させることに特化した軽量実行環境である． hvt は，以前は Linux/KVM で動かすことを前提としていたために， ukvm の名称で開発されていた． KVM/QEMU システムのユーザレベル側に代わるものとして Solo5 と連携して動作を行うことにより，単体では人間とやり取りして入出力を行う機能を持たない Solo5 が，仮想入出力装置と hvt を介して入出力を行えるようになる．従来の Unikernel では，その多くが QEMU 等に代表される仮想マシンエミュレータを用いている．しかし， QEMU は汎用的なプログラムであるために， Unikernel に不要なコードが多く組み込まれている．そのために，軽量であることを目的としている Unikernel にもかかわらず，徒に容量を増加させてしまう．対して hvt の場合，Unikernel を動作させることに特化しており，Solo5 に必要ないものを載せる必要がなく，軽量化することができる．文献 [7] では， QEMU と比較して hvt を使用する場合には， Solo5 を実装するために必要なコードの行数が明らかに少なっていることや，起動時間が大幅に短縮されることが示されている．図 2 は，従来の Unikernel のように QEMU 等 を使用した場合と， hvt を使用した場合の構造を比較した図である．従来の Unikernel では，Linux/KVM 上で QEMU 等の仮想マシンエミュレータを動作させ，その上で Solo5 等の Unikernel を動かしている．それに対し hvt の場合は，Linux/KVM 上で Solo5 と一体化された hvt が実行される．この構造の差が，起動時間の短縮に大きく寄与している．solo5 以外の Unikernel の実装として mirageOS や OSv などが存在する．これらは solo5 と同じくハイパーバイザ上にアプリケーションとアプリケーションのランタイムをパッキングした Unikernel が動作するものであるが，これらが tap や virtio のデバイスドライバを直接操作してハイパーバイザと情報の入出力を行うのに対し，solo5 はメモリバッファを介して hvt に呼びかけ，ハイパーバイザとの入出力を行う．

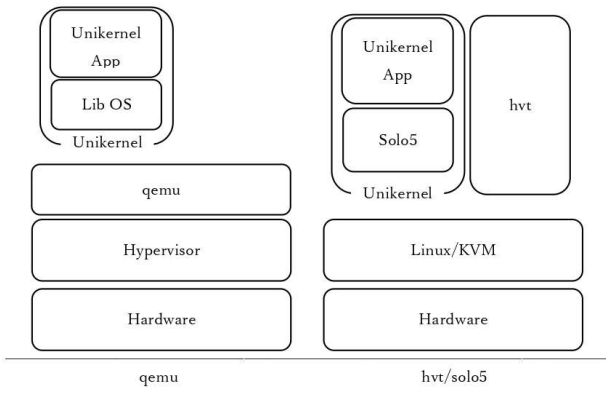

図 2 従来の Unikernel と hvt/solo5 の構造比較

3 軽量 VM 向け入出力制御機構の設計

本研究では，hvt/solo5 に IoT デバイス特有の入出力装置を制御するためのインターフェースの追加を行う．IoT デバイス特有の入出力装置として機能のオンオフを切り替えるスイッチや周囲の情報を取り込むセンサ，周囲の環境に働きかけるアクチュエータなどがあるが， hvt/solo5 で

はこれらの入出力装置を制御するための機構が備わっていない．現状の hvt/solo5 では一般的な仮想環境上で標準的な入出力として利用されるコンソール出力，ネットワーク入出力，ブロック入出力を行うインターフェースが提供されている．ここに IoT デバイスで利用される入出力装置を制御するためのインターフェースを追加することを目的とする．

本節では，Unikernel から IoT デバイスの入出力装置を制御するためのインタフェースの設計について述べる．制御する入出力装置として，多くの IoT デバイスに用意されている GPIO を対象とする．

3.1 GPIO とその入出力

GPIO とは “Genaral-Purpose Input/Output” を意味し，IoT デバイスを含む多くのコンピュータに用意されている汎用的な入出力回路である．多くの場合，集積回路やコンピュータボード上のピンであり，このピンに接続された装置の制御を GPIO を介して行うことができる．GPIO を搭載したデバイス上には，各 GPIO ピンに対応する制御レジスタが存在し，ユーザはこのレジスタにアクセスすることによって各 GPIO ピンの入力/出力の動作を制御する．今回使用する Raspberry Pi 3 には 40 ピンの GPIO が実装されている．任意の GPIO ピンを入力/出力ピンとして指定し, 様々な装置に接続して利用できる．

3.2 Unikernel に対して提供される GPIO インタフェース

GPIO を制御するためには，各ピンの入力/出力動作を設定した後，具体的な入力/出力処理を行う必要がある．そこで，以下のようなインタフェースを提供する．

Initialize　GPIO 制御を初期化する．

Finalize　GPIO 制御を終了する．

In　GPIO ピン番号を引数として入力動作するように設定する．

Out　GPIO ピン番号を引数として出力動作するように設定する．

Read　入力動作するよう設定された GPIO ピン番号を引数として入力を行う．

Write　出力動作するよう設定された GPIO ピン番号と出力データを引数として出力を行う．

4 軽量 VM 向け GPIO 制御機構の試作

3 節で述べた GPIO 制御インタフェースを hvt/solo5 に実装する．solo5 のバージョンは v0.4.1 を使用した．IoT デバイスとして Raspberry Pi3 を用い，OS には Linux ディストリビューションである openSUSE Tumbleweed LXQt を用いる．

4.1 試作したシステムの構成

hvt/solo5 には一般的な仮想環境上で標準的な入出力として利用されるコンソール出力，ネットワーク入出力，ブロック入出力を行うインタフェースが提供されているが，IoT デバイスで利用される入出力装置を制御するための機構となるインタフェースは提供されていない．

よって，図 2 に示した hvt/solo5 の図における，solo5 と Hardware(IoT デバイス上の GPIO) の間のインタフェースを実装した．これにより, solo5 から hvt を介して GPIO の制御をすることが可能となる．

以下は，インタフェースを実装するにあたって追加したプログラムである．

- tenders/hvt/hvt/hvt_module_switch.c
- bindings/hvt/switch.c
- tests/test_switch/test_switch.c

tenders/hvt/hvt_module_switch.c は，LED 操作に関するハイパーコールを定義しているプログラムである．solo5 と GPIO の間のインタフェースの中核となる役割を果たしている．また，hvt から solo5 に対して提供される GPIO インタフェースはすべてここで定義されている．

bindings/hvt/switch.c は，hvt_module_switch.c で定義したハイパーコールを実行するプログラムである．solo5 と hvt 間のインタフェースを補助する役割を果たしている．引数の数や返り値として返す値は，ここで設定する必要がある．

tests/hvt/test_switch.c は，Unikernels を実行するテストプログラムのうち，スイッチで LED の点灯・消灯を行うプログラムである．上記のハイパーコールを用いて solo5 から GPIO に対する制御を行う．このプログラムは Unikernels が形成されるときに読み込まれる．

4.2 solo5 と hvt の間のインタフェース

以下は，Unikernel に対して提供される GPIO インタフェースの詳細説明である．

hypercall_switchinit　GPIO ピン番号を引数として渡し，指定した GPIO ピンを初期化する．

hypercall_switchnalize　GPIO ピン番号を引数として渡し，指定した GPIO ピンを終了する．

hypercall_switchin　GPIO ピン番号を引数として渡し，入力動作するように設定する．

hypercall_switchout　GPIO ピン番号を引数として渡し，出力動作するように設定する．

hypercall_switchtest　GPIO ピン番号を引数として渡し，その GPIO ピンに入力があるかどうかを確認する．この時，GPIO ピン は入力動作するように設定されている必要がある．入力があった場合 1，なかった場合 0 として返り値を返す．

hypercall_switchwrite　GPIO ピン番号を引数 1, 出力デー

タを引数 2 として渡し、その GPIO ピンに出力を行う。この時，GPIO ピンは出力動作するように設定されている必要がある．出力を ON にする場合 1，OFF にする場合 0 を引数 2 として渡す．

なお，Raspberry Pi 3 では OS が動作しているため，直接的に内部レジスタの制御を行うことができない．よって，その代替として仮想ファイルシステムによるレジスタの制御が提供されている．そのため，これらのインタフェースはこれをもとに実装されている．

5 おわりに

本研究では，IoT デバイス上の軽量実行環境における汎用的な入出力制御機構の作成を目的とした実行環境の構築と実装を行った．IoT デバイスとして見立てた Raspberry Pi3 を用いた．軽量実行環境である hvt/solo5 を用意した．hvt/solo5 に GPIO の入出力を行うインタフェースを追加した．Raspberry Pi3 の汎用入出力 (GPIO) を用いて，スイッチの入力を検知して LED の点灯・消灯を行う Unikernel プログラムを実装し動作を確認した．

今回の研究により，IoT デバイスとして見立てた Raspberry Pi 3 上における，Unikernel である hvt/solo5 から GPIO を操作する環境を構築する手法が判明した．

今後の課題としては，Unikernel 向けの一般的な入出力装置制御インタフェースをどう設計するかという点，GPIO に限らない一般の入出力装置の制御をどうすべきかという点，また Unikernel ごとの違いをどう吸収すべきかという点が挙げられる．今回の研究は Raspberry Pi3，GPIO，hvt/solo5 といった部分で限定的であり，同様の考え方やインタフェース，手法を用いることで応用はできると考えられるが，そのまま実装することはできないため，拡張性を高める方法を検討していく必要がある．加えて，Unikernel 向けのインタフェースとして工夫すべき点，例えば Unikernel の特色を生かすためにプログラムサイズを可能な限り小さくしたり，そのうえで効率性を向上させたりするなど，考える必要がある点は多いと言える．

これについては方法の 1 つとして，準仮想化デバイスドライバのフレームワークである virtio を用いることが考えられる．virtio は windows や linux といったオーソドックスな OS に対応した仮想デバイスドライバは存在するが，Unikernel に対応したものは存在しない．また，CPU やメモリといった，汎用的なデバイスを動かす機能は既に存在するが，GPIO のような汎用的でないデバイスを動かす機能もない．よって，hvt/solo5 に virtio で GPIO を制御する仮想デバイスドライバを構築，IoT デバイス上に実装することで，オーバヘッドの削減やパフォーマンスの向上を見込むことができる．

参考文献

[1] Madhavapeddy Anil，Leonard Thomas，Skjegstad Magnus，Gazagnaire Thomas，Sheets David，Scott David，Mortier Richard，Chaudhry Amir，Singh Balraj，Ludlam Jon，Crowcroft Jon and Leslie Ian：“Jitsu: Just-In-Time Summoning of Unikernels”，the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI)，2015.

[2] Bob Duncan，et．al：“Enterprise IoT Security and Scalability: How Unikernels can Improve the Status Quo”，in Proceedings of 2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing，pp．292 − 297，Dec，2016.

[3] Anil Madhavapeddy , Richard Mortier1 , Charalampos Rotsos , David Scott , Balraj Singh , Thomas Gazagnaire , Steven Smith , Steven Hand and Jon Crowcroft：“Unikernels: Library Operating Systems for the Cloud”，in Proceedings of 18th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’13), pp．461472 , 2013.

[4] Unikernels- Rethinking Cloud Infrastructure，`https://unikernel. org/`.

[5] M．J．De Lucia：“A Survey on Security Isolation of Virtualization, Containers, and Unikernels”，US Army Research Laboratory Aberdeen Proving Ground United States，Tech．Rep.，2017.

[6] Maurantonio Caprolu，Roberto Di Pietro，Flavio Lombardi，Simone Raponi：“Edge Computing Perspectives:Architectures，Technologies，and Open Security Issues”, IEEE International Conference on Edge Computing (EDGE)，2019.

[7] Dan Williams and Ricardo Koller：“Unikernel monitors: extending minimalism outside of the box”，In 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)，USENIX Association，2016.