組込み制約下での複数 OS 実行環境の設計

出原 章雄[†]　東山 知彦[†]　落合 真一[†]

近年の組込み H/W の高性能化に伴い，従来は別 H/W プラットフォームで実現していた複数システムを，マルチコアの単一 H/W プラットフォームで実現可能になりつつある．こうした構成では，複数 OS の実行環境が必要となる．

一般的な組込み CPU では，H/W による仮想化支援機能を持たないため，複数 OS を実行する場合に，リアルタイム性能に制約が生じる．そこで，こうした制約下での複数 OS 実行環境の課題を整理し，組込み機器で必要となる機能，および，性能に基づき，組込み機器向けの複数 OS 実行環境を設計した．

既存の組込みハイパーバイザは，H/W リソース競合を S/W 処理にて解決する必要があり，これがオーバヘッドとなっていた．そこで，今回設計したハイパーバイザでは，組込み向けの特性を生かし，各 OS に対し，CPU，メモリ，および I/O を直接割り当てることにより，リソース競合の問題を発生させないようにし，この解決を図った．本稿では，この内容を示す．

A design of multiple OS platform under embedded system restrictions

Akio Idehara[†], Tomohiko Higashiyama[†] and Shinichi Ochiai[†]

Since recent embedded processors have very high performance, multiple applications which need different type of operating systems can be integrated with in single hardware system.

To support executing multiple operating systems, hypervisors are being developed even on embedded processors. But current hypervisors on embedded processors have issues on real-time performance, because few embedded processors have features such as Hardware-Assisted Virtualization (HAV) to reduce software overhead of multiple operating system executions. In this paper, we clarify the problems of hypervisors on embedded processors, and describe the design of a new hypervisor which meets embedded systems' requirements.

Existing embedded hypervisors need a software layer to control hardware resource contention. Considering embedded system characteristic, our hypervisor assigns CPU, memory, and I/O to each operating system to eliminate such a S/W layer and reduces overhead.

1. はじめに

近年の組込みハードウェア(以下，H/W)の高性能化に伴い，従来は複数 H/W プラットフォーム(以下，PF)で実現していたシステムが，マルチコアの単一 H/W PF で実現可能になりつつある．例えば，リアルタイム性の必要な制御処理を行うソフトウェア(以下，S/W)と，情報系処理を行う S/W といった，複数 S/W をマルチコアの単一 H/W PF で実現可能となっている．

こうしたシステムでは，複数 S/W プラットフォーム(複数 OS)を各コア上で動作させるために，複数 OS 実行環境(ハイパーバイザ)が必要となる．一般的な組込み向けの CPU では，H/W による仮想化支援機能(Hardware Assisted Virtualization，以降 HAV)を持たないため，複数 OS を実行する場合に，リアルタイム性能に制約が生じる．そこで，組込み機器で必要とする機能，および，性能に基づき，組込み機器向けの複数 OS 実行環境を設計した．本書では，この設計内容を示す．

† 三菱電機株式会社 情報技術総合研究所
Information Technology R&D Center, Mitsubishi Electric Corporation

ⓒ 2014 Information Processing Society of Japan

2. ハイパーバイザ要件

複数 OS の実行環境として，近年，ハイパーバイザが開発されている．ハイパーバイザの要件は以下の様にまとめられる[1].

(1) 等価性：CPU やメモリ等の H/W リソースを仮想化し，VM(Virtual Machine)として提供することにより，物理 H/W で動作するのと同等の処理を実行可能とする
(2) 効率性：仮想化された CPU に対する命令の大部分は実際の CPU によって実施可能とする
(3) リソース分離：VM を分離し，VM 上の OS が他の OS を意識しない．また，一度 OS に割り当てた H/W リソースを，再度ハイパーバイザが管理し，別 OS に再配分することができる

上で示された機能を実現するために，以下の仮想化が必要となる．

(A) CPU 仮想化((1)に対応)
ゲスト OS からの仮想化された CPU へのアクセスを，物理 CPU へのアクセスとして完全に模擬可能．これにより，OS からのシステム全体に対する変更要求を，VM 内部への変更に限定
(B) メモリ仮想化((1), (2)に対応)
ゲスト OS が扱う仮想化された物理アドレスを物理アドレスに変換して扱うことが可能．これにより，OS からの物理メモリに対する変更要求を，VM 内部の仮想的なメモリへの変更に限定
(C) I/O 仮想化((2), (3)に対応)
仮想化された I/O に対するアクセスを，物理的な I/O へのアクセスとして完全に模擬可能．これにより，OS からの I/O 要求を，VM 内部の仮想的な I/O への変更に限定

上で示した(A)から(C)を効率よく実現するためには HAV の対応が必要である．次章では HAV の機能を用いて，どのように仮想化を実現しているかを示す．

3. 仮想化のための HAV 機能

本節では，ハイパーバイザ上で仮想化を実現するための HAV 機能について示す．

3.1. ハイパーバイザモード追加(CPU 仮想化)

従来のユーザモード，特権モード以外に，ハイパーバイザモードを追加する．これにより，ハイパーバイザが指定する要因にて，OS から処理をトラップし，ハイパーバイザに処理を移すことが可能となる．本機能を用いることにより，仮想化が可能となる．

たとえば，OS が実行する CPU 命令の内，CPU の内部状態を変更する CPU 命令(x86 アーキテクチャのセンシティブ命令等)をハイパーバイザがトラップすることにより，CPU の内部状態を適切に変更した上で，処理を OS に返すことが可能となる．

これにより，他のゲスト OS の影響を受けることなく，OS が動作可能となる．

3.2. ゲスト OS 用ページテーブル追加(メモリ仮想化)

ゲスト OS に対するメモリ空間について，仮想化に伴う処理を高速化する目的で，ゲスト OS 用にページテーブルを用意する．ハイパーバイザは，各ゲスト OS の切り替えのタイミングで，ゲスト OS 用のページテーブルに切り替える．これにより，ハイパーバイザが介在することなく，ゲスト OS が意識している物理アドレス(仮想物理アドレス)と実際の物理アドレスを変換可能となる．

なお，HAV がハイパーバイザモードに対応していれば，ページテーブルへのアクセスをトラップすることで，S/W により同等の処理が可能である．しかしハイパーバイザの高速化のために，本機能の対応が進んでいる．

3.3. 割込みコントローラ仮想化，IOMMU 追加(I/O 仮想化)

I/O 仮想化として，割込みコントローラ仮想化と IOMMU がある．

割込みコントローラ仮想化では，例えば，ハイパーバイザが決めた設定に従い，HAV は実際の割り込みに対する仮想化された割込みを生成する．また，ゲスト OS が受け付けた割込みの応答を返す場合には，仮想化された割込みコントローラに対して応答を返す．HAV は仮想化された割込みに対する応答を，実際の割込みに対する応答に変換して応答を返す．これにより，仮想的な割込みと実際の割り込みを分離可能となるため，仮想化が可能となる．

IOMMU は DMA 転送を行うアドレスとして，ハイパーバイザが管理する実際の物理アドレスではなく，ゲスト OS 用ページテーブルで設定されていた仮想物理アドレスを指定可能にする．これにより，ハイパーバイザにて DMA アドレスの管理を実施する必要がなくなるため，高速に I/O 制御が可能となる．

なお，I/O 仮想化は，S/W により I/O を完全にエミュレーションすることでも実現可能であるが，ハイパーバイザが複雑になること，また，処理のオーバヘッドが大きいことから，HAV の対応が進んでいる．

4. 組込み向けハイパーバイザのターゲット構成

組込み H/W 上にて，複数 OS のシステムを構築する

ⓒ 2014 Information Processing Society of Japan

場合，一般的には，制御処理などのリアルタイム性が必要な S/W を動作させるためのリアルタイム OS(以下 RTOS)と，GUI などの情報系処理を行う汎用 OS との複数 OS で構成されている．

RTOS 側は，センサ情報を定周期で取得するなど，定周期性とレイテンシが重要となる．また，従来からの流用 S/W が多く搭載されており，必要となる I/O も固定化されていることが多い．汎用 OS 側は，マルチメディア処理や Web ブラウザなど，UI を提供する OS であり，OS の頻繁なアップデートにも対応する必要がある[2].

近年，組込み向け CPU に対しても，HAV がサポートされ，これに対応した実装も行われている[3][4][5]．しかし，価格面の制限などにより，そうした CPU が選択されないことも多い．このため，H/W による仮想化支援機能を持たない組込み向け CPU にも，複数 OS の動作を実現するハイパーバイザを提供する必要がある．

以上をまとめると，ターゲット構成は以下となる．

[組込み向けハイパーバイザのターゲット構成]

(1) マルチコアの単一 H/W
マルチコアの単一 H/W を対象とする
(2) RTOS と汎用 OS の複数 OS 構成
RTOS 側はリアルタイム応答性が必要
汎用 OS 側は頻繁なアップデートの可能性あり
(3) HAV 未サポート

5. 課題

組込み向けハイパーバイザについても，仮想化により，OS の修正個所を最小化し，開発工数を削減することが可能となる．しかしながら，前章で示したように，組込み向け CPU は HAV 未サポートのため，HAV を用いた仮想化を行うことは難しい．

そこで，既存の組込み向け複数 OS 実行方式[6][7][8]を参考に，本ターゲット構成に対し，S/W による仮想化を実現する場合の課題を検討する．

5.1. CPU アクセス限定化

OS を無修正で動作させる場合，ハイパーバイザは，システム全体に影響する処理をトラップし，OS の処理を VM 内に限定する必要がある．HAV がハイパーバイザモードをサポートする場合，システム全体に影響する処理をトラップすることにより，OS の処理を VM 内に限定することが可能となる．

しかし，ハイパーバイザモードがない場合，OS の処理をトラップすることができない．この場合，次の二つの手法が利用される．

(1) バイナリトランスレーション
(2) 準仮想化

5.1.1. バイナリトランスレーション

バイナリトランスレーションは，ゲスト OS 実行時にシステムの状態を変更する CPU 命令を書き換えて，影響のない CPU 命令に置き換える手法である．バイナリトランスレーションは OS 実行時に CPU 命令の調査・変更を行うため，オーバヘッドが大きい．このため，組込み向けには適さない．

5.1.2. 準仮想化

準仮想化は，あらかじめ OS のソースコード中のシステムの状態を変更する CPU 命令を置き換える手法である．例えば，組込み向けでも，準仮想化を用いた例がある[10][11].

こうした命令を置き換える方式の例として，ソフトウェア割込み命令を使用する方法が一般的である．しかし，この場合，ソフトウェア割込み命令の実行，および，ハイパーバイザのコンテキスト退避・復帰によるオーバヘッドにより，リアルタイム応答性の悪化が考えられる．また，OS 内のソフトウェア割込みハンドラの変更が必要となり，OS 修正に対する開発工数も増大すると考えられる．こうしたことから，従来とは異なる，新たな組込み向けの準仮想化による対応を検討する必要がある．

5.2. メモリアクセス限定化

OS を無修正で動作させる場合，ハイパーバイザは物理メモリに影響する処理を，VM 内に限定することが必要となる．

HAV に対応している場合，OS ごとに仮想的な物理メモリ空間を用意することでこうした対応が可能となる(ゲスト OS 用ページテーブル追加)．HAV がゲスト OS 用ページテーブルをサポートしない場合でも，ハイパーバイザモードをサポートする場合，ゲスト OS のページテーブルへのアクセスをトラップすることで，該当する機能を実現可能である．

今回，HAV をサポートしない CPU を対象とするため，ゲスト OS 用ページテーブルはない．この場合，組込み向けの仮想化技術では，前節で示した準仮想化の技術を用いて，OS からのページテーブルアクセス処理をトラップし，OS 間で異なる MMU の設定とすることが可能となる．しかし，前節の議論と同様，リアルタイム応答性の悪化，および，OS 修正に対する開発工数の増大から，現行の準仮想化を採用することはできない．このことから，新たな組込み向けの準仮想化による対応を検討する必要がある．

5.3. I/O 直接実行

HAV が割込みコントローラの仮想化に対応しない場合であっても，ハイパーバイザモードをサポートする

ⓒ 2014 Information Processing Society of Japan

場合，割込みを全てハイパーバイザでトラップし，ゲストOS に割込みを再割り当てすることで実現可能である．

組込み向けの場合は，割込みの仮想化は実施しせず，I/O を直接実行可能としている．また，ハイパーバイザによる I/O デバイスの完全仮想化も実施していない．

今回我々も，組込み向けに I/O の扱いを検討する必要がある．

5.4. 課題まとめ

本章の議論から，既存のバイナリトランスレーションや準仮想化を用いたハイパーバイザには，リアルタイム応答性の課題があることが判断した．そこで，リアルタイム応答性を重視した組込み向けハイパーバイザを実現する必要がある．

具体的には，OS のソースコード変更を許容しつつ，以下の機能を S/W にて実現することで，リアルタイム応答性の高いハイパーバイザを開発する．

(1) CPU アクセス限定化：OS からシステム全体への変更要求を VM に限定
(2) メモリアクセス限定化：メモリ仮想化：OS からの物理アドレスへのアクセスを VM に限定
(3) I/O 直接実行：I/O の直接実行実現

6. 設計

組込み向けハイパーバイザの実現に向けて，前節で示した各課題に対する設計を示す．

6.1. CPU アクセス限定化

既存のハイパーバイザによる CPU 仮想化では，システムの状態を変更する処理をトラップすることで，システム全体に影響を与えないようにしていた．

今回の設計では，各 OS に対し，物理的な CPU を直接割り当てることにより，CPU アクセスを限定化することとした．この理由について，システムの状態を変更する CPU 命令は自 CPU の内部状態を変更するのみであり，別 CPU の状態に影響しないためである(図 1)．

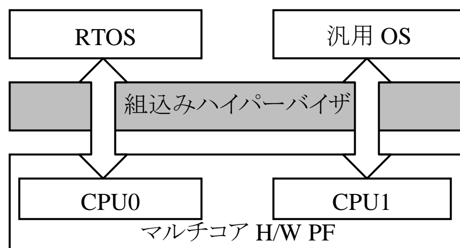

図 1 CPU アクセス限定化

このように，各 OS に対し各 CPU を一対一に割り当てるために，OS の起動処理を設計する必要があった．次節ではこの内容を示す．

6.1.1. OS 起動処理

OS 起動処理の修正について，一般的なマルチコアでは，H/W リセット後に Boot Strap Processor(以下 BSP)が最初に起動し，BSP により Application Processor(以下 AP)が起動する．そのため，以下の手順により OS を起動することとした(図 2)．

(1) BSP 上でブートローダを起動
(2) BSP 上のブートローダが RTOS を起動
(3) BSP 上の RTOS 起動後，BSP が AP のリセット解除
(4) AP 上で汎用 OS を起動

ここで，BSP 側は RTOS を動作させる方針とする．これは，通常，RTOS は制御用の S/W 等，固定的な S/W のみが動作し，ユーザが作成したアプリ等が動作することはない．このことから，RTOS 側の S/W 環境は十分に試験され，信頼性が高いと考えられるためである．

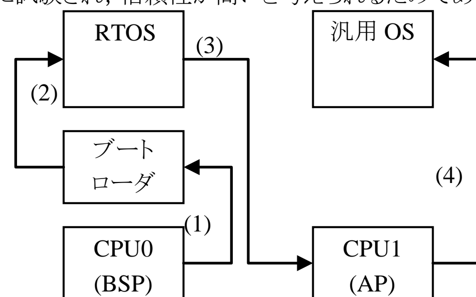

図 2 OS 起動処理

6.2. メモリアクセス限定化

今回の設計では，6.1 節で示した通り，各 OS に対し各 CPU を割り当てることとした．マルチコアの場合，各 CPU に対しページテーブルを個別に設定可能のため，個々の OS のメモリアクセスを限定することができる．

ただし，ハイパーバイザの利点の一つである，VM 間の干渉を防ぐという観点では，OS 分のページテーブルを分割するだけでは不十分である．すなわち，ページテーブルを変更することにより，ある OS から別の OS のメモリ領域へアクセス可能となってしまうからである．そこで，RTOS と汎用 OS で以下の様な対応を実施することとした．

6.2.1. RTOS 側

RTOS 側については，基本的には物理アドレスと仮想アドレスが一対一となるようにする．そのため，RTOS 初期化時に，RTOS 側 RAM・I/O のみを登録するように PTE を構成し，MMU を有効化する．これにより，汎用

ⓒ 2014 Information Processing Society of Japan

OS が使用する RAM・I/O が登録できないため, RTOS から汎用 OS 側の RAM・I/O のアクセスを防ぐ(図 3 上).

6.2.2. 汎用 OS 側

汎用 OS の MMU 処理を変更し, ページテーブルへのアクセスを制限することとした. 具体的な変更箇所としては. 仮想アドレスと物理アドレスのマッピング登録処理を修正し, RTOS 側 RAM・I/O へのメモリアクセスの場合には, ページテーブルに登録しないようにする. ただし, ページテーブル登録の際に毎回確認するのはオーバヘッドが大きいので, アプリおよびドライバからのメモリマップ登録要求に対するアクセスチェックのみに限定して, メモリチェックを行うこととする(図 3 下).

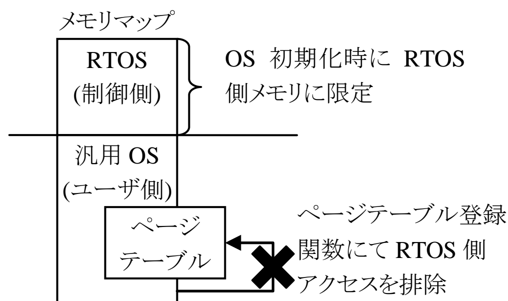

図 3 メモリアクセス限定化

6.3. I/O 直接実行

I/O に関しては, 各 OS に対して個別に I/O を割り当てることにより, 各 OS が I/O を直接実行可能とする. この理由として, 課題でも述べたとおり, S/W による I/O 完全仮想化はオーバヘッドが大きく, 組込み向けには適さないためである. I/O 割り当てについては, 近年, 組込み向けに適用が進んでいる DeviceTree[11]対応を行うこととした. DeviceTree は例えば, 搭載メモリ量や, 搭載 I/O およびメモリマップト I/O アドレスや IRQ 番号といった I/O 情報を保持可能である. このように, DeviceTree の情報を OS 起動時に読み出すことで, 各 OS のバイナリを修正することなく, I/O を割り当てることが可能となる.

ただし, 各 OS に対して個別に割り当てることのできない I/O として以下が存在する可能性がある. それぞれについて詳細を示す.

(1) 割込みコントローラ
(2) L2 キャッシュコントローラ

6.3.1. 割込みコントローラ

最近のマルチコアに対応した割込みコントローラは, デバイスに対し応答を返す場合, CPU 毎のロックは不要となっている. たとえば, 応答レジスタに対し, ビットを書込みするのみで, デバイスに応答を返すことが可能となっている.

しかし, 割込みの優先度や, 割込みのターゲット CPU 設定など, 初期化時にのみ使用されるレジスタについては, CPU 間でロックが必要となっている.

同一 OS 内であれば CPU 間でスピンロックなどのロックを用意すればよい. しかし, 今回は異なる OS 間でロックが必要となる. すなわち, OS 間でロック処理が必要となる. 今回の構成では, 先に起動する RTOS 側にて, 汎用 OS 側の設定を実施しておき, 汎用 OS 側はこうしたロックが必要なコントローラ設定を実施しないこととした. これにより OS 間ロック処理が不要となる(図 4).

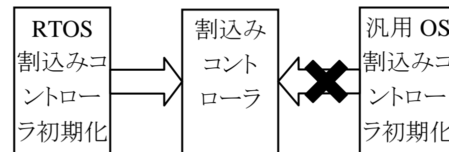

図 4 割込みコントローラ初期化

6.3.2. L2 キャッシュ

今回は RTOS 側と汎用 OS 側で, ともに DMA を使用しているため, RTOS 側と汎用 OS 側でともにキャッシュとメモリのコヒーレンシを取る必要がある. しかしながら, 今回のターゲットでは L2 キャッシュコントローラは一つしかなく, L2 キャッシュの操作を行う場合には, 本コントローラを各 OS から同時に使用する必要がある.

そこで, OS 間で排他制御を行う処理を実施することとした. 具体的には, OS 間の共有メモリ内に, スピンロック用の変数領域を設け, 本変数を用いて, スピンロックを行うこととした.

7. 評価と考察

今回設計した組込みハイパーバイザが章で示したハイパーバイザの要件(1)から(3)に準じているかという観点で, 評価と考察を行う.

7.1. 等価性

各 OS に対し, 物理的な CPU を割り当てたため, CPU 命令を直接実行可能である. ただし, これを実現するために, OS の起動処理, および, 再起動処理を各 OS 向けに修正する必要があった.

7.2. 効率性

CPU 命令は, システムに影響のある CPU 命令を含めて, 全て実際の CPU 上で実現可能である. このことから, 効率性は高いと考えられる.

ただし, OS に対し CPU を一対一で割り当てたことから, OS の性能は各 OS に割り当てた CPU の個数に依

存してしまう. これは, 今回設計した組込み向けハイパーバイザの制限事項である.

7.3. リソース分離

各 OS に対し, CPU を割り当てることにより, CPU は分離可能であった.

メモリについては, OS に修正を加えることにより, 分離を実施した. ただし, OS を介さず, 直接ページテーブルを破壊された場合にはメモリ分離は不可能となる. これは今回設計した組込み向けハイパーバイザの制限事項である.

I/O については, 各 OS に対し, 個別に I/O を割り当てることとした. ただし, OS 間で共有される可能性のある I/O として, 割込みコントローラおよび L2 キャッシュコントローラが挙げられた. 割込みコントローラについては, RTOS のみが初期化を実施し, 汎用 OS 側には処理をさせないこととした. L2 キャッシュコントローラは OS 間で排他制御を行うこととした. これにより, ただし, これに伴う性能劣化の評価は今後の課題である.

なお, ハイパーバイザ要件のリソース分離で挙げられているリソース再分配については, 今回の組込みハイパーバイザでは実現できない. このことから, [1]で示されるハイパーバイザの要件からは外れる結果となった. しかし, 組込み機器ではリソースの再配分を要求されることはないため, 本対応は不要と考える.

8. おわりに

今回, HAV を持たないマルチコア H/W PF 向けの複数 OS 実行環境の設計を行った. 既存の組込みハイパーバイザは, H/W リソース競合を S/W 処理にて解決する必要があり, これがオーバヘッドとなっていた. そこで, 今回設計したハイパーバイザでは, 組込み向けの特性を生かし, 各 OS に対し, CPU, メモリ, および I/O を直接割り当てることにより, リソース競合の問題を発生させないようにし, この解決を図った. 具体的には, 以下の課題を解決する組込み向けハイパーバイザの開発を行った.

(1) CPU アクセス限定化
(2) メモリアクセス限定化
(3) I/O 直接実行

本ハイパーバイザを用いることで, HAV を持たないマルチコア H/W PF を使用した場合であっても, リアルタイム応答性を満足しつつ, 複数 OS 実行環境を容易に実現することが可能となる. 今後は, 今回設計した組込みハイパーバイザを実ターゲットに実装し, 性能評価に取り組む予定である.

参考文献

[1] Gerald J. Popek, Robert P. Goldberg: Formal Requirements for Virtualizable Third Generation Architectures, Communications of the ACM Volume 17 Number 7 pp.412-421(1974)
[2] 出原章雄, 山本整, 東山知彦, 落合真一:組込みCPU向け高信頼基盤ソフトウェアの開発, 情報処理学会第 75 回全国大会(2013)
[3] 岡部 亮: QorIQ P4080 プロセッサ向けハイパーバイザの設計, 電子情報通信学会 2013 年総合大会(2013)
[4] A. Kivity, Y. kamay, D. Laor, U. Lublin, and A. Liguori: kvm: the Linux(R) Virtual Machine Monitor, Otawa Linux Symposium 2007, pp.225 - 230 (2007)
[5] 東山 知彦, 落合 真一: 組込み向け仮想化技術の評価, 組込み技術とネットワークに関するワークショップ ETNET2014(2014)
[6] 金田一勉: 第 17 章 Linux on ITRON - ハイブリッド構造の実装,インタフェース増刊 TECHI vol.16,CQ 出版社(2003).
[7] W. Kanda, Y, Yumura, Y, Kinebuchi, K, Makijima, T, Nakajima: SPUMONE: Lightweight CPU Virtualization Layer for Embedded Systems, 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing(2008)
[8] 太田 貴也, 大橋 孝輔, ダニエル サンゴリン, 本田 晋也, 高田 広章, 堀田 孝一, 大山田 光夫: SafeG-MP: マルチコア対応の組込み向け仮想化環境, 電子情報通信学会論文誌 D Vol. J96-D No10 pp.2163-2183(2013)
[9] OKL4: http://www.ok-labs.com/products/okl4-microvisor
[10] PikeOS: http://www.sysgo.com/products/pikeos-rtos-and-virtualization-concept/
[11] DeviceTree: http://www.devicetree.org/Main_Page

Linux は, Linus Torvalds 氏の日本およびその他の国における登録商標または商標です.
その他の商標は, 日本およびその他の国における登録商標または商標です.