2007－EMB－6 (2)
2007／12／4

uClinux環境下でのOSレベル仮想化の実現

川島 潤† 中西 透† 舩曳信生†

本稿では，OSレベルの仮想化技術であるLinux-VServerをMMU(メモリ管理機構)を持たないプロセッサ環境でも動作するuClinuxに移植し評価を行う．このuClinux-VServerの仮想環境を用いることで，メモリ制約の厳しい組込み用途の実運用機器上にも，実運用環境への影響を制限したテスト環境を構築できる．提案法を実装し評価した結果，VServerの移植に伴うカーネルサブシステムのオーバヘッドは数%以内であり，実際のアプリケーションは移植前と変わらない性能を示した．さらに，仮想環境のCPU使用率を制約できることを確認し，実運用環境への影響を制限したテスト環境を構築可能であることが明らかとなった．

An Implementation of uClinux-VServer for Embedded Systems

Jun Kawashima,† Toru Nakanishi† and Nobuo Funabiki†

In this paper, we report an implementation of *uClinux-VServer* and its evaluations. This uClinux-VServer is actually implemented by porting Linux-VServer, an OS-level virtualization system, to uClinux. uClinux can run on a CPU with no MMU (Memory Management Unit), and thus it is suitable for embedded systems. VServer offers a virtual operating platform to a machine so that application programs can be tested there while restricting the interference to the host system on the machine. The evaluation results of the implemented uClinux-VServer show the porting overhead is very small, and the performance of applications on it has no difference from the performance on the conventional uClinux.

1. はじめに

近年の組込みシステムの大規模化，短納期化により，そのソフトウェア品質の確保が問題となっている．ソフトウェア品質の確保には，評価，検証，テスト工程が重要である．

近年ソフトウェアの短納期化，マルチプラットフォーム化のために評価，検証，テストをコンピュータ上の仮想環境で行うケースが増加している．しかしながら，最終的には実機・実動作環境でのテストが必要となる．この時実動作環境の制約から，実行環境を停止することができない場合には，実行環境に影響を与えることなくテストを行う必要がある．そのような環境では，VPS(Virtual Private Server)として知られるOSレベルの仮想化技術が有効と考えられる．プロセス，ファイルシステム，ネットワーク環境をVPS内に独立して構築できるOSレベルの仮想化技術は，他の仮想化技術と比較してリソース消費が低く，組込み機器に適している．

† 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University, E-mail:kawasima@sec.cne.okayama-u.ac.jp,{nakanisi,funabiki}@cne.okayama-u.ac.jp

本稿では，Memory Management Unit(MMU)を持たない，安価で低性能なプロセッサ上でのOSレベル仮想環境の実装および性能評価を行う．ここでMMUを持たない安価なプロセッサを用いる理由は，昨今のプロセッサの性能向上により，組込み向けプロセッサといえどもMMUを有する高性能なものは一昔前のPC向けプロセッサと遜色のない性能を持っており，OSレベル仮想環境が問題なく動作可能と判断したためである．一方MMUを持たないプロセッサは，組込みシステムとして広く普及している一方で，後述する様々な制約があるために，仮想環境の動作検証が必要となる．

本稿では，組込みシステムのOSとして，MMUを持たないプロセッサ向けにLinuxのメモリ管理機構を改良したuClinux[1]，OSレベル仮想化実装としてLinux-VServer[2]を利用する．また実装環境として，ARM926EJ-SをCPUコアとするArmadillo-300[7]を使用する．本研究では，まずArmadillo-300が提供するLinux kernel-2.6.12.5-at2を元に，実装時点で最新のバージョンであったkernel-2.6.22をArmadillo-300に移植している．その後，同カーネルにuClinuxパッチを当てたものをArmadillo-300上に移植し，最後にLinux-VServerをuClinux向けに移植している．ここでARM926EJ-SはMMUを搭載しているが，本

稿では MMU を無効にして動作させている．

以下，2 章で uClinux，3 章で Linux-VServer について述べる．4 章では Linux-VServer の uClinux へのポーティングについて述べる．5 章にてポーティングを行った Linux-VServer の評価結果を示し，最後に 6 章で本稿のまとめを行う．

2. uClinux

本章では，uClinux の特徴と，Linux との実装上の相違点について述べる．本稿で使用したカーネルは，vanilla kernel-2.6.22 に uClinux 化パッチである linux-2.6.22-uc0-big.patch を適用したものである．

2.1 uClinux の特徴

uClinux とは，MMU(Memory Management Unit) を持たない安価な組込みシステム向けプロセッサへ移植された Linux である．

uClinux は以下の特徴を持つ．

- Linux の全機能を使用可能 (プリエンプティブカーネル，デバイスドライバ，O(1) スケジューリング等)
- キャッシュと TLB のフラッシュが不要なため，コンテキストスイッチが高速
- uClinux 用 FLAT バイナリは，RAM 消費量を抑える XIP(eXecution In Place) に対応
- メモリ保護がないため，ユーザ空間プロセスの不正ポインタ参照によるカーネル空間への干渉が可能
- 仮想アドレス空間を扱えない
- ページテーブルを扱えないため，メモリ使用効率を向上させるデマンドページングと COW(Copy On Write) が利用できない

2.2 Linux との実装上の相違点

uClinux では，主としてメモリ管理機構に関して，Linux と実装が異なっている．

Linux との相違点を明らかにするために，以下に前節で挙げた各特徴に対応する uClinux での実装について述べる．

- XIP に対応した FLAT バイナリ
  XIP とは，実行バイナリファイルの読み込み専用セグメント (.text，.rodata セクションから成る) を ROM に置き，RAM 上に領域を確保しないで実行させる方式である．これにより RAM の消費量を抑えることができる．但し書き込み可能セグメント (.data，.got，.bss，.stack セクション) は RAM 上に確保される．
  uClinux では仮想アドレス空間を持たないため，各プロセスは実行バイナリファイルを異なるアドレスにロードしなければならない．よって，プログラムの実行時に実行バイナリファイルの再配置が必要となる．再配置では，バイナリファイル中で使用されるアドレスを実行時にロードされたアドレスに対応して変更する．
  ここで，実行時にセグメントの変更を必要とする再配置と，変更不可能な ROM 上へセグメントを配置する XIP を同時に実現するためには，読み込み専用セグメントを PIC(Position Independent Code) とする必要がある．PIC では再配置時に変更される情報を GOT(Global Offset Table) に格納し，再配置時には読み込み専用セグメントの代りに GOT を変更する．GOT は RAM 上の書き込み可能セグメントに含まれる．読み込み専用セグメントは，アドレス参照時に GOT を参照することで，再配置されたアドレスを知ることが可能となる．
  上記の実現には，実行バイナリ形式のサポートが必要である．uClinux で使用される FLAT バイナリ形式は，Linux で使用される ELF バイナリ形式を単純化，軽量にしたものである．FLAT バイナリファイルは，ELF バイナリファイルのセクションを並べ変え，上述の GOT，再配置情報を再生成することで作成される．
- メモリ保護がない
  ARM プロセッサの MMU は，ページテーブルエントリ中に特権レベルを表すドメインを保持している．Linux ではカーネルドメイン DOMAIN_KERNEL とユーザドメイン DOMAIN_USER が定義される．ページ参照時，MMU により CP15:C3 レジスタが示す現行のドメインと参照先のドメインが比較され，DOMAIN_USER 時は DOMAIN_KERNEL ページへの参照は拒否され，例外が発生する．これにより，ユーザ空間プロセスの不正ポインタ参照からカーネル空間が保護される．MMU が無効の場合，この保護は機能しない．[6)]
- 仮想アドレス空間を扱えない
  uClinux では，物理メモリアドレスと仮想メモリアドレスが等しい．また，前述したように全プロセスは同じメモリ空間を共有する．これに対応するため，uClinux ではメモリを管理する構造体に変更が加えられている．ユーザ空間プロセスは確保された複数のメモリ領域からなるメモリ空間を利用する．各領域は vm_area_struct 構造体 vma で管理される．Linux では vma は仮想アドレスを管理するためプロセス毎に独立している．プロセスが領域を確保，解放する時に，vma は 2 種類の方法で探索される．アドレス昇順に繋がった単純リストと red-black ツリーである．単純リストの先頭と red-black ツリーのルートノードはプロセス構造体 task_struct メンバの mm 構造体に存在する．uClinux では vma が管理する領域は物理アドレスであるため，各 vma はシステムグ

ローバルとなる．
ここで，プロセスの終了時に vma を解放する処理を考える．Linux では，単純リストを用いてプロセスが管理する vma を全て探索することが可能であるが，uClinux では vma はシステムグローバルであるため，プロセス毎の探索が行えない．そこで，uClinux ではプロセス毎の vma を管理する vm_list 構造体を導入している．vm_list 構造体も前述の mm 構造体からたどることができ，プロセスに関連付けられている．

- デマンドページが利用できない
  MMU が無効の場合，変換フォルトに起因する例外 (ページフォルト) が発生しない．ページフォルトは，MMU 有効時にはページ参照時に参照するページテーブルエントリが無効である際に発生する．ページテーブルエントリが無効とは，仮想アドレスに対応する物理アドレスが確保されていないことを意味する．
  Linux では，ユーザ空間プロセスがメモリを要求する際，ページ確保を実際に利用されるまで遅延させる (デマンドページング)．対象ページへのアクセスの検知にページフォルトを利用するため，uClinux ではデマンドページングが利用できない．uClinux ではメモリ要求時に kmalloc() 関数を用いて静的にメモリ領域を割り当て，動的な領域拡大には対応しない．静的なメモリ確保はフラグメンテーションを起こし，メモリ使用効率が低下する．
  Linux では，プロセスのスタック領域もデマンドページングを利用するが，uClinux では，プログラム起動時に静的に確保する．FLAT バイナリのヘッダには，確保すべきスタックサイズが格納されている．
- COW が利用できない
  MMU が無効の場合，アクセス許可フォルトに起因する例外が発生しない．アクセス許可フォルトは，書き込みが禁止されているページに対しての変更時などに発生する．
  Linux では，故意にページテーブルエントリを書き込み禁止にすることで，対象ページへの変更を検知し COW を行う．COW は fork システムコールに伴う親プロセスメモリ空間の子プロセスへの複製を，メモリ空間が変更されるまで遅延させる際などに利用される．参照されるだけであれば，複製を行う必要はないという考えに基づく．
  uClinux ではアクセス許可フォルトが発生しないため，COW が利用できない．Linux の fork は COW を使用するため，uClinux 上のアプリケーションは COW を使用しない vfork システムコールを代りに用いる必要がある．

3. Linux-VServer

本章では，Linux-VServer の特徴とその仮想化の実装方法について述べる．本稿で使用したバージョンは，vanilla kernel-2.6.22.2 向けのパッチである patch-2.6.22.2-vs2.2.0.3.diff[4] を vanilla kernel-2.6.22 に対し適用したものである．

3.1 VServer の特徴

VServer は Linux での OS レベル仮想化実装の一つで，コンテキストと呼ばれる VPS 毎に，仮想 Linux カーネル環境を提供する．

主流である完全仮想化，準仮想化技術の VMware や Xen がハードウェアの仮想化を行うのに対し，OS レベル仮想化はカーネルの仮想化を行う．ゲスト OS とホスト OS が同じカーネルを用いるため，使用メモリ量が少ない．また，ハードウェアの仮想化に比べカーネルの仮想化に必要なリソースはごくわずかである．しかし，VMware や Xen と異なりカーネル自体に仮想化のための変更が必要である．

VServer には以下の特徴がある．

- コンテキスト毎に仮想 OS が独立
  コンテキスト毎に独立したプロセス，ファイルシステム，ネットワーク，システム情報等を持ち，他のコンテキストからは参照できない．
- コンテキスト毎に資源を制限可能
  プロセス，ソケット，ファイル，メモリ等の各カーネルオブジェクトに対し，コンテキスト毎にリミット値を設定できる．

3.2 仮想化実装手法

本節では，VServer の特徴であるコンテキスト毎にカーネルオブジェクトを独立させる実装手法について述べる．

カーネルオブジェクトを独立させるためには，各オブジェクトに対し所属するコンテキスト識別子の付与と参照時の識別が必要である．以下に具体例を述べる．また，VServer がコンテキストを管理する際に使用する構造体は，struct vx_info とする．

- プロセスの独立
  プロセスは task_struct 構造体で管理される．VServer では，task_struct 構造体にコンテキスト識別子 unsigned long xid，struct vx_info *vx_info メンバを追加している．
  プロセスの生成時は，do_fork() 関数経由で copy_process() 関数が実行される．ここで前述の xid,vx_info が初期化される．子プロセスは親プロセスの所属するコンテキストを引き継ぐ．また，後述するユーザ空間ユーティリティ Util-VServer の vcontext コマンドにより，ホスト OS から任意のプログラムを任意のコンテキスト上で実行させることができる．

プロセスが他プロセス情報を参照する際は，/proc ファイルシステムを経由する必要がある．/proc の read 時に実行される proc_pid_readdir() 関数では，read を行ったプロセスの xid と/proc で表示するプロセスの xid を比較し，一致する場合に表示を行う．このため，他コンテキストからのプロセス情報の参照を防ぐことができる．
ここでプロセスが他プロセスの存在を知る方法として，シグナルを利用することが考えられる．これは，/proc ファイルシステム内情報を隠蔽するだけでは防ぐことができない．VServer ではこれに対し，シグナル送信時に送信元プロセスと送信先プロセスが同コンテキストに属しているか否かを確認し，異なる場合 ESRCH エラーを返すことで他コンテキストのプロセスから独立させている．

- ファイルシステムの独立
VServer では，chroot barrier 環境を用いて仮想的なルートディレクトリ環境を生成し，その中でゲスト OS を動作させている．また，異なる chroot barrier 環境同士は，互いのファイルシステムにアクセスできないため，ファイルシステムの独立を実現できる．
通常の chroot ではコマンドマニュアルに記載されている通り，作業ディレクトリが変更されない，オープンファイルディスクリプタがクローズされないことを悪用することで，chroot ツリーの外にあるファイルにアクセス可能である．chroot barrier とは，コンテキスト起動時に chroot ツリーの親ディレクトリの inode に対し，Util-VServer が提供する setattr コマンドを用いて S_BARRIER フラグを立てることで，ツリー外へのアクセスを拒否する仕組みである．VServer では，ファイルシステムの権限をチェックする fs/namei.c:permission() 関数にて S_BARRIER フラグのついたファイル，ディレクトリへのいかなるアクセスも EACCES エラーを返すことで拒否する．ツリー外へのアクセス時は必ず S_BARRIER をつけた親ディレクトリを経由するため，ツリー外へのアクセスを拒否することが可能となる．ただし，ホスト OS からのアクセスのみは例外としている．

4. Linux-VServer の uClinux への移植

VServer の主要機能であるコンテキストの提供については，VServer のアーキテクチャ非依存の実装方針により uClinux への移植のために変更する必要はない．
しかし，chroot barrier，資源制約など，カーネルモジュール自身に変更を加える必要のある機能については，uClinux を含む組込み固有のモジュールには変更が施されていない，あるいは不十分なため，実装を行う必要がある．以下にそれらの詳細を述べる．

また，VServer の制御を行うユーザ空間ユーティリティ Util-VServer[5] についても uClinux 上への移植が必要であった．

4.1 chroot barrier について

chroot barrier は，テスト環境のアプリケーションによる実行環境のファイルシステムへのアクセスを制限するために重要である．
chroot barrier は，ファイルシステムモジュールに新たなフラグを追加することで有効になる．uClinux で利用される romfs，cramfs，jffs2 ファイルシステムモジュールでは，そのフラグが追加されていない．そこで本稿では，romfs，cramfs について chroot barrier を有効にするフラグを追加した．jffs2 の変更については今後の課題とする．

4.2 メモリ資源制約について

VServer がテスト環境を提供する場合，実行環境であるホスト OS のメモリ資源をテスト環境ゲスト OS から保護する必要がある．これはメモリ資源に乏しい組込み機器では特に重要である．
Linux のメモリ機構は消費されるメモリ資源をカウントし，ユーザに提供する．VServer ではその値を用いて，コンテキスト毎にプロセスが使用可能なメモリ資源を制限する．カウントされるメモリ資源は，デマンドページング時などの実際にメモリが確保される際にカウントされる Resident Set Size(RSS) と，ユーザからのメモリ要求時に vma のサイズ変更に合わせてカウントされる Virtual SiZe(VSZ) である．前者は mm 構造体の_file_rss，_anon_rss メンバの合計で表され，後者は mm 構造体の total_vm メンバに格納される．
uClinux では RSS は使用されていない．これは，前述したようにデマンドページングが利用できずメモリを静的に確保するため，RSS と vma のサイズが等しくなるためと考えられる．
ここで VServer が Linux に追加するメモリ資源制約はほとんどが RSS を利用したものであり，VSZ を用いる制約も uClinux のメモリ管理に対しては適用されていない．
そこで本稿では uClinux のメモリ管理に変更を加え，VSZ を用いてコンテキスト毎にプロセスが使用可能なメモリ資源制限を実装した．具体的には，uClinux がユーザからの要求を受け静的にメモリを確保する do_mmap_pgoff() 関数に，VServer の資源利用の可否を判断する関数 vx_vmpages_avail() を追加した．
しかしながらこの方法では，XIP 時などの実際には RAM が消費されないデバイスマッピング領域のサイズも VSZ にカウントされているという問題がある．メモリ資源に乏しい組込み機器では資源制約のリミット値を厳密に定義する必要があるため，これは問題となる．
この問題の単純な解決策として，XIP 用の領域を確

表 1 測定環境

Processor	NS9750
CPU core	ARM926EJ-S
CPU core clock	200MHz
BUS clock	100MHz
SDRAM	64MB
FLASH	8MB
OS	uClinux-2.6.22-vs2.2.0.3
Linux kernel	vanilla kernel-2.6.22
uClinux patch	linux-2.6.22-uc0-big.patch
VServer patch	patch-2.6.22.2-vs2.2.0.3.diff

保する FLAT バイナリのロード関数が，実際に領域を確保する do_mmap_pgoff() 関数を呼び出す時に使用する flag 引数に，XIP を表す専用のフラグを追加することが考えられる．この実装については今後の課題とする．

また，VServer のメモリ資源制約リミット値はプロセスに作用するが，テスト環境を想定する場合，コンテキストが使用する全メモリ量に対しての資源制約が必要である．これも今後の課題とする．

4.3 Util-VServer について

Util-VServer は，ホスト OS 上から VServer を制御するためのツールを提供する．vcontext ツールは，コンテキストを作成し，その中で任意のプログラムを実行させることができる．Util-VServer ツールは VServer が Linux に追加したシステムコールを通じて制御を行う．

uClinux への移植では，fork の vfork への変更，vfork への変更に伴う親子プロセス間で同期を取っている箇所の修正，uCinux 用の C ライブラリ uClibc の利用，FLAT バイナリ形式への変換などを行った．

5. 評　　価

VServer と他仮想化技術を比較した文献[2)3)] を参考に，カーネルのサブシステムと DB アプリケーションの性能を，今回実装した uClinux-VServer と従来の uClinux で比較する．また，VServer の CPU 資源制限による CPU 使用率の評価を行う．表 1 に測定環境を示す．

5.1 マイクロベンチマークによる評価

カーネルサブシステムの性能を評価するために，UNIX 系 OS 向けマイクロベンチマーク LMBench[8)] を利用する．評価指標としてプロセスの生成に要する時間を用いる．ここで，本稿では LMBench2 alpha8 を元に uClinxu 適用のため fork を vfork へ変更して測定を行った．

プロセスの生成では，fork+exit，fork+execve の 2 通りを用い，生成から終了までの時間を uClinux，uClinux-VServer(コンテキスト数 1,4) にて測定した．表 2 にプロセス生成時間を示す．uClinux-VServer では，測定はコンテキスト内ではなくホスト OS 側で行った．

表 2 LMBench によるプロセス生成時間の測定結果 (マイクロ秒)

Config	uClinux	VServer-1	VServer-4
fork process	1260	1403	1361
exec process	4092	4248	4265

表 3 SQlite による測定結果 (秒)

	uClinux	VServer-host	VServer-context
user	112.5	112.7	112.63
sys	8.06	7.86	7.69

表 2 より，uClinux-VServer でのプロセス生成のオーバヘッドは数%以内であることが分かる．これより，VServer でのプロセスの生成，終了時に行われるコンテキスト情報の比較，複製にかかるオーバヘッドはほとんどないと言える．

5.2 システムベンチマークによる評価

システムベンチマークでは，軽量データベースライブラリ SQlite[9)] を評価に利用する．本稿では，SQlite 2.18.6 に対し uClinux 適用のための修正を行い，測定を行った．修正内容は，スタックの削減，不要なモジュールとのリンクの削除である．

評価指標として，ランダムに生成した 2000 件の数値データに対し，200 回クエリを実行する際に要する時間を用いる．測定には date コマンドを用いた．uClinux，uClinux-VServer での測定時間を表 3 に示す．表中の user はプロセスのユーザ空間，sys はカーネル空間での処理に要する時間を示す．なお，uClinux-VServer ではホスト OS 内とコンテキスト内の両方で測定を行った．

表 3 より，カーネルへの VServer 適用はホスト OS 上のアプリケーションに影響を与えないことが分かる．また VServer コンテキスト上のアプリケーションは，ホスト OS 上と変わらない速度で動作することが分かる．

5.3 CPU 資源制限による CPU 使用率の評価

VServer がテスト環境を提供する場合，実行環境であるホスト OS の CPU 資源をテスト環境ゲスト OS から保護する必要がある．そこで，CPU 資源を制限したコンテキストの CPU 使用率を評価する．

CPU を消費させるアプリケーションとして前述の SQlite を用いる．本環境では SQlite が利用するデータベースは RAM 上にあるため，アプリケーションの動作は I/O に制限されない CPU バウンド型となり，CPU 資源制約による影響を反映しやすく，評価に適している．

評価環境として，10%の CPU 資源利用を許したコンテキストと CPU 資源制約を行わないコンテキスト上で同時に SQlite を実行した場合と，10%の CPU 資源利用を許したコンテキストとホスト OS 上で同時に SQlite を実行した場合の 2 通りを行った．表 4，表 5

表 4 CPU 資源制約による使用率 (%): 制約されたコンテキストと制約されないコンテキスト

	CPU usage
constrained context	8.9
unconstrained context	90.5

表 5 CPU 資源制約による使用率 (%): 制約されたコンテキストとホスト OS

	CPU usage
constrained context	9.2
unconstrained context	90.0

にそれぞれでの CPU 使用率を示す. 表 5 はテスト環境と実行環境を想定している. 測定には, /proc/プロセス ID/stat が提供するシステム CPU 時間, ユーザ CPU 時間の合計を測定間隔で割った値を CPU 使用率として用いた.

表 4 より, 制約されたコンテキストは CPU 使用率が 10%に抑えられているのに対し, 制約されないコンテキストは残りの 90%の CPU 資源全てを使用していることが分かる. また表 5 より, コンテキストが 10%の使用率に抑えられているのに対し, ホスト OS は残りの 90%を使用していることが分かる. よって, VServer の CPU 資源制約によって, テスト環境の実行環境への CPU 使用率による影響は制限できることが分かる.

6. ま と め

本稿では, 組込み機器上での仮想化技術によるテスト環境の提供を目的に, OS レベルの仮想化技術である VServer を MMU を持たない組込み環境で動作する uClinux に移植し, その評価を行った. VServer の uClinux への移植では, 組込み向けファイルシステムでの保護機能と uClinux 環境でのメモリ使用制限を実現した. 性能評価では VServer による仮想環境は uClinux 上で性能を落とすこと無く動作すること, VServer が提供する資源制約によりテスト環境が実行環境に与える影響を制限可能であることを確認した.

今後の課題として, リアルタイム処理が必要な実行環境への適用, デバイスドライバなど, カーネル空間でのテスト環境の提供が挙げられる.

参 考 文 献

1) uClinux Project, http://www.uclinux.org/.
2) H. Ptzl and M. Fiuczynski, "Linux-VServer Resource Efficient OS-Level Virtualization," Proc. of the Linux Symposium, 2007 Ottawa.
3) S. Soltesz, H. Ptzl, M. Fiuczynski, A. Bavier and L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors".
4) Linux-VServer official site, http://linux-vserver.org/.
5) util-vserver project, https://savannah.nongnu.org/projects/util-vserver/.
6) ARM アーキテクチャリファレンスマニュアル.
7) Armadillo-300 official page, http://armadillo.atmark-techno.com/armadillo-300/.
8) L. McVoy and C. Staelin, "lmbench: Portable Tools for Performance Analysis".
9) SQlite Project, http://www.sqlite.org/.
10) 稗田 諭士, 朝倉 義晴, 千嶋 博, 佐藤 直樹, "組込み機器向け仮想 Linux 環境の構築," 情報処理学会研究報告, 2007-EMB-3, Vol.2007, No.4, pp.31-36 (2007).