社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－SLDM－134／2008－EMB－8 (37)
2008 / 3 / 28

携帯端末向けサスペンド機能利用型マルチOS環境における OS間通信方式

江口 悠利† 中川 智尋‡ 太田 賢‡ 竹下 敦‡

株式会社NTTドコモ 総合研究所 〒539-8536 神奈川県横須賀市光の丘3-5

E-mail: †eguchi@mml.yrp.nttdocomo.co.jp ‡{nakagawato, ohtak, takeshitaa}@nttdocomo.co.jp

あらまし 携帯電話の高機能化と共に最近ではPCのようにユーザが自由にネイティブコードのアプリを追加して端末をカスタマイズ可能なオープン端末（スマートフォンと呼ぶ）も増加している．一方で，そのオープン性の代償としてスマートフォンには情報漏洩等の懸念があり，通常の携帯電話と同等の安全性を保つことは難しい．オープン性と安全性を両立する手段として，マルチOS技術があるが，本稿では性能面に優れ，既存OSの修正インパクトが少ない，OSのサスペンド機能を利用したOS切り替え方式に着目する．この方式は，一方のOSが実行状態の際には，他方のOSは全て休止状態となる特徴がある．休止状態のOSはメール着信，電話着信不能であるため，携帯電話に適用する際には，実行中のOSと協調して着信を処理する機構，OS間の通信機構が必要となる．しかし，従来のOS間通信方式はOSの並行動作を想定しているため，OS切り替え環境には適用できない．OS切り替え環境に対応したOS間通信方式を設計，評価ボードに実装した．遅延・スループットの評価の結果，提案方式は着信通知等の少量データ転送に適用可能であること，インタラクティブな通信には不適であることを確認した．

キーワード 携帯電話，スマートフォン，マルチOS，仮想化，OS間通信

Inter-OS Communication Mechanism for Multi-OS Mobile Handset using Suspend/Resume Function

Hisatoshi EGUCHI‡ Tomohiro NAKAGAWA‡ Ken OHTA‡ Atsushi TAKESHITA‡

Research Laboratories, NTT DoCoMo, Inc. 3-5 Hikarino-oka, Yohosuka-shi, Kanagawa, 239-8536, Japan

E-mail: †eguchi@mml.yrp.nttdocomo.co.jp ‡{nakagawato, ohtak, takeshitaa}@nttdocomo.co.jp

Abstract Cellular equipment gets high-performance, and smartphone gets attention in customizations to add-on native code applications like a PC. Unlike cell phone, it is difficult to avoid threats of compromising smartphone. We use multiple-OS technology to combine open environment like smartphone and secure environment like cell phone. As a result of comparison of several technologies in terms of performance, development cost and power consumption, we select OS Switching that uses Suspend/Resume function. This has a restriction that whenever an OS executed, any other OSes are suspended. Suspend OS cannot receive any mails and phone calls. Cellular equipment must be able to receive mails and phone calls to redirect them from application in executing OS to application in suspend OS by using Inter-OS Communication. However, existing Inter-OS communication method is not suitable for OS switching. Therefore, we propose an Inter-OS communication method to cooperate with application programs in OS Switching. As a result of experimentation, our method is not suitable interactive communication, but suitable for a small mount of data communication without switching OS to notificate incoming mails.

Keyword Cell phone, Smartphone, Multiple-OS, Virtualization, Inter-OS Communication

1. はじめに

携帯電話端末の高機能化に伴って，端末に搭載される機能は増加の一途をたどり，数年前のPCと同等の性能を有するようになっている．この流れをうけて現在の携帯電話ではLinuxなどの汎用OSが使用されるようになっており，スマートフォン向けにWindows MobileやS60が提供されている．スマートフォンはPCとの親和性やアプリのカスタマイズ性が高く，利用者が自由にネイティブコードのアプリを追加/削除可能である．一方で，携帯電話は多量の情報も扱えるようになっており，個人情報や企業秘密情報も保存して活用することに十分な性能を持ち始めている．これら情報を保護するために十分なセキュリティも求められる．

このように，端末の利用形態によってカスタマイズ性を重視したスマートフォン，セキュリティを重視した一般的なケータイというように目的別に異なる端末

として存在するのが現状である．オープン性と安全性とを1つの端末で同時に実現する手段の1つとして，複数OSを1つの端末で実行するマルチOS技術がある．

弊社ではマルチOS端末の要件についてOSTI仕様[1]として公開している．マルチOS技術により，例えばWindows MobileとLinuxが1つの端末で動作できるが，互いが完全に独立して動作するだけでは，複数端末を持ち歩く必要がない程度で効果が限定的である．マルチOSが効果を発揮するのは，Windowsで作成したファイルをLinux側のメーラで送付する，一時的にLinux側の電話帳を用いてWindowsで電話するなどのアプリの連携である．

そこで，本研究ではマルチOS技術においてアプリの連携を実現するためのOS間通信技術を提案する．本文は次のように構成される．**エラー! 参照元が見つかりません。**章では本研究のベースとなるマルチOS技術とOS間通信技術を説明する．3章ではOS間通信技術を提案し，4章で実験によりその有効性を検証する．最後に，5章でまとめと今後の課題について述べる．

2. 既存技術

本章では，初めに携帯電話で利用可能な既存マルチOS方式について説明し，次に本研究の目的であるOS間連携を実現する既存のOS間通信技術を説明する．

2.1. 既存のマルチOS技術

携帯電話に代表される組み込み機器で使用可能な方式について以下に順に説明する．

- **デュアルブート**

デュアルブートは複数のOSをストレージに導入し，起動するOSをブートローダで選択する．ハードウェアを一切変更する必要がないので導入は容易である．ところが，起動可能なOSは1つだけであり，OSを切り替えるためにはOSのシャットダウン&ブートが必要になるので非常に時間がかかる．

- **マルチプロセッサ**

マルチプロセッサはOSの数だけCPUを搭載してOSを同時に動作させる．この場合，OSの変更が必要ないためポーティングは容易である．ところが，1つのデバイスを共有する場合に調停処理が必要になる．さらに，CPUやメモリの追加には基盤等のハードウェア構成を変更する必要があり，消費電力は増大する．

- **ユーザモードOS[2], [3], [4]**

ユーザモードOSはCPUで直接実行されるOSの上で別のOSを動作させる．その構成は図 1に示される．この場合，OS 1とOS 2は同時に実行できる．ところが，OS 2はOS 1上で動作するようにOS本体とドライバの双方を改変する必要がある．また，OS 2はOS 1を介して実行されるため，現状よりも性能の高いCPUを用いる必要がある．このため消費電力は増大する．

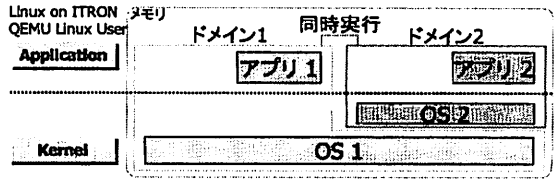

図 1 ユーザモードOSの構造

- **仮想マシン[2]**

仮想マシンはOS 1上で仮想的なハードウェアを作り出すアプリを実行する．そして，その上でOS 2を動作させる．その構成は図 2に示される．この方式ではQEMUが既存のハードウェアと同じものを見せれば，OS本体もドライバも一切改変する必要がない．ところが，ハードウェアをソフトウェアで作り出す処理は非常に重いため，ユーザモードOSよりもさらに高性能なCPUが必要になる．このため消費電力も増える．

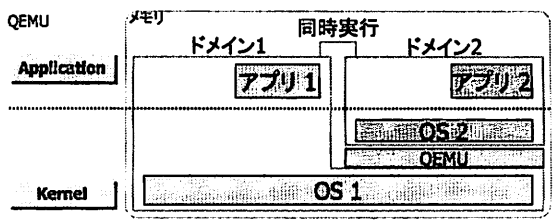

図 2 仮想マシンの構造

- **OS切り替え[5]**

OS切り替えは，物理メモリ空間を各OSに対して固定的に割り当てる．ある時点で実行中のOSは1つのみであり，それ以外のOSはサスペンドしている．休止中のOSに切り替える際には，実行中のOSをサスペンドし，休止中のOSに制御を移し，レジュームすることでOSを切り替える．その構成は図 3に示される．現在の汎用OSはサスペンド/レジューム処理を持っており，この方式ではOSを停止させる直前で次のOSの復帰作業に移るだけである．またOSは直接CPUで実行されるため既存のCPU性能でよい．ところが，複数同時にOSを実行することができない．さらに，OSの切り替え処理がサスペンド/レジュームであるため，デュアルブートよりは高速だが時間がかかる．

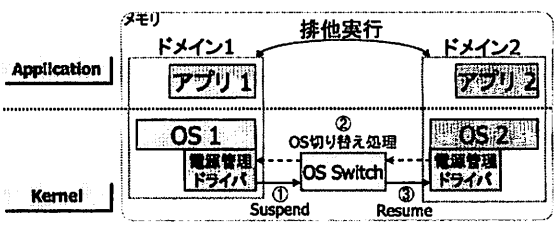

図 3 OS切り替え方式の構造

- **各方式のまとめ**

本節で説明した方式をまとめると表 1となる．OS

切り替え方式は複数のOSを実行できないという制限はあるもののその他の項目に特に欠点はない．そこで，研究ではOS切り替え方式に着目する．

表 1 マルチ OS 技術の特徴

	ユーザビリティ			コスト	
方式	消費電力	複数OS同時実行	OS切替時間	OSの改変	部材費
デュアルブート	○	×	×	○	○(ストレージの増量のみ)
マルチプロセッサ	×	○	○	○	×(複数のCPUとメモリ搭載，ストレージの増量)
ユーザモードOS				×	×(高性能CPU搭載，メモリやストレージの増量)
仮想マシン				○	×(超高性能CPU搭載，メモリやストレージの大増量)
OS切り替え	○	×	△	ほぼ○	△(メモリやストレージの増量のみ)

2.2. OS 間通信

前述のように OS 切り替え方式では，実行中の OS 以外の OS で動作するアプリは休止中である．この制約により，もしメーラが休止すると新着メールの確認を見逃して大事なメールに即座に回答できなくなる．また，通話アプリが休止すると電話を受けられないので．携帯電話として致命的な問題となる．これらを解決するためには，休止アプリの代わりに実行中の OS で着信が発生したことを検知して，検知した場合にはその旨を休止アプリに通知すると同時に，OS を切り換えて休止アプリを復帰させて処理を継続させる仕組みが必要になる．このように異なる OS 上で動作するアプリ同士が連携して動作するには OS 間通信が必要になる．そこで，本研究では OS 切り替えにおける OS 間通信手法を提案する．

- **既存の OS 間通信（Linux on ITRON）**

まず，OS切り替えのOS間通信を考える前に，既存方式を検証する．図 4はユーザモードOSに分類されるLinux on ITRONの通信手法を示している．

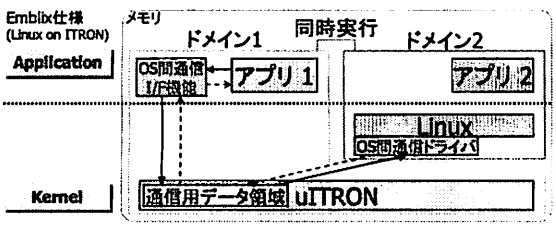

図 4 Linux on ITRON の OS 間通信

この方式では，ITRON が管理するメモリ領域に通信用データ領域を配置し，ITRON の I/F 機能と Linux の通信ドライバがこの領域を用いて通信を行う．ITRON 上のアプリ 1 は OS 間通信用の独自 API を用いて通信する必要がある．Linux 上のアプリは open, read, write など既存の API で通信することができる．

この方式を OS 切り替え方式に適用する場合，各 OS の排他的な実行は考慮されていないため，休止中の OS 上で動作するアプリに対して通信路を開設してデータを送受信させることは難しい．また，通信用データ領域は ITRON のメモリ管理を利用しており，API は ITRON 独自である．したがって，この方式をそのまま OS 切り替え方式で使うことは困難である．そこで，新たに OS 切り替え用の OS 間通信方式を提案する．

3. 提案方式

本章では OS 切り替えにおける OS 間通信方式を提案する．まず，OS 間通信を実現する上で考慮すべき制約や要求条件を整理する．

制約

A) 休眠中の OS はデータ送受信が不可能
OS 切り替えの制約であり，通信相手にデータを送受信させるには OS の切り替えが必要

要求条件

B) OS の改変を最小限に
改変が多くなると作成でなく維持管理のコストより増大する．特に組み込み機器は PC と比べると不具合は許されない傾向にある

C) 既存 API の流用
既存のプログラム作法に基づいて作成できるためアプリ設計が複雑にならなくて済む

D) コネクション志向
異なる OS で動作するアプリ同士が適切に接続を維持し，不正なイベント通知や誤通知を防止することで，安定したアプリ連携を実現する

E) OS 非依存の通信用データ領域管理
どんなOSでもデータの送受信ができると同時に，休止中OSの保護（暗号化[5]）を考慮する

上記の制約A)と条件B)～E)をすべて満たすように設計したOS間通信方式のアーキテクチャが図 5である．

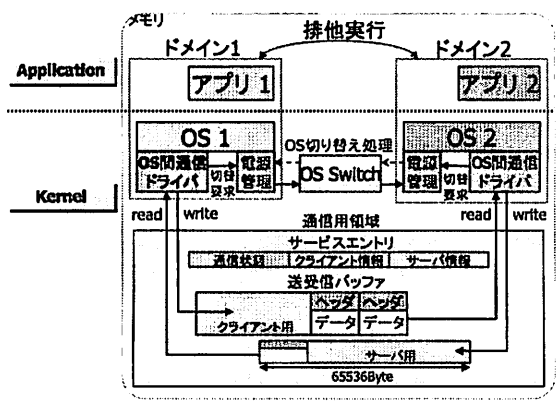

図 5 OS 間通信のアーキテクチャ

図 5を用いて制約と要求条件の解決方法を順に説明する．まず，制約A)はOS間通信ドライバが電源管理ド

ライバにOSの切り替えを依頼できるようにする．これにより，通信相手が休止状態から復帰してデータを送受信可能である．次に，条件B)はOS間通信をOS本体の改変なしでドライバのみで実現する．条件C)を満たすAPIは，通信APIの標準であるsocketと，一般的なOSが備えるファイルAPI（open, read, write）の2つが候補となる．前者はOS本体の改変が必要だが，後者は必要ないため，ファイルAPIを利用する．ただし，接続相手との通信手順はsocketに準拠とした．このため，条件D)では適切な通信相手を指定できるように，socketのポート番号に相当するサービスIDをサーバ側アプリに付与し，このIDを利用して通信路を開設する．最後の条件E)は，各OSに割り当てられた領域とは別の独立した物理メモリ領域を通信用領域として用いて通信する．これにより，休止中のOSに割り当てられたメモリ領域が暗号化などで保護されたとしても，実行中のOSは通信用領域を用いてデータ送受信ができる[5]．次に，通信用領域の詳細と通信開始のフローを図 5と図 6を用いて説明する．

図 5のように 1 つのコネクションは制御情報を含むサービスエントリと送受信バッファの2つから構成される．これらはOS間通信ドライバにより管理される．OS切り替え方式での通信では通信相手が必ず休止中となる．よって，通信相手の状態を確認するための情報として通信状態が存在する．これはコネクションの接続や切断において重要な役割を持つので図 6を用いて説明する．

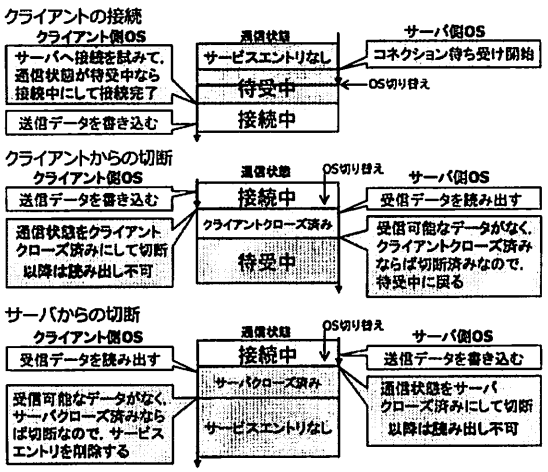

図 6　OS 間通信の接続と切断

まず，サーバで接続を受け付けると通信状態は待受中となる．クライアントが接続する際には，サーバは休止中であるため，サーバが待受中であるかをクライアントが通信状態を参照して確認する．待受中であれば接続完了となり通信状態は接続中にする．

次にクライアント側から切断する場合，通信状態をクライアントクローズ済みとする．この時点で，サーバ側は通信状態によりクライアントの切断が分かる．ところが，切断前に送信されたデータの存在を考慮して，全てのデータ受信が完了した後に切断と判定する．この後は通信状態を待受中として，新たなクライアントの接続を待ち受ける．サーバ側から切断する場合も同様でに，クライアントはサーバからデータをすべて受信した後に切断と判定する．ただし，サーバ側から切断すると他のクライアントは接続できないため，サービスエントリを削除する．

以上のように，OS 切り替え方式の OS 間通信では，接続や切断の状態を設定しても，休止中の OS が復帰した後の処理で正式に状態遷移完了となる．これは OS 切り替え方式の制約による OS 間通信の特徴である．

4. 実験

本研究では提案方式を実装し，有効性を検証するために実証実験を行った．表 2に実験環境ついて示す．

表 2　実験環境と比較方式

実験環境

ターゲット	Sophia Systems Sandgate II-P
CPU	Marvel PXA270 (624MHz)
メモリ	64MB (SDRAM)
ストレージ	128MB (NAND Flash)

比較

方式	QEMU (qemu-linux-user)	OS Switch
メモリ	64MB	Switch:4KB, OS1: 31.996MB, OS2: 32MB
OS1	Linux (Kernel: 2.6.16)	Linux (Kernel: 2.6.16)
OS2	QEMU (Linux Kernel)	Windows CE5.0

実験では表 2の評価端末を用いて，従来方式と提案方式の 2 方式を比較した．提案方式はWindows CE 5.0（以降CEと呼ぶ）とLinuxが動作するOS切り替え方式で異種OS間通信を実装した．各OSには約 32MBのメモリ領域が割り当てている．既存方式は2.1節のユーザモードOSであるqemu-linux-user（以降QEMUと呼ぶ）を用いており，Linux上でQEMUを実行することで，Linuxアプリを実行可能なOSを作り出す．

実験では，上記 2 方式について OS 間通信のスループット（bit/sec）とレイテンシ（sec）を測定して性能を評価する．そして，OS 間通信時の消費電力も評価する．実験の結果は以下に示される．

4.1. 通信性能

- **Linux が送信側**

最初にOS切り替えでのLinux→CEのデータ送信と，QEMUでのLinux→QEMUのデータ送信と，いずれもLinuxから送信する場合において，送信パケット長を 1～2048 バイト，コネクション数を 1～5 と変化させて計測した．その測定結果は図 7と図 8に示される．

図 7に示されるように，OS切り替え方式はOS切り替え処理時間が大きく，その処理中にデータの送受信が

できないことからQEMUと比べてスループットが小さくなっている．ただし，パケット長が256バイト以下の場合には比較的性能の差が小さい．

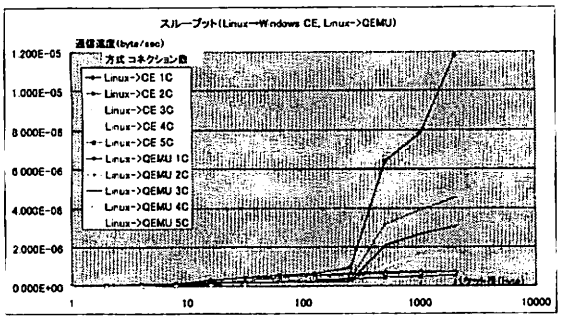

図 7　スループット（Linuxからの送信）

本実験では，一定量のデータを指定のパケット長に分割して送信している．そのため，パケット長が小さくなればなるほどデータの送受信処理の回数が多くなる．パケット長が短いときに差が小さいことからQEMUでの1回の送受信処理がOS切り替え方式よりも重く，回数の増加で差がなくなっている．

また，コネクション数の増加により送受信処理は倍々で増えるが，パケット長が256バイトより大きいと，QEMUでは大きく性能が劣化する．OS切り替え方式は通信処理よりも切り替え処理の方が重いので，コネクションを増加させても性能が劣化していない．

この実験のレイテンシ（通信処理時間）を示す．

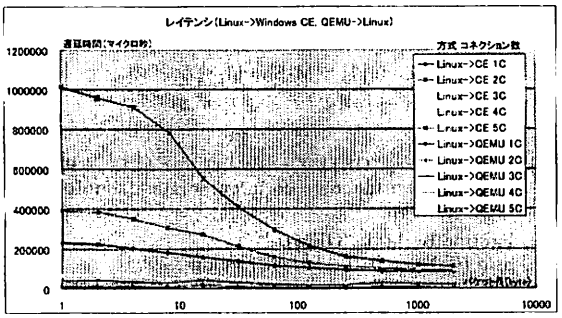

図 8　レイテンシ（Linuxからの送信）

図 8からOS切り替え方式はレイテンシがQEMUよりも大きいことがわかる．これはOS切り替え中にデータ送受信ができないことに起因する．特に，パケット長が小さいときはデータの送受信回数が増えるため，よりレイテンシが大きくなる．OS切り替え方式で性能を向上させる鍵は切り替え時間の短縮といえる．

一方，QEMUは2つのOSが同時に実行されるためレイテンシが小さい．ただし，コネクションの増加に応じてレイテンシも増加することが確認できる．

- **Linuxが受信側**

次に，OS切り替えでのCE→Linuxのデータ送信と，QEMUでのQEMU→Linuxのデータ送信と，いずれもLinuxが受信する場合において，送信パケット長を1〜2048バイト，コネクション数を1〜5と変化させて計測した．その測定結果は図 9，図 10に示される．

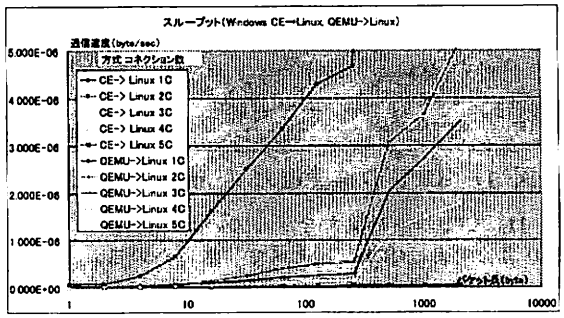

図 9　スループット（Linuxでの受信）

図 9でも図 7と同様にOS切り替え方式はQEMUと比べてスループットが小さくなっている．特にLinuxのレジュームがサスペンドよりも時間がかかっており，CE→Linuxの切り替え時間はLinux→CEと比べると大幅に大きくなるため，この結果となる．

また，コネクション数の増加に関しても図 7と同様の傾向を示しており，QEMUはコネクション数の増加で送受信処理が倍々になることで大きくスループットが減少している．一方，OS切り替え方式は，切り替え処理に大きく時間をとられるため，スループットの減少には至らない．以上の考察は図 10をみると分かる．

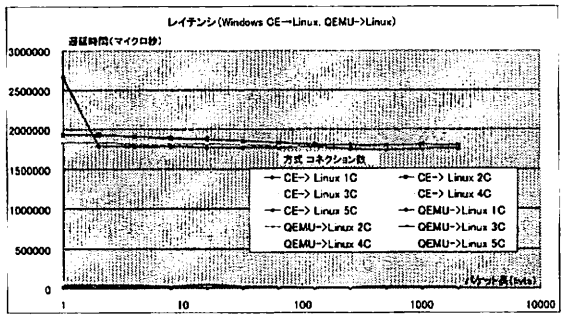

図 10　レイテンシ（Linuxでの受信）

OS切り替えではCE→Linuxが1750ms，Linux→CEの切り替えが約100msである．よって，切り替え処理が17倍遅くなったため，スループットが大幅に減少している．一方，QEMUではどちらの方向でも約50msである．2方式では遅延時間が大幅に違うことからスループットに差が出ることが分かる．

4.2. 消費電力

本節では，OS切り替えのCE→Linuxと，QEMUのQEMU→Linuxとの2つデータ送信において，送信パケット長を32, 1024バイト，コネクション数を1, 5で計測した．その測定結果は図 11，図 12に示される．い

ずれの方式でも1000msから計測を開始している．

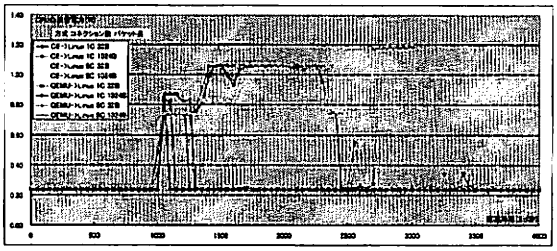

図 11　CPUの消費電力（Linuxでの受信）

図 11はCPUの消費電力を示している．OS切り替え方式の消費電力はQEMUと比べて非常に高いことがわかる．これはOS切り替えには各デバイスについてドライバが初期化処理を行うためである．またOS切り替え処理に約 1750msかかった後，ネットワークの初期化などを行うため，CPUの消費電力が高くなっている．一方，QEMUは通信処理が大きくなるとその分だけ消費電力が高くなることが分かる．

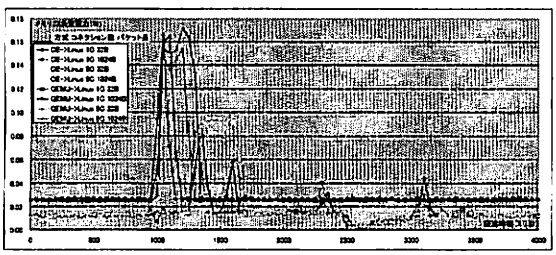

図 12　メモリの消費電力（Linuxでの受信）

図 12はメモリの消費電力を示している．OS切り替え方式はQEMUと比べると約半分の電力消費であり，切り替え後のOSをメモリに読み込まれるときに消費電力が増加する．一方，QEMUでは2つのOSが同時に動作するため消費電力が高くなる．通信時には同時に 2つのOSが処理されるため，OS切り替えの処理よりも消費電力が大きくなる．

ただし，メモリの消費電力は 0.02W と CPU の 0.2W と比べると小さいので，CPU の電力が端末全体の消費に大きく影響する．OS 切り替えは CPU の消費電力が高いため，端末全体の消費電力でも高くなる．

4.3. 考察

4.1，4.2節の実験結果をまとめると，OS切り替え方式は切り替え処理中にデータの送受信ができないため，スループットが低くなる．また，OS切り替え処理の消費電力も大きい．したがって，頻繁にOSを切り替える多量のデータ通信には不向きであり，メッセージ通知な少量のデータ通信が最適である．ただし，OSがCPU上で直接実行されるので，コネクションの増加で通信処理が増えても性能劣化が軽微という利点がある．

OS 切り替え方式の通信性能を改善するには以下の方法が考えられる．現在 OS 間通信用のバッファは 1 コネクション当たり 65536 バイトであるが，これを拡大すれば多量のデータを送信しても OS の切り替え回数が減るので性能を改善できる．また，OS のサスペンド/レジューム処理の高速化や，一部のレジューム処理の実行を必要になるまで先延ばしにすることで，通信性能と消費電力を改善することができる．

5. おわりに

本研究では，携帯電話で実現するマルチ OS 技術として OS 切り替え方式に着目し，異なる OS 上で動作するアプリが連携するユースケース（メール着信，通話着信）を実現するための OS 間通信方式を提案した．

OS 切り替え方式における通信ではサスペンド/レジュームで OS を切り替える必要があり，切り替え処理中にはデータの送受信ができないという制約がある．実験結果から，OS 切り替えのスループットは QEMU よりも劣るが，コネクションが増加してもスループットが低下しにくいことが示された．また，OS 切り替えの処理が多いため，通信時の消費電力では劣ってしまうが，通信しなければ QEMU よりも小さくなることが示された．したがって，切り替え回数の少ない少量のデータ通信に最も適する．この結果はユースケースに十分対応可能である．

今後の課題として，今回の比較ではユーザモードOSとの比較であったが他の方式とも比較検証し，4.3節であげた方式の改善を検討する．さらに，ドメイン間通信のセキュリティとして，暗号化などのソフトウェア対策と，TrustZone[7]やM-Shield[8]などのハードウェア対策とをそれぞれ検討する必要がある．

文　献

[1] NTT ドコモ，"OSTI アーキテクチャ仕様 1.00," http://www.nttdocomo.co.jp/corporate/technology/osti/, Oct. 2006.

[2] Fabrice Bellard, "QEMU," http://fabrice.bellard.free.fr/qemu/, 2005.

[3] "User-mode Linux," http://user-mode-linux.sourceforge.net/, 2005.

[4] Emblix, "Linux における RTOS とのハイブリッド構成に関する仕様書(第 1 版)," http://www.emblix.org/doc.html, Jan. 2003.

[5] 中川 智尋，江口 悠利，太田 賢，竹下 敦，"マルチ OS 環境におけるカーネル補助型難読化方式の提案," 第 42 回 MBL 研究会, Sept. 2007.

[6] 金田一 勉，"Linux on ITRON ハイブリッド構造の実装," Interface 別冊付録, July 2002.

[7] ARM, "TrustZone," http://www.jp.arm.com/products/encryption/index.html, May 2003.

[8] TEXAS INSTRUMENTS, "M-Shield mobile security technology making wireless secure," http://focus.ti.com/pdfs/wtbu/ti_mshield_whitepaper.pdf, 2005.