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

2003－UBI－2 (38)
2003／11／19

ユビキタスデバイスにおけるLinuxリアルタイム拡張

滝沢 允[†] 高橋 ひとみ[†] 守分 滋[†]
権藤 俊一[†] 永田 智大[†] 徳田 英幸[†,††]

機器端末の小型化，無線端末の普及に伴い，計算機器があらゆる場所に遍在するユビキタスコンピューティングが注目を集めている．ユビキタス環境下で用いられる計算機は小型であり，携帯可能が前提である．しかし小型機器の計算資源は通常の機器端末にくらべ有限であり，かつ不変である．そのため計算資源が限られている小型機器端末上ではアプリケーションの動作を保証するリアルタイム性やCPU資源の保証が必要となる．そこで本稿では通常のOSにリアルタイム保証を拡張し，小型機器端末上で評価を行った．リアルタイム保証を拡張したOSを用いることで，計算資源が限られている小型機器端末においてCPU資源の確保，リアルタイム性の保証が可能となった．

Linux Real-time Extensions for Ubiquitous Devices

Makoto Takizawa [†], Hitomi Takahashi [†], Shigeru Moriwake [†], Shun'ichi Gondow [†], Tomohiro Nagata [†] and Hideyuki Tokuda [†,††]

As a spread of wireless media and portable computers, ubiquitous computing, where machines exist everywhere, is paid attention. Under the ubiquitous computing, small and portable computers are required. The guarantee of real-time and CPU resource is needed for applications on small computers to work well, because the resource of such small computers is limited as well as unchanged in comparison with normal computers. We implement a guarantee of real-time as a extension of normal OS and evaluate it. Real time OS can guarantee the real-time performance and acquire CPU resource on the small computers whose resource are limited.

1. はじめに

ユビキタス環境下で用いられる様々な場所に組み込まれた機器は，既存の空間やユーザの行動を物理的に妨げることなく，目的に沿って効果的に動作する必要がある．その物理的制約によりその大きさや性能などの計算資源が制約された機器であることが前提となる．しかし，ユビキタス環境下でサービスを提供するために要する処理はサービスへの要求が高度かつ複雑になるにつれ多くの計算資源を消費する．こうした相反する条件下で信頼できるサービスを提供するためには，計算資源を個々のアプリケーションに対して適切に割り当て，その割り当ての確実性を保証することが求められる．

そこで本稿では，計算資源の限られた小型機器上で動作するアプリケーションに対して，資源割り当てとその確実性を保証するためにオペレーティングシステムの拡張を行った．この拡張により，コモディティOSであるLinux上に計算資源を抽象化したリソースセットを用い資源予約が容易になる．またマルチメディア処理で重要となる実時間周期スレッドを実現した．これにより，小型機器を用いて様々なサービスを目的に沿って効率的に動作させる環境の提供が可能となった．

次節ではユビキタス環境下で使用するユビキタスデ

† 慶應義塾大学大学院 政策・メディア研究科
Graduate School of Media and Governance, Keio University
†† 慶應義塾大学 環境情報学部
Faculty of Environmental Information, Keio University
本研究の周期スレッドに関する研究は文部科学省科学技術進行調整費「人間支援のための分散リアルタイムオペレーティングシステム基盤技術の研究」の支援による．

バイスについて説明する．続く第3節ではLinuxカーネルへのリアルタイム拡張について述べる．そして第4節では評価を示し，第5節では関連研究について述べ，最後に今後の課題について述べる．

2. ユビキタスデバイス

ユビキタスデバイスとはネットワークに接続した情報家電など，他の小型機器と連携し，サービスや機器の動作を制御するデバイスである．ユビキタスデバイスとは以下の特徴がある[1]．

- 携帯が可能
- バッテリ寿命が長い
- 高価なI/Oがない
- ネットワークが構築可能
- 十分なストレージがある

そこで我々は多様なデバイスを着脱できるユビキタスデバイスである，U-coreとU-Jacketを開発している．U-coreとはユビキタスコンピューティングに必要な基本的な計算資源（CPU、メモリなど）を搭載した小型ハードウェアである．またU-coreは必要最小限の入出力装置を持ち，U-Jacketと呼ばれるデバイスの着脱が可能である．U-Jacketとはディスプレイやセンサなどの U-coreを補完する機器を有し，U-coreが内部に組み込まれてはじめて機能する着脱可能なデバイスである．

U-coreの様なユビキタスデバイスは他の周辺機器と連携をしながら動作すると考えられる．しかし周辺機器と連携をする際，自分自身の情報のみで連携を行うことは難しい．そのためユビキタスデバイスでは他の周辺機器やセンサからの情報を取得するため，モニタリングやセンシング機能が必要となる．またユビキタスデバイスは，PDAやユニバーサルコミュニケータとして使用された場合，動画や音声といったマルチメディアデータを扱う場面も想定される．

ユビキタスデバイスではマルチメディアデータやデータのセンシング，モニタリングを行う際，適切なタイミングで処理を行わなければ，サービスの信頼性の低下や，大きなリスクを生じさせる危険性がある．しかしデバイス自体が通常のPCと比べ，小型であり計算資源が限られるため，資源が枯渇しリアルタイム性を保証できない問題が発生する．そのためユビキタスデバイス上では計算資源を個々のアプリケーションに適切に割り当て，リアルタイム性を保証する必要がある．

3. Linuxでのリアルタイム拡張

リアルタイムシステムではリアルタイムタスクの時間的制約により，ハードリアルタイムタスクとソフトリアルタイムタスクに分けられる．ここではvalue function[2]を用いて定義する．あるタスクの時間的制約が満たされない場合に，システムにとってその効用がマイナス無限大になる時，そのタスクをハードリアルタイムタスクという．また時間的制約が満たされない場合に，その効用が穏やかに減少していく時，そのタスクをソフトリアルタイムタスクという．本稿ではソフトリアルタイムタスクを実行するリアルタイムシステムを実現する．

ユビキタスデバイス上でソフトリアルタイム性の保証を実現するため，組み込み機器向けCPUであるStrong ARMおよびXscale上でLinuxに対するリアルタイム拡張を行った．またリアルタイムをサポートするOSとしてCMU(Carnegie Mellon University)のLinux/RK[3]を元に拡張を行った．RK(Resource Kernel)はLKM(Loadable Kernel Module)として提供されているため，既存のLinuxに対する少量の変更でリアルタイム拡張が可能である

3.1 Linux/RK

RKの特徴として，異種プラットホームへ移植が容易である．異なるプラットホームでは同様の動作であっても実行時間が異なる．RKはロード時にOSタイマを計算することで各周波数に適応する．またRKはLKMとして実装されているため，移植性が高い．しかし若干フック関数をカーネル本体にインタフェースとして組み込む必要がある．

以降，第3.2節でRKの基本概念である予約(Reserve)機構について，第3.3項において予約の集合を扱うリソースセットについて述べる．また第3.4項では周期スレッド(Periodic thread)に関して説明する．

3.2 予約 (Reserve)

予約とはプログラムに対する計算資源の割り当てである．計算資源としてCPU時間，物理メモリ，ネットワーク帯域，ディスク容量が予約可能である．CPU

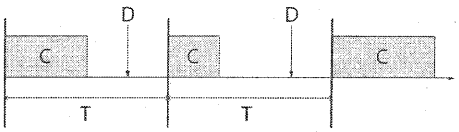

図 1 CPU 時間予約

時間予約を図 1 に示す．CPU 時間予約は C, D, T の 3 つのパラメータで表現され，T はプロセスの周期を表し，C は周期 T 内における実行時間を表す．D はデッドラインを表し周期 T 以下である．実行時間 C は D 以前に完了する必要がある．

周期 T 内で C を使い果たした時点を枯渇状態，C 時間を実行していない時点を非枯渇状態と呼ぶ．また次の周期で新たに実行時間 C を得ることを補充という．枯渇時点から周期 T の終端までの予約機構の動作から，リソース管理モデルは以下 3 種類に分かれる．

- Hard reserves: 周期 T 内において，C を使い果たせば次の周期までスケジューリングされない．
- Firm reserves: 他の予約を使い果たしていないプロセス，または通常のプロセスが実行可能状態になければスケジューリングされる．
- Soft reserves: C をすべて消費したとしても，スケジューリングされる．

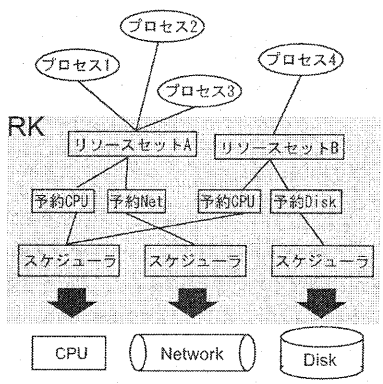

図 2 リソースセット

3.3 リソースセット (Resource Set)

図 2 はリソースセットと予約の関係を示す．リソースセットは予約の集合であり，1 つ以上の予約をグルーピングして，リソースセット単位で 1 つ以上のプロセスに関連付けされる．リソースセットは各資源の予約を抽象化したものであり，ユーザアプリケーションへのインタフェースとなる．

表 1 周期スレッド用 APIs

関数名	動作
rt_make_periodic (周期，開始時間)	周期実行のための相対的な開始時間および周期を設定
rt_wait_for_start_time (void)	周期実行開始時まで休眠
rt_wait_for_next_period(void)	次の周期まで休眠

3.4 周期スレッド (Periodic thread)

RK がサポートする周期スレッドでは，スレッドが周期 t で繰り返し実行される．通常 OS のプロセススケジューラである，タイムシェアリングスケジューラでは困難な周期性を実現し，動画などの連続メディアの再生に必要な周期性を保証する．周期スレッドは，次の手順で動作する．1) プロセスは周期実行のおいて周期 t でタイマ割り込みを設定後，TASK_INTERRUPTIBLE 状態へ遷移かつ休眠する．2)t 経過後のタイマ割り込みにより，プロセスは TASK_RUNNING 状態へ遷移して実行される．3) 処理後，TASK_INTERRUPTIBLE 状態へ遷移かつ休眠し，次のタイマ割り込みを待つ．以降これを繰り返す．ここで，各周期内での処理時間は周期 t 以下である．以上を実現するための API を表 1 にあげ，この API を用いて記述するアプリケーション疑似コードを図 3 に示す．

4. 評　　価

本節では Linux/RK(RK) と通常の Linux(Linux) を比較しリアルタイム性の評価を iPAQ H3970 上で行った．評価実験では周期スレッドを用いて動画再生を行い，各フレームギャップを測定した．Linux のフレームギャップを外乱プロセスの有無それぞれの状態で測定し，また同じ外乱に対して RK のフレームギャップを測定した．この三つの測定をローカルファイルシステムの動画ファイル再生と，ネットワークにストリーミングされる動画データ再生の二種類のタスクに対して行った．

4.1 評価環境

周期スレッドは Hewlett-Packard 社の，iPAQ H3970 にて実行した．H3970 の CPU は XScale PXA 400MHz であり，メモリは 64MB の DRAM と 32MB の Flash ROM を搭載している．OS として hand-

```
main()
{
  time start; /*開始時間*/
  time period;/*周期*/

/* 相対的な開始時間の指定 1 秒 */
  start = 1 s;

/* 周期の指定 500 ミリ秒 */
  period = 500 ms;

/* 以降の処理を周期実行させるための設定 */
  if (rt_make_periodic(&period, &start) < 0)
      return -1 /* エラー */

/* 開始時間まで休眠 */
  if (rt_wait_for_start_time() < 0);
      perror("rt_wait_for_start_time");

  while (TRUE) {
      :
      周期内に実行される処理
      :
/*次周期まで休眠*/
  if (rt_wait_for_next_period() < 0);
      perror("rt_wait_for_next_period");
  }
}
```

図 3　周期スレッド API を組み込んだ疑似コード

helds.org[4] の Familiar Linux ディストリビューションのバージョン 0.7 をインストールし，そのカーネルを Linux/RK に差し替えた．

動画再生ソフトウェアには，GPL で公開されている Mplayer[5] を用い，周期スレッドの拡張を行った．拡張では動画再生を始める前に，リソースセット及び周期スレッドの設定と登録を行った．また動画再生終了時に，リソースセットと周期スレッドを破棄するコードを付け加えた．以上の動作を行うために付け加えたコードはわずか 50 行ほどであった．周期スレッドではリソースセットを用い CPU 時間を予約しており，それぞれのパラメータを $T = 20$ ミリ秒，$C = 10$ ミリ秒，$D = 20$ ミリ秒とし，Hard reserves で設定した．また再生する動画は Quicktime 映像フォーマットでエンコードしたデータを利用し，フレームレートは 30 フレーム毎秒とした．

外乱プロセスとして浮動小数点計算を約 18 万/秒回行うベンチマークソフトを十個動作させた．iPAQ には Floating Point Unit がないため浮動小数点計算はソフトウェアで擬似的に処理している．そのため，ベンチマークソフトで浮動小数点計算を行うとシステムに多大な負荷がかかる．

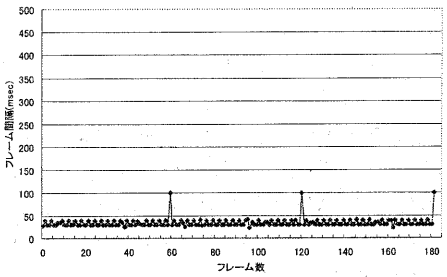

図 4　ローカル：RK なし，外乱プロセスなし

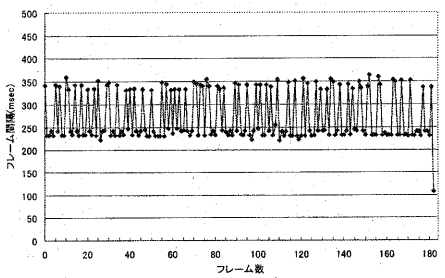

図 5　ローカル：RK なし，外乱プロセスあり

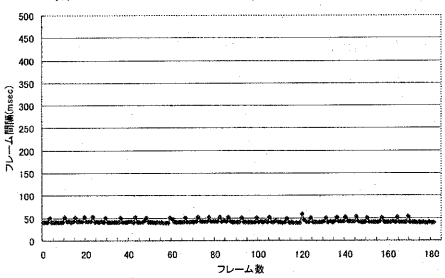

図 6　ローカル：RK あり，外乱プロセスあり

4.2　ローカルファイルシステムからの再生

iPAQ の Flash メモリ上にムービファイルを置き，それを Mplayer のコマンドラインで指定し，再生した．ファイルシステムのフォーマットは JFFS2(The Jounarling Flash Filesystem, version 2) である．

Linux を用い外乱プロセスがない状態で再生した場合，再生時のフレームギャップは図 4 となる．外乱プロセスがないためフレームギャップは約 30 から 40 ミリ秒と周期を守って再生しており，偏差が 9.7 となった．

次に Linux を用い外乱プロセスを発生させ動画再生を行った．再生時のフレームギャップは図 5 となる．外乱プロセスが動作しているため，フレームギャップ

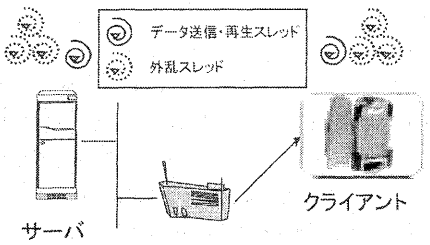

図 7 実験環境

は約 230 から 350 ミリ秒となり大きく動画再生が遅れている．また偏差が 53.2 とフレームギャップに大きなバラツキがでており，周期が安定せず正しく動画を再生できない．

最後に外乱プロセスを動作させ，RK を用い動画再生を行った．再生時のフレームギャップは図 6 となる．外乱プロセスが動作中であってもフレームギャップは約 40 から 50 ミリ秒であり，また偏差が 4.05 となり外乱プロセスがない Linux 上での再生よりも安定して再生していることがわかる．

4.3 ネットワークからの再生

ThinkPad 240X をサーバとして，動画ファイルを TCP にて送信し iPAQ 上で動画再生を行った．実験環境を図 7 に示す．サーバである ThinkPad と iPAQ 間は無線アクセスポイントを使用し IEEE 802.11b でつなげ，無線 LAN カードにはメルコ社製の WLI-CF-S11 を使用した．

ThinkPad には RedHat Linux 8.0 をインストールし，インテル 386 アーキテクチャに対応した Linux/RK カーネルを搭載した．ThinkPad で動作する動画ファイル送信ソフトウェアにも，周期スレッドを用いた．またネットワークの QoS として Linux カーネルに標準で付属している CBQ[6] を利用し，ネットワーク帯域を保障した．クライアントである iPAQ では第 4.2 節と同様に Mplayer を用い動画を再生し，フレームギャップを測定した．

Linux を用い外乱プロセスがない状態で再生した場合，再生時のフレームギャップは図 8 となる．外乱プロセスがないためフレームギャップは約 30 から 40 ミリ秒と周期を守って再生しており，偏差が 9.9 となった．

次に Linux を用い外乱プロセスを発生させ動画再生を行った．再生時のフレームギャップは図 9 となる．

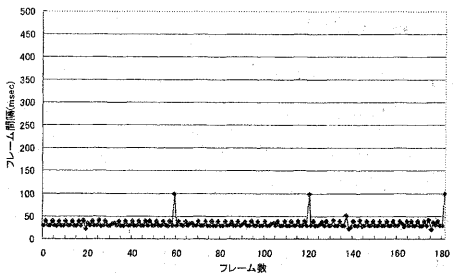

図 8 ネットワーク：RK なし，外乱プロセスなし

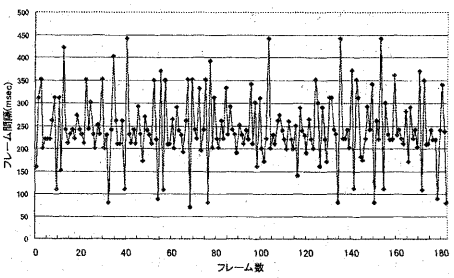

図 9 ネットワーク：RK なし，外乱プロセスあり

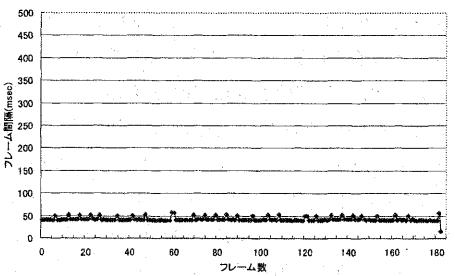

図 10 ネットワーク：RK あり，外乱プロセスあり

外乱プロセスが動作しているため，フレームギャップは約 100 から 450 ミリ秒となり大きく動画再生が遅れている．また偏差が 73.9 とフレームギャップに大きなバラツキがでており，ローカルで再生した場合より周期が安定せず正しく動画を再生できない．

最後に外乱プロセスを動作させ，RK を用い動画再生を行った．再生時のフレームギャップは図 10 となる．外乱プロセスが動作中であってもフレームギャップは約 40 から 50 ミリ秒であり，また偏差が 4.7 となった．

ネットワークを介し動画を再生した場合も，ローカル上で再生した場合と同様に RK を使用することでリアルタイム性の保証が可能であることが判明した．

5. 関連研究

Linux/RKと同様なソフトリアルタイムOSとして，徳田らがMach-3をベースに開発したRT-Mach[7]がある．RT-Machはリアルタイムスレッドやデッドラインハンドラをサポートしている．

TRON(The Real-time Operating system Nucleas)[8]は東京大学の坂村らが開発したリアルタイムOS規格である．仕様は無償で公開されており組み込み用のITRON規格もある．実装は公開の義務がないため，利用できるデバイスドライバやプロトコルスタックの数はLinuxより少ない。また実時間周期スレッドのプリミティブもない．

RTLinuxはFSMLabsが開発したLinuxをベースとしたハードリアルタイムOSである．Linuxのスケジューラの上位にリアルタイムスケージュラを実装した。このスケジューラで，Linuxをもっとも優先度の低いプロセスとして実行し，マイクロ秒単位の実時間性を保証している．しかし実時間プロセスのために独自のデバイスドライバを用意する必要があり，コモディティOSとしてのLinuxの資源を活用できない．

また，Linuxで資源管理を行う機構としてIBMが開発したClass-based Kernel Resouce Management(CKRM)[11]が挙げられる．プロセスをクラスに分類し，クラスごとにCPU，I/O，メモリ，そしてネットワーク資源を割り当てる．Webサーバやデスクトップにおけるサービスの差別化を目的としている．

6. まとめと今後の課題

ユビキタスデバイスにおいてリアルタイム性の保証を行うためLinuxを拡張したLinux/RKをXScale上に移植し，評価を行った．実装ではLinux/RKの提供するリソースセットを用い，実時間周期スレッドを実現できるよう拡張を行った．評価としてローカル，ネットワーク越しの動画再生時のフレームキャップを通常のLinuxと拡張の施したLinux/RKとで比較を行った．外乱プロセスがある場合，Linuxのフレームギャップの偏差が53.2，73.9であるのに対し，Linux/RKは外乱があっても4.05，4.07とソフトリアルタイムタスクに対して実時間を保証した．Linux/RKは通常Linuxが安定して動作できない環境でも，計算資源を予約することで安定してアプリケーションが動作できることを示した．

今後の課題としては，まずデッドラインハンドラ[12]の実装がある．デッドラインハンドラの使用により，動画再生時にCPUサイクルが不足した場合にフレームレートを落としたり，ネットワーク帯域が不足した場合にサーバを変更するといった動的な対応ができる．次に資源管理の機構をCPUからメモリやI/Oにも拡張するために，CKRMの機構と統合する．また今回はPDAを用いLinux/RKの実装を行ったが，センサノードといったさらに小型な機器端末への実装をすすめる．

参考文献

1) Nobuhiko, N., Iwamoto, T., Nagata, T. and Tokuda, H.: Wearable Network: Architecture and Implementation, in *IEEE International Workshop on Networked Appliances(IWNA)* (2000).
2) Jensen, E. D., Locke, C. D. and Tokuda, H.: A Time-Driven Scheduling Model for Real-Time Operating Systems, in *Proceedings IEEE Real-Time Sytems Symposium*, pp. 112–122 (1985).
3) Oikawa, S. and Rajkumar, R.: Linux/RK: A Portable Resource Kernel in Linux, in *In Proceedings of the IEEE Real-Time Systems Symposium* (1998).
4) The Familiar Project,: http://familiar.handhelds.org.
5) MPlayer, : http://www.mplayerhq.hu.
6) Hubert.B.: Linux Advanced Routing & Traffic Control HOWTO,http://lartc.org/howto/
7) Savage, S. and Tokuda, H.: Real-Time Mach Timers: Exporting Time to the User, in *Proceedings of the Third USENIX Mach Symposium*, pp. 111–118 (1993).
8) Sakamura, K.: TRON – Total Architecture, 情報処理学会アーキテクチャワークショップ, pp. 41–50 (1984).
9) Vodaiken,V. and Barabanov,M.: A Real-Time Linux, *Linux Journal* (1996).
10) ART-Linux, : http://www.movingeye.co.jp.
11) Franke, H.: Class-based Prioritized Resource Control in Linux, in *In Proceedings of the Linux Symposium* (2003).
12) Tokuda, H. and Kitayama, T.: Dynamic QOS Control based on Real-Time Threads, in *Proceedings of the Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 113–122 (1993).