2005－OS －98（6）
2005／2／22

組み込みLinux の実時間機能によるリアルタイム通信機構

間 博人[1]　滝沢 允[1]　守分 滋[1]　徳田 英幸[1 2]

[1] 慶應義塾大学大学院 政策・メディア研究科
[2] 慶應義塾大学 環境情報学部

小型情報機器の普及と共に，それらに対するサービス要求も複雑化してきている．ネットワーク資源，計算資源が限られている小型情報機器でリアルタイム通信を実現するには，ネットワーク資源の確保以外にもアプリケーションの動作を保証するリアルタイム性や計算資源の保証が必要となる．本稿では，ハンドヘルドや組み込みシステムを対象とした実時間機能を実現する Linux を開発し，ストリーミングデータを無線通信で受け取りリアルタイム性を失わずにディスプレイに描画させる評価実験に成功した．

Real-time Communication based on Embedded Linux/RK

Hiroto Aida[1]　Makoto Takizawa[1]　Shigeru Moriwake[1]　Hideyuki Tokuda[1 2]

[1]Graduate School of Media and Governance, Keio University
[2]Faculty of Environmental Information, Keio University

To construct a more predictable computing environment, resource reservation mechanisms are indispensable in the modern multimedia systems. Especially, the guarantee of real-time and CPU resource is needed for applications on embedded computers to work well, because the resource of such embedded 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 and Network Resource on the embedded computers whose resource are limited.

1　はじめに

組み込み機器の高性能化や利用形態の拡大に伴い，ネットワーク機能や連続メディア処理といった従来よりも高度なリアルタイム処理を組み込み機器に対して要求するサービスが増えてきた．計算資源に制約のある小型計算機器上で信頼できるサービスを提供するには，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．さらに高度なリアルタイム処理を実現するには，計算資源だけにとどまらず，ネットワーク資源，メモリ資源など複数の資源を統合的に確保する必要がある．また，連続メディア処理のセッションなどは複数スレッドが協調動作することにより構成されているため，これらのスレッドが使用する資源をまとめて制御する必要がある．

本研究は，ハンドヘルドや組み込み機器を対象とした Embedded Linux/RK を開発し，組み込み機器上でリアルタイム通信や連続メディア処理をする際に，統合的な資源予約と資源予約の継承が有効であることの実証実験を目的とする．

本論文の構成は，第 2 節で Embedded Linux/RK の概要と，資源管理の概念について説明する．次に第 3 節で，組み込み機器上でのリアルタイム通信を行う場合に必要となる事項について議論する．第 4 節では，iPAQ を用いた評価実

験を説明し，最後にまとめと今後の課題について述べる．

2 組み込み機器上でのリアルタイム通信

組み込み機器上での通信を完全なハードリアルタイムで実現するためには，オペレーティングシステムの支援のみでは難しい．特に無線ネットワーク環境においては，マルチパスフェージングや干渉などでメディアアクセス制御時に予測できない遅延が発生する．また反対に，物理層およびリンク層で完全にハードリアルタイムを実現したネットワークが存在しても，送信側，受信側の計算資源およびネットワーク資源を確保は必要である．送信プロセスに対して計算資源，およびネットワーク資源を確保していない場合，外乱プロセスや外乱フローが存在すると物理層およびリンク層に到達する以前にリアルタイム性が阻害されスムースなデータ送信は望めない．また受信側も同様に，受信プロセスの計算資源を確保は必要である．さらに受信したデータを再生，描画といった処理を行う場合は，それらの派生したプロセスに対する資源確保も重要である．

2.1 想定するアプリケーション

組み込み機器の高性能化に伴い，ネットワーク機能とマルチメディア処理が重要な要件となっている．こうした状況を踏まえ，本研究では組み込み機器上で動画や音声といったストリーミングデータを無線通信で受け取り，リアルタイム性を失わずに再生やディスプレイに描画を行うアプリケーションを考える．

このアプリケーションで生じる遅延は，大きく送信側の遅延，中間経路での遅延，受信側の遅延に分類できる．このうち中間経路での資源確保はリアルタイム通信を実現する際の重要な要素であり，DiffServ や RSVP といった機構が提案されている．本研究では，エンドシステムの組み込み機器内でソフトウェアによって制御可能な資源予約管理を対象とし，中間経路での資源予約については触れない．

こうしたアプリケーションをを実現する上で，「複数資源の統合」および「複数プロセスの統合」の 2 つがキーワードとなる．次にこの 2 つについて簡単に説明する．

複数資源の統合

動画転送処理を行うためには，プロセッサやメモリ，ネットワークなど複数の資源が必要となる．そのため，アプリケーションが複数の資源を扱うことできるフレームワークが必要となる．

複数プロセスの統合

ネットワークから動画を受信し，描画するアプリケーションでは，複数プロセスが協調動作することにより構成される．動画を受信するプロセスと，それを描画する X Window のワーカプロセスは協調して一つのセッションを構成している．このセッションのリアルタイム性を維持するためには，これらのプロセスに必要な計算機資源の予約はまとめて行われるべきである．そのために，資源予約は協調動作するプロセス間で共有が出来るようにすべきである．

次節では，組み込み機器でのリアルタイム通信を実現する手法として用いる Embedded Linux/RK の概要と，資源管理の概念について説明する．

3 Embedded Linux/RK

我々は，Real-Time Mach マイクロカーネル [7] に対し計算資源の予約 [1]，ネットワーク資源の予約 [11]，統合的な資源予約 [10, 3] を導入してきた．Embedded Linux/RK は，CMU (Carnegie Mellon University) の Linux/RK [4] を組み込み機器向け CPU である StrongARM および，その後継である Xscale 上で動作する Linux へ移植し拡張を施したものである．この Embedded Linux/RK により，これまで Real-Time Mach マイクロカーネルで実現してきたリアルタイム機能を組み込み機器上で実現することが可能となる．現状では，Linux カーネルバージョン 2.4.19 を採用し，Familiar v0.7.2，および Embedix v2.0 に対応している．

Linux/RK は，リアルタイム性を拡張する方式として Resource Kernel (RK) [5] の手法を用いている．さらにオリジナルの Linux カーネルのソースプログラムを改変を最小限に抑え，拡張部分をカーネルモジュールとして実装している．これらのカーネル内に実装された機能は，C 言語によって実装したライブラリ API としてシステムコールを通じて，ユーザレベルのアプリケーションから呼び出すことが可能となっている．

図 1 は，Embedded Linux/RK の資源管理を図示したものである．RK の資源管理機構は，「資源予約」と「リソースセット」の 2 つの概念から成る．2.1，2.2 では「資源予約」と「リソースセット」の概念について簡単に説明し，2.3 ではユー

ザー空間のプログラムと RK のインタフェースとして提供する API とラッパーモジュールについて説明する．

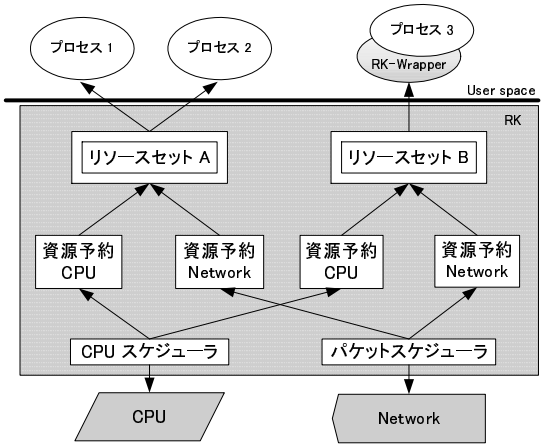

図 1: Embedded Linux/RK の資源管理

3.1 資源予約

資源予約とはプログラムに対する資源の割り当てである．資源として計算時間，ネットワーク帯域，ディスク帯域が予約可能である．図 1 で示すとおり，各資源毎のスケジューラが資源予約を実現する．この際，各スケジューラで用いるスケジューリングポリシは選択することが出来る．

資源予約は C，D，T の 3 つのパラメータで表現され，T は周期を表し，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，資源管理モデルは以下 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．

この 3 種類を，資源予約時に選択することで柔軟な資源予約を実現する．

3.2 リソースセット

リソースセットとは，資源予約の集合を抽象化した概念である．図 1 にリソースセットと資源予約，プロセスの関係を示す．リソースセットは，一つ以上の資源予約をグループ化して，ユーザ空間のプロセスへのインタフェースとなる．ユーザ空間のプロセスは，リソースセットとバインドすることで，複数ある下位の資源を一元的に予約することが出来る．

3.3 RK とのインタフェース

Embedded Linux/RK は，ユーザプログラムのインタフェースとして API とラッパーモジュールを用意している．ユーザプログラムは API を用いて，資源予約とリソースセットの生成や，プロセスを生成したリソースセットに割り当てることが出来る．しかしユーザプログラムが API を使用して RK の資源管理機構を用いるには，ソースコードの変更が必要となる．図 2 に，CPU 資源予約および周期スレッド API を組みこんだ擬似コードを示す．API を用いて資源予約をする場合，通常 50 行ほどのコードを新規に加える必要がある．

Embedded Linux/RK は，ソースコードの変更なしに RK の資源管理機構を利用する方法として RK-Wrapper を用意している．RK-Wrapper は，RK のラッパーモジュールでコマンドを用いて資源予約とリソースセットを生成し，特定のプロセスに割り当てることが出来る．図 1 において，プロセス 1, 2 は直接 リソースセット A と結びついている．それに対してリソースセット B は，RK-Wrapper が生成している．プロセス 3 は，RK-Wrapper を介してリソースセット B が割り当てられている．

RK-Wrapper を用いることで，既存のアプリケーションのソースコードを書き換えずに RK の資源管理機構を利用することが出来る．また既に起動中のプロセスに対して，リソースセットを割り当てることも可能である．

4 評価実験

本節では，Embedded Linux/RK の性能を評価するために，iPAQ h5550 を用いて実験を行う．実験に用いた Hewlett-Packard 社の iPAQ h5550 は，CPU として XScale PXA255 400MHz を搭載し，メモリは 128 MB の DRAM と 48 MB の Flash ROM を搭載している．OS は，handhelds.org が提供している Familiar Linux ディス

```
/*リソースセットを作成*/
  if ((rs = rk_resource_set_create
      (rs_name)) == NULL_RESOURCE_SET) ...
/* CPU 資源を予約  */
  if ( rk_cpu_reserve_create(rs, &attr)
                                  < 0)  ...
/* CPU 資源をリソースセットに割り当てる*/
  while(rk_resource_set_get_cpu_rsv(rs)
                                  == 0);
/*プロセスにリソースセットを割り当てる*/
  if (rk_resource_set_attach_process
                        (rs,pid) < 0) ...
/*自スレッドを周期スレッドに変更*/
  if (rt_make_periodic(&attr.period,
                &attr.start_time) < 0) ...
/* 開始まで待つ */
  if (rt_wait_for_start_time() < 0) ...
  while (TRUE) {
     /* Thread body */
     /* 次の開始までブロック */
     if (rt_wait_for_next_period()
                                  < 0) ...
  }
/* リソースセットの削除 */
  if (rk_resource_set_destroy(rs)
                                  < 0) ...
```

図 2: CPU 資源予約および周期スレッド API の擬似コード

トリビューションの version 0.7 をインストールし，カーネルを Embedded Linux/RK に差し替えた．評価実験における CPU 資源予約は，パラメータを T = 20 ミリ秒，C = 18 ミリ秒，D = 18 ミリ秒とし，Hard reserves で設定した．この際の CPU スケジューラは，RM (Rate-Monotonic) スケジューリングポリシーを用いている．また，ネットワーク資源予約時のパケットスケジューラは，Linux カーネルに標準で付属している CBQ (Class-Based Queueing) を利用した．

はじめに予備実験として，計算資源がパケットの送受信にどのような影響を及ぼしているかを確認した．次に，ストリーミングデータを無線通信で受け取りディスプレイに描画するアプリケーションでの評価実験を行う．

4.1 計算資源がパケットの送受信に及ぼす影響

予備実験として，計算資源がパケットの送受信にどのような影響を及ぼしているかを確認する．予備実験の環境を図 4 に示す．2 台の PDA 間での VoIP を想定し，無線デバイスとして Bluetooth (ver 1.1b) を用いる．UDP パケットを Constant Bit Rate (CBR) で送信し，受信側でパケット到着間隔を測定した．また送信側，受信側双方に，CBR/UDP の通信プロセスに加え，外乱プロセスとして浮動小数点計算を繰り返すプロセスを 10 個動作させる．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，外乱プロセスで浮動小数点を行うとシステムかかる負荷は大きい．

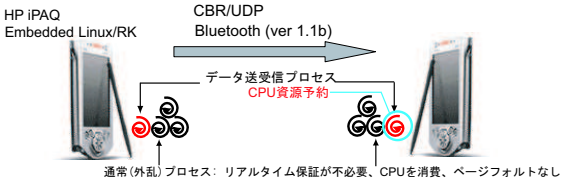

図 3: 予備実験環境

図 4 に 20 ミリ秒間隔で 330 バイトの UDP パケットを送信した際の，受信側のパケット受信間隔を示す．RK による計算資源予約がない場合，到着間隔は 10 ミリ秒から 60 ミリ秒まで振動している．それに対し，RK による計算資源予約がある場合は，ほぼ 20 ミリ秒間隔でパケットを受信できている．若干の揺らぎは，無線の干渉やマルチパスフェージングによるメディアアクセス制御での遅延が影響していると考えられる．

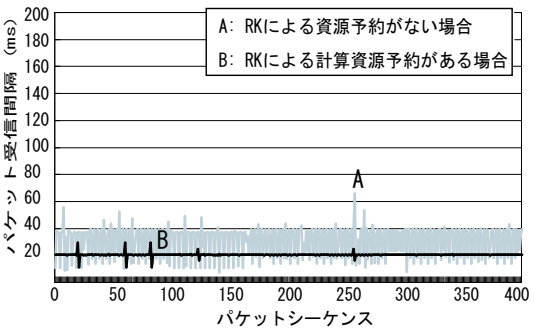

図 4: Bluetooth ネットワークでの周期パケットの到着間隔

この予備実験の結果から，計算資源の不足はスムースなパケットの送受信の妨げになり，リアルタイム通信の実現には，RK による計算資源予約が有効であることが分かる．

4.2 資源予約の共有と複数資源の予約の有効性

Embedded Linux/RK の資源予約管理機構が実現する「資源予約の共有」と「複数資源の予約」の有効性を検証するために，ストリーミングデータを無線通信で受け取りディスプレイに描画するアプリケーションでの評価実験を行う．

実験環境

図 5 に実験環境を示す．送信ホストから受信側の iPAQ h5550 へ IEEE 802.11b を用いて動画を送信する．送信ホストは，CPU がインテル Pentium M プロセッサ 2.1GHz，メモリを 1GB 搭載した IBM ThinkPad T42p を用いた．また送信ホストのカーネルは，x86 用の Linux/RK を用いた．動画の各フレームは平均 13 kbyte の Jpeg 画像である．次に，受信側の外乱の影響と，送信側の外乱の影響に分けて実験を行う．双方の実験の評価軸として，受信側のフレーム処理時間を用いる．受信側のフレーム処理時間は，次のフレームを受信するまでの recv のブロック時間，受信したフレーム (Jpeg 画像) を復号する時間，復号したメモリ上のフレームを画面に描画する時間に分けることが出来る．recv のブロック時間は，1 フレームを復号しを描画するまでの時間よりも 1 フレームを受信する時間が長いと発生する．

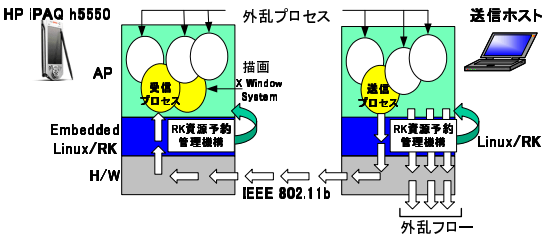

図 5: 動画の送受信と描画

受信側の外乱の影響

受信側の外乱の影響を調べるために，受信側のフレーム処理時間を評価軸とし，以下の 5 つの実験を行った．

- 実験 1: 外乱プロセスなし，資源予約なし
- 実験 2: 外乱プロセスなし，CPU 資源予約あり (受信プロセス)
- 実験 3: 外乱プロセスあり，資源予約なし
- 実験 4: 外乱プロセスあり，CPU 資源予約あり (受信プロセス)
- 実験 5: 外乱プロセスあり，CPU 資源予約あり (受信プロセス+ X Window System)

外乱プロセスとして浮動小数点計算を繰り返すプロセスを 10 個動作させる．また CPU 資源予約を持つリソースセットに対し，実験 2 と 実験 4 では，動画を受信するプロセスをアタッチした．実験 5 では，受信プロセスと X Window System の双方をリソースセットにアタッチし，CPU 資源予約を共有している．送信側の環境は，外乱プロセスおよび外乱フローがなく，資源予約を行なっていない状態で固定している．

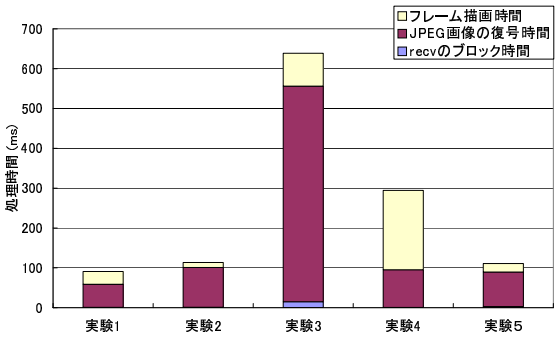

図 6: 実験 1-5: 受信側のフレーム毎の処理時間

実験 1 から実験 5 までの受信側の平均フレーム処理時間を図 6 に示す．実験 1 と実験 2 を比較すると，実験 1 の平均フレーム処理時間が 90.9 ミリ秒であるのに対し，実験 2 の平均フレーム処理時間が 113.5 ミリ秒であった．このことから，Embedded Linux/RK の資源予約管理機構のオーバヘッドが 1 フレームあたり 20 ミリ秒程度あることが分かる．これは，主にリアルタイム支援を行うことでコンテキストスイッチの回数が増えたことが理由だと考えられる．

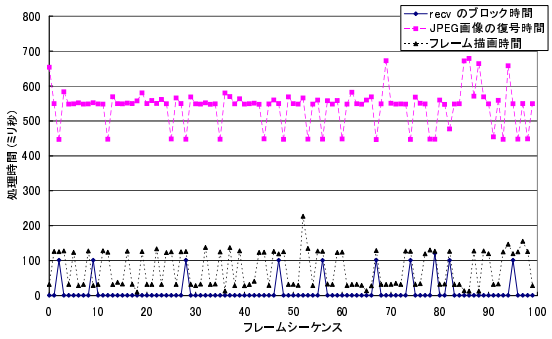

図 7: 実験 3 のスナップショット

干渉プロセスを動作させた実験 3 は平均フレーム処理時間が 638.7 ミリ秒となり，処理時間が実

験 1，2 の 6 倍近く増加している．このうちフレームの復号時間が　541.3 ミリ秒と大きく増加している．図 7 に示した実験 3 のスナップショットを見ると，フレーム復号時間とフレーム描画時間が，スケジューリングのタイミングによって大きく振動している様子がわかる．

実験 4 は，受信プロセスに CPU 資源予約を行いリソースセットを割り当てることで平均処理時間は 294.5 ミリ秒となり，実験 3 と比較して半減している．しかしながら，実験 4 を実験 2 と比較すると 3 倍近く処理時間がかかっている．実験 4 の処理時間の内訳を見ると，JPEG 画像の復号時間は 94.5 ミリ秒で，実験 2 の JPEG がぞの吹く号時間である 99.9 ミリ秒とほとんど変わらない．それに対し，実験 2 のフレーム描画時間が 12.6 ミリ秒であるのに対し，実験 4 のフレーム描画時間は 199.2 ミリ秒と大幅に増加している．受信プロセスに CPU 資源予約を行ったにもかかわらず，フレーム描画時間が増加したのは，受信プロセスはフレーム描画時に X Window System のワーカープロセスと協調動作を行うためである．

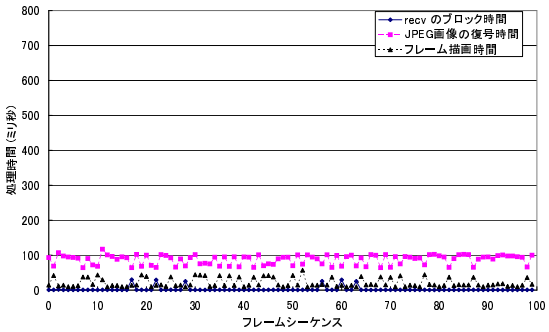

図 8: 実験 5 のスナップショット

実験 5 は，複数プロセスが資源予約を共有する例として，受信プロセスとフレーム描画部分の処理を行う X Window System にリソースセットを割り当てた．実験 5 のスナップショットを図 8 に示す．図 7 と比較すると，振動が抑制されフレームの各処理が短時間で終了していることが分かる．また，おなじ資源予約を共有しているため，画像の復号時間が短くなると相対的にフレーム描画時間が長くなる様子が分かる．実験 5 の平均フレーム処理時間は，110.9 ミリ秒であり，実験 2 の平均フレーム処理時間とほぼ同じ値となった．このことから，実験 5 では外乱プロセスの有無に影響せずフレーム処理が行われているといえる．

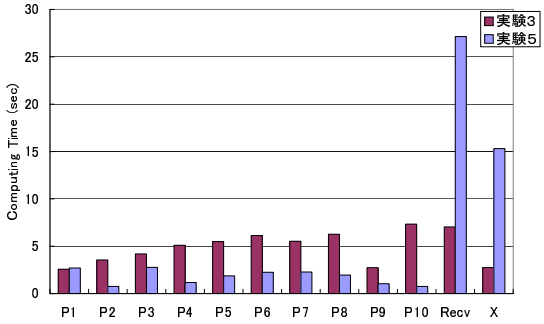

図 9: プロセス毎の計算時間

プロセス間のスケジューリングとオーバヘッド

実験 3 と実験 5 を比較し，プロセス間のスケジューリングがどのように行われているかを調査した．図 9 は，プロセス毎の 1 分間に割り当てられた計算時間である．P1 から P10 は外乱プロセス，Recv は受信プロロセス，X は X Window System のワーカプロセスをそれぞれあらわす．一分間の実行時間のうち実験 3 では，受信プロセスに 7.0 秒，X Windows System に 2.7 秒の計算時間が割り当てられた．それに対し実験 5 では，受信プロセスに 27.1 秒，X Windows System に 15.3 秒の計算時間が割り当てられている．実験 5 では，リソースセットを共有している 2 つのプロセスに対し CPU 資源予約の結果，42.4 秒間の計算時間が消費している．この結果，フレームの復号および描画がスムースに完了したと考えられる．

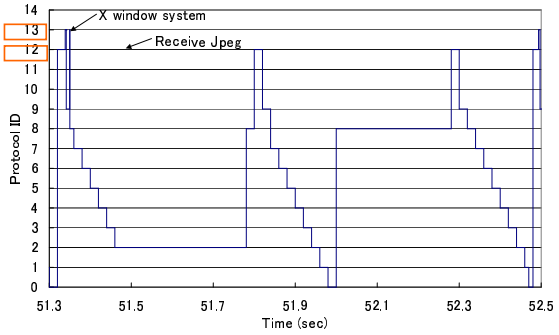

図 10: リアルタイム支援がない場合のコンテキストスイッチ

図 10，11 は，実験 3，5 におけるコンテキストスイッチのスナップショットである．Y 軸の 0 から 9 までは干渉プロセス，12 は受信プロセス，13 は X Window System のワーカプロセスをそれぞれあらわす．対象プロセスへの 1 秒間のコン

テキストスイッチは実験 3 が平均 28.4 回，実験 5 が平均 35.7 回であった．リアルタイム支援がある場合のオーバヘッドは主にこのコンテキストスイッチ回数が多くなることが原因だと考えられる．

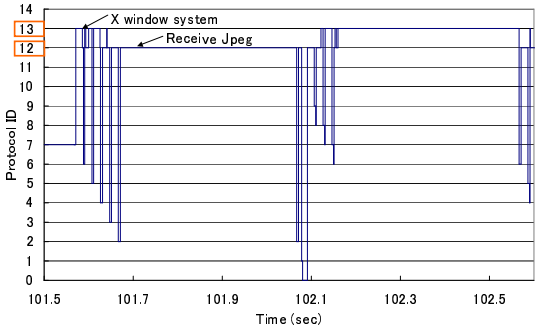

図 11: リアルタイム支援がある場合のコンテキストスイッチ

送信側の外乱の影響

送信側の外乱の影響を見るために，受信側を実験 5 の環境 (外乱プロセスあり，受信プロセス と X Window System に CPU 資源予約あり) に固定し，送信側に外乱プロセスおよび外乱フローを加えた場合について実験する．評価軸は，受信側の外乱の影響の実験と同様に受信側の平均フレーム処理時間とし，以下の 3 つの実験を行った．

- 実験 6: 外乱プロセスあり，外乱フローあり，資源予約なし
- 実験 7: 外乱プロセスあり，外乱フローあり，ネットワーク資源予約あり
- 実験 8: 外乱プロセスあり，外乱フローあり，CPU 資源予約あり，ネットワーク資源予約あり

外乱プロセスは，浮動小数点計算を繰り返すプロセスを 10 個動作させる．また外乱フローとして，2 Mbps の CBR/UDP フローを 5 本送信している．

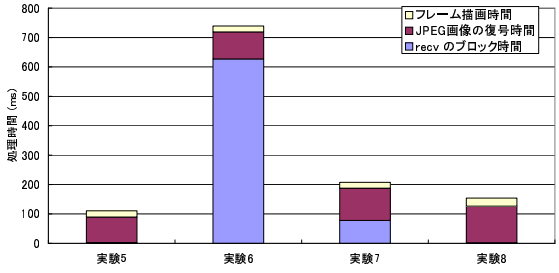

図 12: 実験 5-8: 受信側のフレーム毎の処理時間

実験 5 から 実験 8 までの受信側の平均フレーム処理時間を図 12 に示す．実験 6 を見ると，送信側の外乱プロセス，外乱フローにより，recv のブロック時間が大幅に増えているのが分かる．実験 2 の recv の平均ブロック時間は 2.7 ミリ秒であるのに対し，実験 6 では 平均 627 ミリ秒である．この実験 6 のスナップショットを図 13 に示す．JPEG の復号時間およびフレーム描画時間は，実験 5 とほぼ同じ時間に抑えられているのに対し，recv のブロック時間は大きく振動している様子がわかる．このことから、リアルタイム性の確保には受信側のみならず送信側の資源予約によるスムースな送信が必要だといえる．

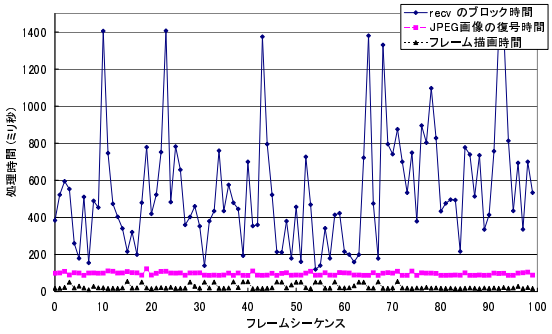

図 13: 実験 6 のスナップショット

次に，実験 7 を見るとネットワーク資源予約により recv のブロック時間が平均 78 ミリ秒と大分減少している様子がわかる．さらに CPU 資源予約を行った実験 8 では， recv のブロック時間は 1.9 ミリ秒となり実験 5 とほとんど変わらない値を達成している．これらの結果から，リアルタイム通信を行ううえで，外乱の発生しうる複数の資源に対して資源予約を実行することは有効であるといえる．

5 関連研究

Embedded Linux/RK と同様の組み込み用リアルタイム OS として，TRON (The Real-time Operating system Nucleas) [6] の組み込み用規格である ITORON がある．ITRON の仕様は無償で公開されているが，実装は公開の義務がないため，利用できるデバイスドラバやプロトコルスタックといったソフトウェア資源は Linux よりも少ない．また実時間周期スレッドのプリミティブもない．

Linux をベースとした組み込み用途に利用できるハードリアルタイム OS として，FSMLabs が開

発しているRTLinuxがある．Linuxのスケジューラの上位にリアルタイムスケジューラを実装し，マイクロ秒単位の実時間を保証している．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．

6 まとめと今後の課題

ハンドヘルドや組み込みシステムを対象とし，計算資源の限られた組み込み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Embedded Linux/RKを開発した．またEmbedded Linux/RK用いることで，計算資源を抽象化したリソースセットを用いた資源予約を実現した．評価として，送信側，受信側双方の外乱がストリーミングデータを無線通信で受け取りリアルタイム性を失わずにディスプレイに描画させる際にどのような影響を及ぼすかについての実験を行った．結果として，外乱プロセスや外乱フローにより著しくリアルタイム通信が悪化する環境でも，Embedded Linux/RK用い，複数の資源予約と資源予約の共有を行うことで一定の性能を保証できること示した．

今後の課題として，動画再生時にCPU資源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を行うために必要となるデッドラインハンドラの実装がある．また，今回の実験では，iPAQを用い評価を行ったが，センサノードといった更に小型な機器端末への実装をすすめる．

7 謝辞

本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」および，総務省「ユビキタスネットワーク制御・管理技術の研究開発（ubilaプロジェクト）」の一部として行われました．

参考文献

[1] Clifford W. Mercer and Stefan Savage and Hideyuki Tokuda, “Processor Capacity Reserves: Operating System Support for Multimedia Applications,” International Conference on Multimedia Computing and Systems, 1994.

[2] Masana Murase and Takeshi Iwamoto and Tomohiro Nagata and Nobuhiko Nishio and Hideyuki Tokuda, “Implementation and Evaluation of Wapplet Framework,” Proceedings of IEEE International Workshop on Networked Appliances(IWNA), 2002.

[3] Nobuhiko Nishio, Hideyuki Tokuda, “Design and an Implementation of Resource Reservation with QOS Profiling Handler,” The 8th International Conference on Real-Time Computing Systems and Applications (RTCSA 2002), March. 2002.

[4] S. Oikawa and R. Rajkumar, “Linux/RK: A portable Resource Kernel in Linux,” IEEE Real-Time System Symposium, 1998.

[5] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa, “Resource Kernels: A Resource-Centric Approach to Real-time Systems,” In *Proc. SPIE Conf. on Multimedia Computing and Networking*, January, 1998.

[6] K. Sakamura, “TRON - Total Architecthure,” 情報処理学会アーキテクチャワークショップ, 1984.

[7] Hideyuki Tokuda and Tatsuo Nakajima and Prithvi Rao, “Real-Time Mach: Towards a Predictable Real-Time System,” USENIX Mach Workshop, 1990.

[8] 永田智大, 村瀬正名, 西尾信彦, 徳田英幸, “Wapplet: ウェアラブルネットワークにおけるサービスフレームワーク”, 情報処理学会 DiCoMo: Multimedia, Distributed, Cooperative and Mobile Symposium, 2000 年 6 月.

[9] 永田 智大, 西尾 信彦, 徳田英幸, “サービス利用状況の変化に対する適応支援機構”, 情報処理学会論文誌, Vol.44 (3) 2003 年 3 月.

[10] 西尾 信彦, 徳田 英幸: “ゆReserve アーキテクチャ: 統合的資源予約機構,” **情報処理学会論文誌**, Vol.40, No.6, , 1999.

[11] 安田泰勲, 稲村浩, “適応的アプリケーションのための資源管理機構および資源量通知機構を持つフレームワーク S-MAX の実装”, 情報処理学会 OS 研究会, 1999.