Vol.2013-SLDM-160 No.4
Vol.2013-EMB-28 No.4
2013/3/13

RTOSからLinuxへのマイグレーションおよびRTOS教育のためのRTOS on POSIXの開発

清水 尚彦[1,2,a)]

概要：本報告は、RTOS の Linux へのマイグレーションや、RTOS 教育を目的として、POSIX レイヤー上にタスクスイッチャーを構築した RTOS on POSIX の開発について述べる。RTOS レイヤーには、Toppers プロジェクトの ToppersATK1 を採用し、i386 用のタスクスイッチコードおよび POSIX レイヤーとの接続コードを新たに開発した。OSEK において必須となっているにもかかわらず、ATK1 に不足するメッセージ通信については、新たに COM/CCCA コードを作成し、提供した。POSIX レイヤーとして Linux および Cygwin を用いて実働を確認した。

1. はじめに

組込みシステムの OS として Linux を採用する例が増えている [4]。しかし、小型の ECU では Linux は大きすぎて利用しにくい。そのため、小型の RTOS と Linux は適材適所で住み分けが可能である。一方、当初、RTOS で開発をスタートしたシステムが大型化し、ネットワーク対応など Linux にマイグレーションするニーズがあるが、API の違いが大きく、アプリケーションの構造からの再構築となることも少なくない。さらに、大学等の機関において、RTOS の教育を行う場合、組込みシステムを用意できるケースは少ない上、計算機上に新たなソフトウェアのインストールを行うことが、事務処理上困難な場合も多い。

本報告は、RTOS の Linux へのマイグレーションや、RTOS 教育を目的として、POSIX レイヤー上にマイクロスケジューラとなるタスクスイッチャーを構築した RTOS on POSIX の開発について述べる。教育用途のため、RTOS は学生が無料で利用可能であること、国際標準に準拠していること、小型であることなどを勘案し、OSEK/VDX[9](以下、単に OSEK と略す) を採用することにした。OSEK のフリーソフトによる実装は複数存在するが、内外の OSEK 実装を比較し、受講生の学習のため、ソースコード内に日本語のコメントがある Toppers プロジェクト ATK1 を採用することした [17]。ところが、ATK1 には OSEK OS では必須となっているメッセージ通信 (COM/CCCA 以上) が実装されていない。そのため、新たに COM/CCCA のコードも開発し、OSEK 学習用として最低限の内容を整えた。また、POSIX レイヤーとして、Windows 上で動作する Cygwin でも動作させた。さらに、PC にインストールすることなく、ファイルを展開するだけで Cygwin プログラムの利用が可能な LiveCygwin[16] を併用することで、プログラムのインストールが許されない計算機環境においても演習が可能となる環境を開発した。

[1] 東海大学
Tokai University, Minato, Tokyo 108–8619, Japan
[2] IP ARCH, Inc.
Hawaii, USA
[a)] nshimizu@keyaki.cc.u-tokai.ac.jp

2. 既存研究

OS の上に OS を実装する手法としては、ハイブリッド OS として、RTOS 上に Linux を実装する研究が行われている [5], [18]。Linux 本体も組込み用途向けのリアルタイム機能の強化が行われ、特に Ingo Molnar が開発し、2.6 系カーネルに採用された O(1) スケジューラ [14] には、プライオリティベースのリアルタイムタスクが実装され、リアルタイム性が強化された結果として、組込み OS として Linux を採用する実績が増えている。Linux の柔軟性は、大学における教育目的としても有効であり、実世界とコンピュータを連携させる組込み分野の利用においても進んでいる [6], [7]。

しかし、Linux の API と従来組込み分野で用いられてきた RTOS の API は大きく異なり、学習者もしくはアプリケーションの移植者への負担は小さくない。

また、数多くの開発者が混在し、それぞれが独自の実装を行うデバイスドライバーでは、リアルタイム性が損なわれる実装が残っており、Linux だけではハードリアルタイムに対応するのは難しい [1]。そのため、Lee らは、割込みの予測可能性に着目し、割込み応答性を高めている [11], [12]

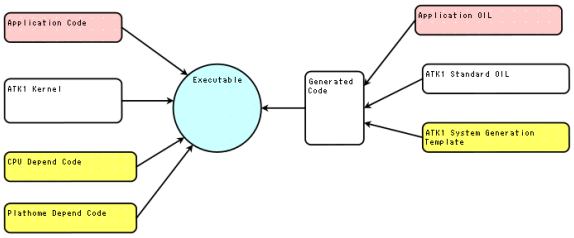

図 1 ATK1 によるアプリケーション作成

このように、Linux の対象とする領域は広がりを見せているが、超小型の ECU やハードリアルタイム領域における適用は未だに難しいといえよう。

一方、組込み技術者の教育はわが国ならずとも重要性を増している。小型 ECU 向けの RTOS を搭載した組込みシステムを教育することを考えるとき、実際の ECU やマイコンの評価キットをベースとした開発環境を整えることは、教育側/受講側双方の負担が非常に大きいため、玩具として簡単に入手可能な LEGO 社の LEGO MindStorms[13] を用いる例が増えている [2], [3], [10] 組込みシステムの開発困難さの主たる原因のひとつに、プロセスの観測性の低さが挙げられよう。Linux などの汎用 OS 上に構築された、アプリケーションは、汎用 OS 向けの広範な開発支援ソフトウェア群を無償で用いることができる。ところが、評価キットなどの組込みシステムでは、開発支援ソフトウェアは一般に有償もしくは機能限定されている評価版を使うことになり、教育用途としては、その観測性も十分満足できるものではない。そこで、実機による開発に加えて、シミュレータを用いた早期開発の試みが行われている。Röblitz らは、POSIX スレッド上に LEGO MindStorms 向け OS である LegOS を実装し、シミュレーションが行えるようにした [15]。彼らのターゲットは現在販売が中止されている旧世代の MindStorms を対象にしていてる。また、ターゲットの RTOS のタスクを Pthread に写像するため、ベースの OS におけるプライオリティ制御とターゲット RTOS との不一致がある場合、タスクの動きを正しくシミュレーションできない可能性がある。

3. 実装

本研究では、Linux などの POSIX 互換 API を有する OS のプロセス上に、マイクロカーネルを実行し、タスク切り替えをひとつのプロセス内で行う RTOS on POSIX を開発・実装した。ターゲットとする RTOS には、Toppers ATK1（以下 ATK1）を採用した。OSEK OS は API を提供する関数群であり、構成制御である OIL ファイルから生成するコードと OS、アプリケーションのコードをコンパイル・リンクしてアプリケーションを作成する。ATK1 は、ユーザ OIL と、システムの OIL 定義の OSEK 仕様に掲載される構成制御の他にプラットホームに合わせた構成制御テンプレートファイルを必要とする。図 1 に ATK1 を用いるアプリケーションの構成を示す。アプリケーション開発者は、アプリケーションの C ソースコードならびに構成制御ファイルであるアプリケーション OIL ファイルを作成する。ATK1 プロジェクトは、ATK1 カーネルソースコードならびに ATK1 標準 OIL ファイルを提供する。また、システムのプラットフォーム依存部分、アーキテクチャ依存部分はカーネルコードと分離し、階層的に提供することになっており、新しいアーキテクチャ・プラットフォームへの移植者は、ATK1 を移植するにあたり、下記のコードを新たに開発する必要がある。

(1) アーキテクチャ依存コード
(2) プラットホーム依存コード
(3) 構成制御テンプレート

これらの移植のためのコード群に加えて、OSEK OS では必須となっていながら、ATK1 には実装されていない、メッセージ通信のための通信パッケージのコードを新たに開発した。

本実装は、マイクロカーネルによるタスクスイッチを、ひとつのプロセス内で実施するため、Röblitz らの方法で問題となったようなシミュレータと実 OS での制御の不一致の問題が発生しない。一方、タスクスイッチのため、CPU のアセンブラレベルでのスタック切り替えルーチンを開発する必要があり、CPU 依存性が高い。そこで、プラットホームとして PC を用いることを前提に、Intel 社 i386 アーキテクチャをターゲットとする。なお、同社 64 ビットアーキテクチャで動作する OS も 32 ビットアプリケーションの実行が可能なため、64 ビット版の Linux においても、本実装は動作することを確認している。本研究の目的のひとつである教育用ととして用いる場合、全ソースコードを容易に読解可能であることが要求される。そこで、純粋なアセンブラではなく、C 言語のインラインアセンブラにより、タスク切り替えコードを記述した。

以下、各実装について説明する。

表 1 アーキテクチャ依存コード C ファイル

ファイル名	行数	備考
cpu_config.c	74	割込みレベル擬似処理
cpu_support.c	104	スタック切り替えなど
cpu_config.h	101	ヘッダファイル
cpu_context.h	74	ヘッダファイル
cpu_defs.h	68	ヘッダファイル
cpu_insn.h	54	ヘッダファイル
tool_config.h	116	ヘッダファイル
tool_defs.h	82	ヘッダファイル
合計	673	

3.1 アーキテクチャ依存コード

アーキテクチャ依存コードとして、表 1 の 2 本の C ファイルを作成した。スタックの切り替えやレジスタの扱い

```
void i386_support(void) {

asm("_activate_r:\n");
asm("activate_r:\n");
    enable_int();
    tcb_curpri[runtsk] = tinib_exepri[runtsk];
    asm(    "jmp *%0\n": :"m"(tinib_task[runtsk]));


asm( "_exit_and_dispatch:\n");
asm( "exit_and_dispatch:\n");
    if(PostTaskHook!=NULL) call_posttaskhook();

asm( "_start_dispatch:\n");
asm( "start_dispatch:\n");

loop:
    runtsk = schedtsk;
    if(runtsk==INVALID_TASK) goto idle;
    asm(    "movl %0, %%esp\n" ::"m"(tcxb_sp[runtsk]));
    if(PreTaskHook!=NULL) call_pretaskhook();
    asm(    "jmp *%0\n": :"m"(tcxb_pc[runtsk]));
idle:
    asm(    "movl %0, %%esp\n" ::"m"(system_stack));
    callevel = TCL_ISR2;
idle_loop:
    enable_int();
    halt_cpu();
    disable_int();
    if(schedtsk == INVALID_TASK) goto idle_loop;
    callevel = TCL_TASK;
    goto loop;
}
```

図 2 アーキテクチャ依存部におけるスタック切り替えコード

```
void __attribute((regparm(2)))
  interrupt(IsrType irq, FP isr)
 {
    asm(    "pushl %%ebp\n"
        "movl %%esp, %%ebp\n"
        "cmpb %4,%2\n"
        "jne  1f\n"
        "movl %5, %%esp\n"
        "1:\n"
        "movb %2, %%ch\n"
        "movb %3, %%cl\n"
        "pushl %%ecx\n"
        :"=a"(irq),"=d"(isr):"m"(callevel),
        "m"(runisr),"i"(TCL_TASK),
        "m"(system_stack),"0"(irq),"1"(isr));
    callevel=TCL_ISR2;
    runisr = irq;
    enable_int();
    (isr)();
    disable_int();
    asm (   "popl %%eax\n"
        "movb %%al,%0\n"
        "movb %%ah,%1\n"
        "movl %%ebp, %%esp\n"
        "popl %%ebp\n"
        :"=m"(runisr),"=m"(callevel));
    if(callevel!=TCL_TASK) return;
    asm("movl $1f, %0\n" :"=m"(tcxb_pc[runtsk]));
    if(schedtsk==runtsk) return;
    dispatch();
    asm( "1:\n");
    return;
}
```

図 3 ISR ディスパッチコード

は、C 言語だけでは実現できないため、GNU C コンパイラの持つインラインアセンブラの機能を用いた。カーネルソースコードから呼び出される関数は、スタック切り替え処理においては、C の関数として実現することが困難である。そこで、インラインアセンブラ中にグローバルなラベルを挿入して、C 言語からは関数として呼び出しながら、インラインアセンブラの途中に分岐ポイントを設定することとした。

図 2 は、アーキテクチャ依存部のスタック切り替えコードを示す。この図に示すように、PC の切り替えやスタックポインタの切り替えなど、最低限の部分のみをインラインアセンブラで記述し、できるだけ多くの部分を C 言語で記述することで、可読性をあげることができた。なお、図 2 において、ほぼ同一名の複数のラベルを併記しているのは、コンパイラに依存し、ラベル名が変化することに対応させるものである。

一方、割り込みにかかる部分は、スタック操作のため、割込みエントリと ISR ディスパッチ関数の呼び出しをレジスタ渡しにした。その結果、C 言語では引数のコントロールが難しく、引数を受け渡す部分にインラインアセンブラを用いることとした。ISR ディスパッチ関数を図 3 に示す。

表 2 プラットホーム依存コード C ファイル

ファイル名	行数	備考
sys_config.c	72	割込みレベル擬似処理
sys_support.c	232	システム固有処理
sys_config.h	55	ヘッダファイル
sys_defs.h	54	ヘッダファイル
合計	423	

i386 アーキテクチャには、割込みによって特権レベルが変更されるときにタスク状態セグメントから新しいスタックポインタをロードする仕組みがある [8]。Linux など、汎用 OS では、この仕組みを用いて、スムースにスタック切り替えを行うが、本実装は、ユーザモードですべての処理を行っているため、タスク状態セグメントは用いず、直接スタックポインタを変更する。

3.2 プラットホーム依存コード

プラットホーム依存コードの C ファイルを表 2 に示す。POSIX のシグナルを割込みとして扱うために、割込みレベルとシグナルとの対応を行う関数群を集めている。プラットホーム依存部分に POSIX 固有の処理を集めているため、この部分を書き換えることで、POSIX 以外のプラットホー

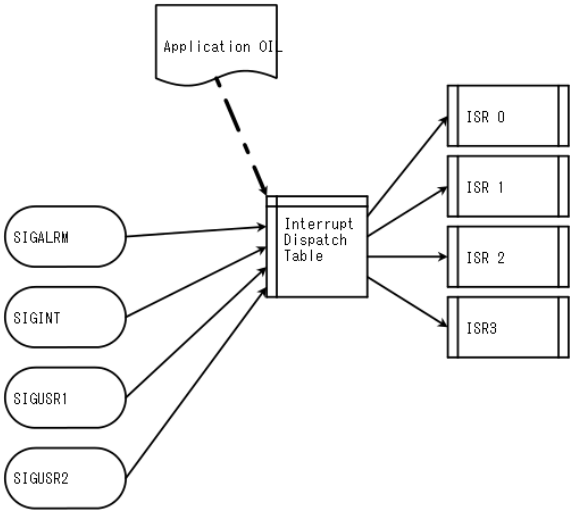

図 4 割込みルーチンディスパッチ

表 3 擬似割込みに対応するシグナル

シグナル名	エントリ番号	説明
SIGALRM	0	インターバルタイマ
SIGINT	1	キーボード割込み
SIGUSR1	2	ユーザ定義シグナル
SIGUSR2	3	ユーザ定義シグナル
SIGPOLL	4	key polling

ムに対する移植は比較的容易である。

OSEK を POSIX 上に実装する上で、割込みのプライオリティが問題となった。POSIX シグナルはプライオリティを持たないため、プライオリティを擬似的に実現する必要があった。そこで、割込みのプライオリティレベルを設定すると、有効な割込みに相当するシグナル以外のシグナルをマスクすることでプライオリティの擬似処理を実現した。[*1]

Linux では、リアルタイムシグナルを利用することで、シグナルのキューイングが実現できるが、本実装では、Linux 以外もターゲットとしているので、用意したコードではユーザから使いやすい 5 つのシグナル（SIGALRM, SIGINT, SIGUSR1, SIGUSR2, SIGPOLL）のみを扱う。表 3 に 5 つのシグナルの概要を示す[*2]。シグナルは、POSIX ではプロセス外部から kill コマンドにより送ることができるが、それのみならず、内部から raise システムコールでシグナルを発生させることができる。シグナルと ISR の対応関係は OIL で記述するが、すべてのシグナルに対してディスパッチ表を作成するのは記憶領域の無駄であり、対象シグナルだけを集めたディスパッチ表を経由して ISR の起動を行っている。そこで、ディスパッチ表のエントリ番号とシグナルは一対一に対応させ、OIL でエントリ番号を指定するようにした。表のエントリ番号は、構成制御ファイル OIL の ISR 定義に記載する番号である。

図 4 に割込みルーチンディスパッチの様子を示す。アプリケーション OIL が構成制御ソフト SG によって処理されるとき、後述の構成制御テンプレートに従い、ディスパッチテーブルを初期化する。初期化を終えたディスパッチテーブルは、割込みシグナルが発生したときに、その割込みレベルおよび割込みサービスルーチンを出力する。デバッグのため、OIL に定義されないシグナルが入力された場合、メッセージを出力後プログラムを終了している。

OSEK に要求される周期タイマはインターバルタイマによる周期シグナルによって実現する。このインターバルタイマは、OS スタート前に起動する必要があるが、アプリケーション設計者に分かりやすいように、タイマ起動の関数をプラットホーム依存コードとして提供している。これ以外にもサービスルーチンとして提供する関数群を表 4 に示す。これらの関数群は、後述の教育パッケージのための例題を作成する際に必要性を感じたものを順次実装した。

表 4 プラットホーム依存サービスルーチン

関数名	説明
sys_start_timer	インターバルタイマ周期設定。二つの整数引数を取り、秒、ナノ秒の周期時間を指定する
halt_cpu	CPU 停止。一般の OS ではホルト命令の実行はブロックされるため、長時間のスリープでエミュレートする
sys_raise	シグナル発行。シグナルはディスパッチテーブルに対応して 0 から 3 までの値を与える
sys_exit	システム終了。キーボード入力を OS 内で扱うため、端末属性を変更しているので、端末属性を標準に戻す作業をしたのち終了する
sys_kbhit	非ブロックキーボードチェック

3.3 構成制御テンプレート

ATK1 では、アプリケーション OIL を処理して構成制御ファイルを作成するときに、構成制御テンプレートに記載したマクロを展開する。そこで、本実装に合わせてテンプレートを作成した。本実装では、割込みのディスパッチテーブルをマクロで記述し、割込みエントリごとの割込みサービスルーチンを記述する配列を初期化する。また、OSEK は ISR1 と ISR2 と呼ぶ二つの割込みサービスルーチンを提供するので、割込みタイプを記述する配列を用意し、同様に構成制御テンプレートによるマクロ展開によって初期化する。

*1 Cygwin 上では、インターバルタイバ割込みをマスクしても、他の処理の間に再びマスクが解除される事例が発生しており、インターバールタイマの処理には注意が必要となる。もっとも、OSEK OS では、最低 1 つの周期タイマを前提とするので、インターバルタイマ割込みをマスクする必要はなく、その場合には、タイマ割込みのマスクについては問題とならない。
*2 ただし、Cygwin は SIGPOLL を提供しない

表 5　COM パッケージコード C ファイル

ファイル名	行数	備考
com.c	116	通信パッケージ
com.h	85	COM ヘッダファイル
com_ext.h	194	内部実装プロトタイプ宣言
osek_com.h	32	システム型宣言
合計	427	

表 6　COM API 関数

関数名	備考
SendMessage	メッセージの送信を行う
RecieveMessage	メッセージの受信を行う
StartCOM	メッセージ通信を開始する
StopCOM	メッセージ通信を停止する
GetCOMApplicationMode	COM のアプリケーションモードを取得する
InitMessage	メッセージを初期化する

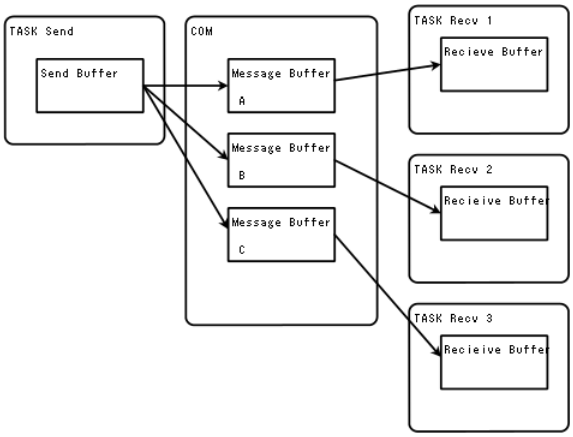

図 5　OSEK COM のメッセージ同報通信

3.4　メッセージ通信パッケージ

OSEK OS では、メッセージ通信は必須となっていて、最低限 COM に定義される CCCA クラスを実装する必要がある。OSEK COM では、複数のメッセージ通信クラスを定義しているが、本実装は、その中でもっともサポート範囲の狭い CCCA のみを対象とした。CCCA は同一 ECU 内の同期通信のみを提供する。表 5 に開発したコードの一覧を示す。ATK1 にはメッセージ通信は実装されていないが、OSEK 仕様では COM は OS と独立の仕様となっており、OSEK OS を変更することなく COM の実装を行った。実装した関数を表 6 に示す。

COM CCCA のメッセージ通信の様子を図 5 に示す。CCCA では、メッセージは送信タスクに同期して送信される。送信側は、SendMessage 関数を呼び出し、送信バッファ内の情報を COM の受信バッファに送る。このとき、構成制御ファイル OIL により同報通信が指定されていたら、複数の受信バッファにデータを転送する。受信タスクは、イベントやタスク起床など、OIL ファイルに記載された方法で、受信の通知を受ける。受信通知を受けた受信タスクは、RecieveMessage 関数により、自タスクの領域にデータをコピーする。

```
#include "kernel.h"
#include "kernel_id.h"
#include <stdio.h>

TASK(OSEK_Task_Background)
{
        printf("OSEK HelloWorld!\n");
        ShutdownOS(E_OK);
}

void ShutdownHook(StatusType ercd) {
        printf("OSEK Good Bye!\n");
}

int main(int argc, char *argv[]) {

        StartOS(appmode1);
}
```

図 6　例題コード (HelloWorld)

異なるプライオリティのタスクに対して同報通信を行う場合、メッセージにより起床されるタスクもしくはイベントなどは、優先順位順に起床する必要があるが、OS 内部に踏み込まずに優先順位を取り出す方法はないため、構成制御においてメッセージのあて先を優先順位順にソートした状態でメッセージテーブルが作成され、本パッケージが呼ばれることを前提としている。同報通信の場合には、メッセージの転送を全ターゲットに送出後、各受信側の起床を順次行う。

4.　教育用パッケージ

ATK1 の提供は、OS ソースコードをまとめて圧縮した形態で行われる。そこで、教育目的に用いる場合には、開発環境の構築が不可欠である。Cygwin 環境下で LEOG MindStorms 向けの開発環境としてまとめられた nxtOSEK[3] を参考に、教育向け開発環境を開発した。また、この開発環境下で教育コンテンツとしてサンプルコードを提供し、容易に技術者向け教育ができるようにした。教育用パッケージとして nxtOSEK と異なる特徴を下記に示す。

(1) タイマ割込みを含むすべての割込み（シグナル）処理を OIL から登録可能とした
(2) アプリケーションが明示的に StartOS を起動する
(3) Shutdown はアプリケーション終了とする
(4) OSEK OS の主要 API に対するサンプルコードを作成した

このパッケージにより、実際の組込みシステムの構築方法に近い形で OSEK OS の教育が可能となり、学習効果が高いと考える。

図 6 に本パッケージに用意したサンプルプログラムの一つを示す。一般の組込みソフトウェアと異なり、POSIX

表 7 タスク切り替え時間比較（単位は μS）

OSEK	Linux
2.59	4.72

表 8 ISR・タスク起床時間（単位は μS）

ISR	TASK
4.39	5.52

上に実装した本パッケージのプログラムは、通常のアプリケーションとしてコンソールから起動するため、main ルーチンより StartOS を呼び出す。COM パッケージを用いる場合、StartupHook などから StartCOM を起動することになる。

5. 性能評価

OSEK のイベント待ちによるタスク切り替え時間を POSIX スレッドのセマフォ待ちによるタスク切り替えと比較した。

実験は、Linux カーネル 2.6.32 を実行する Linux ワークステーション（PentiumIII 1130MHz）で行い、プライオリティ 10 のリアルタイムタスクとして、他タスクよりも優先順位を上げ、100,000 回実行した平均値を示す。

表 7 に示すように、本環境による OSEK タスク切り替えは Linux pthread のタスク切り替えよりも高速である。

表 8 に、本実装におけるシグナル送信から ISR 起動までの時間と、ISR がタスクをアクティベートするまでの時間を示した。計測環境は前述の通りである。シグナルを発行してから、ISR を経由してタスクを起床するまで約 $10\mu S$ となっている。この例では、周期タイマによって起動される ISR がシグナルを発行し、他の ISR を起動したが、組込みシステムで利用する場合、デバイスドライバから直接シグナルを発行することになるため、さらに必要な時間は短縮されると考えられる。

6. まとめ

本報告は、教育もしくは RTOS アプリケーションの移植のための RTOS on POSIX について述べた。PC を用いることで、RTOS アプリケーションの教育が比較的容易に行えること、RTOS API を Linux 上に実現することで、RTOS アプリケーションの Linux への移植が容易となる効果が期待できる。性能評価で示したように、Linux 上にスレッドを作成する場合に比べてタスク切り替え速度が早く、RTOS からのアプリケーションの移植に有効な手法といえよう。

今後の課題として、デバイスドライバからのシグナル送信の性能検証を進めることと、i386 以外のアーキテクチャにおける検証を進め、本手法の有効性を示していきたい。

参考文献

[1] Abeni, L., Goel, A., Krasic, C., Snow, J. and Walpole, J.: A measurement-based analysis of the real-time performance of linux, *Real-Time and Embedded Technology and Applications Symposium, 2002. Proceedings. Eighth IEEE*, pp. 133 – 142 (online), DOI: 10.1109/RTTAS.2002.1137388 (2002).

[2] Bradley, P. J., de la Puente, J. A. and Zamorano, J.: Real-time system development in ada using LEGO mindstorms NXT, *Ada Lett.*, Vol. 30, No. 3, pp. 37–40 (online), DOI: 10.1145/1879097.1879077 (2010).

[3] Chikamasa, T.: nxtOSEK Project, (online), available from ⟨http://lejos-osek.sourceforge.net/⟩ (accessed 2013-01-30).

[4] Electronics, U.: 2012 Embedded Market Survey, Technical report, UBM Tech (2012).

[5] Fu, K.: RTLinux: an interview with Victor Yodaiken, *Crossroads*, Vol. 6, No. 1, pp. 0.09– (online), DOI: 10.1145/331636.331647 (1999).

[6] Gifford, K. K.: Linux in Education: Linux at the University, *Linux J.*, Vol. 2000, No. 77es (online), available from ⟨http://dl.acm.org/citation.cfm?id=350314.350365⟩ (2000).

[7] Huang, Y.-L. and Hu, J.-S.: Hands-on oriented curriculum and laboratory development for embedded system design, *SIGBED Rev.*, Vol. 6, No. 1, pp. 3:1–3:8 (online), DOI: 10.1145/1534480.1534483 (2009).

[8] Intel: *INTEL 80386 Programmer's Reference Manual* (1986).

[9] ISO: ISO 17356-3 OSEK OS (2005).

[10] JESA: ET Robot Contest, Japan Embedded Systems Technology Association (online), available from ⟨http://www.etrobo.jp/⟩ (accessed 2013-01-30).

[11] Lee, J. and ho Park, K.: Delayed locking technique for improving real-time performance of embedded Linux by prediction of timer interrupt, *Real Time and Embedded Technology and Applications Symposium, 2005. RTAS 2005. 11th IEEE*, pp. 487 – 496 (online), DOI: 10.1109/RTAS.2005.16 (2005).

[12] Lee, J. and Park, K. H.: Prediction-Based Micro-Scheduler: Toward Responsive Scheduling of General-Purpose Operating Systems., *IEEE Trans. Computers*, Vol. 58, No. 5, pp. 648–661 (online), available from ⟨http://dblp.uni-trier.de/db/journals/tc/tc58.html#LeeP09⟩ (2009).

[13] LEGO: LEGO MINDSTORMS Hardware Developer Kit (HDK), LEGO Group. (online), available from ⟨http://mindstorms.lego.com⟩ (accessed 2013-01-30).

[14] Molnar, I.: O(1) Scheduler, RedHat (online), available from ⟨http://people.redhat.com/mingo/O(1)-scheduler/README⟩ (accessed 2013-01-30).

[15] Röblitz, T., Mueller, F. and Bühn, O.: LegoSim: simulation of embedded kernels over Pthreads, *J. Educ. Resour. Comput.*, Vol. 2, No. 1, pp. 117–130 (online), DOI: 10.1145/545197.545203 (2002).

[16] Shimizu, N.: LiveCygwin, IP ARCH, Inc. (online), available from ⟨http://www.ip-arch.jp⟩ (accessed 2013-01-30).

[17] TOPPERS: TOPPERS Project/ATK1, The TOPPERS Project (online), available from ⟨http://www.toppers.jp/atk1.html⟩ (accessed 2013-01-30).

[18] 金田一勉：Linux on ITRON ―ハイブリッド構造の実装，『Interface』2002 年 7 月号別冊付 (2002).