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

2007－MBL－40 (29)
2007－UBI－13 (29)
2007／2／23

無線センサノード向けハードリアルタイムオペレーティングシステムの設計

猿渡 俊介† 水野浩太郎† 鈴木 誠† 森川 博之†

† 東京大学大学院新領域創成科学研究科 〒113-8656 東京都文京区本郷7-3-1
E-mail: †{saru,mizuno,suzuki,mori}@mlab.t.u-tokyo.ac.jp

あらまし 無線センサネットワークは，来たるべきユビキタスコンピューティング環境における神経系としての技術として注目されている．これに向けては，多様なアプリケーションをサポートすることが可能な無線センサネットワークの基盤技術が必須である．このような観点から，無線センサノード向けのハードリアルタイムオペレーティングシステム PAVENET OS の開発を行った．PAVENET OS は TinyOS がサポートしていないハードリアルタイム処理をサポートする．また，スレッドを用いて OS を実現することでイベントを用いた TinyOS に比べてプログラムが書きやすいという特徴も持つ．さらに，無線センサネットワーク向けの無線プロトコルスタックを提供することでユーザがルーティングプロトコルや MAC プロトコルを開発しやすい環境を提供している．本稿では PAVENET OS の設計と初期的な評価について述べる．

Designing Hard Realtime Operating System for Wireless Sensor Nodes

Shunsuke SARUWATARI†, Kotaro MIZUNO†, Makoto SUZUKI†, and Hiroyuki MORIKAWA†

† Graduate School of Frontier Sciences, The University of Tokyo
Hongo 7-3-1, Bunkyo-ku, Tokyo, 113-8656 Japan
E-mail: †{saru,mizuno,suzuki,mori}@mlab.t.u-tokyo.ac.jp

Abstract Wireless sensor networks are attracting attention as a key technology in the coming ubiquitous computing environment. We need a testbed that meets various demands to develop the wireless sensor networks. Therefore, we have designed and implemented operating system for wireless sensor nodes called PAVENET OS. PAVENET OS supports hard realtime operation that is not supported in TinyOS. Additionally, PAVENET OS's programability is superior to TinyOS ' s, because PAVENET OS is designed using threads instead of events. PAVENET OS has also the wireless protocol stack that enables us to develop various routing protocols and media access protocols. This paper shows the design and preliminary evaluation of PAVENET OS.

1. はじめに

無線センサネットワークは小型，数が膨大，低消費電力という特徴を持っているため，今まで取得することができなかった粒度で空間の情報を取得することが可能になる．そのため，無線センサネットワークはビルオートメーションや自然科学，農業，軍事，品質管理などさまざまな分野への応用が期待されている．このような無線センサネットワークの多様な分野への応用を促進するためには，多様なアプリケーションや無線通信プロトコルを簡単に開発するためにオペレーティングシステムなどの基盤技術の整備が必須である．

現在，無線センサネットワークのオペレーティングシステムとして TinyOS [1] が標準として扱われている．筆者らは 2002 年から無線センサネットワークの研究を進めており，そのとき既に標準になりつつあった TinyOS をわれわれの研究に用いることを検討した．しかしながら，TinyOS が event model を用いているためにプログラムが書き辛いこと，ハードリアルタイム性をサポートしていないことの 2 点の理由から筆者らの研究の要求に合致しなかったため，独自にオペレーティングシステムを開発することにした．

このような背景から開発されたオペレーティングシステムが本稿で述べる PAVENET OS である．PAVENET

OSは，thread modelを用いることでユーザに対してプログラムの書きやすさとハードリアルタイム処理のサポートの2つを提供する．PAVENET OSはpre-emptiveとco-operativeのハイブリッドのスケジューラを具備している．pre-emptiveのスケジューラではCPUの動的な優先度割り込みの機能とハードウェアによるコンテキストスイッチの機能を利用することで少ないオーバヘッドでハードリアルタイム処理を実現する．co-operativeのスケジューラでは，os_yieldやsleepなどユーザが明示的にCPUを開放する関数を実行することで排他制御やコンテキストスイッチの負荷を軽減する．さらに，無線センサノードにおけるタスク間のデータの交換が無線通信における各層のパケットの交換時に多発することに着目し，オペレーティングシステムとして無線通信プロトコルの階層化の機能を提供して排他制御をAPIに隠蔽することで各層のモジュール性を実現する．

PAVENET OSの初期的な評価としてハードリアルタイム処理の観点でTinyOSと比較を行った．具体的には，正確な100Hzでの加速度のサンプリング処理と無線通信の処理を並列に実行させた．その結果，TinyOSでは正確な100Hzでのサンプリングが実現できなかったのに対し，PAVENET OSでは無線通信の処理を並列に行う場合でも正確な100Hzの計測が実現できた．

本稿ではまず，2.においてTinyOSの使い辛い点とその原因を示し，本研究のモチベーションを明らかにする．それを受け，3.ではPAVENET OSの設計と実装について述べる．4.では，PAVENET OSの初期的評価について述べる．5.で関連研究との違いを明らかにし，最後に6.でまとめとする．

2. Why TinyOS Is A Bad Idea?

2.1 TinyOSの問題点

無線センサネットワークは，無線センサノードを空間に散布し，無線センサノード同士が通信を行いながら膨大な量の空間情報を取得するための技術である．無線センサノードはセンサからセンサ情報を取得，解析，センサノード同士でホップバイホップで基地局（シンクノード）までルーティング，などの処理を行う．さらに無線センサネットワークではアプリケーションに応じて通信に求められる要件が大幅に変わってくるため，未だMACプロトコルに関する試行錯誤が続いている[2]〜[9]つまり，無線センサノードではソフトウェアでアプリケーションから通信プロトコルまでの機能を効率よく開発し，評価するための使いやすいオペレーティングシステムが求められている．

このような背景の中で，本研究のスタート地点は「TinyOSで本当にいいのだろうか?」という疑問であった．筆者らは2002年から無線センサネットワークの研究を進めてきた．当時すでにカリフォルニア大学のバークレー校において開発されたMICA mote[10]上で動作するnesC[11]で作成されたTinyOS[1]がセンサネットワークの分野の標準として扱われていた．筆者らも当初，研究の基盤としてTinyOSを使用することを検討したが，検討していく中で「なぜTinyOSを皆使っているのだろうか?」という疑問が生まれた．その理由は2つある．

1つ目の理由はTinyOSではプログラムが非常に書き辛いということである．TinyOS上で1秒おきにLEDを点滅させるプログラムを以下に示す．

```
[Blink.nc]

configuration Blink {
}implementation {
  components Main, BlinkM, SingleTimer, LedsC;
  Main.StdControl -> BlinkM.StdControl;
  Main.StdControl -> SingleTimer.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}

[BlinkM.nc]

module BlinkM {
    interface StdControl;
  }
  uses {
    interface Timer;
    interface Leds;
  }
}

implementation {
  command result_t StdControl.init() {
    call Leds.init();
    return SUCCESS;
  }

  command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, 1000);
  }
  command result_t StdControl.stop() {
    return call Timer.stop();
  }
  event result_t Timer.fired()  {
    call Leds.redToggle();
    return SUCCESS;
  }
}
```

TinyOSではLEDを点滅させるだけで，このような非常に冗長なコードをかかなければならない．

2つ目の理由はTinyOSではハードリアルタイム処理が実現できないことである．TinyOSでは[1]において無線センサネットワークではソフトリアルタイム処理ができれば十分に対応できると述べている．しかしながら，筆者らの経験ではソフトリアルタイム処理だけでは十分ではない．筆者らはこれまで無線センサネットワークにおける無線通信プロトコルからアプリケーションまでを統合的な観点から研究を進めてきた[12]〜[16]．その中で，たしかにソフトリアルタイム処理だけでも実現できるアプリケーションも存在した．その一方で，MACプロトコルをTDMAで実装しようとしたり，正確な100

Hz のサンプリングで地震のゆれを計測するといったアプリケーションを実現しようとする場合にはソフトリアルタイム処理では十分ではなくハードリアルタイム処理が必要となる．例えば実行周期とデッドラインが 26 μs，計算時間が 12.5 μs の無線通信物理層のタスクと，加速度を正確な 100 Hz のサンプリング周期で取るために実行周期が 10 ms，デッドラインと計算時間が 2.2 μs のタスクを同時にこなさなければならないという状況は容易に発生する．TinyOS は，このようなタスクを効率的に実行するような仕組みを持ち合わせていない．

以上のような背景により，筆者らは

- プログラムの書きやすさ
- ハードリアルタイム処理のサポート

の 2 つを満たすオペレーティングシステムを実現することを目指す．

2.2 Threads v.s. Events

2.1 に挙げた TinyOS の 2 つの使い辛い点は，TinyOS が event model で構築されていることに起因する．

システムを構築するのに event model を用いるか，thread model を用いるかは長い間議論されてきた [1], [17]〜[20]．本稿では thread model を「少ない数の大きなタスクを複数の実行ストリームによって実行するモデル」，event model を「たくさんある非常に小さいタスクを 1 つの実行ストリームで実行するモデル」と定義する．1979 年に発表された [17] では event model で構築されたシステムは thread model でも構築可能であり，性能や必要とされる計算資源には差が無いことが示された．しかしながら，1996 年に発表された [18] では，thread model よりも event model の方がよいという主張がされている．[18] では，thread model を用いるとタスクが pre-emption されるため，コンテキストスイッチや同期のオーバヘッド，デッドロックなどの問題が発生するのでイベントを用いた方がよい，と述べられている．しかしながら，[18] の主張はある 1 つの誤った前提を置いている．確かに既存の POSIX thread や Win32 thread は pre-emption を前提とした time-sliced multithreading であるが，2004 年に発表された [19] で述べられているように，thread model を用いた場合でもタスクを pre-emption せずに co-operative でスイッチすることで event model を用いた場合と同等のシステムとして扱うことができる．また，event model を用いた場合ではタスクが細かく分割されるので TinyOS のソースコードで示したように制御フローが把握し辛くなる．それに対して thread model を用いると制御フローが明確になるというメリットが存在する．例えば LED を 1 秒毎に点滅させるアプリケーションを thread model を用いると

```
void thread(void)
{
  while(1){
    toggle_led();
    sleep(1);
  }
}
```

のように記述することができる．先ほどの TinyOS のコードと見比べても書きやすさや制御フローの把握のしやすさは thread model を用いた方がよいことは明らかである．また，TinyOS [1] や Contiki [20] によると，thread model を用いた場合には各スレッド毎にスタックを用意しなければならないので必要とされる計算資源が多いと主張されている．しかしながら，事前にスレッドにメモリを割り当てるなどの各スレッドにスタックを用意しなくても良いような仕組みを実現できれば event model を用いた場合と同程度の省資源性を実現できる．以上の議論を踏まえて，筆者らは「プログラムの書きやすさ」を実現するために thread model を用いてオペレーティングシステムを構築する．

thread model を用いる場合に考えなければならないのは pre-emption が必要あるのか，無いのか，である．結論から述べるとハードリアルタイム処理を実現したい場合には pre-emption は必須である．しかしながら，pre-emption はタスクスイッチのオーバヘッドが非常に大きい．タスクスイッチの主なオーバヘッドとしては，コンテキストの切り替え（CPU の状態の保存と復元）と割り込み毎に発生するタスクの優先度の判定の 2 つである．また，pre-emption する場合には共有データに対して排他制御を行わなければならない．特に無線センサネットワークでは無線通信の各層の間でデータを共有するため，スレッド間での排他制御の仕組みは必須である．

3. PAVENET OS

2. での議論を踏まえて，筆者らは無線センサノード向けのオペレーティングシステムである PAVENET OS の設計と実装を行った．PAVENET OS では，スレッドを用いることでユーザに対してプログラムの書きやすさを提供する．また，リアルタイム性が必要となるスレッドを pre-emptive で実行する．このとき，pre-emptive multithreading の実現に CPU の機能を積極的に利用することでコンテキストスイッチ時のオーバヘッドを削減する．さらに，リアルタイム性の必要無いスレッドを co-operative multithreading で実行することで，イベントを用いた場合と同様な特徴を実現しつつもユーザに対してプログラムの書きやすさを提供する．

また，無線センサネットワークにおいて共有メモリにアクセスする場合に問題になってくるのは通信レイヤ間でのパケットの受け渡しの時である．これに向けて，各

層の間のインタフェース内で排他制御する仕組みをオペレーティングシステムが提供することでハードリアルタイムタスクを処理する場合でもユーザが排他制御を意識せずに無線プロトコルを開発できる仕組みを提供する.

3.1 ハードリアルタイムタスクスケジューラ

PAVENET OS ではハードリアルタイム処理を少ないオーバヘッドで実現するために Microchip 社の CPU である PIC18 [21] の動的な優先度割り込みを使用して deadline-monotonic scheduling [22] を実現する. deadline-monotonic scheduling は 1 つのプロセッサの場合に最適な優先度割り当てが可能であることが知られている [22].

PIC18 はタイマ割り込み, ポートチェンジ割り込み, 外部割込み, シリアル送信割り込み, シリアル受信割り込みなどさまざまな割り込みを持つ. PIC18 ではこれら全ての割り込みを高優先度割り込みか低優先度割り込みかの 2 種類から選択することができる. 低優先度割り込みベクタは 0018h 番地であり, 高優先度割り込みベクタは 0008h 番地である. また, 高優先度割り込みは低優先度割り込み実行中でも割り込むことが可能である. さらに, PIC18 は高優先度の割り込みを高速に行う機能を持っており, 割り込み時のコンテキストの保存と復元をハードウェアで実行する. この PIC18 のハードウェアによる動的な割り込み優先度判定の機能と, コンテキストの保存と復元のハードウェア処理の機能を利用することで, 少ないオーバヘッドでハードリアルタイム処理を実現する.

PIC18 の各割り込みはレジスタに割り込み優先度ビット, 割り込み有効ビット, 割り込みフラグビットの 3 つのビットを持っている. たとえば PIC18 の持つタイマ 0 の割り込みは優先度ビットが TMR0IP, 有効ビットが TMR0IE, フラグビットが TMR0IF である. 割り込み優先度ビットにはその割り込みが高優先度 (1) なのか低優先度 (0) なのかが設定される. 割り込み有効ビットにはその割り込みが現在有効なのか (1) 無効なのか (0) が設定される. 割り込みフラグビットは, 割り込み有効ビットが 1 に設定されている場合に割り込みフラグビットが 1 に設定されると割り込み優先度ビットに応じたレベルの割り込み処理が実行される.

筆者らは, このような PIC18 の動的な優先度割り込みの機能を最大限に利用することで少ないオーバヘッドでハードリアルタイム処理を実現する. まず, 各割り込みには

```
void (*task_timer0)(void); //タイマ 0
void (*task_timer1)(void); //タイマ 1
void (*task_int1)(void);   //外部割込み 1
void (*task_int2)(void);   //外部割込み 2
void (*task_rc)(void); //シリアル受信割り込み
```

のように関数ポインタを用意する. ハードリアルタイム用のタスクを追加するための API では

```
void add_rtask(uint8 isr_type,
               uint8 priority,
               void (*func)(void))
{
    switch(isr_type){
    case ISR_TMR0:
        if(priority == ISR_HIGH)
            TMR0IP = 1;
        else if(priority == ISR_LOW)
            TMR0IP = 0;

        TMR0IE = 1;
        task_tmr0 = func;
        break;
    case ISR_TMR1:
        :
```

のように, 各割り込みに対して CPU の割り込みレベルを設定し, 関数を割り当てる. 高優先度と低優先度の割り込みベクタは

```
0008h: call isr_high
000Ah: nop
       :
0018h: call isr_low
001Ah: nop
```

のように記述されており, isr_high は

```
void isr_high(void)
{
    if(TMR0IP && TMR0IE && TMR0IF){
        TMR0IF = 0;
        task_timer0();
    }

    if(TMR1IP && TMR1IE && TMR1IF){
        TMR1IF = 0;
        task_timer1();
    }

    if(INT1IP && INT1IE && INT1IF){
        INT1IF = 0;
        task_int1();
    }
            :
```

のように, isr_low は

```
void isr_low(void)
{
    if((TMR0IP == 0) && TMR0IE && TMR0IF){
        TMR0IF = 0;
        task_timer0();
    }

    if((TMR1IP == 0) && TMR1IE && TMR1IF){
        TMR1IF = 0;
        task_timer1();
    }

    if((INT1IP == 0) && INT1IE && INT1IF){
        INT1IF = 0;
        task_int1();
    }
            :
```

のように記述されている. つまり, PAVENET OS では同じ優先度のタスクを複数使用することを許している.

3.2 ベストエフォートタスクスケジューラ

PAVENET OS はハードリアルタイム性の必要な処理は 3.1 に示した pre-emptive multithreading で実現される．それ以外の処理は，ベストエフォートタスクスケジューラによって実行される．無線センサノードにおけるルーティングプロトコルの処理や，フラッシュメモリに対する遅延書き込みの処理，センサ情報の問い合わせに対して返事を返す処理などはベストエフォートで十分に処理可能である．

ベストエフォートタスクスケジューラはできるだけ単純で軽量な仕組みで実現するために，multithreading の実現にいくつかの制限を設ける．まず，1 つめの制限はタスクのスイッチを co-operative で行うことである．co-operative でタスクスイッチをすることで共有資源に対するアクセスや，CPU のコンテキストの保存と復元が不要になり，プログラムカウンタを管理するだけでスレッドを実現できる．2 つ目の制限は，タスクをスイッチさせるシステムコールを呼ぶことのできる場所の制限である．PAVENET OS ではスレッドから呼ばれた関数の中からはタスクをスイッチさせるシステムコールを呼べないようにしている．コールスタックを保存することでスレッドから呼ばれた関数の中でもタスクをスイッチさせることは可能であるが，新たにコールスタックを保存するための計算資源が増加するのでこのような制限を設けた．3 つ目の制限は 1 つのコードからは 1 つのスレッドしか起動できないことである．この制限は PAVENET OS がコンパイル時にスレッドが使用するメモリを確保することに起因する．あらかじめコンパイラが使用するメモリを決めておくことでスレッドに対して動的にスタックを割り当てる必要が無くなり，少ない計算資源でスレッドを実現できる．

ベストエフォートタスクスケジューラの TCB（タスク制御ブロック）を表 1 に示す．`tid` はスレッドの ID である．スレッド作成時にシステムがスレッドに対して割り当てる．`state` はスレッドの状態である．スレッドはデッド状態，実行状態，スリープ状態，待ち状態の 4 つの状態を持つ．タスク操作関数には表 2 の 7 つが存在する．`pc` はプログラムカウンタであり，現在実行中のスレッドのプログラムカウンタが記録されている．`sleep_time` はタスクがスリープ状態から実行状態に遷移する時間である．

表 1 タスク制御ブロック
Table 1 Task control block

名前	サイズ	内容
`tid`	8 bit	スレッド ID
`state`	8 bit	スレッドの状態
`pc`	16 bit	プログラムカウンタ
`sleep_time`	8 bit	待ち状態の残り時間

表 2 タスク制御関数
Table 2 Task control functions

関数名	処理	実行後の状態
`add_task(`*funcname*`)`	タスクを TCB に追加	実行状態
`os_yield()`	OS に制御を渡す	実行状態
`sleep(`*time*`)`	*time* 秒スリープ	スリープ状態
`sig_wait()`	シグナルを待機	待ち状態
`suspend_task(`*pid*`)`	*pid* を待ち状態に	実行状態
`signal_task(`*pid*`)`	*pid* を実行状態に	実行状態
`kill_task(`*pid*`)`	*pid* をデッド状態に	実行状態

PAVENET OS では 100 ms に 1 つ増加する `jiffies` という変数を具備しており，12.7 秒までのスリープをサポートしている．12.7 秒以上のスリープはユーザがプログラム内で時刻を記録し，自分で処理する必要がある．

表 2 に PAVENET OS が提供するタスク制御関数を示す．`os_yield`，`sleep`，`sig_wait` はタスクをスイッチさせるための関数である．`os_yield` のソースコードを以下に示す．

```
void os_yield(void)
{
    pcounters[current_task] = TOS;
    asm("pop");
}
```

TOS はプログラムカウンタを意味する．PAVENET OS では前述したようにさまざまな制限を加えているので，このような非常に単純な関数でタスクをスイッチすることができる．

タスクスケジューラのソースコードを以下に示す．

```
uint8 task_schedule(void)
{
    uint8 i;

    for(i = 0; i < task_num; i++){
        if(tcb[i].state == TASK_SLEEP){
            if(tcb[i].sleep_time == jiffies)
                tcb[i].state = TASK_RUN;
        }

        if(tcb[i].state == TASK_RUN)
            exec_task(i);
    }
}
```

このように PAVENET OS のベストエフォートタスクスケジューラは非常に単純な構成になっている．

3.3 無線プロトコルスタック

PAVENET OS では，ハードリアルタイム処理を実現するために pre-emptive multithreading を用いているため，スレッド間での共有データの排他制御が重要になってくる．筆者らは，無線センサノードにおける共有データのアクセスが通信レイヤ間のデータの受け渡し時に頻繁に発生することに着目し，オペレーティングシステムとして用意した通信レイヤ間の API 内に排他制御を隠蔽

することで，ユーザが他の層のタスクを意識しなくてもMACプロトコルやルーティングプロトコルを開発可能な環境を提供する．さらに各レイヤの独立性を実現することでアプリケーションに応じてさまざまなプロトコルを組み合わせて使用することができる．

図1にPAVENET OSの無線プロトコルスタックを示す．PAVENET OSでは通信層として物理層，MAC層，ネットワーク層，ソケット層を提供する．また，レイヤ間のパケットの受け渡しを効率化するためにBSD mbufに似たpbufという仕組みを提供する．

pbufはある識別子に対してバッファを割り当てており，レイヤ間で識別子の受け渡しだけで済むように実装されている．pbuf用のAPIを以下に示す．

```
uint8 get_new_pbuf(void);
byte *get_pbuf_next(uint8 index, uint8 size);
byte *get_pbuf_head(uint8 index);
uint8 release_pbuf(uint8 index);
uint8 get_pbuf_size(uint8 index);
```

これらのAPIの使われ方をパケット送信時と受信時における各層での処理を例に説明する．

送信時の処理を図2に示す．まず，ユーザがsendを呼ぶとソケット層に対してデータが渡される．ソケット層では，get_new_pbufを用いて新しいpbufを作成する．作成された直後のpbufのサイズは0byteである．ソケット層はget_net_optを用いてネットワーク層とMAC層からヘッダ長を取得し，get_pbuf_nextを用いて取得したヘッダ長とペイロードの長さを足した分だけのpbufの領域を拡大する．そして，取得したヘッダに対してset_sock_optを用いて宛先アドレスなどの書き込みを行い，down2netを用いてネットワーク層の送信キューに対してpbufの識別子を書き込む．ネットワーク層の処理はco-operative multithreadingで実現されているため，ソケット層からネットワーク層に渡すときの排他制御は必要無い．ネットワーク層ではset_mac_optを

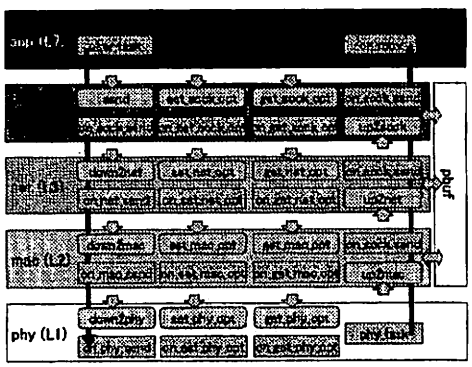

図1　無線プロトコルスタック
Fig. 1　Wireless protocol stack

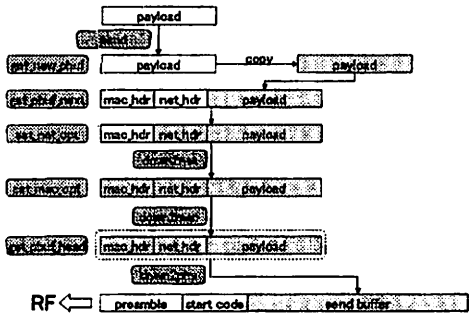

図2　送信時の処理
Fig. 2　TX transaction

用いてpbufに宛先アドレスなどの設定を行い，MAC層の送信キューに対してpbufの識別子を書き込む．MAC層の処理はハードリアルタイムが必要となる場合がありうるので，識別子の書き込み時にはdown2mac内で割り込み禁止などの排他制御を行う．MAC層では，pbuf識別子からパケットデータを抽出し，パケットデータを物理層の送信バッファに対して書き込む．物理層では送信バッファを1つだけ持っており，その送信バッファが使用中の場合には送信バッファに対する書き込みは禁止される．この場合にはMAC層に対してはビジーが返るので，MAC層はビジーが終了するのを待つことで排他制御を行う．物理層はMAC層から送信バッファにデータが書き込まれるとそのデータにプリアンブルとスタートコードを付与し，1bitずつハードリアルタイム処理で送信する．

このように，pbufとレイヤ間のAPI内の排他制御により，各層のモジュール性と開発のしやすさが提供される．

4.　初期的評価

PAVENET OSの初期的な評価としてTinyOSとのハードリアルタイム処理の比較を行った．

TinyOSではタスクを100Hzでセンサからの取得するというタスクのみを実装した場合にでも100Hzを作ることができなかった上に，毎回のセンサの値を取得する間隔にばらつきがあった．さらに，無線通信を行いながら100Hzのセンサの値の取得を行った場合では毎回のセンサの値を取得する間隔のばらつきが大きくなった．

PAVENET OSでは，正確な100Hzでのセンサの値の取得ができた．また，無線通信を行った場合はごくまれにセンサのデータを取得する間隔が10.001msになることがあった．これは無線通信の処理の中でデータの受信を完了して上位層にデータを渡す場合にPAVENET OSの無線プロトコルスタックの中でクリティカルセクションを経由することがあることに起因する．しかしながら，

表 3 ハードリアルタイム処理
Table 3 Hard realtime transaction

	TinyOS				PAVENET OS			
	最大 (ms)	最小 (ms)	平均 (ms)	標準偏差	最大 (ms)	最小 (ms)	平均 (ms)	標準偏差
100Hz	9.713	9.717	9.715	0.0125	10.000	10.000	10.000	0.0000
100Hz + RF	9.321	9.736	9.699	0.6748	10.001	10.000	10.000	0.0000

このクリティカルセクションを通る時間は非常に小さい上にごくまれにしか起こらないためデッドライン内に処理を完了できている.

5. 関連研究

5.1 オペレーティングシステム

無線センサネットワークのオペレーティングシステムに関する研究としては, TinyOS [1], [11], SOS [23], Contiki [20], MANTIS [24], protothreads [25] などが挙げられる. これらの技術の多くは event model で構築されているため thread model に比べてプログラムの記述が難しく, かつハードリアルタイム処理をサポートしていない.

TinyOS [1] は現在無線センサネットワークの研究分野で標準的に使用されている OS である. TinyOS は nesC [11] と呼ばれる C 言語に似た event model に特化した言語を用いて構築されており, 少ない計算資源とオーバヘッドで複数のタスクを処理することができることを特徴としている. しかしながら, TinyOS は 2.1 で述べたようにプログラムの書き辛さとハードリアルタイム性の欠如の問題を持っている. さらに, ユーザは nesC の記述の仕方を学ばなければならないという問題点も持っている.

SOS [23] は TinyOS と同様に event model を用いて構築されている. TinyOS が動的なプログラムモジュールの変更ができないのに対し, SOS は少ないオーバヘッドでプログラムモジュールの動的な追加や削除を行うことができる. また, タスクを C 言語で記述可能という特徴も持っている. しかしながら, SOS は TinyOS と同様に event model であるがゆえにハードリアルタイム処理をサポートしておらず, プログラムも書き辛い.

Contiki [20] は C 言語を用いて event model と thread model の両方を用いて構築されている. Contiki は各スレッドに対してスタックを割り当てることで time-sliced pre-emptive multithreading を実現している. そのため, PAVENET OS に比べてスレッドを実現する時のメモリ使用量が多く, ハードリアルタイム処理も実現できない.

protothreads [25] は event model におけるプログラムの書き辛さを軽減するために考案された仕組みである. event model では各タスクは run-to-completion で実現されるため, ユーザは何かを処理したい場合にその処理を複数のタスクに分割して記述しなければならない. protothreads では, event handler を途中で中断可能な仕組みを取り入れることで event handler の書きやすさを提供している. protothreads を用いることで event model の制御フローの把握し辛さは若干軽減されるものの, 本質的な event model のプログラムの書き辛さは解決していない. さらに, ハードリアルタイム処理もサポートしていない.

MANTIS は PAVENET OS と同じく thread model のオペレーティングシステムである. MANTIS は time-sliced pre-emptive multithreading で実現されており, 各スレッドに対してスタックを割り当てなければならない. そのため, 必要とされる計算資源が PAVENET OS に比べて多い. また, ハードリアルタイム処理もサポートしていない.

5.2 階層化

無線センサノードにおける通信プロトコルの階層化に関する研究としては [26] が存在する. [26] では, TinyOS 上で多様なプロトコルをサポートするために L2 と L3 の間に SP(Sensornet Protocol) と呼ばれるプロトコルを提供している. 本研究の目的は API 内で排他制御を隠蔽することで各層の独立性を実現することであるので [26] と目的が異なる. そのため, SP と PAVENET OS の無線プロトコルスタックは共存可能であるので, 今後 SP を PAVENET OS 上に実装することも検討する必要がある.

6. おわりに

本稿では, 無線センサノード用のハードリアルタイムオペレーティングシステムである PAVENET OS の設計と初期的な評価について述べた. PAVENET OS は TinyOS と同程度の省資源と低オーバヘッドで実現できるうえに, TinyOS が持っていないハードリアルタイム処理とプログラムの書きやすさを提供する. さらに, ハードリアルタイム処理を記述した場合でもオペレーティングシステムとしてレイヤ間の排他制御を隠蔽する機能を提供することでユーザは各層を独立に開発することができる. 現在, PAVENET OS と TinyOS の詳細な比較を行っている.

文　献

[1] Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D. and Pister, K.: System Architecture Directions for Networked Sensors, *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'00)*, Boston, Massachusetts, ACM, pp. 93–104 (2000).

[2] Gutierrez, J. A., Naeve, M., Callaway, E., Bourgeois, M., Mitter, V. and Heile, B.: IEEE 802.15.4: A Developing Standard for Low-Power, Low-Cost Wireless Personal Area Networks, *IEEE Network*, Vol. 15, No. 5, pp. 12–19 (2001).

[3] Ye, W., Heidemann, J. and Estrin, D.: An Energy-Efficient MAC Protocol for Wireless Sensor Networks, *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'02)*, New York, New York, pp. 1567–1576 (2002).

[4] van Dam, T. and Langendoen, K.: An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks, *Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems (SenSys'03)*, Los Angeles, California (2003).

[5] Rajendran, V., Obraczka, K. and Garcia-Luna-Aceves, J. J.: Energy-Efficient, Collision-Free Medium Access Control for Wireless Sensor Networks, *Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems (SenSys'03)*, Los Angeles, California (2003).

[6] Polastre, J., Hill, J. and Culler, D.: Versatile Low Power Media Access for Wireless Sensor Networks, *Proceedings of the 2nd ACM Conference on Embedded Network Sensor Systems (SenSys'04)*, Baltimore, Maryland (2004).

[7] Rhee, I., Warrier, A., Aia, M. and Min, J.: ZMAC: A Hybrid MAC for Wireless Sensor Networks, *Proceedings of the 3rd ACM Conference on Embedded Network Sensor Systems (SenSys'05)*, San Diego, California (2005).

[8] 猿渡俊介, 森川博之, 青山友紀：家庭内センサネットワークにおける低消費電力 MAC プロトコル, 電子情報通信学会総合大会 (2005).

[9] Jamieson, K., Balakrishnan, H. and Tay, Y.: Sift: A MAC Protocol for Event-Driven Wireless Sensor Networks, *Proceedings of the 3rd European Workshop on Wireless Sensor Networks (EWSN'06)*, Zurich, Switzerland, pp. 260–275 (2006).

[10] Hill, J. and Culler, D.: MICA: A Wireless Platform For Deeply Embedded Networks, *IEEE Micro*, Vol. 22, No. 6, pp. 12–24 (2002).

[11] Gay, D., Levis, P. and von Behren, R.: The nesC Language: A Holistic Approach to Networked Embedded Systems, *Proceedings of Conference on Programming Language Design and Implementation (PLDI'03)*, San Diego, California, ACM, pp. 1–11 (2003).

[12] 森戸 貴, 南 正輝, 鹿島拓也, 猿渡俊介, 森川博之, 青山友紀：バッテリレス無線センサネットワークの設計と実装, 電子情報通信学会技術研究報告 (2004).

[13] 堀江信吾, 猿渡俊介, 倉田成人, 森川博之, 青山友紀：無線センサネットワークを用いた地震モニタリングにおける同期性能の評価, 第 2 回センサネットワーク研究会予稿集 (2005).

[14] Saruwatari, S., Kashima, T., Minami, M., Morikawa, H. and Aoyama, T.: Pavenet: A Hardware and Software Framework for Wireless Sensor Networks, *Transaction of the Society of Instrument and Control Engineers, Volume E-S-1*, No. 1, pp. 74–84 (2006).

[15] 鈴木 誠, 鹿島拓也, 猿渡俊介, 森川博之, 青山友紀：1 チップマイクロコンピュータにおける動的機能モジュール機構の実装と評価, 情報処理学会マルチメディア, 分散, 協調とモバイルシンポジウム (DICOMO 2005) (2005).

[16] 猿渡俊介, 南 正輝, 森川博之：ユーザによる制御が可能なセンサ/アクチュエータネットワークの設計, 電子情報通信学会技術研究報告, 第 3 回センサネットワーク研究 (2006).

[17] Lauer, H. C. and Needham, R. M.: On the Duality of Operating System Structure, *ACM SIGOPS Operating System Review*, Vol. 13, No. 2, pp. 3–19 (1979).

[18] Ousterhout, J.: Why Threads Are A Bad Idea (for most purposes), *USENIX 1996 Annual Technical Conference (Invited Talk)*, San Diego, California (1996).

[19] von Behren, R., Condit, J. and Brewer, E.: Why Events Are A Bad Idea (for High-Concurrency Server), *Proceedings of the 9th Workshop on Hot TOpics in Operating Systems (HotOS IX)*, Lihue, Hawaii (2003).

[20] Dunkels, A., Gronvall, B. and Voigt, T.: Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors, *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN'04)*, Tampa, Florida, pp. 455–462 (2004).

[21] Microchip: *PIC18F2525/2620/4525/4620 Data Sheet*. http://www.microchip.com/.

[22] Leung, J. Y. and Whitehead, J.: On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks, *Performance Evaluation*, Vol. 2, No. 4, pp. 237–250 (1982).

[23] Han, C. C., Kumar, R., Shea, R., Kohler, E. and Srivastava, M. B.: A Dynamic Operating System for Sensor Nodes, *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services (MobiSys'05)*, Seattle, Washington, pp. 163–176 (2005).

[24] Bhatti, S., Carlson, J., Dai, H., Deng, J., Rose, J., Sheth, A., Shucker, B., Gruenwald, C., Torgerson, A. and Han, R.: MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms, *ACM Mobile Networks & Applications (MONET), Special Issue on Wireless Sensor Networks*, Vol. 10, No. 4, pp. 563–579 (2005).

[25] Dunkels, A., Schmidt, O., Voigt, T. and Ali, M.: Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems, *Proceedings of the 4th ACM Conference on Embedded Networked Sensor Systems (SenSys'06)*, Boulder, Colorado (2006).

[26] Polastre, J., Hui, J., Levis, P., Zhao, J., CUller, D., Shenker, S. and Stoica, I.: A Unifying Link Abstraction for Wireless Sensor Networks, *Proceedings of the 3rd ACM Conference on Embedded Network Sensor Systems (SenSys'05)*, San Diego, California (2005).