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

2008－ＳＥ－160 (7)
2008－EMB－ 9 (7)
2008／6／19

組込み Linux 向けスレッドフレームクラスの開発

瀬下博之†
倉又一郎††

近年オープンソースである Linux オペレーティングシステムを組込み系機器に搭載し利用する例が増えてきている。制御用アプリケーション開発を行なう場合、プロセス間通信を用いたシステム構築例は多いが Linux スレッドクラスライブラリはあまり使用されていない。しかし、コンテキスト切替時のオーバヘッド低減やメモリ効率の向上などの点からスレッドを用いたプログラム開発は有用である。そこで Linux を制御分野に用いるにあたり、簡易にマルチスレッドプログラミングができる仕組の構築を行なった。また開発したスレッドライブラリの有用性を調べるためにサンプル例題を用い仕様分析・実装を試み評価を行なった。

Development of thread Frame Class for Embedded Linux

Hiroyuki Seshimo and Ichiro Kuramata

Linux OS ,open source is well used by developing embedded system recently. In case of developing embedded application system, InterProcess Communication is well used. Linux thread library is not so used. But at decreasing overhead of context switchng and improving the memory efficiency, the development of using thread library is useful. So, we tryed to develop the program structure which can constructed with the programming of using thread library easily. And we evaluated the new thread library by using sample model and analyzed the specication of the library and made the program codes for examing the effectiveness of the thread library.

1. はじめに

高機能 CPU、大容量メモリの登場により組込機器の分野にも OS として Linux[1] が利用されるようになってきている。 組込み機器のソフトウェアは、近年仕様が複雑、大規模になってきており、同時に開発する人員も増え、ソフトウェアバグが引き起こす損害が話題にもなってきている。 こうした状況に対応する手段として、組込み系ソフトウェアにおいてもオブジェクト指向を用いた開発が取り入れられるようになってきている。

組込み Linux を利用する場合、ソフトウェア開発では制御系処理プログラムの場合プロセス間通信、共有メモリなどを用いたソフトウェア設計が一般的に行なわれてきていた。

言語として C++を用いることにより、オブジェクト指向を用いた設計をすることが可能となる。またメモリを効率的に利用したり、より短時間のコンテクストスイッチングなどを行いたい場合 Linux 環境ではスレッドを用いた開発を行なうことが可能である。しかしながら、Linux スレッドライブラリ使用方法は煩雑なため、スレッド処理を行なうソフトウェアの開発は容易ではなく簡易にスレッドライブラリを利用できる状況ではなかった。スレッドの仕組みをクラスライブラリ化したソフトウェアを利用し、容易にオブジェクト指向を用いたスレッドアプリケーションが開発できるものがあるか調査したが出回っているものは少なかった。

そこで、スレッドライブラリを利用してスレッドクラスを構築し、これをもとに比較的容易にソフトウェア開発する方法を開発した。また、オブジェクト指向を用いてシステム分析する場合、クラス分析レベルから実装レベルに移行する場合、クラス構成などが分析レベルと実装レベルのものとで乖離が大きくなる方向にあり、乖離が少なくなる方式を必要とした。

今回、分析レベルから、スレッドフレームワークを用いることを前提とした分析方法も検討した。その方式を実例を用いて示す。

2. スレッドライブラリの解説

2.1 概要

Linux 上でスレッドを利用する場合、pThread ライブラリを[1][2]用いて実現する。pThread ライブラリをそのまま利用した場合、プログラム記述が複雑で容易に利用することが困難であり、これがプログラムバグを引き起こす要因になると予想された。また、

† シスメックス株式会社、Sysmex Corporation
†† カンダシステム株式会社、kandasystem Inc
[1] Linux は LinusTrvalds 氏の商標または登録商標である

スレッドを用いたアプリケーションでは、スレッド間の通信によりスレッドの相互動作が行われる。スレッド間でメモリを容易に共通に利用することができるが、これによりプログラムバグを誘発する可能性が大きくなると考えられた。そこで下記要求が実現できるスレッドライブラリの仕組みを検討した。

2. 2設計目標

スレッドライブラリを開発するにあたって開発目標を下記の様に設定した。

①少ないプログラム記述量で簡易にスレッドを使用したプログラムが記述できること

②スレッド特有の共有するメモリ情報を不用意にアクセスすることによる不具合が発生しにくい仕組みを実現できること

上記①②を満足させる方法を下記に示す。

①スレッドをクラス化し、自スレッドの生成・実行が簡易にできるようにする

②スレッド間のメッセージ送受信が容易になるようなメールクラス、メールボックスクラスを備えること

スレッドのクラス化を検討するにあたり以下の項目に関して検討を行なった。

①タイミング要求の厳しい仕様への対応
②メッセージ通信仕様
③排他制御への対応
④時間待ちへの対応

1) タイミング要求の厳しい仕様への対応検討

組込み Linux 上でスレッドを用いたアプリケーションを開発する場合、ハードウェアに関連する動作を実装するのは主にデバイスドライバにより行われる。アプリケーションで ioctrl()関数をコールすることによりデバイスドライバのリード/ライトが実行される実動作が行われる。割り込みなどの処理もデバイスドライバ内で対応する。

その結果として、リアルタイム処理で時間的に厳しいハードウェア処理はデバイスドライバで処理を担当することことができるため、比較的リアルタイム性のゆるい処理に主眼をおいてスレッドクラスを対応させることが可能となる。

2)メッセージ通信仕様検討

スレッド間の通信として下記機能を各アプリケーションスレッドで利用できる様にした。

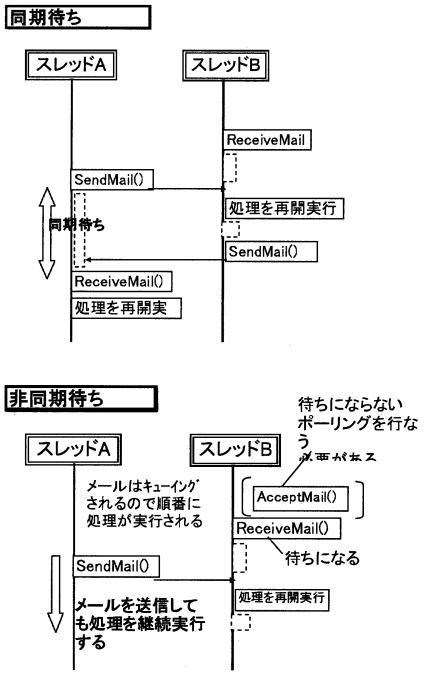
①スレッド間で同期待ちあわせができること。(メッセージを受信するスレッドはメッセージ受信メソッドを実行すると wait 状態になる)

②非同期でメッセージを送ることができること。(メッセージ送信したスレッドが wait 状態になることなく処理を継続できる。メッセージを受信するスレッドはメールが到着するまで wait 状態にある)

③メールが到着したかを wait 状態にならずに確認することができること。

スレッド間の情報送受では上記を実現するため3種類のメッセージ通信関数を用いて仕様の実現を行なった。(Sendmail() 、ReceiveMail() 、AcceptMail())

メッセージ通信の具体的シーケンスを図―1に示す。



図－1 メッセージ間通信

3)排他制御への対応

ハードウェア資源の排他制御は、デバイスドライバをアクセスするスレッドクラスを1つ用意し、資源を利用するクラスは、メッセージを該当スレッドにメッセージを送信する。メッセージは、デバイスドライバ処理用のスレッドクラス内のメールボックス Que に FIFO の形でキューイングされ、メールが到着する都度順次処理が実行される。以上の仕組みを搭載することにより資源の排他制御を実現する。

4)時間待ちへの対応

タイマーを管理するスレッドを専用に用意し時間管理を行い、タイマースレッドから一定時間経過した時、該当スレッドにタイマ経過メッセージを送ることにより各ユーザスレッドは時間待ちを実現する。usleep()もしくは sleep()関数を用いて時間待ちを行なう。これによりユーザスレッド自身で usleep()を実行することによる処理遅延を防ぎ、より正確な時間待ちを実現することができる。

以上の検討を踏まえ今回開発するスレッドライブラリの利用に関しての注意事項を以下の様に定めた。

- ●スレッドは処理を早急に終え、メッセージ待ちを発行し CPU 占有を放棄する
- ●排他制御の必要なものは、資源をアクセスするスレッドを1つに限定し、そのスレッドにメッセージを送り処理を行なってもらう。

上記検討したスレッドクラス仕様を用いてアプリケーションの実現が可能かをモデルサンプルを用いて検証を試みた。詳細は3章で示す。

2. 3スレッドクラス概要・動作

下記にスレッドクラス内の主要なクラス概要を下記に示す。

1)KSThread クラス

①KSThread を継承することによりスレッドを用いたアプリケーションクラスを構築することができる

②クラス内に自分宛用のメール BOX を持ちメールを受け取ることができる

③メールを送りたいスレッドクラスのメール BOX のポインタを取得することにより他のスレッドクラスにメールを送ることができる。

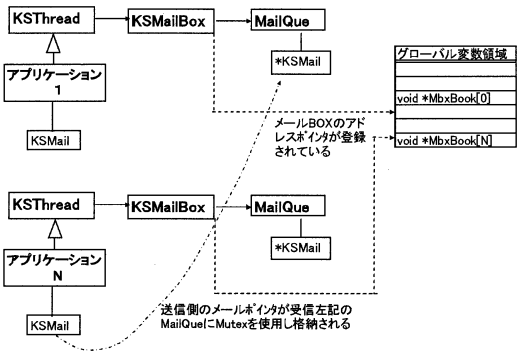
2)KSMail クラス

①string データをメール情報として格納するクラス。メールのタイプなどを設定できる属性設定用の情報を持つ。

3)KSMailBox クラス

①Mail オブジェクトを保存する MailQue を持つ。MailBox 内の MailQue はミューテックスを用い排他制御・管理される。

②Mail オブジェクを Que に格納、取り出し操作することができる。下記に各クラスのオブジェクトの関連図を示す。



図－2 スレッドクラスオブジェクト図

アプリケーションの基本的記述例を下記に示す。

```
//同期型クラス例
class TestA : public KSThread {
 public:
   TestA(){}
   ~TestA(){}
 protected:
   int Run() {
     KSMail *pMail;
     while(1){
         //
         SendmailDataToClass(
                 TESTB_CALSS,"SampleData");
       pMail = ReceiveMail();
         //メッセージを送ったあとTestBからのレスポンスを待つ
       delete pMail;
         sleep(1);
     }
   }
  void SendmailDataToClass(int classno,char *data){
      KSMail *pReqMail = new KSMail;
      strcpy(pReqMail->m_cBuf,data);
      SendMail( classno,pReqMail);
   }
};

class TestB : public KSThread {
 public:
   TestB(){}
   ~TestB(){}
 protected:
   int Run() {
     KSMail *pMail;
     while(1){
       pMail = ReceiveMail();
       delete pMail;
     }
   }
};
```

```
//非同期型クラス例
class TestA : public KSThread {
 public:
   TestA(){}
   ~TestA(){}
 protected:
   int Run() {
     KSMail *pMail;
     while(1){
         //
         SendmailDataToClass(
                 TESTB_CALSS,"SampleData");
       //メッセージを送ったあと自由に処理を実行できる
       sleep(1);
     }
   }
  void SendmailDataToClass(int classno,char *data){
      KSMail *pReqMail = new KSMail;
      strcpy(pReqMail->m_cBuf,data);
      SendMail( classno,pReqMail);
   }
};

class TestB : public KSThread {
 public:
   TestB(){}
   ~TestB(){}
 protected:
   int Run() {
     KSMail *pMail;
     while(1){
       pMail = ReceiveMail();
       delete pMail;
     }
   }
};
```

図－3 基本的なスレッドアプリケーションクラス例

KSThread を継承したアプリケーションスレッドクラスは Virtual になっている Run()メッソド内にメールを受信するルーチンを ReceiveMail()関数を用いて記述する。また、メールを送信したい場合は、同様に SendMail()関数を用いてメッセージを送信する。

2. 4メール送受方法

1)メッセージ送受信機構

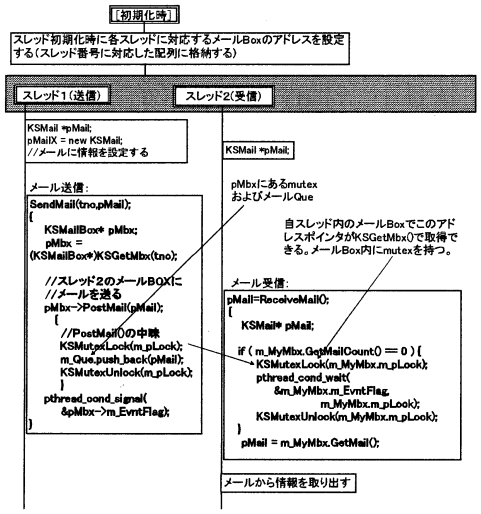
以下にメッセージ送受信のアプリケーション例を示す。

```
//-------------------------------//
// Test A(Receive & Send class)  //
//-------------------------------//
class TestA : public KSThread {
public:
  TestA(){ }
  ~TestA(){ }
protected:
  int Run() {
    KSMail *pMail;
    while(1){
      //メール受信待ちを行なう
      pMail = ReceiveMail();
      //TestBへの処理依頼がくるとメール
      //を作成する
      KSMail *pReqMail = new KSMail;
      strcpy(pReqMail->m_cBuf,"Test");
      //メールをTestBに送る
      SendMail( T_TESTB ,pReqMail);
      delete pMail;
      //メールをフリーする
    }
  }
};
```

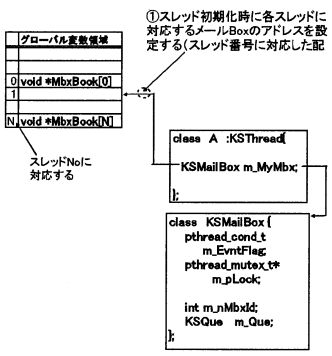
```
class TestB : public KSThread {

public:
  TestB(){
  }
  ~TestB(){
  }
protected:
  int Run() {
    KSMail *pMail;
    while(1){
      //メール待ちを行なう
      pMail = ReceiveMail();
      //処理依頼メールがくると
      //処理を実行する
      delete pMail;
      //メールをフリーする
    }
  }

};
```

図－4 メッセージ送受信基本アプリケーション例

このアプリケーション例で行なっているメッセージ送受信の仕組みを以下に説明する。



図－5 メッセージ送受信シーケンス



図－6 KSMailBox の仕組み

①スレッドクラスは自クラスに依存するメッセージ Box クラスを内包する。メッセージ Box クラスのインスタンスポインタをグローバルエリア内のメッセージ Box 管理配列に格納しておく。このメッセージ Box クラスは自クラス内に内包するスレッド Que への格納を制御する Mutex を管理する。

②メールを待つスレッドクラスは、自メール Box の Mutex を用い pthread_cond_wait()を実行し wait 状態に遷移する。

③メッセージ送信スレッド側で、メッセージ格納領域として、ヒープ領域のメモリ(メール情報)を確保する。ここに送信すべき情報を書き込む。さらにメールクラスを生成し、メールクラス内にメール情報ポインタを格納する。

④送信先のメール Box のアドレスをメッセージ Box 管理配列から取得する

⑤メール Box 内の Mutex を使用し、ロックをかけ送信先のメール Que にメール内容のアドレスポインタを格納する。pthread_cond_signal()を用いメール待ちスレッドにシグナルを送る。

図－4，5にメール送受信シーケンス、MailBox の仕組みの図を示す。

2. 5スレッドクラスの記述性の評価

1)スレッドクラスの優位性の検討

メッセージ型のスレッドクラスライブラリを使用することにより、図―4に示すように簡易にスレッドククラスを作成することが可能になった。また継承を用いることで必要な部分のみ記述するだけでよくなるため、コーディング作業量も減ることとなった。

送受信の基本的な関数は基本クラスに記述されているため、Run()仮想関数を実装し、その中でメッセージを送受信する仕様を記載するのみで済む。

また、クラス化されているため、不用意な別スレッドからのメモリアクセスを防ぐことが可能になっている。クラスにしていない場合は容易に別スレッドが使用するメモリ領域をアクセスできるため、不用意に別スレッドがアクセスし誤動作するような危険性を持つ。

2)メッセージを送受信することの優位性の検討

意味のある内容をメッセージ電文に入れることによりコマンド、パラメータなどをスレッド間で送受信させることが複雑な仕組みもなく可能となる。

例えば、電源 ON を 2 秒後に実行要求する様な場合、"Power:ON,2"という様なメッセージを定義し送信することにより仕様を実現することができる。メッセージでない場合には、構造体などをメッセージごとに定義し、構造体単位で Mutex などを用いて情報を送受することになる。

しかし、メッセージが増えると構造体の管理が複雑になりアプリケーション開発が複雑になる。一方メッセージを ASCII 電文で送受した場合、電文仕様を上記のように定義するだけで、アプリケーション内は簡易な字句解析ルーチンを配置するのみで、メッセージ仕様が増えても簡易なプログラム記述で対応可能になる。

ただし、データ量が多い場合はメッセージ解析のオーバヘッドが大きくなるので注意が必要である。

3)シングルトンクラスへの複数スレッドアクセスの検討

多量な情報を保持するクラスをシングルトンにして複数のスレッドが利用する仕様がある場合、通常はシングルトンのクラスに対応するサーバスレッドクラスを作り、ユーザスレッドはそのサーバスレッドに利用依頼を行なう形が本システムの利用方法になる。

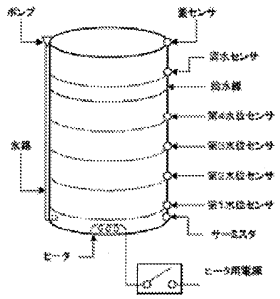
しかし、早急なレスポンスを実現したい場合は、サーバスレッドを介さないで直接データクラスをアクセスする必要がある。現時点ではこのような仕様を実現する機能を本スレッドクラスは実装していない。ロック機能を搭載した基本クラスを別途開発し、このクラスを継承したユーザクラスを開発することにより実現可能と考える。

次にモデルサンプルを用い、実際に仕様を展開し、実装・動作確認を行ない作成したクラスで仕様の記述が可能かどうかの評価を行なった。以降にその結果を示す。

3. Linux スレッドシステム事例による評価

3. 1概要

例題には、SESSAME のセミナーで用いられている「話題沸騰ポット」の要求仕様書(第 3 版)を使用し、システムの分析・実装・評価を行なった。[3][4]

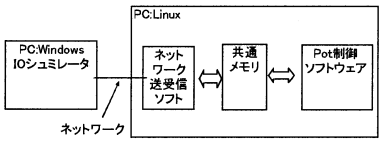


図－7 話題沸騰ポットの構成図

(概略仕様)

「高温保温、節約保温、ミルク保温の3つのモードを持ち、保温設定ボタンで設定する。沸騰ボタンを押すとポット内の水を沸騰させ、カルキ抜きを行なった後、保温になる。」

開発したソフトウェアの動作環境として、PC(Linux)を利用した。これにより、ハードウェア環境が準備できない環境下でもソフトウェアを開発し評価できるようなった。ネットワーク経由でターゲットとなる PC と通信を行い、Linux 内の共通メモリをアクセスするようにすることにより、ハードウェアをシュミレートできるソフトウェアを開発し利用した。擬似 IO ポートを共有メモリにマッピングし、ポットシステムはこの共通メモリをアクセスすることにより擬似 IO をアクセスすることが可能となっている。



図－8 ソフトウェア開発・試験環境

Linux ソフトウェアシステムでは、ハードウェアをデバイスドライバが吸収してくれるため、時間制約が厳しい仕様を実装する要求が少ないため、本ハードウェアシュミレーション環境を用いても開発が充分可能であると考えられる。

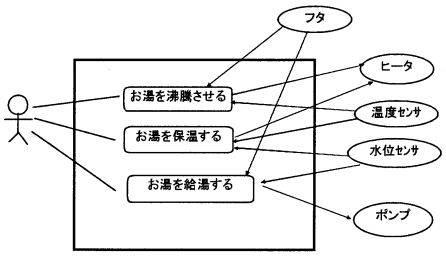
3．2分析実装手順

UML 図表を用いて仕様の展開を行なった。その結果を以下に示す。

①ユースケース分析
②ドメイン抽出
③ドメイン間メッセージ仕様作成
④ドメイン毎のドメイン内クラス抽出
⑤各実動作仕様の抽出・クラス詳細仕様実装

1)ユースケースを用いた仕様の分析

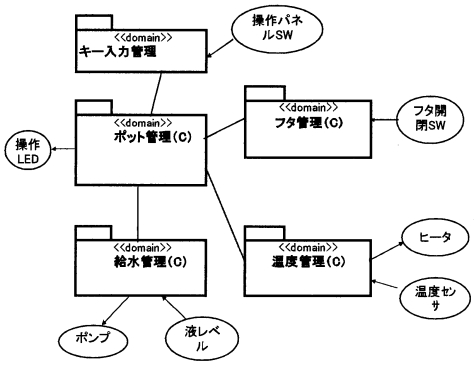
概略仕様からユースケース図を作成した。下記に結果を示す。[5][6]



図－9　ユースケース図

2) ドメイン抽出

上記ユースケース図からドメインを抽出しドメイン構造図を作成した。



図―10　ドメイン構成図

要求仕様から制御管理対象を抽出する。(温度の管理、水位の管理など)この管理1つがスレッドクラスに対応するようにマッピングする。抽出した結果を下記に示す。

①フタ管理:フタが開/閉したかを管理する
②ポット管理:ポットの全動作を管理する。並びにLED 表示の制御を行なう。
③操作パネル管理:操作各ボタンの入力を管理する。押されたキー情報をポット管理に送る。
④給水管理:給水動作を制御する。水位情報を管理する。水位 LED を制御する。
⑤温度管理:ポットの温度制御管理を行なう。

ここで各ドメインが行なう責務が明確になる。

3) ドメイン間メッセージ仕様作成

次に各ドメイン間で送受されるメッセージの仕様を抽出した。メッセージの送受するルーチンは KSThread クラスを利用するクラス内の Run()メソッドに割り付けられ、Run()メソッド内でメッセージの分析が行なわれる。

下記に抽出したメッセージ仕様を示す。

表―1　ドメイン間メッセージ仕様表

ポット管理クラス		メッセージ	温度管理クラス
温度情報を問い合わせる	→	ASKTEMP	
	←	CURRENT_TEMP:***.**	現状の温度を***.**でポット管理クラスに返す
沸騰要求を行なう***で高温保温/節約保温/ミルク保温情報を0，1，2、で送る	→	BOILMODE:**	
	←	BOIL_END	沸騰終了になった時点で沸騰終了イベントを返す

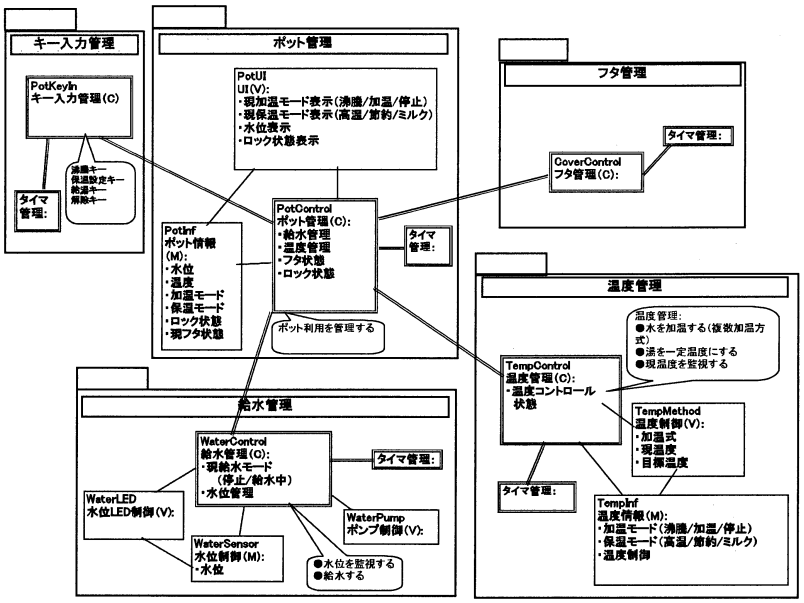
ポット管理クラス		メッセージ	給水管理クラス
給水ポンプOFF要求	→	WATER:0	
給水ポンプON要求	→	WATER:1	
水位情報問合せ	→	ASKWATER	
	←	A_WATER_LEVEL:*	水位情報送信を返す(センサーレベル:0、1、2、3)

ポット管理クラス		メッセージ	フタ管理クラス
蓋の状態を問い合わせる	→	ASKFUTA	
	←	A_FUTA:*	蓋情報問い合わせへのレスポンスコマンド0:蓋オープン　1:蓋クローズ

ポット管理クラス		メッセージ	キー入力管理クラス
	←	BOILSW:*	沸騰SWの状態が変化したとき＊に0:OFF、1:ONを送る
	←	FILLSW:*	給湯SWの状態が変化したとき＊に0:OFF、1:ONを送る
	←	KEEPSW:*	保温SWの状態が変化したとき＊に0:OFF、1:ONを送る
	←	NOTHOLDSW:*	解除SWの状態が変化したとき＊に0:OFF、1:ONを送る

4)ドメインごとのドメイン内クラスの抽出

各デバイスの実動作の制御を行う仕様を抽出し、クラスにする(制御クラスと呼ぶ)。それらクラスを関連するコントロールクラスの配下に置く。抽出したクラス図を下記に示す。



図—11 ポットコントロールクラス図

各管理コントロールクラスに関連する情報を情報クラスにしコントロールクラスの配下に置き、またデバイス制御用のクラスも関連するコントロールクラスに配置した。

結果を下記に示す。

①給水管理(WaterControl)
　-水位 LED 制御(WaterLED)
　-水位情報(WaterSensor)
　-ポンプ制御(WaterPump)
②温度管理(TenpControl)
　-温度制御(TempMethod)
　-温度情報(TempInf)
③ポット管理(PotControl):
　-ポット LED 制御(PotUI)
　-ポット情報(PotInf)
④フタ管理(CoverControl)
⑤操作パネル管理(PotKeyIn)

5)各実動作仕様の抽出・クラス詳細化実施

各ドメイン内のクラス構成は、デザインパターンである Model(情報クラス)、View(デバイス制御クラス)、Controller(管理クラス)となるようにクラスを配置している。ドメイン内の大まかな処理は下記のように記載した。

①コントロールクラスにメッセージが到着するとメッセージを解析し、関連する要求メッセージの場合、必要な情報を情報クラス(Model)に設定する
②コントロールクラスは、デバイス制御クラス(View)に処理更新(update)を実行する。
③デバイス制御クラス(View)は情報クラス(Model)クラスから情報を取得し、実際にデバイスを更新する上記の処理の利点は、処理手続きを決まった手順でプログラムコーディングできる点で、プログラムミスの低減に役立つという点である。

図－12に参考として、ポット管理のソースを一部抜粋し示す。

3. 3評価

テストサンプルをスレッドクラスを用いて仕様分析し、実装・動作確認をした結果を下記に示す。

①スレッドライブラリを用いてスレッド動作を行うアプリケーションを開発することは充分可能であると考える
②温度の管理、水位の管理など制御管理対象をコ

ントロールする仕様を1つのスレッドに対応させるという切り分けを行なうことで仕様の展開が容易であること

```
class PotControl : public KSThread {
    pthread_t   m_tid;
    pthread_t   m_TotId;
    int m_timer;
    char r_buf[20];
    char s_buf[20];
    PotInf *potInf;
    PotUI *potui;
    TOKEN token;

public:
//コンストラクタ、Run()関数部分のみ抜粋表示
    PotControl(){
        potInf=new PotInf;
        potui=new PotUI(potInf);
    }
    ~PotControl(){}
protected:
    int Run() {
        KSMail *pMail;
        char temp[20];
        double dt;
        while(1){
            pMail = ReceiveMail();
            memset(r_buf,0,sizeof(r_buf));
            strcpy(r_buf,pMail->m_cBuf);
            printf(" Mail=%s ¥n",r_buf);

            Ctoken ot(1);ot.GetToken(
                                r_buf,strlen(r_buf),&token,":¥r¥n");
            if(strcmp(token.token[0],"SOL_SW")==0){
                execute_kaon_mode();
            }else if(strcmp(token.token[0],"FILL_SW")==0){
                if( strcmp(token.token[1],"1")==0){ dt=1;
                }else{                              dt=0;  }
                potInf->Set_fill_mode(dt);
                execute_fill_mode(dt);
            }else if(strcmp(token.token[0],"KEEPSW")==0){
                execute_hoon_mode();
            }else if(strcmp(token.token[0],"NOTHOLDSW")==0){
                execute_nothold_mode();
            }else if(strcmp(token.token[0],"TIMER")==0){
                execute_get_temp();
                execute_get_futa();
                execute_get_water_level();
            }else if(strcmp(token.token[0],"CURRENT_TEMP")==0){
                memset(temp,0,sizeof(temp));
                strcpy(temp,(char*)token.token[1]);
                dt=atof(temp);
                set_current_temp(dt);
            }else if(strcmp(token.token[0],"A_FUTA")==0){
                if( strcmp(token.token[1],"1")==0){ dt=1;
                }else{                              dt=0;  }
                potInf->Set_futa_mode(dt);
            }else if(strcmp(token.token[0],"A_WATER_LEVEL")==0){

                if( strcmp(token.token[1],"0")==0){
                    dt=0;
                }else if (strcmp(token.token[1],"1")==0){
                    dt=1;
                }else if (strcmp(token.token[1],"2")==0){
                    dt=2;
                }else if (strcmp(token.token[1],"3")==0){
                    dt=3;
                }else if (strcmp(token.token[1],"4")==0){
                    dt=4;
                }
                potInf->Set_water_level(dt);
            }
            ot.FreeToken(&token);
            delete pMail;
        }
    }
};
```

図—12 ポット管理クラスソースサンプル

が分かった。このドメイン分析を行なう作業は、多分に経験則によるところが多いという感触を得た。このドメイン分割が適切でないと、不要なメッセージ通信が発生しスレッド間通信のオーバヘッドが発生すると考えられた。

③メッセージ仕様を定義しドメイン間の通信シーケンスを記述するのが容易であった

今回は送信情報量が少なく排他制御を利用するクラスは発生しなかったが、開発する物件によっては発生する可能性がある。多量のデータを扱うクラスがある場合がこれに相当するスレッド間でデータを送る場合データの切り出しが多く発生し、時間的要求がある場合不利である。クラスで排他制御をコントロールできるコントロール基本クラスが必要である。

4. おわりに

今回、組込み Linux 用のスレッドクラスライブラリを開発したが、組込みLinuxにおける制御用のソフトウェア開発に本スレッドクラスライブラリを適用することは、充分可能であると考える。また、製品開発に適用が可能であると考えられた。

また、スレッドクラスを用い、オブジェクト指向を取り入れる際①ドメインを明確にする②ドメイン間のメッセージを抽出する③ドメイン内のクラスをModel,Viewl,Controllerの観点から分析するという作業を行なうことにより比較的スムーズにクラス化ができ実装することが可能であった。

今回ハードウェアをシュミレートするツールを作成し、PC(Linux)の環境下でソフトウェア開発を行い動作確認も PC 上で行なったが、動作したソースプログラムは、実機用の Linux 搭載基板の CPU 用クロスコンパイルで実行モジュールを作成することにより実機基板で動作ができる。実際に自社内の基板で動作を試み動作できることを確認した。

今まで、基板搭載用のプログラム開発では対応するICEを用いたプログラム開発を行なっていたが、組込み LinuxOS を用いることによりソフトウェア開発を PC 上で開発しほぼバグのない状態にした後、実機基板に搭載することにより、効率のよいソフトウェア開発が可能になると考える。

また、PC 上のソフトウェア開発でも Eclipse 等の開発環境を積極的に利用することにより、効率の高い開発が可能になると考える。

参考文献

[1] http://www.nslabs.jp/pthread.rhtml

[2] http://www.linux.or.jp/JM/html/LDP_man-pages/man7/pthreads.7.html#lbAE

[3] 組込みソフトウェア管理者・技術者育成研究会(SESSAME) http://www.sessame.jp/

[4] 三瀬敏朗ほか:組込みソフト非正常系における基礎モデル及び仕様分析手法の提案 情報処理学会ソフトウェア工学研究報告 No.29、2005

[5] 渡辺博之、渡辺雅彦、堀末和人、渡守武和記、組込み UML、株式会社翔泳社

[6] SESSAME WG2、組込みソフトウェア開発者のためのオブジェクト指向モデリング