第15章

サービス・コールの概要(1)
――タスクの生成と終了

岸田 昌巳

本章からは，第4章で解説したシミュレーション環境を使ってμITRON4.0の各サービス・コールについて説明します．Windows上でシミュレーション環境を使うため，Microsoft Visual C++ 6.0/Visual Basic 6.0が必要となります．

また，サービス・コールの説明ということもあり，初心者でも理解できるように，μITRON4.0仕様を網羅した全般的な解説になっています．

1. μITRONの特徴

組み込み用リアルタイムOS(RTOS)として大きなシェアを持つμITRON仕様は，以下の設計方針に基づいています．仕様書に明記されているので，ご存じの方も多いことでしょう．

1) ハードウェアを過度に仮想化することを避け，ハードウェアに対する適応化を考慮する
2) アプリケーションに対する適応化を考慮する
3) ソフトウェア技術者の教育を重視する
4) 仕様のシリーズ化やレベル分けを行う
5) 豊富な機能を提供する

この5項目については前章までの解説やμITRON4.0仕様書に詳細な説明があるので，詳細はそちらを見てください．

ここでは初心者が理解するために必要な部分に注目してみましょう．一貫性のある用語の使い方や名称の付け方は，理解を深めるためにも，知識として蓄積するためにも効果があります．サービス・コールの名前の付け方については**表1**を参照してください．

これらを覚えておくだけで，サービス・コールが類推できるようになります．また，このわかりやすさもμITRONの特徴の一つです．

2. タスク関連のサービス・コール

サービス・コールの説明に入る前に考慮しておきたい点を少し述べます．

● タスク分割に関して

システムの概略設計段階や構成を検討する段階で，どのような方針でタスクを分割するかを考えると思います．

この検討の中で，タスクをいくつ用意するのかが決まります．具体的な分割の方法は，システム全体で提供している機能を部分機能ごとに分割する方法や，デッドラインをもとに分割する方法など，複数の種類があります．

また逆に，いったんは機能別に分割したタスクを，性能を上げるために再度結合することもあります．結合する理由としては，全体のふるまいから必要ないタスク・スイッチなどが減るように処理のむだを省く，いわゆるチューニングに伴う変更の事例が多いようです(**図1**)．

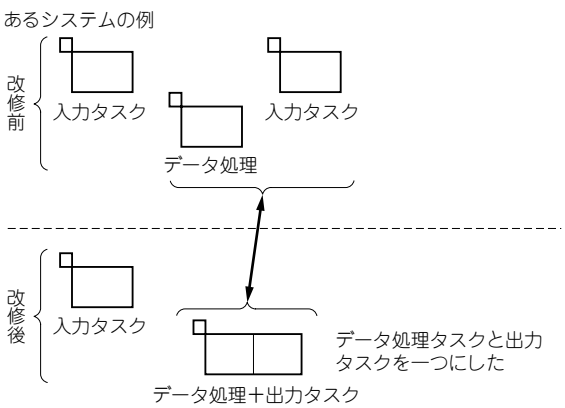

図1　タスク分割，結合

第16章

サービス・コールの概要(2)
――タスク管理機構と同期機構

岸田 昌巳

1. タスク付属同期機能

TOPPERS/JSPでサポートしているタスク付属同期機能にはslp_taskとtslp_task，wup_task，iwup_task，can_wup，rel_wai，irel_wai，sus_task，rsm_task，frsm_task，dly_taskの11個があります．これらは**図1**のように関連性を持った対になっています．

◆ 起床待ち

C言語API
ER slp_tsk(void); ER tslp_tsk(TMO tmout);
パラメータ
TMO tmout; タイムアウト指定
リターン・パラメータ
ER E_OK(正常終了)またはエラー・コード
エラー・コード
(E_SYS), (E_NOSPT), (E_RSFN), (E_MACV), (E_OACV), (E_NOMEM), E_CTX, E_PAR, E_RLWAI, E_TMOUT E_CTX　：コンテキスト・エラー 　(待ち状態に入れない状態で呼び出した) E_PAR　：パラメータ・エラー(待ち時間が不正) E_RLWAI：待ち状態の強制解除 　(待ち状態の間にrel_waiを受け付けた) E_TMOUT：ポーリング失敗またはタイムアウト 　(tslp_taskのみ)

slp_taskのサービス・コールを呼び出したタスク(自タスク)は，起床要求があるまで待ち状態に入ります．この起床要求はキューイングされます．このため，slp_taskのサービス・コールを呼び出す前に，wup_taskのサービス・コールで起床要求が発行された場合は待ち状態に入りません．

また，TOPPERS/JSPではこのキューイングは1回しか行われません．これはJSPカーネルではwup_taskの起床要求はact_taskと同じように，1ビットのキューイング用バッファをTCB内に用意しているためです．

タイムアウトにはTMO_FEVR，TMO_POLが指定できます．

● 返り値に関して(slp_tsk，tslp_tsk)

ディスパッチ保留の場合にはE_CTXが返ります．ディスパッチ保留状態とは，タスク外から呼び出した場合やCPUロック状態へ移行した場合が該当します．ディスパッチ保留状態では待ち状態に入ることができないため，slp_taskはエラーを返します．

tslp_taskを使用して，待ち時間の指定におかしな値を渡した場合はE_PARが返ります．

このタイムアウトを指定するパラメータの型はTMO型で，符号付き整数型です．

待ち時間の指定は正の値になりますが，ゼロとマイナス側をシステム側で使用しているので，ここでパラメータに使用できないTMO_NBLKを指定した場合や，システム側で使用していない負の値を指定した場合にはE_PARが返ります．

待ち状態の間にrel_waiを受け付けた場合は，E_RLWAIを返すことで自タスクが待ち状態から強制解

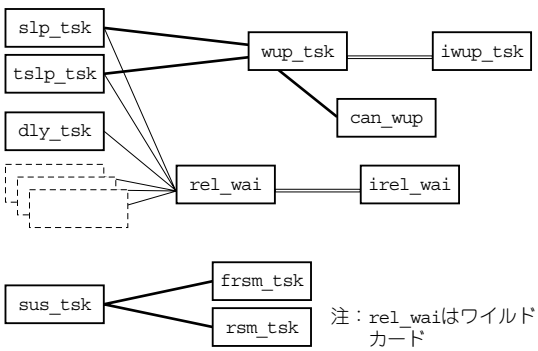

図1　各サービス・コールの関連

第17章

サービス・コールの概要(3)
――セマフォ，イベント・フラグ，データ・キュー

岸田 昌巳

1. 同期・通信機能のサービス・コール

μITRON4.0の同期・通信機能としては，セマフォやイベント・フラグ，データ・キュー，メール・ボックスがあります．これらは，同期機能と通信機能を別々に提供しているのではなく，どちらの用途にも使える機能が提供されています．

これらのサービス・コールとしては，xxx_sem（xxxはsigやwaiなど）やxxx_flg（同じくxxxにはsetやwaiなど），xxx_dtq（xxxはsndやrcvなど），xxx_mbx（xxxはsndやrcvなど）があります．

表1に，TOPPERS/JSPでサポートしているこれらの機能を掲載します．

状態遷移図から見ると，それぞれの待ちに入るサービス・コールと待ちを解除するサービス・コール以外に，強制的に待ち状態を解除するサービス・コールがタスクのふるまいに関連します（**図1**，**図2**）．

2. セマフォ

● セマフォの概要

セマフォは排他制御に使用します．排他制御は，同時に利用できないクリティカル・セクションを複数のタスクから共有する場合など，「自分が使用している途中なので，ほかから使用されないようにする」という，ほかを排除したい/使わせたくない場合に使います．

ここで言う管理対象のクリティカル・セクションとは，

- 一度に一つのタスクしかアクセスしてはいけない関数
- あるアドレスのデータ（変数）
- 入出力を行うI/Oなど

といったリソースやコードを指しています．

これらのクリティカル・セクションにアクセスする場合には排他制御が必要になります．たとえば，非同期に動作する複数のタスクが同じアドレスのメモリを書き換えたい場合は，セマフォを使用して排他制御を行います（**図3**）．

表1　同期・通信機能のサービス・コール一覧

セマフォ
ER　sig_sem(ID semid); ER　isig_sem(ID semid); ER　wai_sem(ID semid); ER　pol_sem(ID semid); ER　twai_sem(ID semid, TMO tmout);
イベント・フラグ
ER　set_flg(ID flgid, FLGPTN setptn); ER　iset_flg(ID flgid, FLGPTN setptn); ER　clr_flg(ID flgid, FLGPTN clrptn); ER　wai_flg(ID flgid, FLGPTN waiptn, 　　MODE wfmode, FLGPTN *p_flgptn); ER　pol_flg(ID flgid, FLGPTN waiptn, 　　MODE wfmode, FLGPTN *p_flgptn); ER　twai_flg(ID flgid, FLGPTN waiptn, 　　MODE wfmode, FLGPTN *p_flgptn, 　　TMO tmout);
データ・キュー
ER　snd_dtq(ID dtqid, VP_INT data); ER　psnd_dtq(ID dtqid, VP_INT data); ER　ipsnd_dtq(ID dtqid, VP_INT data); ER　tsnd_dtq(ID dtqid, VP_INT data, 　　TMO tmout); ER　fsnd_dtq(ID dtqid, VP_INT data); ER　ifsnd_dtq(ID dtqid, VP_INT data); ER　rcv_dtq(ID dtqid, VP_INT *p_data); ER　prcv_dtq(ID dtqid, VP_INT *p_data); ER　trcv_dtq(ID dtqid, VP_INT *p_data, 　　TMO tmout);
メール・ボックス
ER　snd_mbx(ID mbxid, T_MSG *pk_msg); ER　rcv_mbx(ID mbxid, T_MSG **ppk_msg); ER　prcv_mbx(ID mbxid, T_MSG **ppk_msg); ER　trcv_mbx(ID mbxid, T_MSG **ppk_msg, 　　TMO tmout);

第18章

サービス・コールの概要(4)
――メール・ボックス

岸田 昌巳

1. メール・ボックス

● メール・ボックスの概要

表1にメール・ボックスに関するサポート機能の一覧を示します．

メール・ボックスの機能は，メッセージを実行中のタスクからほかのタスクに送付するものです．メッセージとは，送りたいデータをタスク間で共有可能なメモリ上に書いたもののことです．μITRON4.0仕様のカーネルでは，このメッセージを扱うためのサービス・コールとしてメール・ボックスを用意しています．

このメール・ボックスにより，メッセージを使って同期や通信を行うことができます．この同期や通信の相手は，タスク間を対象としています．

このメッセージを書くためのメモリ領域はアプリケーションで確保します．メモリを確保する方法は任意ですが，TOPPERS/JSPではメモリ領域を確保するためのサービス・コールも用意されており，通常はこのサービス・コールを使用します．このメモリ領域を管理する機能のことをメモリ・プール管理機能といいます．なお，サービス・コールの説明の順序から，このメモリ・プール管理機能で用意しているサービス・コールは次章で説明します．

ここまでは機能の説明でしたが，メール・ボックスの機能を中身から見ると，もう少しわかりやすくなるかと思います．処理の流れとしては，**図1**のようになっています．

● タスク例外などの処理から

メール・ボックスの機能は，タスク間を対象としているため，タスク以外の処理で使用するixxx_yyyというサービス・コールは用意されていません．タスク外で使用可能なサービス・コールを**表2**に示します．つまり，ほかのタスク管理機能や同期・通信機能では，タスク間のやりとりだけでなく，タスク以外からもイベントをタスクに送ることができました．けれども，メール・ボックスではできないということをこの表は示しています．

● メール・ボックス使用の勧め

TOPPERS/JSPカーネルでは，同期や通信を行うためのサービス・コールには，セマフォとイベント・フ

表1　メール・ボックス機能のサービス・コール一覧

メール・ボックス	
ER	snd_mbx(ID mbxid, T_MSG *pk_msg);
ER	rcv_mbx(ID mbxid, T_MSG **ppk_msg);
ER	prcv_mbx(ID mbxid, T_MSG **ppk_msg);
ER	trcv_mbx(ID mbxid, T_MSG **ppk_msg, TMO tmout);

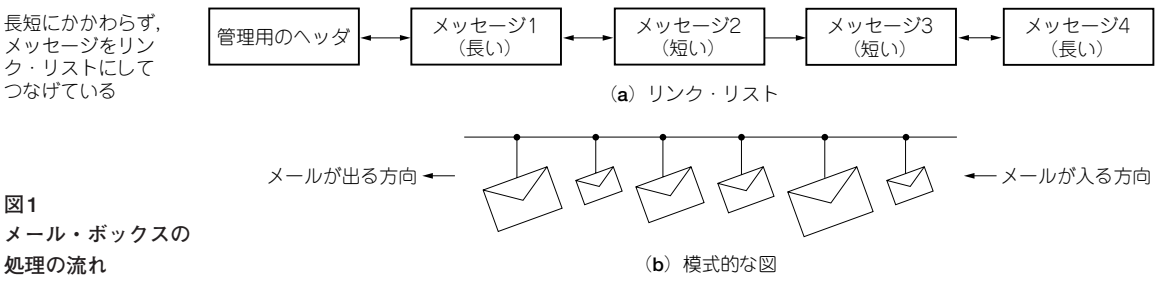

図1
メール・ボックスの処理の流れ

第19章

サービス・コールの概要(5)
――システム状態管理機能

岸田 昌巳

ここまでの章では，シミュレーション環境を使いながら各サービス・コールについて説明してきましたが，本章ではシステム状態管理機能の解説を行います．

1. システム状態管理機能

TOPPERS/JSPカーネルでは，システム管理機能として**表1**に示す10項目と，μITRON4.0仕様にはない項目を一つ提供しています．

ここでいう「システム」とは，カーネルからアプリケーションまで含めたものを指し，これらのサービス・コールでアプリケーションからシステム全体を操作することができます．

サービス・コールとして提供している機能は，カーネルの状態を把握したり，ディスパッチの許可/禁止，CPUのロックなどです．大まかには「タスクに関するもの」と「システム全体の状態を変えるもの」があります．

タスクに関するものは，優先順位の回転や現在実行中のタスクを知るためのものです．これらは，TSS(タイム・シェアリング・システム)のように処理をまんべんなく行うために優先順位を回転させたり，複数のリクエストに応えるため，UNIXのinetd(インターネット・デーモン)のように，同じ処理をタスクとして複数登録し，個々に自分のプロセスID(処理単位を判別するためのID，ITRONにおけるタスクIDのようなもの)を取得してふるまいを変えるようにするため用意されています．

システム全体の状態を変えるものは，割り込み禁止許可の状態やCPUのロックなど，そのシステムに一つしかないものの状態を対象にしています．これらは，割り込み処理を記述するため，割り込みを禁止して処理したい場合などに，カーネルへ指示を与えるために使用します．特にこのサービス・コールは，システムに一つしかないものの状態を変えているため，処理の抜けや禁止後の許可のし忘れなどに注意する必要があります．

最後に，μITRON4.0仕様にはない項目「カーネル動作状態の参照」は，JSPカーネルの初期化が終了したかどうかを判断可能にします．これは機能拡張を行う際に必要となります．

表1
TOPPERS/JSPカーネルのシステム管理機能

μITRON4.0仕様
タスクの優先順位の回転
実行状態のタスクID の参照
CPU ロック状態への移行
CPU ロック状態の解除
ディスパッチの禁止
ディスパッチの許可
コンテキストの参照
CPU ロック状態の参照
ディスパッチ禁止状態の参照
ディスパッチ保留状態の参照
TOPPERS/JSP
カーネル動作状態の参照

● タスクの優先順位の回転

タスクの優先順位の回転については**図1**をご覧ください．実行可能状態にある，tskpriで指定した優先度のタスクのキュー(レディ・キュー)の優先順位の一番高いものを取り出し，優先度の一番低いところにつなぎます．

つないだ順番がそのままタスクの優先度となるため，先ほどまで優先度の一番高かったタスクが，一番低くなります．

◆ タスクの優先順位の回転

C言語API
`ER rot_rdq(PRI tskpri);` `ER irot_rdq(PRI tskpri);`
パラメータ
`PRI tskpri`　　タスクの優先度

第20章

μITRON4.0のアプリケーション sample1を読む

邑中 雅樹

本章では，ソース・コードを追いながら，TOPPERS/JSPがどのようにサンプル・プログラムを処理・実行しているのかについて説明します．

1．μITRON4.0仕様書を片手に…

TOPPERS/JSPは，「μITRON4.0仕様スタンダード・プロファイル」を忠実に実装したものです．TOPPERS/JSP自身にも添付の解説文書が存在しますが，カーネル作成者からの視点で書かれている点は否めません．ある程度理解が進むと使える資料なのですが，最初の文書としてはややハードルが高いかもしれません．

そこで，技術者のみなさんには，社団法人 トロン協会が発行している「μITRON4.0仕様書」を参考書として手元に置いて学習されることをお勧めします．μITRON4.0仕様書は，最新のVer4.02がPDF形式で存在し，

```
http://www.ertl.jp/ITRON/SPEC/
                              mitron4-j.html
```

から無償でダウンロード可能です．μITRON4.0仕様書も図版が少なく，入門書としては難解です．しかし，TOPPERS/JSP添付の文書よりはわかりやすいものとなっています．

2. 標準サンプル・プログラム sample1

● カーネルといっしょに配布されているsample1

本章で扱うサンプル・プログラムsample1は，TOPPERS/JSPカーネルの公式リリースにも収録されています．標準的に配布されており，TOPPERS/JSPを使ったアプリケーション開発者がかならず最初に実行するといっても過言ではありません．

sample1は，スタック・サイズなど，ターゲットに大きく影響を受けるいくつかの例外を除き，ターゲット非依存な構成になっています．同時に，TOPPERS/JSPカーネルを各種ターゲット・ボードに移植する際の簡単なテスト・プログラムも兼ねており，TOPPERS/JSPカーネルでアプリケーションを作成するうえで肝となる部分が適度な長さの中に凝縮されています．

● どうやってsample1を理解するか

sample1には短いソース・コードの中にμITRON仕様OSの特徴を知るためのいろいろな要素が含まれています．しかし，限られた紙面ですべてを説明し尽くすのは困難です．そこで，「タスク」，「周期ハンドラ」，「割り込み」の三つの観点からsample1を解説していきます．

3. タスクからsample1を追う

● sample1も複数のタスクから構成されている

まずは，タスクという切り口から入ってみましょう．オペレーティング・システム（OS）の話題でよく使われる「マルチタスク」という言葉のとおり，TOPPERS/JSPカーネルを用いたアプリケーションの多くは，複数のタスクで構成され，実行されます．

● main文はどこにあるのか

どのようなプログラムでも，処理の開始から順に追っていくのが定石です．処理の頭を探すことにしましょう．

その前に，C言語でのプログラミングの基礎の基礎である，Hello Worldを思い出してみましょう．典型的な例は，以下のようなものです．

```
int
main(int argc, char **argv)
```

第21章

C++ APIテンプレート・ライブラリを作ったプログラミング

高木 信尚

1. TOPPERS C++ API テンプレート・ライブラリとは

この章では，TOPPERS C++ APIテンプレート・ライブラリについて解説を行います．

このTOPPERS C++ APIテンプレート・ライブラリ（以下，C++ APIテンプレート・ライブラリ）は，μITRON4.0仕様に準拠したカーネルのAPIをC++で使えるようにラッピングしたテンプレート・ライブラリです．μITRON4.0仕様は，APIをC言語の関数として定義しています．そのためμITRON4.0を使った開発は自然とC言語で行うことになっていましたが，本ライブラリを用いることによって，C++言語での開発が容易に行えるようになります．

このライブラリは，TOPPERSプロジェクトから公開されているTOPPERS/JSPカーネル（スタンダード・プロファイル準拠カーネル）およびTOPPERS/FI4カーネル（フルセット・カーネル）のどちらとも組み合わせて使用できるように設計されています．なお，C++ APIテンプレート・ライブラリは，TOPPERSプロジェクトのWebサイトからダウンロードできます[注1]．

C++ APIテンプレート・ライブラリは，情報処理推進機構（IPA）によるオープンソフトウェア活用基盤整備事業の採択テーマの一つである「μITRON4.0仕様に完全準拠し拡張を含むオープンソースμITRON仕様OSの開発」の一環として，2003年から2004年にかけて開発されました．

TOPPERS/JSPカーネルのバージョン1.4，および同じ採択テーマの一環であったTOPPERS/FI4カーネルと並行して開発を行った結果，μITRONカーネルの環境でC++を利用するうえで必要になるカーネル側のくふうや拡張を，JSPおよびFI4カーネル本体にも反映することができました．

注1：http://www.toppers.jp/cxx-download.html

2. C++ APIテンプレート・ライブラリの実際

● ライブラリを使ったサンプル

それでは，C++ APIテンプレート・ライブラリとはどのようなものなのか，イメージをつかんでいただくために，**リスト1～リスト3**に簡単なサンプルを示します．

C++のテンプレートになじみの深い方が見れば，上記のような簡単なサンプル・コードでも，テンプレートの特徴である<・・・>という記述がいくつも現

リスト1　C++で書いたμITRONプログラム
（my_task.cpp）

```
// my_task.cpp
#include <toppers/task.hpp>
#include "my_task.h"
#include "kernel_id.h"

class my_task_body : public toppers::task_body<>
{
public:
    explicit my_task_body(VP_INT exinf)
        : toppers::task_body<>(exinf)
    {
        // タスク初期化処理
    }
    void run()
    {
        // タスク主処理
    }
    ~my_task_body()
    {
        // タスク終了時処理
    }
};

extern "C"
void main_task(VP_INT exinf)
{
    toppers::task<MY_TASK> my_task;
    my_task.create<my_task_body>();
}
```

第22章

TOPPERSの開発をモデル駆動型で行う

竹内 良輔

これまでの章では，TOPPERSで動作するプログラムはおもにC言語を用いて書いていました．本章では，モデル駆動型開発（MDD）という新しい開発技術の紹介と，この技術を用いた開発作業と従来開発の作業を比べ，どのように異なるかを，TOPPERS上で動作するC言語ソースを例にして紹介します．

1. モデル駆動型開発（MDD）とは何か？

● UMLをベースとした開発へ移行しつつある

組み込みソフトウェアは大規模化，複雑化の一途をたどっています．これに対応してUMLを用いたモデル化によるソフトウェアの可視化が可能になりました．

MDDは，Model Driven Developmentの略です．オブジェクト指向の標準化団体であるOMGは，MDA（Model Driven Architecture：モデル駆動型アーキテクチャ）[注1]という開発技術の標準化を行っています．モデル指向の開発技術のうちでもUMLにこだわらないものをMDDと呼びます．

● モデルからソース・コードを生成する

モデル駆動型開発では，モデルから実行可能なソース・コードとそれを実行するためのメカニズムを生成します．ここでいう「モデル」はPIM（Platform Independent Model）と呼ばれ，要求仕様のみを含んだ，実装に依存部を含まないモデルです．このモデルから実装に依存したモデルやソース・コードを生成することによって開発を行います．

「生成されたソース・コード中に実行メカニズムをもつ」ということばの意味は，リアルタイムOSにおけるタスク機構と同等の機能をもつということです（複数のソース・コードにそのつど，実行権を与えて実行する）．

この部分の詳細を，MDDの開発手法の一つであるExecutable UMLの代表的なツールBridgePoint（Mentor Graphics社）の実行メカニズムを例に説明します．

2. Executable UMLでのモデル駆動型開発

● 要求仕様モデル――PIMの作成

Executable UMLではPIMを**図1**のような三つの流れで作成します．

1）クラス図の作成

まず，データに視点をおき，クラスとその関係を静的構造モデルとして表します．これにはUMLの「クラス図」を用います．

2）制御のモデル化――アクティブ/パッシブ・クラス

次に，クラスの動作に視点をおき，制御のモデル化を行います．

クラスには状態が時々刻々と変化する「アクティブ・クラス」と，データと関数の保持を目的とし，状態を持たない「パッシブ・クラス」があります．制御の対象となるのはアクティブ・クラスです．アクティブ・クラスとは，あたかもタスクを割り当てられたように動作するクラスで，状態の変化はイベントによって発生します．イベントはμITRON4.0仕様のデータ・キューと同じような構造をしており，自分またはほかのアクティブ・クラスがイベントを発行し，受け取ったクラスが別の状態に変化します．このようすはステート・チャートで記述します．

3）アクション記述言語

最後に手続きの視点から，状態の詳細な動作をアクション記述言語という言語を用いて記述します．アクション記述言語は通常のプログラミング言語ではな

注1：MDAはOMG（Object Management Group）の登録商標．