第2章 ITRON仕様の概要

2.1　ITRON仕様とは

1987年にITRON1，1989年にITRON2とμITRON2，1993年にμITRON3.0，1999年にはμITRON4.0の仕様が，トロン協会から公開されています．ITRONは仕様の名称であり，特定の製品の名称ではありません．したがって，ITRONカーネルというものは存在しません．まずITRONという仕様があり，その仕様に準拠したOSをITRON仕様準拠カーネルというのです．ITRON仕様準拠カーネルは，半導体メーカやOSメーカなどから製品として出荷されているだけで数十種類あります．

ITRON仕様はトロン協会が規定した仕様で，内容がオープン(公開)になっています．ITRON仕様は「ゆるい標準化」をテーマにし，カーネルを作りやすい仕様としています．そのため，ITRON仕様準拠カーネル間での互換性に問題が出てきました．そこでμITRON4.0仕様では，プロファイルの概念を導入し，少し強い標準化を示して，互換性の問題に少しでも対応しようとしています．2002年には，μITRON4.0仕様の拡張版として保護機能を拡張したμITRON4.0/PX仕様が公開されています．この仕様では保護機能が追加されています．

μITRON4.0の仕様書は，社団法人トロン協会のWebページ[注1]からダウンロードできます．リアルタイムOSの仕様書なので，カーネルの動作や機能の仕様も記載されており，ITRON仕様準拠カーネルの開発者向けになっています(仕様書のすべてを理解するためには，OSについての基礎知識が必要)．本書では，ITRON仕様準拠カーネルを使う立場に立って解説していきます．

2.2　組み込みシステムの内容

カーネルは，タスクとタスク，ハンドラとタスクで事象の発生通知や情報の受け渡しを行うための手段を提供します．カーネルは，事象通知のためにセマフォやイベント・フラグ，情報通知のためにメールボックスやメモリ・プールなどの機能をもっています．ITRON仕様ではこれらをカーネル・オブジェクト(もしくは単にオブジェクト)といいます．オブジェクトはカーネルにより管理されており，プログラムはこのオブジェクトに対して各種の要求を行います(**図2.1**)．

ハードウェアからの事象に対する処理を行うプログラムは，システムの中ではタスクもしくはハンドラ[注2]として動作します．外部からの入力もしくは出力で割り込みを利用する場合の割り込み処理プログラムは割り込みハンドラといいます．一定の時間間隔で処理するプログラムや指定時刻に処理されるプログラムは，タイム・イベント・ハンドラといいます．

組み込みシステムのほとんどはタスクでの処理になります．上記したハンドラやタスクの処理との事象や情報

注1：http://www.assoc.tron.org/jpn/document.html
注2：ITRONでは，これらを「アクセス主体」といっている．

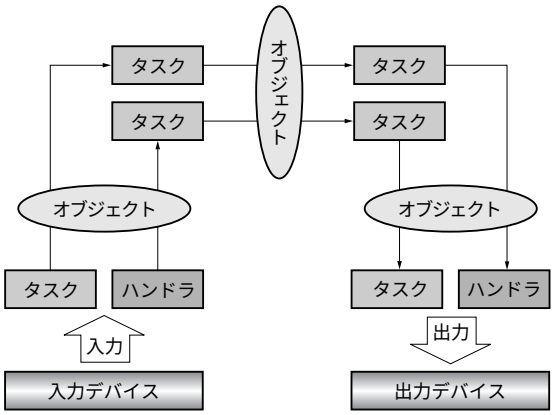


図2.1　事象や情報通知はオブジェクトを利用
タスク間やタスクとハンドラ間で事象通知や情報通知はカーネルのオブジェクトを利用する

図2.2　実行できるのは順番待ち

の通知に，カーネルが提供する機能を利用します．

2.3　オブジェクト

生成要求を行うことでオブジェクトを，カーネルに認識させます．オブジェクトにはID番号を割り付け，オブジェクトに対する要求(サービス・コール)をする場合，このID番号で指定します．システムに複数のタスクを生成できるように，オブジェクトも複数のシステムに生成することができます．システムにおいて必要な機能をもつオブジェクトを必要なだけ作成し，システムで利用します．ITRON仕様で規定しているオブジェクトは，いろいろな機能をもっています．それぞれのオブジェクトの機能を理解して利用するようにしましょう．

● よく利用されるオブジェクト

ITRON仕様で規定されているオブジェクトのなかから，よく利用されるオブジェクトについて，その機能の概要を説明します．詳細な説明やほかのオブジェクトについては『解説編』を参照してください．

▶ タスク

OSを利用した組み込みシステムのほとんどはタスクが実行します．OSによってタスクに設定される優先度に基づきタスクの実行が制御されます．また，OSは，タスクに対して「状態」をもたせることでスケジューリングを行います．タスクのいちばん大事な状態は，「実行」と「待ち」です．通常のプログラムであれば，一度実行を開始すると，プログラムを終了するまで，何かしらの命令(プログラム)を実行し続けます．CPUが一つのため，CPUが実行できるプログラムは一つです．複数のタスクにわたるプログラムを実行させようとした場合，OSはそれらのタスクの中でいちばん優先度の高いタスクに制御を渡します[注3]．ほかのタスクは，OSから制御が渡されるまで待ちの状態[注4]となります(**図2.2**)．

「待ち」の状態には，オブジェクトに対する事象待ち[注5]もあります．タスクは，対象となるオブジェクトを使って事象の通知を受けたり，あるいは資源[注6]を獲得できるまで待ちの状態となることができます．プログラムが目的とする事象が発生するまで待ちの状態になることで，その間，ほかのプログラムを実行できます．

OSを利用しない場合，事象が発生するまでプログラムでポーリング[注7]することになります．プログラムでポー

注3：CPUで実際に実行させることを「制御を渡す」という．
注4：この待ち状態は，実行できるが実行することができない状態ということで，「実行可能状態」という．
注5：こちらの事象待ちを，一般に「待ち状態」という．
注6：この資源とは，セマフォやメモリ・ブロックをいう．

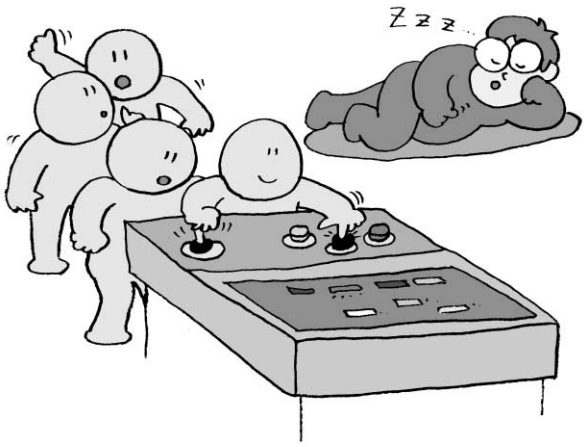


図2.3　待ち時間を使って，ほかのプログラムを実行する

リングしている時間を利用してほかのプログラムを実行させることができれば，CPUを有効に利用できることになります．OSを動作させ，タスクで事象の発生を待つ場合にその状態を「待ち状態」とし，ほかのタスクに制御が渡るようになっています．待ち対象の事象が発生すると，OSによって待ちの状態は解除されます(**図2.3**)．

上記のように，カーネルが提供する機能を利用し，実行したり待ち状態になり，そして事象や情報を通知しながら，いろいろなプログラムがシステムとしての処理を実行していきます．

▶セマフォ

セマフォは初期値をもったカウンタです．カーネルは，このカウンタが0のときに資源がない，1以上のときに資源があるとします．カーネルは，資源があるか/ないかを監視しています．また，セマフォには資源が獲得できるまでタスクを待たせる機能があります．タスクは，このセマフォに対して資源の獲得要求や返却要求を行います．

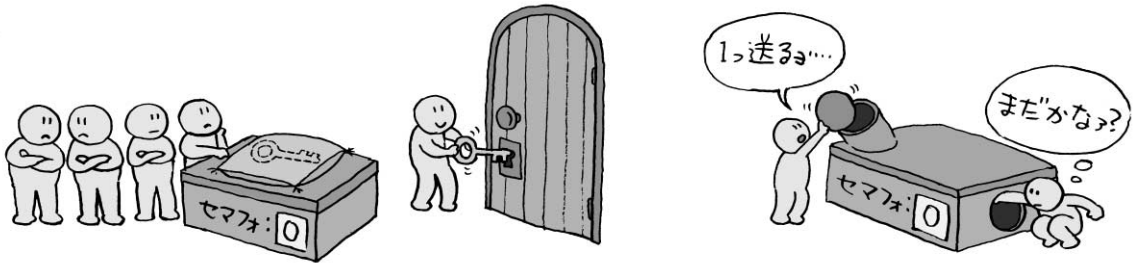
セマフォは二つの使い方をすることができます．一つは，排他制御に利用する方法です．セマフォのカウンタの初期値を1とし，排他制御したい処理の前にセマフォの獲得要求[注8]を行い，処理の後にセマフォの返却要求[注9]を行います．カーネルは，排他制御をしたい処理が何であるかは知りません．その処理が何であるかは，システムを設計する人が定義します．

もう一つは，事象の回数を記憶させる方法です．セマフォのカウンタの初期値を0とし，事象が発生したらセマフォの返却要求を行い，事象の通知を受けたい場合にセマフォの獲得要求を行います．たとえば，外部からデータを受信したらセマフォの返却要求を行います．そして，受信したデータに対する処理をするためにセマフォの獲得要求を行うことで，受信した回数分だけの処理を行うことができるようになります．カーネルは回数を記憶するだけで，その回数が何を対象にしているかは知りません．その回数が何であるかは，これもシステムを設計する人が定義します．システムに排他制御したい処理や回数を記憶したい事象がいくつかある場合は，設計者がそれぞれにセマフォを割り当てていきます(**図2.4**)．

▶イベント・フラグ

イベント・フラグとは，事象の発生を通知する旗です．0の状態と1の状態をもち，1の状態が事象発生を表すこととしています．イベント・フラグは，この0/1の状態をビットに対応させた変数をもっており，ビットが1になるまでタスクを待たせる機能をもっています．変数には，16ビットや32ビットなど[注10]があり，そのビットの数分の事象を同時に通知することができます．事象Aが発生した場合はビット0，事象Bが発生した場合は

注7：事象の発生が発生するまで，プログラムで検索しながら，待ち続けることになる．その間，CPUは命令を実行し続ける．
注8：「セマフォの獲得要求」はカウンタを－1する操作になる．
注9：「セマフォの返却要求」はカウンタを＋1する操作になる．
注10：イベント・フラグのビット数は，カーネルや動作させるCPUによって異なる．



（**a**）排他制御に使用する場合　　（**b**）同期（事象カウンタ）に利用する場合

図2.4　セマフォは排他制御や同期に利用できる

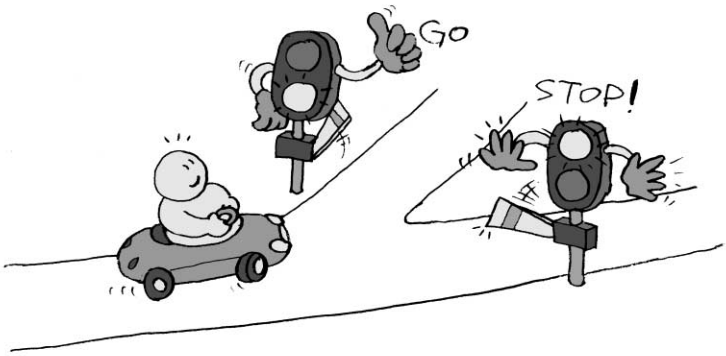


図2.5　イベント・フラグ管理
信号を見て，どのような事象が発生したかを知る

ビット1というように，事象を通知する側はそれぞれのビットを1にする要求[注11]を発行します．

事象を受け取るタスクは，複数の事象の発生を待つことができます．事象発生の通知を受けたら，イベント・フラグのビットを調べることで，どの事象が発生したかを知ることができます．カーネルは事象の発生/未発生を管理するだけで，それぞれの事象が何であるかは知りません．事象とそのビットの割り付けは，システムを設計する人が定義するものです．このようにシステムに通知したい事象がいくつもある場合は，それぞれにイベント・フラグを割り当てていきます（**図2.5**）．

▶ メールボックス

セマフォやイベント・フラグは，事象の発生を通知する目的をもったオブジェクトです．メールボックスというのは，メッセージという情報を通知する目的をもったオブジェクトです．このメッセージというのは，文字列であってもバイナリのデータであってもかまいません．

メールボックスは，よく郵便ポストにたとえられます[注12]．情報を送信したいタスクは，手紙（メモリ領域）にメッセージを書き込み，メールボックスに送信要求を行います．情報を受信したいタスクは，メールボックスに受信要求を行い，手紙を受け取ってメッセージを読み出します．メールボックスは，複数のメッセージを蓄えることができ，メッセージを受信するまでタスクを待たせる機能をもっています．システムに，メッセージの送受信をさせたいものがいくつかある場合は，それぞれにメールボックスを割り当てます（**図2.6**）．

▶ メモリ・プール

タスクで行うデータ処理のために一時的にメモリ領域が必要になる場合があります．そのようなときに，メモリ・プールの中からメモリ・ブロックを獲得し，そのメモリ・ブロックを利用します．利用し終わったら，必ずメモリ・ブロックをメモリ・プールに返却します．返却するのを忘れると，メモリ・プールから未使用のメモ

注11：イベント・フラグはビットが1で事象発生，0で未発生としている．
注12：実際は少し異なる．詳細は『解説編』を参照のこと．

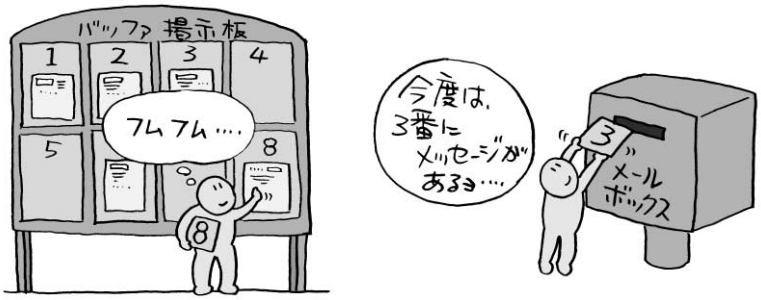


図2.6　メールボックスが通知するのは「バッファのアドレス」

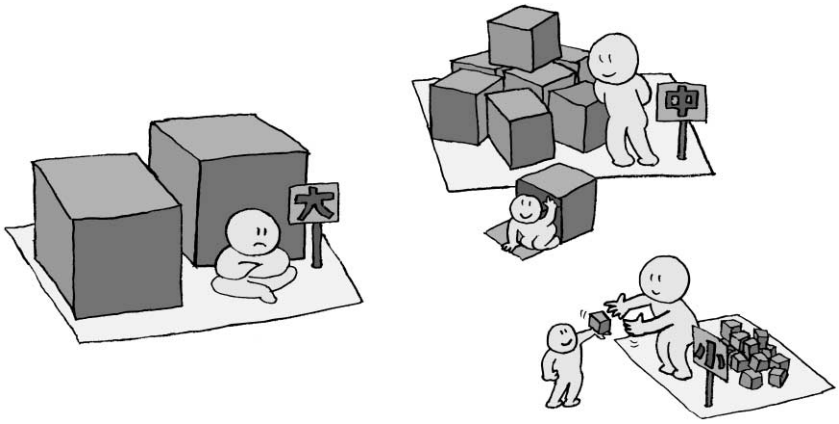


図2.7　メモリ・ブロックの利用

リ・ブロックがなくなり，メモリ・ブロックの再利用ができなくなってしまいます．

メモリ・プールには固定長メモリ・プールと可変長メモリ・プールがあります．固定長メモリ・プールは固定サイズのメモリ・ブロックを複数もっており，獲得要求を行ったタスクにそのメモリ・ブロックを与えます．また，可変長メモリ・プール[注13]は，タスクが要求するメモリ・サイズ分のメモリ領域をメモリ・ブロックとして与えます．

メモリ・プールは，複数のタスクから獲得要求ができます．システムで必要とするメモリ・サイズの複数のメモリ・プールを用意し，必要とするサイズにより，的確なメモリ・プールからメモリ・ブロックを獲得するようにします．メールボックスに送信するメッセージのためにメモリ・プールのメモリ・ブロックが，よく利用されます(**図2.7**)．

● 事象の待ち

ほとんどのオブジェクトは，事象が通知されるまでタスクを待たせる機構をもっています．この機構を使って，タスクは事象の待ちを行うことができます．事象の通知には，事象を通知する側と通知を受け取る側の二つがあります．

オブジェクトは事象の発生を記憶するため，事象の発生通知を一方方向[注14]にすることができます．そのため，事象を通知する側は，通知を受ける側の状態に関わらず，オブジェクトに対して通知を行えます．そして通知を受ける側は，オブジェクトに対して通知の有無を問い合わせます．このとき，事象の通知がされていない場合は，

注13：カーネルによっては，可変長メモリ・プールをサポートしていない場合がある．
注14：ランデブ機能を除く．ランデブ機能は，事象の通知側と事象の受け側で待ち合わせを行うことができる．

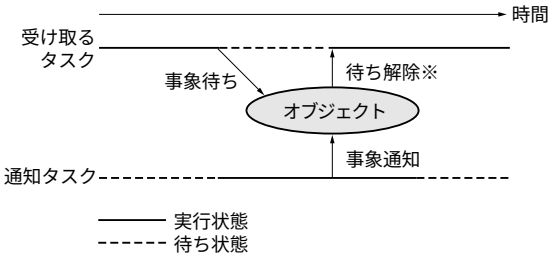


図2.8　事象を受け取るタスクが先に待っていた場合
※印の待ち状態解除時に，受け取るタスクの優先度が通知タスクより高い場合，事象通知の要求時にタスク切り替えが発生する．通知タスクの優先度が高い場合は，受け取るタスクの待ち状態が解除されるだけで，事象通知の要求から制御が戻ってくる

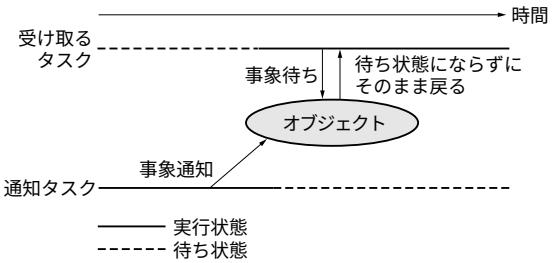


図2.9　事象の通知が先に行われた場合
先に事象通知が行われていると，受け取るタスクが事象待ち要求を発行しても，待ち状態にならずにそのまま制御が戻ってくる

事象通知まで待つことができます．事象の通知がされていれば，その通知を受けることができます．このように，事象の発生を待つ機構があるおかげで，プログラムを簡素化することができます．

以下に事象を通知する側と事象を受け取る側との関係を説明します．

▶ 事象を受け取る側が先に事象待ちになった場合

事象を受け取るタスクは，オブジェクトに対し事象待ち要求を発行し，待ち状態になります．事象を通知するタスクが事象の通知を行うと，受け取るタスクの待ち状態が解除されます(**図2.8**)．

▶ 事象通知が先に行われた場合

タスクによって先に事象を通知されている場合，事象を受け取るタスクがオブジェクトに対し事象待ち要求を発行しても待ち状態にならず，制御はそのまま戻ってきます(**図2.9**)．

オブジェクトによっては，事象だけでなく情報まで記憶してくれるオブジェクトもあります．そのオブジェクトの特性を利用し，タスク間，非タスク部とタスク間で連動して，システムとして処理をしていきます．

代表的なオブジェクトの特性を簡単に説明します．詳細は『解説編』を参照してください．

●イベント・フラグ

複数の事象の発生を同時に通知することができ，またそれを記憶することもできます．ただし，それぞれの事象の発生回数を記憶することはできません．

●セマフォ

一つの事象が発生した回数を記憶することができます．排他制御に用いることもできます．

●メールボックス，データ・キュー，メッセージ・バッファ

情報を通知することができ，また複数の情報を記憶することもできます．そして情報を受け取るタスクは，情報が通知されるまで待つことができます．

●メモリ・プール

メモリ領域を管理します．タスクで一時的にメモリ領域を利用したい場合やタスク間で情報をやりとりしたい場合に利用します．メモリ・プールに使用できるメモリ・ブロックがない場合，ほかのタスクからメモリ・ブロックが返却されるのを待つことができます．

● 資源の獲得

ここでの資源とは，各オブジェクトが管理しているものをいいます．資源としては，メモリ・プールのメモリ・ブロックやメールボックスのメッセージ，セマフォなどがあります．各オブジェクトから資源を獲得するためには，サービス・コールを発行します．資源を獲得する要求には，大体次の三つの方法が用意されています．

●ポーリング
●永久待ち
●タイム・アウト指定

ポーリングの獲得要求の場合，要求の時点で資源が獲得ができなければエラー(E_TMOUT)が返却されます．永久待ちの獲得要求の場合は，資源が獲得できるまで待ち状態になります．またタイム・アウト指定の獲得要求の場合は，指定時間経過しても資源が獲得できなければエラー(E_TMOUT)が返却されます．永久待ちもしくはタイム・アウト指定の獲得待ち状態になっている間に，ほかのタスクや非タスク部から資源の返却要求が発行されると，待ち状態が解除され資源を獲得する[注15]ことができます．そしてどの要求でもエラーが返却された場合は，資源が獲得できなかったことを示します．またサービス・コールに正常(E_OK)もしくは正数が返却された場合は，資源の獲得ができたことを示します．各オブジェクトが管理している資源を，次に記します．

- セマフォ　　：セマフォ
- メールボックス　　：メッセージ
- データ・キュー　　：メッセージもしくはメッセージ格納領域
- ミューテックス　　：ロック状態
- メッセージ・バッファ：メッセージもしくはメッセージ格納領域
- 固定長メモリ・プール：メモリ・ブロック
- 可変長メモリ・プール：メモリ・ブロック

2.4　基礎知識

ここでは，ITRON仕様に関する基礎知識について，筆者なりに解説します．

● オブジェクト

ITRON仕様では，カーネルが管理する対象をカーネル・オブジェクト(または，単にオブジェクト)といっています．オブジェクトには，タスク，イベント・フラグ，セマフォ，メモリ・プールなどがあります．オブジェクトごとにID番号があり，同じID番号の1番であってもタスクID番号の1番とイベント・フラグID番号の1番は当然ながら違うオブジェクトになります．ID番号は，サービス・コールのパラメータに指定します．サービス・コールの種別ごとに，指定するID番号の種別が異なります．たとえば，タスクを起動するサービス・コールにはタスクID番号を指定し，イベント・フラグをセットするサービス・コールにはイベント・フラグID番号を指定します．

● アクセス主体とアクセス対象

アクセス主体とは，プログラム自体のことをいい，タスクや非タスクコンテキスト[注16]をいいます．アクセス対象とは，アクセス主体がアクセスする対象であり，おもにサービス・コールの対象になるオブジェクト(タスクやイベント・フラグなど)を示します．μITRON4.0/PX仕様では，メモリ領域もアクセス対象としています．

● 自タスク

サービス・コールに対し，「自タスク」を指定できるものがあります．「自タスク」とは，サービス・コールを発行したタスクを指し，カーネルとしては実行状態のタスクになります．タスク間の共通関数で発行するサービス・コールで自タスクを指定すると，その共通関数を呼び出したタスクを指定することになります．非タスク部では，「自タスク」を指定することはできません．

● 優先度と優先順位

優先度とは，重要度の重みを示す値です．「高い優先度」とは，それより「低い優先度」の資源[注17]より優先的

注15：実際は，待ち優先順位や待ち資源の特性などにより，返却された資源を獲得するという表現ができない場合がある．
注16：μITRON3.0仕様では「タスク独立部」といっていた．本書では「非タスク部」と略する．

に処理させるために，資源に与える値です．「低い優先度」とは，それより「高い優先度」の資源がある場合，処理が後回しになってもよい資源に与える値です．ITRON仕様で優先度をもつ資源は，タスクとメッセージです．ITRON仕様における優先度は，数値で示され，数値が小さいほうが高い優先度となっています．

優先順位とは，処理される順番を示し，先に処理される資源が「高い優先順位」をもつといいます．処理が後回しになる資源は「低い優先順位」をもつことになります．たとえば，同じ優先度のタスクA，タスクB，タスクCがあり，レディ・キューにタスクC，タスクB，タスクAの順に並んでいた場合，タスクCは高い優先順位をもっていることになります．

● コンテキスト

コンテキストとは，プログラムの動作環境です．汎用OSでは，プログラムの動作環境の情報はかなりの量になります．そのため，プログラムの切り替え(コンテキストの切り替え)に時間がかかります．

一般のITRON仕様準拠カーネル上で動作するタスクのコンテキストは，CPUがもっているレジスタの内容になります．CPUのレジスタをメモリ上に保存したり，メモリ上にある情報をCPUのレジスタに復旧して，タスクの切り替えなどを行います．

● サービス・コール

タスクや各種ハンドラが行うカーネルへの要求をサービス・コールといいます．サービス・コールの処理はカーネルが行います．サービス・コールは，オブジェクトを対象とする場合もあるし，システムの制御に関する要求の場合もあります．サービス・コールにより，タスク間で事象の発生を通知したり，情報の通知を行ったりして，システムとしての処理を行っていきます．そしてプログラムを作成する人は，各サービス・コールの機能を理解してこれを有効に利用します．

サービス・コールにメモリ領域のアドレスを指定するものがあります．このアドレスに不正なアドレスを指定した場合，一般のカーネルではそのままアクセスしてしまい，システムが誤動作する可能性があります．たとえば,サービス・コールの返却値を格納するアドレスにほかのタスクのデータ領域のアドレスを指定してしまうと，そのアドレス先のメモリ領域に対し，カーネルはデータを書き込んでしまいます．その結果，対象になったタスクが誤動作する可能性が生じてしまいます．パラメータにアドレスを指定するようなサービス・コールについては，そのアドレスの指定内容に注意をはらいます．

● サービス・コールの返却値

サービス・コールの多くは処理の結果として返却値を返します.サービス・コールが正常に処理された場合は,通常は正常(E_OK)が返却されます．サービス・コールによっては，ID番号やサイズなどを返却するものもあります．そして負数が返却された場合はエラーを示します．

せっかくサービス・コールを発行しても，カーネルによってエラー返却されたら，要求したサービス・コールは処理されません．正常に処理されたことを確認するためにも，サービス・コールの返却値を必ず確認するようにしてください．もし，メモリ・ブロックを獲得するようなサービス・コールで返却値を確認しないと，エラーが返却されてメモリ・ブロックを獲得していないにも関わらず，メモリ・ブロックを示すポインタ先[注18]のメモリ領域をアクセスし，システムが誤動作する可能性があります．

サービス・コールのエラーの原因には，静的なものと動的なものの二つがあります．静的なものとは，サービス・コールのパラメータに指定するID番号や優先度などが不正である場合です．これは，システムの処理の状態に関わらず，ほぼいつでもエラーになります．

注17：ここでいう資源とは，オブジェクトだけでなくオブジェクトに関連する要素(メモリ・ブロックやメッセージなど)も含んだものをいう．

注18：エラーが発生した場合は，メモリ・ブロックのアドレスを示すポインタの内容は不定値になる．

動的なものというのは，サービス・コールの対象の状態によって，正常であったりエラーであったりするものです．たとえば，タスクを休止させる要求で，対象のタスクが休止状態以外である場合は正常ですが，すでに休止状態であればエラーになってしまいます．また，メモリ・ブロックの獲得において，メモリ・プールに使用できるメモリ・ブロックがあれば正常ですが，ない場合はエラーになります．対象の状態によってエラーになる可能性があるサービス・コールの場合も，返却値を必ず確認するようにします．

● ハンドラ

ITRON仕様では各種のハンドラを生成(もしくは定義)することができます．このハンドラは，割り込みが起因となり実行[注19]されます．ハンドラの処理は，タスクとは独立して実行されます(非タスク部)．各ハンドラの起動要因にしたがい処理を行い，その結果をタスクに通知するようにします．

たとえば，ハードウェアからの割り込みを割り込みハンドラで受け取り，割り込みハンドラでハードウェアから情報を取り出し，その情報をタスクに通知します．また，一定時間間隔で起動される周期ハンドラでは温度情報を取得し，温度が上がりすぎていたり，下がりすぎている場合に，温度調節をするような指示をタスクにすることもできます．

● 非タスク部のサービス・コール

非タスク部のサービス・コールが要球時に処理されるタイプか遅延処理されるタイプかは，カーネルにより異なります．

非タスク部のサービス・コールを遅延処理することの長所を以下に示します．

- 実際のシステム・コール処理を行わないため，要求したサービス・コール関数からすぐ戻ってくる．そのため，非タスク部の処理時間を短くすることができる．
- サービス・コール関数は，要求があったことを記憶するだけのため，関数の処理が少なくなる．

遅延処理することの短所を以下に示します．

- 要求に対する結果を求めるような要求ができない．たとえば，メモリ・プールからメモリ・ブロックを獲得する要求などがそれにあたる．
- 要求された情報を記憶するためにメモリを使用する．記憶するためのメモリがなくなるとメモリ不足エラーになり，サービス・コールの発行そのものができなくなる．メモリ不足エラーが発生した情報を残すことはできるが，エラーが発生したことをタスクに通知することもできなくなる．
- 要求した非タスク部では，サービス・コールの処理の結果を知ることができない．たとえば，タスクを強制待ち状態にする要求を発行して，そのタスクが休止状態である場合，サービス・コールの処理としてはエラーになる．しかし，要求そのものは正常で返却される．サービス・コールの処理の結果がエラーであったことはカーネルによって記録されることはあるが，要求した非タスク部は要求時にそのことを知ることはできない．

非タスク部のサービス・コールを要求時点で処理することの長所を以下に示します．

- 要求時点で処理を行うため，メモリ・ブロックの獲得のような結果を求める要求が提供される可能性がある．

要求時点で処理することの短所を以下に示します．

- サービス・コールの処理を要求時点で行うので，処理にかかる分だけ，非タスク部の処理時間が長くなる．

非タスク部のサービス・コールを遅延処理するタイプかどうかは，使用するITRON仕様準拠カーネルについての項目で調べておきましょう．

● ハードウェア制御

組み込みシステムは，ハードウェア制御を行います．一般のOSではデバイス・ドライバがハードウェア制御を行いますが，ITRON仕様ではデバイス・ドライバという独立した形式をとっていません．これは，デバイ

注19：周期ハンドラやアラーム・ハンドラなどはタイマの割り込みが起因して呼び出される．

ス・ドライバという独立した形式をとることによりカーネルの処理が増え，システム全体で行うハードウェア制御に負荷がかかるためです．

デバイス・ドライバは，ハードウェアを抽象化することで，タスクの処理を簡素化することを目的としています．ITRON仕様では，抽象化することによる処理時間を省くために，ハードウェアの抽象化を極力行っていません．そのため，ITRON仕様ではハードウェア制御に関してはほとんど規定がありません．しかし，組み込みシステムが複雑化してきていることと，プログラムの流用性を考慮し，デバイス・ドライバの構想をガイドラインとして策定する活動が社団法人トロン協会で行われています．

ITRON仕様におけるデバイス・ドライバの構想が導入されれば，一つのデバイスを制御するプログラムを複数のシステムで流用でき，システムの開発効率を向上させることができます．

● 起動要求と起床要求

起動要求と起床要求は，一般にタスクに対する要求のことをいいます．起動要求は，タスクとしての開始アドレスから実行をさせる要求で，タスクのいろいろな情報は初期状態になっています．起床要求は，タスクの待ち状態を解除する要求で，待ち状態になる前の状態が復旧されて処理を再開します．

● 遅延処理

遅延処理とは，要求した時点では実際の処理は行われず，後で処理が行われるもののことです．ITRON仕様において，遅延処理されるものは二つです．

一つ目は，ディスパッチ禁止(タスク切り替えの禁止)状態で，非タスク部が動作して，現在実行中のタスク(ディスパッチ禁止を要求したタスク)の状態を変化させる要求[注20]です．要求が行われても実行状態のタスクの状態は変化せず，実行を続けます．タスクがディスパッチ許可要求を行った時点で，さきほど要求されていた状態を変化させる要求の処理が行われます．つまり，ディスパッチ禁止状態で，タスクに対して強制待ち要求が行われてもタスクの実行は継続するわけです．しかし，ディスパッチ許可要求が行われた時点で強制待ち状態になります．

二つ目は，非タスク部のサービス・コール処理です．非タスク部で要求したサービス・コール処理は，要求時点では要求されたことを記憶するだけで処理を継続し，非タスク部の処理が終了した時点で，要求されていたサービス・コールの実際の処理を実行するというものです．

遅延処理してはいけないサービス・コールについては，仕様で規定されています．また，非タスク部が要求するサービス・コールの処理を遅延処理するか要求時に処理するかは，ITRON仕様準拠カーネルにより異なっています．

● 遅延ディスパッチ

ITRONの仕様書には，遅延ディスパッチについての説明があります．遅延ディスパッチとは，非タスク部からのサービス・コールによって，現在実行中のタスクより高い優先度のタスクを起床したような場合，要求時点ではディスパッチが起こらず，非タスク部の処理が終了してからディスパッチが行われることをいいます．優先順位は，高いほうから，

割り込み処理(非タスク部)　→　カーネル，スケジューラの処理　→　タスクの処理

となっているので，非タスク部が終了しなければ，タスクの処理は行われません．この遅延ディスパッチについては，それほど意識する必要はありません．

注20：強制待ちにする要求や休止状態にする要求をいう．