計算機アーキテクチャ 61－6
（1986. 3. 14）

マイクロプロセッサ用オペレーティングシステムインターフェイス MOSI（IEEE855）とITRONの国際標準化について

International Standard of Micrprocessor Operating System Interfaces MOSI(IEEE855) and ITRON

櫛木好明
Yoshiaki KUSHIKI
（松下電器産業株式会社　中央研究所）
Matsushita Electric Ind. Co. Ltd., Central Research Lab.

1．はじめに

マイクロプロセッサのオペレーティングシステム（以下ＯＳと略す）はすでに数多く存在し、マイクロプロセッサの普及につれてそのＯＳ上に開発した応用プログラムは増大している。これらの応用プログラムの各ＯＳ間での互換性を維持するためいくつかの試みがなされている。マイコン用ＯＳの互換性には大きく分類して次の２つのものがある。

１）ＯＳと応用プログラムのインターフェイスの互換性

あるＯＳ上で動作する応用プログラムの移植性を向上させるためには応用プログラムとＯＳのインターフェイス（システムコール）に互換性が必要である。

２）ＯＳそのものの互換性

これはＯＳのシステムコールのほかにスケジューリング方式、割り込み優先度決定方式、メモリ管理方式など、ハードウェアとも密接に関連するアーキテクチャを含む互換性である。これにより、応用プログラムの移植性に加えて、ＯＳそのものの異なるＣＰＵ上への移植性を向上させるのが目的である。

１）のこころみとしてよく知られているものにＵＮＩＸに類似のシステムコールで標準化案を検討してきた／ｕｓｒ／ｇｒｏｕｐの活動がある。この活動はその後正式の標準化活動として、現在ＩＥＥＥのＰ１００３ワーキンググループとして標準化案を出版するに至っている。Ｐ１００３における標準化案は、ＵＮＩＸＶ７、SystemIII、SystemVをベースにしてその他のＵＮＩＸ搭載システム上の応用プログラムの互換性を維持すべく検討された案である。したがってＵＮＩＸアプリケーションインターフェイス標準化案というべきものであり、リアルタイムＯＳとして必要な割り込み処理、メモリ管理などのハードウェア資源の管理に関するシステムコールは含んでいない。この小文ではリアルタイムＯＳについて述べるため、Ｐ１００３には言及しない。また、本文でとりあげるＭＯＳＩはこの１）に属する。[1]

２）については、複数のマイクロプロセッサ用に可能な限り統一性を持たせたリアルタイムモニターがすでにいくつか実用化されており、（米）Hunter & Ready社のＶＲＴＸや（米）Industrial Programming社のＭＴＯＳなどがよく知られている。

日本では複数のマイクロプロプロセッサに互換性を持たせたリアルタイムモニターがなかったため、１９８３年の日本電子工業振興協会マイクロコンピュータ委員会ＯＳ専門委員会においてリアルタイムモニターにたいする要求仕様が提案され、これに基づいて東京大学坂村健氏を中心にＩＴＲＯＮが開発された。したがって設計のプロセスには比較的多くのユーザの意見が取り込まれており、標準化案として議論するに足るものであると考えられる。[2][3]

2．ＭＯＳＩの概要

ＩＥＥＥ Computer Society Computer Standard Comittee Task Group ８５５はマイクロプロセッサ用ＯＳの標準インターフェイスに関する活動を行うべく１９８０年９月に結成された。このグループが決めるインターフェイスがＭＯＳＩ(Microprocessor Operating System Interface)である。

ＭＯＳＩの議長は当初インテルのＪａｃｋ　Ｃｏｗａｎが務め、その後モトローラのＤｏｎ　Ｊａｃｋｓｏｎが担当し、現在ではウェストバージニア大学のＪｉｍ　Ｍｏｏｎｅｙになっている。現在報告書としてＭＯＳＩ　Ｄｒａｆｔ６．２が１９８６年２月にＩＥＥＥから正式に交付されており、ＩＥＣに対して国際標準化規約として提案されている。[4][5][6]

日本ではこれを受けて日本電子工業振興協会を事務局としてＩＥＣ／ＳＣ４７Ｂワーキンググループにおいて東京大学森下厳教授を主査として１９８５年１０月よりＭＯＳＩの検討をはじめている。

ＭＯＳＩは応用プログラムの移植性を向上させる目的で設計された。したがって、既存のハードウェアやＯＳの上にＭＯＳＩをかぶせ、その上で動くアプリケーションを他のハードウェアやＭＯＳＩ上に移植することが容易にできるようになることをねらっている。

アプリケーションの記述は移植性を高めるために高級言語で書かれるものとし、Ｃ，Ｐａｓｃａｌ，ＦＯＲＴＲＡＮ，

Ａｄａ，ＣＯＢＯＬ，ＰＬ／ＭとＭＯＳＩとのインターフェイスを定めている。これらの高級言語とＭＯＳＩインターフェイスを用いて応用プログラムを書けば、ＣＰＵやＯＳアーキテクチャが異なってもその移植性を高める目的は達成できるだろう。

ＭＯＳＩの実現には、ＭＯＳＩと同じのシステムコールを有するＯＳを開発する方法と、既存のＯＳにＭＯＳＩの仮想インターフェイスをエミュレートする方法がある。前者の場合は、ＭＯＳＩがＯＳのアーキテクチャを厳密に規定したものではないだけに、異なるハードウェア上に構築したＭＯＳＩ間の互換性は保証されないだろう。また、後者の場合は、エミュレータのオーバヘッドのために性能の劣化をまねくこともあるだろう。

ＭＯＳＩはリアルタイムサポート、簡易開発サポート、複合開発サポートなどとサポートレベルで分類している。あるレベルのＭＯＳＩ仕様を用いて書かれた応用プログラムを同一レベルのＭＯＳＩインターフェイスを持つハードウェア上に移植するとき、さらに移植性は高まる。

しかし、ＣＰＵやＯＳアーキテクチャの違いからＭＯＳＩのインプリメンテーション時に生じる差異により、応用プログラムのある程度の書き直しはやむをえないだろう。

３．ＩＴＲＯＮとＭＯＳＩの基本思想の比較

ＩＴＲＯＮとＭＯＳＩの基本思想の比較をし、両者の違いについて述べる。

ＭＯＳＩがアプリケーションインターフェイスの互換性を重視するのが基本思想なら、ＩＴＲＯＮはリアルタイム処理の性能を重視するなかでマシン独立部を明確に分離することを基本思想としている。つまり、ＭＯＳＩではシステムコールに標準化のポイントをおき、システムコールをレベル分けすることにより、アプリケーションの移植時の共通性をたかめている。そのため、ＯＳのアーキテクチャはまったくインプリメント依存であり細かい制約は設けていない。

ＩＴＲＯＮでは性能を重視するようなＯＳアーキテクチャを採用し、ＯＳの移植性をたかめるため、マシン独立部とマシン依存部を分離し、マシン独立部の範囲内でかなり厳密にＯＳの仕様を規定している。

また、ＭＯＳＩでは複数の高級言語インタフェイスを用意しており、高級言語ごとにＭＯＳＩシステムコール仕様が異なる。これはアプリケーションの移植性のみを考えると問題はないが、ＯＳをシステムコールで理解するには混乱をひきおこす可能性がある。

一方、ＩＴＲＯＮではアプリケーションよりも、ＯＳ自身の移植性を重視するためシステムコールすなわちＯＳアーキテクチャを表現するものとして極力バリエーションを許さないようにしている。高級言語としても、比較的リアルタイム処理に向いているＣ言語のみを標準とし、他の高級言語については特に規定していない。これによって、ＩＴＲＯＮではＯＳ仕様の統一性と教育の一貫性が実現されている。

ＩＴＲＯＮは、ＴＲＯＮトータルアーキテクチャとして、ＴＲＯＮチップ核、ＩＴＲＯＮチップ核周辺、ＢＴＲＯＮ、ＭＴＲＯＮなどのなかで、その位置付けが明確にされており、将来の展開および拡張性に特筆すべき特長をもっている。

以上を要約すると、ＭＯＳＩがアプリケーションインターフェイス層を中心に「横」への標準化を進めたのに対して、ＩＴＲＯＮではＯＳのアーキテクチャからＴＲＯＮチップ核へ「縦」への仕様設計を進めている。さらに、そこからＢＴＲＯＮ、ＭＴＲＯＮなど「円」としてのひろがりをねらっている。[7][8][9]

以下、ＭＯＳＩとＩＴＲＯＮの各機能について比較と評価をおこなう。[10][11]

４．メモリ管理

メモリ管理はメモリの動的な獲得と解放をおこなうことをいうが、現在ＭＯＳＩでは単一のメモリ管理しか定義されておらず、多重のメモリ管理は将来のバージョンになるという。以下ＭＯＳＩのシステムコールは機能レベル１を●機能レベル２を◎で表す。

●ALLOCATE(size),{location},(err)	呼び出し元のプロセスが使用するメモリを指定した量だけ獲得する。
●GET_ALLOCATION_UNIT{unit_size},(err)	割り当て可能な最小単位のサイズをバイトで返す。
●GET_AMOUNT_REMAINING(largest),{total}(err)	未割り当ての最大の連続メモリサイズや未割り当ての総バイト数を返す。
●FREE(location),(err)	メモリの解放
●GET_SIZE(location),{size}(err)	割り当てられたメモリサイズをバイトで返す。

ＩＴＲＯＮでは、使用可能なメモリをメモリプールに分割した後、指定メモリプールからメモリブロックの割り当て管理をおこなう。割り当てられたメモリブロックは、全プロセスから使用可能である。以下ＩＴＲＯＮのチップ核システムコールを■印で、チップ核周辺のシステムコールを□印で示す。

■cre_mpl(option,p_mplid,mplid,pk_crmpl)　　メモリプールを生成する

optionは、メモリ管理を固定長とするか可変長とするかを指定する。また、プロセスがメモリ獲得の待ち行列に並ぶときの並び方として、優先度順かＦＩＦＯかを指定する。

pk_crmplは、メモリプール全体のサイズとブロックサイズデータへのポインタである。

■del_mpl(mplid)　　メモリプールを削除する

■get_blk(option,p_blk,mplid,blkcnt,p_tmout)　メモリブロックを獲得する

■rel_blk(blk,mplid)　　メモリブロックを解放する

■mpl_adr(pa_mpl,mplid)　　メモリプールアクセス番地を得る

ＩＴＲＯＮにはメモリプールの概念や、割り当て時のプロセス待ちや、可変長　のメモリ管理や、連続領域指定などＭＯＳＩに現在定められていないきめの細かいメモリ割り当て機能が多い。一方、ＭＯＳＩでは使用可能なメモリサイズなどを知るＧＥＴで始まるシステムコールが充実している。また、ＭＯＳＩでは多重メモリ管理を次ステップとしている。

５．時間管理とスケジューリング

ＭＯＳＩではユーザがプロセスにタイマーを結びつけ、ユーザがプロセスの待ちや起床のタイマー管理を記述する。

●DELAY(time_val),(time_unit),(err)　　指定時間の間プロセスの実行を遅延させる。

◎CONNECT_TIMER{timer_id},(err)　　特定のタイマーをプロセスに割り当てる。

◎DISCONNECT_TIMER(timer_id),(err)　　割り当てられていたタイマーを解放する。

◎INITIALIZE_TIMER(timer_id),(time_val),(time_unit),(timer_mode),(err)

タイマーに初期値とモードを割り当てる。

◎CONTROL_TIMER(timer_id),(timer_contl),(err)　タイマーの始動、停止、再セットをおこなう。

◎READ_TIMER(timer_id),{time_val},(time_unit),(err)

プロセスに割り当てられているタイマーの現在のタイマー値を読む。

◎GET_EXPIRED_TIMER{timer_id},(err)　　最も新しく例外を起こしたタイマーＩＤ番号を知る。

◎GET_TIME{time_val},(err)

◎GET_DATE{day_val},(err)　　現在時刻（TIME）と日付（DATE）を知る。

◎SLEEP_UNTIL(day_val),(time_val),(err)　　指定された日付と時刻までプロセスを遅延させる。

◎FORMAT_DATE/FORMAT_TIME/ENCODE_DATE/ENCODE_TIME

日付や時刻のフォーマッティングとそのエンコーディング。

一方、ＩＴＲＯＮでは時間管理は日付と時刻の設定と読み取りだけである。

■set_tim(pk_time)　　現在の日付と時刻を設定する。

■get_tim(pk_time)　　現在の日付と時刻を知る。

したがって、ＩＴＲＯＮではタイマーの操作によるプロセスの待ちや起床はプロセス管理に含まれている。このためタイマーによるプロセス管理の考え方を固定化していることになるが、タイマーの意味することはより明確になっている。ＭＯＳＩではタイマーの操作をユーザがおこなうことにより、より柔軟なプロセスの処理が記述できるがそれだけプログラムが複雑になる。また、従来のリアルタイムＯＳにおいても、例の少ないタイマーの取り扱いであるため、汎用的でない。

６．データ転送

ＭＯＳＩのデータ転送には、ファイルとして扱われるデバイス依存の同期Ｉ／Ｏと、Ｉ／Ｏ待ちを出さないデバイス独立の非同期Ｉ／Ｏがある。ファイル編成も、シーケンシャルファイル、直接編成ファイル、ディレクトリの構造、シーケンシャルストリーム、シーケンシャルレコード、直接編成レコードのアクセス方法、および固定長、可変長、ストリームのレコード編成を承認している。

◎CONNECT(fl_nm),{cnnct_id},(err)　　呼び出し元プロセスと使用するファイル名を結合する。

◎DISCONNECT(cnnct_id),(err)　　呼び出し元プロセスと使用中のファイル名の結合を解く。

◎CREATE(fl_nm),(fl_org),(rec_org),(rec_size),(txt_fl),(init_size),{cnnct_id},(err)

ファイルを生成する。ファイルがすでに存在するときは、CONNECTを、存在しないときはCREATEを使用するが、OPENは常におこなう必要がある。

◎EXTEND(cnnct_id),(extend_leng),(err)　　指定したファイルを、あるレコード数もしくはバイト数だけ拡張

する。

◎TRUNCATE(cnnct_id),(err)　　ファイルの現在位置から終わりまでを切り捨てる。

◎OPEN(cnnct_id),(accs_perm),(accs_method),(expected_rec_size),(txt_fl),{rec_size},(effnt_size),(err)
　　指定のファイルにたいする入出力を可能にする。

◎CLOSE(cnnct_id),(err)　　ファイルをクローズする。

◎READ(cnnct_id),(dat),(dat_leng),{amnt},(err)　　ファイルからデータを読み込み完了を待つ。

◎WRITE(cnnct_id),(dat),(dat_leng),{amnt},(err)　　ファイルにデータの書き込みをおこない完了を待つ。

◎SEEK(cnnct_id),(seek_org),(seek_forward),(seek_rec),{crrnt_position},(err)

指定されたレコード位置に現在位置ポインターを移動する。

◎GET_FILE_INFORMATION(cnnct_id),{fl_nm},(open_flg),(temp_fl_flg),(txt_fl),(rec_size),(rec_orgt),(fl_leng),(crrnt_position),(effnt_size),(accs_perm),(accs_method),(fl_orgt),(cret_time),(cret_date),(last_mod_time),(last_mod_date),(last_accs_time),(last_accs_date),(err)
　　結合しているファイルの情報を得る。

◎GET_ACCESS_CONTROL(cnct_id),(ident),{accs_contl},(err)
　　指定したファイルのアクセス権を調べる。

◎CHANGE_ACCESS_CONTROL(cnct_id),(ident),(accs_contl),(err)
　　指定したファイルのアクセス権を変更する。

◎DELETE(cnct_id),(err)　　結合されていたファイルを解除する。

◎RENAME(cnct_id),(new_nm),(err)　　ファイル名の変更。

◎GET_WORKING_DIRECTORY{dir_nm},(err)

◎CHANGE_WORKING_DIRECTORY(dir_nm),(err)

ワーキングディレクトリ（ディレクトリ指定のないファイルに使用されるディレクトリ）に対する操作。

◎GET_IDENTITY(identity_code),{identity},(err)　　ユーザあるいはユーザの属する識別コードを得る。

◎INITIATE_READ(cnct_id),(dat),(dat_leng),{asynchop_id},(err)

◎INITIATE_SEEK(cnct_id),(seek_origin),(seek_forward),(seek_rec),{asynchop_id},(err)

◎INITIATE_WRITE(cnct_id),(dat),(dat_leng),{asynchop_id},(err)

入出力処理の完了を待たないファイルのREAD,SEEK,WRITE。

◎TRANSFER_WAIT(asynchop_id),(wait_time),(amnt),{transfer_status},(err)

入出力処理の完了を待ち、終了情報を知らせるか、動作状態を知らせる。または非同期入出力処理を終了させる。

ＩＴＲＯＮでは入出力処理を、簡単なビット操作であるディスクリート入出力と、文字の入出力をおこなう周辺入出力とに分類する。ディスクリート入出力は周辺入出力チップを直接操作することを目的としたもので、入出力チップの複数の制御レジスタをグループ化し、それらの排他制御もおこなう。ＩＴＲＯＮではこのディスクリート入出力をＩＴＲＯＮ外核の機能としてサポートし、周辺入出力は、ユーザがドライバをつくるものとする立場をとっている。ＭＯＳＩではハードウェア制御に近いこのレベルのシステムコールは規定されていない。

□cre_dio(p_dioid,dioid,pk_dio)　　ディスクリート入出力を生成する

□del_dio(dio_id)　　ディスクリート入出力を削除する

□set_dio(option,p_prptn,dioid,offnum,mskptn,setptn)
　　ディスクリート出力をおこなう

□get_dio(option,p_crptn,dioid,offnum)　ディスクリート入力をおこなう

□dio_adr(pa_dio,dioid)　　ディスクリート入出力のアクセスアドレスを得る

ＭＯＳＩのファイルＩ／Ｏに関する規定はかなり充実したものである。一方、ＩＴＲＯＮではチップレベルのディスクリートＩ／Ｏしか規定していない。ＭＯＳＩではアプリケーションインターフェイスを中心に設計しており、ＩＴＲＯＮではリアルタイムカーネルインターフェイス中心に設計している思想の違いがよく現れている。

７．プロセス管理

オペレーティングシステムが最初に制御をわたすルートプロセスは、それが保有するデータ構造とともにＭＯＳＩプロセスとよばれる。ＭＯＳＩのプロセス管理の基本は、このＭＯＳＩプロセスを親プロセスとして子プロセスを生成し、子プロセスは自分および自分の生成した孫プロセスを、停止したり回復したり破壊する制御を行うところにある。つまり、

ＭＯＳＩプロセスはルートプロセスを頂点として子プロセスを階層的に管理し、動的に生成、破壊する。

通常、リアルタイムモニタでのプロセス生成破壊の管理は、他プロセスのみを対象とし自分自身を対象としない。また、管理構造を特定せず任意の管理構造をユーザが自由に規定しそのようにプログラムを記述すればよい。ＩＴＲＯＮを初めとしてＩＲＭＸ８６，ＲＭＳ６８Ｋなどはこれにあたる。

●INITIALIZE_PROGRAM{process_id},(err)

●TERMINATE_PROGRAM(process_id),(termination_status),(err)

ＭＯＳＩプロセスの宣言と初期化をINTIALIZE_PROGRAMでおこない、ＭＯＳＩプロセスの終了をTERMINATE_PROGRAMでおこなう。この宣言がなされると、以後ＭＯＳＩ関数やＭＯＳＩのパラメータブロック（ファイル、セマフォ、システムタグ、エクセプションハンドラなどに関するパラメータ領域で一般的にはＴＣＢに相当）が使用可能になる。

◎CREATE_PROCESS(program_name,active_flag,process_contl_block,priority,memory_allocation,leng_parameter_block,parameter_block),({process_id},err)

◎DESTROY_PROCESS(process_id),(err)

親プロセスが子プロセスの生成をCREATE_PROCESSでおこない、子プロセスまたは自分自身の破壊をDESTROY_PROCESSでおこなう。生成されたプロセスは、active_flagが真ならば直ちに実行可能状態になり、偽ならＷＡＩＴ状態となってRESUME_PROCESSがコールされるまで実行されない。

◎SUSPEND_PROCESS(process_id),(err)　自分が生成した子プロセス、または自分自身をSUSPEND(中止）状態にする。

◎RESUME_PROCESS(process_id),(err)　子プロセスのSUSPEND(中止）状態を解く。

◎GET_PROCESS_STATUS({process_state},process_time,termination_status,err)

プロセスの状態を知るために用いられ、process_stateにプロセスの状態が返される。

プロセスの状態には、実行(executing)、スケジュール対象(schedulued for execution)、中止(suspend)、イベント待ち(waiting for on event)、終了(termination)の５つの状態がある。

◎GET_PROCESS_INFO(),({process_id},priority,memory_allocation,leng_parameter_block,parameter_block,err)

生成された子プロセスは、親プロセスからパラメータブロックを受け取るためにこのシステムコールを呼ぶ必要がある。

◎CHANGE_PRIORITY(process_id),(new_priority),(err)

自分自身および他のプロセスの優先度を変更する。プロセス優先度はインプリメント時に規定されている。

ＩＴＲＯＮでは、タスク管理の機能を充実させ、アプリケーションの制御の種類に応じて使いわけて、ディスパッチングの速いリアルタイムシステムが記述できることをねらっている。タスクの状態は、実行状態、実行可能状態、待ち状態、強制待ち状態、休止状態、未登録状態の６つ（待ち状態を１つに数えると５つ）の状態がある。

■cre_tsk(p_tskid,tskid,pk_crtsk)タスクを生成する

■sta_tsk(tskid,initcode)　タスクを起動する

■del_tsk(tskid)　タスクを削除する

■def_ext(extrtn)　終了時処理ルーチンを定義する

■exi_tsk()　自タスクを正常終了する

■exd_tsk()　自タスクを正常終了後、削除する

■abo_tsk(abocode)　自タスクを異常終了させる

■ter_tsk(tskid,abocode)　他タスクを強制的に異常終了させる

■chg_tsk(tskid,tskpri)　タスク優先度を変更する

■rot_rdq(tskpri)　タスクのレディキューを回転する

■tcb_adr(pa_tsk,tskid)　タスクのＴＣＢアドレスを得る

■tsk_sts(pk_tskst,tskid)　タスクの状態を見る

■sus_tsk(tskid)　タスクを強制待ち状態へ移行する

■rsm_tsk(option,tskid)　強制待ち状態のタスクを再開する

■slp_tsk()　タスクを待ち状態へ移行する

■wai_tsk(p_tmout)　タスクを一定時間待ち状態へ移行する

■wup_tsk(tskid)　待ち状態のタスクを起床させる

■can_wup(p_wupcnt,tskid)　タスクの起床要求を無効にする

■cyc_wup(option,tskid,pk_cywup)　タスクを周期起床する

■can_cyc(tskid)　　タスクに出された周期起床要求を無効にする

ＩＴＲＯＮでは、ＭＯＳＩと異なりタスクのスケジューリング方式を規定している。すなわち、タスクに与えられた優先度を基にしたプライオリティベースのスケジューリング方式であり、同一優先度のタスクに対してはＦＣＦＳ(First Come First Service)方式をとる。また同一優先度タスクのレディキューを回転させるシステムコール(rot_rdq)があり、これによりラウンドロビン・スケジューリングがユーザ側で記述できる。

ＩＴＲＯＮでは、自タスクを操作するシステムコールと他タスクを操作するシステムコールを明確に分離している。さらに正常、異常終了および終了処理ルーチンの起動などのシステムコールがきめ細かく用意されており、ディスパッチング速度が速くなるよう工夫されている。

また、待ち、強制待ち、起床、周期起床などタスクを時間的に管理し制御するシステムコールが用意されている。これはＭＯＳＩにはない機能で、タイマー関連のシステムコールを用いてユーザが記述するようになっている。

以上のことから、ＩＴＲＯＮではリアルタイム制御に必要なタスク管理機能を十分に備え、効率のよいシステムが実現できることをねらっている。

８．プロセス同期と通信

ＭＯＳＩでは、プロセス間の同期をとるためにセマフォが、プロセス間の通信のためにメッセージが提供されている。

セマフォが生成されると、システムタグがつくられる。このシステムタグは親子プロセス間をパラメータブロックで渡されるか、あるいは複数プロセス間をメッセージを用いてやりとりされて、プロセス間を結びつける。

プロセス間のメッセージの通信は、メイルボックスを交換場所としておこなわれる。このメイルボックスが生成されたときに参照用のシステムタグが返され、セマフォと同じ方法で複数のプロセス間が結びつけられる。

●CREATE_SEMAPHORE(init_val),{smph_id},(err)　　セマフォの生成

●SIGNAL_SEMAPHORE(smph_id),(err)　　セマフォ待ちのプロセスがないときは、カウンタを１増す。待ちプロセスがある時には、セマフォ待ちの解除をおこない、カウンタは変更しない。

●WAIT_SEMAPHORE(smph_id),(time_lim),{time_out_flg},(err)　セマフォのカウンタが０ならば、このシステムコールを呼んだプロセスをSUSPENDする。

●DELETE_SEMAPHORE(smph_id),(uncond_del),{in_use_flg},(err)セマフォの削除

◎MULTIPLE_SIGNAL_SEMAPHORE(smph_id),(sig_count),(err)　　カウンタをsig_count分だけ増やす。セマフォ待ちのプロセスがあれば、待ちを解除しカウンタを減らす。

◎MULTIPLE_WAIT_SEMAPHORE(smph_id),(wait_count),(time_lim),{time_out_flg},(err)

カウンタがwait_countを満たしていないときはプロセスをsuspendする。他はWAIT_SEMAPHOREに同じ。

◎CREATE_REPOSITORY{reposit_id},(err)　　メッセージを保持する領域（メイルボックス）をつくる。

◎SEND_MESSAGE(reposit_id),(msg_leng),(msg_blk),(rspns_id),{msg_id},(err)

メッセージをキューにつなぐ。

◎RECEIVE_MESSAGE(reposit_id),(time_lim),(leng_msg_blk),(msg_blk),(msg_leng),(rspns_id),{ret_status},(err)

指定された領域からメッセージブロックにメッセージを入れる。

◎DELETE_REPOSITORY(reposit_id),(uncond_del),{ret_status},(err)　指定された領域を削除する。

◎RESPOND_MESSAGE(rspns_leng),(rspns_blk),(msg_id),(err)　　受け取ったメッセージに対して応答する。

◎WAIT_RESPONSE(msg_id),(time_lim),(leng_rspns_blk),(rspns_blk),(rspns_leng),{ret_status),(err)

送信したメッセージに対する応答を待つ。

ＩＴＲＯＮでは、プロセス間の待ち合わせなどのためにイベントの発生を検知するイベントフラグと、プロセス間の同期や相互排除をおこなうためのセマフォと、プロセス間のメッセージ通信をおこなうメイルボックスを提供している。

イベントフラグは１ワードを単位として生成、削除、設定、待ちをおこない、待つプロセスのキューイングはおこなわない。

セマフェは、計数型セマフォであり、カウンタ値が指定できるため幅広く使用できる。

メイルボックスは、プロセスと独立に設定でき、受信プロセスにメッセージの先頭アドレスが渡される。

■cre_flg(p_flgid,flgid)　　イベントフラグを生成する

■del_flg(flgid)　　イベントフラグを削除する

■set_flg(option,p_prptn,flgid,setptn)　イベントフラグをセットする

■wai_flg(option,p_prptn,flgid,mskptn,p_tmout)
イベントフラグの条件満足を待つ
■flg_adr(pa_flg,flgid) イベントフラグアクセスアドレスを得る
■cre_semo(ption,p_semid,semid,initcnt) セマフォを生成する
■del_sem(semid) セマフォを削除する
■sig_sem(semid,cnt) セマフォに対する信号操作（V命令）
■wai_sem(option,semid,cnt,p_tmout) セマフォに対する待ち操作（P命令）
■sem_adr(pa_sem,semid) セマフォアドレスを得る
■cre_mbx(option,p_mbxid,mbxid) メイルボックスを生成する
■del_mbx(mbxid) メイルホックスを削除する
■snd_msg(mbxid,p_msg) メッセージを送信する
■rcv_msg(option,pp_msg,mbxid,p_tmout) メッセージを受信する
■mbx_adr(pa_mbx,mbxid) メイルボックスアクセスアドレスを得る

ＭＯＳＩ、ＩＴＲＯＮともかなり充実した機能を持っている。

ＭＯＳＩとＩＴＲＯＮでの違いは、ＭＯＳＩにイベントフラグがサポートされていないことである。ＩＴＲＯＮのイベントフラグは１ワードあり、これにより事象の解析が高速にできる。ＭＯＳＩでは事象の解析ををメッセージ通信でおこなうことになるが、この場合はオーバーヘッドが大きくなるだろう。

９．例外処理

ＭＯＳＩの例外は３つのクラス（手続き例外、オペレータ例外、ハードウェア割り込み）に分類されている。各クラスごとに、例外ハンドラの動作、非動作、変更の設定ができる。

●DISABLE(excep_class),(err) ある例外クラスの例外ハンドラを非動作にする。
●ENABLE(excep_class),(err) 例外ハンドラの動作する例外クラスを設定する。
●GET_ENABLE_STATUS(excep_class),{status},(err) 例外ハンドラの状態を返す。
◎GET_EXCEPTION_HANDLER{crrnt_handler},(err) 現在の例外ハンドラの識別子を得る。
◎SET_EXCEPTION_HANDLER(new_handler),(err) 新しい例外ハンドラに置きかえる。
◎RESET_EXCEPTION_HANDLER システムのディフォルトハンドラを現在の例外ハンドラとして再セットする。
◎RAISE_EXCEPTION(excep_class),(sub_class),(err) 例外ハンドラに例外を通知する。
◎GET_EXCEPTION_CODE{class_code},(sub_class),(err)
例外ハンドラから呼ばれ、例外クラスとサブクラスを知らせる。
◎EXIT_FROM_HANDLER(dispos),(err) 例外ハンドラが、例外を起こしたプロセスへ制御を戻す。

ＩＴＲＯＮの例外は５つ（システムコール例外、ハードウェア例外、各プロセス固有の例外、非プロセス部分の例外、全プロセス共通の例外）に分類されている。それぞれの例外に対して例外処理ハンドラを定義することができる。

■def_exc(option,exchdr,excds) 例外ハンドラの定義
■ret_exc() 例外ハンドラからの復帰

ＭＯＳＩでは例外クラスごとに例外ハンドラの動作、非動作、変更の設定ができ、ＩＴＲＯＮでは、むしろ割り込み処理にその機能を持たせている。システムのエラーが比較的よくおこるような場合（デバッグ時など）にこの機能は役立つだろう。

１０．割り込み処理

ＭＯＳＩには割り込みに関する規定がない。リアルタイム制御には必須と考えられるが、あえてこれを含めていない。割り込みはハードウェア制御の問題であり、したがってインプリメント依存であるとしているものと考えられる。

一方、ＩＴＲＯＮでは、割り込みはリアルタイム制御には最も重要な手段であるとし、割り込みの定義と、割り込み処理ハンドラ中でのシステムコール使用時のシステムの動作がマシン独立に規定される。つまり、割り込みハンドラ中のシステムコール実行ではディスパッチは起こらず、ret_intにより割り込み処理ハンドラを出るまで、ディスパッチが遅延される。こうすることにより、割り込み時のオーバーヘッドを最小にすることができる。

■def_int(intvec,inthdr) 割り込みハンドラを定義する

■ret_int()　　割り込みハンドラから復帰する
■set_int(intmsk)　(ITRON/68K) 割り込みマスクをセットする
■dis_int(intdvn)　(ITRON/86) 割り込みを禁止する
■ena_int(intdvn)　(ITRON/86) 割り込みを許可する
■fet_dat(intdvn)　(ITRON/86) 割り込みハンドラ用DSをセット
■get_dvn(p_intdvn)　(ITRON/86) 割り込みを発生したデバイス番号を知る

１１．むすび

ＭＯＳＩもＩＴＲＯＮもシステムコールのサブセット分類を試みている。ＭＯＳＩでは、リアルタイムサポート、単純開発サポート、複合開発サポートなどと、主として適用範囲のレベルで分類している。一方、ＩＴＲＯＮでは、チップ核、チップ核周辺、外核、ユーティリティと、ＣＰＵからアプリケーションまでの層別に分類している。両者のシステムコールの設計思想は出発点が異なっており、ＭＯＳＩがアプリケーションインターフェイス標準案なら、ＩＴＲＯＮはリアルタイムカーネル標準案である。

したがって、Ｐ１００３やＭＯＳＩのそれぞれ性質のことなるＯＳ規格の下部層もしくは横並びにＩＴＲＯＮが位置づけられる。つまりＰ１００３やＭＯＳＩをＩＴＲＯＮをベースに実現する。お互いに整合がとれていることが望ましいが、すべてが完全な包含関係にある必要はない。またお互いのシステムコールが重複せずに完全に補間しあっている必要もない。なぜならそれぞれの規格は異なる目的のもとに設計されたものであるから。ただ、インプリメントする側から見ると、ＭＯＳＩの方に改善の余地を多く残している。

マイコン用ＯＳの標準化のアプローチとしては、両者とも基本思想は明解であり、いずれもこの分野の互換性というニーズ動向に適合したものといえる。したがって、国際標準として先行提案のＭＯＳＩのみを採用するのではなく、ＩＴＲＯＮもあわせて採用し、「横」型ユーザにも「縦」型ユーザにも便宜を供する巾のある標準規格であってほしいと願うものである。

参考文献

1) P1003 Working Group:Portable Operating System Environment P1003/D5,IEEE,October,1985
2) 日本電子工業振興協会:マイクロコンピュータ用リアルタイム組み込みＯＳ用語集、59-A-213,59年3月
3) 日本電子工業振興協会:マイクロコンピュータに関する調査報告書－リアルタイム組み込みＯＳ（ＴＲＯＮ）仕様の概要と将来動向、60-A-221,60年3月
4) Don L.Jackson and Jack Cowan:The Proposed IEEE 855 Microprocessor Operating Systems Interface Standard, IEEE Micro,August,pp63-71 (1984)
5) Robert G.Stewart et al.:MicroStandards,IEEE Micro June ,pp80-82(1985)
6) IEEE Trial-Use Standard Specification for Microprocessor Operating Systems Interfaces,IEEE Computer Society,December,1985
7) 坂村健:ＴＲＯＮプロジェクトの設計思想,bit,1984.9
8) 坂村健:「マイクロプロセッサ用標準リアルタイムオペレーティングシステム原案：ＴＲＯＮ 1.アーキテクチャ」情報処理学会第28会(昭和59年前期)全国大会
9) 坂村健:「I-ＴＲＯＮアーキテクチャ」情報処理学会第29会(昭和59年後期)全国大会
10)坂村健:ＩＴＲＯＮ概説書,1985
11)坂村健:ＩＴＲＯＮシステムコール仕様書,1985