付録

コンカレントDOS
CDOSについて
――16ビット時代のOS――

本稿では8086/8088用の標準OSであるCP/M-86，およびその系列下にあるOSについて簡単に説明し，その後にディジタル・リサーチ社の主力製品であるコンカレントDOSについて詳しく説明をします．

額田　忠之

はじめに

1970年代の末にインテル社の16ビットCPU，8086が開発されました．当時は8080A，8085，6800，6502，ならびにZ80などで代表される8ビットCPUの全盛期であり，その名残りは現在にまでおよんでいます．

8086が開発された当時は，メモリの価格が現在とは比較にならないくらい高価であったため，パーソナル・コンピュータの世界は，相変らず8ビットCPUが支配していました．しかしながら，パーソナル・コンピュータの分野においても，8ビットCPUのパフォーマンス，およびメモリ容量では不十分である，という認識が芽生えてきていました．

ところが，1980年代に入ってメモリの低価格化が進むにしたがい，パーソナル・コンピュータにおいても16ビットCPUを採用した機種が出現しました．そしてこれらの機種のうち，8086/8088を使用したものに対する標準OSとして，ディジタル・リサーチ社が発表したのが，CP/M-86です．

当時，8ビットの8080，8085，Z80系のパーソナル・コンピュータのOSとしてはCP/M-80が標準であったので，16ビットの世界でもCP/M-86が支配的な位置を占めるであろう，と誰もが思っていたのですが，米国IBMがそのパーソナル・コンピュータ，IBM-PC（8088を採用）において，PC-DOS（MS-DOSと等価）を採用したことにより情勢が一変し，少なくとも米国ではCP/M-86に代ってMS-DOSが圧倒的なシェアを持っています．

もちろん，IBM-PCにおいてもCP/M-86はサポートされていましたが，IBM社はPC-DOSが40ドル（約1万円）であるのに対して，CP/Mには200ドル（約5万円）の価格を設定したため，米国におけるMS-DOS対CP/Mの勝負は火を見るよりも明かであったのです．しかしながら，日本においては情勢は少し異なり，CP/M-86とMS-DOSは共存し，ほぼ互角のシェアを持っています．

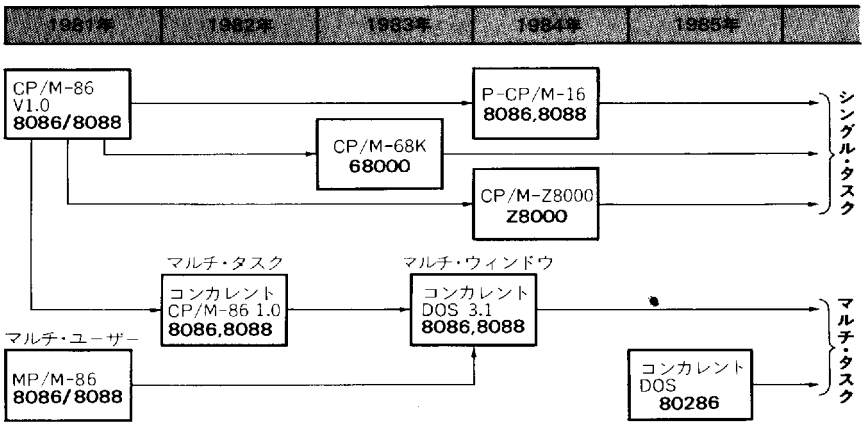
16ビット用CP/Mの系列

図1にディジタル・リサーチ社の16ビット用OSの系列を示します．

■シングル・タスクの系列

ディジタル・リサーチ社の16ビットCPU用OSは，1981年に発表されたCP/M-86 V1.0（V：バージョン）に始まります．このCP/M-86 V1.0は当時8ビットCPU用の標準OSであったCP/M-80 V2.2を，16ビットCPUである8086/8088用に改造したものです．

このCP/M-86 V1.0はCP/M-80 V2.2と同様に，



〈図1〉
ディジタル・リサーチ社の16ビット用OSの系列

シングル・ユーザー／シングル・タスクのOSです．このCP/M-86 V1.0は，1984年始めにバージョン・アップがなされ，名前もP-CP/M-16(Personal CP/M-16)と変更されました．このP-CP/M-16には，CP/M-80がバージョン2.2からバージョン3.0 (CP/M-Plus)にアップデートされたのと同様の改良が施されています．

P-CP/M-16はどちらかといえば，比較的下位の16ビット機種をターゲットとしており，PC-DOSモードが用意されており，かつタイマによる時分割処理により，三つのバック・グラウンド・ジョブが走るようになっています．

PC-DOSモードというのは，IBM-PC用に作られた，PC-DOSの制御下で動作するアプリケーション・プログラムを，P-CP/M-16上でも走行可とする機能です．ただし，現在までのところ，PC-DOS V1.1 (MS-DOS V1.25)用に作られたアプリケーション・プログラムしかサポートされておらず，PC-DOS V2.0 (MS-DOS V2.0)以降のバージョンに対してはサポートされていません．

またCP/M-86 V1.0は8086/8088以外の16ビットCPUに対してもサポートされるようになり，モトローラ68000用にはCP/M-68K (68キロ，つまり68000)が1983年に発表され，ザイログ社Z8000用にはCP/M-Z8000が1984年に発表されています．

CP/M-68Kについては日立が開発に協力したこともあって，その全貌が明かになっていますが，CP/M-Z8000については内容はほとんど知られていません．両者ともCP/M-86用に作られた，膨大なソフトウェア遺産を継承することを目的としています．

しかしながらBASICやC言語などの高級言語で書かれたプログラムであれば，インタプリタやコンパイラを開発することにより対処できますが，アセンブラで書かれたプログラムのコンバージョンには相当の労力と費用がかかるはずです．

■シングル・ユーザー／マルチ・タスクの系列

ディジタル・リサーチ社のシングル・ユーザー／マルチ・タスクのOSは，1982年に発表されたコンカレントCP/M-86 V1.0に始まります．コンカレント(Concurrent)ということばは，“同時の”とか“併発の”という意味があり，コンピュータ用語では“同時並行処理ができる”といった形容詞として使われています．

コンカレントCP/M-86 (以下C-CP/M-86) V1.0では，時分割により同時並行的に動くことのできるプログラムの単位をプロセス(Process)と呼びます．そしてC-CP/M-86下では，複数(数10個)のプロセスが同時に動くことができます．

また，C-CP/M-86では，複数個(通常は4個程度)の仮想コンソールがあり，ある一時点にはそれらの中の一つが実コンソール（スクリーン・バッファ）上に表示されています．表示すべき仮想コンソールの選択は，キーボード操作 (ctrl-0～3) によって行いますが，XIOSコールによりプロセスから行うこともできます．

さらにこのC-CP/M-86では，マルチ・タスク（マルチ・プロセス）制御ができるため，以下のような機能が追加されています．

- プロセス間通信および同期制御（キュー）
- プロセス間の排他制御（キュー）
- ファイル・ロック／共有機能
- パスワードによるファイルの保護

C-CP/M V1.0は，1984年にバージョン3.1にアップデートされ，名前もコンカレントDOS(以下CDOS)と改められました．しかしながら始めのうちは，コンカレントCP/M(C-CP/M) 3.1と呼ばれていたので，この名前のほうが通りが良いのですが，ここではあえて新しい呼び名，コンカレントDOSを使うことにします．

CDOS 3.1は基本的にはC-CP/M 1.0に対してマルチ・ウィンドウ・コントロールとPC-DOSモードを

付加し，MP/M-86 の流れをくむマルチ・ユーザー機能を組み入れたものです．また，この CDOS 3.1 には GSX によるグラフィック・インターフェース，FSX による外国語入力（英語以外の言語）のサポート，およびローカル・エリヤ・ネットワーク（LAN）機能である DR-NET のサポートも追加されています．

CDOS 3.1 のマルチ・ウィンドウ機能は複数の仮想コンソールの内容を一つの実スクリーン上に切り出す方式をとっており，ウィンドウの形態はタイリング (tiling)，およびオーバラップの双方が可能です．タイリングというのは，画面を風呂場のタイルのような形でいくつかに分けるやりかたで，オーバラップというのはウィンドウ同士が重なって表示され，重なった部分は上側のウィンドウが表示されるやりかたです．

CDOS 3.1 におけるマルチウィンドウは，キャラクタ・スクリーンのみを対象としており，ビット・マップ・タイプのスクリーンのマルチウィンドウ化は除外されています．これは CDOS 3.1 が8086/8088用のものであり，ビット・マップ・タイプのスクリーンに対する仮想コンソールを複数個持つことは，メモリ容量的に無理があるからです．

■マルチ・ユーザー／マルチ・タスクの系列

ディジタル・リサーチ社のマルチ・ユーザー／マルチ・タスク用 OS は，1981年に発表された MP/M-86 です．この MP/M-86 は必要なメモリ・サイズが大きすぎ，かつパフォーマンス的にも難があったため，あまり良い評判は得られませんでした．この MP/M-86 のマルチ・ユーザー／マルチ・タスク機能は1984年のコンカレント DOS 3.1 に統合され，現在に至っています．

またディジタル・リサーチ社は，1985年上半期に80286用のコンカレント DOS をリリースする予定でいます．これはマルチ・ユーザー／マルチ・タスクの OS であり，かつビット・マップ・タイプのマルチウィンドウ機能が備えられたものになるようです．

ごぞんじのように現在の 8086/8088 マシンにはメモリ・マネージメントの機能がないにもかかわらず，マルチ・タスクの機能を持たせていますが，メモリ保護（タスク同士が互いに相手のメモリを壊し合わないようにする）や機密保持の立場からすると，大いに難があります．

したがって，メモリ・マネージメント機能を持った 80286 用のコンカレント DOS になって始めて，本格的なマルチ・ユーザー／マルチ・タスクの OS であるということができるでしょう．

コンカレント DOS 3.1

コンカレント DOS（以下 CDOS）3.1 は，マルチ・ユーザー／マルチ・タスク，またはシングル・ユーザー／マルチ・タスクの OS ですが，ここではシングル・ユーザー／マルチ・タスクの機能に絞って説明をします．

CDOS 3.1 は 8086/8088 を対象にしており，これらの CPU を用いた機種はメモリ・マネージメント機能を持っていることはほとんどなく，パフォーマンス的に見ても，マルチ・ユーザー・システムとして効率よく動作できるほどの性能はありませんので，CDOS 3.1 がマルチ・ユーザー・システムとして使われることはあまりないと思われます．

さて，CDOS 3.1 はマルチ・タスク機能以外に，複数の仮想コンソール（Virtual Console）があり，それらの内容をウィンドウとして実画面に切り出す機能を持っています．そのためウィンドウごとに別のアプリケーション・プログラムを走らせたり，ウィンドウ間での情報のやりとりなども可能です．

たとえば，あるウィンドウでエディタを動かしながら，他のウィンドウではアセンブル，またはコンパイルをやらせ，さらに他のウィンドウではリストのプリント・アウトを行わせるようなことができます．

さらに CDOS 3.1 ではシステム・エクステンション (System Extention) という概念で，システムの拡張ができ，日本語（英語以外の言語）処理に対しては FSX (Foreign System Extention)，グラフ機能については GSX (Graphic System Extention) という形でサポートされています．また，ローカル・エリヤ・ネットワークである DR-NET もやはりシステム・エクステンションの形で実現されています．

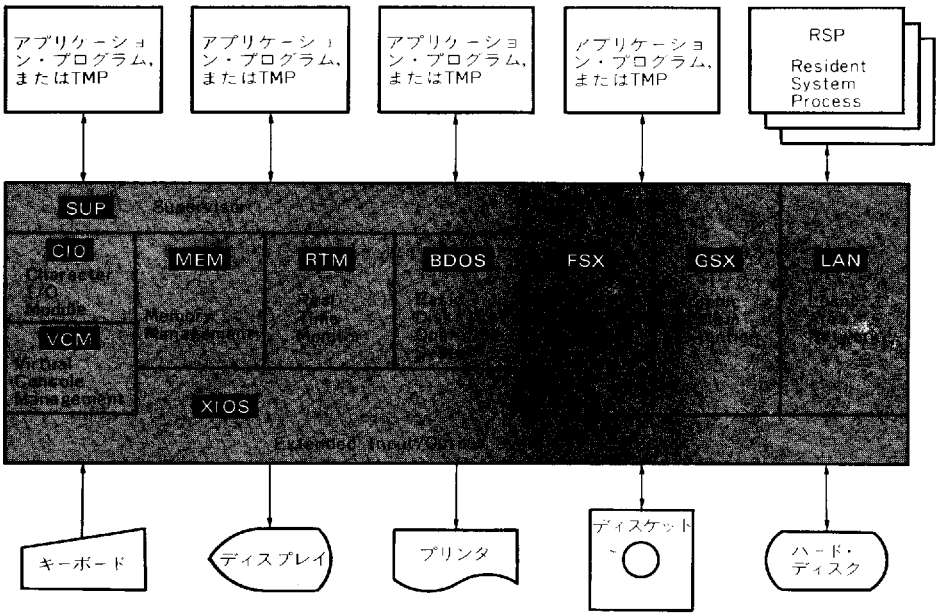
CDOS 3.1 は複数の仮想コンソールを持つことができると述べましたが，シングル・ユーザー・システムとしては，仮想コンソールは 4 個の場合がほとんどです．したがって，以後の説明ではすべて四つの仮想コンソールの場合を想定して話を進めていきます．

コンカレント DOS 3.1 の構成

図 2 に CDOS 3.1 の構成を示します．以下に各構成要素の説明をします．

■SUP（Supervisor；スーパバイザ）

SUP はアプリケーション・プログラム，またはシステム・プロセスなどが実行するシステム・コール (INT 224) の受付け窓口であり，システム・コールが出されるとレジスタ CL（システム・コール番号）の値にしたがって各部所，つまり CIO, MEM, RTM, または BDOS



〈図2〉
コンカレントDOS 3.1の構成

に対してコントロールを移します．

■MEM (Memory Management；メモリ管理)

MEMはCDOS内のメモリ管理機構であり，固定パーティション型のメモリ管理を行います．パーティションの大きさは，GENCCPMによるシステム・ジェネレーションにより自由に設定できますが，通常は8Kバイト～32Kバイトの範囲に設定します．このMEMはメモリ管理に関連したシステム・コール，M_ALLOC，M_FREE，MC_ABSALLOC，MC_ABSMAX，MC_ALLFREE，MC_ALLOC，MC_FREE，MC_MAXの処理を行います．

■RTM (Real Time Monitor；リアル・タイム・モニタ)

このRTMはCDOS 3.1のマルチ・タスク機能を司どり，プロセスのディスパッチ（プロセスにコントロールを配分すること），キューの管理，フラグの管理，デバイスのポーリング，システム・タイミング作業（P_Delayの管理）などを行います．

なお，CDOS 3.1ではシステム内で時分割的に並行動作可能なプログラムの最小単位をプロセスと呼びます．このプロセスという用語はマルチ・タスクという用語におけるタスクと同義語です．

■BDOS (Basic Disk Operating System)

BDOSはCDOS 3.1における，ファイル管理システムです．ディスケットやディスクに対するファイル操作は，すべてこのBDOSの助けを借りて行われます．このBDOSの機能はCP/M-86のそれに対して上方互換性を持ち，マルチ・タスク環境に対処するため次のような機能追加が行われています．

- ファイル・ロッキング
- ファイルの共有使用
- タイム・スタンプ機能
- パスワードによるアクセス保護
- エラー処理機能の追加

■CIO (Character I/O Module)

CIOはコンソール（キーボード／ディスプレイ）に対する入出力，ならびにリスト・デバイス（プリンタ）への出力操作を管理します．したがってコンソール関係のシステム・コール(C_XXX)，およびリスト・デバイス関係のシステム・コール（L_XXX）は，すべてこのCIOにより処理されます．

■RSP (Resident System Process)

RSPはレジデント・システム・プロセスの略であり，日本語にすると常駐システム・プロセスとなります．このRSPはCDOSの管理に必要な機能を実行するプロセスで，CDOSがロードされるときに一緒にロードされ，常にメモリ内に存在し，普段はコントロールを放棄して眠っていますが，必要に応じて目をさまし，自分に与えられた仕事を始め，それが終ると再度眠りにつきます．CDOS 3.1中に存在するRSPは次の6個です．

- **ABORT**　ABORTコマンド，およびP_ABORT

システム・コールを実行します．

- **CLOCK**　XIOS のタイマ割り込みルーチンにより1秒ごとに起こされ，時刻の更新とステータス・ラインの表示の更新を行います．
- **DIR (Directory)**　ディレクトリの表示を行うプロセスです．
- **PIN (Physical Input Process)**　PIN はキーボードから入力されたデータを，XIOS コール (io_conin) により読み取り，それをキュー (VINQ0～3) に詰め込む作業をするプロセスです．この PIN はキーボード入力データのエコー・バック，および各種 ctrl ファンクション(ctrl-C，ctrl-P，…)の処理も行います．
- **TMP (Terminal Message Process)**　TMP はターミナル・メッセージ・プロセスの略で，その機能は CP/M-86 における CCP のそれに相当します．TMP は仮想コンソールごとに存在し，システムが起動されたとき，および当該仮想コンソールを使用するアプリケーション・プログラムが存在しないときには常にこの TMP が姿を現わし，ログイン・ドライブ名とコマンド・プロンプト（A＞など）を表示し，オペレータからのコマンド入力を待ちます．

　コマンド入力があると，P_CLI システム・コールにより，指定されたコマンドを呼び出し，自分は眠ってしまいます．そしてアプリケーション・プログラムが正常に終了したとき，およびアボートされたときには，再びこの TMP が動き出します．

　TMP は仮想コンソールごとにあると述べましたが，コード・シェア (Code Share) というテクニックを使って，メモリ上には一つしかないが，プロセスとしては仮想コンソールの数だけ走れるようになっています．

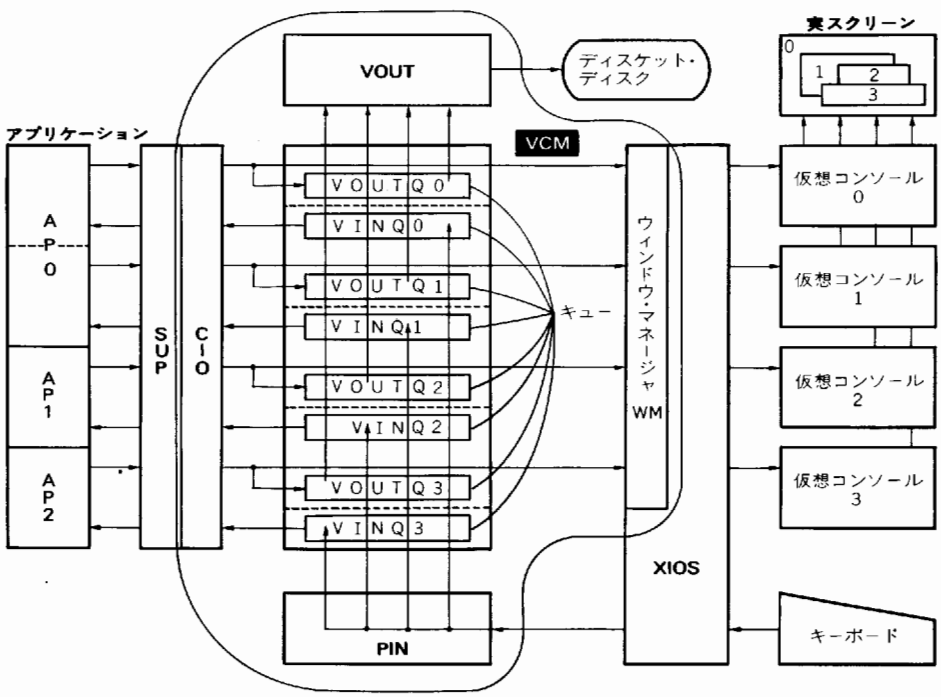
- **VOUT (Virtual Output Process)**　これはコンソールがバック・グラウンド・バッファード・モードで動作しているときに，当該コンソールに対するコンソール出力データをディスケット，またはディスクにいったん蓄積し(スプーリング)，コンソールがフォア・グラウンドに変わったときに PIN と連携して，スプールされたデータをコンソールに出力するためのプロセスです．

■XIOS (Extended I/O System)

　これは CP/M-86 における BIOS に相当する部分であり，CDOS の他の部分とハードウェアとのインターフェースをとる役割をします．

　CDOS 3.1 で追加されたウィンドウ・マネージメント(WM)の機能も，この XIOS 内にあります．CDOS を自分の設計したマシンに移植する場合，この XIOS をハードウェアに合わせて作ることにより，そのマシン上で CDOS を動かすことができます．

〈図3〉
仮想コンソール・マネージャ(VCM)の構成



■VCM (Virtual Console Manager)

VCM はさきに説明した CIO, PIN, VOUT, ならびに XIOS 中の WM により構成され，キーボード入力データをアプリケーション・プログラムに渡すルート，およびアプリケーション・プログラムから出力されたデータを仮想コンソールに書き込み，各仮想コンソールの状態を実スクリーン上にウィンドウとして切り出すパスを形成します．**図3** に VCM の構成とデータの流れを示します．

キーボードから入力されたデータが，アプリケーション・プログラムに渡される過程を説明します．PIN は常に XIOS に対してキーボード入力命令(io_conin)を出しており，キーボードからデータが入力されると，XIOS からそのデータをもらい，その時点にアクティブなコンソールの VINQ に書き込みます．

アプリケーション・プログラムが，システム・コールにより SUP を介してキーボード・データを読む命令を出すと，CIO は VINQ の内容をデキュー(Deque)して，データをアプリケーション・プログラムに渡します．この時点に VINQ の内容が空である場合には，PIN により VINQ にデータが書き込まれるのを待ちます．

またアプリケーション・プログラムがシステム・コールによりコンソールに対してデータを出力すると，CIO は XIOS の WM をコールして，そのデータを仮想コンソールに書き込み，WM はその時点のウィンドウの形態にしたがって，各仮想コンソールの内容を実スクリーン上に切り出します．

図3 では三つのアプリケーション・プログラム(AP 0～2)が走っていることを想定し，AP0 は仮想コンソール0および1を使用していることになっています．このように一つのアプリケーションが複数の仮想コンソールを使うことも，CDOS では可能です．ただし，かなり高度のテクニックを要することは確かです．

■FSX (Foreign System Extention)

これは英語圏以外の言語を使用する地域のためのシステム・エクステンションです．われわれ日本人にとっては，日本語入力は必須であり，かな漢字変換，JIS コード，または区点コード入力なども，この FSX によりサポートされます．

■GSX (Graphic System Extention)

これは CDOS 3.1 におけるグラフィック機能をシステム・コール・レベルでサポートするための，システム・エクステンションです．

■LAN (Local Area Network)

CDOS 3.1 では，ローカル・エリヤ・ネットワーク(LAN)の機能をシステム・エクステンションとしてサポートします．ディジタル・リサーチ社は，この LAN のことを DR-NET という商標で呼んでいます．

この DR-NET を用いると，CDOS 3.1 で動くシステム同士を，ローカル・ネットワークで結ぶことができ，かつファイル・サーバ，プリント・サーバにより，システム資源の共有化を図ることができます．

〈図4〉
コンカレント DOS 3.1 のメモリ配置とサイズ (IBM-PC/XT の場合)

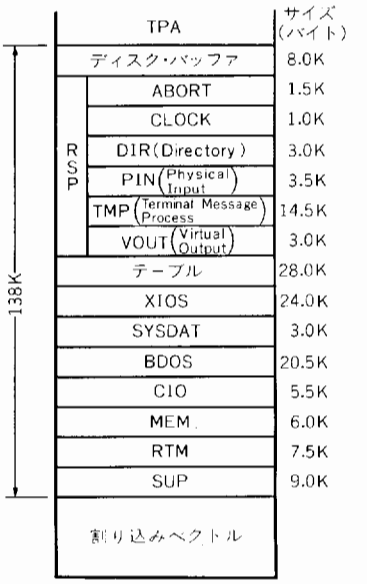


図4 に CDOS 3.1 のメモリ配置と各モジュールのサイズを示します．各モジュールのサイズは，IBM-PC/XT の PC-DOS モード付きの場合です．この図からわかるように，FSX, GSX, それに LAN 機能なしで，CDOS のサイズは 138K バイトとなっています．したがって，これらの機能を加えた場合には，OS のサイズは優に 200K バイトを超えてしまいます．

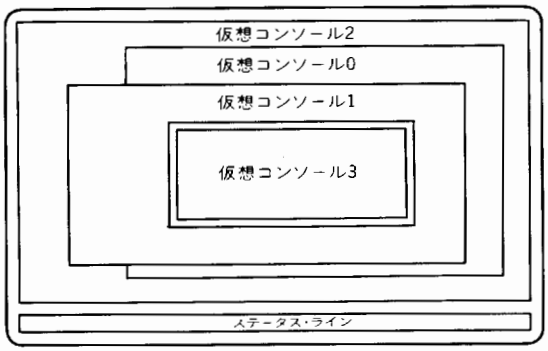
また，CDOS のサイズは XIOS のサイズにもろに影響を受け，とくに漢字表示を用いる日本では XIOS のサイズは図4の 24K バイトくらいではとても収まり切れません．したがって，国内で発売または販売されている，CDOS を採用したシステムのほとんどが，最小メモリ容量として 512K バイトを設定しています．

筆者が一昔前に使用していた中型コンピュータのメモリ・サイズが 131K バイトであったにもかかわらず，5～6本のジョブが同時に走っていたことを思うと，これを進歩というべきなのかどうか，わからなくなってしまいます．

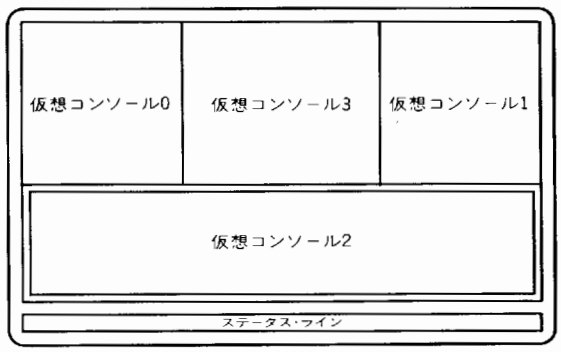
CDOS 3.1 におけるウィンドウ

ここでは CDOS 3.1 におけるマルチ・ウィンドウについて説明をします．CDOS 3.1 における実スクリー

〈図5〉 オーバラップ型のウィンドウ



〈図6〉 タイリング型のウィンドウ



ン・サイズは，任意に設定できるのですが，説明を簡単にするため，横80文字(ANK：80，漢字：40)，縦25行のスクリーンを想定して話を進めます．

■ウィンドウの形態

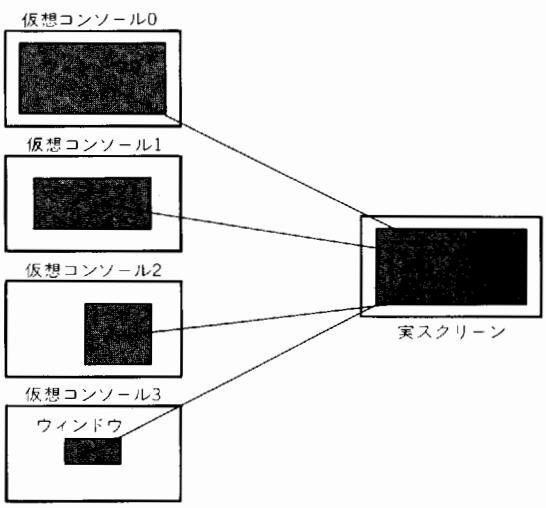
CDOS 3.1 におけるマルチ・ウィンドウの形態は2種類あり，その一つはオーバラップ型のウィンドウであり，他はタイリング型のウィンドウです．**図5**にオーバラップ型のウィンドウの例を示し，**図6**にタイリング型のウィンドウの例を示します．

オーバラップ型のウィンドウでは，各仮想コンソールの一部が重なり合って表示されます．そして重なり合った部分は，表側にある仮想コンソール部分が表示され，裏側のウィンドウは見えなくなります．**図5**では一番裏側のウィンドウは仮想コンソール2であり，一番表側にあるウィンドウは仮想コンソール3です．各ウィンドウの境界は罫線により仕切られます．

図6のタイリング型のウィンドウでは，各ウィンドウは互いに重なり合うことをせず，実スクリーンを4個の矩形部分に分けてウィンドウが表示されます．いずれの型式のウィンドウにおいても，その時点にキーボードを所有しているコンソール（アクティブ・コンソールという）のウィンドウは2重の罫線によって囲まれます．

図7に各仮想コンソールからのウィンドウの切り出しの概念図を示します．各仮想コンソールの一部（全部でもよい）がウィンドウとして設定され，それらのウィンドウはその時点のウィンドウの優先順位にしたがって，下から上へと表示されます．

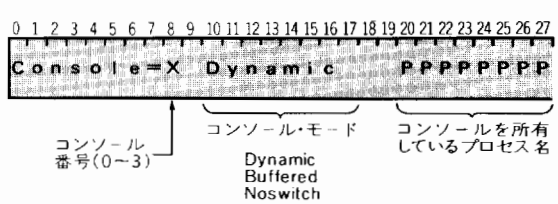
〈図7〉 ウィンドウの切り出し



■ステータス・ライン

スクリーンの最下行（通常25行目）には，ステータス・ラインが表示されます．**図8**に IBM-PC/XT におけるステータス・ラインを示します．このステータス・ラインは，システム全体の状態と各仮想コンソールの状態が示されています．以下にステータス・ラインの各カラムの意味を説明します．

- **0～8（コンソール番号）**　その時点にキーボードを所有しているコンソール（アクティブ・コンソール）番号を表示します．アクティブ・コンソールが切り替えられると，この値も変化します．アクティブ・コンソールの切り替えは，コントロール・シフトのテンキー0～3を入力することにより行われます．ctrl-0 を押すとアクティブ・コンソールは0になり，ctrl-3 を押すと，アクティブ・コンソールは



3になります．また，ctrl DELキーを押すと，アクティブ・コンソールはフルスクリーンになります．

- **10〜17(コンソール・モード)**　これはコンソールのモードを示します．このエリヤに3種類のモード(Dynamic, Buffered, Noswitch) のいずれかが表示されます．Dynamicというのは，一番ふつうのモードであり，コンソールがバック・グラウンドになっているときにも，そのコンソールに出力されるデータはどんどんコンソール上に出力され，ロール・アップにより前のほうに出力されたデータは失われていきます．

ところがバッファード・モードにしておくと，そのコンソールがバック・グラウンドにいるときに出力されるデータは，いったんディスクまたはディスケットに蓄積され，コンソールがアクティブになったときに，蓄積されたデータがコンソール上に出力されます．つまり，コンソールが裏に隠れて，オペレータの目に見えないような場合に，出力されるデータを失いたくない場合に，このバッファード・モードを用います．

コンソールをバッファード・モードにセットするには，VCMODEというコマンドを用います．コンソール・モードのデフォールト状態はダイナミックです．

また，Noswitchという状態は，アクティブ・コンソールの切り替えができない状態をいいます．アプリケーション・プログラムの都合で，どうしてもコンソールを切り替えて欲しくないようなときに，Noswitchとします．

- **20〜27(プロセス名)**　このエリヤには，現在そのコンソールが割り当てられているプロセスの名前が入ります．したがって，アプリケーション・プログラムが動いていない場合には“Tmp”(ターミナル・メッセージ・プロセス) と表示されています．
- **29〜34(オープン・ベクター)**　このエリヤにはその時点にファイルがオープンされているドライブ名がセットされます．SYSDAT (System Data Area) のオープン・ベクターのうち，“1”が立っているドライブ名 (A, B, C, D, M, *) が表示されます．このエリヤには6個のベクターしか表示できませんので，ドライブE以上に対してはアスタリスク(*)が表示されます．なお，Mはメモリ・ディスクのことです．
- **36, 37(ctrl-S, ctrl-O)**　当該コンソールがctrl-S入力により，コンソール出力を停止しているときには，^Sと表示され，ctrl-Oにより出力が無視されているときには，^Oと表示されます．
- **39〜42(ctrl-Pプリンタ番号)**　仮想コンソールへの出力データをプリンタに打ち出す機能です．ctrl-Pを押すと，^P=という文字と，そのコンソールに割り当てられているプリンタ番号が表示されます．
- **44〜52(プリンタ番号)**　そのコンソールに割り当てられているプリンタ番号が表示されます．
- **55〜57(Wmenu)**　ウィンドウ・マネージャ(Wmenu)がロードされると，Winと表示されます．
- **59〜62(Capsロック)**　キーボードがCaps (大文字) ロックになっていることを示します．
- **64〜66(Numロック)**　テンキーがナンバー(数字)ロック状態であることを示します．
- **69〜76(時刻)**　hh：mm：ssの形で時刻を表示します．
- **78, 79(am/pm)**　午前 (am)，または午後 (pm) の表示を行います．

CDOS 3.1の日本語版では，このステータス・ラインの一部がカナ文字になっています．

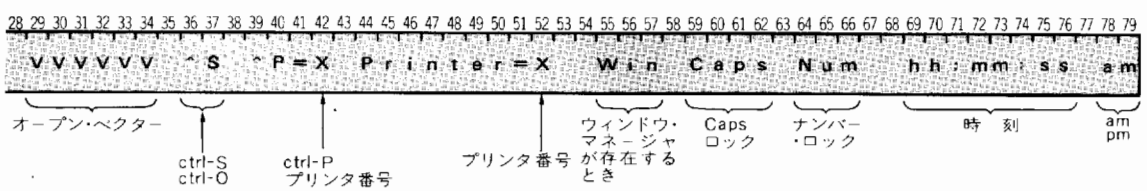
■ウィンドウの設定

ウィンドウを設定するには，WINDOWというコマンドを用います．このWINDOWコマンドに対するコマンド列をサブミット・ファイルとして登録しておけば，このファイルを実行させることにより，いつでも同じウィンドウを作ることができます．

このWINDOWコマンドには五つのサブ・コマンドがあります．

```
d>WINDOW  VIEW
d>WINDOW  TOP
d>WINDOW  FULL
d>WINDOW  WRITE
d>WINDOW  CHANGE
```

〈図8〉 IBM-PC/XTのステータス・ライン



これらのサブ・コマンドのうち，VIEWは現在のウィンドウの状態を調べるためのコマンドであり，WRITEはファイルの内容をウィンドウに書き込むことを目的としています．したがって，この二つのサブ・コマンドはウィンドウの設定とは直接関係がないので，ここでは説明を除外します．

・CHANGE（ウィンドウの変更）

これはウィンドウを設定するサブ・コマンドです．例として次のコマンドを取り上げます．

```
d>window change n=0 prow=8, pcol=2,
   nrows=16, ncols=65, tracking=no,
   display=color, fgcolor=red, bgcolor=white CR
```

n=0	：コンソール番号（n=0～3）
prow	：Physical Row（実スクリーンの行）
pcol	：Physical Column（実スクリーンのカラム）
nrows	：ウィンドウの縦方向のサイズ（行）
ncols	：ウィンドウの横方向のサイズ（カラム）
tracking	：カーソル・トラッキング
display	：ディスプレイの種類（color/B & W）
fgcolor	：Foreground Color（フォアグラウンド・カラー）
bgcolor	：Background Color（バックグラウンド・カラー）

このコマンドは，コンソール0に対して，実スクリーンの行8カラム2を左上隅とし，16行65カラムの大きさのウィンドを開くことを命じます．またこのウィンドウのバックグラウンドは白色で，フォアグラウンドは赤色です．したがって白の背景色の上に赤色の文字が表示されることになります．

trackingというパラメータのオプションは，no(ノー)とrow(行)が許され，ウィンドウをスクローリングしたときに，カーソル位置が常にウィンドウ内にあるようにするか(row tracking)，それともカーソル位置とは無関係にスクローリングするか（no tracking）の指定を行います．

WINDOWコマンドの次の例は，CARDDECK.SUBという名でよく知られたウィンドウ設定用のサブミット・ファイルです．

```
window    change  n=0  prow=8, pcol=2,
                  nrows=16, ncols=65
window    change  n=1  prow=6, pcol=6,
                  nrows=16, ncols=65
window    change  n=2  prow=4, pcol=10,
                  nrows=16, ncols=65
window    change  n=3  prow=2, pcol=15,
                  nrows=16, ncols=65
```

このCARDDECK.SUBは四つのコンソールに対して同じ大きさのウィンドウを設定します．各ウィンドウのprowとpcolが少しずつ異なっていますので，四つのウィンドウはトランプのカードを少しずつずらして重ねたように表示されます．図9にCARDDECK.SUBにより開かれるウィンドウを示します．

・TOP（トップ・ウィンドウの指定）

このサブ・コマンドは指定したウィンドウを一番上(トップ)にもってき，かつそのウィンドウを持つコンソールをアクティブにします．したがって，キーボードからコントロール・シフトの0～3をキーインした場合と同じ働きをします．コマンドの形式は次のとおりです．

```
d>window  top  n=3
```

このコマンドにより，ウィンドウ3が最上部になり，かつコンソール3がアクティブ・コンソールになります．

・FULL（フル・スクリーン）

このサブ・コマンドは，指定したウィンドウをフル・スクリーンにします．したがって，このコマンドの機能は，キーボードからコントロール・シフトのDELキーを押したのと同じ働きをします．コマンドのフォーマットは次のとおりです．

```
d>window  full  n=2
```

このコマンドは，ウィンドウ2をフル・スクリーンにします．

この他ウィンドウ設定用のコマンドとして，WMENU(ウィンドウ・メニュー)があります．このWMENUはコマンド，

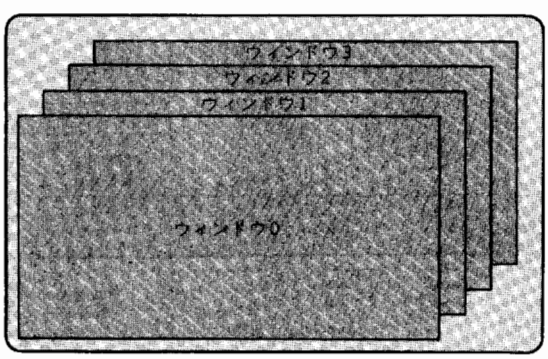
```
d>WMENU CR
```

により，メモリ上にロードされます．WMENUはロードされると，

```
Window Manager Installed
```

というメッセージを表示して，Dev_Waitflagというシステム・コールを出して眠ってしまいます．WMENUがロードされていることは，ステータス・ライン上の

〈図9〉 CARDDECK.SUBにより作られるウィンドウ



カラム55～57に，“Win”という文字が表示されるのでわかります．

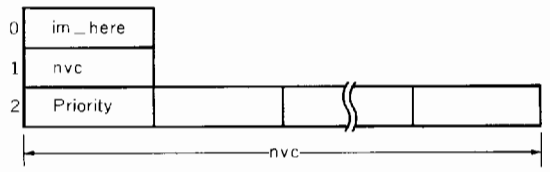
眠っているWMENUは，コントロール・シフトにてテンキーの＋(プラス)を押すことにより起動されます．このキーが押されると，XIOSのio_conin ルーチンは，Dev_Setflagというシステム・コールを出して，眠っているWMENUを起こします．

WMENUは，コンソールをNoswitch状態にして動作します．WINDOWコマンドでは，ウィンドウを設定するのにいくつものパラメータを用いましたが，このWMENUではメニュー形式によりパラメータを選択することができ，かつ矢印キー（↑↓→←）を使って，ウィンドウを画面上に作りながら，ウィンドウのサイズや位置を決めることができます．

WMENUはオペレータに指示されたことがらを，さきに述べたWINDOWコマンドに変換して，ウィンドウを設定します．

WMENUが動いているときは，キーボード入力はすべてWMENUに渡り，アクティブ・コンソールのプロセスには渡りません．WMENUが動くと，通常の

〈図10〉 ウィンドウ・データ・ブロック



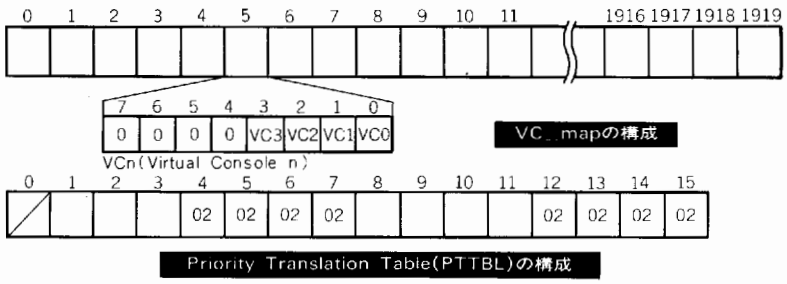
・im_here：ウィンドウ・マネージャ(WMENU)の状態
　00H：マネージャは存在しない
　01H：マネージャーは常駐しているがアクティブではない
　02H：マネージャが存在し，アクティブである
・nvc　：number of virtual console
　仮想コンソールの数
・Priority：ウインドウ表示の優先度
　下にある仮想コンソール番号から上にある仮想コンソールへと順に並んでいる

	0	1	2	3
00H	vs_Cursor		vs_top_left	
04H	vs_bot_right		vs_old_t_l	
08H	vs_old_b_r		vs_crt_size	
0CH	vs_win_size		vs_view_point	
10H	vs_rows		vs_cols	
14H	vs_correct		vs_vc_seg	
18H	vs_crt_seg		vs_list_ptr	
1CH	vs_attrib	vs_mode	vs_cur_track	vs_width
20H	vs_number	vs_bit	vs_save_cursor	
24H	vs_vector		vs_xlat	
28H	vs_qpb		vs_true_view	
2CH	vs_cur_type			
30H				
34H	vs_mx			
38H				
3CH				

ラベル名	変位(16進数)	長さ(バイト)	説明
vs_cursor	00	2	仮想コンソール内のカーソルの行（Row）と桁（Col）
vs_top_left	02	2	直前のウィンドウの左上隅位置
vs_bot_right	04	2	直前のウィンドウの右下隅位置
vs_old_t_l	06	2	ウィンドウの左上隅位置のセーブ・エリヤ
vs_old_b_r	08	2	ウィンドウの右下隅位置のセーブ・エリヤ
vs_crt_size	0A	2	仮想コンソールの大きさ（行と桁）マイナス1
vs_win_size	0C	2	WMENUがウィンドウの大きさを憶えておくのに用いる
vs_view_point	0E	2	ウィンドウの左上隅位置
vs_row	10	2	その時点のウィンドウの行数
vs_cols	12	2	その時点のウィンドウの桁数
vs_correct	14	2	ウィンドウ・トラッキング補正係数
vs_vc_seg	16	2	仮想コンソール・バッファのセグメント・アドレス
vs_crt_seg	18	2	実スクリーン・メモリのセグメント・アドレス
vs_list_ptr	1A	2	開始行更新リストのポインタ
vs_attrib	1C	1	カレント・アトリビュート
vs_mode	1D	1	ビット0；ラップ(wrap) ON/OFF　ビット1：カーソルON/OFF
vs_cur_track	1E	1	固定ウィンドウ／トラッキング・スクロール
vs_width	1F	1	ラップ・アラウンドすべきカラム
vs_number	20	1	仮想コンソール番号
vs_bit	21	1	ビット位置による仮想コンソール番号
vs_save_cursor	22	2	ESCファンクションのセーブ／リストア・カーソル用のエリヤ
vs_vector	24	2	IO_conout ルーチン用ステート・ベクター
vs_xlat	26	2	モノクローム／カラー優先度変換テーブル・アドレス
vs_qpb	28	2	予　備
vs_true_view	2A	2	補正後のウィンドウ位置
vs_cur_type	2C	2	モノクローム／カラー
vs_mx	34	1	仮想コンソールの排他制御用セマフォ

←↑　〈図11〉
仮想コンソール・データ・ストラクチャ

〈図12〉
ウィンドウ切り出しのためのテーブル



キーボード・データの入力パスである XIOS の io__conin にはデータは渡されず，ww__key のほうへデータが行くからです．

WMENU は XIOS コール，ww__key により，キーボード・データを読み取ります．XIOS のキーボード割り込みルーチンは，WMENUが生きているか否かをチェックし，データを io__conin に渡すべきか，ww__key に渡すべきかの判断をしています．

■ウィンドウ制御用テーブル

XIOSのウィンドウ管理部(WM)にはウィンドウ制御用に二つのテーブルがあります．その一つはウィンドウ・データ・ブロックであり，他は仮想コンソール・データ・スラトクチャ (Data Structure) です．前者はシステム内にただ一つだけあり，後者は仮想コンソールごとに一つずつあります．

図10にウィンドウ・データ・ブロックの構成を示し，**図11**に仮想コンソール・データ・ストラクチャの構成を示します．

■ウィンドウの切り出し制御

XIOS 中のウィンドウ・マネージャは，仮想コンソールを実スクリーン上に切り出す制御を行うために，二つのテーブルを管理しています．

その一つは vc__map と呼ばれ，実スクリーンと同じ大きさを持つテーブルです．他の一つは Priority Translation Table (PTTBL) と呼ばれ，2^n (n＝仮想コンソールの数) バイトの長さを持つテーブルであり，仮想コンソールの数が4である場合には16バイトの長さを持ちます．

図12に vc__map と PTTBL の構成を示します．この図では実スクリーンの大きさは1920 (80×24) バイトであり，仮想コンソールの数は4であると仮定しています．

vc__map は各ビットが実スクリーンの各バイトと対応しており，仮想コンソールの実スクリーン上での重なり具合を示します．このテーブルの下位4ビットはビット0～3が仮想コンソール0～3に対応し，これらのビットは仮想コンソールの切り出し部分に相当するときに“1”となります．

PTTBL はどの仮想コンソールを，最上部に表示すべきかを決めるためのテーブルです．このテーブルは，最上部に表示すべき仮想コンソールの vc__map にセットされるビットが“1”である，エントリにその仮想コンソール番号がセットされます．

図12の例では仮想コンソール2 (vc__map のビット2)がトップですので，PTTBL のエントリ，4，5，6，7，および12，13，14，15にコンソール番号02がセットされています (これらのエントリ番号はすべてビット2が“1”である)．

XIOS のウィンドウ・マネージャはウィンドウを切り出すときに，vc__map の内容を PTTBL により変換し，変換した値がトップにすべきコンソール番号に一致するときには，そのコンソールのデータを実スクリーンに書き込みます．

システム・コマンド

図13に CDOS 3.1 におけるシステム・コマンドの一覧表を示します．システム・コマンドはすべて，XXXXX.CMD というコマンド・ファイルで，コマンド・プロンプトの後に XXXXX とパラメータを入力することにより実行されます．

図13に示したシステム・コマンドは，IBM-PC/XT の場合であり，システムによってはこれより多くのシステム・コマンドがあります．これらのコマンドのうち，注欄にアスタリスク (＊) を付けたものは，OEM サプライド・コマンド (OEM Supplied Command) と呼ばれており，CDOS をディジタル・リサーチ社から買ってきて，自分のシステムに乗せる際に勝手に作るべきコマンドです．

したがって，これらのコマンドのスペックは各社各様で，極端な場合にはコマンド名も異なります．

コマンド名	機　　能	注
ABORT	指定した仮想コンソール上のプロセスの実行を中断する	
CHSET	コマンド・ファイル（XXX.CMD）のコマンド・ヘッダを変更する	
CONFIG	システム構成情報をセットする	*
DATE	日付および時刻の表示，および設定をする	
DDT-86	8086/8088用プログラム・デバッガ	
DIR	ディレクトリ（SYS ファイル以外）の表示を行う	
DSKMAINT	ディスケットのフォーマッティング，コピー，ベリファイを行う	*
ED	ソース・ファイル作成用のライン・エディタ	
ERA	ディスケット，またはディスク上のファイルをイレーズする	
ERAQ	ERA と機能は同じだが，イレーズする前にオペレータに確認する	
FUNCTION	ファンクション・キーに対して，ストリング・データを設定する	*
HDMAINT	ハード・ディスクのフォーマッティング，コピー，ベリファイを行う	*
HELP	コマンド使用法の解説をする	
INITDIR	CDOS にてタイム・スタンプを使えるように，ディレクトリをイニシャライズする	
PIP	デバイス間のファイル・コピーを行う	
PRINT	ファイルをプリントする	
PRINTER	仮想コンソールに対してプリンタを割り当てる	
REN	ファイルの名前を変更する	
SDIR	全ファイルのディレクトリの表示をする	
SET	ファイル・アトリビュートの設定／変更をする	
SHOW	システム資源の状態を表示する	
SUBMIT	ファイルとして記録されたコマンド列を実行する	
SYSDSK	システム・ドライブを設定する	
TYPE	テキスト・ファイルをコンソール上に表示する	
USER	ユーザー番号の表示／設定をする	
VCMODE	仮想コンソールの状態（ダイナミック／バッファード）を設定する	
WINDOW	ウィンドウを設定する	*
WMENU	メニュー形式でかつ会話的にウィンドウを設定する	*

〈図13〉
CDOS 3.1のシステム・コマンド

（注）　＊を付したコマンドはOEMサプライド・コマンドで，CDOS 標準コマンドではない．したがって，これらのコマンドのスペックはシステムごとに異なる

システム・コール

CDOS 3.1では103種類のシステム・コールをサポートしています．図14にこれらのシステム・コールの一覧表を示します．CDOS 3.1におけるシステム・コールは次の11種類のジャンルに分けられます．

①コンソール・システム・コール（C_XXX）
②デバイス・システム・コール（DEV　XXX）
③ディスク・ドライブ・システム・コール（DRV_XXX）
④ファイル・アクセス・システム・コール（F_XXX）
⑤リスト・デバイス・システム・コール（L_XXX）
⑥MP/M-86メモリ管理システム・コール（M_XXX）
⑦CP/M-86メモリ管理システム・コール（MC_XXX）
⑧プロセス・システム・コール（P_XXX）
⑨キュー管理システム・コール（Q_XXX）
⑩システム・コール（S_XXX）
⑪タイム・システム・コール（T_XXX）

■マルチ・タスク環境をサポートするシステム・コール

CDOS 3.1のシステム・コールのうち，とくにマルチ・タスク環境を実現するためのシステム・コールについて説明をします．マルチ・タスク環境をサポートするために必要な機能は，

・プロセスの生成
・プロセス間通信および同期
・プロセス間の排他制御
・プロセス遊休状態の処置

となります．

第1のプロセスの生成に関しては，P_CREATEというシステム・コールが用意されています．またプロセス間の通信および同期制御はキュー（Queue）とフラグにより行います．このキューというのは，プロセス同士がコミュニケーションをするためのメール・ボックス（郵便箱）です．キューの制御に関しては，Q_MAKE, Q_OPEN, Q_WRITE ならびに Q_READというシステム・コールが設けられています．

プロセス間の同期制御については，フラグによるコントロールも可能です．このフラグというのは，あるプロセスがフラグがセットされるのを待ち，フラグがセットされていないときには，そのプロセスは休眠状態に入り，フラグがセットされると，待っていたプロセスは再度走り出します．このフラグによるコントロ

〈図14〉 CDOS 3.1 のシステム・コール

システム・コール番号（10進数）	システム・コール名	機　能
149	C_ASSIGN	デフォールト・コンソールを他のプロセスに割り当てる
146	C_ATTACH	デフォールト・コンソールの所有権を確立する．コンソールが他のプロセスに割り当てられているときは待たされる
162	C_CATTACH	条件付きのC_ATTACHであり，コンソールが他のプロセスに割り当てられているときには，エラー・コードが返る
110	C_DELIMIT	C_WRITESTRで用いる，ストリング・デリミタを設定する
147	C_DETTACH	デフォールト・コンソールの使用を放棄する
153	C_GET	割り当てられている仮想コンソール番号を得る
109	C_MODE	コンソール・モードのセット／ゲット
6	C_RAWIO	デフォールト・コンソールに対するRAWIO
1	C_READ	デフォールト・コンソールからデータを読み取る
10	C_READSTR	デフォールト・コンソールからエディットずみのストリング・データを読み取る
148	C_SET	デフォールト・コンソール番号をセットする
11	C_STAT	デフォールト・コンソールのステータスを得る
2	C_WRITE	デフォールト・コンソールに対してデータを出力する
111	C_WRITEBLK	デフォールト・コンソールに対してブロック・データを出力する
9	C_WRITESTR	ストリング・データをデフォールト・コンソールに対して出力する
133	DEV_SETFLAG	システム・フラグをセットする
132	DEV_WAITFLAG	システム・フラグがセットされるのを待つ
131	DEV_POLL	デバイスの状態（レディ／ノット・レディ）をポーリングする
38	DRV_ACCESS	指定したドライブにアクセスする
27	DRV_ALLOCVEC	ディスク・アロケーション・ベクターのアドレスを得る
13	DRV_ALLRESET	全ディスク・ドライブをリセットする
31	DRV_DPB	デフォールト・ディスク・ドライブのDPBアドレスを得る
48	DRV_FLUSH	ディスク・バッファに残っているデータをディスクに書き込む
39	DRV_FREE	指定したドライブに対するアクセスを止める
25	DRV_GET	デフォールト・ドライブを得る
101	DRV_GETLABEL	指定したドライブのディレクトリ・ラベル・データを読む
24	DRV_LOGINVEC	ログ・イン・ディスク・ドライブのビット・マップを得る
37	DRV_RESET	指定したディスク・ドライブをリセットする
29	DRV_ROVEC	読み取り専用ディスク・ドライブのビット・マップを得る
14	DRV_SET	デフォールト・ドライブをセットする
100	DRV_SETLABEL	ディレクトリ・ラベルの生成または更新を行う
28	DRV_SETRO	デフォールト・ドライブを読み取り専用にセットする
46	DRV_SPACE	指定したドライブの空き領域を知る
30	F_ATTRIB	ファイル・アトリビュートをセットする
16	F_CLOSE	ファイルをクローズする
13	F_DELETE	ファイルを削除する
52	F_DMAGET	DMAバッファのアドレスを得る
26	F_DMAOFF	DMAバッファのオフセット・アドレスをセットする
51	F_DMASEG	DMAバッファのセグメント・アドレスをセットする
45	F_ERRMODE	BDOSのエラー・モードをセットする
42	F_LOCK	アンロック・モードでオープンされたファイル中のレコードをロックする
22	F_MAKE	ファイルを生成する
44	F_MULTISEC	BDOSマルチ・セクタ・カウントをセットする
15	F_OPEN	ファイルをオープンする
152	F_PARCE	ASCIIストリングを解析し，FCBをセットする
106	F_PASSWR	デフォールト・パスワードをセットする
36	F_RANDREC	シーケンシャル・レコード位置から，ランダム・レコード・フィールドをFCBにセットする
20	F_READ	ファイルのレコードをシーケンシャルに読み取る
33	F_READRAND	ランダム・レコードを読み取る
23	F_RENAME	ファイルの名前を変更する
17	F_SFIRST	指定したFCBと一致する最初のFCBをサーチする

システム・コール番号（10進数）	システム・コール名	機　　能
35	F_SIZE	ファイル・サイズを得る
18	F_SNEXT	F_SFIRST システム・コールで指定された FCB と一致する，次の FCB をサーチする
102	F_TIMEDATE	ファイルのタイム・スタンプとパスワードを得る
99	F_TRUNCATE	ファイルをトランケートする
43	F_UNLOCK	レコード・ロックを解除する
32	F_USERNUM	デフォールト・ユーザー番号のセット／ゲット
21	F_WRITE	レコードをシーケンシャルに書き込む
34	F_WRITERAND	ランダム・レコードを書き込む
103	F_WRITEXFCB	ファイルの XFCB の生成または更新を行う
40	F_WRITEZF	ランダム・レコードを書き込み，データ・ブロックの空きエリヤにゼロを書き込む
158	L_ATTACH	デフォールト・リスト・デバイスの所有権を確立する．リスト・デバイスが空いていないときには待たされる
161	L_CATTACH	条件付き L_ATTACH であり，リスト・デバイスが空いていないときにはエラー・コードが返される
159	L_DETACH	デフォールト・リスト・デバイスの所有権を放棄する
164	L_GET	デフォールト・リスト・デバイス番号を得る
160	L_SET	デフォールト・リスト・デバイスを変更する
5	L_WRITE	デフォールト・リスト・デバイスへのデータの書き込み
112	L_WRITEBLK	ブロック・データをデフォールト・リスト・デバイスに書き込む
128	M_ALLOC	メモリ・セグメントを割り当てる
130	M_FREE	指定したメモリ・セグメントをフリーにする
54	MC_ABS	指定されたアドレスで可能な最大量のメモリを割り当てる
58	MC_ALLFREE	そのプロセスが所有していた全メモリをフリーにする
55	MC_ALLOC	MCB で指定されたメモリ・セグメントを割り当てる
56	MC_ALLOCABS	指定されたアドレスから始まる，指定された量のメモリ・セグメントを割り当てる
57	MC_FREE	指定されたアドレスから始まるメモリ・セグメントをフリーにする
53	MC_MAX	システム中で可能な最大量のメモリを割り当てる
157	P_ABORT	名前またはプロセス・ディスクリプタで指定されるプロセスの実行を終了させる
47	P_CHAIN	指定されたプログラムをロードし，イニシャライズし，プログラムにジャンプする
150	P_CLI	指定したコマンド・ラインを解析し，実行する
144	P_CREATE	子プロセスを生成する
141	P_DELAY	指定されたティック回数期間プロセスの実行を中止する
142	P_DISPATCH	ディスパッチャにコントロールを戻す
59	P_LOAD	指定されたコマンド・ファイルをロードする
156	P_PDAR	プロセス・ディスクリプタのアドレスを得る
145	P_PRIORITY	プロセスの優先度をセットする
151	P_RPL	RPL からシステム・コールを行う
143	P_TERM	プロセスの実行を終了させる
0	P_TERMCPM	無条件のプロセスの実行を終了させる
138	Q_CREAD	条件付きキュー・リード，キューにメッセージがないときは，エラー・コードが返される
140	Q_CWRITE	条件付きキュー・ライト，キューが一杯のときにはエラー・コードが返される
136	Q_DELETE	指定されたキューを削除する
134	Q_MAKE	キューを生成する
135	Q_OPEN	キューをオープンする
137	Q_READ	キューからメッセージを読み取る
139	Q_WRITE	キューにメッセージを書き込む
12	S_BDOSVER	BDOS のバージョン番号を得る
50	S_BIOS	CP/M-86 BIOS のキャラクタ I/O ルーチンをコールする
163	S_OSVER	CDOS のバージョン番号を得る
107	S_SERIAL	CDOS のシリアル番号を得る
154	S_SYSDAT	SYSDAT のアドレスを得る
105	T_GET	日付および時刻（HH：MM）を得る
155	T_SECONDS	日付および時刻（HH：MM：SS）を得る
104	T_SET	日付および時刻（HH：MM：SS）をセットする

〈図15〉 CDOS 3.1 における XIOS コール

ファンクション番号(16進数)	ファンクション名	機能	
00	io_const	コンソール（キーボード）のステータス（データの有無）を得る	
01	io_conin	コンソール（キーボード）からデータを読み取る	
02	io_conout	コンソール（仮想コンソール）にデータを出力する	
03	io_listst	リスト・デバイス（プリンタ）のステータス（レディ／ノット・レディ）を得る	
04	io_list	リスト・デバイス（プリンタ）にデータを出力する	
05	io_auxin	aux デバイスからデータを読み取る	
06	io_auxout	aux デバイスにデータを出力する	
07	io_switch	その時点のアクティブ・コンソールをバック・グラウンドにし，指定された仮想コンソールをアクティブにする	
08	io_statline	ステータス・ラインのアップデート，またはステータス・ラインへストリング・データの書き込みを行う	
09	io_seldsk	ディスケット／ディスクの選択	
0A	io_read	ディスケット／ディスクからデータを読み取る	
0B	io_write	ディスケット／ディスクにデータを書き込む	
0C	io_flushbuf	ディスケット／ディスク・バッファに残っているデータを書き込む	
0D	io_poll	I/Oデバイスのステータス（レディ／ノット・レディ）をポーリングする	
0E	予　備		
0F	予　備		
10	ww_pointer	指定された仮想コンソールの仮想コンソール・データ・ストラクチャのアドレス，またはウィンドウ・データ・ブロックのアドレスを得る	
11	ww_key	ウィンドウ・マネージャ（WMENU）は，このファンクションを使ってキーボードからデータを読む	
12	ww_nstatline	データとアトリビュートをステータス・ラインに書き込む	
13	ww_im_here	ウィンドウ・マネージャ（WMENU）の状態をセットする	
14	ww_new_window	ウィンドウを設定する	
15	ww_cursor_view	カーソル・トラッキング・モード（固定／自動スクロール）とウィンドウの左上隅位置を指定する	
16	ww_wrap_column	仮想コンソールのラップ・アラウンド（wrap around）カラムを指定する	
17	ww_full	ウィンドウの状態をフル・スクリーン／ウィンドウの間でフリップフロップ的に切り替える	
18	ww_display	仮想コンソールのディスプレイ・タイプ（カラー／モノクロ）の切り替え	
19~1D	予　備		
1E	io_screen	スクリーンに対するデータの読み書きを行う	PC モード用の XIOS コール
1F	io_video	スクリーンに対する各種操作を行う	
20	io_keybd	PC モードのイネーブル／ディスエーブル	
21	io_shft	キーボードのシフト状態を得る	
22	io_eqck	装置構成情報を得る	
23	io_int13_read	PC モードのディスケット／ディスクの読み取り	
24	io_int13_write	PC モードのディスケット／ディスクの書き込み	

ールは XIOS 内部でも多用されており，割り込みルーチンと割り込み待ちルーチン間の同期制御に用いられています．

プロセス間の排他制御は，プロセス間通信と同じようにキューを介して行われます．この場合のキューは相互排他キュー（Mutual Exclusion Queue）と呼ばれています．

プロセス遊休状態の処置に関しては，P_DELAY および P_DISPATCH というシステム・コールが用意されています．P_DELAY は指定した回数のティック期間（1ティックは 10ms 程度），コントロールを放棄するためのシステム・コールであり，P_DISPATCH は RTM のディスパッチャにコントロールを戻し，より優先度の高いプロセスにコントロールを移す働きをします．

マルチ・タスク環境では，たとえ1ms の期間であっても，自分にすべきことがないときには他のプロセスにコントロールを移すよう心がけなければなりません．すべきこともないのにコントロールを持ち続けていると，CPU タイムがムダになり，システム全体の効率低下を招くことになります．

■CDOS の XIOS コール

CDOS 中の BDOS，CIO，PIN は XIOS をコールして物理的なデバイスをアクセスします．**図15**に，CDOS 3.1 における XIOS コールを示します．

この表中のファンクション 10H～18H はマルチ・ウィンドウ・サポートのために，XIOS のウィンドウ・マネージャをアクセスする XIOS コールであり，XIOS のバック・ドア・エントリ（裏口）と呼ばれているものです．ファンクション 1EH～24H は PC-DOS モードをサポートするための XIOS コールです．

XIOS コールは基本的には CDOS だけが行い，マルチ・タスク環境を破壊したくなかったら，アプリケーション・プログラムは XIOS コールを避けるべきです．