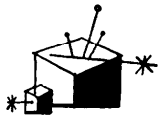
講　座



OS/2 における並行プログラムの作り方 (I)†

鷹　野　澄††

1. 総　論

並行プログラムとは，複数の逐次処理プログラムが並行して処理される形態のプログラムのことを言う．このとき，各逐次処理プログラムの具体的な並行処理方式についてはあまり問題にしない．すなわち，物理的に複数の処理装置を使って同時に処理してもよいし，1台の処理装置を時間的に分割して処理してもよい．

並行プログラムは，以前はどちらかというとオペレーティングシステムの中や，一部のシステムプログラムなどでしか作成されないものと考えられていた．しかし今日では，ハードウェアの高性能化とネットワークの高速化などにともない，迅速な応答を必要とするリアルタイムなコンピュータの利用が拡大し，いろいろなところで並行プログラムの必要性が高まってきている．

一方，並行プログラムを作成するための環境，すなわちオペレーティングシステムの機能やプログラミングツールは必ずしも満足なものではなかった．例えば，あるオペレーティングシステムでは並行プログラムを作成するための機能が十分でないとか，アセンブラ言語でなければ並行プログラムを作成できないとか，高級言語が使えてもそれは並行プログラミング用の特別な言語で，普段使っている高級言語では作成できないなどの問題があった．そして，このようなことが並行プログラムの作成を難しいものにしていたと考えられる．

しかし，最近のパソコンやワークステーションのオペレーティングシステムには，並行プログラムを作成するための便利な機能を整備し提供するものが増えてきた．そしてそれらをCや Pascal などの一般の高級言語からも利用できるようになってきた．このような環境では，パソコンやワークステーション上で高級言語を使って，だれでも気軽にそして容易に並行プログラムを作成することが可能である．

そこで本講座では，本格的な並行プログラミング環境を提供しているパソコン用オペレーティングシステム OS/2 (Operating System/2) を用いて，マルチスレッドやマルチプロセスといった並行プログラムの作り方と，それにともなうプロセス間通信や相互排他制御などについて，なるべく具体的に解説する予定である．本講座により，並行プログラミングの経験のない初心者から従来の“難しい”並行プログラミングの経験豊富なシステムプログラマまでの多くの方に，並行プログラムの“気軽な”作り方をお伝えできれば幸いである．

なお，OS/2 には，16 ビットの 80286 マイクロプロセッサ用のバージョン 1.0，1.1，1.2 と，32 ビットの 80386 マイクロプロセッサ用のバージョン 2.0 があるが，並行プログラミングに関しては，バージョンごとの差異は比較的少ないと思われる．本講座では，現在筆者の手元にあるパソコン NEO PC 9801 RA 5 上の日本語 MS OS/2 バージョン 1.1 を仮定しているが，他のバージョンで違いがある場合はできるだけ説明をつけるようにした．

1.1 MS-DOS の限界と OS/2 の特徴

OS/2 については，すでに石田（情報処理 1990 年 3 月号）において詳しく解説されているが，以下に並行プログラムの作成に関連した点などを中心に若干の解説を述べる．

OS/2 は，現在のパソコンで主流となっている MS-DOS オペレーティングシステムの後継として，1987 年 4 月に発表された．MS-DOS には，

- 利用可能なメモリが 640 KB しかない．
- 一度に一つの応用プログラムしか処理できないシングルタスク・オペレーティングシステムである．

という本質的な限界があり，今日のハードウェアの進

† Concurrent Programming in OS/2 (I) by Kiyoshi TAKANO (Earthquake Research Institute, University of Tokyo).

†† 東京大学地震研究所

歩や利用者ニーズの拡大に対応できなくなってきた．このためこれらの限界を克服し，さらに大幅な機能追加を行って新しく開発されたのが OS/2 である．

OS/2 で新たに追加された機能のうち，特に重要なものをあげると次のようになる．

（1） マルチタスク機能

OS/2 は MS-DOS と同様のシングルユーザ用であるが，本格的なマルチタスク機能を有している．

OS/2 では，プロセス (process) は現在実行中のプログラムを意味する．またプロセスの中にあって CPU の割り当ての単位となるものをスレッド (thread) と呼ぶ．OS/2 では複数のプロセスが生成され並行処理されるだけでなく，プロセス内においても複数のスレッドが生成され並行処理される．プロセスが生成されたときに生成される最初のスレッドは，プロセスを代表するスレッドで，スレッド 1 あるいはメインスレッドと呼ばれる．

並行処理の単位としてプロセスしかないと，プロセスの切り替えのオーバヘッドが大きいため迅速に切り替えられないという問題がある．このために，単に CPU を切り替えるだけの最小限のオーバヘッドで済む「軽いプロセス」としてスレッドが提供された．

なお OS/2 のマルチタスク機能を利用できるのは，インテル社の 80286 マイクロプロセッサのプロテクトモード上のみである．したがって，本講座では特に断わらないかぎり，単に OS/2 と言えば 80286 のプロテクトモード上で動作する OS/2 のことを指すものとする．

（2） 仮想記憶と保護機構

OS/2 では，一つの応用プログラムの暴走などにより他のプログラムやシステムが重大な影響を受けないように，ハードウェアによる仮想記憶や保護機構が採用されている．OS/2 が動作する 80286 のプロテクトモードでは，最大 16 MB まで実メモリを使用できる．しかし，実メモリのアドレスは直接指定できず，16 ビットのセレクタ (selector) と 16 ビットのオフセット (offset) からなる 32 ビットの仮想アドレスのみを指定できる．仮想アドレスから実アドレスへの変換は図-1 のように行われる．ここで，図-1 のディスクリプタテーブルは，実メモリ上のメモリ・セグメントの先頭の実アドレスと大きさなどを格納したものである．

このような仮想記憶機構に加えて，80286 のプロテクトモードには図-2 に示すような 4 段階の特権レベルがあり，OS/2 ではこのうちの三つのレベルが使われている．特権レベルによりディスクリプタテーブルを更新する命令などの CPU 特権命令の使用の制限や，カーネル内のセグメントへのアクセスの制限などの保護が実現される．

（3） ダイナミックリンク機能

OS/2 では，プログラムの実行開始時や実行中にライブラリを動的にリンクするダイナミックリンク機能が提供されている．ここで使用されるライブラリはダイナミックリンク・ライブラリ (dynamic-link library : DLL) と呼ばれる新しい形のライブラリである．DLL は，図-3 に示すようにデータとその初期化ルーチンおよびそれを外から利用するための操作関数群からな

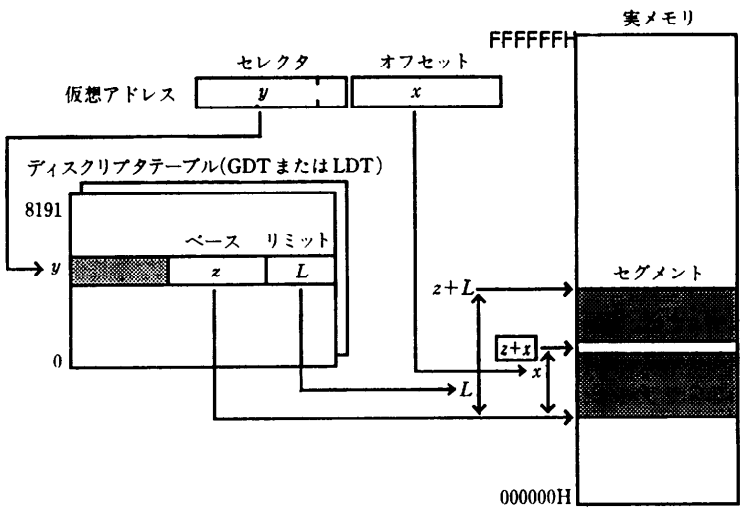


図-1　80286 の仮想アドレス方式

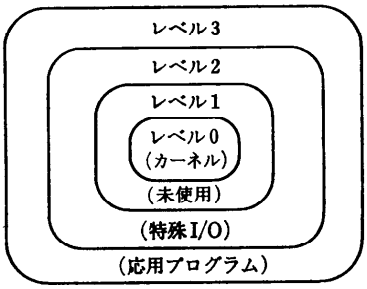


図-2 80286 のリング保護機構

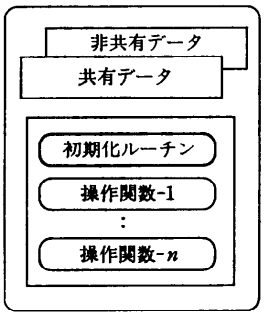


図-3 ダイナミックリンク・ライブラリの構造

り，いわゆる抽象データ型 (abstruct data type) と呼ばれる構造を持っているのが特徴である．DLL はメモリ上に一つだけ置かれ，それを利用するすべてのプログラムから共通にアクセスされる．DLL 内のデータには，すべてのプロセスから共通にアクセスできる共有データと，各プロセスごとに個別に割り当てられる非共有データがある．

並行プログラムの作成における抽象データ型の有効性は P. Brinch Hansen の Concurrent Pascal 言語における class や monitor によって示されたが，DLL はそのような機構を言語ではなく OS の機能として提供したものと言えよう．なお，DLL には monitor のような自動的な排他制御機能はないが，DLL の中で排他制御を記述できるので monitor を使ったのと同様なプログラミングも可能である．

（4） API とその標準化

OS/2 では，オペレーティングシステムが提供しているサービス関数を API (Application Program Interface) と呼んでいる．OS/2 では，すべての API が DLL で提供されているのが大きな特徴となっている．MS-DOS の場合，オペレーティングシステムを呼び出すにはオーバヘッドの大きい内部割り込みが使用されていた．しかし，OS/2 の場合は，ダイナミックリンク機能によって応用プログラムのロード時に OS/2 と DLL のリンクが済んでいるので，単にセグメント間のサブルーチン呼び出しである far コール (far call) を行うだけでよい．特権レベルの高いカーネルを呼び出すときも，まったく同じ far コールが使われる*，このため API の呼び出しにかかるオーバヘッドが少ない，API は名前で呼び出されるので分かりやすい，API の数については事実上制限がない，高級言語のサブルーチン呼び出し規約に従っているため，呼び出しのためのライブラリを介さずに直接高級言語から呼び出せるなどの利点がある．さらに OS/2 ではどの機種の上でも同じ仕様の API が提供されるように API の標準化が計られている．その結果，プログラムのロードモジュールレベルでの互換性が非常に高いものとなっている．

（5） マルチウィンドウ機能

OS/2 バージョン 1.1 から提供されている PM (Presentation Manager) 上では，複数のウィンドウをグラフィックスクリーンに重ね合わせたマルチウィンドウ機能が提供されている．マルチウィンドウ自体は並行プログラムでなくとも利用できるが，マルチプロセスやマルチウィンドウと組み合わせることにより，マルチウィンドウを利用した並行プログラムといったものの作成が可能である．

1.2 OS/2 カーネル上の並行プログラムの種類

OS/2 は，バージョンを重ねるごとに新たな機能が提供されているが，すべてのバージョンの基本となるカーネル部分の機能はほぼ共通である．ここではこの OS/2 のカーネル機能を使って作成できるマルチセッション，マルチプロセス，マルチスレッドといった並行プログラムの概要を述べる．

（1） マルチセッション・プログラム

OS/2 では，スクリーンをメモリ上で仮想化した仮想スクリーンという概念を導入し，複数の応用プログラムが互いに別々の仮想スクリーンを使ってそれぞれ独立に出力を行いながら並行に実行できるようにしている．そして OS/2 では，この一つの仮想スクリーン上の一連の処理をセッション (session) と呼んでいる．例えばエディタとコンパイラの二つのプログラムを別々のセッションで実行すれば，エディタを終了せずに別のセッションでその出力をコンパイルするなどの処理を簡単に行える．仮想スクリーンは，図-4 に示すよ

* カーネルのエントリアドレスとして 80286 のコールゲートと呼ばれるタイプのディスクリプタを使うことにより，呼び出しと同時に特権レベルの移行が行われる．

うに物理スクリーン全体に投影されるものと，PM のテキストウィンドウに投影されるものとがあり，前者の上での処理をフルスクリーン・セッション，後者の上での処理をウィンドウ・セッションと呼ぶ．なおOS/2の応用プログラムには NOTWINDOWCOMPAT, WINDOWCOMPAT, WINDOWAPI の各属性を付けることができるが，これらはそれぞれフルスクリーン・セッションでのみ実行可，ウィンドウ・セッションでも実行可，PM 上でのみ実行可ということを意味している．

OS/2 では，ユーザの応用プログラムから新たなセッションを起動し，そこで実行するプログラムを指定することができる．このときフルスクリーン・セッションとして起動するのか，ウィンドウ・セッションとして起動するのか，強制終了などの制御が可能な子セッションを起動するのか，制御が不可能な独立セッションを起動するのかなどの指定が可能である．マルチセッション・プログラムとは，この機能を使って複数のセッションを起動し，複数のプログラムを複数の仮想スクリーン上で並行処理するタイプの並行プログラムである．例えば，デバッガを作るときにプログラムからの出力をデバッグのための入出力とは別の仮想スクリーンに出したいような場合にはマルチセッション・プログラムが有効である．

（2） マルチプロセス・プログラム

マルチプロセス・プログラムは，複数のプロセスを並行処理するタイプの並行プログラムである．マルチセッション・プログラムも，複数のプロセスが並行処理されるという点ではマルチプロセス・プログラムの一形態と言えるが，ここでは分かりやすくするために，一つのセッションの中で複数のプロセスを並行処理するものをマルチプロセス・プログラムと呼ぶことにする．最も簡単なマルチプロセス・プログラムは，次のような複数のプロセスの標準入出力をパイプで結んだものであろう．

```
#include 〈process. h〉
main( ) {
system(" dir|sort|more "):
}
```

ここで system(cmd) は，OS/2 のコマンド行 cmd を実行するCランタイムライブラリで，この例では dir と sort と more の三つのプロセスがパイプによりデータを転送しながら並行処理される．

OS/2 ではプロセス間の保護機構がしっかりしているため，並行に走るプロセス間での処理の同期，メッセージの通信，データの共有などをどのように行うかがマルチプロセス・プログラムにおける最大の問題となる．このため OS/2 には，次の **1.3** で述べるような豊富なプロセス間通信機構が用意されている．

（3） マルチスレッド・プログラム

マルチスレッド・プログラムは，プロセスの中で複数のスレッドを生成しそれらを並行処理するタイプの並行プログラムである．OS/2 では，メモリやファイルなど CPU 以外のものはすべてプロセスに割り当てられ，CPU に関するもののみスレッドに割り当てられるため，プロセス内でのスレッドの切り替えの時間は非常に小さい．このためマルチスレッド・プログラムは，迅速な応答を要求するリアルタイムの応用プログラムに最も適したものである．たとえば，高速ネットワークのための通信プログラムや工場や実験室など

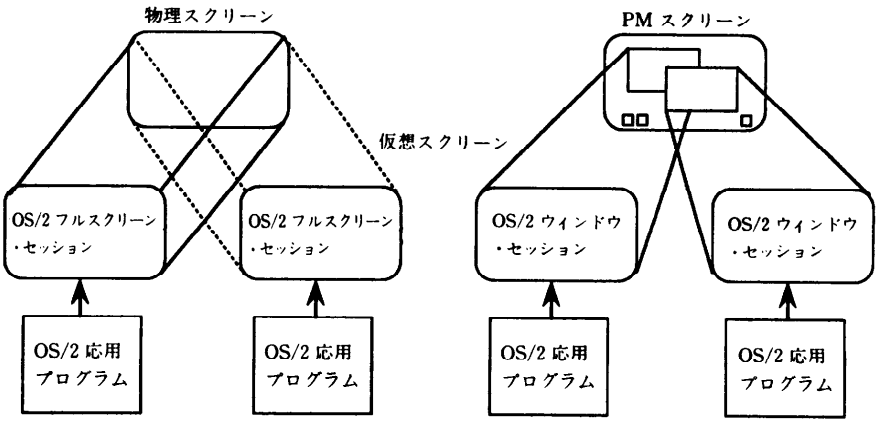


図-4 OS/2 のフルスクリーン・セッションとウィンドウ・セッション

での計測・制御用のプログラムなどは，マルチスレッド・プログラムで作成するのに適している例である．なお，リアルタイムの応用プログラムでは，高速な入出力を受け持つスレッドのように，特に迅速な処理を必要とするスレッドがしばしば存在する．そこで，そのようなスレッドに対して優先的に CPU を割り当てるプライオリティ（priority：優先順位）の設定が可能になっている．

マルチスレッド・プログラムにおいては，メモリやファイルなどのあらゆるものが共通にアクセスされるため，アクセスの競合が発生しやすいのが最大の問題である．このため，しばしば排他制御の必要性が発生する．しかし一方，スレッド間での処理の同期やメッセージの通信，データの共有などは容易である．

1.3 OS/2 カーネル上のプロセス間通信機構の種類

並行プログラムでは，プロセスやスレッドの間で互いの処理を同期したり，データやメッセージを交換することがよく行われる．このような通信をプロセス間通信（interprocess communication：IPC）と呼ぶ．

OS/2 ではカーネルの機能として，共有メモリ（shared memory），セマフォ（semaphore），パイプ（pipe），キュー（queue：待ち行列），シグナル（signal）などの IPC 機構が提供されている．なお，OS/2 では IPC と呼んでいるが，特にプロセス間に限ったものではなく多くのものはスレッド間でも利用可能である．

（1） 共有メモリ

OS/2 にはプロセス固有のメモリ・セグメントとプロセス間で共有できるメモリ・セグメントの2種類があり，前者をローカルメモリ，後者を共有メモリと呼んでいる．プロセスが生成されるときに割り当てられるのはローカルメモリで，これは別のプロセスと共有することはできない．このためプロセス間でメモリの共有を行いたい場合は，オペレーティングシステムに対して共有メモリの割り当てを要求することが必要である．

OS/2 では，名前つき共有メモリと名前なし共有メモリの2種類の共有メモリの割り当てと利用のためのサービスを用意している．名前つき共有メモリは，ファイル名と同様の ¥SHAREMEM¥path¥name.ext といった名前をつけて割り当てられたもので，その名前さえ知っていればどのプロセスでも共有可能である．一方，名前なし共有メモリは，プロセスが通常のメモリ割り当て要求を行うときに，供与可能／取得可能といった指定をすることにより割り当てられるものである．これは，それを割り当てたプロセスからそのメモリ・セグメントのセレクタを通知されるなどによって使用を許可されたプロセスのみが共有できる．

以上のほかに，DLL 内の共有データも共有メモリの一つである．上述の共有メモリはプロセスの要求によって割り当てられて利用可能となるが，DLL 内の共有データは，あらかじめメモリ上に割り当てられているので，それにリンクしているプロセスならいつでも利用可能である．

なお，共有メモリのためのサービスは，おもにプロセス間に対するもので，スレッド間ではプロセスのすべてのメモリが常に共有されるためこの種のサービスは不要である．

（2） セマフォ

セマフォは，同期や排他制御のために E. W. Dijkstra が提案したツールである．OS/2 のセマフォも同様に，プロセス間やスレッド間での同期や排他制御に使用されるが，Dijkstra のセマフォに比べて自分自身が再帰的に排他制御を要求してもよいなどの拡張がなされている．

OS/2 ではシステムセマフォ，RAM セマフォそしてバージョン 1.1 から導入された FS RAM セマフォ（Fast-Safe RAM semaphore）の3種類のセマフォが提供されている．システムセマフォはおもにプロセス間の同期や排他制御に使用されるもので，ファイル名と同様の ¥SEM¥path¥name.ext といった名前でシステム内に生成される．システムセマフォでは，プロセスが排他要求を出したまま終了するようなときに，セマフォが自動的に解放されるのが特徴である．一方 RAM セマフォは，プロセス間でも使えるが，特にスレッド間での同期や排他制御に有効なもので，それらの共有メモリ上に置かれた4バイトの変数である．RAM セマフォの場合は，セマフォを解放しないまま終了したときにそのままの状態で残ってしまうのが欠点である．これは特に複数のプロセスから利用される DLL の中で RAM セマフォによる排他制御を再帰的に行っているような場合に厄介な問題となる．このためにバージョン 1.1 で新しく導入されたのが FS RAM セマフォである．FS RAM セマフォは排他制御にのみ利用でき，現在どのスレッドが何回再帰的に要求を出しているかなどの情報を 14 バイトの変数領域の中に保持し，プロセスが強制終了されるような場合にセマフォをきれいに解放できるという特徴を有する．

(3) パ イ プ

パイプは，スレッド間やプロセス間でファイルの入出力関数を使ってデータ転送を行うために考案されたツールで，その実体はプロセス間の共有メモリ上に置かれた先入れ先出し (first-in first-out, FIFO) 型のバッファである．OS/2 では，名前なしパイプとバージョン 1.1 から導入された名前つきパイプが提供されている．名前なしパイプの場合は，それを生成したプロセスとその子供のプロセス同士，あるいはその孫プロセスなどでしか使えない．このため一つのサーバプロセスに対し複数の任意のクライアントプロセスがデータを送受信するようなサーバ／クライアント型のデータ転送には使いにくいという不満があった．このために導入されたのが名前つきパイプである．名前つきパイプでは，サーバプロセス側でファイル名と同様の ¥PIPE¥path¥name. ext というような名前でパイプを生成しておけば，クライアントプロセス側ではこの名前を通常のファイルと同様にオープンし，あたかもファイルの読み書きのようにサーバプロセスとデータの送信や受信ができるのが特徴である．

(4) キ ュ ー

キュー(待ち行列とも言う)も，パイプと同様にスレッド間やプロセス間でデータ転送を行うためのツールである．キューも名前つきパイプと同様に，おもにサーバ／クライアント型のデータ転送に使用されることを想定している．このためキューの場合も，サーバプロセス側で ¥QUEUES¥path¥name. ext のような名前でキューを生成しておき，クライアントプロセス側でその名前を指定してデータの転送を行う．ただし，データの転送方向はクライアントからサーバのみであること，ファイル入出力関数ではなくキュー専用の入出力関数を使うこと，転送データはキューの中ではなく共有メモリ上に置かれキューの中にはそのアドレスのみが置かれることなどが異なる．そしてキューの最も大きな特徴は，キューの生成時に先入れ先出し型だけでなく，後入れ先出し (LIFO : last-in first-out) 型や優先順出し型の取り出しが指定でき，さらにそれにこだわらずに任意の順序で特定のメッセージを抜き出すこともできるなど，キューからの取り出し方が非常に柔軟な点にある．

(5) シ グ ナ ル

シグナルは，実行中のプロセスに対してその処理を中断して強制的に事象を通知するメカニズムである．OS/2 には，キーボードからの Ctrl+C や Ctrl+Break (NEC では STOP キー) の入力シグナル，プロセス終了シグナル，パイプ消滅シグナルおよびプロセスフラグA，B，Cの通知シグナルなどが定義されている．

プロセスは各シグナルに対し，シグナル・ハンドラ (signal handler) と呼ばれるシグナル処理ルーチンをあらかじめ登録しておくことにより，それを受信し処理することができる．シグナル・ハンドラが用意されていない場合は，Ctrl+C シグナル，Ctrl+Break シグナルおよびプロセス終了シグナルではプロセスが終了し，パイプ消滅シグナルやプロセスフラグA，B，Cシグナルについては無視される．

なお，シグナル・ハンドラはプロセスに対してのみ用意でき，プロセス内のスレッド1によって処理される．したがってシグナルはプロセス間でのみ利用され，スレッド間では利用されない．

1.4 PM における並行プログラムと IPC の種類

OS/2 の応用プログラムのうち，カーネルが提供する機能のみを利用して作成したものを OS/2 基本応用プログラム，PM が提供する機能を利用して作成したものを PM 応用プログラムと呼ぶ．PM 応用プログラムにおいても並行処理の単位はプロセスやスレッドであり，カーネルのマルチタスク機能や IPC が利用可能である．さらに，PM が提供する独自の IPC 機能を使った並行プログラミングも可能である．

(1) メッセージ駆動型のプログラミングにおける並行処理

PM 応用プログラムは，従来型のプログラムとは異なり，メッセージ駆動型 (message-driven) のプログラムであるという点が大きな特徴である．PM 応用プログラムは，常にメッセージキュー (message queue) からメッセージを取り出してそれに応じた処理を行うということを繰り返している．例えばキーボードからの文字入力の場合を例にとると，従来型のプログラムではキー入力関数を呼んだときにのみ入力を検知できた．しかし，PM では，システムが常にキー入力を検知して何かあるとすぐに新しいメッセージを生成しメッセージキューに入れてくれるので，プログラムは単にメッセージを取り出すという一つの動作だけでキー入力などすべての事象の発生を簡単に知ることができるというぐあいである．

メッセージ駆動型プログラミングでは，メッセージの取り出しから次のメッセージの取り出しまでが，一つの連続した処理の単位である．もし最初のメッセー

ジ処理と次のメッセージ処理が互いに独立で実行順序についてもどちらでもよいなら，それらを並行処理できるということに気づくであろう．そこで例えば時間がかかるメッセージの処理については別のスレッドに下請けを頼むというような並行処理が可能である．別のスレッドに頼むことでメッセージキューからのメッセージの取り出しを迅速に行える．

（2） マルチウィンドウとマルチスレッド

PM では，一つのスレッドでたくさんのウィンドウを生成できる．このときメッセージキューは一つしかないため，スレッドは取り出したメッセージをそのうちの一つのウィンドウに振り分けることになるが，それではマルチウィンドウ上での迅速な並行処理を期待できない．このため OS/2 ではマルチスレッドを用い，各スレッドが独立にメッセージキューとウィンドウを生成するという形でのマルチウィンドウの作成が可能となっている．この場合，各ウィンドウには，それぞれ独立したメッセージキューとそれを処理するスレッドが存在するので，メッセージの取り出しもその処理も完全に並行に行われる．同様なことは複数のプロセスでも行えるが，プロセスの切り替えのオーバヘッドを考えるとマルチスレッドのほうが適しているように思える．

（3） クリップボードと DDE

クリップボード (clipboard) やダイナミック・データエクスチェンジ (dynamic data exchange : DDE) は，PM が提供する IPC のツールである．クリップボードは，カット，コピー，ペーストなどの操作で，互いに独立な PM 応用プログラムの間でもデータの転送を行うことができる．クリップボードでは利用者が操作を行ったときにのみ転送できるが，DDE を使うと，PM 応用プログラム間で片方から自動的にデータを転送するようなことができる．例えば A 社の表計算プログラム上でデータを変更すると，それが自動的に別の B 社で作成されたグラフ描画プログラムに転送され，ウィンドウ上のグラフが瞬時に書き変わるというようなことが DDE を用いると可能になる．

1.5 OS/2 の応用プログラムの作成の概要

（1） OS/2 基本応用プログラムの作成の概要

OS/2 基本応用プログラムは，OS/2 対応のコンパイラとリンカさえあれば容易に開発できる．例えば MS-C の場合，ソースプログラムが xx.c のとき，

```
cl -AL xx.c xx.def
```

と入れるだけで，図-5 に示したような手順でロードモ

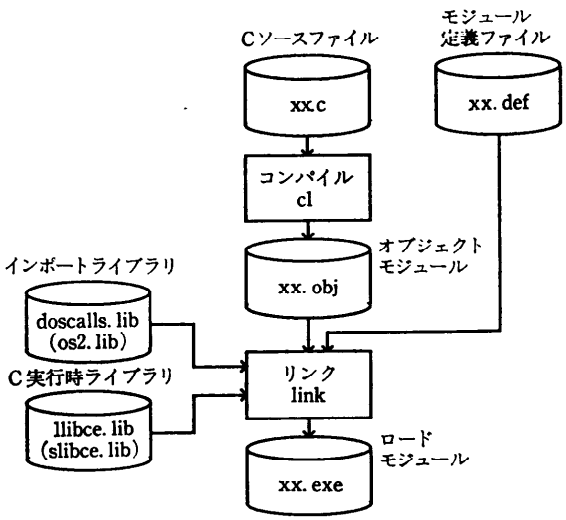


図-5 OS/2 基本応用プログラムの作成手順

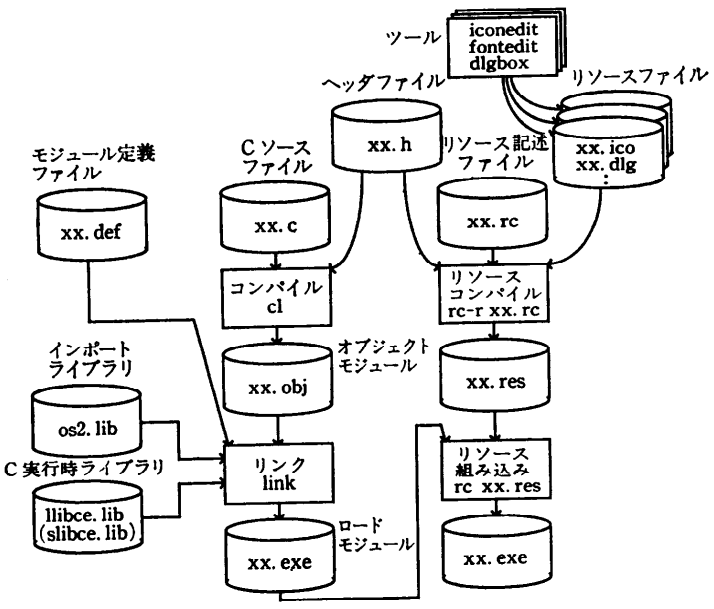


図-6 PM 応用プログラムの作成手順

ジュールの xx. exe を作成できる．ここで，モジュール定義ファイル (module-definition file) は応用プログラムの属性などを定義するときに用いられる．またインポートライブラリ (import library) は API のエントリを定義しているもので，通常は OS/2 に付属の doscalls. lib を使用する．なお MS-C にも doscalls. lib がついているが，古いバージョン用なのでこれを使わないように注意する必要がある．またPM応用プログラムの開発キットをもっている場合は，os2.lib を次のように指定してもよい．

cl -AL xx.c xx.def os 2.lib

使用する C コンパイラのメモリモデルは，API が far コールでありそのパラメータも far ポインタが使われているので，上例のように -AL を指定してラージモデルにしておくのが無難である．もちろん，簡単なものならスモールモデルでコンパイルしても問題ない．

(2) PM 応用プログラムの作成の概要

PM 応用プログラムを作成する場合は，コンパイラとリンカのほかにリソースコンパイラやアイコンエディタなどの専用の開発ツールが必要である．図-6 に PM 応用プログラムの開発手順を示した．各処理に対応したコマンドの例は次のようなものである．

cl -c -AL -G2sw -W3 xx.c …コンパイル
link xx, /align : 16, NUL, os2, xx.def…リンク
rc -r xx.rc …リソースのコンパイル
rc xx.res …リソースの組み込み

ここでリソースコンパイラ rc は，リソース記述ファイル xx. rc からバイナリ形式のリソース xx. res を作成したり，それをロードモジュールの xx. exe の中に組み込むのに用いられる．なおリンク時のインポートライブラリは，PM 応用プログラムの開発キットの os 2. lib のほうを使う必要がある．

参 考 文 献

【OS/2 開発の背景やねらいなど】
1) Letwin, G.: INSIDE OS/2, Microsoft Press (1988) [三浦訳：OS/2 システム・アーキテクチャ，アスキー (1988)].
2) マイクロソフト(株)監修：日本語 OS/2 プレリリース・セミナー，アスキー出版局 (1988).
3) 石田晴久：OS/2 の現状と展望，情報処理，Vol. 31, No. 3, pp. 380-389 (1990).
4) 脇 英世：OS/2 への招待，講談社ブルーバックス (1989).
【OS/2 の概念やコマンド操作】
5) Jamsa, K.: Using OS/2, Osborene McGraw-Hill (1988).
【カーネルプログラミング】
6) Duncan, R.: Advanced OS/2 Programming, Microsoft Press (1989) [福崎訳：プログラミング OS/2 I, アスキー (1990), 山田訳：プログラミング OS/2 II, アスキー (1990)].
7) Iacobucci, E.: OS/2 Programmer's Guide, Osborene McGraw-Hill (1988).
8) 川井健一，佐貫俊幸，船木義昭：OS/2 入門，培風館 (1988).
9) 石田晴久，鷹野 澄：OS/2, 共立出版 (1990).
【PM プログラミング】
10) Petzold, C.: Programming the OS/2 Presentation Manager, Microsoft Press (1989).
11) 富士ソフトウェア：OS/2 プレゼンテーション・マネジャ，富士ソフトウェア (1989).
【並行プログラミングと OS】
12) Peterson, J. L. and Silberschatz, A.: Operating System Concepts 2nd Edition, Addison-Wesley (1985). [宇都野宮，福田共訳：オペレーティンシステムの概念上／下，培風館 (1987)].
13) Hansen, P. B.: Operating System Principles, Prentice-Hall, Englewood Cliffs, NJ (1973).
14) Hansen, P. B.: Concurrent Programming Concepts, Computing Surveys, Vol. 5, No. 4 (1973).
15) Hansen, P. B.: The Programming Language Concurrent Pascal, IEEE Tran. SE Vol. SE-1, No. 2 (1975).
16) Dijkstra, E. W.: The Structure of the "THE" —Multiprogramming System, CACM, Vol. 11, No. 5 (1968).
17) Hoare, C. A. R.: Monitors: An Operating System Structuring Concept, CACM, Vol. 17, No. 10 (1974).

(平成 2 年 8 月 13 日受付)