MS OS/2 PMのプログラミングについて

―― マルチスレッド及びヘルプ ――

行 方 常 幸

目　次

1．はじめに

ここでは具体的なプログラムを通してOS/2のマルチスレッド及びPM（プレゼンテーションマネージャ）におけるヘルプに関して述べる。

シングルユーザマルチタスクのOSであるOS/2は複数のプログラム（プロセス）を同時に実行できる。さらに1個のプログラム（プロセス）をいくつかのスレッドに分割してそれらを同時に実行できる。すなわちOS/2において実行単位はスレッドであり，1個のプログラム（プロセス）が1個のスレッドから成り立っている場合や，複数のスレッドから成り立っている場合があるわけである。例えば，C言語でプログラムを書くときはmain関数はメインスレッドを構成し，プログラム中で新しくスレッドを生成しない限り1個のスレッドからなるプログラムとなる。OS/2のGUI環境であるPMのプログラムにおいては長い処理時間を要する仕事を別スレッドとして生成し，ユーザーからの（キーボードやマウスによる）入力を処理する部分と別けることによりユー

ザーからの入力に対する反応を早くすることが出来る。新しいスレッドを生成するのはシステム関数を呼び出すだけでよいのだが，マルチスレッド化により新たに対処しなければならない問題（資源の共有）も生じる。

PM のプログラムではユーザーの便宜のために各メニュー項目等にヘルプをつけることが勧められている。ユーザーがどうしたら良いか分からない時に＜ｆ・１＞キーを押せば適当な指示を記載したヘルプ画面を表示するようにするのである。

以上のマルチスレッド及びヘルプを実現する方法を以下に説明する。利用したプログラム例は以前発表した「フルスクリーン用プログラムの始動」（STARTFUL.EXE）である。図にソースプログラムの一部を入れてあるが，これらは元の完全なものから大部分を削除してあり，プログラムの流れの大体の感じを表すことだけを意図している。また，関数のうち Win と Dos で始まるものはシステム関数であり，それ以外は作成したものである。

2．マルチスレッド

OS／2 の実行単位はスレッドと呼ばれるものである。このスレッドをある時間実行して次にまた別のスレッドを実行する。これを続けることにより複数のスレッドを同時に実行している。ここであるスレッドの処理を中断したときにどのスレッドの処理を始めるかであるが，各スレッドは３つの優先順位のクラス（タイムクリティカル，レギュラー，アイドルタイム），さらに各クラスが32のレベルに分かれている，のどれかに属しており，このうち優先順位の高いものから順に CPU 時間が割り当てられる。長い処理時間がかかる仕事を別スレッドとして生成し，ユーザーからの入力を処理するスレッドの優先権を上げておくと長い仕事の処理中でも優先権の高いユーザーからの入力を先に処理することになる。「フルスクリーン用プログラムの始動」（STARTFUL.EXE）において以下のようにマルチスレッド化を行った。

「フルスクリーン用プログラムの始動」ではリストボックスからプログラムを選択することによって始動させる。このプログラムを実際に始動させる部分

を別スレッドとして生成した。まず，main 関数内でユーザーの入力処理を含むクライアントウィンドウプロシージャ ClientWndProc の属するメインスレッドの優先権を１レベル上げる。（図. １参照）

DosSetPrty(PRTYS_THREAD, PRTYC_NOCHANGE, 1, 0) ;

```
#include "myhelp.h"          /* ヘルプのためのヘッダファイル */

USHORT main(SHORT argc)
      {
      hab = WinInitialize(0);
      hmq = WinCreateMsgQueue(hab, 0);
      WinRegisterClass(hab, szClientClass,
              ClientWndProc, CS_SIZEREDRAW, 0);
      hwndFrame = WinCreateStdWindow(HWND_DESKTOP, 0L,
                    &flFrameFlags, szClientClass,
                    NULL, 0L, (HMODULE)NULL, ID_RESOURCE,
                    &hwndClient);
      if (!InitHelp(hab, hwndFrame))  /* ヘルプインスタンスの作成 */
              {
              WinMessageBox(HWND_DESKTOP, HWND_DESKTOP,
                            "ヘルプがインストール出来ません！",
                            szClientClass, 0,
                            MB_OK | MB_ICONEXCLAMATION);
              WinDestroyWindow(hwndFrame);
              WinDestroyMsgQueue(hmq);
              WinTerminate(hab);
              return 1;
              }
      DosSetPrty(PRTYS_THREAD, PRTYC_NOCHANGE, 1, 0);
                    /* メインスレッドの優先権を１レベル上げる */
      while (TRUE)
              {
              while (WinGetMsg(hab, &qmsg, NULL, 0, 0))
                      WinDispatchMsg(hab, &qmsg);
              if (WinDlgBox(HWND_DESKTOP, hwndClient,
                            EndDlgProc, (HMODULE)NULL, IDD_END, NULL))
                      break;
              WinCancelShutdown(hmq, FALSE);
              }
      TerminateHelp(hwndFrame);       /* ヘルプインスタンスの破壊 */
      WinDestroyWindow(hwndFrame);
      WinDestroyMsgQueue(hmq);
      WinTerminate(hab);
      return 0;
      }
```

図. 1　main 関数

後はリストボックスからのプログラムが選択されたというメッセージを受け取った時点でそのプログラムを始動するスレッドを生成するだけである。新しいスレッドを生成するにはシステム関数 DosCreateThread を呼び出すか，C言語のライブラリ関数_beginthread を呼び出すことによって可能である。後者を使う場合はメモリモデルとしてラージモデルを利用する必要があり，私としてはなるべくスモールモデルでプログラミングを行いたいので，前者を利用することにした。リストボックスからのメッセージにより関数 StartSession が呼び出されスレッド用スタックなどの領域を確保し必要なデータを初期化し，その後，システム関数 DosCreateThread を呼び出す(図. 2 参照)。システム関数 DosCreateThread の引数は生成するスレッド（スレッドと言っても見かけは普通の関数と同じ）のアドレス，スレッド ID を受け取る変数のアドレス，スタック領域のアドレスである。ここで少しだけ注意するところがある。システム関数 DosCreateThread の第 1 引数でアドレスを指定する関数は引数を持つことが出来ない，とマニュアルではなっている。しかしながら文献［1］に習って，強引にスタックに引数を入れ，それを使う関数も引数を取るように定義しておくと実際はうまく行くようである（コンパイラをだますために少し細工は要るようだが！）。別スレッド StartThread では引数から始動すべきプログラム情報を獲得しシステム関数 DosStartSession でプログラムを始動させ，その後このスレッドが利用したスタック領域を解放するようにメッセージをメインスレッドにポスト（WinPostMsg）し，自分自身だけを終了させる。このようにプログラムしておくと，スレッド StartThread はメインスレッドより優先権が 1 レベル低いためメインスレッドに処理しなければならない仕事が残っている間はメインスレッドを先に実行する。すなわちユーザーの入力処理が待たされることはない。

さて，別スレッド StartThread を生成する毎にそれ用のスタック等の領域を確保しなくてはならない。別スレッドをいくつ生成するかはあらかじめ分からないことと，共通の資源であるメモリを有効に利用するめに，別スレッドを生成する毎に領域を確保し，スレッドが終了すれば解放することが望ましい。

```
USHORT StartSession(HWND hwnd)
      {
      VOID FAR        *dummy;

      /* スレッド用スタックなどの領域確保 */
      /* 構造体DOSSTARTDATAの初期化 */

      dummy = (VOID FAR *)StartThread;
      DosCreateThread((PFNTHREAD)dummy, &(pDosStData->tid), stack);
      return VALID;
      }

VOID FAR StartThread(PDOSSTARTDATA pDosStData)
      {
      DosStartSession(&(pDosStData->stdata), &(pDosStData->idSession),
                  &(pDosStData->pid));
      WinPostMsg(pDosStData->hwnd, WM_MYMESSAGE_FREE,
            MPFROMSHORT(pidinfo.tid), MPFROMSHORT(pDosStData->pData));
      DosExit(EXIT_THREAD, 0);
      }

VOID SuspendAllThread(VOID)
      {
      MEMBLOCK *pMemBlock;
      DOSSTARTDATA    *pDosStartData;
      DosEnterCritSec();
      for (pMemBlock = pStartData; pMemBlock;
                  pMemBlock = (MEMBLOCK *)(pMemBlock->ppChar))
            {
            pDosStartData = (DOSSTARTDATA *)pMemBlock->Char;
            DosSuspendThread(pDosStartData->tid);
            }
      DosExitCritSec();
      }
```

図. 2　スレッドの生成、実行中断

そのために別スレッド StartThread 終了間際に作成者のメインスレッドに確保してくれたメモリがもう不要になる由を伝えれば良い。細かい話になるが，メッセージを送るには 2 個のシステム関数 WinSendMsg と WinPostMsg があり，関数 StartThread では WinPostMsg を利用している。それはスレッド StartThread はメインスレッド main のようにシステム関数 WinCreate MsgQue を呼び出してメッセージキューを作成していないので WinSend-Msg を利用できないためである。WinSendMsg は送付したメッセージの処

理が終わるまで呼び出し元へ戻ってこないが，WinPostMsg はメッセージを送付先のメッセージキューに入れたらすぐに戻ってくる。さて，最後にメインスレッドが別スレッド用に確保したメモリを解放するときに若干の問題点が残る。別スレッドが終了する前にこのスレッド用に確保したメモリを解放してしまうと最悪の場合暴走してしまう。また，確保したメモリを解放する必要があるのは別スレッドがもう自分は終了するから解放せよ，という場合のほかにユーザーがプログラム自体を終了する時の後処理の場合もある。前者の処理は図. 3 の ClientWndProc のメッセージ WM__MYMESSAGE__FREE の所，後者の処理はメッセージ WM__DESTROY の所で行っている。どちら

```
MRESULT CALLBACK ClientWndProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2)
        {
     switch(msg) {
             case WM_COMMAND:
                     switch (COMMANDMSG(&msg)->cmd)
                             {
                             case IDM_ABOUT: /* 必ず処理するようにする */
                                     WinDlgBox(HWND_DESKTOP, hwnd,
                                             AboutDlgProc, (HMODULE)NULL,
                                             IDD_ABOUT, NULL);
                                     return 0;
                             CASE_REPLY_HELP_AAB_MSG
                             /* WM_COMMANDメッセージ処理文中に入れる */
                             }
                     break;
             CASE_REPLY_HM_MSG
                     /* メッセージ処理文中(一番外のswitch)に入れる */
             case WM_MYMESSAGE_FREE:
                     DosSuspendThread(SHORT1FROMMP(mp1));
                     pStartData = (MEMBLOCK *)MemFree(pStartData,
                                     (CHAR *)SHORT1FROMMP(mp2));
                     return 0;
             case WM_DESTROY:
                     SuspendAllThread();
                     if (pStartData)
                             MemFree(pStartData, NULL);
                     return 0;
             }
     return WinDefWindowProc(hwnd, msg, mp1, mp2);
     }
```

図. 3 ウィンドウプロシージャ ClientWndProc

もメモリを解放する前に利用者であるスレッドをシステム関数 DosSuspend Thread で中断し，この後メモリを解放している。これでプログラムが暴走してしまう心配はない。メッセージ WM_DESTROY 処理中に呼び出している関数 SuspendAllThread ではスレッドを中断する前後でシステム関数 DosEnterCritSec と DosExitCritSec を呼び出している。この 2 個の関数呼び出しの間に書かれている処理は他のスレッドによって中断されない。

以上のようにマルチスレッド化したことによる実際の効果は次のようである。バックグラウンドで実行，実行時最小化しない，と設定しておいてから，例えば，マウスをシステムメニューボックスの所に移動させておき，CMD. EXE にカーソルを移動し素早くリターンキーを10回押す。その後また素早くマウスをダブルクリックし終了してもよいかの質問に対してリターンキーを押しはいと答える。画面には 2，3 個の今始動させた CMD. EXE のアイコンが残り，STARTFUL. EXE はすでに終了している。これにより CMD. EXE を始動させるスレッド StartThread を10回作成したが，優先権が 1 レベル高いメインスレッドがユーザーからの終了要求を処理するまでに 2，3 個の StartThread が CMD. EXE を始動することができ，その他は途中で中断させられたことが分かる。マルチスレッド化していない場合は CMD. EXE が10個始動されるまで STRTFUL. EXE が終了することはない。この例は多分に恣意的なものであるが，時間のかかる処理を間違って実行させてしまってからそれを取り消したくてもその手段がなく長い時間待つ羽目に陥ったユーザーならこのマルチスレッド化の有用性は納得して頂けると思う。

3．ヘルプ

ソフトウェアを利用している時，途中で操作法が分からなくなって困った，等ということは誰でも経験することと思われる。その時まさに困っている場面のヘルプがあったらどんなにソフトウェアが使いやすくなるであろうか？ここでは，このヘルプ機能を如何にすれば実現できるかを説明する。現在，日本語 MS OS／2 は Ver. 1. 21 になっているが，私がヘルプ機能を最初に実現しよ

うとした5カ月ぐらい前はVer.1.1であった。Ver.1.1の時には無かったヘルプマネージャと呼ばれるものがVer.1.21には組み込まれており，ヘルプを実現する労力が大分軽減されている。ここではヘルプマネージャを使わない方法と使う方法の両方を説明する。Ver.1.21のヘルプマネージャを使う方が目次やハイパーテキストを利用できる等の利点があるが，ヘルプを関連付けたウィンドウをヘルプウィンドウの前面に持ってきたりヘルプウィンドウをアイコンにできない欠点がある。

ユーザーはヘルプを必要とする時に＜f・1＞キーを押したり，ヘルプボタンがあればマウスでそれをクリックする。この動作を感知してその時にキー入力を待っていたウィンドウに関するヘルプを表示すればよいわけである。通常では＜f・1＞キーを押したりMIS_HELPスタイルを持つメニュー項目（メニューバーの右端にある「F1＝ヘルプ」）をマウスでクリックすると，その時にキー入力を待っていたウィンドウにWM_HELPメッセージが送付される。また，ヘルプボタンをクリックするとボタンのオーナーウィンドウへWM_HELPメッセージが送付される。そこで，ヘルプを処理したいウィンドウプロシージャにWM_HELPメッセージの処理として適当なヘルプを表示するようにプログラムすればよい。しかしながら各ウィンドウプロシージャでWM_HELPを処理する方法ではメニユー項目に対するヘルプを処理することができない。メニュー項目に対するヘルプを処理するためにはヘルプフックと呼ばれるものを利用しなくてはならない。ヘルプマネージャを利用しない「方法1」もヘルプマネージャを利用する「方法2」も共にヘルプフックを利用している。「方法1」ではヘルプフックを直接利用し，ヘルプを表示するウィンドウを自分で作る必要があるためプログラムの中心はこのウィンドウの管理であり，全ての場合に対するヘルプを準備することとヘルプを利用し易くすること等に関しては対処していない。それに対して「方法2」ではヘルプフックを利用している部分は我々ユーザーには隠されており，全ての場合に対するヘルプを準備でき，さらに細かい点にも配慮がなされている。ただ，最初にも述べた通り，ヘルプウィンドウが背面に移動しないこと，アイコン化出来ないこ

とが私には欠点と思われる。以下，「方法１」に関してはヘルプウィンドウの管理を中心に，「方法２」ではヘルプマネージャの利用法を中心に述べる。

「方法１」

方法１ではメニューとメッセージボックスのヘルプにヘルプフックを利用し，それ以外は各ウィンドウプロシージャでWM__HELPを処理することで対応している。また，ダイアログボックスでは１つのヘルプしか表示できずその子ウィンドウのヘルプには対応していない。先ずmain関数でメッセージループへはいる前にヘルプフックを設定しておく。（図. 4参照）

WinSetHook (hab, hmq, HK__HELP, (PFN) HelpHook, NULL)；

こうしておくと，例えば，メニュー項目を矢印キーで選択して＜ f・1 ＞キーを押すとその情報が４番目の引数に指定してある関数 HelpHook へ送られる。メッセージループが終了したら

WinReleaseHook(hab, hmq, HK__HELP, (PFN)HelpHook, NULL)；

と設定を無効にしておく。関数 HelpHook（図. 4参照）ではヘルプ要求がメニューからか処理すべきメッセージボックスからかを判断しその場合には

WinSendMsg (hwndHelp, WM__MYMESSAGE__HELP__OPEN, MPFROMSHORT(idHelp), NULL)；

と表示すべきヘルプ内容を区別するidHelp（この場合はメニュー項目の識別子またはメッセージボックスの識別子）を含むメッセージWM__MYMESSAGE__HELP__OPEN（表示要求を意味するユーザー定義のメッセージ）をヘルプを表示するウィンドウに送付する。その他ヘルプを付けるウィンドウ（クライアントウィンドウ，ダイアログボックス等）でも，図. 4のClientWndProcのようにWM__HELPメッセージの処理として適当なidHelpを指定して

WinSendMsg(hwndHelp, WM__MYMESSAGE__HELP__OPEN, MPFROMSHORT(idHelp), NULL)；

```
#include "HELPWND.H"

MRESULT CALLBACK ClientWndProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2);
BOOL CALLBACK HelpHook(HAB hab, USHORT usMode, USHORT idTopic,
                       USHORT idSubTopic, PRECTL prcPosition);
USHORT main(VOID);

USHORT main(VOID) {
      hab = WinInitialize(0);
      hmq = WinCreateMsgQueue(hab, 0);
      WinRegisterClass(hab, szClientClass, ClientWndProc, CS_SIZEREDRAW, 0);
      hwndFrame = WinCreateStdWindow(HWND_DESKTOP, 0L,
                                     &flFrameFlags, szClientClass,
                                     NULL, 0L, NULL, ID_RESOURCE,
                                     &hwndClient);
      if (hwndFrame == NULL)
            {
            WinDestroyMsgQueue(hmq);
            WinTerminate(hab);
            return 1;
            }
      if (!(hwndHelp = CreateHelpWindow(hab, hwndFrame))) /* ヘルプ画面の生成 */
            {
            WinMessageBox(HWND_DESKTOP, HWND_DESKTOP,
                          "ヘルプがインストール出来ません！",
                          szClientClass, 0,
                          MB_OK | MB_ICONEXCLAMATION);
            WinDestroyWindow(hwndFrame);
            WinDestroyMsgQueue(hmq);
            WinTerminate(hab);
            return 1;
            }
      WinSetHook(hab, hmq, HK_HELP, (PFN)HelpHook, NULL); /* ヘルプフック設定 */
      while (WinGetMsg(hab, &qmsg, NULL, 0, 0))
            WinDispatchMsg(hab, &qmsg);
      WinReleaseHook(hab, hmq, HK_HELP, (PFN)HelpHook, NULL); /* ヘルプフック破棄 */
      DestroyHelpWindow(hwndHelp); /* ヘルプ画面の破棄 */
      WinDestroyWindow(hwndFrame);
      WinDestroyMsgQueue(hmq);
      WinTerminate(hab);
      return 0;
      }

MRESULT CALLBACK ClientWndProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2) {
      switch(msg) {
            case WM_HELP:
                  WinSendMsg(hwndHelp, WM_MYMESSAGE_HELP_OPEN,
                                 MPFROMSHORT(ID_HELP_INTRO), NULL);
                  return 0;
            }
```

図. 4 （つづく）

```
    return WinDefWindowProc(hwnd, msg, mp1, mp2);
    }

BOOL CALLBACK HelpHook(HAB hab, USHORT usMode, USHORT idTopic,
                       USHORT idSubTopic, PRECTL prcPosition)
    {
    switch (usMode)
            {
            case HLPM_MENU:
                    switch (idTopic)
                            {
                            case IDM_SET /* トップレベルメニューのid */:
                                    switch (idSubTopic)
                                            {
                                            case 0xFFFF:
                                            /* case サブメニューのid */
                                            case IDM_ADD_PROG:
                                            case IDM_REV_PROG:
                                            case IDM_DEL_PROG:
                                            case IDM_BACK:
                                            case IDM_MIN:
                                            case IDM_ABOUT:
                                            case SC_CLOSE:
                                                    WinSendMsg(hwndHelp,
                    WM_MYMESSAGE_HELP_OPEN, MPFROMSHORT(idSubTopic), NULL);
                                                    return TRUE;
                                            }
                            }
                    break;
            case HLPM_WINDOW:
                    switch (idTopic)
                            {
                            case MB_ID_DELETEPROGRAM /* メッセージボックスのid */:
                                    WinSendMsg(hwndHelp, WM_MYMESSAGE_HELP_OPEN,
                                            MPFROMSHORT(idTopic), NULL);
                                    return TRUE;
                            }
            }
    return FALSE;
    }
```

図. 4　方法１の TEMPLATE.C

とする。

残っているのはヘルプを表示するウィンドウ（hwndHelp が意味するもの）を作成することである。このウィンドウの作成と破壊は図. 4 の

hwndHelp = CreateHelpWindow(hab, hwndFrame) ;

DestroyHelpWindow (hwndHelp) ;

である。これらの実際の内容とこのウィンドウの動作を次に述べる。

```
#include "MYHELP.H"

MRESULT CALLBACK HelpWndProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2);
MRESULT CALLBACK HelpTextWndProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2);
HWND CreateHelpWindow(HAB hab, HWND hwnd);
BOOL DestroyHelpWindow(HWND hwnd);

typedef struct
      {
      USHORT  id;
      PCHAR   Title;
      } HELPITEM;

HWND CreateHelpWindow(HAB hab, HWND hwnd) {
      habSaved = hab;
      WinRegisterClass(hab, szHelpClass, HelpWndProc, CS_SIZEREDRAW, sizeof(HWND));
      WinCreateStdWindow(HWND_DESKTOP, 0L,
                              &flHelpFrameFlags, szHelpClass,
                              NULL, WS_CLIPCHILDREN, (HMODULE)NULL,
                              ID_HELP_RESOURCE, &hwndHelp);
      WinSetWindowPtr(hwndHelp, 0, hwnd);
      return hwndHelp;
      }

BOOL DestroyHelpWindow(HWND hwnd){
      return WinDestroyWindow(WinQueryWindow(hwnd, QW_PARENT, FALSE));
      }

MRESULT CALLBACK HelpTextWndProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2) {
      switch (msg) {
              case WM_CREATE:
                      /* 水平及び垂直スクロールバーの作成 */
                      return 0;
              case WM_SIZE:
                      /* スクロールバーの位置と大きさの変更 */
                      /* スライダーの位置決め */
                      /* スクロールバー表示非表示の変更 */
                      return 0;
              case WM_HSCROLL:
                      /* 水平スクロールバーからのメッセージの処理 */
                      /* 表示画面のスクロール等 */
                      return 0;
              case WM_VSCROLL:
                      /* 垂直スクロールバーからのメッセージの処理 */
                      /* 表示画面のスクロール等 */
                      return 0;
              case WM_MYMESSAGE_HELP_OPEN:
                      idHelp = (SHORT1FROMMP(mp1));   /* ヘルプのID */
                      if (DosGetResource((HMODULE)NULL,
                                              IDT_TEXT, idHelp, &selResource))
```

図. 5 （つづく）

```
                        {
                        idHelp = ID_HELP_INTRO;
                        DosGetResource((HMODULE)NULL, IDT_TEXT,
                                        ID_HELP_INTRO, &selResource);
                        }
                /* 読み込んだヘルプテキストの行数と最大桁数を計算 */
                /* 必要な画面の大きさを求めスクロールバーの表示非表示 */
                /* スライダーの位置の決定 */
                /* クライアントウィンドウの描画要求 */
                return 0;
        case WM_CHAR:
                /* 矢印キー入力の処理 */
                break;
        case WM_PAINT:
                /* クライアントウィンドウへヘルプテキストを表示する */
                return 0;
                }
        case WM_DESTROY:
                /* 後処理 */
                return 0;
        }
    return WinDefWindowProc(hwnd, msg, mp1, mp2);
    }

MRESULT CALLBACK HelpWndProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2) {
    switch(msg) {
        case WM_CREATE:
                /* ヘルプテキストを表示するウィンドウを作成 */
                /* ヘルプ索引を表示するリストボックスを作成 */
                /* この２つは重なっておりどちらか一方のみを表示する */
                /* リストボックスに索引を記入する */
                return 0;
                }
        case WM_MYMESSAGE_HELP_OPEN: /* ヘルプの要求 */
                /* ヘルプを終了するときにアクティブにするウィンドウを調査 */
                /* ヘルプ画面を可視状態にする */
                if (SHORT1FROMMP(mp1) == ID_HELP_INDEX) /* ヘルプの索引 */
                        {
                        /* ヘルプ索引を表示するリストボックスを */
                        /* ヘルプテキストを表示するウィンドウの前面に移動 */
                        /* リストボックスにキーボードフォーカスを設定する */
                        return 0;
                        }
                else /* SHORT1FROMMP(mp1) はヘルプID */
                        {
                        /* ヘルプテキストを表示するウィンドウを */
                        /* ヘルプ索引を表示するリストボックスの前面に移動 */
                        /* テキスト表示画面にキーボードフォーカスを設定し */
                        /* 同じメッセージを送付 */
                        return 0;
                        }
```

図. 5 （つづく）

```
        case WM_SIZE:
                /* ヘルプテキストを表示するウィンドウ位置と大きさを変更 */
                /* ヘルプ索引を表示するリストボックス位置と大きさを変更 */
                return 0;
        case WM_CONTROL:
                /* リストボックスの選択された索引項目を調べ */
                /* 対応するヘルプテキストを表示させる。 */
                break;
        case WM_COMMAND:
                switch (COMMANDMSG(&msg)->cmd)
                        {
                        case IDM_HELP_EXIT:
                                WinSendMsg(hwnd, WM_CLOSE, NULL, NULL);
                                return 0;
                        case IDM_HELP_INDEX:
                                WinSendMsg(hwnd, WM_MYMESSAGE_HELP_OPEN,
                                        MPFROMSHORT(ID_HELP_INDEX), NULL);
                                return 0;
                        break;
                        }
                break;
        case WM_CLOSE:
                /* ウィンドウを不可視状態にし */
                /* ヘルプが要求される前にアクティブであったウィンドウを */
                /* アクティブにする。実際には終了しない */
                return 0;
        }
    return WinDefWindowProc(hwnd, msg, mp1, mp2);
    }
```

図. 5　方法１の MYHELP.C

```
/* ヘルプをインストールする場合以下をリソース定義ファイルに追加する */
#include "HELPWND.H"

/* ヘルプの内容 */
/* 各ヘルプの内容をファイルに書き、そのヘルプを起動するときの */
/* WinSendMsg(hwndHelp, WM_MYMESSAGE_HELP_OPEN , MPFROMSHORT(id), NULL); */
/* のidとそのファイル名を以下のように */
/* 「RESOURCE IDT_TEXT」の後にスペースで区切って並べる */
/* アプリケーションの概要は必須で、そのidはID_HELP_INTROとする */
/* ヘルプの索引はファイル「HELPWND.H」で記述する */

RESOURCE IDT_TEXT ID_HELP_INTRO helptmp0.asc          /* この行は必要 */
```

図. 6　方法１の TEMPLATE.RC

図. 8で＜f・1＞キーを押すかメニュー項目「F 1＝ヘルプ」をマウスでクリックすると図. 9のようなヘルプウィンドウが現れる。クライアントウィンドウにはヘルプテキストが表示されている。メニュー項目は「終了」と「索引」があり，「終了」を選択すると，ヘルプウィンドウが消え，「索引」を選択すると，図. 10のようなリストボックスがヘルプテキストにかわってクライアントウィンドウに現れる。リストボックスで索引項目を選択すると，リストボックスにかわって選択された項目に対するヘルプテキストがクライアントウィンドウに表示される。リストボックスに入れる索引項目とヘルプテキストはユーザーが準備する必要がある。このように簡単なものであるが，それを実現するのが図. 5のプログラムである。

先ず，関数 CreateHelpWnd で図. 9のようなフレームウィンドウを作成する。そのクライアントウィンドウのウィンドウプロシージャ HelpWndProc は自分と同じ大きさのウィンドウを2個作成する。1つはウィンドウテキストを表示するウィンドウプロシージャ HelpTextWndProc を持つウィンドウであり，もう1つは索引を表示するリストボックスである。HelpWndProc の主な仕事はヘルプの要求メッセージ WM__MYMESSAGE__HELP__OPEN をその idHelp の値に応じて自分の子ウィンドウに振り分けることである。索引要求の場合はリストボックスをヘルプテキストを表示するウィンドウの前面へ移動させる。それ以外のヘルプ要求の場合はヘルプテキストを表示するウィンドウをリストボックスの前面へ移動させ，自分に送られてきたメッセージをそのまま HelpTextWndProc へ送付し，テキストの表示を要求する。また，HelpWndProc への終了要求（システムメニューボックスの終了を選択した場合などに発生する）はヘルプウィンドウを非表示とするだけで実際には終了させない。

次は実際にヘルプテキストを表示する HelpTextWndProc の動作を見てみる。第1の問題点はユーザーが作成するヘルプテキストをどのように管理するかである。ここではユーザー定義のリソースとして扱うことにした。先ず，1つのヘルプ毎に1つのファイルを準備し，そのファイルにヘルプテキストを書

```
/* 配列 aHelpItem の内容以外変更不可 */

#define       ID_HELP_RESOURCE        1000
#define       IDM_HELP_EXIT           1050
#define       IDM_HELP_INDEX          1051

#define       IDS_HELP_TITLE          1060
#define       IDT_TEXT                2048

#define       WM_MYMESSAGE_HELP_OPEN  (WM_USER + 1000)
#define       ID_HELP_INTRO           4000
#define       ID_HELP_INDEX           4001

HWND CreateHelpWindow(HAB, HWND);
BOOL DestroyHelpWindow(HWND);

typedef struct
      {
      USHORT  id;
      PCHAR   Title;
      } HELPITEM;


/* ヘルプの索引を作成する場合は */
/* 配列の内容を変更すること */

HELPITEM aHelpItem[] =
      {
      /*「id, "索引の項目名",」の様に記述すると、 */
      /* 索引で「索引の項目名」を選択するとリソース定義ファイル(*.rc)で */
      /* RESOURCE IDT_TEXT id file.nam */
      /* と定義されているファイル file.nam を表示する */
      /* 例えば、「ID_HELP_INTRO, "ヘルプ"」と記述する */

      IDM_ADD_PROG, "プログラムの追加",
      IDM_REV_PROG, "プログラムの変更",
      IDM_DEL_PROG, "プログラムの削除",
      IDM_BACK, "バックグラウンドで実行",
      IDM_MIN, "実行時最小化",
      ID_HELP_INTRO, "ヘルプ",
      NULL                                    /* 最後はNULLで終わること */
      };
```

図. 7　方法１の HELPWND.H

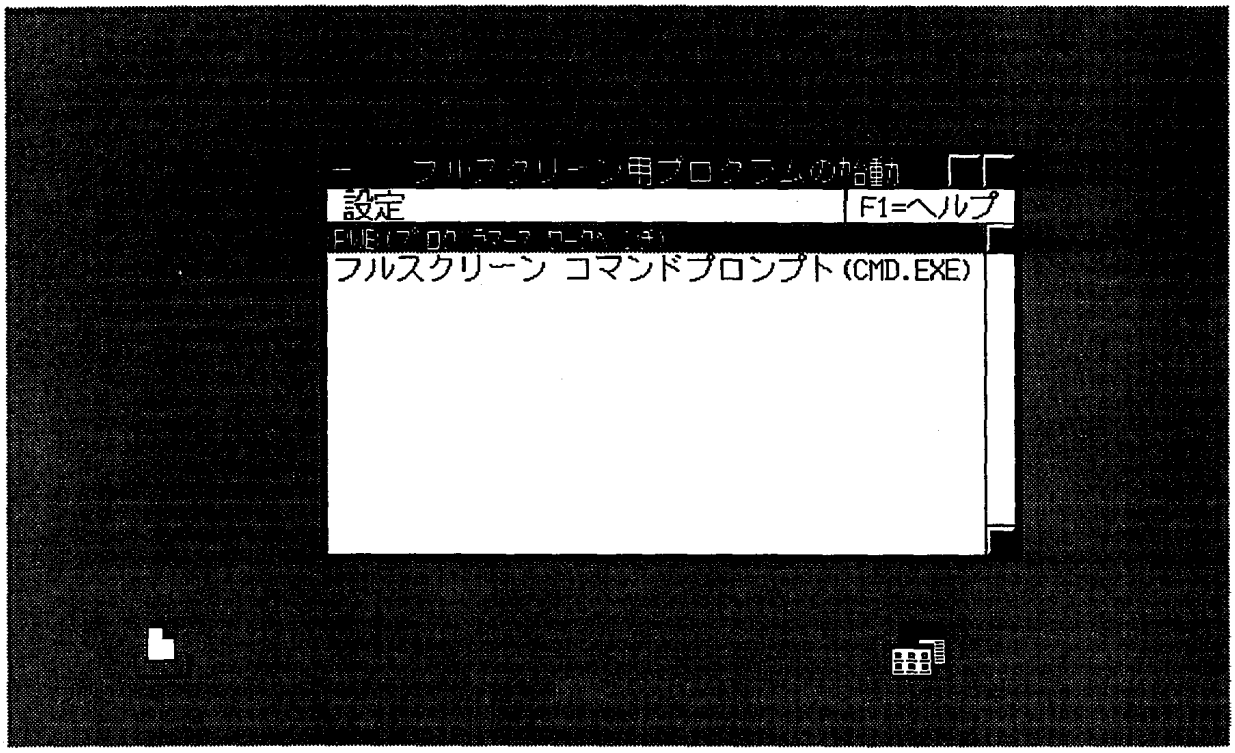

図. 8　メインウィンドウ（「方法１」）

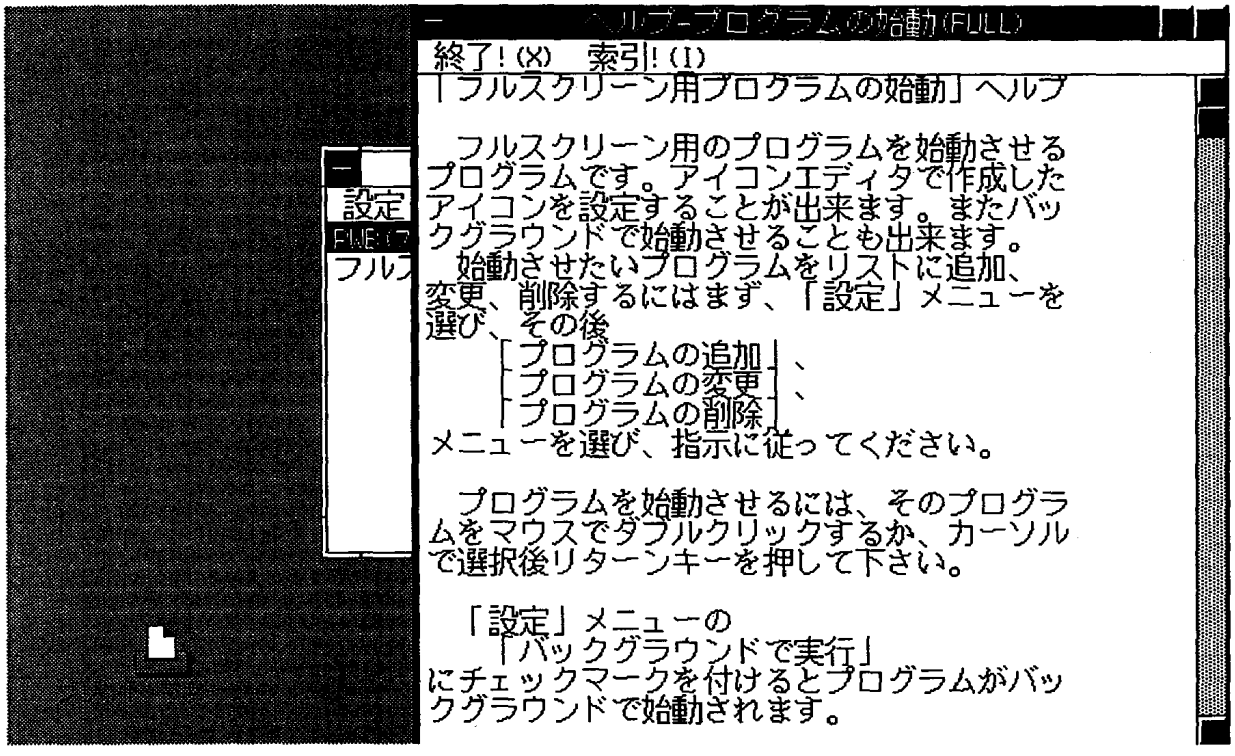

図. 9　ヘルプウィンドウ（「方法１」）

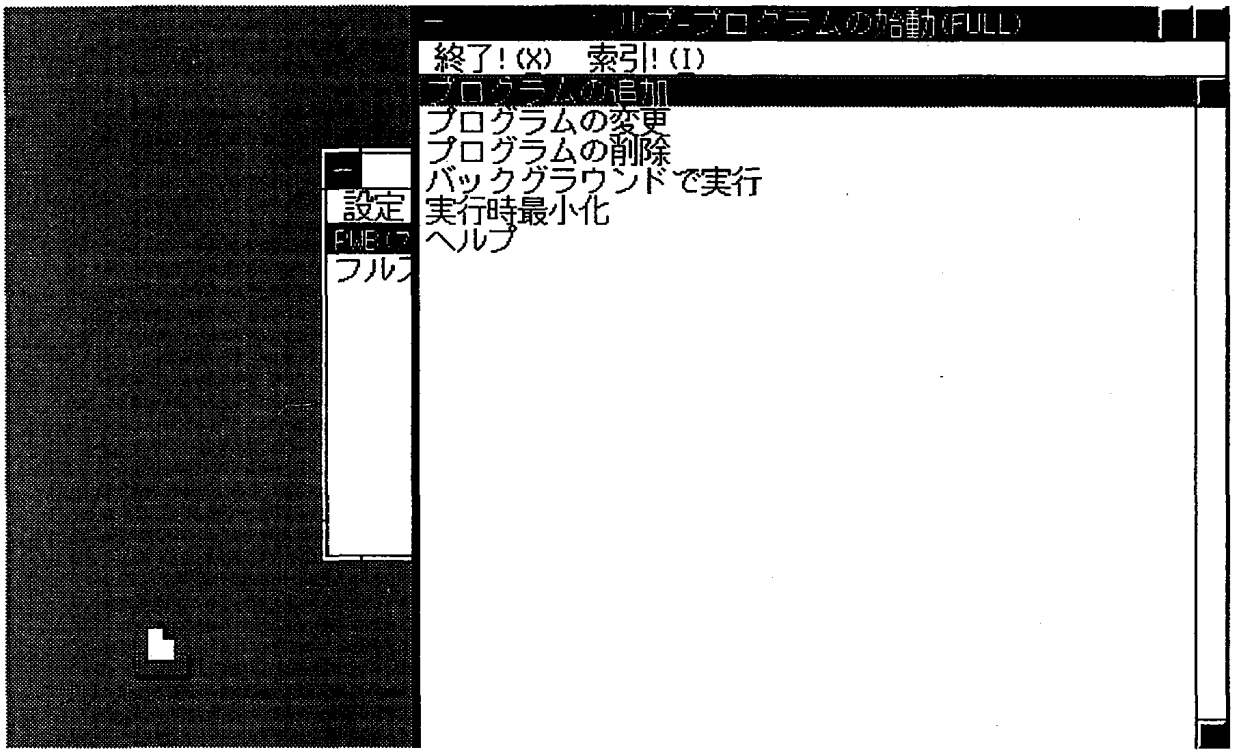

図.10　ヘルプウィンドウ（索引）（「方法１」）

き込む。これは通常のアスキーファイルである。これらをリソース定義ファイルにユーザー定義のリソースとして定義しておく。その際，識別子としてメニュー項目のヘルプにはその項目の識別子を，メッセージボックスのヘルプにはメッセージボックスのウィンドウの識別子を，その他のウィンドウのヘルプには一意となるように適当な識別子を付けておく。この識別子がメッセージ WM__MYMESSAGE__HELP__OPEN と共に送られてくる idHelp である。HelpTextWndProc は idHelp の値のユーザー定義のリソースを

DosGetResource((HMODULE)NULL, IDT__TEXT, idHelp, &selResource);

で読み込んで，表示すればよい。しかしここでまた問題が生じる。表示するウィンドウの大きさは変化するし，表示すべきヘルプテキストの行数及び桁数も変化する。ここではユーザーがファイルにヘルプテキストを書いたそのままを表示することにした。すなわち，改行マークまでを１行としたのである。そのためウィンドウの大きさが小さすぎてヘルプテキストが入りきらないときには，

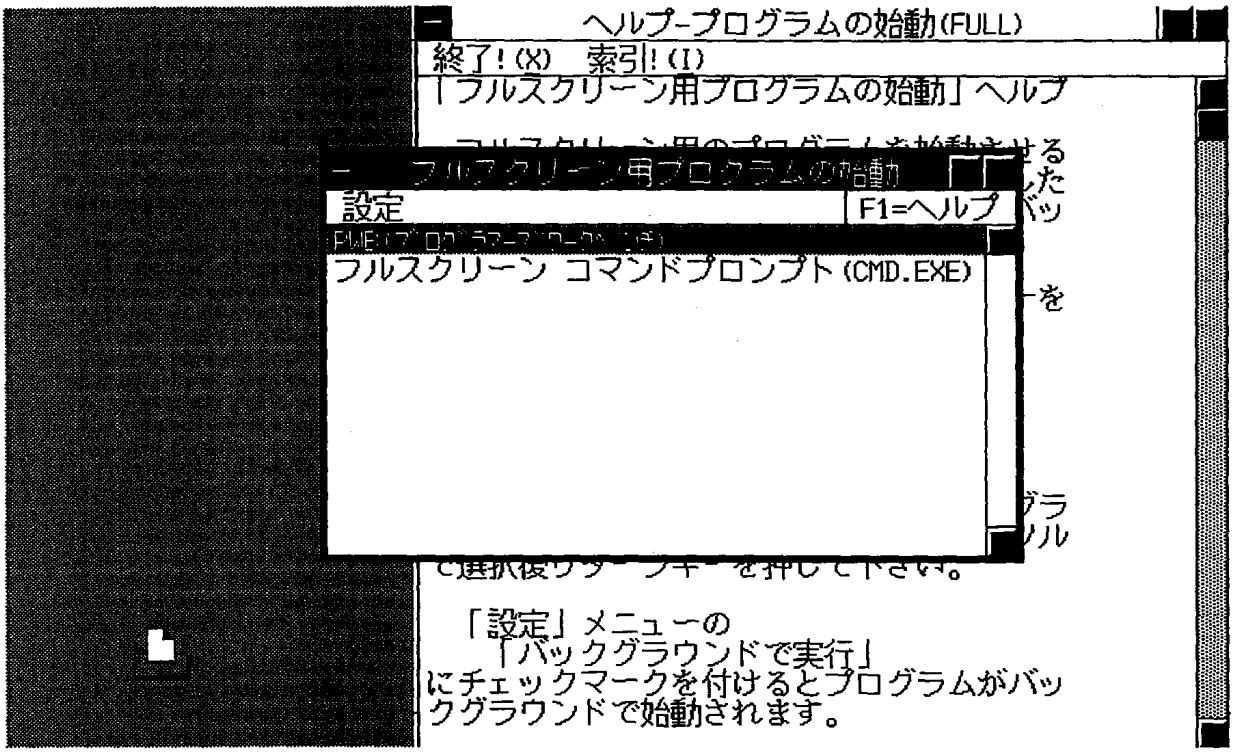

図.11 バックグラウンドへ移動したヘルプウィンドウ（「方法１」）

水平垂直スクロールバーを表示し対処した。図. 6及び7でヘルプテキストが書いてあるファイルをユーザー定義のリソースとして定義する方法，及び，索引項目の定義の仕方を例示した。

以上が「方法１」の大体の概略である。図. 9のヘルプウィンドウをバックグラウンドへ持って行ったのが図. 11である。私にはこのようにヘルプ画面が後ろに隠れるのが便利なように思われる。「方法１」では普通にヘルプ画面のフレームウィンドウを作ったのだから，ヘルプ画面が後ろに隠れない「方法２」ではそうしない何か特別な理由があるのかも知れない。

「方法２」

次にVer. 1. 21で利用できるようになったヘルプマネージャの最小限の機能を利用する方法を述べる。ヘルプマネージャもヘルプフックをその内部で利用しているので，ヘルプマネージャを利用する場合はWM_HELPメッセージを処理する時は処理後に必ずデフォルトのウィンドウプロシージャ Win-

DefWindowProc を呼び出す，等の注意が必要である。ヘルプマネージャを利用するにはヘルプインスタンスの作成と破棄，ヘルプインスタンスへのヘルプ表示要求，ヘルプインスタンスからの要求への返答，の手続きが必要であり，その必要最小限の部分をカプセル化したのが図. 12の関数，図. 13のマクロである。これらを実際に利用しているのが図. 1 の main 関数及び図. 3 のウィンドウプロシージャ ClientWndProc である。main 関数ではヘルプインスタンスの作成と破棄のところ，ClientWndProc では CASE__REPLY__HELP__AAB__MSG と CASE__REPLY__HM__MSG のところで利用している。それぞれのヘルプテキストのことはヘルプパネルと呼ばれているが，残りは，実際のヘルプパネルの内容とヘルプが要求された時どのヘルプパネルを表示するかの対応付けである。後者の対応付けは図. 14のようにリソース定義ファイルの HELPTABLE 文，HELPITEM 文，HELPSUBTABLE 文，HELPSUBITEM 文で行う。ヘルプを表示したいのはメニュー，クライアントウィンドウの子ウィンドウであるリストボックス，ダイアログボックス，メッセージボックスである。先ず，HELPTABLE 文中の HELPITEM 文にこのことを指定する。例えば，

HELPITEM ID__RESOURCE, SUBTBL__MAIN__MENU, GENERAL__HP

はメニューとリストボックスでヘルプの要求があった場合は識別子 SUBTBL__MAIN__MENU の HELPSUBTABLE を参照し，拡張ヘルプパネルとしては GENERAL__HP を利用することを表している。また，

HELPITEM MB__ID__DELETEPROGRAM, SUBTBL__DEL__MBOX, DELETE__HP

は MB__ID__DELETEPROGRAM のメッセージボックスでヘルプの要求があった場合は識別子 SUBTBL__DEL__MBOX の HELPSUBTABLE を参照し，拡張ヘルプパネルとしては DELETE__HP を利用することを表している。この拡張ヘルプパネルはヘルプを要求した子ウィンドウのヘルプパネルが見つからない場合に表示されるヘルプパネルである。次に HELP-

```
#define INCL_WIN
#include <os2.h>
#include "MYHELP.H"             /* ヘルプ用ヘッダファイル */
#include "MYHP.H"               /* ヘルプパネル識別子のヘッダファイル */

HWND hwndHelpInstance = NULL;

/* 標準的なヘルプインスタンスを作成しフレームウィンドウに関連付ける */
BOOL InitHelp(HAB hab, HWND hwndFrame) {
      HELPINIT stHMInit;
      stHMInit.cb                       = sizeof(HELPINIT);
      stHMInit.ulReturnCode             = 0L;
      stHMInit.pszTutorialName          = NULL;
      stHMInit.phtHelpTable             = MAKELONG(MY_HELP_TABLE,0xffff);
      stHMInit.hmodHelpTableModule      = (HMODULE)NULL;
      stHMInit.hmodAccelActionBarModule = (HMODULE)NULL;
      stHMInit.idAccelTable             = 0;
      stHMInit.idActionBar              = 0;
      stHMInit.pszHelpWindowTitle       = MY_HELP_WINDOW_TITLE;
      stHMInit.usShowPanelId            = CMIC_HIDE_PANEL_ID;
      stHMInit.pszHelpLibraryName       = MY_HELP_LIBRARY;
      if (!(hwndHelpInstance = WinCreateHelpInstance(hab, &stHMInit)))
            {
            return  FALSE;
            }
      if (!WinAssociateHelpInstance(hwndHelpInstance, hwndFrame))
            {
            return FALSE;
            }
      }

/* ヘルプインスタンスへのメッセージの送付(ヘルプメニューの処理時に利用) */
VOID SendHelpMsg(USHORT usMsg) {
      WinSendMsg(hwndHelpInstance, usMsg, 0, 0);
      }

/* ヘルプインスタンスを破棄するときの後処理 */
VOID TerminateHelp(HWND hwnd) {
      if (hwndHelpInstance)
            {
            WinAssociateHelpInstance(NULL, hwnd);
            WinDestroyHelpInstance(hwndHelpInstance);
            }
      }
```

図.12 方法2のMYHELP.C

```
/* 下のタイトル及びライブラリ名を適当に修正すること */

#define MY_HELP_WINDOW_TITLE "STARTFUL:ヘルプ"  /* ヘルプウィンドウのタイトル */
#define MY_HELP_LIBRARY "MYHELP.HLP"            /* ヘルプライブラリ名 */

#define IDM_ABOUT                   4900
#define IDM_HELP                    4901
#define IDM_DISPLAY_HELP            4902
#define IDM_EXT_HELP                4903
#define IDM_KEYS_HELP               4904
#define IDM_INDEX_HELP              4905

/* ヘルプメニューのマクロ（リソース定義ファイルで利用） */
#define HELP_SUBMENU                                                    ¥
      SUBMENU "ヘルプ(¥036H¥037H)", IDM_HELP                            ¥
              {                                                         ¥
              MENUITEM "ヘルプのヘルプ(¥036H¥037^)...",     IDM_DISPLAY_HELP ¥
              MENUITEM "拡張ヘルプ(¥036E¥037A)...",         IDM_EXT_HELP     ¥
              MENUITEM "キーのヘルプ(¥036K¥037K)...",       IDM_KEYS_HELP    ¥
              MENUITEM "ヘルプ索引(¥036I¥037I)...",         IDM_INDEX_HELP   ¥
              MENUITEM SEPARATOR                                        ¥
              MENUITEM "プロフィール(¥036B¥037O)...",       IDM_ABOUT        ¥
              }

/* 上のヘルプメニューを処理するマクロ */
/* ウィンドウプロシージャのWM_COMMANDを処理する部分で利用 */
#define  CASE_REPLY_HELP_AAB_MSG                ¥
      case IDM_DISPLAY_HELP:                    ¥
              SendHelpMsg(HM_DISPLAY_HELP);     ¥
              return 0;                         ¥
      case IDM_EXT_HELP:                        ¥
              SendHelpMsg(HM_EXT_HELP);         ¥
              return 0;                         ¥
      case IDM_INDEX_HELP:                      ¥
              SendHelpMsg(HM_HELP_INDEX);       ¥
              return 0;                         ¥
      case IDM_KEYS_HELP:                       ¥
              SendHelpMsg(HM_KEYS_HELP);        ¥
              return 0;

/* ヘルプマネージャからのキーのヘルプのヘルプパネルの要求に答えるマクロ */
#define CASE_REPLY_HM_MSG                       ¥
      case HM_QUERY_KEYS_HELP:                  ¥
              return (MRESULT)KEYS_HELP_HP;

BOOL    InitHelp(HAB, HWND);
VOID    TerminateHelp(HWND);
VOID    SendHelpMsg(USHORT);
```

図.13　方法２の MYHELP.H

```
/* このファイルにヘルプテーブル等を定義する */

#include "MYHP.P"

HELPSUBTABLE SUBTBL_MAIN_MENU
      {
      HELPSUBITEM             IDM_SET,                SET_AAB_HP
      HELPSUBITEM                     IDM_ADD_PROG,   ADD_PROG_HP
      HELPSUBITEM                     IDM_REV_PROG,   REV_PROG_HP
      HELPSUBITEM                     IDM_DEL_PROG,   DEL_PROG_HP
      HELPSUBITEM                     IDM_BACK,       BACK_HP
      HELPSUBITEM                     IDM_MIN,        MIN_HP
      HELPSUBITEM                     SC_CLOSE,       EXIT_HP
      HELPSUBITEM             IDM_HELP,               HELP_AAB_HP
      HELPSUBITEM                     IDM_DISPLAY_HELP, HELPONHELP_MENU_HP
      HELPSUBITEM                     IDM_EXT_HELP,   EXTHELP_MENU_HP
      HELPSUBITEM                     IDM_KEYS_HELP,  KEYS_MENU_HP
      HELPSUBITEM                     IDM_INDEX_HELP, INDEX_MENU_HP
      HELPSUBITEM                     IDM_ABOUT,      ABOUT_HP
      HELPSUBITEM                     IDL_MYLIST,     GENERAL_HP
 }

HELPSUBTABLE  SUBTBL_DEL_MBOX
      {
  HELPSUBITEM                 -1, -1
      }

HELPTABLE MY_HELP_TABLE
      {
      HELPITEM        ID_RESOURCE,    SUBTBL_MAIN_MENU,       GENERAL_HP
      HELPITEM        IDD_ABOUT,      SUBTBL_ABOUT_DIALOG,    ABOUT_HP
      HELPITEM        IDD_ADD_PROG,   SUBTBL_ADD_PROG_DIALOG, ADD_PROG_DIALOG_HP
      HELPITEM        IDD_REV_PROG,   SUBTBL_REV_PROG_DIALOG, REV_PROG_DIALOG_HP
      HELPITEM        IDD_END,        SUBTBL_END_DIALOG,      END_HP
      HELPITEM        MB_ID_DELETEPROGRAM,    SUBTBL_DEL_MBOX,DELETE_HP
      }
```

図.14　方法2のMYHELP.RC

SUBTABLE文のHELPSUBITEM文で最終的な対応関係を与える。例えば，上記の識別子SUBTBL__MAIN__MENUのHELPSUBTABLE文中の

HELPSUBITEM IDM__ABOUT, ABOUT__HP

は，IDM__ABOUTで表示されているメニュー項目「プロフィール」にカーソルがあった時に＜f・1＞キーが押されヘルプ要求があった場合にヘルプパ

ネルABOUT__HPを表示せよ，という意味である。また，

HELPSUBITEM IDL__MYLIST, GENERAL__HP

は，図.16の時に＜f・1＞キーが押されヘルプ要求があった場合にヘルプパネルGENERAL__HPを表示せよ，という意味である。結果は図.17のようになる。また，MB__ID__DELETEPROGRAMのメッセージボックスでヘルプ要求があった場合，

HELPSUBITEM −1, −1

を参照して，ヘルプを要求した子ウィンドウのヘルプパネルが見つからないので拡張ヘルプパネルDELETE__HPが表示される。ここには示されていない

```
#include "MYHP.H"
:userdoc.
:body.
:h1 res=GENERAL_HP.フルスクリーン用プログラムの始動ヘルプ
:p.フルスクリーン用のプログラムを始動させる
プログラムです。アイコンエディタで作成した
アイコンを設定することが出来ます。またバッ
クグラウンドで始動させることも出来ます。
:p.始動させたいプログラムをリストに追加、
変更、削除するにはまず、「設定」メニューを
選び、その後
:p.　　「プログラムの追加」、
:p.　　「プログラムの変更」、
:p.　　「プログラムの削除」
メニューを選び、指示に従ってください。
:p.プログラムを始動させるには、そのプログラ
ムをマウスでダブルクリックするか、カーソル
で選択後リターンキーを押して下さい。
:p.「設定」メニューの
　　「バックグラウンドで実行」
にチェックマークを付けるとプログラムがバッ
クグラウンドで始動されます。
:p.「設定」メニューの
　　「実行時最小化」
にチェックマークを付けるとプログラムを始動
後、この「フルスクリーン用プログラムの始動」
自身がアイコンに最小化されます。ただし、
プログラム終了後には復元されません。
:eh1.
:euserdoc.
```

図.15　方法2のMYHELPO.ITL

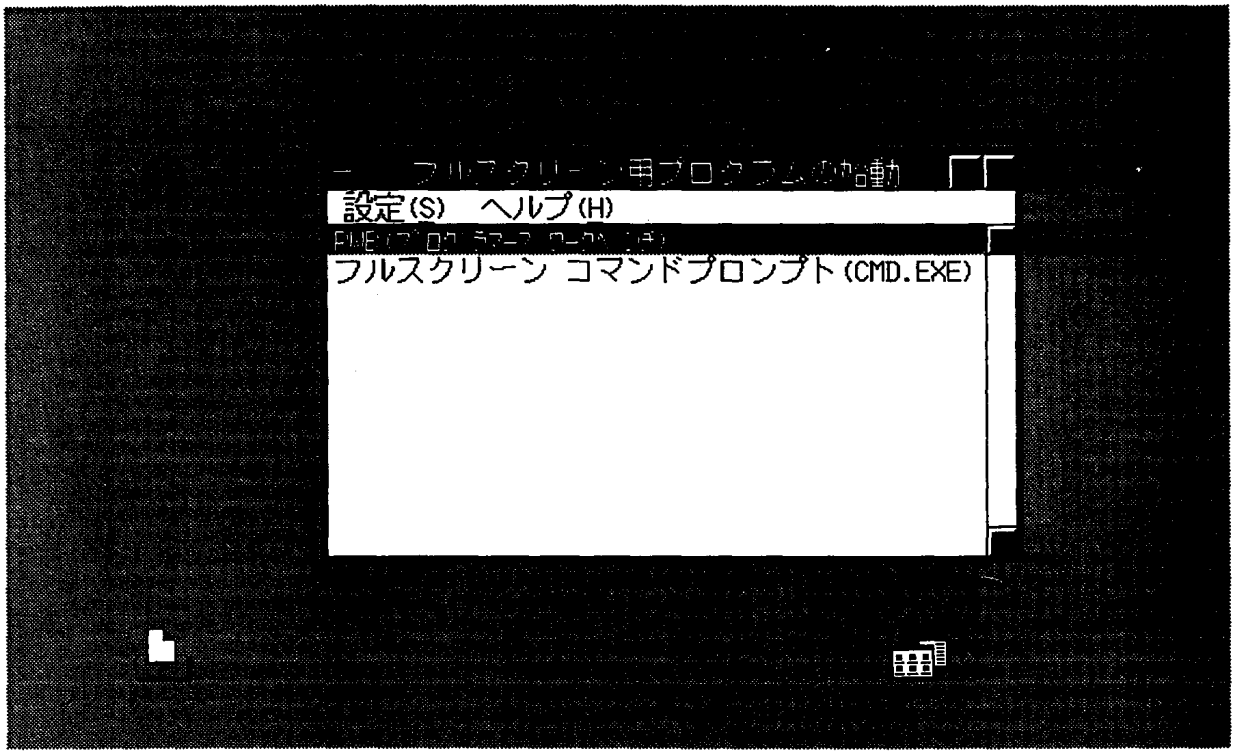

図.16 メインウィンドウ（「方法2」）

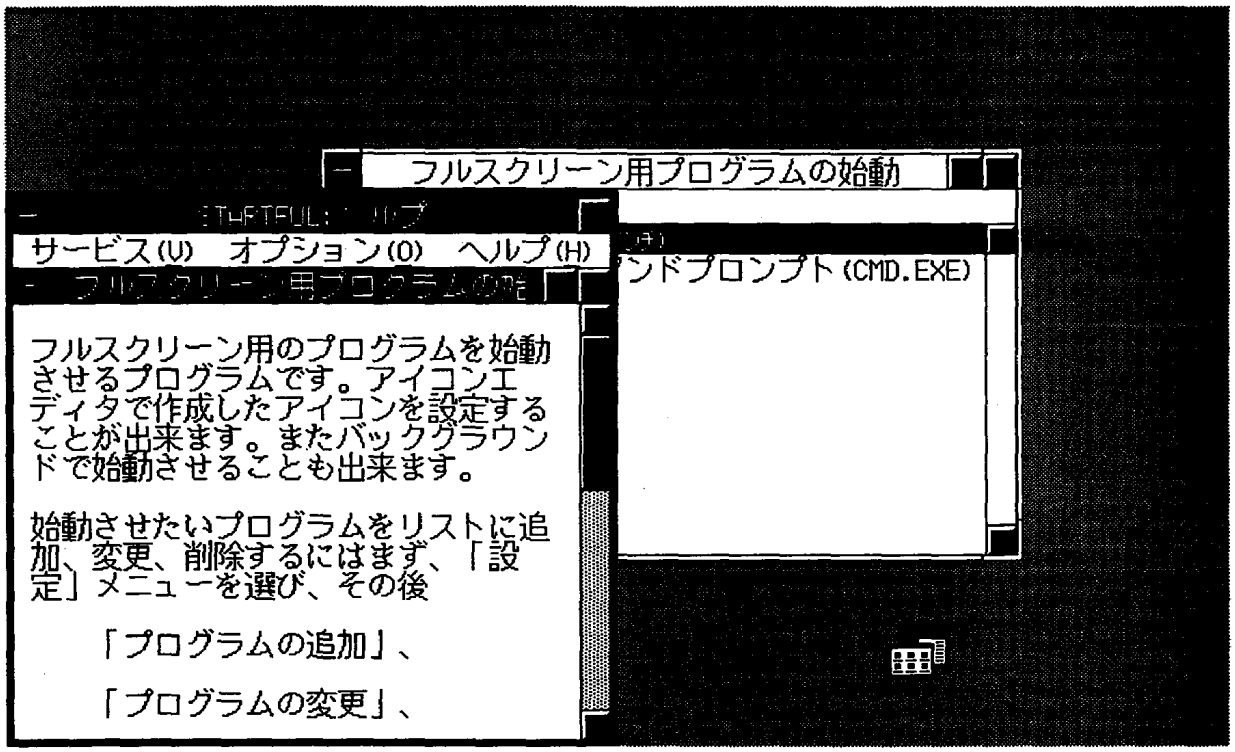

図.17 ヘルプウィンドウ（「方法2」）

が，ダイアログボックス中のたくさんのコントロールに対しても HELP-SUBTABLE 文できちんとそれぞれのヘルプパネルに対応させることにより，簡単に全てのコントロールのヘルプが実現できる。

最後は，実際のヘルプ内容，すなわち，ヘルプパネルの準備の仕方である。図.15のようなファイルを準備する。「：userdoc.」のように「：」で始まるものはブックマスタ　タグと呼ばれる書式を制御するものである。3行目の「：body.」と最後の行の「：euserdoc.」の間に準備する必要がある全てのヘルプパネルを入れる。1つのヘルプパネルは「：h1」で始まり「：eh1.」で終わり，「：h1」の次にある「res = GENERAL_HP. フルスクリーン用プログラムの始動ヘルプ」でこのヘルプパネルが GENERAL_HP であること，表示タイトルが「フルスクリーン用プログラムの始動ヘルプ」であることを示している。ここでは詳しく述べないが，文字の色を変えたり，強調表示させたり，ハイパーテキスト化させたり，等を行うタグがある。このヘルプソースファイルを情報表示コンパイラ（IPFC）にかけることによりヘルプマネージャが利用できる MYHELP.HLP の作成が完了する。このファイルをカレントディレクトリか環境変数 HELP で指定されているディレクトリへ入れておけばよい。

「方法1」は自分でヘルプ機能を実現しようとしたものでありまだ不完全であるので，今後は「方法2」によりヘルプを備えることになる。その際ヘルプパネルの表現の統一，読み易さ，理解のし易さをいかに実現していくかが課題となる。

4．おわりに

MS OS／2 のプレゼンテーションマネージャ（PM）のプログラミングにおいてマルチスレッドとヘルプをいかに実現するかについて例を用いて概要の説明を行った。マルチスレッド化はユーザーの入力を迅速に処理するためには大変利用価値のあるものであり，プログラミングは困難になるが，その労に報い

られるほどの効果があると思われる。ヘルプに関してはヘルプマネージャの利用によりヘルプパネルと対応表を準備するだけでよいようになった。しかしながらヘルプパネルの内容に関しては十分に検討を重ねる必要があると思われる。なぜなら，実際にヘルプを利用した時，読もうとする気が起こらないような内容が多いからである。

参 考 文 献

〔1〕中西隆著「OS/2標準テクノバイブル〔基本編〕」技術評論社 1988
〔2〕Petzold, C. “Programming The OS/2 Presentation Manager”, Microsoft Press, 1989.
〔3〕「日本語 MS OS/2 (Ver 1.21) ソフトウェア開発支援ツール」NEC