Windows アプリケーションプログラミング

生命･情報等教育研究支援室(電子･情報工学系)　中山　勝

０．はじめに

私が携わっている情報学類主専攻実験のテーマのひとつに、プロセッサの設計開発というものがある。この実験テーマは、プログラム可能な論理素子 FPGA(Field Programmable Gate Array)を用いて行われる。この FPGA 上に MIPS プロセッサやシリアルインターフェース（RS-232C）制御回路などを構成し、Windows コンピュータとシリアル通信させる。

今回、その Windows コンピュータ上で動作し、MIPS プロセッサの制御及び内部データや制御信号の流れをモニタするためのアプリケーション MIPS プロセッサコントロールパネル（MPCP）を開発したので、その MPCP を中心に Windows アプリケーションプログラミングの基礎について述べる。(開発環境は Microsoft Visual C++ 6.0)

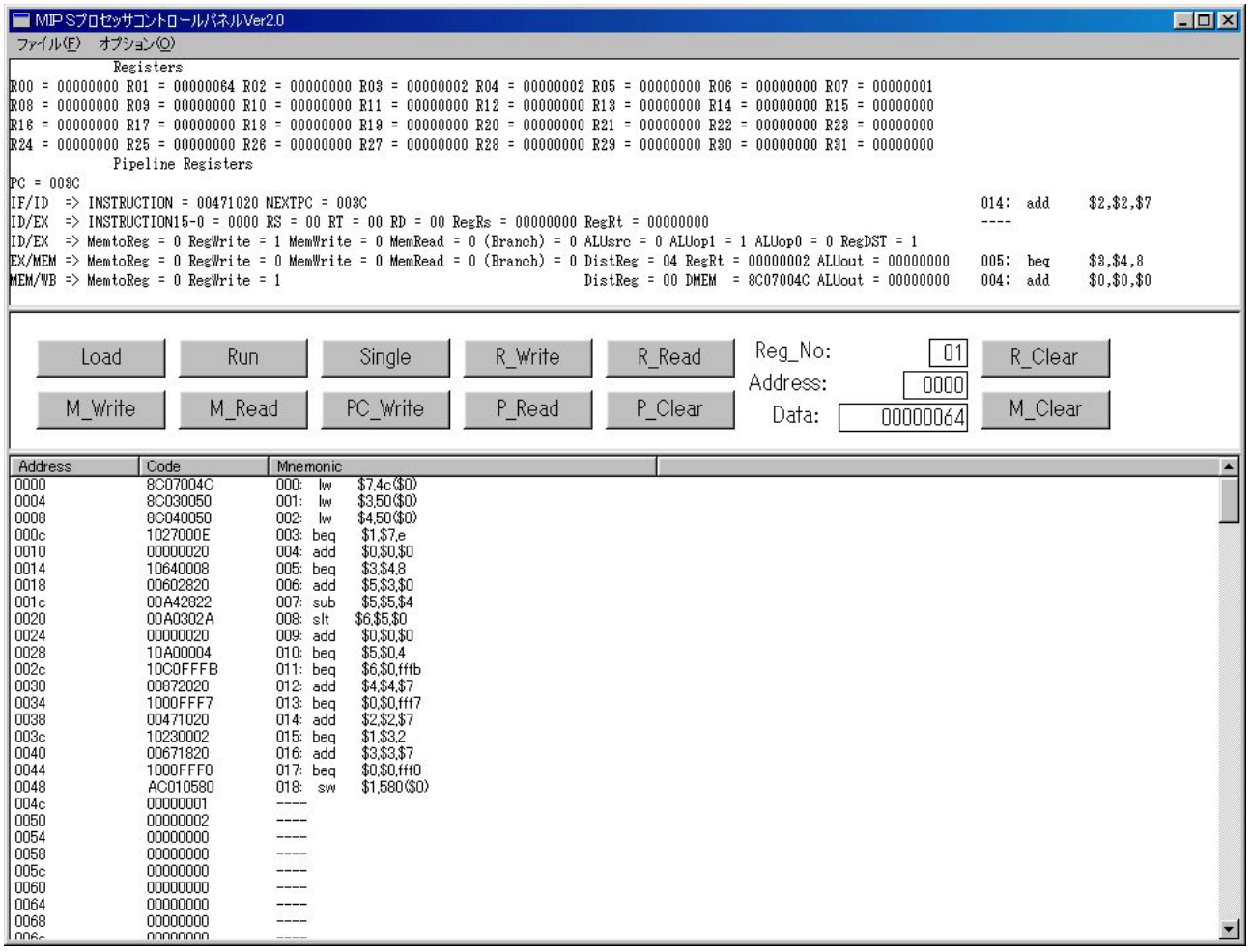

図１．MPCP のウィンドウ

１．Windows アプリケーションの基本構成

WindowsOS は、GUI(Graphical User Interface)を持っている。そのため Windows アプリケーションでは、Win32API が提供する様々なボタン、エディットボックスなどコントロールと呼ばれる部品を使ってデザインすることになる。図１は MPCP のウィンドウである。また、図２に

はアプリケーションの動作の流れを示す。

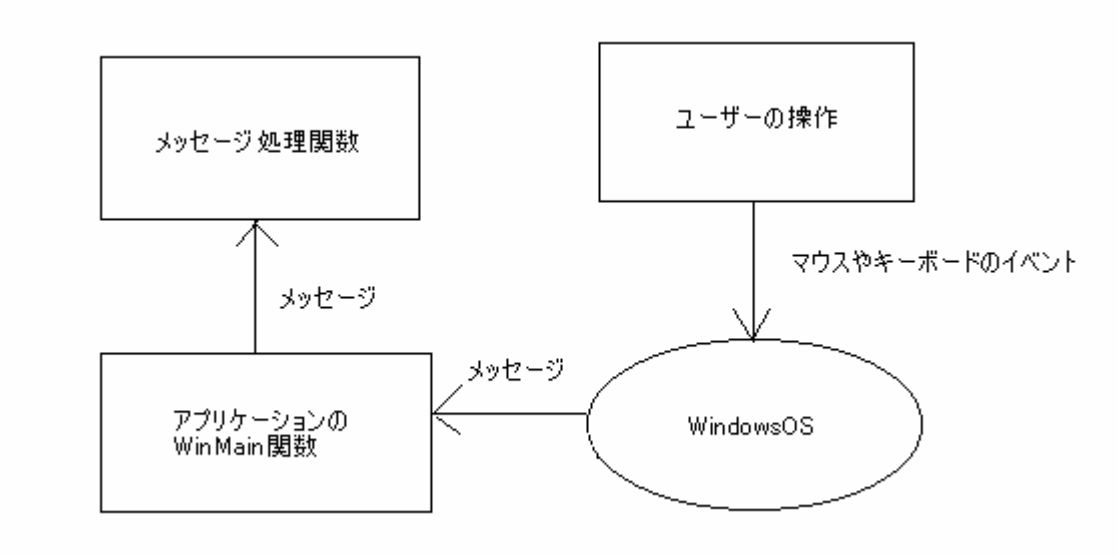

図２．アプリケーションの動作の流れ

１．１．WinMain 関数

WinMain 関数とはC 言語で言う main 関数に相当するもので通常 Windows プログラムはここから実行される。

この関数の中では、主に以下の３つの操作を行う。

- 必要な全ての初期化作業
- 送られてきたメッセージを取り出し、処理関数に渡す。
- 終了処理

表１は、WinMain 関数内のメッセージ処理ループのサンプルコードである。ここで、終了メッセージを受け取りループを抜けるまで、メッセージを取り出し、処理関数に渡すという作業が繰り返される。

```
        /*アプリケーションが終了するまでアプリケーションメッセージを処理*/
while( (bRet = GetMessage( &msg, NULL, 0, 0 )) != 0){
    if (bRet == -1){
       MessageBox(NULL,"メッセージ取得時にエラーが発生しました","エラーメッセージ",
                                                                MB_OK);
    break;
    }else{
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}
return(msg.wParam);
```

表１．メッセージ処理のサンプルコード

１．２．メッセージ処理関数

通常アプリケーションは、1 つ以上のウィンドウを持っている。メッセージ処理関数は、そのウィンドウと関連付けられ、そのウィンドウ上で発生する様々なイベントによって、OS がアプ

リケーションに送ってくるメッセージを処理する関数である。以下に MPCP で実際に処理しているメッセージを記す。

- WM_CREATE（ウィンドウが作られるときに発生）
- WM_COMMAND（メニューが選択された、ボタンが押されたりしたときに発生）
- WM_SIZE（ウィンドウのサイズが変更されたときに発生）
- WM_CLOSE（閉じるボタンが押されたときに発生）
- WM_DESTROY（ウィンドウが破棄されたときに発生）
- WM_PAINT（ウィンドウ内の表示データに、再描画の必要があるときに発生）

表２は、そのメッセージ処理関数内の一部である。

```
/*メッセージ処理*/
switch(message){
        ・
        ・
case WM_DESTROY:
    PostQuitMessage(0);/*終了メッセージの発行*/
    break;
default:
    return(DefWindowProc(hWnd,message,wParam,lParam));
}
return(0L);
```

表２．メッセージ処理関数の一部

２．アプリケーションの処理

ここでは，開発したアプリケーション MPCP を例にとり処理の流れを説明する。

２．１．アプリケーションの起動

アプリケーションのアイコンやショートカットが、ダブルクリックされると、OS がそのアプリケーションを起動する。起動されたアプリケーションは WinMain 関数内で変数などの値の初期化を行い、ウィンドウクラスの登録、メインとなるウィンドウの作成、表示を行いメッセージ処理のループに入る。ここでは他に、シリアルポートのオープンなども行っている。

次にアプリケーションは、WM_CREATE メッセージを受け取る。図１をみるとわかると思うが、MPCP では、１枚のメインウィンドウの上に３枚の子ウィンドウを貼り付けている。それぞれ上からレジスタ内容表示ウィンドウ、コントロールパネルウィンドウ、命令／データメモリ内容表示ウィンドウである。この３枚のウィンドウはこの時点で作成される。コントロールパネルウィンドウ内の、ボタンや、エディットボックスも同様である。

ここまでくると起動プロセスは終了し、デスクトップには、図１のようなウィンドウが描画され、ユーザーのウィンドウ上での操作待ちとなる。

２．２．アプリケーションの機能

はじめにも述べたが、MPCP はシリアル通信により MIPS プロセッサを制御するためのアプリケーションである。そしてその機能は、コントロールパネルウィンドウ内のボタンコントロールやエディットボックスコントロールにより実現されている。ユーザーの操作としては、コン

トロールパネル内のエディットボックスコントロールに必要な値をセットしボタンをクリックするとで、MIPS プロセッサに各々のコマンドが送信される。またそのコマンドには、その機能により返信（戻り値）が存在するものもある。以下にそのコマンドと機能を示す。(コマンドは全てアスキーキャラクターコードで表現される)

・ダウンロードコマンド

：CCAAAATTDDDD‥DDSS

：＝IntelHex フォーマットのヘッダ（0x3A)、CC＝バイトカウント、AAAA＝アドレス、TT＝レコードタイプ、DD＝データ、SS＝チェックサム、戻り値＝なし、対応ボタン＝Load

アセンブラ･プログラムによりアセンブルされ出力された、Intel hex 形式のオブジェクトファイルを読み込み、MIPS プロセッサ実験装置の命令／データメモリにロードする。

・プログラム実行コマンド

R（0x52)

戻り値＝S（0x53)、対応ボタン＝Run

プログラムカウンタの指す番地からプログラムを実行する。押された瞬間ボタン表示が Run から Stop に変わり、もう一度ボタンを押し停止状態に戻すか、MIPS プロセッサ実験装置からの停止コマンドを受け取るまで、他の処理は受け付けない。

・プログラム停止コマンド

O（0x4F)

戻り値＝S（0x53)、対応ボタン＝Stop

プログラム実行中は、Run ボタンが Stop ボタンになっている。プログラムが暴走したときなどに使用する。

・シングルクロック実行コマンド

S（0x53)

戻り値＝なし、対応ボタン＝Single

１クロックサイクルプログラムを実行した後停止する。MIPS プロセッサ実験装置の内部状態監視に用いる。

・レジスタ値変更コマンド

GNNDDDDDDDD

G＝レジスタ値変更コマンド（0x47)、NN＝レジスタ番号（0x00～0x1F)、DDDDDDDD＝変更するレジスタ値（32 ビット 16 進)、戻り値＝なし、対応ボタン＝R_Write（R_Clear)

レジスタ番号指定エディットボックス（Reg_No）に指定したレジスタの値を、変更値指定エディットボックス（Data）の値に変更する。また、R_Clear ボタンは、全てのレジスタの値をクリアする。

・レジスタ値読み出しコマンド

QNN

Q＝レジスタ値読み出しコマンド(0x47)、NN＝レジスタ番号(0x00～0x1F)、戻り値＝DDDDDDDD レジスタの値（32 ビット 16 進）対応ボタン＝R_Read

R_Read ボタンでは、連続してレジスタ読み出しコマンドを発行し、全てのレジスタの値を読み出す。

・メモリ値変更コマンド

XAAAADDDDDDDD

X ＝メモリ値変更コマンド（0x58)、AAAA＝命令／データメモリ番地、DDDDDDDD＝変更するメ

モリ値（32 ビット 16 進)、戻り値＝なし、対応ボタン＝M_Write（M_Clear)

命令／データメモリ番地指定エディットボックス（Address）に指定した番地の内容を、変更値指定エディットボックス（Data）の値に変更する。また、M_Clear ボタンは、全てのメモリの値クリアする。

・メモリ値読み出しコマンド

MAAAA

M＝メモリ値読み出しコマンド（0x4D)、AAAA＝命令／データメモリ番地、戻り値＝DDDDDDDD メモリの値（32 ビット 16 進）対応ボタン＝M_Read

M_Read ボタンでは、連続してメモリ読み出しコマンドを発行し、全てのメモリの値を読み出す。

・プログラムカウンタ値変更コマンド

PAAAA

P＝プログラムカウンタ値変更コマンド（0x50)、AAAA＝プログラムカウンタ値、対応ボタン＝PC_Write

プログラムカウンタの値を、命令／データメモリ番地指定エディットボックス（Address）の値に、変更する。

・全パイプラインレジスタ読み出しコマンド

T（0x54)

戻り値＝全パイプラインレジスタの値（38 バイト)、対応ボタン＝P_Read

全てのパイプラインレジスタの値を読み出す。

・全パイプラインレジスタのクリアコマンド

L（0x4C)

戻り値＝なし、対応ボタン＝P_Clear

全てのパイプラインレジスタの値をクリアする。

・オプション機能（シングルクロック実行コマンドの連続発行）

オプションメニューの設定を選択すると図３のダイアログが出る。

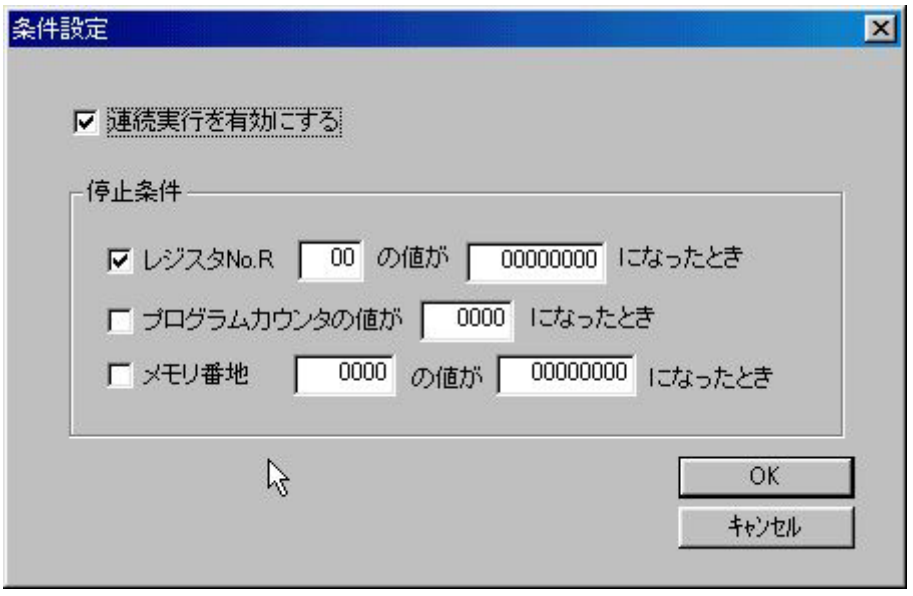

図３．連続実行の停止条件設定ダイアログ

このダイアログの連続実行を有効にするをチェックし､停止条件を設定すると(条件の重複可)、Single ボタンが Start ボタンとなり、停止条件が成立するまで自動的に、シングルクロック実

行を繰り返すことが出来る。(Start ボタンは Run ボタン同様、実行中は Stop ボタンとなる)これは、ユーザーが連続して Single ボタン(シングルクロック実行コマンド)を操作する変わりに、アプリケーション側で自動的に、MIPS プロセッサの状態を監視しながらシングルクロック実行コマンドを送信し、設定された終了条件を満たすまでそれを繰り返すことで実現されている。

２．３．アプリケーションの終了

ファイルメニューから終了を選択するか､ウィンドウの閉じるボタンを押すと WM_CLOSE メッセージが発生する。その対応処理の中で終了確認ダイアログを出し、ウィンドウの破棄やシリアルポートの開放を行う｡ウィンドウが破棄されることにより WM_DESTROY メッセージが発生しそのなかで、WM_QUIT(アプリケーションの終了)メッセージを自分自身に送ることでアプリケーションが終了する。

３．アプリケーションの開発

アプリケーション MPCP の機能実現において、ほとんどのシリアル通信(コマンド)は、逐次実行され戻り値もすぐに返されるためハンドシェイク処理で出来るが、唯一、Run ボタンにより MIPS プロセッサが実行状態になったときのみ特別な工夫が必要となった。MIPS プロセッサが実行状態になると、その戻り値である終了コマンドが送られてくるのを監視しなければならない。しかし、いつ戻るかわからない終了コマンドを、ポーリングで監視しつづけると、処理がWinMain関数に戻らずStopボタンが使えないだけではなく一切のユーザーの操作を受け付けなくなってしまう。

３．１．タイマー関数(SetTimer)

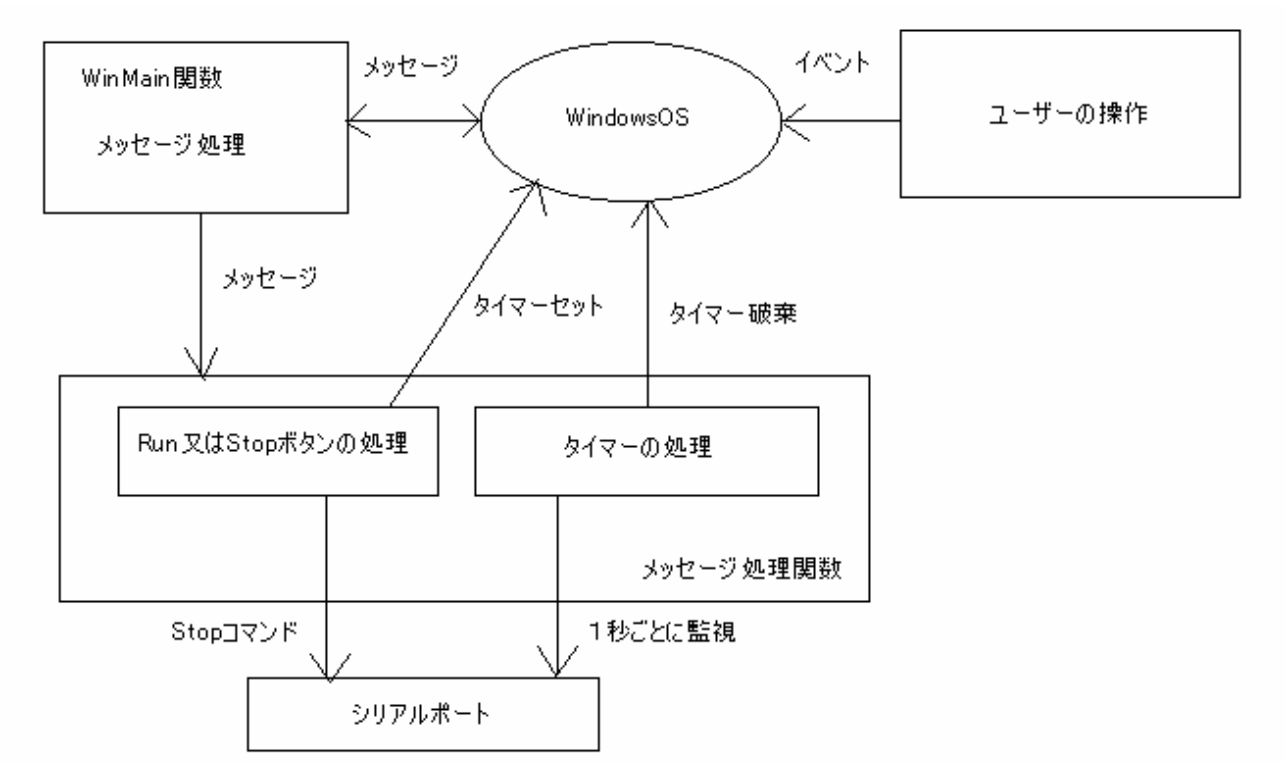

図４．タイマーによるシリアルポートの監視

はじめに思いついたのが、タイマー関数のイベントを利用することである。Run ボタンが押されると、１秒間隔にタイマーをセットする。１秒後にタイマーのメッセージが届いたら、シリアルポートのチェックをし、終了コマンドを受け取っていればそこでタイマーを破棄しウィ

ンドウ内の表示データを更新しユーザーの操作待ち状態に戻る。受け取っていなければ、終了コマンドが来るまで、次の１秒後のタイマーメッセージを待つという方法をとった。この方法だと Stop ボタンを押したときのレスポンスが悪くなるが(ボタンが押されてもすぐにはシリアルポートのチェックに行かないため）機能的には問題なかった。図４はタイマーによるシリアルポートの監視をブロック図で表したものである。

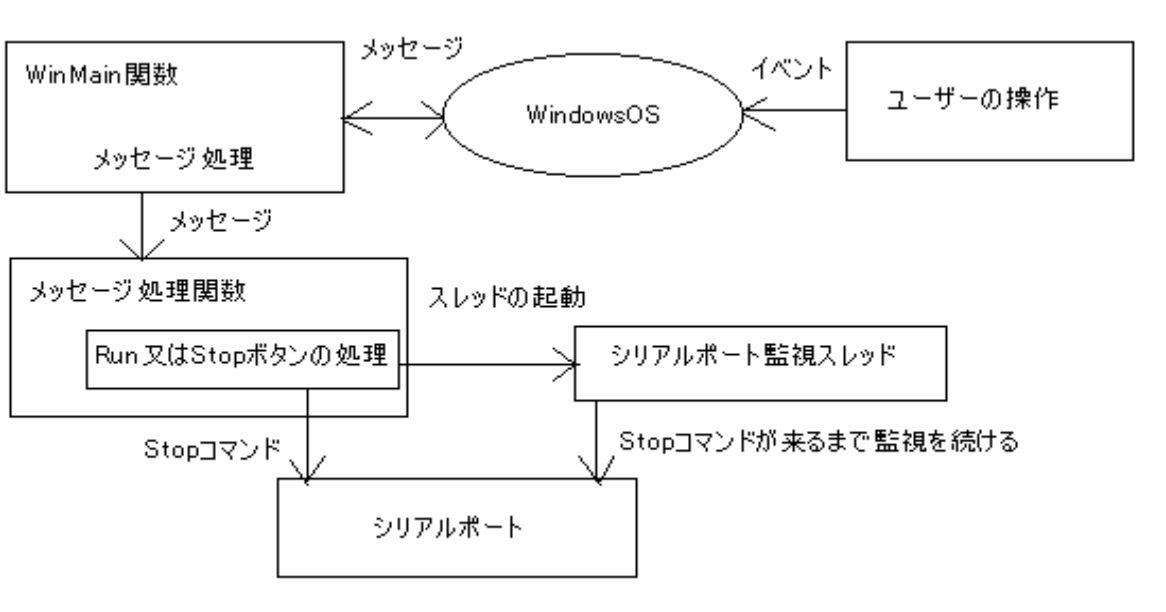

図５．別スレッドによるシリアルポートの監視

３．２．マルチスレッド

しばらくすると、CPU の実行時間が知りたいという要求が出てきた。タイマーイベントを利用している限り最大１秒の誤差が出てしまう。タイマー間隔を短くしても誤差が生じることに違いはない。

そこで今度は、シリアルポート監視用に別のスレッドを走らせることにした。これで監視用スレッドの中でポーリングに入っても処理は戻り、CPU 実行時間の計測誤差もない。図５にマルチスレッドによるシリアルポートの監視状態を示す。

３．３．オーバーラップ

学生実験用の Windows コンピュータには、OS として Windows98 がインストールされている。ところがあるときから、学生に対する説明のために使うコンピュータの OS が Windows2000 になった。Windows2000 上でも動作可能と考えていたが、シリアルポートの扱いが違うことが判明した。Windows98 では、シリアルポートをノーマルでオープンしても問題なくオーバーラップ（ポートを読み出していながら書き込むことも出来る、逆もまた可）として動作するが、Windows2000 では、オーバーラップとしてシリアルポートをオープンしなければならない。シリアルポートをオーバーラップでオープンすると、読み込み・書き込み・ポート監視用の関数の扱いが違ってくるため若干のコード変更が必要になった。表３は、そのコード変更の例である。

ノーマルでオープン	オーバーラップでオープン
/*シリアルポートのオープン*/ hComm = (HWND)CreateFile("COM1", GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,NULL);	/*シリアルポートのオープン*/ hComm = (HWND)CreateFile("COM1", GENERIC_READ | GENERIC_WRITE, 0, NULL,OPEN_EXISTING, FILE_FLAG_OVERLAPPED,NULL);
/*ポートの監視*/ SetCommMask(hComm, EV_RXFLAG); WaitCommEvent(hComm, &dwEvent, NULL); ReadFile(hComm, &stop, 1, &dwAccBytes, NULL);	/*ポートの監視*/ SetCommMask(hComm, EV_RXFLAG); WaitCommEvent(hComm, &dwEvent, &ovlpd); WaitForSingleObject(ovlpd.hEvent, INFINITE); ReadFile(hComm,&stop,1,&dwAccBytes, &ovlpd); WaitForSingleObject(ovlpd.hEvent, INFINITE);

表３．コード変更の例

４．むすび

今後、また改良の必要も出てくるであろうが、現在では、アプリケーション MPCP は、OS に依存することなく動作している。

昨今では、Windows コンピュータの普及に伴いどこの部署に所属していても、アプリケーションプログラミングの機会はありそうである。この発表が、これから Windows アプリケーションの開発に着手する方達の参考に、少しでもなれば幸いである。

参考文献

［１］山本信雄著：プログラミング学習シリーズ Visual C++ 1　はじめての Windows プログラミング（翔泳社）

［２］小野雅晃、中山勝：MIPS プロセッサ実験システムの開発（筑波大学　技術報告 No.21 p33）