並行処理の同期に関する実例の実現

大　西　孝　臣*

An implementation of an example to show a technique of the multiple processing synchronization

Takaomi OHNISHI

Abstract

In this article, the author has manufactured and implemented an example, which is an intranet-controlled model railroad system, to show a technique of the multiple processing synchronization.

The "synchronization" is now one of the key techniques to create any of Win32-based applications.

1．はじめに

今後，Win32 以降の API に基づく Windows アプリケーションを作成する際には，いずれの機会にかマルチスレッド等の並列処理を意識したプログラミングを必須のものとして学ばなければならない。しかし並列処理プログラミングを扱う書籍の多くは，そのアプローチが抽象的である。

しかし，マルチスレッドに限らなくとも，Windows アプリケーションを作成する立場から見れば，ごくありふれた場所にも解決すべき並列処理の問題が存在する。その事実を，プログラマを目指す学生に早期に気付かせなければならない。

そこで本稿では，複数のプログラムによるファイルアクセス問題を解決する簡単な実例を取り上げ，並列処理における「同期」の技法を紹介する。

2．制御対象について

ドイツのウルム大学[1]では，現地にある定置カメラより送られるリアルタイムの映像を見ながら，映像にある模型の電車を WWW ブラウザを用いて遠隔操作するという内容のホームページを提供している。対象は 2 両の模型の電車と環状の路線と 3 つの駅で構成されており，いずれかの電車をどちらの駅に進めるかという命令を，終日世界中から受け付け，もし命令が受理されれば，現地で模型の電車が動く様子を映像として見る事が出来る。

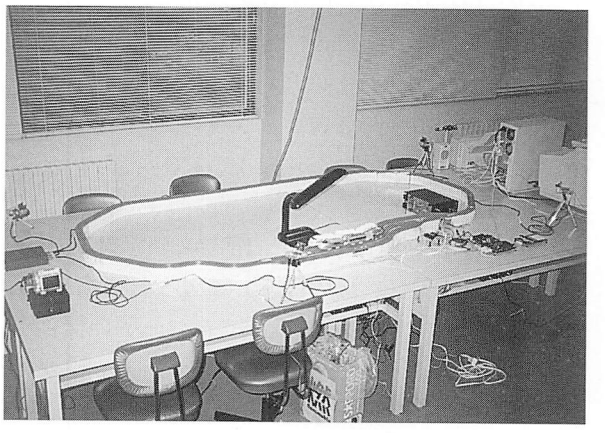

図 1　システムの概観

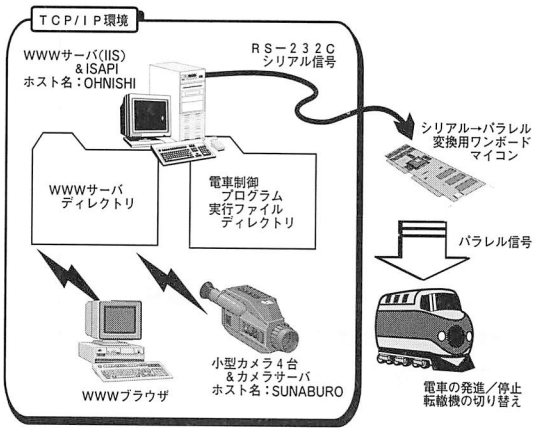

図 2　システムの構成

本稿では，上記の機能を持つホームページ及び制御対象を安価に実現する。

＊　助　手　情報工学科

3．システムの概要

本稿で紹介するシステムの概観を前頁の図1に，システムの機器の構成図を前頁の図2に示す。

図2では，TCP/IPプロトコルに従うイントラネット上に，“OHNISHI”と名付けたホストがあり，OHNISHIにはWindowsNTSer. Ver. 4.0と，Microsoft社のIIS（Internet Information Server）[2]と呼ばれるWWWサーバが起動している。

IISはISAPI（Internet Server API）[3]と呼ばれるAPIを用いて，WWWサーバの機能をDLLの形で拡張する事が出来る。このDLLをISAPIサーバ拡張DLLと呼ぶ。ISAPIによるWWWサーバの拡張は，MFCを用いても記述する事が出来るので，ISAPIサーバ拡張DLLはVisualC＋＋Ver.6.0により作成出来る。

ISAPIサーバ拡張DLLは，CGIに類似する手続きでWWWブラウザからのフォーム入力を受け付けるが，その上WWWサーバの全てのリソースにアクセスする機能があり，CGIより柔軟である。

ホストOHNISHIにあるIISが，WWWサーバディレクトリ（次頁の図7）にあるHTMLファイル群を提供する事によるホームページの概観を図3に，図3のページでの各HTMLファイルの配置を図4に示す。

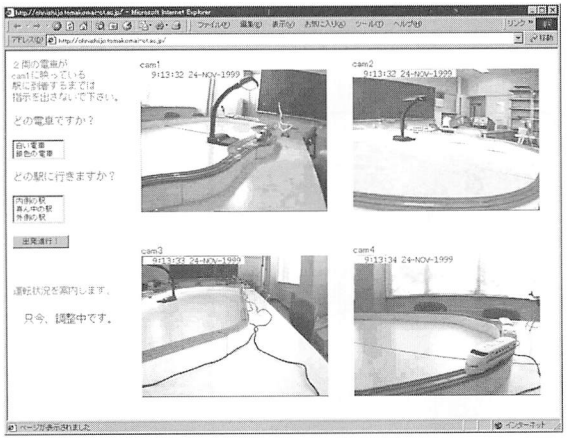

図3　模型の電車を遠隔操作するページ

イントラネット上には，ホスト名“SUNABURO”と名付けた，AXIS社製のネットワークカメラサーバAXIS 240 CameraServer（図5）[4]を10Base-Tを伝送媒体として接続し，1/3インチ小型ビデオカメラ4台（図1，図5，図6）で撮影した映像をJPEG圧縮して図4のcam1.html からcam4. html という形でページに載せている。映像の載った4枚のページはそれぞれ約10秒毎に更新される。4台のカメラは環状の路線の順路に従って配置した。

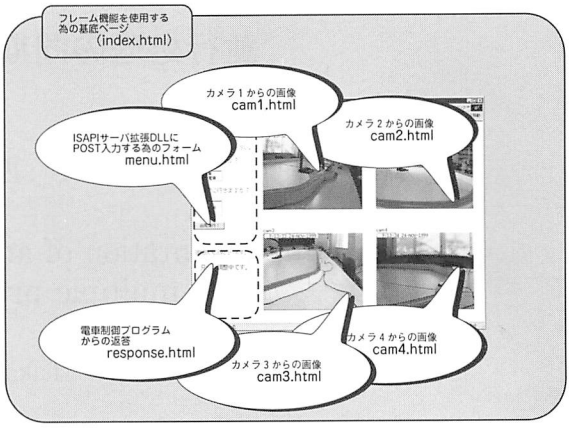

図4　図3でのHTMLファイルの配置

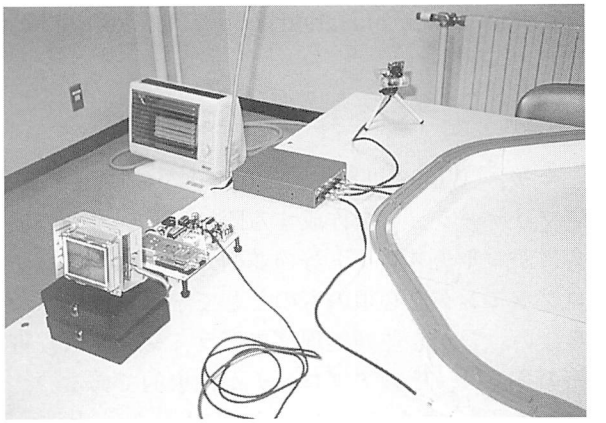

図5　小型カメラ（机の角），カメラサーバ（写真中央），手製のモニタ（写真左下）

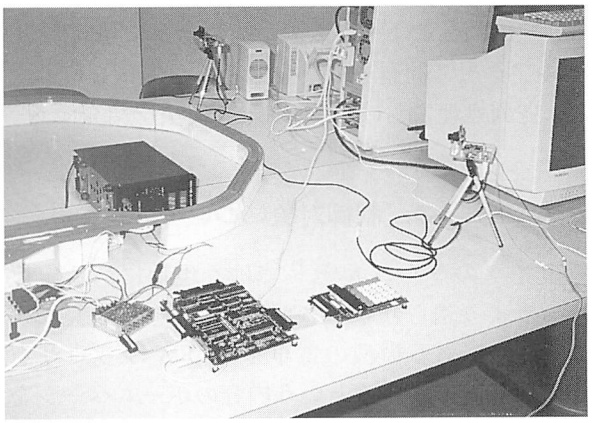

図6　小型カメラ（写真右），Z80ワンボードマイコン（写真中央よりやや下）

WWW ブラウザから適切な入力が行われると，ホスト OHNISHI は RS-232C ケーブルを使ってシリアル信号を送り[5]，Z80ワンボードマイコン（図6）にてシリアルからパラレルへの信号変換を行い，電車への発進／停止の指示や電車を３つの駅に振り分ける為の２台の転轍機への指示を行う。

4．イベントとファイルの競合

4．1　ディレクトリの構成とファイルの配置

ホスト OHNISHI での本システムに必要なファイル群の構成を図７に示す。

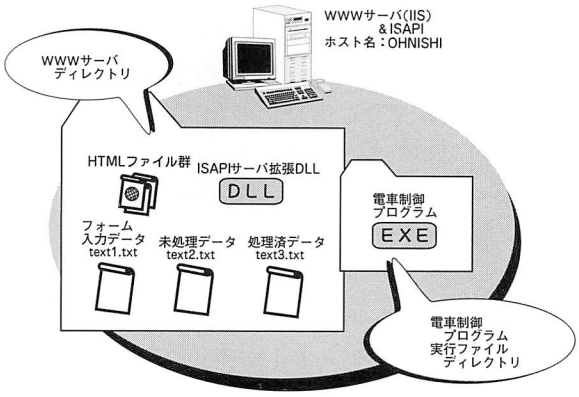

図７　ディレクトリ・ファイル群の構成

ホスト OHNISHI には，WWW ブラウザに対する情報を提供する為のファイルを持つ「WWW サーバディレクトリ」と，電車を制御する為のプログラムがある「実行ファイルディレクトリ」とにディレクトリを分けて構成した。WWW サーバディレクトリには HTML ファイル，ISAPI サーバ拡張 DLL の他に，4.3節で紹介する各イベントの間での情報伝達の手段として，３つのデータファイルを配置した。WWW サーバディレクトリに配置したのは，管理者がシステムの運営上，WWW ブラウザから実行状況を見る必要がある為である。

4．2　２つの実行ファイル

本システムでの実行プログラムは ISAPI サーバ拡張 DLL と電車制御プログラムの２つである。

電車制御プログラムの実行画面を図８に示す。システムの状況は WWW ブラウザ経由で見る事が出来るので，図８の画面としてはシンプルな SDI で充分である。

他の，２つの実行ファイルの動作による本システムへの効果の主なるものは，次節に説明する“イベント”の起動に伴うものして，説明した方が良い。

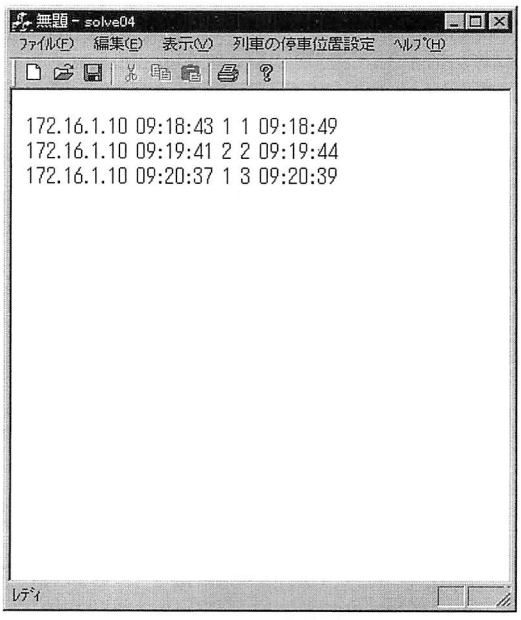

図８　電車制御プログラムの実行画面

4．3　発生する主なイベント

ISAPI サーバ拡張 DLL と電車制御プログラムの２つの実行プログラムは共に VisualC ＋＋6.0 を使用して作成した Win32アプリケーションである。つまりイベント駆動型のアプリケーションである（ただし，本稿でいう“イベント”とは，オブジェクト指向プログラミングで言う抽象的な“イベント”に近く，むしろ VisualBasic プログラミングでの“イベント”と同義である。Win32 プログラミングで同期オブジェクトの１種としての具体的な“イベント”とは異なる）。

ISAPI サーバ拡張 DLL での主なイベントは，次のフォーム入力イベントである。

◎フォーム入力イベント

WWW ブラウザのユーザによる menu.html でのフォーム入力を，ISAPI サーバ拡張 DLL が受け取る事により，不定期に発生する。

フォームに必要な入力をしているか，どの様な入力かを解析し，図９の様に解析結果を返答するページ isapi02.dll?train=＃&station=#を，menu.html を置き換える形で表示する。

返答のページは，正しい入力なら10秒間，

必要な入力がされていなければ3秒間表示され，元のmenu.htmlに戻る。

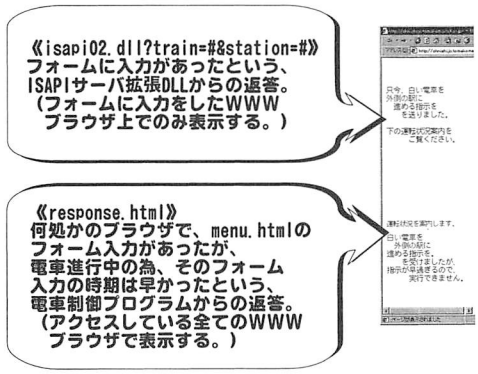

図9　フォーム入力による図3からの変化

もしフォームに必要な入力がされていれば，フォーム入力データファイル（text1.txt）を開いて，フォーム入力データを追加する。

その後，本イベントを終了する。

電車制御プログラムでの主なイベントは，タイマにより起動される以下の4つのイベントである。

◎タイマ・イベント1

電車制御プログラムでの本イベント専用のタイマにより，基本的に約1秒毎に発生する。

フォーム入力データファイル（text1.txt）に記述があるかどうか調べ，もし記述が無ければ，response.htmlを「運転状況なし」に書き変え，本イベントを終了する。

もしtext1.txtに記述があれば，電車ビジー（電車進行中）のフラグが発生していないかを調べ，もし発生していれば，response.htmlを「指示が早過ぎる」に書き変え（図9），本イベントを終了する。

もし電車ビジーのフラグが発生していなければ，text1.txtの記述に従うと電車同士で衝突しないかを調べ，もし衝突するならば，本イベント専用のタイマを停止させ，response.htmlを「衝突するので実行出来ない」に書き変え，イベント4が10秒後に発生する様，イベント4専用のタイマを起動させ，本イベントを終了する。

もし電車同士で衝突しないならば，未処理データファイル（text2.txt）を開いて，text1.txtの記述を転記し，response.htmlを「指示通り進める」に書き変え，電車ビジーのフラグを発生させ，イベント3が45秒後に発生する様，イベント3専用のタイマを起動させ，本イベントを終了する。

なお，本イベントにより描き変えられたresponse.htmlは10秒間表示される。

◎タイマ・イベント2

電車制御プログラムでの本イベント専用のタイマにより，基本的に約5秒毎に発生する。

未処理データファイル（text2.txt）に記述があるかどうか調べ，もし記述が無ければ，本イベントを終了する。

もしtext2.txtに記述があれば，text2.txtでの指示を電車制御プログラムの実行画面上に表示し，text2.txtでの指示を電車・転轍機が理解出来る信号に変換してRS-232Cに送り，処理済データファイル（text3.txt）を開いて，text2.txtの記述を転記し，本イベントを終了する。

◎タイマ・イベント3

タイマ・イベント1が本イベント専用のタイマを起動させてから約45秒後に，本イベントが発生する。

response.htmlを「運転状況なし」に書き変え，本イベント専用のタイマを停止させ，電車ビジーのフラグを取り下げ，本イベントを終了する。

◎タイマ・イベント4

タイマ・イベント1が本イベント専用のタイマを起動させてから約10秒後に，本イベントが発生する。

本イベント専用のタイマを停止させ，イベント1が10秒後に発生する様，イベント1専用のタイマを起動させ，本イベントを終了する。

4．4　共有ファイルの競合，イベントの同期

フォーム入力イベントでは実行中にtext1.txtを押さえなければならず，タイマ・イベント1ではtext1.txtとtext2.txtを，タイマ・イベント2ではtext2.txtとtext3.txtを同時に押さえなければならない。

プログラムを作成する際は，各イベントは不定期に発生する事を想定しなければならないし，タイマにより定期的に起動するイベントであって

も，イベントの処理時間が実行中たとえ１回でもタイマの設定時間を超える場合に，イベントの多重起動が起きる事も想定しなければならない。

従って，ISAPIサーバ拡張DLLと電車制御プログラムという複数のプログラム間や複数のイベント間で共有のデータファイルの競合が発生するのを回避する為，ファイルのアクセスに関する相互排除的な制御を行う必要がある。言い換えると，イベントの処理の同期[5)6)]を取る必要がある。

5．ソースレベルでの実現

5．1　ミューテックス

WindowsNT等のWin32によるOSでは，主要な同期オブジェクト[5)6)]として，クリティカルセクション，ミューテックス，セマフォ，（5．1節での意味とは異なる，狭義での）イベント，ウェイタブルタイマがあり，これらの手法はマルチスレッドプログラミングの際の必須項目である。

本システムの様に，複数のプログラム間でのデータファイルへのアクセスの同期を取る為には，ミューテックスを良く使用する。

5．2　ミューテックスによる同期

4．4節で述べた様にISAPIサーバ拡張DLLのフォーム入力イベントではではtext1.txtを押さえなければならない。

次のソースに示す様に，text1.txtを使用する関数MyFunc02の最初にはCreateMutex関数を用いてミューテックスに対するカーネルオブジェクトなるものを作成し，“text1.txt”という名称をそのオブジェクトに与え，そのオブジェクト固有のハンドルを返す。MyFunc02の最後には，そのオブジェクトを閉じる為にCloseHandle関数を用いる必要がある。

“text1.txt”というミューテックスカーネルオブジェクトの名称は，データファイルtext1.txtのファイル名に故意に一致させただけで，別の名称でも構わない。しかし，“text1.txt”というオブジェクトの名称は，OSのシステムが固有のものとして了解しており，他のプログラムにおいて“text1.txt”という名称のミューテックスカーネルオブジェクトを作成しようとすると，以前に作成した“text1.txt”オブジェクトに与えたハンドルを返す。

関数MyFunc02でのtext1.txtを使用する領域の前後には“text1.txt”オブジェクトに与えたハンドルを引数にしたWaitForSingleObject関数とReleaseMutex関数があり，これら２つの関数の間を（広義の）クリティカルセクションと呼ぶが，このクリティカルセクションの中の処理を行う間は，他のプログラムでの同じハンドルを引数にしたクリティカルセクションの処理を行わなず，関数MyFunc02でのクリティカルセクションの処理が終わるのを待つ事になる。

```
// ISAPI02.CPP - インターネット サーバー用のインプリメンテーション ファイル
//    isapi02 Extension

...

void CIsapi02Extension::MyFunc02(CHttpServerContext *pCtxt,
                                 LPCSTR train, LPCSTR station)
{
    ...
        HANDLE hMutex = CreateMutex(NULL, FALSE, "text1.txt");
        WaitForSingleObject(hMutex, INFINITE);

    ...    // ->-> ここで、text1.txtを使用する。<-<-

        ReleaseMutex(hMutex);
        CloseHandle(hMutex);
    ...
}

...
```

電車制御プログラムでは，4．4節で述べた様に，イベントの処理において，複数のデータファイルを同時に押さえなければならない。

この場合，次頁のソースの様に，複数のミューテックスカーネルオブジェクトのハンドルを配列に代入して，配列を引数にしたWaitMultipleObjects関数とReleaseMutex関数を用いてクリティカルセクションを形成する。

6．おわりに

本システムは，中学生の為の見学会や高専祭の学科展において情報工学科を紹介する展示物として紹介したものである。一般の見学者からは，その概観から分かり易い事例であるとの評価を得ている。

一方，Windowsアプリケーションを作成する立場から見れば，アプリケーションの並行処理という性質から，ごくありふれた場所にも解決すべき問題が存在する，あるいは問題が想定される事，そしてプログラマはプログラムの時点で問題解決しなければならない使命がある事を，本システムの実現により学生に教授出来るのではないかと考える。

```
// solve04View.h : CSolve04View クラスの宣言およびインターフェイスの定義をします。

class CSolve04View : public CView
{
    ...
    public:
        int timer;
        HANDLE hMutex1, hMutex2, hMutex3;
        HANDLE hArray[2];
    ...
};


// solve04View.cpp : CSolve04View クラスの動作の定義を行います。

...

#include "solve04View.h"

...

CSolve04View::CSolve04View()    // コンストラクタ
{
    ...
        hMutex1 = CreateMutex(NULL, FALSE, "text1.txt");
        hMutex2 = CreateMutex(NULL, FALSE, "text2.txt");
        hMutex3 = CreateMutex(NULL, FALSE, "text3.txt");
    ...
}

CSolve04View::~CSolve04View()    // デストラクタ
{
    ...
        CloseHandle(hMutex1);
        CloseHandle(hMutex2);
        CloseHandle(hMutex3);
    ...
}

...

void CSolve04View::OnDraw(CDC* pDC)
{
    ...
        timer=SetTimer(1, 1000, NULL);
        timer=SetTimer(2, 5000, NULL);
    ...
}


void CSolve04View::OnTimer(UINT nIDEvent)
{
    ...
        if(nIDEvent == 1)
        {
            ...
                hArray[0] = hMutex1;
                hArray[1] = hMutex2;
                WaitForMultipleObjects(2, hArray, TRUE, INFINITE);

            ...    // ->->->-> ここからtext1.txt、text2.txtを使用する。

                if(電車はビジーではなかった)
                {
                    ...
                        if(衝突しないフォーム入力であった)
                        {
                            ...
                                SetTimer(3, 45000, NULL);
                        }
                        else
                        {
                                KillTimer(1);
                            ...
                                SetTimer(4, 10000, NULL);
                        }
                    ...
                }

            ...    // ここまでtext1.txt、text2.txtを使用する。<-<-<-<-

                ReleaseMutex(hArray[0]);
                ReleaseMutex(hArray[1]);
        }
        else
        {
                if(nIDEvent == 2)
                {
                    ...
                        hArray[0] = hMutex2;
                        hArray[1] = hMutex3;
                        WaitForMultipleObjects(2, hArray, TRUE, INFINITE);

                    ...    // ->-> ここでtext2.txt、text3.txtを使用する。<-<-

                        ReleaseMutex(hArray[0]);
                        ReleaseMutex(hArray[1]);
                    ...
                }
                else
                {
                        if(nIDEvent == 3)
                        {
                            ...
                                KillTimer(3);
                            ...
                        }
                        else    // すなわち(nIDEvent==4)の場合。
                        {
                                KillTimer(4);
                                SetTimer(1, 10000, NULL);
                        }
                }
        }
    ...
}
```

参考文献

1) Interactive Model Railroad, http://rr-vs.informatik.uni-ulm.de/rr/, Distributed Systems Dept., University of Ulm

2) Microsoft Internet Information Server インストール アンド アドミニストレーション ガイド, Microsoft Corporation

3) MSDN ライブラリ VisualStudio6.0, Microsoft Corporation

4) AXIS 240 Camera Server ユーザーズマニュアル, AXIS COMMUNICATIONS

5) Jeffry Richter 著, 長尾 高弘訳, Advanced Windows 改訂第3版, アスキー, 1997

6) James E. Beveridge, Robert Wiener 共著, インフォビジョン訳, Win32マルチスレッドプログラミング, アスキー, 1997

(平成11年11月30日受理)