オブジェクト指向とゲームプログラミング

▷　フレームワーク編 － 第９回　多重起動の防止　◁

アプリケーションの多重起動

多重起動とは、同じアプリケーションが複数起動することです。通常、エディタやワープロなどのように、いくつものファイルを参照、編集するアプリケーションでは、ほとんどの場合、多重起動を許しています。これにより、ウィンドウシステムを有効利用して複数のファイルを操作することができます。

反対に、多重起動を許さないアプリケーションもあります。たとえば、ゲームやシステム関連ツールなどです。これらのアプリケーションは、システム上で１つだけ起動すれば良く、複数起動すると、データの整合性などの問題が発生しかねません。

多重起動の検出

多重起動を許さないアプリケーションを作成する場合、多重起動を許さないための処理を行わなければなりません。この処理は、アプリケーションの起動時の動作のため、初期化部分に実装する必要があります。

多重起動を防止するための方法は、以下の２つが多く用いられています。

・FindWindow関数の利用
・ミューテックスの利用

FindWindow関数による多重起動の検出

多重起動を検出するための方法として、FindWindow関数を利用する方法がいちばん多く使われています。この方法は、ウィンドウ作成に使用したウィンドウクラス名とウィンドウタイトル名から判断し、多重起動を検出する方法です。ただし、何らかのアプリケーションによってタイトル名を変えられてしまうと、この方法では検出できなくなってしまいます。

FindWindow関数は、ウィンドウを検索する関数です。起動したアプリケーションのメインウィンドウと同じクラス名、ウィンドウ名を持つウィンドウをFindWindow関数で検索し、見つかった場合は多重起動であるとみなします。

```
if(FindWindow("検索するウィンドウのクラス名", "検索するウィンドウのタイトル") != NULL) {
    // 多重起動である
    MessageBox(GetDesktopWindow(), "すでに起動されています", "アプリケーション名",
               MB_ICONINFORMATION | MB_OK);
}
```

このように、FindWindow関数は検索条件としてウィンドウのクラス名とタイトル(タイトルバーに表示されている文字列)を指定します。どちらかの引数にNULLを指定することもできます。Windows上に存在するすべてのウィンドウが検索対象で、条件を満たすウィンドウが見つかった場合はそのウィンドウのハンドル、見つからない場合はNULLを返します。

FindWindow関数による方法の問題点

FindWindow関数を使って多重起動を検出する方法は、いくつか問題点があり、推奨されていません。ウィンドウクラス名がたまたま重複したり、ウィンドウタイトルを変更するアプリケーションでは、検出できない場合があるからです。また、ウィンドウが生成されていないと検索できないため、アプリケーションが起動されてからウィンドウが生成されるまでの間(ウィンドウが生成されてからＯＳに登録されるまでの間)に複数起動されてしまう場合があります。

複数起動を検出する方法でもっとも推奨されているのは、ミューテックスを使う方法です。

ミューテックス

ミューテックスとは、MUTual EXclusion(相互排他)からきた名前で、複数のプログラム(プロセス、スレッド)間で排他制御や同期を取る場合に使います。ミューテックスは、１度に１つのプログラム(正確にはスレッド)だけが所有権を取得できます。この特性を多重起動の検出に利用します。

ミューテックスは作成時に名前を割り当てることができます。名前を割り当てると、ＯＳのどの場所にあるプログラムからでもミューテックスにアクセスすることができます。ほかのプログラムに直接ミューテックスを渡すことはできませんが、ミューテックスの名前からそのハンドルを取得する関数があるので、これを利用して多重起動を検出します。
名前を持つミューテックスを生成したとき、同じ名前のミューテックスがすでに生成されている場合に、多重起動とみなすわけです。

ミューテックス名

ミューテックスの名前は、ほかのアプリケーションやプログラムで生成されるミューテックスと重複する確率がなるべく低いものにしなければなりません。
以下の例では、ドライブ名とパス名を含めたアプリケーションのファイル名を名前にしています。この方法は、ほかのプログラムのミューテックス名と重複することはありませんが、ドライブやパスが異なるところから同じアプリケーションを起動された場合の検出ができません。

アプリケーションのフルパスを含めたファイル名は、GetModuleFileName関数で取得することができます。ここで、パスの区切り文字'\'は、ミューテックス名として使うことができないので、'/'などほかの文字に置き換える必要があります。

```
// アプリケーションファイル名の取得
TCHAR   AppFileName[MAX_PATH + 1];
GetModuleFileName(NULL, AppFileName, MAX_PATH);

// 文字'\'を'/'に変更
for(int i = 0; i < lstrlen(AppFileName); i++) {
    if(AppFileName[i] == '\\')
        AppFileName[i] = '/';
}
```

ミューテックスの利用

ミューテックスを利用した多重起動の検出は、以下の手順で行います。

１．ミューテックスの生成
２．多重起動かどうかの判別
３．初めての起動の場合、ミューテックスの所有権を取得する

ミューテックスの生成は、CreateMutex関数で行います。ミューテックスの生成に成功すると、ミューテックスのハンドルが返されます。すでに作成されていた場合でも正常終了となり、ハンドルが返されますが、GetLastError関数がERROR_ALREADY_EXISTSを返すので、この場合は多重起動とみなします。よって、起動処理を中止します。

初めての起動の場合、ミューテックスの所有権をWaitForSingleObject関数で取得し、「起動済み」ということを示しておきます。同じの名前のミューテックス所有権はひとつだけです。ほかのプログラムがWaitForSingleObject関数で所有権を取得しようとすると、タイムアウトとなり、WAIT_TIMEOUTが返されます。

ミューテックスを解放するにはCloseHandle関数を使います。ミューテックスが解放されないままプログラムが終了した場合、ＯＳはそのミューテックスを自動的に解放します。所有権の解放はReleaseMutex関数で行います。所有権を取得したままプログラムが終了した場合、ＯＳはその所有権を解放しません。

CreateMutex関数

－説明－
CreateMutex関数は、ミューテックスオブジェクトを生成します。

－パラメーター－
　１つ目の引数は、セキュリティ属性構造体のアドレスです。NULLを指定するとデフォルトのセキュリティが使用されます。

　２つ目の引数は、所有権の指定です。TRUEを指定すると所有権を要求します。FALSEを指定すると所有権を要求しません。

　３つ目の引数は、ミューテックスの名前です。

－戻り値－
　関数が正常に終了した場合は、ミューテックスオブジェクトのハンドルを返します。同じ名前のミューテックスオブジェクトがすでに存在している場合も正常終了になります。この場合、GetLastError関数はERROR_ALREADY_EXISTSを返します。関数が正常に終了しなかった場合は、NULLを返します。

WaitForSingleObject関数

－説明－
　WaitForSingleObject関数は、指定されたオブジェクトがシグナル状態になったとき、またはタイムアウト時間が経過したときに、制御を戻します。

－パラメーター－
　１つ目の引数は、オブジェクトのハンドルです。指定できるハンドルの種類は、イベント、ミューテックス、プロセス、スレッド、セマフォです。

　２つ目の引数は、タイムアウト時間をミリ秒単位で指定します。タイムアウト時間が経過すると、関数は、オブジェクトの状態が非シグナル状態である場合でも、制御を戻します。

－戻り値－
　関数が正常に終了した場合は、関数が制御を戻す原因となったイベントを返します(WAIT_OBJECT_0, WAIT_ABANDONED, WAIT_TIMEOUT)。それ以外の場合は、WAIT_FAILEDを返します。

```
    // ミューテックス生成
    hMutex = CreateMutex(NULL, 0, AppFileName);

    if(GetLastError() == ERROR_ALREADY_EXISTS) {
        // ミューテックスがすでに生成されていた場合、多重起動である
        MessageBox(GetDesktopWindow(), "すでに起動されています", "アプリケーション名",
                   MB_ICONINFORMATION | MB_OK);
                ⋮
    }

    // ミューテックスの所有権を取得する
    const DWORD   Ret = WaitForSingleObject(hMutex, 0);
    if(Ret != WAIT_OBJECT_0 && Ret != WAIT_ABANDONED) {
        // 失敗または別のスレッドが所有権を保持している
    }
```

多重起動検出後の処理

　多重起動を検出した場合、以前に起動しているアプリケーションのウィンドウをアクティブにしたりタイトルバーを点滅させるなどの処理を行います。アプリケーションが最小化されている場合は、元の状態に復元するのもよいでしょう。

　最小化されているアプリケーションを復元にするには、まずShowWindow関数でウィンドウを元の表示状態にします。次に、SetForeground関数でウィンドウをフォアグラウンド(前面)、かつアクティブ(キーボードやマウスの入力権を得る)状態にします。最小化されているかどうかはIsIconic関数で調べることができます。
　この機能を使えば、アプリケーションが最小化されているときに多重起動された場合、最初に起動していた方を元の状態に戻すことができます。

```
    // hWndはウィンドウのハンドル
    if(IsIconic(hWnd) != 0)
        ShowWindow(hWnd, SW_RESTORE);    // 最小化されているウィンドウを元の状態に戻す
    SetForegroundWindow(hWnd);           // フォアグラウンドに設定する
```

ミューテックスとウィンドウの関連づけ

以前に起動しているアプリケーションのウィンドウをアクティブにしたり、最小化を復元するには、ウィンドウのハンドルが必要になります。FindWindow関数を利用する場合は、戻り値をそのまま使用できますが、ミューテックスのハンドルやミューテックス名から、ミューテックスを所有しているアプリケーションのウィンドウを取得するAPIはありません。

ウィンドウには、プロパティ(属性)という領域があり、任意の文字列とハンドルを登録することができます。これを利用すれば、ウィンドウとミューテックス名、ミューテックスのハンドルを関連づけることができます。
プロパティの追加は、SetProp関数で行うことができます。この関数は文字列とハンドルのペアをウィンドウに関連づけることができます。

```
// プロパティ追加
SetProp(hWnd, AppFileName, hMutex);      // hWndはメインウィンドウのハンドル
```

プロパティ(文字列)が追加されたウィンドウに対してGetProp関数を呼び出すと、SetProp関数の３つ目の引数で渡したハンドルが返されます。プロパティが一致しない場合やプロパティを持たない場合はNULLが返されます。これを利用すれば、ミューテックス名からミューテックスを所有するウィンドウを探すことができます。もし、ミューテックスがすでに生成されていた場合、Windows上に存在するすべてのウィンドウに対し、GetProp関数でプロパティの取得を試みます。有効なハンドルを返すウィンドウならミューテックスを所有している、すなわち最初に起動したアプリケーションなので、最小化されている場合は復元し、フォアグラウンドに設定してフォーカスを取得させます。

```
// すべてのウィンドウから、ミューテックス名を持つウィンドウを探す
HWND   hWnd = GetWindow(GetDesktopWindow(), GW_CHILD);  // デスクトップが基点
while(hWnd != NULL) {
    // ミューテックス名を持つウィンドウか調べる
    if(GetProp(hWnd, AppFileName) != NULL) {
        // 最小化されている場合は復元
        if(IsIconic(hWnd) != 0)
            ShowWindow(hWnd, SW_RESTORE);

        SetForegroundWindow(hWnd);                          // フォアグラウンドに設定
        SetForegroundWindow(GetLastActivePopup(hWnd));  // 子ウィンドウにフォーカス設定

        break;
    }

    // プロパティが一致しない場合は次のウィンドウへ
    hWnd = GetWindow(hWnd, GW_HWNDNEXT);
}
```

課　題

ミューテックスを利用してアプリケーションの多重起動を検出するCSingleAppクラスを作成し、多重起動を防止しましょう。

(１)CSingleAppクラスのヘッダファイルを作成しましょう。

CSingleAppクラスに必要な属性を考えます。ミューテックスを利用するので、ミューテックスのハンドルとミューテックス名が必要です。また、ミューテックス名をウィンドウのプロパティに登録したり解除したりするので、ウィンドウのハンドルも必要です。
まとめると、必要な属性は、以下のようになります。

・ミューテックスのハンドル(HANDLE型)
・ミューテックス名(文字列)
・ミューテックス名が関連づけられたウィンドウのハンドル(HWND型)

次に、必要な機能を考えます。コンストラクタとデストラクタのほかに、CSingleAppオブジェクトを初期化する機能とプロパティやミューテックスなど、資源を解放する機能が必要です。また、ウィンドウのプロパティにミューテックスを登録したり、解除したりする関数が必要です。
まとめると、必要な機能は、次のようになります。

・初期化
・解放
・ウィンドウのプロパティにミューテックスを登録する
・ウィンドウのプロパティからミューテックスを解除する

以上をふまえると、CSingleAppクラスのヘッダファイル(SingleApp.hpp)は以下のようになります。

－SingleApp.hpp－

```
/*
================================================================================================
                                オブジェクト指向ゲームプログラミング
    Programmed by Hibikino software. Copyright (c) 2005 Hibikino software. All rights reserved.
================================================================================================

  【対象ＯＳ】
    Microsoft Windows2000/XP

  【コンパイラ】
    Microsoft Visual C++ 2005

  【プログラム】
    SingleApp.hpp
                シングルアプリケーションクラスヘッダ

  【履歴】
    * Version    1.00       2005/03/dd  hh:mm:ss

================================================================================================
*/

#pragma ????

/*********************************************************************************************/
/*                                      インクルードファイル                                 */
/*********************************************************************************************/
#include <windows.h>

/*********************************************************************************************/
/*                                  シングルアプリケーションクラス定義                       */
/*********************************************************************************************/
class CSingleApp {
public:
    CSingleApp();
    CSingleApp(LPCTSTR inMutexName);
    ??????? ~CSingleApp();

    bool Initialize();
    void Release();

    bool SetHWnd(const HWND hWnd);
    void RemoveHWnd();

    bool OnCreate(const HWND hWnd) { return SetHWnd(hWnd); }
    void OnDestroy() { RemoveHWnd(); }

private:
    HANDLE    m_hMutex;                   // ミューテックスのハンドル
    TCHAR     m_MutexName[MAX_PATH + 1]; // ミューテックス名
    HWND      m_hWnd;                     // ミューテックスと関連づけたウィンドウのハンドル

    CSingleApp(const CSingleApp&);
    CSingleApp& operator=(const CSingleApp&);
};
```

(２)CSingleAppクラスのメンバは、以下のとおりです。

CSingleApp　デフォルトコンストラクタ

CSingleAppオブジェクトをファイル名(フルパス)で構築します。

CSingleApp　コンストラクタ

CSingleAppオブジェクトを指定された名前で構築します。

書式 CSingleApp(LPCTSTR inMutexName);

inMutexName　ミューテックスの名前

~CSingleApp　デストラクタ

CSingleAppオブジェクトを解放します。

Initialize　初期化

多重起動検出のためのミューテックスを生成し、所有権を取得します。コンストラクタで指定された名前のミューテックスが生成済みの場合、それと関連づけられたウィンドウを復元し、falseを返します。ミューテックスの所有権が取得できない場合も多重起動とみなし、falseを返します。

書式 bool Initialize();

Return　成功：true　それ以外：false

Release　解放

ミューテックスとウィンドウの関連づけを解除し、ミューテックスを解放します。

SetHWnd　設定

指定されたウィンドウとミューテックスを関連づけます。

書式 bool SetHWnd(const HWND hWnd);

Return　成功：true　それ以外：false

hWnd　ウィンドウのハンドル

RemoveHWnd　解除

指定されたウィンドウとミューテックスの関連づけを解除します。

OnCreate　メッセージの応答

WM_CREATEメッセージに応答する場合に呼び出します。指定されたウィンドウとミューテックスを関連づけます。

書式 bool OnCreate(const HWND hWnd);

Return　成功：true　それ以外：false

hWnd　ウィンドウのハンドル

OnDestroy　メッセージの応答

WM_DESTROYメッセージに応答する場合に呼び出します。ウィンドウとミューテックスの関連づけを解除します。

m_hMutex　メンバ変数

CSingleAppオブジェクトが使用するミューテックスのハンドルです。

書式 HANDLE m_hMutex;

m_MutexName　メンバ変数

CSingleAppオブジェクトが使用するミューテックスの名前です。

書式 TCHAR m_MutexName[MAX_PATH + 1];

m_hWnd　メンバ変数

CSingleAppオブジェクトが所有するミューテックスと関連づけられたウィンドウのハンドルです。

書式 HWND m_hWnd;

(３)CSingleAppクラスのソースファイル(SingleApp.cpp)を以下のように作成しましょう。

－SingleApp.cpp－

```
/*
==========================================================================================
                        オブジェクト指向ゲームプログラミング
     Programmed by Hibikino software. Copyright (c) 2005 Hibikino software. All rights reserved.
==========================================================================================
```

```
【対象ＯＳ】
   Microsoft Windows2000/XP

【コンパイラ】
   Microsoft Visual C++ 2005

【プログラム】
   SingleApp.cpp
            シングルアプリケーションクラス

【履歴】
   * Version    1.00       2005/03/dd  hh:mm:ss

================================================================================================
*/

/*****************************************************************************************************/
/*                                          インクルードファイル                                     */
/*****************************************************************************************************/
#include ※ここは各自考えましょう

/*****************************************************************************************************/
/*                                        デフォルトコンストラクタ                                   */
/*****************************************************************************************************/
CSingleApp::CSingleApp() : m_hMutex(NULL), m_hWnd(NULL)
{
}

/*****************************************************************************************************/
/*                                          コンストラクタ                                           */
/*****************************************************************************************************/
CSingleApp::CSingleApp(LPCTSTR inMutexName) : m_hMutex(NULL), m_hWnd(NULL)
{
}

/*****************************************************************************************************/
/*                                          デストラクタ                                             */
/*****************************************************************************************************/
CSingleApp::~CSingleApp()
{
    Release();
}

/*****************************************************************************************************/
/*                                            初期化                                                 */
/*****************************************************************************************************/
bool CSingleApp::Initialize()
{
    return true;
}

/*****************************************************************************************************/
/*                                            解　放                                                 */
/*****************************************************************************************************/
void CSingleApp::Release()
{
}

/*****************************************************************************************************/
/*                                        ウィンドウ関連づけ                                         */
/*****************************************************************************************************/
bool CSingleApp::SetHWnd(const HWND hWnd)
{
    return true;
}

/*****************************************************************************************************/
/*                                      ウィンドウ関連づけ解除                                       */
/*****************************************************************************************************/
void CSingleApp::RemoveHWnd()
{
}
```

(４)デフォルトコンストラクタの足りない部分を補い、プログラムを完成させましょう。

```
CSingleApp::CSingleApp() : m_hMutex(NULL), m_hWnd(NULL)
{
    ::ZeroMemory(m_MutexName, sizeof(m_MutexName));

    // ミューテックス名をファイル名に設定
    ::????????????????(NULL, m_MutexName, sizeof(m_MutexName));

    // '\'を'/'に変更
    for(int i = 0; i < lstrlen(m_MutexName); i++) {
        if(m_MutexName[i] == '\\')
            m_MutexName[i] = '/';
    }
}
```

(５)引数付きコンストラクタを以下のように作成しましょう。

```
CSingleApp::CSingleApp(LPCTSTR inMutexName) : m_hMutex(NULL), m_hWnd(NULL)
{
    // ミューテックス名設定
    ::strncpy(m_MutexName, inMutexName, MAX_PATH);
}
```

(６)Initialize関数の足りない部分を補い、プログラムを完成させましょう。

```
bool CSingleApp::Initialize()
{
    Release();      // 初期化済みかもしれないので、解放しておく

    // ミューテックス生成
    m_hMutex = ::???????????(NULL, 0, m_MutexName);

    // ミューテックスがすでに生成されていた場合、
    // ミューテックスを所有するウィンドウを探して復元する
    if(::GetLastError() == ????????????????????) {
        HWND   hWnd = ::GetWindow(::GetDesktopWindow(), GW_CHILD);
        while(hWnd != NULL) {
            // ミューテックス名を持つウィンドウか調べる
            if(::GetProp(hWnd, m_MutexName) != NULL) {
                // 最小化されている場合は復元
                if(::????????(hWnd) != 0)
                    ::ShowWindow(hWnd, ??????????);

                ::SetForegroundWindow(hWnd);                          // フォアグラウンドに設定
                ::SetForegroundWindow(::GetLastActivePopup(hWnd));  // 子ウィンドウにフォーカスを設定

                break;
            }   // if(GetProp)

            // ミューテックス名が一致しない場合は次のウィンドウへ
            hWnd = ::GetWindow(hWnd, GW_HWNDNEXT);
        }   // while(hWnd)

        // ミューテックス解放
        ::CloseHandle(m_hMutex);
        m_hMutex = NULL;

        return false;
    }   // if(GetLastError())

    // ミューテックス所有権取得
    const DWORD   Ret = ::????????????????????(m_hMutex, 0);
    if(Ret != WAIT_OBJECT_0 && Ret != WAIT_ABANDONED)
        return false;

    return true;
}
```

(７)Release関数の足りない部分を補い、プログラムを完成させましょう。

```
void CSingleApp::Release()
{
    // プロパティ解除
    RemoveHWnd();

    // ミューテックス解放
    if(m_hMutex != NULL) {
        ::???????????(m_hMutex);    // 所有権解放
        ::???????????(m_hMutex);    // ミューテックス解放
        m_hMutex = NULL;
    }
}
```

(８)SetHWnd関数の足りない部分を補い、プログラムを完成させましょう。

```
bool CSingleApp::SetHWnd(const HWND hWnd)
{
#ifdef _DEBUG
    if(m_hMutex == NULL || hWnd == NULL)
        return false;
#endif
    if(m_hWnd == hWnd)
        return true;    // 同じウィンドウが渡された場合は、処理しない

    RemoveHWnd();       // 古いウィンドウハンドルの解放

    // ミューテックス名をウィンドウのプロパティに設定する
    if(::???????(hWnd, m_MutexName, m_hMutex) == 0) {
        ::OutputDebugString("*** Error - プロパティ設定失敗(CSingleApp_SetHWnd)\n");
        return false;
    }
    m_hWnd = hWnd;  // ウィンドウハンドルの保存

    return true;
}
```

(９)RemoveHWnd関数の足りない部分を補い、プログラムを完成させましょう。

```
void CSingleApp::RemoveHWnd()
{
    if(m_hWnd == NULL)
        return;

    // ウィンドウのプロパティに設定したミューテックス名を解除する
    ::??????????(m_hWnd, m_MutexName);
    m_hWnd = NULL;
}
```

(10)以下のプログラムを適切な場所に追加しましょう。

```
    CSingleApp   m_SingleApp;
```

ヒント：多重起動の検出は、アプリケーションの機能の一部となります。アプリケーションを表すクラスはCGameAppです。CGameAppクラスは、CSingleAppクラスを用い、多重起動検出を行います。このとき、CGameAppクラスはCSingleAppクラスを継承ではなく、集約で自身の一部とします。集約とは、別のクラスのオブジェクトが属性(メンバ変数)になっていることをいい、CGameAppクラスのメンバ変数にCSingleAppオブジェクトがある、という関係になります。

(11)(10)を追加したクラスのコンストラクタ初期化子に、以下のプログラムを追加し、CSingelAppオブジェクトが適切に初期化されるようにしましょう。なお、"名前"は作成するゲームの名前など、適切なものに変更しましょう。

```
m_SingleApp("名前")
```

ヒント：CSingleAppクラスの場合、コンストラクタ初期化子は下線部をいいます。

```
CSingleApp::CSingleApp() : m_hMutex(NULL), m_hWnd(NULL)
```

(12)多重起動かどうかを調べる以下のプログラムを適切な場所に追加しましょう。

```
// 多重起動検出
if(m_SingleApp.Initialize() == false)
    return false;
```

(13)CSingleAppオブジェクトとウィンドウを関連づける以下の追加ａまたは追加ａ’を適切な場所に追加しましょう。

－追加ａ－

```
// ウィンドウ関連づけ
m_SingleApp.SetHWnd(GetHWnd());
```

－追加ａ’－

```
// ウィンドウ関連づけ
m_SingleApp.OnCreate(hWnd);
```

(14)CSingleAppオブジェクトとウィンドウの関連づけを解除する以下の追加ｂまたは追加ｂ’を適切な場所に追加しましょう。

－追加ｂ－

```
// ウィンドウ関連づけ解除
m_SingleApp.RemoveHWnd();
```

－追加ｂ’－

```
// ウィンドウ関連づけ解除
m_SingleApp.OnDestroy();
```

(15)以下のプログラムが必要かどうか検討し、必要ならば適切な場所に追加しましょう。

```
m_SingleApp.Release();      // CSingleAppの解放
```