インターフェイスの街角 (46) — Windows のプログラミング

増井俊之

私は、文書作成やプログラミングなど、ほとんどの仕事を UNIX 上でおこなっています。ただし、最近は〝本物の UNIX〟ではなく、Windows の Cygwin 環境のような擬似 UNIX 環境もよく使います。

テキストベースのツールに関していえば、Cygwin 環境は本物の UNIX 環境とほとんど変わりありませんし、Windows 版の Emacs は UNIX 版とほぼ同じ感覚で使えます。telnet で UNIX システムにアクセスして利用するのなら、Windows マシンでも大きな違いはないでしょう。すくなくとも、テキストベースの作業をしているかぎり、Windows でも UNIX でも大差ない生活ができます。

GUI プログラムの開発についても、OpenGL/Glut や Java のように、Windows でも UNIX でもまったく同じソースコードが使えるのであれば、両者の環境の違いをそれほど意識せずに作業することができます。とはいえ、Windows のシステムに強く依存する機能は、当然のことながら上記のような方法で扱えません。

このところ、私はあらゆる環境で予測型テキスト入力システム「POBox」を使って文章を書いています。UNIX 環境や Window 版の Emacs ならこれでよいのですが、Windows のアプリケーションでテキストを入力するときは、Windows の IME を利用しなければならないのが悩みのタネでした。Windows 上でもつねに POBox を使えるようにするには、Windows のシステムの深いところに関係したプログラムを開発しなければならず、二の足を踏んでいたわけです。

しかし、今回思い立って、Windows の API を用いた POBox IME を実装してみました。ただし、Windows の〝ビジュアルな〟統合開発環境は使いたくなかったので、Cygwin を用いて UNIX ふうにアプリケーションを開発することにしました。

Windows プログラミング環境

Windows でのアプリケーション開発では、Microsoft の Visual C++や Visual Basic、あるいは Borland の C++ Builder といった統合開発環境を利用することが多いようです。

しかし、これらのシステムは高価なうえに、その環境にどっぷり浸からないと効果的な開発はできません。UNIX 上での開発に慣れた私のような人間にとって、こういった環境を使うのはかなりの苦痛です。その理由は、次のようなところにありそうです。

- 好みのエディタが使いにくい
  通常、統合環境には専用のプログラム・エディタが含まれていて、それを用いたプログラム開発が推奨されています。しかし、Emacs や vi などのエディタに慣れた人にとっては、開発環境ごとに異なるエディタを使わなければならないのは苦痛です。
- コンパイル・オプションなどの一覧やテキストでの指定が難しい
  統合環境では、オプションの指定といった作業にもメニューなどを使うのが普通です。画面設計をビジュアルにおこなえるツールが用意されているのはいいとしても、コンパイル・オプションのように、その必要がまったくないものも GUI ツールで設定するのでは、かえって一覧性が悪くなってしまいます。
- ほかのツールとの併用が難しい
  統合環境はそれ自体で閉じているため、その他のテキストベースのツールを併用したプログラム開発は容易では

図 1　WinMain() の代わりに main() を使う

```
#ifdef USEMAIN
int main(int argc, char **argv, char **env){
  STARTUPINFO StatUpInfo;
  HANDLE instance;
  HANDLE hPrevInstance;
  LPSTR lpszCmdParam;
  int cmdShow;

  GetStartupInfo(&StatUpInfo);
  instance = GetModuleHandle(0);
  hPrevInstance = 0;
  lpszCmdParam = ""; // 引数を使う場合はargcから生成
  cmdShow = (StatUpInfo.dwFlags & STARTF_USESHOWWINDOW)?
             StatUpInfo.wShowWindow:SW_SHOWNORMAL;
#else
int WINAPI WinMain(HINSTANCE instance, HINSTANCE prevInstance,
           PSTR cmdLine, int cmdShow){
#endif
  ......
```

ありません。

●プログラムが勝手に生成される

統合環境では、自動生成されるプログラムのテンプレートにプログラムを追加するかたちで開発を進めなければならない場合があります。システムによって生成された謎のコードに、自分のプログラムを追加するのはなんとなく気が進まないものです。

このように、統合開発環境は UNIX に慣れたプログラマーにとっては、あまり使いやすいものとは思えません。ただし、このような環境を使わなければ Windows のプログラムが作れないわけではありません。Cygwin を用いた開発環境で Windows のプログラミングをおこなえば、通常の UNIX プログラムの作成とほぼ同じ感覚、すなわちテキストベースの作業で Windows の GUI プログラムが作成できます。

Windows プログラムの例

まず、簡単な Windows プログラムの例をみてみましょう。ウィンドウを作成し、そこに〝Hello, World!〟を表示する Windows プログラムは、末尾のリスト 1 のようになります。

たんに文字列を表示するだけのプログラムですが、ご覧のようにかなり多くの初期設定などが必要になります。OpenGL のもとになった SGI の GL ライブラリでは、

```
prefsize(300,100);
winopen("hello");
```

とするだけでウィンドウを開くことができました。しかし、Windows では多数の属性を設定してからウィンドウの作成を指示しなければなりません。

そのほかにも、UNIX プログラマーにとっては見慣れない記法や関数呼出しが並んでいます。

ここで、UNIX プログラマーが Cygwin を用いて Windows のプログラミングをおこなう際の注意点をいくつか挙げておきましょう。これらを頭に入れておけば、それほど苦しまずに Windows プログラムが作れるのではないかと思います。

メインルーチン

Windows プログラムのメインルーチンには、main() ではなく WinMain() が使われます。Cygwin 環境では、図 1 のようにすれば、WinMain() の代わりに main() が利用できます（ただし、この場合は WinMain() への引数を適当に作成する必要があります）。main() を使えば、環境変数を引数から簡単に取得できるので、端末から起動するプログラムではこのほうが便利だと思います。

メッセージの利用

Windows のプログラムは、〝メッセージ〟にもとづいて動作します。アプリケーションは、GetMessage() でユーザーや別のプログラムなどからのメッセージを受け取ったあと、DispatchMessage() でコールバック関

数 WndProc() を呼び出すという作業を繰り返します。WndProc() は、〝ウィンドウクラス〟を登録するときに登録しておきます。

ユーザーがマウスやキーボードを操作すると、アプリケーションにメッセージが送られます。たいていのウィンドウ・システムでは、このようなイベントループはおもにユーザーの操作を処理するために使われますが、Windows ではメッセージはもっと幅広い用途に使われます。たとえば、別のウィンドウとして表現されたテキスト入力枠に文字を設定するといった処理にも利用されます。

命名規則

■ 参考文献を追加しました

リスト 1 のプログラムには、hWnd や lpfnWndProc といった変数名やメンバー名が出てきます。これは、〝ハンガリアン命名規則〟(Hungarian Notation)[1] という手法にもとづくもので、型を示す h や lp などの略号に続けて名前を記述します。たとえば、hWnd は〝ウィンドウのハンドル〟を意味します。これはたんなる名前なので、このようにしなければプログラムが作れないわけではありません。しかし、Windows API のマニュアルやサンプル・プログラムは、たいていはハンガリアン命名規則にもとづいて記述されているため、Windows プログラムではこの規則を踏襲するほうが無難なようです[1]。

標準ライブラリ

一般に、Windows プログラムでは独自の API を用いてファイル操作や通信をおこないます。一方、Cygwin には fopen() や fseek() といった標準ライブラリ、socket() などの通信ライブラリが用意されているので、これらについては UNIX とまったく同じ感覚でプログラムが作れます。

描画

Windows では、画面への描画に GDI (Graphics Device Interface) という API を使用します。アプリケーションのウィンドウなどの出力デバイスごとに、その状態を保持するデバイス・コンテクスト (Device Context) が対応づけられており、これらに対して文字列描画や塗潰しなどの GDI 関数を呼び出すことができます。リスト 1 のプログラムでも、現在のウィンドウのデバイス・コンテクストに対して〝Hello, World!〟という文字列を出力しています。

1 私は、このように子音の並んだ省略形にどうしても馴染めません。変数が自明ならば h などでよいでしょうし、そうでなければ windowHandle などとしたほうが分かりやすいように思います。欧米の人は子音のみからなる文字列に抵抗がないのでしょうか。

図 2　Makefile

```
#
#       $Date: 2001/08/10 14:37:37 $
#       $Revision: 1.1 $
#
LIBS=-luser32 -lgdi32

hello: hello.o
        $(CC) -o hello hello.o $(LIBS)
```

図 3　リソース定義ファイルと Makefile

● **hello.rc**

```
TEST_ICON ICON test.ico
```

● **Makefile**

```
hello: hello.o
        $(CC) -o hello hello.o $(LIBS)
resource.o: hello.rc hello.ico
        windres -i pobox.rc -o resource.o
```

コンパイルとリンク

Cygwin では、CreateWindow() のような基本的な Windows API は libuser32.a に、描画ライブラリは libgdi32.a に用意されています。そして、これらを経由して Windows の user32.dll や gdi32.dll などを呼び出す仕組みになっています。したがって、Makefile もごく普通に記述できます (図 2)。

リソース

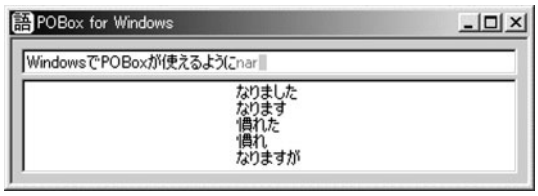
Windows では、メニューのような GUI 部品やアイコンなどの設定は、プログラム中に記述するのではなく、リソースファイルとしてプログラムとは独立に設定できます。

Visual C++などではリソースをビジュアルなツールを使って定義するのが普通ですが、Cygwin 環境ではテキスト形式で定義し、リソース・コンパイラ (windres) を用いて作成することができます (図 3)。

端末からの切離し

Cygwin で普通に作成したアプリケーションを Windows のデスクトップなどでクリックすると、Cygwin のシェル窓が開き、その後にアプリケーションが起動します。GUI アプリケーションではこのような動作は邪魔ですが、リンク・オプションに〝–mwindows〟を追加する

図 4　POBox for Windows



とシェル窓を開かずにアプリケーションを起動することができます。

■ ■ ■

これらの点を頭に入れておけば、Windows の GUI アプリケーションも UNIX とほぼ同じ感覚で開発できるのではないでしょうか。Makefile やリソースファイルなど、開発に必要なすべてのファイルも、UNIX 上のプログラム開発と同様にテキストベースで編集できます。

POBox for Windows

以上に紹介した方法を用いて、POBox for Windows を作ってみました。Windows では IME を実装するための API が規定されているので、本来はこれに従って実装すべきですが、IME の API はかなり複雑で扱いにくいため、今回は下記の単純な方針に従って実装しました。

- POBox for Windows はアプリケーションとして実装する。
- ユーザーのキー入力をすべて捕捉し、ウィンドウ内で予測変換をおこなったあと、確定文字列をアプリケーションに送る。
- アプリケーションに送るときは、WM_CHAR メッセージ（ユーザーのキー入力に対応して発生するメッセージ）を使う。

この方式では、カーソル位置でのインライン変換はできませんが、とりあえず予測式入力は可能です。

図 4 が、POBox for Windows のウィンドウです。ローマ字で単語を入力していくと下に候補のリストが表示され、スペースキーで選択することができます。

POBox for Windows の実行ファイルとソースコードは私の Web ページ[2]で公開しています。

2 http://www.csl.sony.co.jp/person/masui/OpenPOBox/Windows/

Windows プログラミングの楽しみ

できることなら、Windows 上でのプログラミングとは無縁な生活を送りたいと思っていましたが、IME などを実装しようとするとそうもいきません。そのような場合も、Cygwin を使えば UNIX の流儀に沿った開発ができるようになったのは喜ばしいことです。

どのような GUI ツールキットでもそうですが、Windows の API は数が膨大なうえに機能がよく分からないものが多く、何もないところからプログラムを書くのは至難の業です。しかし、さいわいなことに現在は Web 上に多くのサンプルコードや解説ページが公開されているため、たいていは簡単に検索できます。API について調べる場合など、マニュアルを読むより、Web で検索したサンプルコードを眺めたほうがよく理解できることさえあります。うまくいけば、Web 上のプログラムリストを切り貼りするだけで目的とするプログラムが作れる可能性もあるでしょう。もちろん時と場合によりますが、Windows ではこのような ˝Web-based Programming˝がもっとも有効なプログラム開発環境なのかもしれません。

Windows 特有の機能をどうしても使わなければならない場合以外は、やはりどこでも共通に使えるライブラリや言語を利用するほうがよいでしょう。たとえば、UNIX でも Windows でも同じように動く 3 次元視覚化プログラムを作るのなら、これらのプラットホームで共通に扱える OpenGL/Glut などのライブラリを利用するのが得策です。その場合も、今回紹介した windres を用いてアイコンを設定したり、–mwindows オプションを付けてリンクすれば、通常の Windows アプリケーションと同じ挙動をするプログラムが作れます。

（ますい・としゆき　ソニー CSL）

[参考文献]

[1] Steve C. McConnell, *Code Complete : A Practical Handbook of Software Construction*, Microsoft Press, May 1993（邦訳：『コードコンプリート』（石川 勝訳）、アスキー、1994 年）

リスト 1　Hello, World プログラム

```
//
//      $Revision: 1.1 $
//      $Date: 2001/08/10 14:37:37 $

#include <windows.h>
#include <stdio.h>

LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE prevInstance,
                   PSTR cmdLine, int cmdShow)
{
    MSG uMsg;
    HWND hWnd;
    WNDCLASSEX wndclass;

    //ウィンドウのスタイルを登録
    wndclass.cbSize = sizeof(wndclass);
    wndclass.style = CS_HREDRAW | CS_VREDRAW;
    wndclass.lpfnWndProc = WndProc;
    wndclass.cbClsExtra = 0;
    wndclass.cbWndExtra = 0;
    wndclass.hInstance =  hInstance;
    wndclass.hIcon = NULL;
    wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
    wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wndclass.lpszMenuName = NULL;
    wndclass.lpszClassName = "test";
    wndclass.hIconSm = NULL;
    RegisterClassEx(&wndclass);

    hWnd = CreateWindowEx(
               WS_EX_OVERLAPPEDWINDOW,
               "test",
               "Test Window",
               WS_OVERLAPPEDWINDOW,
               CW_USEDEFAULT,
               CW_USEDEFAULT,
               300,
               100,
               NULL, // parent
               NULL, // menu
               hInstance,
               NULL // lpvParam
               );

    ShowWindow(hWnd, cmdShow);
    UpdateWindow(hWnd);

    while(GetMessage(&uMsg, NULL, 0,0)) {
            TranslateMessage(&uMsg);
            DispatchMessage(&uMsg);
    }

    return uMsg.wParam;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg,
                         WPARAM wParam, LPARAM lParam) {
    switch(uMsg) {
    case WM_PAINT:
            Display(hWnd);
            return 0;
    case WM_DESTROY:
            PostQuitMessage(0);
            return 0;
    }
    return DefWindowProc(hWnd, uMsg, wParam, lParam);
}

int Display(HWND hWnd)
{
    PAINTSTRUCT ps;
    HDC hDC;
    hDC = BeginPaint(hWnd, &ps);
    SetBkMode(hDC,TRANSPARENT);
    TextOut(hDC,80,25,"Hello, World!",13);
    EndPaint(hWnd, &ps);
    return 0;
}
```