筑波大学技術報告
24: 19-21, 2004

Unix/Windows 一石二鳥プログラミング

木村博美[1]
筑波大学　人文・数理等教育研究支援室（加速器センター）
〒305-8577 茨城県つくば市天王台 1-1-1

概要

Unix/Windows 混在環境では、両方の環境で動作するようなプログラムを作成することが望まれます。幸い、最近ではそのような要求に応えるツールが簡単に入手できます。この報告ではそのようなツールを使った実例をいくつか紹介します。

１．はじめに

ここで言う一石二鳥とは、一つのプログラムがそのままあるいは再コンパイルするだけで Unix と Windows 上で同じ様に使用できることを指します。その為には、両環境の違いを吸収する何らかの仕掛けが必要ですが、幸い、現在ではインターネットからそのようなツールを容易に入手できます。本稿では、スクリプト言語、Java 言語、ライブラリの使用という３つの方法を紹介します。

２．スクリプト言語の使用

Perl に代表されるスクリプト言語の多くは Unix/Windows 両環境で動作します。一石二鳥プログラミングにおいてスクリプト言語を使用する最大の利点は、開発サイクルの短縮でしょう。つまり、プログラムの作成・修正とテストの繰り返しを短時間で実行することができます。また Windows の開発環境が不要である点も有利でしょう。更に、スクリプト言語の多くは追加モジュールで機能を追加することができます。しかし環境依存になってしまう恐れがあるので、注意が必要です。

欠点は、処理速度の遅さとメモリ消費量の多さでしょう。

2.1 スクリプト言語の紹介

Perl は最も良く使われている言語でしょう。追加モジュールも豊富で、万能と言っても良いでしょう。書籍やインターネットで得られる情報も豊富です。処理速度もスクリプト言語の中では速い方です。ただ、プログラムは読みにくくなりがちです。

Perl は後付けでオブジェクト指向の機能を追加しましたが、Ruby や Python は最初からオブジェクト指向の言語として設計されました。そのため、Perl に比べるとずっと読みやすいプログラムを書くことができます。ただ、Perl に比べると処理速度は遅いようです。

Tcl/Tk はスクリプト言語の中で唯一標準で GUI をサポートする言語です。ただ、言語仕様が貧弱なため、他の言語と組み合わせて使用されることが多いようです。

2.2 実例

当センターでは実験装置から計算機へのデータ転送／形式変換プログラムを Perl で作成しました。この実験装置はデータの取り出し口が RS-232-C しか無いため、ターミナルサーバを利用して、ネットワーク経由でデータを取り出せるようにしてあります（図１）。PC 上のプログラムは TCP/IP でターミナルサーバと通信し、取り出したデータをユーザの指定した形式に変換してファイルに書き出すという単純な仕様です。そのため GUI は必要ありませんでした。

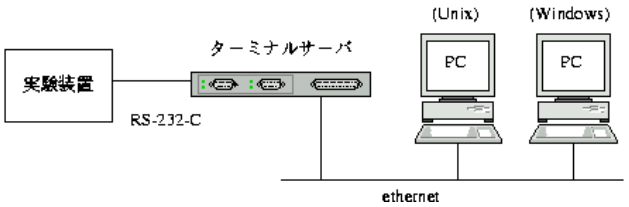

図１．実験装置の接続

```
$vs = "";
vec($vs,fileno(SOCKET1),1) = 1;
vec($vs,fileno(SOCKET2),1) = 1;
$in1 = $in2 = 0;
$timeout = 2;
$minsize = 200;
while(1){
    $nin = select($vsout=$vs,undef,undef,$timeout);
    if($nin < 0){ die "select: $!"; }
    if($nin > 0){
        if(vec($vsout,fileno(SOCKET1),1)){
            sysread(SOCKET1,$rbuf,$rsize)|| die "read: $!";
            $n = length($rbuf);
            @buf1[$in1 .. $in1+$n-1] = unpack "C$n",$rbuf;
            $in1 += $n;
        }elsif(vec($vsout,fileno(SOCKET2),1)){
            sysread(SOCKET2,$rbuf,$rsize)|| die "read: $!";
            $n = length($rbuf);
            @buf2[$in2 .. $in2+$n-1] = unpack "C$n",$rbuf;
            $in2 += $n;
        }
    }else{                          # timeout
        last if ($in1 > $minsize || $in2 > $minsize);
    }
}
```

図２．スクリプト（一部）

[1] http://www.tac.tsukuba.ac.jp/~hiromi/

実際のスクリプトの一部を図２に示します。装置を2台まで使用できるようにするためにselectで入力を待ち、読み込んだデータをそれぞれのバッファに蓄えます。入力終了はselectのタイムアウトで判断し、その後でバッファを解析し、データを変換します。

このプログラムは、かつてミニコンを使っていた頃は FORTRAN で書き（装置はミニコン本体に接続）、Unix に切替えた時は C で書きましたが、Windows にも対応する必要から Perl で再度書き直しました。なお、Windows 版 Perl には ActivePerl を使用しました。

３．JAVA 言語の使用

Java の大きな特徴はGUIも標準ライブラリに含まれている点です。ただし、古いバージョンの AWT と新しい Swing の２つあるので、移植性を考えるとAWT を使用することになります。最近では統合開発環境 Eclipse の SWT(Standard Widget Toolkit)を使うという方法もあります。

一石二鳥プログラミングでの利点は、Java は中間コード方式なので、コンパイルして作成した jar ファイルをプラットホームに関係無く使用できるという点です。

3.1 実例

当センターではデータ表示解析プログラムの１つを Java で書きました（図３）。元々は FORTRAN で書かれたプログラムですが、GUI ライブラリが可搬性のないものだったので、Windows への対応のために Java で書き直しました。このプログラムはデータファイルを読み書きするので、ブラウザに表示する Java applet ではなく、単独アプリケーションとして作成しました。GUI は AWT を使用しました。

このプログラムは実験データのファイルを読み込み、画面に表示した後、マウス操作でデータの切り出しとファイルへの書き込みを行います。

Java のおかげで Unix/Windows だけでなく MacOS でも使用できました。更にグラフィクスの印刷機能も 簡単に追加できました。ただし、メモリと CPU を要求するのが欠点です。

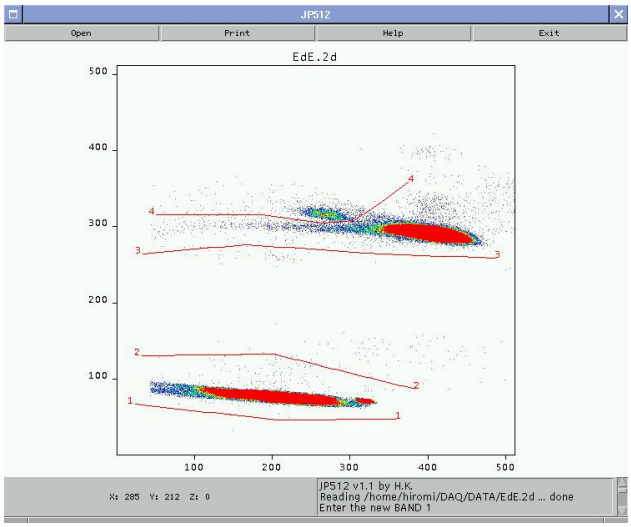

図３．Java で書いたプログラムの動作画面

開発は Unix に複数の JDK（1.1.8、1.2.2、1.3.1）を入れ、それぞれで動作を確認してから Windows で最終的な動作確認をしました。

４．ライブラリの使用

一石二鳥プログラミングに使用可能なライブラリもいくつか存在しますが、プラットホーム毎にライブラリを用意し、個別にコンパイルしなければならないのが欠点です。しかし、出来上がったアプリケーションは最も効率良く動作します。

4.1 ライブラリの紹介

GTK+ は画像処理プログラム GIMP を開発する過程で作成された C 言語用の GUI ライブラリです。Unix ではデスクトップ環境の GNOME をはじめ広く使われており、Windows にも移植されています。GUI 画面の設計は Unix 上で Glade を使用します。

C++用では Qt が最も優れています。デスクトップ環境の KDE に使用されている他、ボーランド社の製品である Delphi/Kylix のライブラリとしても採用されています。Unix 版はフリーですが、Windows 版は教育用のみフリーで使用できます。GUI 画面の設計は Qt Designer で行います。

Unix/Windows 共通の統合開発環境が必要なら WideStudio という選択もあります。Windows 版には GCC も含まれていますので、インストールの手間も掛かりません。

4.2 実例

当センターではネットワーク接続された装置の制御・データ読み出しプログラムに GTK+を使用しています（図４）。ただし、GTK+ にはネットワークライブラリは含まれていないので、Unix の socket と Windows の winsock を呼び出すラッパーを作る必要がありました（図５）。

このプログラムは画面の左半分にデータを、右半分に装置の設定を常時表示することで、ユーザの利便性を高めています。また、ネットワークライブラリを使ってプロセス間通信を行い、複数のプログラムの協調動作を実現しています。

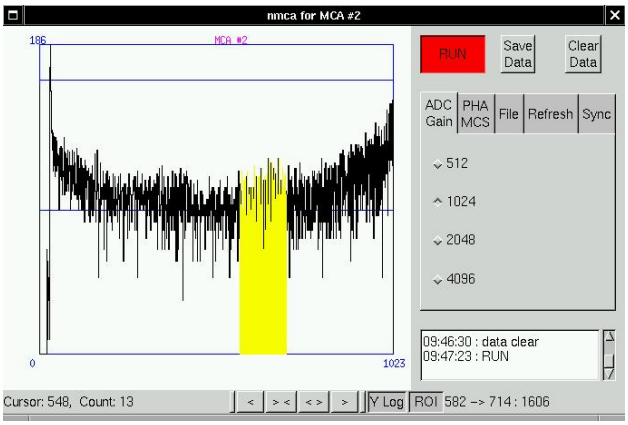

図４．GTK+を使用したプログラムの動作画面

使用言語は C で、Unix では GCC、Windows では Visual C++でコンパイルしました。画面設計は Galde を使用しました（図６）。

```
int
Net_udp_client (char *server, int port,
                struct sockaddr_in*serv_addr)
{
   struct sockaddr_in cli_addr;
   struct hostent *hp;
   int sock;

#ifdef WIN32
   WORD vreq;
   WSADATA wsadata;
   /* init winsock */
   if(!init){
      vreq = MAKEWORD(1,1);
      if(WSAStartup(vreq,&wsadata) != 0){
         return NG;
      }
      atexit((void(*)(void))(WSACleanup));
      init = 1;
   }
#endif

   hp = gethostbyname(server);
   if(hp == NULL) return NG;
   memset((char *)serv_addr, 0, sizeof(struct sockaddr_in));
   serv_addr->sin_family = AF_INET;
   memcpy((char *)&(serv_addr->sin_addr),
            hp->h_addr,hp->h_length);
   serv_addr->sin_port = htons(port);

   if((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0){
      return NG;
   }
```

図５．ネットワークライブラリ（一部）

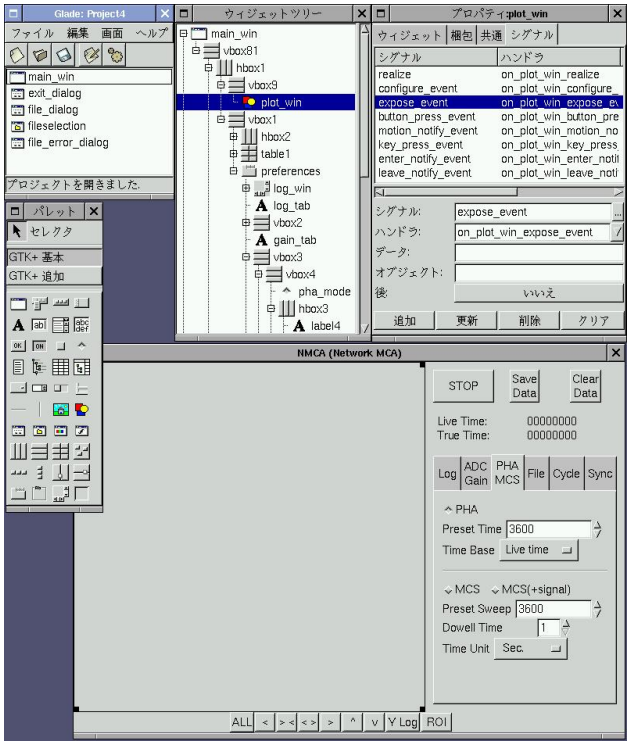

図６．Glade による画面設計

参考文献

[1] Perl
URL: http://www.perl.com/
URL: http://www.perl.org/
[2] Ruby
URL: http://www.ruby-lang.org/ja/
[3] Python
URL: http://www.python.org/
[4] ActivePerl
URL: http://www.activestate.com/Products/ActivePerl/
[5] Tcl/Tk
URL: http://tcl.sourceforge.net/
[6] Eclipse
URL: http://www.eclipse.org/
[7] The GUI Toolkit, Framework Page
URL:http://www.geocities.com/SiliconValley/Vista/7184/guitool.html
[8] GTK+
URL: http://www.gtk.org
[9] GIMP
URL: http://www.gimp.org/
[10] GTK+ for Win32
URL: http://www.gimp.org/~tml/gimp/win32/
[11] GNOME
URL: http://www.gnome.org/
[12] Qt
URL: http://www.trolltech.com/products/qt/index.html
[13] KDE
URL: http://www.kde.org/
[14] Delphi/Kylix
URL:http://www.borland.co.jp/
[15] WideStudio
URL: http://www.widestudio.org/
[16] Glade
URL: http://glade.gnome.org