A Windows driver program written in Forth

N.J. Nelson

Abstract

We have a company policy of writing all our software in Forth, from tiny programs for 8 pin PIC chips, up to major networked Windows applications. There was just one area where we were still obliged to use "C", and that was driver programs for interfacing between Windows and hardware. A solution has now presented itself, which works in 95% of cases.

N.J. Nelson B.Sc., C.Eng., M.I.E.E.
Micross Electronics Ltd.,
Units 4-5, Great Western Court,
Ross-on-Wye, Herefordshire.
HR9 7XP U.K.
Tel. +44 1989 768080
Fax. +44 1989 768163
Email. njn@micross.co.uk

Introduction to driver programs

As you browse through the Windows directory of your Personal Computer, you will come across lots of small files with extensions such as .drv or .sys. For example, on a Windows 2000 PC running our commercial laundry automation system "Tracknet", in the directory \WINNT\System32\Drivers, there will be a file called Vplc.sys. The purpose of this program is to act as an interface between the Tracknet program and a PCI circuit card called the "Virtual Programmable Logic Controller" which does the actual control of the whole laundry.

There are two reasons why we need this interface program. First, the PCI system has dynamically assigned addresses, so the Tracknet program doesn't know where to read and write when it needs to access the I/O or memory space of the circuit card. Second, even if it knew, it wouldn't be allowed to. In Windows 2000, applications programs are not allowed to access directly the I/O space or fixed memory space.

There are similar interface programs for every hardware facility on the PC. In addition to resolving dynamic resource assignments, and providing protection of the hardware from badly behaved programs, they have a further key function in Windows, which is that of hardware abstraction. For example, when an applications program wants to write a character to a serial communications port, it does not need to know what sort of a UART is fitted to the port. It simply calls a Windows Applications Programming Interface function, which in turn calls the correct driver program for the selected device.

In addition, driver programs are necessary to implement the "Plug and Play" system, in which hardware is identified automatically as it is added to the PC, and the corresponding software to support it is loaded.

The Vplc.sys, and another similar program for interfacing with the Data Logger Card on our "Rabit" microbiological testing system, are the only two programs by Micross that are not written in Forth.

Types of driver programs

There are three types of driver programs that an applications programmer may need to get involved with. The older Virtual Device Drivers have the extension .vxd and can be used with Windows 95 and 98. Windows NT required a completely different type of driver program with the extension .sys. This meant that all hardware manufacturers had to provide two programs for every device. To overcome this problem, Microsoft came up with the "Windows Driver Model" or WDM specification. This also has the extension .sys and is in some ways a development of the NT driver model. However, WDM drivers can be used only on Windows 2000 and 98 systems and their successors ME and XP.

How Microsoft think driver programs should be written

To assist programmers in writing drivers, Microsoft have provided a succession of Device Driver Kits or DDKs. If you have ever wondered why your printer or modem didn't work properly straight out of the box, and you had to download umpteen bug fixes from the manufacturer's website, then don't immediately come to the conclusion the Hewlett-Packard et. al. employ incompetent software engineers. Simply try examining a Microsoft DDK.

To add insult to injury, the DDK library files support only Microsoft C. There are not even any C++ abstractions supplied. (This did not stop Microsoft from issuing one version of their Visual C/CPP with a critical bug that prevented all driver programs from compiling.)

More practical ways of writing driver programs

It is not a practical proposition for even the most experienced Windows programmer to write a driver program using only the documentation in the DDK. The books written by a handful of specialist practitioners are an absolute necessity. All of these books recommend the need for additional third-party tools. Most of these work by providing C libraries or C++ classes. These either completely hide the more esoteric requirements of the driver specifications, or at least provide intelligibly documented interfaces.

Why would we want to use Forth

We successfully wrote two driver programs to the VxD specification, and subsequently updated both of them to the WDM specification, using C and with the help of the DriverStudio tool from Compuware Corporation. However, we require on average only one new driver program per year, so there is no accumulation of experience in the company. Every new program has a new learning curve. Clearly, the ability to write drivers using our normal programming language, with which everyone in the company is familiar, would be a great advantage.

The "Windriver" system

About six months ago, we became aware of a new concept in driver program development, Windriver, by Jungo Ltd., an Israeli company. The concept behind this program is that of a completely universal driver, which is configured by and interacts with ones own application in user mode, either directly or via a dynamic link library (DLL). Not only does this completely eliminate the need for the Microsoft DDK, it also means that the interface is language independent. Jungo even provide a set of wizards for generating interface code, although these generate only C. However, they do also provide samples in other languages.

A simple device

We decided to try this concept first on a very simple interface card. This "legacy" ISA card is intended to count pulses on a set of up to 16 industrial type 24V input signals, and has a small amount of on-board intelligence so as to eliminate the need for reliable polling from the Windows program to avoid losing counts. (Reliable timed operations cannot be achieved in Windows in user mode.) The interface occupies a single byte in the I/O space of the PC, with the applications program writing a command byte and the card replying with count or status data. In Windows 9x, this could be achieved directly from the applications program. In Windows 2000, however, the I/O space of the PC is protected from applications programs, and a device driver must be used.

Implementing a Windriver interface in Forth

Accessing the Windriver system using Forth proved to be quite straightforward, using the sample code provided in several other languages. I would be happy to supply source code for all the standard Windriver functions to anyone interested. Briefly, this contains

a) The required Windriver constants
b) The I/O control codes, generated according to the defined Windows formula, e.g.

```
38200 CONSTANT WD_TYPE \ File type of Windriver

: WD_CTL_CODE ( wFuncNum---n ) \ Construct IO control codes for Windriver
  WD_TYPE SWAP METHOD_NEITHER FILE_ANY_ACCESS CTL_CODE ;

$901 WD_CTL_CODE CONSTANT IOCTL_WD_DMA_LOCK
$902 WD_CTL_CODE CONSTANT IOCTL_WD_DMA_UNLOCK
$903 WD_CTL_CODE CONSTANT IOCTL_WD_TRANSFER
```

c) The required Windriver structures, e.g.

```
STRUCT WD_CARD                                  \ Card details
  DWORD                           WC.DWITEMS    \ Number of items
  WD_CARD_ITEMS WD_ITEMS ARRAY-OF WC.ITEM       \ Details of each item
END-STRUCT

STRUCT WD_CARD_REGISTER                         \ Card registration details
  WD_CARD         FIELD WCR.CARD                \ Card to register
  DWORD                 WCR.FCHECKLOCKONLY      \ Only check if card is
                                                \ lockable, return hCard=1
                                                \ if OK
  DWORD                 WCR.HCARD               \ Handle of card
  DWORD                 WCR.DWOPTIONS           \ Should be zero
  32              FIELD WCR.CNAME               \ Name of card
  100             FIELD WCR.CDESCRIPTION        \ Description
END-STRUCT

STRUCT ISACARD_STRUCT                           \ Card details
  INT                   CCS.HWD                 \ Windriver handle
  WD_CARD_REGISTER      FIELD CCS.CARDREG       \ Card registration data
END-STRUCT
```

d) The basic functions for opening and closing the access to the driver program

```
: WD-OPEN ( ---hwd ) \ Open Windriver
  Z"" \\.\WINDRVR"
  GENERIC_READ
  FILE_SHARE_READ FILE_SHARE_WRITE OR
  NULL
  OPEN_EXISTING
  FILE_FLAG_OVERLAPPED
  NULL
  WINCREATEFILE
;

: WD-CLOSE { hwd -- } \ Close Windriver
  hwd WINCLOSEHANDLE DROP
;
```

e) The generalised interface function

```
VARIABLE WD-OUTCOUNT    \ Place to put bytes returned

: WD-FUNCTION { wfuncnum hwdin pparam dwsize fwait | hwd -- rc }
\ Generalised IO function call
  fwait IF WD-OPEN ELSE hwdin THEN -> hwd      \ Use input handle, or get
                                               \ new, according to waitflag
  hwd INVALID_HANDLE_VALUE = IF                \ Invalid handle
    -1                                         \ Error exit
  ELSE                                         \ Handle OK
    hwd wfuncnum                               \ hdevice,dwiocontrolcode
    pparam REL>ABS dwsize                      \ lpinbuffer,ninbuffersize
    NULL 0                                     \ lpoutbuffer,noutbuffersize
    WD-OUTCOUNT REL>ABS NULL                   \lpbytesret,lpoverlapped
    WINDEVICEIOCONTROL
    fwait IF hwd WD-CLOSE THEN                 \ If a wait function,
                                               \ close instance
  THEN
;
```

f) The individual command functions, e.g

```
: WD-CARDREGISTER { hwd pCard -- }
  IOCTL_WD_CARD_REGISTER hwd pCard WD_CARD_REGISTER FALSE WD-FUNCTION DROP
;

: WD-TRANSFER { hwd pTransfer -- }
  IOCTL_WD_TRANSFER hwd pTransfer WD_TRANSFER FALSE WD-FUNCTION DROP ;
```

Using the Forth Windriver to access the simple ISA card

A single instance of the Windriver can be used to provide access to any number of ISA, PCI or other types of card. The example below is for the simplest possible card as described above.

a) Necessary constants and variables

```
1 CONSTANT ISACARD_TOTAL_ITEMS     \ Number of items

1    CONSTANT ISACARD_IORange0_BYTES

: ISACARD_IORange0_ADDR ( ---addr ) \ Address of counter card
  CTRADDR @       \ Actually comes from the registry
;

VARIABLE HISACARD \ "Handle" of "card"
                  \ - actually base address of card group data
```

b) Describing the card to the Windriver

```
: ISACARD-SETCARDELEMENTS { | pitem -- } \ Set card item data
  ISACARD_TOTAL_ITEMS HISACARD @
  CCS.CARDREG WCR.CARD WC.DWITEMS ! \ Set number of items
  HISACARD @ CCS.CARDREG WCR.CARD
  0 WC.ITEM -> pitem                \ Calculate address of item details
  ITEM_IO pitem WI.ITEM !           \ Set item type
  TRUE pitem WI.FNOTSHAREABLE !     \ Set not shareable flag
  ISACARD_IORange0_ADDR pitem
  WI.I WI.IO WIO.DWADDR !           \ Set base address
  ISACARD_IORange0_BYTES pitem
  WI.I WI.IO WIO.DWBYTES !          \ Set range
;
```

c) Opening access to the card

```
: ISACARD-OPEN { | ver[ WD_VERSION ] -- f }
\ True if address of card structure placed in phisacard
  HISACARD OFF                      \ Assume failure
  ISACARD_STRUCT ALLOCATE 0= IF     \ Memory allocated for card details
    HISACARD !
    HISACARD @ ISACARD_STRUCT ERASE \ Clear card structure
    WD-OPEN HISACARD @ CCS.HWD !    \ Try to open Windriver
    HISACARD @ CCS.HWD @
    INVALID_HANDLE_VALUE <> IF      \ Windriver opened OK
      ver[ WD_VERSION ERASE         \ Clear version structure
      HISACARD @ CCS.HWD @
      ver[ WD-VERSION               \ Get version information
      ver[ WV.DWVER @ WD_VER U>= IF \ Version number OK
        ISACARD-SETCARDELEMENTS     \ Set card item data
        HISACARD @ CCS.CARDREG
        WCR.FCHECKLOCKONLY OFF      \ Clear check lock only flag
        HISACARD @ CCS.HWD @
        HISACARD @ CCS.CARDREG
        WD-CARDREGISTER             \ Attempt to register card
```

```
        HISACARD @ CCS.CARDREG
        WCR.HCARD @ 0<> IF          \ Card registered OK
          TRUE                      \ Success
        ELSE                        \ Card failed to lock
\ ide-drivers cr ." Card failed to lock"
          FALSE
        THEN
      ELSE                          \ Incorrect Windriver version
\ ide-drivers cr ." Incorrect Windriver version"
        FALSE
      THEN
    ELSE                            \ Failed to open Windriver
\ ide-drivers cr ." Failed to open Windriver"
      FALSE
    THEN
  ELSE                              \ Failed to allocate memory
\ ide-drivers cr ." Failed to allocate memory"
    DROP FALSE
  THEN
  DUP FALSE = IF                    \ Something went wrong
    ISACARD-CLOSE                   \ Close everything
  THEN
;
```

d) Transferring data to and from the card

```
: CCARD-TRANSFER { pbuf fread | trans[ WD_TRANSFER ] -- }
\ IO transfer on counter card
  trans[ WD_TRANSFER ERASE
  fread IF RP_SBYTE ELSE WP_SBYTE THEN
  trans[ WT.CMDTRANS !
  ISACARD_IORange0_ADDR trans[ WT.DWPORT !
  TRUE trans[ WT.FAUTOINC !
  ISACARD_IORange0_BYTES trans[ WT.DWBYTES !
  pbuf REL>ABS trans[ WT.DATA !
  HISACARD @ CCS.HWD @ trans[ WD-TRANSFER
;
```

e) Reading and writing the card

These words replace the direct port I/O words used in the Windows 9x versions of the applications program.

```
: CCARD@ { | pbuf[ 1 ] -- byte } \ Read data from counter card
  HISACARD @ pbuf[ TRUE CCARD-TRANSFER
  pbuf[ C@
;

: CCARD! { byte | pbuf[ 1 ] -- } \ Write data to counter card
  byte pbuf[ C!
  pbuf[ FALSE CCARD-TRANSFER
;
```

More sophisticated systems – Forth within a driver

There is one serious restriction to the method described above. Each I/O transaction requires a user mode function. Even when handling a hardware interrupt, the Windriver will simply queue a request to be serviced later by the applications program. This greatly restricts the usefulness of the driver in machine control applications such as a soft Programmable Logic Controller (PLC) where deterministic behaviour is required.

Recently Jungo tried to address this issue by providing a duplicate set of interface functions, but designed for use in kernel mode. The idea is to debug the basic operation of your program in user mode, then recompile the time-critical sections in kernel mode, with no essential change in the source code. The complex parts of the interface continue to be handled by the standard Windriver, with the card-specific functions only in a dedicated driver file, running at full kernel-mode priority.

For example, it would be possible to respond to an interrupt by creating a fully interactive instance of Forth, capable of being called every 10us.

Summary

The Windriver system offers the possibility of opening one of the last areas of software development previously inaccessible to Forth.

References

http://www.jungo.com/windriver
Full information on Windriver, including functional demonstration for download.

http://www.compuware.com/products/numega/drivercentral/driverstudio.htm
The next most practical way to write a driver, using C++.

http://www.microsoft.com/ddk/
Paradise for nerds.

"Writing Windows VxDs &Device Drivers" by Karen Hazzah
2nd ed. 1997 CMP Books, ISBN 0879304383 $49.95
The best guide to the old style 9x drivers

http://www.oneysoft.com
Homepage of Walter Oney, leading consultant on the newer WDM style drivers, including review of his book "Programming the Windows Driver Model", which is essential reading. Also essential are the downloadable errata.