Win64® Device Driver Porting

Dely Sy
Software Engineer
ESG-SST
Intel Corp.

February 15-17, 2000

Intel Developer Forum Spring 2000

Intel Labs

Agenda

- Tools for Win64 Device Driver Development
- Win64 Device Driver Porting Process
- Case Studies on Win64 Device Driver Porting
- Debugging Win64 Device Drivers

Tools for Win64 Device Driver Development

- Win64 Software Development Kit (SDK)
- Win64 Driver Development Kit (DDK)
- Hardware Platform
  - IA-32 machine to run Win64 SDK & DDK
  - Prototype Itanium™ processor platform to run 64-bit Windows®

Tools

Win64 SDK

- Components
  - C/C++ and Fortran Compilers
  - Remote Application Debugger
  - Intel Assembler
  - 64-bit Windows libraries, header files, runtime libraries
  - 64-bit Windows OS

Tools

Win64 SDK (contd.)

- System Requirements
  - IA-32 machine
    - 350 MHz Pentium® II processor
    - 64 MB RAM
    - 1 physical disk (minimum 4 GB)
  - Microsoft Windows 2000

Tools

Win64 DDK

- Components
  - C/C++ Compiler
  - 64-bit Windows libraries, headers, and sample source files
  - 64-bit Kernel Debugger

Tools

Win64 DDK (contd.)

- System Requirements
  - IA-32 machine
    - At least 64 MB RAM. 128 MB of RAM or more is recommended
    - Up to 200 MB hard-disk space for a full installation. Up to 750 MB may be required to compile all samples
  - Microsoft Windows 2000
  - Microsoft Visual C++ 6.0

Tools

Win64 DDK (contd.)

- Installing the DDK
  - Current version
    - Manually copy the entire contents of the CD into a new directory on the hard drive
  - Future version
    - Setup program will install the DDK tools, headers, libraries, and source files that are necessary for driver development

Tools

Win64 DDK (contd.)

- Creating the build environment
  - Go to %NTDDK%\bin directory
  - Type “setenv64.bat DDK_DIRECTORY [free\checked]”
    - where DDK_DIRECTORY is the full path of the location of Win64 DDK

Tools

Win64 DDK (contd.)

- To build the device driver
  - Go to your device driver directory
  - Type “build -c”

Tools are easy to use!

Win64 Device Driver Porting Process

- General IA-64 considerations
- 64-bit Driver source code clean-up
- 32-bit IOCTL support on Win64
- Regression test on the code clean driver
- Installing a device
- Setup to run 64-bit Windows

Porting Process

General IA-64 Considerations

- Uniform Data Model
  - Same source runs on both 32-bit and 64-bit systems
- LLP64 (or P64) Data Model
  - Only pointers expand to 64 bits
  - All other basic data types (integer and long) remain 32 bits long

Porting Process

General IA-64 Considerations

- New Data Types
  - Fixed-precision data types
    - Same length in both Win32 and Win64 programming; e.g. INT32, INT64
  - Pointer-precision types
    - As the pointer precision changes, these data types reflect the precision accordingly; e.g. UINT_PTR, ULONG_PTR

Porting Process

General IA-64 Considerations

- Specific-precision pointers
  - New pointer types that explicitly size the pointer; e.g. POINTER_32, POINTER_64

Porting Process

General IA-64 Considerations

- Predefined Macros
  - To identify platform
    - _WIN64, _WIN32
  - For architecture-specific code
    - _M_IA64, _M_IX86

Porting Process

General IA-64 Considerations

- 64-bit Compiler Switches and Warnings
  - The -Wp64 -W3 switch enables the warnings for truncation, conversion to bigger-size, passing zero length etc.

Porting Process

Win64 Driver Source Code Clean-up

- Use new data types
- Do not cast pointers to int, long, ULONG or DWORD; use UINT_PTR or INT_PTR
- Use the PtrToLong or PtrToUlong function to truncate pointers

Porting Process

Win64 Driver Source Code Clean-up (contd.)

- For devices supporting more than 4 GB
  - Use Mm64BitPhysicalAddresses value to determine if 64-bit addressing is needed
  - Use Dma64BitAddresses member of the DEVICE_DESCRIPTION structure to indicate that 64-bit addressing is supported

Porting Process

32-Bit IOCTL Support on Win64

- X86 32-bit applications run on Win64 through Wow64
- Wow64 is to thunk Win32 APIs and converts them to proper 64-bit parameters before transition to the kernel

Porting Process

32-Bit IOCTL Support on Win64

- 32-bit IOCTLs are needed
  - If driver exposes an IOCTL to user-mode applications AND
  - The input buffer and/or output buffer used by this IOCTL contain data types that are pointer-dependent
  - Driver has to provide two sets of IOCTLs
    - one for 32-bit process and one for 64-bit process

Porting Process

32-Bit IOCTL Support on Win64

- IoIs32bitProcess(IN PIRP Irp)
  - A new API to detect if the IOCTL is from a 32-bit process
  - Use this API to detect and to properly thunk the IOCTL structures your driver exposed to user-mode applications

Porting Process

Regression Test

- Validating code in IA-32 environment
- After cleaning up driver code to make it Win64 ready
  - Build the driver using IA-32 tools (Windows 2000 DDK)
  - Validate the driver under 32-bit Windows environment by running the necessary tests

Porting Process

Installing a Device

- Needed from vendor
  - Driver and INF file
- Setup in 64-bit Windows detects the device and asks for the INF file for information such as driver images, registry information and version information

Porting Process

Setup to run 64-bit Windows

- Installing 64-bit Windows
  - Prototype Itanium processor based systems come with SCSI drive, CD-ROM drive
  - Insert the 64-bit Windows CD into the CD-ROM drive to do the installation
- Running 64-bit Windows
  - After power-on, the system will boot to Extensible Firmware Interface (EFI) shell

Porting Process

Setup to run 64-bit Windows (contd.)

- Type “fs0:” <Enter>
- Type “os\winnt50c\ia64ldr” to invoke the IA-64 EFI Windows loader

- To shutdown the system
  - Same process as in 32-bit Windows

Porting Process is simple!

Case Studies

- Data structure packing & padding
- Unaligned data access
- Page size
- Address pointers
- Pointer Operation
- Ambiguous pointer usage
- Assumption on Data Size

Case Studies

Data Structure Packing

- Structure packing
  - Compiler generates codes with naturally aligned boundaries.
- Keep pragma pack() in your 32-bit driver code

```
#pragma pack(1)
:
#pragma pack()
```

Case Studies

Data Structure Padding

- Structure using hard-coded size for padding

```
Struct Buffer {
        PVOID   Ptr[10];
        char    Padding[88];
        }
```

To pad data to 128-byte chunk (4*10 + 88) in Win32

Case Studies

Data Structure Padding (contd.)

- Pointer is 8-byte in Win64
  - Potential bug for size overflow
  - Performance impact due to padding error
- Better to use

  Char Padding[128 - (10 * sizeof(PVOID))]

Case Studies

Unaligned Data Access

- No unaligned data access in kernel
- Use "__unaligned" qualifier to access unaligned data, if necessary

```
__unaligned * pmyStruct;
```

- Performance impact
  - Code size increased and slow IO

Case Studies

Page Size

- 32-bit drivers using hard-coded OS page size
  - Align internal buffer size for performance optimization

```
if ( Sizeof(MyStruct) % 4K ) {
   … Pad MyStruct to 4K …
}
```

Case Studies

Page Size (contd.)

- Perform device specific operations

  if (PageSize == 4K) { … }

- Use system defined PAGE_SIZE
  - Avoid using hard-coded page size
- 64-bit Windows is currently using 8K page-size

Case Studies

Address Pointers

- Virtual address pointers allocated by 64-bit Windows
  - Always 8-byte length
- Physical address pointers used by the device
  - Windows 2000 always uses 8-byte as internal data structure for physical address

Case Studies

Address Pointers (contd.)

- For existing devices with only 32-bit IO ability
  - Can still be 4-byte size
  - NdisGetPhysicalAddressLow()
- Devices with 64-bit addressing or DAC capability
  - 8-byte pointer size

Case Studies

Ambiguous Pointer Usage

- Operation on pointers

  NumberBytes = (ULONG) (Ptr1 - Ptr2);

  – Windows 2000 is to allocate buffers from any virtual address space

  – Fail in 64-bit Windows if NumberBytes is greater than 4 GB

  – Should be written as

    NumberBytes = (ULONG_PTR) (Ptr1 - Ptr2);

Case Studies

Ambiguous Pointer Usage (contd.)

- Passing array of pointers

  Caller - move data to ArrayPtr

  Callee(ArrayPtr)

  Callee - decode pointers from ArrayPtr

  – Data passed by caller may only be ULONG in size

Case Studies

Data Size Assumption

```
Struct  {
        ULONG  Space;
        PVOID   Buffer;
        ULONG  Offset;
        ULONG  Length;
} IoBlock;
```

```
Struct  {
        PVOID  Argument1;
        PVOID  Argument2;
        PVOID  Argument3;
        PVOID  Argument4;
} OtherStruct;
```

- **In Win32, it is ok to use UNION of structures "IoBlock" and "OtherStruct"**

Case Studies

Data Size Assumption (contd.)

- and use data structures interchangeably

- In Win64, the data structures will be corrupted

Make sure you follow proper guidelines!

Debugging Win64 Device Drivers

- Win64 debug tools
- Debugging environment
- Win64 KD components
- References on KD
- Setting-up IA64KD
- Starting IA64KD
- IA-64 Software conventions

Debugging Drivers

Win64 Debug Tools

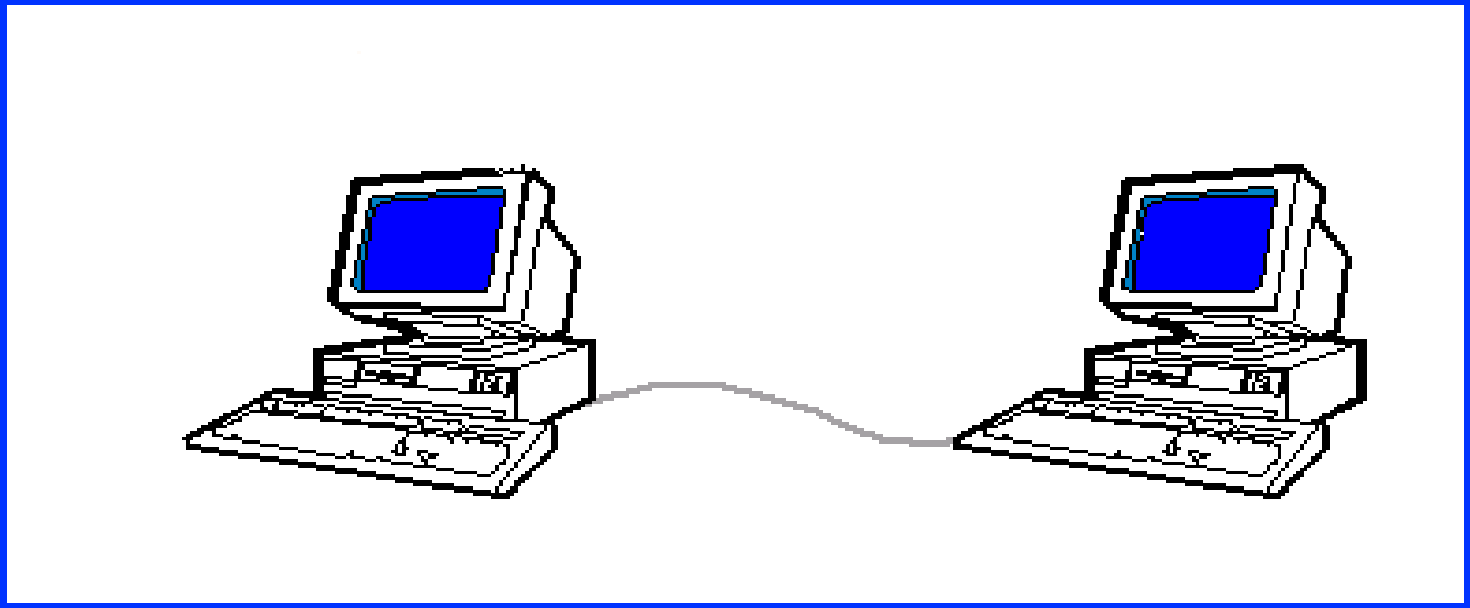
- Kernel mode debugger
  - ia64kd.exe
- User mode debugger
  - ntsd.exe

Debugging Drivers

Debugging Environment

HOST
Running Windows 2000
free build and kernel debugger

TARGET
Running 64-bit Windows
checked build



Connected by null modem cable

Debugging Drivers

Win64 KD Components

- Kernel Debugger Client or KD Stub
  - Build-in to Windows kernel. Can be enabled or disabled via boot.nvr
- Kernel Debugger Server or ia64kd.exe
  - A Win32 application that executes on the debugger machine

References on KD

- http://www.microsoft.com/hwtest/sysdocs
  - NT Debugging Overview
- Documents that come with DDK

Debugging Drivers

Setting-up IA64KD

- Enable kernel debugging in boot.nvr on target system

SYSTEMPARTITION=multi(0)disk(0)rdisk(0)partition(1);multi(0)disk(0)rdisk(0)partition(1)
OSLOADER=multi(0)disk(0)rdisk(0)partition(1)\os\winnt50C\ia64ldr.efi;multi(0)disk(0)cdrom(1)\setupldr.efi

OSLOADPARTITION=multi(0)disk(0)rdisk(0)partition(2);multi(0)disk(0)cdrom(1)

OSLOADFILENAME=\WINNT64;\IA64

LOADIDENTIFIER=Microsoft Windows 2000 Server;Microsoft Windows 2000 Setup

OSLOADOPTIONS=/debug /baudrate=56000 /debugport=com1

Debugging Drivers

Setting-up IA64KD (contd.)

- Enable kernel debugging in boot.nvr on target system

```
COUNTDOWN=30
AUTOLOAD=YES
LASTKNOWNGOOD=False
```

Debugging Drivers

Setting-up IA64KD (contd.)

- Create a batch file kd.bat on debugger system

```
mode 80,160
SET _NT_SYMBOL_PATH=y:\ia64kd\symbols
SET _NT_DEBUG_LOG_FILE_OPEN=y:\ia64kd\kd.log
SET _NT_DEBUG_PORT=COM1
SET _NT_DEBUG_BAUD_RATE=56000

ia64kd -v -b  %1
```

Debugging Drivers

Starting IA64KD

- Startup ia64kd.exe before starting the 64-bit Windows target system

ia64kd -v -b

option	
-?	Display help
-b	Cause initial break in kernel
-v	Verbose output

Debugging Drivers

IA-64 Software Convention

General Registers	Usage
r1	global data pointer(gp)
r8	return value(ra)
r12	memory stack pointer(sp)
r32-r127	register stack

Branch Register	Usage
b0	return address pointer(bp)

IA-64 Software Convention (contd.)

Procedure calls

Parameter passing	up to eight arguments on register stack

Debugging in 64-bit Windows is similar to debugging in 32-bit Windows!

Call to Action

- Read related documents
- Start porting your Windows 2000 device drivers to run on 64-bit Windows

Useful URLs and References

- http://developer.intel.com/design/ia-64/
- Documents in Win64 SDK and Win64 DDK
- Documents in Platform SDK of the latest release MSDN Library
  - Getting Ready for 64-bit Windows
  - Designing 64-bit Compatible Interfaces


Intel
Developer
Forum
Spring 2000
intel