Middleware for NICs in Bare PC Applications

Faris Almansour, Ramesh Karne, Alexander Wijesinha, Hamdan Alabsi, and Rasha Almajed
Department of Computer and Information Sciences
Towson University
Towson, MD

***Abstract*—A bare PC application runs without the support of an operating system (OS) or kernel and includes the necessary device drivers with the application. We describe the implementation of novel OS-independent middleware that enable bare PC applications integrated respectively with bare Ethernet drivers for 3COM and Intel network interface cards to use either driver/card combination. This work provides insight into developing middleware usable with bare PC drivers for a variety of Ethernet cards.**

Keywords—bare machine computing; bare PC; middleware; NIC; OS-independent NIC driver

I. Introduction

Bare PC applications run on ordinary Intel Architecture 32-bit (IA-32) compatible PCs without the support of any operating system (OS) or kernel. Advantages of bare PC applications include the elimination of OS/kernel overhead and a limited attack surface: no administrative privileges, no DLLs, and no support for scripts. A bare PC application integrates a network interface controller/card (NIC) device driver as part of its application code. Middleware is needed to enable two bare PC applications integrated with their respective NIC drivers to work with both drivers. We design and implement such middleware for a bare PC Web server [1] integrated with a driver for an Intel 82540EM NIC [2]; and a bare PC Webmail server [3] integrated with a driver for a 3COM 905CX NIC [4]. We also identify features of NICs that will be useful for designing a generic NIC architecture for bare PC applications.

II. Related Work

In [5], static analysis tools are used to analyze the code of Linux device drivers, and the interaction between drivers, the OS, and the hardware is examined. In [6], hardware abstraction and APIs for hardware and software interfaces are used as a basis for developing device drivers, and a device object model to separate OS-dependent and device-dependent driver components is presented. The Uniform Driver Interface (UDI) is intended for developing device drivers that are portable with respect to platforms and OSs [7]. The Network Driver Interface Specification (NDIS) library is supported by many Windows versions, while the NDIS wrapper [8] enables Windows drivers to be used with Linux.

III. Design and Implementation

An API is provided for bare PC applications to directly access the Ethernet interface. The code segment in Fig. 1 illustrates how calls are made by the application to methods that format the TCP, IP, and Ethernet headers in an outgoing packet. These calls are made within a single thread of execution avoiding the need for buffer copying and switching between layers. The Ethernet buffers are accessed and the send buffer address is formed in step 1. Formatting headers and aligning the data is done in steps 2 through 7.

```
//1. Get Transmit Buffer Pointer
x = EO.getTDLPointer() + EO.getSendInPtr() * 16;
p1 = (long*)x;
send_buffer = (char*)*p1;

//2. Add Ethernet and IP Header Lengths
send_buffer = send_buffer + 14 + 20;

//3. Format TCP Packet
TCPPack_size = FormatTCPPacket(send_buffer, tcb->IP,
      tcb->PORT, tcb->tempflags,tcb->RCVWND,
      tcb->tempSeqNum,tcb->RCVNXT, sendbuffer, TCPSegSize,
     tcb->temprIndex, currenttask);

//4. Adjust for IP Header
send_buffer = send_buffer - 20;
//5. Format IP Packet
retcode = ip.FormatIPPacket(send_buffer, TCPPack_size, tcb->IP,
      tcb->destmac, TCP_Protocol, currenttask);

//6. Adjust for Ethernet Header
send_buffer = send_buffer - 14;
//7. Format Ethernet Header and Send Data
retcode = EO.FormatEthPacketN(send_buffer, TCPPack_size+20,
    IP_TYPE , tcb->destmac,InPtr, tcb->count1, tcb->sendtype,
    currenttask);
```

Fig. 1. Bare PC application calling the Ethernet driver.

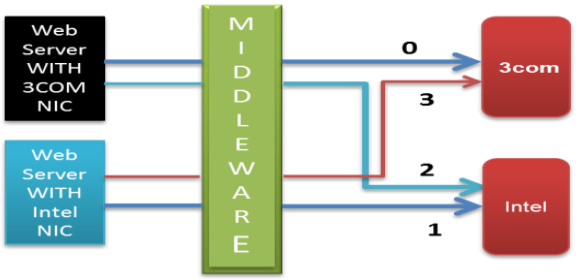

Fig. 2. NIC access via middleware.

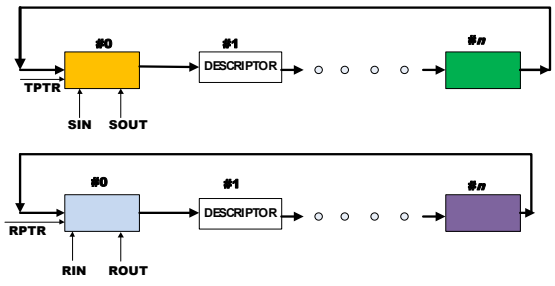

Fig. 3. Circular list send and receive data structures.

Fig. 2 shows four configurations/paths through the middleware. The **"0"** and **"1"** paths are the same as each server accessing its original driver except that access is now through the middleware. The **"2"** and **"3"** paths require middleware functions that allow the respective server to access the second driver. To dynamically choose the driver and NIC to be used by a server, a variable called **"NIC_FLAG"** taking values 0-3 is used. Fig. 3 shows two circular list data structures, one for sending and one for receiving. Each list consists of **"descriptor"** elements that store information needed to communicate with the NIC. Pointers SIN/SOUT and RIN/ROUT are used to insert and remove items from the send and receive lists that have head pointers TPTR and RPTR. The circular list size depends on the queue size required for sending and receiving. Sizes of 4096 (3COM) and 20,000 (Intel) are sufficient to buffer packets based on the data rates for the NICs (100 Mbps for 3COM and 1 Gbps for Intel). The NIC descriptor sizes are 32 and 16 bytes for 3COM and Intel respectively. The middleware works at the driver interface level and only needs details of some fields e.g., memory address, and previous/next link pointers.

The Init() functions in the two applications are specific to each application and perform different tasks. Applications use NIC interfaces (functions) and global variables (static in a class) to compute the values of other variables or data. For example, an application uses SIN and SOUT pointers to compute the descriptor address to access the data structure and modify its controls. These pointers are not directly addressable by the application as it uses the middleware, which accesses the actual NIC driver. We address this issue by writing getSIN(), setSIN(), getSOUT(), and setOUT() functions to replace the SIN and SOUT variables. In general, all NIC variable instances must be replaced with equivalent functions to access the variables. Additionally, global variables are used in some expressions. For example, TDLPointer may occur in an expression E that is used to compute a value X. Then a function getTDLPointer() is defined in the middleware that returns the value of TDLPointer, and this function is used instead of TDLPointer in the expression E to form a new expression newE. Also, a new function getXNew() is defined in the middleware to compute X using the expression newE instead of E.

The applications instantiate the NIC interfaces using the middleware object EtherObj (EO), which accesses the 3COM or Intel driver objects. The existing Ethernet objects were renamed as EthernetObjIntel and EtherObj3Com to distinguish the drivers. The new middleware object EO is included in the system as part of the application. This requires some changes to the bare PC compile and link commands. The middleware class consists of a header and a class file. The header file consists of all function prototypes needed for the middleware, and some constants needed for initializing data structures. There is a separate memory map for the NIC in existing server applications. In the middleware, a new memory area is used for data structures. This allows an application to use the respective Intel and 3COM memory maps for the data structures depending on which NIC is accessed. The memory map defines the DPD, UPD, TDL, and RDL addresses, and their sizes. These entities are different for the 3COM and Intel server applications and needed to be redefined. The implementation file for this class consists of functions that call the 3COM and Intel driver methods. The middleware code is written in C++. The NIC middleware was tested by running the servers on a Dell Optiplex 960 desktop.

IV. Generic NIC Architecture

The key elements used to communicate with a NIC are its data structure and controls. The device driver for a NIC manages these descriptors and ensures that they are used correctly. The data buffer and length fields in the transmit descriptor are required in both NICs. The transmit status field occupies 4 bytes in 3COM and 4 bits in Intel. Frame start, send flag, and previous pointers are not available in Intel; and the fields CSO, CSS and CMD are not applicable to 3COM. Similarly, in the receive descriptor, many fields are not used, or reserved. Also, the status and error fields are not applicable to 3COM. A generic NIC architecture could use a uniform 32-byte descriptor for transmit and receive. It would include common functionality for the 4-byte fields Descriptor Command, Status, Data Buffer Pointer, Data Buffer Length, Enable/Disable, Packet ID, Next DPD Ptr, and Next UPD Ptr.

V. Conclusion

We designed OS-independent NIC middleware that enables a bare PC application integrated with one driver to use a driver integrated with another application. We discussed a generic NIC architecture that will serve as a first step towards unifying NICs at the data structure and descriptor levels. The middleware needs to be extended in the future so that it can be used with many bare PC applications and NICs.

References

[1] L. He, R. K. Karne, A. L. Wijesinha, and A. Emdadi, A study of bare PC Web server performance for workloads with dynamic and static content, 11th IEEE International Conference on High Performance Computing and Communications (HPCC), 2009, pp. 494-499.

[2] Intel; PCI/PCI-X Family of Gigabit Ethernet Controllers Software Developer's Manual.

[3] P. Appiah-Kubi, R. K. Karne, and A. L. Wijesinha. The design and performance of a bare PC Webmail server, 12th IEEE International Conference on High Performance Computing and Communications, (HPCC), 2010, pp. 521-526.

[4] 3COM; 3C90xC NICs Technical Reference. (1999).

[5] A. Kadav and M. M. Swift, Understanding modern device drivers, 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2012.

[6] A. Amar, S. Joshi and D. Wallwork, Generic Driver Model, http://www.design reuse.com/articles/a8584/generic-driver-model.html, [Accessed: 28-Apr-2017].

[7] Uniform Driver Interface, http://www.projectudi.org/, [Accessed: 28-Apr-2017].

[8] NDISWrapper, http://ndiswrapper.sourceforge.net/wiki/index.php/Main_Page [Accessed: 28-Apr-2017].