I_2O Architecture Overview

Mark Brown
Intel Corporation
Technical Marketing Engineer

Abstract

This paper addresses the I_2O architecture and the motivation behind forming the I_2O Special Interest Group (SIG). The I_2O specification defines an architecture for managing devices that is independent of the implementation of the device and the operating system. The resulting architectural framework, or structure, will promote interoperability, performance, and ease-of-use.

The model behind the I_2O architecture is to separate the functionality of the driver, which manages devices, from the specific nature of the operating system that it serves. It acts as an abstraction layer to allow the part of the device driver that actually manages devices to become portable among different operating system environments and to be portable and usable across different vendor implementations.

This paper provides an introduction to the dilemmas that exist in the I/O of server platforms, an overview of the I_2O architecture split driver model and message interface, system models available with the I_2O architecture and an explanation of the structure and membership of the I_2O Special Interest Group.

Introduction

The demand for device driver portability between operating systems and host platforms, combined with increasing requirements for intelligent, distributed I/O processing has led to the development of the Intelligent Input/Output, or I_2O (pronounced eye-two-oh) specification. The basic objective of the I_2O architecture is to provide an I/O device driver architecture that is independent of both the specific device being controlled and the host operating system. This is achieved by logically separating the portion of the driver that is responsible for managing the device from the specific implementation details for the operating system that it serves. By doing so, the part of the driver that manages the device becomes portable across multiple operating systems. The I_2O architecture also acts to hide the nature of the communication between various mechanisms, and in doing so provides processor and bus technology independence.

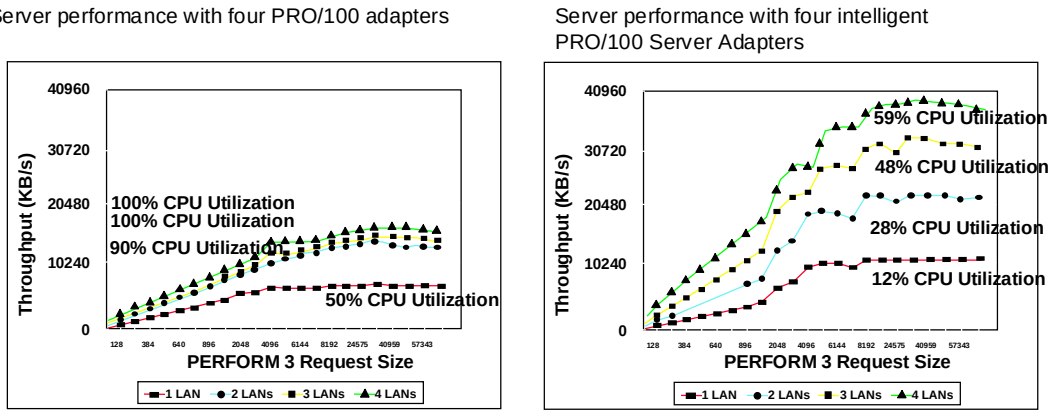
The I_2O architecture is also designed to facilitate intelligent I/O subsystems, with support for message-passing between multiple independent processors. By relieving the host of interrupt intensive I/O tasks required by the various layers of a driver architecture, the I_2O architecture greatly improves I/O performance. I_2O architecture compatible systems will be able to more efficiently deliver the I/O throughput required by a wide range of high bandwidth applications, such as networked video, groupware, and client/server processing. At the same time, the I_2O architecture imposes no restrictions as to where the layered modules execute, providing support for single processor, multiprocessor, and clustered systems.

While the I_2O architecture provides a compelling alternative to traditional monolithic driver models, its intent is not to create an entirely new interface to replace the driver architectures currently in existence. Rather, the objective is to provide an open, uniform approach, which is complementary to existing drivers, and provides a framework for the rapid development of a new generation of portable, intelligent I/O solutions.

The I/O Bottleneck and Driver Proliferation Dilemma

System balance is critical to optimizing overall system performance. The host processor, local memory, and I/O devices must all work in concert to provide the end-user with a productive working environment. If one aspect of the system, such as I/O, becomes a bottleneck, overall performance will suffer (Figure 1). This is particularly true in the client/server computing

environment, where end-users rely heavily on network resources, such as data, applications and peripherals, and interact with other users on the network.



Source: Intel Benchmarks

Figure 1: Intelligent I/O Performance Analysis for Fast Ethernet

While there is a growing range of new technologies designed to offer desktop computer users improved I/O performance over the network, including ATM, Fast Ethernet, FDDI, and Fibre Channel, these high-speed networks actually make system I/O balance even more critical. At the same time, there has been a proliferation of network operating systems for users to choose from, most notably NetWare 3, NetWare 4, Windows NT Server, and UnixWare. The result is what is known as the RM x NS problem, where each new combination of I/O technology and operating system requires a different device driver to be written, tested, and integrated. As the number of new drivers spirals, development and system management overhead grows.

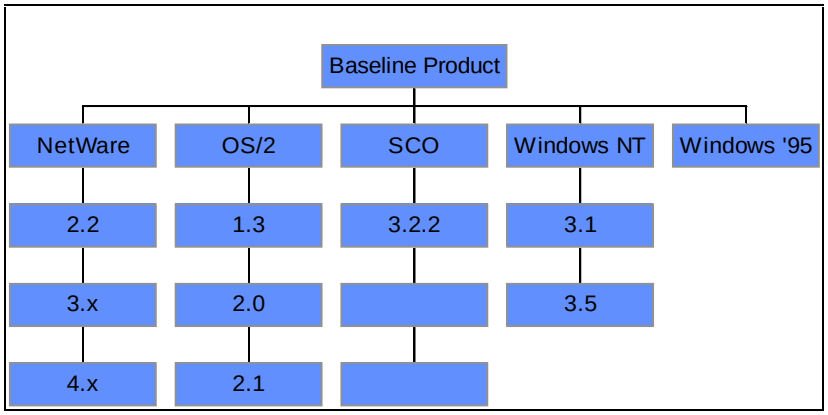


Figure 2: Device Driver Proliferation Dilemma

Today, the task of writing, integrating, and testing device drivers for each new release of an operating system is a significant burden for the entire computing industry (Figure 2). It has even been suggested that the problems associated with deploying and supporting new network technologies at the driver level have played a role in slowing the adoption of these exciting new high-speed networking options. The solution to this problem is the standardization of the I/O architecture.

I_2O Architecture Conceptual Overview

The I_2O architecture provides an ideal environment for creating drivers that are portable across multiple operating systems and host platforms. As shown in Figure 3, drivers are divided into two parts; the OS Services Module (OSM) which interfaces to the host operating system's interface; and the Hardware Device Module (HDM) that interfaces with the particular device, media or server that the driver must manage. These modules interface with each other through a communication system comprised of two layers: a Message Layer which sets up a communications session between two parties, and a Transport Layer which defines how the two parties will share information. Much like a standard communications protocol, the Message Layer resides on the Transport Layer.

The I_2O communication model, when combined with an execution environment and configuration interface, provides the HDM with a host independent interface. The modules are able to communicate without knowledge of the underlying bus architecture or system topology. Messages form a meta-language for the modules to communicate that is independent of the bus topology and host Operating System interfaces. Just as a networking protocol stack, such as TCP/IP, is able to isolate communicating parties for each other's specific implementation details, the I_2O architecture is able to provide device driver portability.

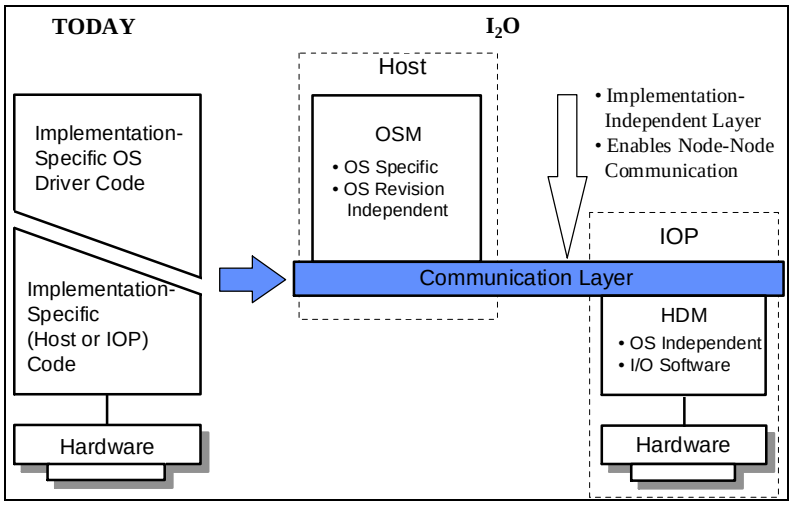


Figure 3: I_2O Split Driver Model

How It Works

The communications model for the I_2O architecture is a message passing system. The communication model is analogous to a connection oriented networking protocol or the OSI layered model, in which two parties interested in exchanging messages utilize the Message Layer to set up a connection and exchange data and control.

When the OSM is presented with a request from the host operating system, it translates the request into an I_2O message and dispatches it to the appropriate HDM for processing. Upon completion of the request, the HDM dispatches the result back to the OSM by sending a message through the I_2O Message Layer. The OSM behaves just like any other device driver from the host operating system's perspective.

The Message Layer

The foundation of the I_2O architecture is the Message Layer, which provides the glue that connects the framework of the driver model. The Message Layer is responsible for the management and dispatching of all requests. It provides a set of APIs for delivering messages, along with a set of support routines that process them.

There are three basic components in the Message Layer: the message handle, the Message Service Routine (MSR), and the message queue. The message handle is essentially the 'address' of the MSR registered in the call. A message handle is returned for every call to the Message Layer. The message queue provides the link between the requester and the desired service.

When a driver request is made, a message is deposited in a message queue and an MSR is activated to process the request. Messages themselves are made up of two parts: a header and a payload, where the header describes the types of request along with the return address of the originator.

The I_2O architecture is based on a queue between the requester and the MSR as shown in Figure 4. The requester and service module can reside either on separate execution environments, or on a single processor system. The I_2O architecture also defines a neutral memory format, which provides independence for the host operating system memory model.

By providing an open, standard and neutral format mechanism for communication between the service modules, the Message Layer is the foundation.

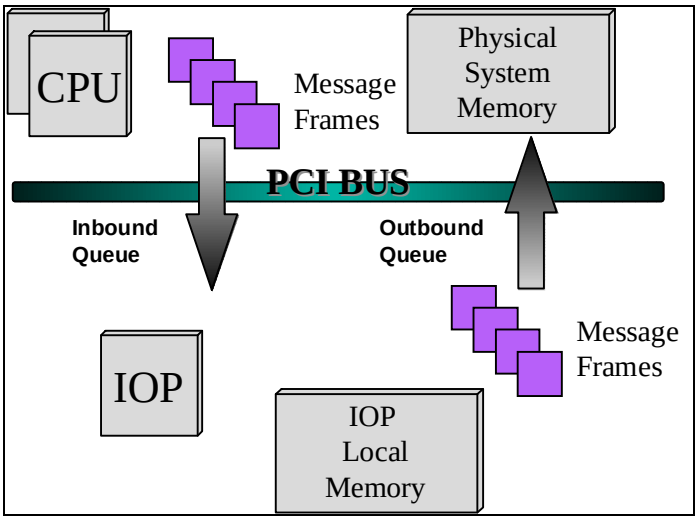


Figure 4: I_2O Message Interface

The Operating System Services Module - OSM

The OSM provides the interface between the host operating system and the I_2O Message Layer. In the split driver module, the OSM represents the portion of the driver that interfaces to host specific Application Programming Interfaces (APIs), translating them to a neutral message based format that is then sent to an HDM for processing.

The OSM translates requests from the host operating system into messages which can be dispatched to the appropriate HDM for processing. HDM information is forwarded back to host operating system through the OSM via the I_2O Message Layer.

Developers can also create host OSM's that works with multiple HDM's. By implementing an OSM with a single message handle which services multiple queues from different service modules, a single OSM can send and service requests to/from multiple, different devices.

The Hardware Device Module - HDM

The HDM is the lowest level module in the I_2O environment, and provides the device specific portion of the driver that understands how to interface with the particular controller and devices. HDM's are roughly analogous to the hardware-specific portion of the network and SCSI drivers that exist today. The HDM translation layer is unique to each individual hardware device and vendor, and supports a range of operation types, including synchronous, asynchronous request, event driven, and polled.

The HDM itself is surrounded by the I_2O environment, which provides the necessary support for the operating system processes, and bus independent execution. HDM's are typically written in C or C++ and can be architected in a manner which minimizes changes when moving from one hardware platform to another.

System Environments

The I_2O architecture is designed for single processor, multiprocessor, and clustered processor systems, as well as desktop, communications, and real-time system environments.

Both the HDM and OSM interface to a basic I_2O API set. The execution environment for OSM's consists of the execution environment provided by the hosting operating system along with the basic I2O API set. The host-based I_2O environment complements the operating system services by providing a bridge between the operating system device APIs and the HDM.

In order to accommodate access to real-time operating system environments, HDM's have an additional set of I_2O Embedded Kernel Services API's (Figure 5). This interface provides HDM's with access to required operating system functions, without exposing the actual Embedded Operating System's interfaces to the HDM. This layer provides the set of services needed to establish a cocoon that HDM's load into, thereby making them independent of their surrounding execution environment. This is especially important in meeting the I_2O architecture objective of allowing HDM's to run in multiple target execution environments.

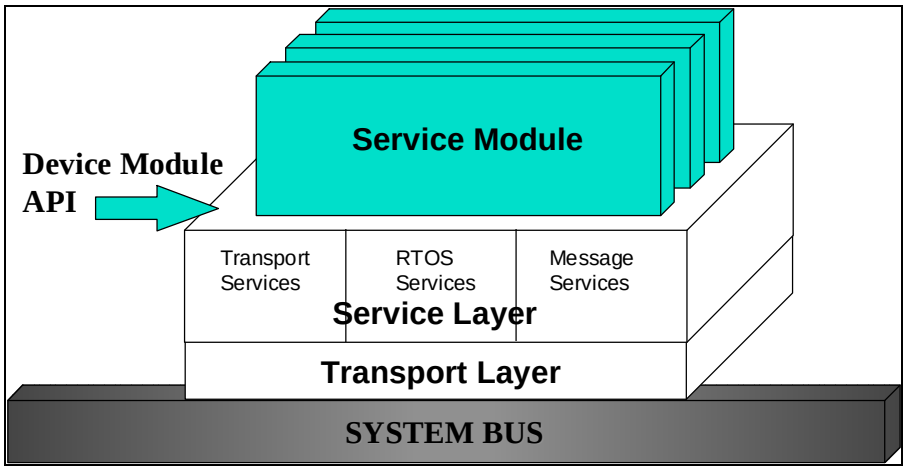


Figure 5: I_2O Message Service Model

Because the I_2O architecture is designed to readily support single processor systems, the service layer is scaleable. It can easily be simplified in scope for more restricted processing environments.

System Models

The I_2O architecture provides for a variety of possible system models. Each I_2O architecture segment is composed of a primary host node and one or more I/O nodes. For each host node, there is an I_2O resource manager responsible for initialization and configuration of the I_2O components within the segment.

Figure 6 shows the host CPU node and two I/O nodes. Although I/O devices A & B and G & H are directly addressable from the system bus, I/O devices A & B are directly controlled by the host OS, while I/O devices G & H are controlled by I/O platform 2.
A device appearing behind an IOP is an abstracted device. I/O devices C & D are abstracted from the system as they are private devices in I/O platform 1 and are only addressable on the IOP's local bus.

Further device abstraction is accomplished by I/O devices D & G which contain devices accessible through their own bus. An example of this abstraction would be a SCSI adapter with SCSI devices.

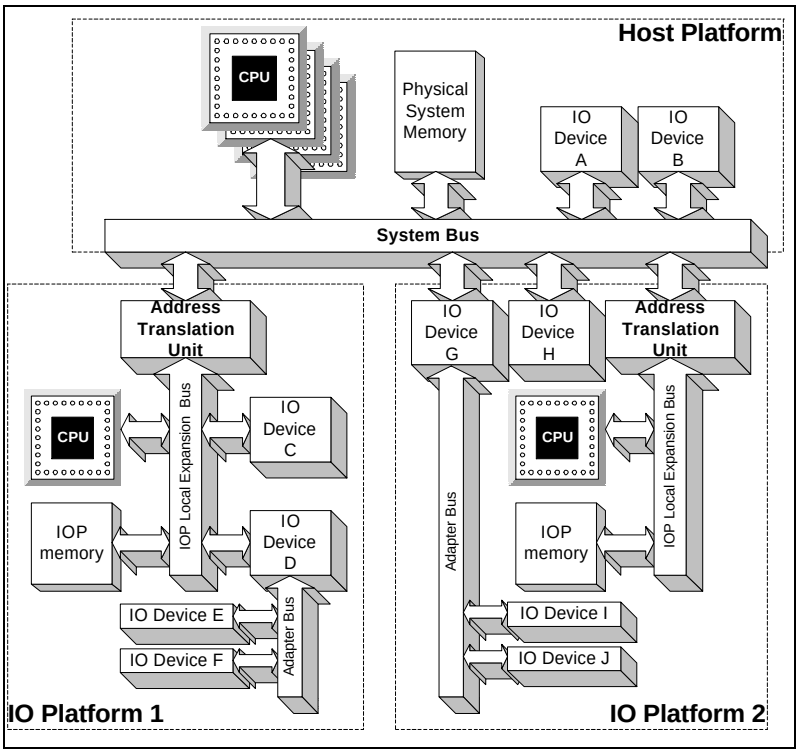


Figure 6: I_2O System Model

I_2O Special Interest Group

The I_2O Special Interest Group (SIG) is committed to establishing the I_2O architecture as the industry preferred specification for high performance I/O systems, and to attracting the widest possible range of vendors to become members of the consortium. A number of leading suppliers have worked closely to develop the specification, including members of the I_2O steering committee - 3Com Corporation, Compaq Computer, Hewlett-Packard Company, Intel Corporation, Microsoft Corporation, NetFRAME Systems Incorporated, Novell, Inc., and Symbios Logic.

Other I_2O SIG members include Acer, Adaptec, AMI, Cyclone Microsystems, Inc., Distributed Processing Technologies, Fore Systems, Harris & Jeffries, ICP/Vortex, ISI, Mylex Corporation, PLX, SCO, Siemens, SMC, Tandem Computers, Inc., Topmax, V3, Veritas Software, Western Digital Corporation, WindRiver Systems, and Xpoint Technologies, Inc.

The I_2O SIG is open to additional membership. Companies can join either as Contributing or Associate members. Companies interested in participating in the I_2O SIG can obtain copies of the I_2O technical specification and membership details by contacting LoBue & Associates at 415.750.8352. General information is available through the I_2O SIG web page at http://www.i2osig.org.

Conclusion

The I_2O architecture model is intended to provide a unifying approach to device driver design by creating a logical split and off-loading of functions between host specific and device specific interfaces. The intent behind the I_2O architecture is create a framework for rapid product development and the implementation of portable, high-performance, intelligent I/O systems. The

result is an open specification, defined by proven leaders from all segments of the server and I/O industry, which provides a clear migration path from today’s legacy driver model.

Biography

Mark Brown
Technical Marketing Engineer
Intel Corporation - Enterprise Computing I/O
(602) 554-3864

Mark Brown works as a Technical Marketing Engineer in Intel's Computing Enhancement Division. His current assignment is providing technical support for the I_2O Architecture and i960® RP microprocessor and currently serves as the chair for the I_2O SIG Server Management Working Committee.

Prior to his technical marketing assignment, he was responsible for system validation of the i960® JX and HX microprocessors. He has 5 years of experience in writing software for embedded applications including two years writing software for embedded avionics applications. Mark has a B.S. in electrical engineering from Kansas State University.