What is UDI?

OS-Neutral
Platform-Neutral
Device Driver Interface
UDI Driver Portability

• 100% Driver Source Portability
  – Defines architecture, APIs and packaging format

• Binary Portability (where applicable)
  – IA-32 and IA-64 ABIs defined
    (Recent work on AMD64, ARM in 2010)

• Source and Binary Distributions
The “Driver Problem”

- One Device, Many Operating Systems Requires Many Drivers for One Device

  - Who writes all these drivers?
    - Device Manufacturer (IHV)? OS Vendor (OSV)? 3rd-Party Contractor or Systems Integrator?
    - Business decisions or personal priorities mean that some combinations get left out.

  - Less popular OSes get fewer drivers
    - (Everybody but Microsoft 😊 )
The UDI Solution

• One driver source for all UDI-compliant OSes
• UDI moves up IHV porting order
  – More bang for the buck for IHVs
• UDI-compliant OSes get better coverage (once critical mass reached)
UDI In Action

First Prototype Completed 12/9/1997

Single Driver Source
No #ifdefs
No modifications

Adapters
Adaptec SCSI
Interphase 100BT
Gigabit NIC

HP-UX
32 bit PA_RISC

SUN Solaris
32 bit UltraSparc

Tru64 UNIX
64 bit Alpha

NCR MP-RAS
32 bit Intel

IBM AIX
PowerPC

SCO Unixware
32 bit Intel

Copyright 2010 Kurt Gollhardt

Uniform Driver Interface
Project UDI - Slide 6
The Versioning Problem

• OS Driver APIs Change Over Time
• Driver and OS Development Cycles Unnecessarily Linked
  – IHV should change driver as H/W evolves
  – OSV should be able to evolve OS w/o waiting for new drivers
UDI Versioning

• Stable API
  – Strict separation of responsibilities
  – Eliminate cross-cutting concerns

• Link-Level Versioning
  – Enables support for multiple versions simultaneously, even at binary level
Uniformity Across Device Types Reduces Learning Curve

- Common Execution Model
- Common Data Model
- Common Inter-Module Communication Mechanism
- Common System Services
UDI: Next-Generation Technology

- **Instance Independence**
  - Hot plug/hot swap adapters and devices

- **Location Independence**
  - Distributed environments and I/O processors
U DI: Next-Generation Technology
(continued)

• Implicit Synchronization
  – Thread-safe drivers W/O locking calls
  – High parallelism between driver instances

• Support for Field-Installable 3rd-Party Extensions
  – Add new device classes w/o OS updates
UDI as Technology Enabler

• **UDI simplifies support for:**
  – Future platforms (new CPU & I/O bus architectures)
  – Mixed-endian platforms & arbitrary bus hierarchies
  – User-mode drivers
  – Advanced driver debugging tools
  – Fault recovery and validation environments

• **None of these require driver changes!**
Free and Open Specification

- Published on the Web (1999 & 2001)
- No Licensing Fees
- Developed Jointly by a Multi-Company Team of OS Architects and Driver Writers
Primary Participants (thru 2002)
UDI Architecture

Driver Encapsulation
UDI Fully Encapsulates Drivers

UDI Environment

Application Programs → I/O Requests

Operating System

UDI Drivers

System Services:
- Configuration
- Resource Allocation
- Inter-Module Communication
- Tracing & Logging
- Error Handling
- Time Management
- Buffer Management

Physical I/O Abstraction

CPU and I/O Hardware (PIO, DMA, Interrupts)
UDI System Services

• **System interface & resource management**
  – Implemented for all UDI environments
  – Abstract OS services

• **Calls from driver to environment services are called service calls**
Path From Application to Driver
Integrated Implementation

Application

Embedding OS

I/O Subsystem

UDI Environment

UDI Services

UDI Driver

Physical I/O

Hardware Access

Adapter or System Hardware

OS Requests

UDI Channel Operations

Interrupts
Path From Application to Driver
Layered Implementation

Application

Embedding OS
I/O Subsystem
Native Driver Interface
UDI Environment
OS-to-UDI External Mapper
UDI Channel Operations
UDI Services
UDI Driver
Physical I/O
Hardware Access
Interrupts
Adapter or System Hardware
Example Driver Hierarchy

UDI Environment

Embedding OS

Monitor External Mapper

Keyboard External Mapper

Disk External Mapper

Tape External Mapper

Monitor Driver Instance

Keyboard Driver Instance

Disk Driver Instance

Tape Driver Instance

Base I/O Adapter Driver Instance

SCSI Adapter Driver Instance

I/O Bus Adapter Driver Instance

I/O SOFTWARE

PROCESSOR-MEMORY INTERCONNECT

I/O HARDWARE

I/O Bus Adapter

Base I/O Adapter

SCSI Adapter

Monitor

Keyboard

Disk

Disk

Tape

Child

Parent

UDI Environment

Copyright 2010 Kurt Gollhardt

Uniform Driver Interface

Project UDI - Slide 20
UDI Architecture
Execution Model
UDI Regions

- **Basic unit for execution and scheduling**
  - Each call into the driver region is serialized

- **No direct data sharing between regions**
  - *You have to go through channels*

- **Region attributes (e.g. priority hints)** specified at build time
UDI Regions (continued)

- One driver instance per device instance
- One or more regions per driver instance
  - Multi-region drivers may have higher parallelism
- Enables *instance-independence*
  - Driver state separate for each device instance
- Enables *location-independence*
  - Each region may operate in a different domain
    » e.g. address space, NUMA or network node
Regions and Channels

Region A (driver or environment)

Region Data

Region B (driver or environment)

Region Data

communication channel

channel endpoints
UDI Service Calls
Two Styles

- **Synchronous service calls**
  - Complete without blocking
  - Results returned “immediately”

- **Asynchronous service calls**
  - Return without blocking
  - Delayed completion
  - Results returned via callback function
Non-Blocking Execution Model

- All service calls and channel operations return without blocking
- Drivers usually return after making one service call or channel operation call
- Gives environment complete control over thread usage and driver scheduling
- Sequence of call chains between callbacks can be viewed as a “Pseudo-Thread”
UDI Architecture

Inter-Module Communication
UDI Channels
Basis for Inter-Module Communication (IMC)

• Bi-directional channels connect regions

• Communication via channel operations
  – Strongly typed function-call interface
  – Paired asynchronous one-way operations
    » Each request has a corresponding response
    » Context managed via control blocks
UDI Architecture
Data Model
UDI Data Model

• **Context managed via control blocks**
  – Used with channel ops & async service calls
  – Environment uses CB to hold service call state
  – Driver uses context pointer in CB to find its data

• **No memory shared between regions**
  – Memory allocated in a region is private to that region
  – Regions share data by using channel operations
UDI Control Blocks

• CB contains scratch and context pointers (preserved across service calls, not ops)
  » Scratch space in CB holds per-request state
  » Context pointer lets driver find the context of a channel op or callback
    • Initially set to channel context
    • Channel context struct points to global data

• All CBs can be cast to generic udi_cb_t
Implicit Synchronization

- No locking primitives required in UDI
  - All data accesses implicitly synchronized
    » Region data accessible only from that region
    » Only one thread per region active at a time
  - Other calls deferred until active call returns
    » Typically by adding CB to a region queue
  - Driver controls its parallelism by picking number and type of regions
More Information on UDI

Project UDI Website

http://project-udi.org

Reference Implementation

http://projectudi.sourceforge.net
UDI Specifications Now Available

- **UDI 1.01 Specifications at project-udi.org**
  - UDI Core Specification (2 volumes)
  - UDI Physical I/O Specification
  - UDI PCI Bus Binding Specification
  - UDI System Bus Binding Specification
  - UDI SCSI Driver Specification
  - UDI Network Driver Specification
  - UDI IA-32/IA-64 ABI Binding Specification
Informative Documents

• **Introductory Info**
  – UDI FAQ & Data Sheet
  – UDI Management & Technical Overviews
  – UDI Advantages & Opportunities

• **Other Materials**
  – Various Presentations & Tutorials
  – UDI Environment Implementer’s Guide
Reference Implementation

- Sample drivers & metalanguage libraries

- Sample OS implementations, including:
  - Linux*, UnixWare*, OpenServer*, Solaris* (partial)
  - Easily portable to other OSes

- User-mode test environments (no PIO) for:
  - Linux, UnixWare, Solaris, FreeBSD, Mac OS X

- Jointly developed by Project UDI members

- BSD-style Open Source License
UDI News Headlines (1999)

- Intel, Computer Makers to Forge Common Guidelines for Unix
  - Wall Street Journal

- Heavyweights Unite Behind Interface for Unix Servers
  - PC Week

- Intel Moves Closer to Unix in Standards Effort
  - Information Week

- Uniform Driver Interface Spells Relief
  - EE Times

- Intel Pushing Unified Unix
  - InfoWorld/C-Net
UDI Architecture

More Details & Examples
UDI Metalanguages

• Device-type specific communication for a particular device class

• Defines communication paradigm between cooperating modules
  – Operations and sequences to implement technology-specific functionality

• Analogous to SCSI CAM, DLPI, etc.
Metalanguages and Channels

• Metalanguages define:
  – Number and types of channels
  – Legal “channel operation” types for each channel type
    » Control block plus meta-specific parameters
  – Structure of control block for each operation type
    » Meta-specific “subclasses” of generic udi_cb_t
    » Implemented in C by using udi_cb_t as first member of each udi_xxx_cb_t structure

Copyright 2010 Kurt Gollhardt
Uniform Driver Interface
Project UDI - Slide 41
UDI Execution Model

• Driver’s udi_init_info structure contains entry-point pointers, size requirements, etc.

• No other global entry points
  – Avoids possible name collisions
  – No need to wrap driver with generated entry tables

• All driver code executed in context of a region
  – Regions are associated with driver instances
    » At least one region for each adapter/device controlled
Fundamental Data Types

• Specific-length types
  - udi_ubit8_t, udi_sbit8_t,
    udi_ubit16_t, udi_sbit16_t,
    udi_ubit32_t, udi_sbit32_t
  - udi_boolean_t (udi_ubit8_t)

• Abstract types
  - udi_size_t, udi_index_t
Fundamental Data Types (continued)

• **Opaque types**
  – Contain environment-private fields and structure
  – Must be allocated using UDI service calls
  – Opaque handles
    » `udi_channel_t, udi_constraints_t`

• **Semi-opaque types**
  – `udi_cb_t *, udi_buf_t *`
Core Services

- Inter-Module Communication (IMC)
- Memory Management
- Buffer Management
- Time Management
- Tracing and Logging
Core Utility Functions

- **String/Memory Utilities**
  - udi_strncpy, udi_strlen, udi_memcmp et al
  - udi_snprintf, udi_strtou32

- **Queue Management Utilities**

- **Endianness Management Utilities**
Core Metalanguages

- **Management Metalanguage**
  - Environment-initiated control operations

- **Generic I/O Metalanguage**
  - Generic read/write plus custom ops
  - Useful for prototyping and “one-off” extensions
  - Used to access driver diagnostics
Region Kill

• Different environments have different levels of trust in drivers

• UDI environments can:
  – detect misbehaved drivers (e.g. bad pointers)
  – track resource ownership and transfers
  – abruptly terminate (“region-kill”) driver instances
    » Frees all resources and shuts down device
Example UDI & I$_2$O Combination