The Volume Enterprise UNIX Platform
IBM - SCO - Intel

Intel IDF February 2000
Ahmed Chibib - Barry Feild
Agenda

- Monterey /64 Features and Functions
- Compilation Models
- Alpha and Beta
- Education Plan and Porting Centers
- Monterey /64 IHV Program
- Monterey /64 DDK
- Monterey /64 UDI
- Summary
Project Monterey Goals

- Establish the Monterey product line as the volume, enterprise class industry leader in UNIX OS segment
  - POWER and Intel architectures
  - Standards based offering
  - Largest UNIX application portfolio
  - Single offering for channel delivery
  - Distributed broadly to OEMs and resellers
  - Single offering from "workgroup-class servers" to "enterprise-class servers"
Monterey/64 Execution Roadmap

Monterey/64 OEMs
- Compaq
- Sequent
- Bull
- Unisys
- ACER
- ICL
- CETIA
- Netfinity
- Samsung

Sun OEMs
- NCR
- SNI
- Toshiba
- Fujitsu

HP OEMs
- NEC
- Stratus

Undecided Vendors
- Dell
- SGI
- Data General
- Hitachi

OEM Council Formed

Expand OEM Support

First Customer Ship

UDG-PI

Monterey/64

Build Mindshare

Development Tools

ISV Support

IHV Support

OEM Support
Monterey OS Delivers

- A Robust, Scalable UNIX Platform for Critical Applications
- The Connections You Need for e-business and Network Computing
- Security You Can Count On
- Systems and Network Management that Puts You In Control
- A User Experience that Speaks for Itself
- The Tools to Build Tailored Solutions
- Service and Support to Help Keep Your Business Running
Monterey-64 and Standards

- An IA-64 ABI and API for LP64
- Elf/Dwarf2 object model
- Based on UNIX98 APIs
- Standard header files
- Standards Compliant Tools
- Standard definition for derived data types
- Common Install format
- Universal Device Driver Interface (UDI)

Monterey/64 is a Standards based Operating System
Monterey/64 supplies a new pthread debug library (libpthdebug.a):

- Provides a set of function (API) to use for pthread debugging.
- Allow 3rd party debugger tools to access this information.

**New environment variables for libpthreads.a**

- Determine how pthread library maintains lists for the debugger.

<table>
<thead>
<tr>
<th>Environment Variables</th>
<th>Default</th>
<th>Information Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIXTHREAD_MUTEX_DEBUG</td>
<td>ON</td>
<td>Mutual exclusion lock (mutex)</td>
</tr>
<tr>
<td>AIXTHREAD_RWLOCK_DEBUG</td>
<td>ON</td>
<td>Read-write locks</td>
</tr>
<tr>
<td>AIXTHREAD_COND_DEBUG</td>
<td>ON</td>
<td>Conditional variables</td>
</tr>
</tbody>
</table>

- The default setting is ON. Set to OFF may improve the performance of applications.

*Easy to Debug Multithreaded applications*
Debug Malloc Implementation

Make application development easier and more efficient. Debug Malloc is a new facility that:

- Provides memory overlay detection capabilities
- Can be turned on by simply exporting the `MALLOCTYPE` and `MALLOCDEBUG` environment variables.
  - For example, `# MALLOCTYPE=debug`
    `# MALLOCDEBUG=align:n, .`
  - No other modifications on executable files necessary

Memory leaks debugging assistance
Malloc Multiheap

By default, the malloc subsystem uses only single heap

- Memory allocation requests are done serially.
- Impact on multiprocessor system performance.

Malloc Multiheap can be enabled.

- `MALLOCMULTIHEAP=[heaps:n] | [considersize]

- Example:
  - `MALLOCMULTIHEAP=true;
  - `MALLOCMULTIHEAP=heaps:3, considersize`

Better performance for malloc
Multiple Run Queues

Improve SMP Scalability of the dispatcher

• Implement multiple run queues with load balancing on SMP Systems
  - Single global run queue with a set local run queues (1:1 Queue/CPU)

• Better processor affinity and cache affinity

Better SMP performance
Online JFS Backup (Split Mirroring)

1. 3-Way Mirror
   - Production

2. One Mirror Split Off
   - Production
   - Backup

3. Production

4. Production

5. Reintegrate a Mirrored Copy
   - Production

Making an online backup of a mounted JFS file system.

- Creates a snapshot of the logical volume that contains the file system.
  - Logical volume and its JFS log logical volume must be mirrored.
  - File system activity should be minimal (quiescing) while the split is taking place.

Reliable and highly available file system
Mirroring and Striping

Support RAID 0+1 entirely in software

- Combines RAID 1 (mirror) data availability with RAID 0 (striped) performance
- No special hardware needed
- Example

```
# mklv -y 'raid10' -c '2' '-S4K' rootvg 5 hdisk0 hdisk1 hdisk2 hdisk3
```

Utilize a new partition allocation policy called super strict

- New \(-s\) option flag in `mklv` command
- Does not allow partitions from one mirror to share a disk with partitions from a second or third mirror

Striping for Speed, Mirroring for Reliability
Workload Manager

Virtual Partitioning

16-way SMP

System Administrator setup
Monterey/64 automatically gives resources according to entitlements

Same H/W, one system

Balancing the workload
### Partitioning vs. WLM...

**O/S Partitioning**
- O/S Fault isolation
- Resources can be wasted when requirements do not neatly match fixed partition boundaries

**Workload Management**
- CPU Time vs. CPUs
- Independent CPU time and memory management provide flexibility
- Single O/S Administration

---

<table>
<thead>
<tr>
<th>CPU</th>
<th>MEM</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="CPU Usage" /></td>
<td><img src="image2.png" alt="MEM Usage" /></td>
</tr>
</tbody>
</table>

---

**Waste**

Less Waste = More Applications
What is NUMA?

- “Non-Uniform Memory Access”
  - A method to respond to the SMP big-bus scaling road block
  - Two or more processor / memory nodes ("quads") coupled to form single MP (multi-processor) server
  - Runs a single NUMA-aware OS instance
  - NUMA fabric coupling supports low latency, cache coherent traffic

Scalability beyond SMP
IA-32 NUMA-Q Implementation

- System provides one physical address space and one I/O address space
- Scales massively and retains SMP programming model
- Design Tuned for Performance
  - Maximizes use of lowest latency memory...key advantage
  - L3 Cache and scheduling affinity mitigate remote memory access
- Customer investment protection continues as new processors come
Based on IA-32 implementation -- ready for IA-64!

NUMA APIs provide the following types of services:
- Services used to make queries on the system topology
- Services to manipulate assignment and allocation of system resources

APIs are advisory in nature -- use on non-NUMA hardware will cause no problems

Differentiates Monterey from other IA-64 operating systems
- Monterey can assist applications to take advantage of NUMA-based platforms for high scalability and data-center capacities

NUMA Support in Monterey/64
Monterey/64 Support

- Three compilation models:
  - IA-32
  - ILP32
  - LP64

No “Mixing” of Models Permitted

Choose the Appropriate Compilation Model
IA-32 Environment

- Targeted Binary Compatibility for existing UnixWare7 applications
- Allows Single Binary to be used on all IA Platforms
- As in UnixWare 7 today
ILP32 Environment

- 32-bit Source compiled for IA-64
  - Same Data Layout As IA-32
- Similar Performance to LP64
  - Data Conversion In/Out Of Kernel
  - Some Misaligned Data Objects
  - Smaller Data Size (better cache use)
  - Address Space and therefore Memory limited to 4GB
- Fully Supported
- Source Compatibility with Little-Endian
- Appropriate for Recompile-and-Go Software
Monterey/64 Environments

- LP64 (IA-64 64-bit) Environment
  - New and High-End Software
  - UNIX Industry-wide 64-bit Model
  - New Instruction Set, Longs, Pointers are 64 Bits
Alpha Program Update

Alpha “package” includes SDE and OS and will be available Feb. 29th

- OS and SDE must stay in sync
- Beta updates will refresh the “whole” package

UnixWare Based SDE

- Sample code will be provided for compile/debug example
- Cross-compiler and debugger will be in SDE package

Itanium™ Processor based systems planned for Alpha and Beta

- Systems will be shipped by Intel; OS and SDE by IBM
- Shared systems will be available at Solution Partnership Centers (SPCs)

Alpha is just around the corner
Beta Program Schedule

Beta planned for spring 2000

- Beta in May for key dependencies
- Expanded beta over June, July, and August
- Beta will have all key Monterey/64 OS capabilities
- Solution centers can provide Migration help
- ISV Migration Training will be available in April and beyond
SDE Contents

• Packaged for UnixWare 7
• IBM VisualAge V5.0 Cross Compiler
• Monterey/64 header files and libraries
• Startup scripts to override include and library paths
• Basic ReadMe documentation
Porting Centers

1st qtr: Setting up SPCs and training materials
2nd qtr: ISVs can get porting/training assistance
3rd qtr: Broader coverage w/partner centers added

Practical hands-on experiences

UnixWare SDE systems and Monterey/64 systems

- Itanium™ processor based systems to be available for scheduled test use

Over the year, centers will acquire “focus areas” for

- Performance tuning
- Scalability and high-end tuning
- Applications and systems available
- Porting and migration assistance

Porting Centers Available to you
Education Plans

- **Education Modules:**
  - Overview of IA-64 Architecture
  - Software Conventions
  - Assembly coding with Samples
  - Cross-Compiler flags, Usage with Samples
  - Shared Library Creation for Monterey vs AIX
  - Linking, Dynamic Linking/Loading
  - Debugger Commands and How-to-Use

Education to help you port to Monterey/64
Education Plans (cont)

- **Additional Modules:**
  - Unixware 7 course
  - Endian issues, samples, hands-on workshop
  - Migrating to 64 bit
  - Monterey feature differences vs AIX

- **Education will be customizable:**
  - Offered at Partnership Centers starting in April
  - Available online
  - Mix and match desired modules
Monterey Developers Website

- www.projectmonterey.com
- 700+ Developers signed up
- Links to Partners, Porting Resources
- Online Solution Developers Toolbox being established
- Migrating C and C++ Applications’ Guide

**SPC and Education sites**

Compilers and other Tools

IBM Compiler:

- VisualAge C/C++ Version 5.0 (ANSI 98)
- Same compiler as AIX
- Already used to compile Monterey/64 OS
- Includes IBM Distributed Debugger
- Cross-compiler is available for early development
- Native version available in beta timeframe

ANSI C/C++ Optimizing compiler
Alternative Compilers

- **Cygnus GNUpro Tools:**
  - gcc C Compiler
  - g++ C++ Compiler
  - gdb Debugger
  - gas Assembler
  - Same tools will be available on AIX

- **Availability Schedule for compilers/Debugger:**
  - Cygnus will support 5 beta customers
  - Beta 1-->> 5/2000; Beta 2-->>7/2000
  - GA-->>10/2000
Alternative Compilers

- Edinburgh Portable Compiler (EPC):
  - C Compiler
  - C++ Compiler ANSI 92
  - Fortran 95 Compiler with F77, VAX, Sun extensions
  - Debugger that supports all 3 languages
  - Same compilers will be available on AIX

- Availability Schedule for compilers/Debugger:
  - C/C++ Beta->> cross compiler 5/2000; Native 7/00
  - F95 Beta->> 5/2000
  - GA->> 10/2000 for all 3 compilers
Other Tools

- Perl
  - Beta->> with OS
  - GA->> with OS
- Apache Web Server
  - Beta->>
  - GA->>
- Java Support:
  - JVM V1.3
  - JIT
Contacting Us

Visit the Monterey developer web site @

www.projectmonterey.com

Visit the Monterey partner web sites @

www.ibm.com/servers/monterey
www.sco.com/monterey
www.sequent.com/monterey
The Volume Enterprise UNIX Platform
IBM - SCO - Intel

Monterey IHV Program
IHV Program Overview

- Monterey is a family of UNIX OS environments
- Consistent device driver model going forward
  - UDI
  - Support for Legacy driver models will continue
Project Monterey and UDI
What is Project UDI?

- Open industry group since 1993
  - Platform and OS vendors
  - IHVs
  - Solutions providers
- Enables 100% driver source portability
  - Defines architecture, APIs and packaging format
  - Supports source and binary distributions
- Provides uniformity across device types
  - Defines common execution model, inter-module communication and system services
  - Communication tailored to each device model
- Co-exists with legacy driver support
Why develop UDI?

- IHVs have huge matrix of drivers to develop/port
  - # Devices × OSes × OS versions × platforms
- Finite development & support resources
  - Must choose porting order (target prioritization)
  - Some OSes and/or platforms not supported
- Driver porting not core business
- UDI requires one driver source for all compliant OSes
  - UDI abstracts H/W and S/W environment
  - All driver interfaces completely specified
- More bang for the buck for IHVs
  - UDI moves up IHV porting order
  - UDI-compliant OSes get better coverage
UDI Encapsulates Drivers

Application Programs  I/O Requests

UDI Environment

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

UDI Drivers

Physical I/O Abstraction

Operating System

CPU and I/O Hardware (PIO, DMA, Interrupts)
Path From Application to Driver
Layered Implementation

Application

Embedding OS

I/O Subsystem

UDI Environment

OS-to-UDI External Mapper

UDI Services

UDI Driver

Physical I/O

Hardware Access

Adapter or System Hardware

Interrupts

Native Driver Interface

OS Requests

UDI Channel Operations
Path From Application to Driver
Integrated Implementation

Application

Embedding OS

I/O Subsystem

UDI Environment

UDI Services

UDI Driver

Physical I/O

Hardware Access

Interrupts

Adapter or System Hardware

OS Requests

UDI Channel Operations
Uniformity Across Devices

- Basic model common for all drivers
  - Execution and Data Models
    - Common buffer model
  - Configuration Model
  - Inter-Module Communication
    - Between drivers and/or environment modules
  - System Services and Utility Functions
UDI Metalanguages

- Device-type specific communication
- Defines communication paradigm between cooperating modules
  - Operations and sequences to implement technology-specific functionality
- Analogous to SCSI CAM, DLPI, etc.
UDI Execution Model

- No global entry points
- Driver’s udi_init_info structure contains entry-point pointers, size requirements...
- All driver code executed in the context of a *region*
  - Regions are associated with driver instances
    - One for each adapter/device controlled
Example Driver Hierarchy
UDI Regions

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

- No direct data sharing between regions
  - Data and events are passed through channels

- Provide implicit multi-processor synchronization
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
UDI Channels
Basis for Inter-Module Communication (IMC)

- Bi-directional channels connect regions
- Communication via channel operations
  - Strongly typed function-call interface
  - Asynchronous one-way operations
    - Each request has a corresponding response
    - Context managed via control blocks
UDI Channel Communications

Region A
- control block
- channel handle
- channel endpoints
- communication channel

Region B
- region data
- channel context
- control block
- channel ops vector
- entry point

Channel Operation
UDI Metalanguages

- Metalanguages define:
  - Number and types of channels
  - Valid Channel operations
    - Control block plus metalanguage-specific parameters
    - Control block types for each operation
      - Structures include metalanguage-specific fields
      - Generic control block header common to all
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
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
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 region private to that region
  - Regions share data with 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 begin with generic `udi_cb_t`
Implicit Synchronization

- No locking primitives required in UDI
  - All data access is implicitly synchronized
    - Region data accessible only from within that region
    - No global data
    - Only one thread per region active at a time
    - Other calls deferred until active call returns
  - Driver controls its parallelism by picking number and type of regions
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
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
Additional Metalanguages

- Physical I/O
  - DMA
  - Interrupts
  - Programmed I/O
  - PCI Bus bindings
- Network Interface (NIC drivers)
- SCSI
Developing Drivers Today

- Native drivers developed on AIX PPC or prototype system
  - UnixWare 7 cross compiler
- UDI driver development kit on UnixWare 7
  - Develop test and run on Unixware 7
  - Cross compile for Monterey/64
  - www.sco.com/udi/sco/udidk.html
Monterey/64 Native DDK

- Alpha (Native model)
  - Installs on UW7.1
  - Requires Monterey/64 SDK to build drivers
  - Toronto compiler also runs on UW7.1
  - Test results on prototype system
Monterey/64 Native DDK - Cont’d

- Beta
  - Installs on Monterey/64
  - Supports UDI and non-UDI models
  - Requires Monterey/64 SDK to build drivers
  - Toronto compiler also runs on Monterey/64
  - Test results on prototype system
Monterey/64 Native DDK - Contents

- Documentation
  - DDK specific (Installation, reqs., etc.)
  - Reference and guide material
  - Support information
  - Device driver API
  - Device man pages
Monterey/64 Native DDK - Contents cont’d

- Sample source code
  - Explanation of the key code blocks
  - Build scripts
- Debugging tips and techniques (tracing, logging, etc.)
- Use of tools (debugger, log analysis, etc.)
Monterey/64 Native DDK - Contents cont’d

- Packaging and installing driver instructions
- Test suites (beta)
  - hbacert and ndcert ported to Monterey
  - UDIG compliant test suites for UDI, SCSI & NIC
Monterey/64 Native DDK - Delivery

- Alpha - web based delivery
  - web based doc search
  - HTML format

- Beta - CD and web based delivery
  - HTML format (maybe printable format also)

- All updates through the web
Monterey Native DDK Schedule

- DDK Schedule
  - Alpha
    - Hybrid environment
  - Beta
    - Native Monterey/64 environment

- FCS 9/00
Summary

- Monterey/64 supports existing and standard driver models
  - Legacy and UDI based
- Tools exist today to start driver development
  - UDI
    - UnixWare 7 environment
  - AIX / Native for legacy drivers
    - UnixWare 7 UDI environment and dev kit
- For more information contact
  - www.projectmonterey.com/ihv
  - www.project-udi.org