Linux on Sun Logical Domains

David S. Miller

Red Hat Inc.

linux.conf.au, MEL8OURNE, 2008

Outline

1. Background
   - SUN4V and Niagara
   - Sun's Logical Domains
2. Userland Simulator
3. Implementation
   - LDC: Logical Domain Channels
   - VIO: Virtual I/O
   - DS: Domain Services
   - VNET: Virtual Network
   - VDC: Virtual Disk Client
   - Console
4. Challenges/Futures

Niagara: All Virtual, All the Time

- The “V” in SUN4V stands for Virtualized
- Most of the hardware is only hypervisor accessible, even on a non-virtualized node.
- Supervisor makes hypercalls using software traps.
- Supervisor only sees real addresses.
- I/O devices behind PCI, however can be directly programmed

Niagara: 64-bit Sparc traps

- Traps vectored as offset from Trap Base Address Register.
- Each trap slot is 8 instructions (32 bytes).
- Extremely simple traps done inline.
- More complicated work branches out to rest of handler.
- “Very Important” traps given multiple slots (f.e. TLB misses)
- Half of trap table for hardware exceptions, half for SW traps.
- SW traps are for system calls etc.
- Special SW traps are used for hypercalls.

Niagara: Hypercalls

- Looks like a system call.
- Arguments passed in outgoing argument registers (o0-o4).
- Hypercall number passed in o5.
- Status always returned in o0.
- o1-o5 can provide other return value state.

```
mov     cpuid, %o0
mov     HV_FAST_CPU_STOP, %o5
ta      HV_FAST_TRAP
cmp     %o0, HV_EOK
bne     cpu_stop_error
 nop
```

Niagara: Fast Hypercalls

- Dedicated SW trap vector
- No need to indicate call in o5, available for args
- Used for TLB load/flush and trap tracing.

```
mov     vaddr, %o0
mov     tlb_context, %o1
mov     pte, %o2
mov     HV_MMU_IMMU, %o3
ta      HV_MMU_MAP_ADDR_TRAP
cmp     %o0, HV_EOK
bne     itlb_load_error
 nop
```

LDOM Node types

1. Control node: has full access to devices and primary console.
2. Service node: has access to some physical devices.
3. Guest node: has only virtualized devices.

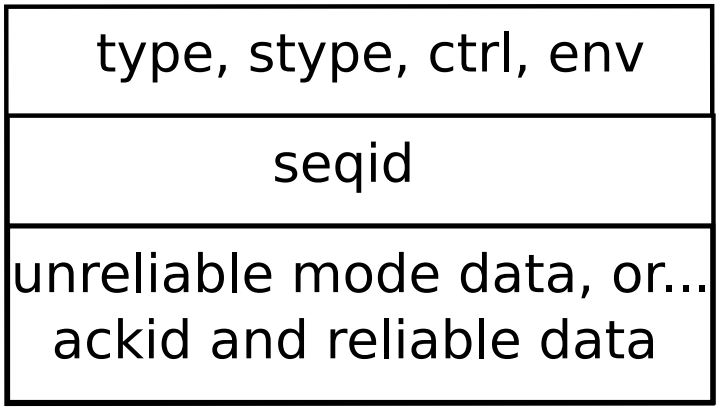
MD: Machine Description

- Complete logical description of machine the node executes on.
- Provided by hypervisor as a compact datastructure.
- Stored on the ALOM/ILOM.
- Dynamically updated.
- Control node constructs MDs for service and guest nodes.

LDC: Logical Domain Channel

- Communications link between nodes, via hypervisor.
- Bidirectional communications path, each end of the channel establishes a receive and transmit queue.
- Simple fixed sized, 64-byte, packets.
- Initial handshake establishes protocol version and synchronizes connection.
- If receive queue of either endpoint is unregistered, this resets the channel.

LDC: Packet format



1. type: indicates control, data, error
2. stype: indicates INFO, ACK, NACK
3. ctrl: indicates type of control packet
4. env: gives fragmentation state

LDC: Map Table Entries

Real Address	CW	CR	IW	IR	RWX	SZ

- Allows memory transfers between nodes.
- Similar to MMU or IOMMU PTE.
- Provides for transfer type protection.
  1. COPY: read and write
  2. IOMMU: read and write
  3. MMU: exec read and write
- LDC COPY operations have alignment restrictions.

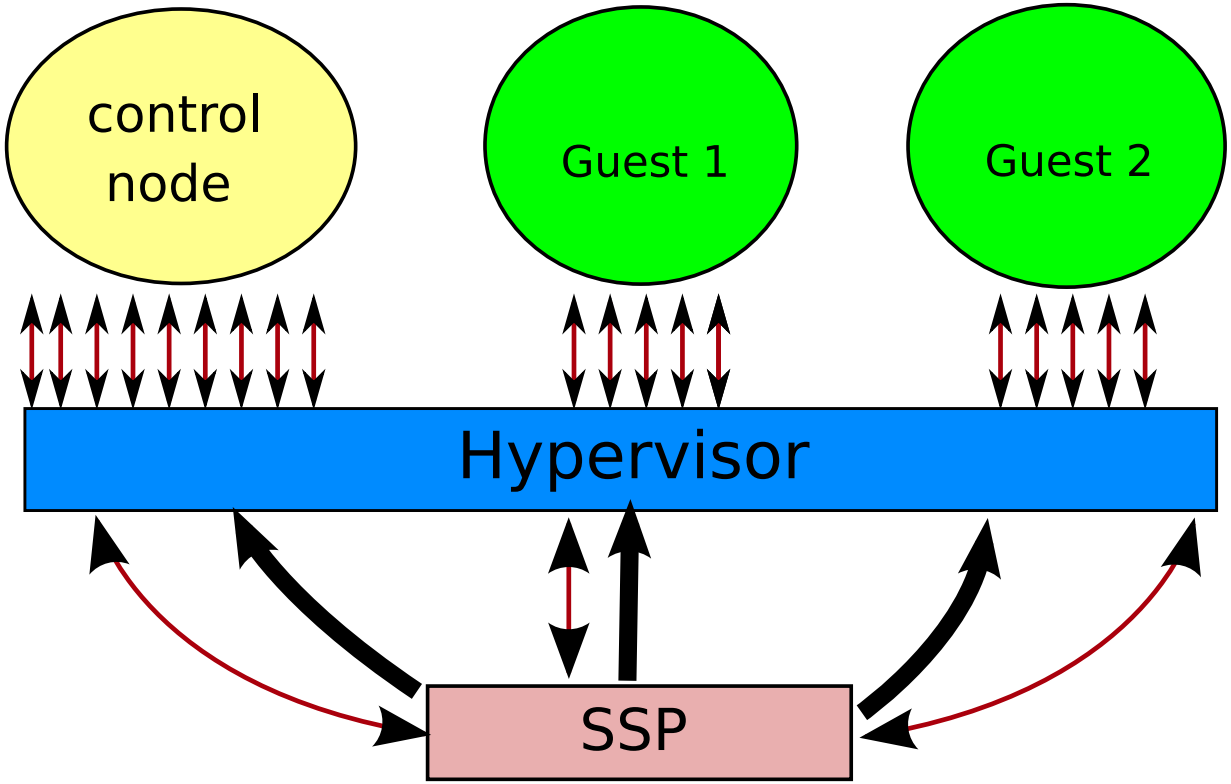
VIO: Virtual I/O

- I/O protocol built on top of channels.
- Just like LDC, has a handshake to synchronize, negotiate protocol versions, and to negotiate I/O parameters.
- Definitions exist for block, network, and console devices.

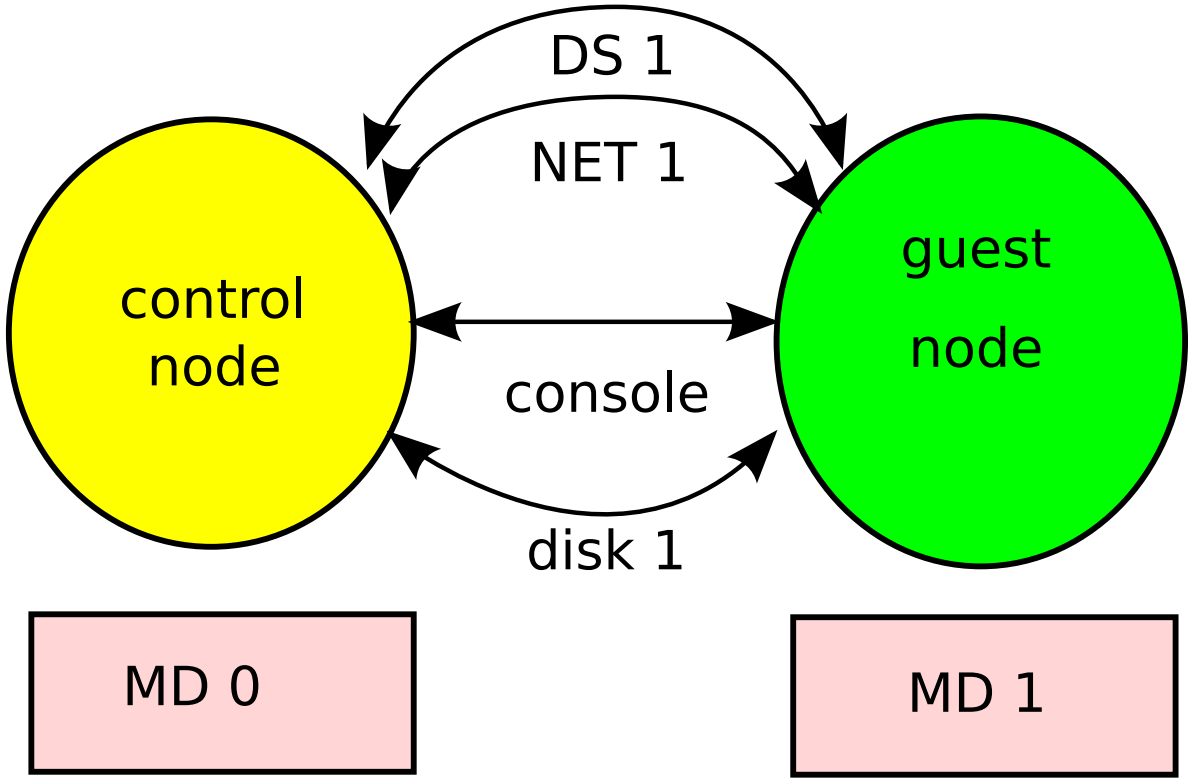
DS: Domain Services

- Miscellaneous communications, again built on top of channels.
- Remote reboot of guests.
- CPU hotplug.
- Machine description updates.
- Setting persistent firmware variables such as the boot device.

LDC: Example System



LDC: Zooming In



Purpose

- Userland is great for fast prototyping and debugging.
- Userland “reboots” faster.
- I had ethical issues with installing Solaris on my computers
- But I’m over that now...

Implementation

- Software implementation of all LDC hypervisor calls.
- Use same C interfaces as the kernel does.
- LDC protocol module could be compiled both in userland and kernel.
- Subsequently, VIO layer built on top could be just as flexible.
- Problem: Initially only compatible with itself.

TX Interfaces

```
unsigned long sun4v_ldc_tx_qconf(unsigned long id,
                                 unsigned long ra,
                                 unsigned long num_entries);

unsigned long sun4v_ldc_tx_qinfo(unsigned long id,
                                 unsigned long *ra,
                                 unsigned long *num_entries);

unsigned long sun4v_ldc_tx_get_state(unsigned long id,
                                     unsigned long *head,
                                     unsigned long *tail,
                                     unsigned long *state);

unsigned long sun4v_ldc_tx_set_qtail(unsigned long id,
                                     unsigned long tail);
```

RX Interfaces

```
unsigned long sun4v_ldc_rx_qconf(unsigned long id,
                                 unsigned long ra,
                                 unsigned long num_entries);

unsigned long sun4v_ldc_rx_qinfo(unsigned long id,
                                 unsigned long *ra,
                                 unsigned long *num_entries);

unsigned long sun4v_ldc_rx_get_state(unsigned long id,
                                     unsigned long *head,
                                     unsigned long *tail,
                                     unsigned long *state);

unsigned long sun4v_ldc_rx_set_qhead(unsigned long id,
                                     unsigned long head);
```

Client LDC Interfaces, Part 1

- Clients work with opaque “ldc channel” object.
- Creation, destruction, and state management.
  1. Allocate
  2. Free
  3. Bind
  4. Connect
  5. Disconnect
  6. Get current state

Client LDC Interfaces, Part 2

- Data Transfer
  1. Write
  2. Read
- Mapping Translation Management
  1. Map SG, Map Single
  2. Unmap
  3. Copy
  4. DRING Alloc and Free helpers (for VIO)

Virtual Device Layer

- Tree of “struct vio_dev” nodes.
- Dummy root, all virtual devices underneath.
- Populated by machine description notifier.
  1. Notifier registration triggers MD add events.
  2. All initial devices created.
  3. Future hot-plug triggers MD add/remove.
- Infrastructure closely mimicks powerpc VIO layer.

VIO Device Properties

- Three properties in MDESC node for VIO device.
- LDC channel ID
- LDC RX interrupt
- LDC TX interrupt
- Device type specific properties
  1. Network MAC address, port type
  2. Device Number, mainly for disks
  3. Etc.

VIO Driver Helpers

- Driver Init: validate config and setup helper state
- LDC Alloc: Allocated LDC channel and records state
- LDC Free: Shut down LDC channel and free state (incl. DRINGS)
- LDC Port Up: Bring LDC port up, retrying periodically
- Handshake Engine: Runs handshake using driver callbacks
- LDC Link State: Bulk of link UP/DOWN work
- LDC Send: Looping LDC write retry with delay

VIO Driver Flow

1. vio_driver_init()
2. vio_ldc_alloc()
3. Allocate TX DRING and buffers if needed
4. Device UP: vio_port_up()
5. Port UP: Run handshake, obtain attributes
6. Send work on TX DRING using DATA+INFO
7. Process incoming TX DRING DATA+ACKs
8. Receive work on RX DRING as DATA+INFO
9. Send RX DRING work DATA+ACKs

DS Basics

- YAHS: Yet Another HandShake
- Packet Classes: VER, REG, UNREG, DATA
- Services
  1. md-update: Machine Description Update
  2. domain-shutdown: Remote /sbin/shutdown
  3. domain-panic: Remote panic()
  4. dr-cpu: CPU Hot-Plug
  5. pri: Physical Resource Inventory
  6. var-config: Firmware variable handling

Salient DS Details

- Two DS LDC channels
  1. Primary to Control Node
  2. Backup to Service Processor
- PRI arrives as MDESC-like data block
- Firmware variables can be set and deleted
- Firmware variables stored on Service Processor with MDESC
- DS work processed in kernel thread

VNET Attributes and Calls

- Packet transfer mode
- Remote MAC address
- MTU
- Multicast list upload

VNET Switch

- Control node implements a switch
- Switch connects to guests and outside network
- Guests have links to switch
- Guests also may have links to other guests

VDC Attributes

- Packet transfer mode
- Block size
- Maximum transfer size
- Bitmask of supported operation

VDC Specific Calls

- Block read and write
- Flush (I/O barrier)
- Get/set write cache enable
- Get/set VTOC (disk label)
- Get/set EFI (disk label)
- Get/set geometry
- SCSI command submission

Console: Guest And Service Node Side

- Nothing to do
- Use normal hypervisor console write/read
- LDC endpoint exists internal to hypervisor
- Hypervisor sends/receives LDC packets

Console: Control Node Side

- Implements VCC, Virtual Console Concentrator
- Console accessed by telnetting to various ports
- One port per guest or service node

Implementation Challenges

- Handshake initiation
- VIO sequence number handling
- VIO disk label ownership (fixed now)
- VIO variable sized packet data structures

Things TODO...

- Fault tolerance of control node crash
- Fill in missing VDC stuff (SCSI I/O, EFI, etc.)
- Infrastructure for Linux as control node
  1. LDC channel usage in userland
  2. Control node userland daemon
  3. Configuration framework
  4. VCC console server

Summary

- LDOMs is a framework for full virtualization on Niagara systems
- Userland prototyping of support can help enormously
- Linux works as a full guest node
- VDC, VNET, and DS implemented
- Specification and implementation are two different things