Linux Virtualization on IBM POWER5 Systems

Dave Boutcher
IBM
`boutcher@us.ibm.com`

Dave Engebretsen
IBM
`engebret@us.ibm.com`

Abstract

In 2004 IBM® is releasing new systems based on the POWER5™ processor. There is new support in both the hardware and firmware for virtualization of multiple operating systems on a single platform. This includes the ability to have multiple operating systems share a processor. Additionally, a hypervisor firmware layer supports virtualization of I/O devices such as SCSI, LAN, and console, allowing limited physical resources in a system to be shared.

At its extreme, these new systems allow 10 Linux images per physical processor to run concurrently, contending for and sharing the system's physical resources. All changes to support these new functions are in the 2.4 and 2.6 Linux kernels.

This paper discusses the virtualization capabilities of the processor and firmware, as well as the changes made to the PPC64 kernel to take advantage of them.

1 Introduction

IBM's new POWER5** processor is being used in both IBM iSeries® and pSeries® systems capable of running any combination of Linux, AIX®, and OS/400® in logical partitions. The hardware and firmware, including a *hypervisor* [AAN00], in these systems provide the ability to create "virtual" system images with virtual hardware. The virtualization technique used on POWER™ hardware is known as paravirtualization, where the operating system is modified in select areas to make calls into the hypervisor. PPC64 Linux has been enhanced to make use of these virtualization interfaces. Note that the same PPC64 Linux kernel binary works on both virtualized systems and previous "bare metal" pSeries systems that did not offer a hypervisor.

All changes related to virtualization have been made in the kernel, and almost exclusively in the PPC64 portion of the code. One challenge has been keeping as much code common as possible between POWER5 portions of the code and other portions, such as those supporting the Apple G5.

Like previous generations of POWER processors such as the RS64 and POWER4™ families, POWER5 includes hardware enablement for logical partitioning. This includes features such as a hypervisor state which is more privileged than supervisor state. This higher privilege state is used to restrict access to system resources, such as the hardware page table, to hypervisor only access. All current systems based on POWER5 run in a hypervised environment, even if only one partition is active on the system.

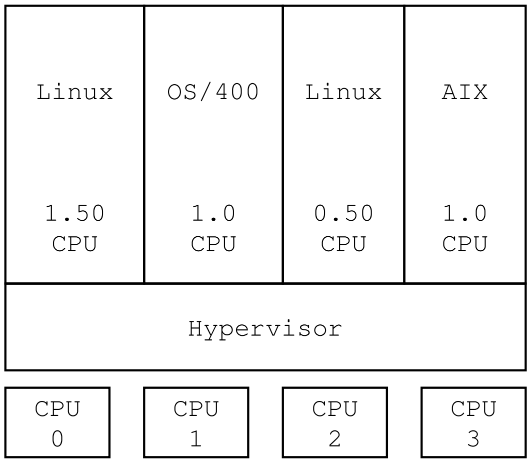

Figure 1: POWER5 Partitioned System

2 Processor Virtualization

2.1 Virtual Processors

When running in a partition, the operating system is allocated virtual processors (VP's), where each VP can be configured in either shared or dedicated mode of operation. In shared mode, as little as 10%, or 10 *processing units*, of a physical processor can be allocated to a partition and the hypervisor layer timeslices between the partitions. In dedicated mode, 100% of the processor is given to the partition such that its capacity is never multiplexed with another partition.

It is possible to create more virtual processors in the partition than there are physical processors on the system. For example, a partition allocated 100 processing units (the equivalent of 1 processor) of capacity could be configured to have 10 virtual processors, where each VP has 10% of a physical processor's time. While not generally valuable, this extreme configuration can be used to help test SMP configurations on small systems.

On POWER5 systems with multiple logical partitions, an important requirement is to be able to move processors (either shared or dedicated) from one logical partition to another. In the case of dedicated processors, this truly means moving a CPU from one logical partition to another. In the case of shared processors, it means adjusting the number of processors used by Linux on the fly.

This "hotplug CPU" capability is far more interesting in this environment than in the case that the covers are going to be removed from a real system and a CPU physically added. The goal of virtualization on these systems is to dynamically create and adjust operating system images as required. Much work has been done, particularly by Rusty Russell, to get the architecture independent changes into the mainline kernel to support hotplug CPU.

Hypervisor interfaces exist that help the operating system optimize its use of the physical processor resources. The following sections describe some of these mechanisms.

2.2 Virtual Processor Area

Each virtual processor in the partition can create a *virtual processor area* (VPA), which is a small (one page) data structure shared between the hypervisor and the operating system. Its primary use is to communicate information between the two software layers. Examples of the information that can be communicated in the VPA include whether the OS is in the idle loop, if floating point and performance counter register state must be saved by the hypervisor between operating system dispatches, and whether the VP is running in the partition's operating system.

2.3 Spinlocks

The hypervisor provides an interface that helps minimize wasted cycles in the operating system when a lock is held. Rather than simply spin on the held lock in the OS, a new hypervisor call, `h_confer`, has been provided. This interface is used to confer any remaining virtual processor cycles from the lock requester to the lock holder.

The PPC64 spinlocks were changed to identify the logical processor number of the lock holder, examine that processor's VPA *yield count* field to determine if it is not running in the OS (even values indicate the VP is running in the OS), and to make the `h_confer` call to the hypervisor to give any cycles remaining in the virtual processor's timeslice to the lock holder. Obviously, this more expensive leg of spinlock processing is only taken if the spinlock cannot be immediately acquired. In cases where the lock is available, no additional pathlength is incurred.

2.4 Idle

When the operating system no longer has active tasks to run and enters its idle loop, the `h_cede` interface is used to indicate to the hypervisor that the processor is available for other work. The operating system simply sets the VPA *idle* bit and calls `h_cede`. Under this call, the hypervisor is free to allocate the processor resources to another partition, or even to another virtual processor within the same partition. The processor is returned to the operating system if an external, decrementer (timer), or interprocessor interrupt occurs. As an alternative to sending an IPI, the ceded processor can be awoken by another processor calling the *h_prod* interface, which has slightly less overhead in this environment.

Making use of the cede interface is especially important on systems where partitions configured to run *uncapped* exist. In uncapped mode, any physical processor cycles not used by other partitions can be allocated by the hypervisor to a non-idle partition, even if that partition has already consumed its defined quantity of processor units. For example, a partition that is defined as uncapped, 2 virtual processors, and 20 processing units could consume 2 full processors (200 processing units), if all other partitions are idle.

2.5 SMT

The POWER5 processor provides symmetric multithreading (SMT) capabilities that allow two threads of execution to simultaneously execute on one physical processor. This results in twice as many processor contexts being presented to the operating system as there are physical processors. Like other processor threading mechanisms found in POWER RS64 and Intel® processors, the goal of SMT is to enable higher processor utilization.

At Linux boot, each processor thread is discovered in the open firmware device tree and a logical processor is created for Linux. A command line option, `smt-enabled = [on, off, dynamic]`, has been added to allow the Linux partition to config SMT in one of three states. The *on* and *off* modes indicate that the processor always runs with SMT either on or off. The dynamic mode allows the operating system and firmware to dynamically configure the processor to switch between threaded (SMT) and a single threaded (ST) mode where one of the processor threads is dormant. The hardware implementation is such that running in ST mode can provide additional performance when only a single task is executing.

Linux can cause the processor to switch between SMT and ST modes via the `h_cede` hypervisor call interface. When entering its idle loop, Linux sets the VPA *idle* state bit, and after a selectable delay, calls `h_cede`. Under this interface, the hypervisor layer determines if only one thread is idle, and if so, switches the processor into ST mode. If both threads are idle (as indicated by the VPA *idle* bit), then the hypervisor keeps the processor in SMT mode and returns to the operating system.

The processor switches back to SMT mode if an external or decrementer interrupt is presented, or if another processor calls the `h_prod` interface against the dormant thread.

3 Memory Virtualization

Memory is virtualized only to the extent that all partitions on the system are presented a contiguous range of logical addresses that start at zero. Linux sees these logical addresses as its real storage. The actual real memory is allocated by the hypervisor from any available space throughout the system, managing the storage in *logical memory blocks* (LMB's). Each LMB is presented to the partition via a memory node in the open firmware device tree. When Linux creates a mapping in the hardware page table for effective addresses, it makes a call to the hypervisor (`h_enter`) indicating the effective and partition logical address. The hypervisor translates the logical address to the corresponding real address and inserts the mapping into the hardware page table.

One additional layer of memory virtualization managed by the hypervisor is a *real mode offset* (RMO) region. This is a 128 or 256 MB region of memory covering the first portion of the logical address space within a partition. It can be accessed by Linux when address relocation is off, for example after an exception occurs. When a partition is running relocation off and accesses addresses within the RMO region, a simple offset is added by the hardware to generate the actual storage access. In this manner, each partition has what it considers logical address zero.

4 I/O Virtualization

Once CPU and memory have been virtualized, a key requirement is to provide virtualized I/O. The goal of the POWER5 systems is to have, for example, 10 Linux images running on a small system with a single CPU, 1GB of memory, and a single SCSI adapter and Ethernet adapter.

The approach taken to virtualize I/O is a cooperative implementation between the hypervisor and the operating system images. One operating system image always "owns" physical adapters and manages all I/O to those adapters (DMA, interrupts, etc.)

The hypervisor and Open Firmware then provide "virtual" adapters to any operating systems that require them. Creation of virtual adapters is done by the system administrator as part of logically partitioning the system. A key concept is that these virtual adapters do not interact in any way with the physical adapters. The virtual adapters interact with other operating systems in other logical partitions, which may choose to make use of physical adapters.

Virtual adapters are presented to the operating system in the Open Firmware device tree just as physical adapters are. They have very similar attributes as physical adapters, including DMA windows and interrupts.

The adapters currently supported by the hypervisor are virtual SCSI adapters, virtual Ethernet adapters, and virtual TTY adapters.

4.1 Virtual Bus

Virtual adapters, of course, exist on a virtual bus. The bus has slots into which virtual adapters are configured. The number of slots available on the virtual bus is configured by the system administrator. The goal is to make the behavior of virtual adapters consistent with physical adapters. The virtual bus is *not* presented as a PCI bus, but rather as its own bus type.

4.2 Virtual LAN

Virtual LAN adapters are conceptually the simplest kind of virtual adapter. The hypervisor implements a switch, which supports 802.1Q semantics for having multiple VLANs share a physical switch. Adapters can be marked as 802.1Q aware, in which case the hypervisor expects the operating system to handle the 802.1Q VLAN headers, or 802.1Q unaware, in which case the hypervisor connects the adapter to a single VLAN. Multiple virtual Ethernet adapters can be created for a given partition.

Virtual Ethernet adapters have an additional attribute called "Trunk Adapter." An adapter marked as a Trunk Adapter will be delivered all frames that don't match any MAC address on the virtual Ethernet. This is similar, but not identical, to promiscuous mode on a real adapter.

For a logical partition to have network connectivity to the outside world, the partition owning a "real" network adapter generally has both the real Ethernet adapter and a virtual Ethernet adapter marked as a Trunk adapter. That partition then performs either routing or bridging between the real adapter and the virtual adapter. The Linux bridge-utils package works well to bridge the two kinds of networks.

Note that there is no architected link between the real and virtual adapters, it is the responsibility of some operating system to route traffic between them.

The implementation of the virtual Ethernet adapters involves a number of hypervisor interfaces. Some of the more significant interfaces are `h_register_logical_lan` to establish the initial link between a device driver and a virtual Ethernet device, `h_send_logical_lan` to send a frame, and `h_add_logical_lan_buffer` to tell the hypervisor about a data buffer into which a received frame is to be placed. The hypervisor interfaces then support either polled or interrupt driven notification of new frames arriving.

For additional information on the virtual Ethernet implementation, the code is the documentation (`drivers/net/ibmveth.c`).

4.3 Virtual SCSI

Unlike virtual Ethernet adapters, virtual SCSI adapters come in two flavors. A "client" virtual SCSI adapter behaves just as a regular SCSI host bus adapter and is implemented within the SCSI framework of the Linux kernel. The SCSI mid-layer issues standard SCSI commands such as Inquiry to determine devices connected to the adapter, and issues regular SCSI operations to those devices.

A "server" virtual SCSI adapter, generally in a different partition than the client, receives all the SCSI commands from the client and is responsible for handling them. The hypervisor is not involved in what the server does with the commands. There is no requirement for the server to link a virtual SCSI adapter to any kind of real adapter. The server can process and return SCSI responses in any fashion it likes. If it happens to issue I/O operations to a real adapter as part of satisfying those requests, that is an implementation detail of the operating system containing the server adapter.

The hypervisor provides two very primitive interpartition communication mechanisms on which the virtual SCSI implementation is built. There is a queue of 16 byte messages referred to as a "Command/Response Queue" (CRQ). Each partition provides the hypervisor with a page of memory where its receive queue resides, and a partition wishing to send a message to its partner's queue issues an `h_send_crq` hypervisor call. When a message is received on the queue, an interrupt is (optionally) generated in the receiving partition.

The second hypervisor mechanism is a facility for issuing DMA operations between partitions. The `h_copy_rdma` call is used to DMA a block of memory from the memory space of one logical partition to the memory space of another.

The virtual SCSI interpartition protocol is implemented using the ANSI "SCSI RDMA Protocol" (SRP) (available at `http://www.t10.org`). When the client wishes to issue a SCSI operation, it builds an SRP frame, and sends the address of the frame in a 16 byte CRQ message. The server DMA's the SRP frame from the client, and processes it. The SRP frame may itself contain DMA addresses required for data transfer (read or write buffers, for example) which may require additional interpartition DMA operations. When the operation is complete, the server DMA's the SRP response back to the same location as the SRP command came from and sends a 16 byte CRQ message back indicating that the SCSI command has completed.

The current Linux virtual SCSI server decodes incoming SCSI commands and issues block layer commands (`generic_make_request`). This allows the SCSI server to share any block device (e.g., `/dev/sdb6` or `/dev/loop0`) with client partitions as a virtual SCSI device.

Note that consideration was given to using protocols such as iSCSI for device sharing between partitions. The virtual SCSI SRP design above, however, is a much simpler design that does not rely on riding above an existing IP stack. Additionally, the ability to use DMA operations between partitions fits much better into the SRP model than an iSCSI model.

The Linux virtual SCSI client (`drivers/scsi/ibmvscsi/ibmvscsi.c`) is close, at the time of writing, to being accepted into the Linux mainline. The Linux virtual SCSI server is sufficiently unlike existing SCSI drivers that it will require much more mailing list "discussion."

4.4 Virtual TTY

In addition to virtual Ethernet and SCSI adapters, the hypervisor supports virtual serial (TTY) adapters. As with SCSI adapter, these can be configured as "client" adapters, and "server" adapters and connected between partitions. The first virtual TTY adapter is used as the system console, and is treated specially by the hypervisor. It is automatically connected to the partition console on the Hardware Management Console.

To date, multiple concurrent "consoles" have not been implemented, but they could be. Similarly, this interface could be used for kernel debugging as with any serial port, but such an implementation has not been done.

5 Dynamic Resource Movement

As mentioned for processors, the logical partition environment lends itself to moving resources (processors, memory, I/O) between partitions. In a perfect world, such movement should be done dynamically while the operating system is running. Dynamic movement of processors is currently being implemented, and dynamic movement of I/O devices (including dynamically adding and removing virtual I/O devices) is included in the kernel mainline.

The one area for future work in Linux is the dynamic movement of memory into and out of an active partition. This function is already supported on other POWER5 operating systems, so there is an opportunity for Linux to catch up.

6 Multiple Operating Systems

A key feature of the POWER5 systems is the ability to run different operating systems in different logical partitions on the same physical system. The operating systems currently supported on the POWER5 hardware are AIX, OS/400, and Linux.

While running multiple operating systems, all of the functions for interpartion interaction described above must work between operating systems. For example, idle cycles from an AIX partition can be given to Linux. A processor can be moved from OS/400 to Linux while both operating systems are active.

For I/O, multiple operating systems must be able to communicate over the virtual Ethernet, and SCSI devices must be sharable from (say) an AIX virtual SCSI server to a Linux virtual SCSI client.

These requirements, along with the architected hypervisor interfaces, limit the ability to change implementations just to fit a Linux kernel internal behavior.

7 Conclusions

While many of the basic virtualization technologies described in this paper existed in the Linux implementation provided on POWER RS64 and POWER4 iSeries systems [Bou01], they have been significantly enhanced for POWER5 to better use the firmware provided interfaces.

The introduction of POWER5-based systems converged all of the virtualization interfaces provided by firmware on legacy iSeries and pSeries systems to a model in line with the legacy pSeries partitioned system architecture. As a result much of the PPC64 Linux virtualization code was updated to use these new virtualization interface definitions.

8 Acknowledgments

The authors would like to thank the entire Linux/PPC64 team for the work that went into the POWER5 virtualization effort. In particular Anton Blanchard, Paul Mackerras, Rusty Rusell, Hollis Blanchard, Santiago Leon, Ryan Arnold, Will Schmidt, Colin Devilbiss, Kyle Lucke, Mike Corrigan, Jeff Scheel, and David Larson.

9 Legal Statement

References

[AAN00] Bill Armstrong, Troy Armstrong, Naresh Nayar, Ron Peterson, Tom Sand, and Jeff Scheel. *Logical Partitioning*, `http://www-1.ibm.com/servers/eserver/iseries/beyondtech/lpar.htm`.

[Bou01] David Boutcher, *The iSeries Linux Kernel* 2001 Linux Symposium, (July 2001).

Proceedings of the Linux Symposium

Volume One

July 21st–24th, 2004
Ottawa, Ontario
Canada