The Linux Kernel on iSeries

Dave Boutcher
IBM
Rochester, Minnesota
boutcher@us.ibm.com, http://www.ibm.com/iSeries/linux

Abstract

Between May 2000 and May 2001 PowerPC Linux was enabled to run on the IBM iSeries system. While the title of this talk could be "YALP" (Yet Another Linux Port) there are some interesting technical aspects to this port that make it worth discussing.

1 Introduction

The iSeries port was accomplished by a small group of programmers working at IBM in Rochester Minnesota. The port was made easier because of the extensive experience of working with the lowest levels of the PowerPC processor to deliver and optimize the OS/400 operating system.

This paper describes some of the implementation details of enabling Linux to run on the iSeries platform beside the OS/400 operating system. For additional details, see the code.

Specifically, the fact that the entire kernel always runs relocatable (i.e. with address translation occurring through a page table), and the fact that many I/O events are not delivered as interrupts, but rather as "event" on a queue are interesting twists on the existing PPC implementation.

Part I

iSeries Background

2 What is an iSeries?

The IBM iSeries system was known as "AS/400" until earlier this year. It is 100% a business computer. People buy it to run their stores, their payroll, their warehouses, their trucks.

Despite the plebeian tasks to which it is put, however, the technology has remained very advanced. One of the interesting things about the system is that it ranges from single processor systems (costing under $20,000 US) to the largest system with 24 680MHz PowerPC processors, 128 GB of memory and 18.5 TB of disk space (costing way too much to be worth mentioning here.)

The processor is identical to the IBM pSeries (aka the RS/6000) but the I/O structure is significantly different. In order to support features like 2000 disk units, I/O is off loaded to separate I/O Processor (IOP) cards, which in turn drive the I/O adapters.

The iSeries operating system is OS/400, and it has a few unique features:

- Single level store
  - Pointers are 128 bits long
  - All storage (including disk) is addressed as memory using these pointers
  - All processes share a single flat memory space

- Built in relational database
  - Built into the kernel
  - Used heavily by the operating system itself
- Ability to run Windows on a card plugged into the system

Additionally it has features such as

- A built-in Java^{TM} Virtual Machine
- Ability to run the Apache web server
- Ability to run AIX binaries
- Ability to logically partition the physical system up to 32 partitions.

A discussion of how a single address space can be shared in a secure fashion between multiple concurrent processes is fascinating, but beyond the scope of this paper. Suffice it to say that the architecture mandates the following:

- A hidden 65th bit (called a "tag") is associated with every quad-word (64-bit) in memory.
- The processor can be set such that a pointer is only valid if it has the tag bit on.
- Turning the tag bit on is a privileged instruction (i.e. only valid in the kernel.)

Consequently only the kernel can create pointers. See [Soltis] for additional information.

Similar with most other operating systems on the planet, OS/400 has both "user" state and "system" state, where user state code requests system state functions using the **sc** (System Call) processor instruction.

The PowerPC architecture allows both 32 bit and 64-bit processors. OS/400 has been a 64-bit system since 1995 and all iSeries systems ship with 64-bit PowerPC processors. 64-bit PowerPC processors also support running in 32 bit mode. While the PowerPC processor supports both little endian and big endian modes, OS/400 always runs in big endian.

The current iSeries processor is the "SStar" processor. Table 1 shows some of the features of this processor.

Two-way multithreaded
128KB on-chip L1 Instruction cache
128KB on-chip L1 Data Cache
On-chip L2 cache directory supporting up to
16MB of off-chip L2 cache
512-entry translation lookaside buffer
Four-way superscalar
Five-stage pipeline
32-byte-wide on-chip buses
500 - 680 MHz operating frequencies

Table 1: SStar Processor Features in iSeries machines

3 iSeries Logical Partitioning

The Logical Partitioning (LPAR) facility permits processors, portions of real storage, and I/O devices to be assigned to partitions. It is extremely important that a program executing in one partition not be able to interfere with any program executing in a different partition. This isolation is provided for both user and system state programs by using a layer of trusted software, called the hypervisor.

The iSeries system can be partitioned into as many as 32 logical partitions. While I/O devices are always owned by a single partition, processors and memory are shared by partitions. A partition can be assigned from 0.1 of a processor to 23.9 of a processor (assuming that some other partition is using the remaining 0.1) in increments of 0.01. The hypervisor provides pre-emptive timeslicing between partitions sharing a processor. If a partition is allocated 0.45 of a processor, the hypervisor will guarantee that it gets 0.45 of a processor, no more and no less, even if the remainder of the processor is unused.

Processors, memory and I/O devices can be added and removed from OS/400 partitions without requiring a re-IPL. This allows some interesting scenarios where, for example, a single system can be split into three partitions to support three different geographic areas. Each partition may be running using a different timezone and language. On a 24

hour clock cycle, processor and memory resources can be scheduled to be moved to the partition with the most activity.

The PowerPC architecture supports extensions to the standard and the IBM PowerPC processors used in the i and p series systems contain a number of extensions. In addition to support for the iSeries single level store memory model, the processor includes specific support for logically partitioning the system.

Specifically, a "Hypervisor" (HV) bit was defined in the Machine Status Register (MSR). This bit, along with the Problem State (PR) bit controls whether the processor is in hypervisor state.

LPAR provides an environment in which only the hypervisor can run with address translation disabled, and all interrupts except the System Call Vectored Interrupt invoke the hypervisor. It also ensures that manipulation of the Translation Lookaside Buffer can only be done from the hypervisor. This is key to guaranteeing partition isolation.

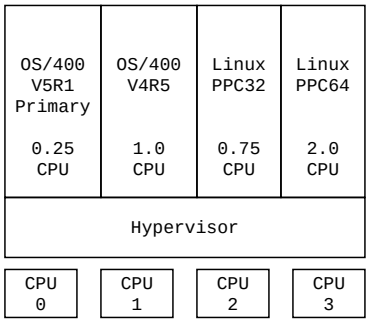

Figure 1: iSeries Partitioned System

4 Hardware Multi-Threading

In a hardware multithreaded processor (HMT), the processor holds the state of several tasks/threads. When one thread would normally be stalled because of a cache miss, instructions from the other threads can utilize the processor's resources. Current iSeries processor contain HMT support.

Measurements of commercial workloads, as opposed to the kind of tight processor loops found in many scientific calculations, have demonstrated up to a 30% performance improvement from HMT.

To the operating system, the processor "appears" as multiple processors. In general, all thread switching on an HMT processor is done by the hardware without intervention by the operating system. In some cases, however, intelligent hints to the processor can have dramatic impacts on the behavior of the system. The spin-lock code and the idle loop are instances where it is appropriate to hint to the processor that a thread should *not* work too hard, even without cache misses.

The typical no-op instruction in the PPC instruction set is **or 0,0,0** (or GPR0[1] with GPR0, storing the result in GPR0.) In what can be described either as an incredible hack or a very elegant side effect, the instructions **or 1,1,1**, **or 2,2,2**, **or 3,3,3** (which functionally act as no-ops) indicate to the HMT hardware that this thread should be given low, medium, or high priority related to other HMT threads.

See [Borkenhagen] for an excellent technical article on hardware multithreading in the PowerPC processor.

5 Linux in a Logical Partition

The decision was made to implement iSeries Linux only in a Logical Partition. If someone wants to run Linux on a nice, black, IBM PowerPC system, the pSeries[2] is available. Since all iSeries systems come with the deep (and proprietary) IOP based I/O structure, significant I/O rework would be required to support Linux on the cold hardware.

The initial port of Linux to the iSeries was based on the existing 32 bit PPC kernel as the 64-bit PPC kernel was not yet available. Since the existing PPC Linux implementations are big endian, this fits well with the existing interfaces to the hypervisor and the remainder of the system.

[1] General Purpose Register 0
[2] aka RS/6000

Part II

Linux Kernel Changes

6 Existing PPC Implementation

The iSeries kernel port began with the existing 32 bit Linux PPC architecture. The goal was to keep the iSeries support being a sub-architecture of PPC, as opposed to creating a new architecture. Additionally, the changes will be merged into the 64-bit PPC architecture as it is developed. Since PPC already has a few different sub-architectures, this is a natural extension. The new processor type screen on the PPC menuconfig is now shown in Table 2.

An even more important (and obvious) goal was that all PPC programs outside the kernel should run unchanged on the iSeries system. At the time of writing it is possible to take the existing SuSE 7.1 PPC CDs and, by using an updated kernel, install and run all the software. Other than software which is very hardware specific (e.g. there are no sound cards supported on the iSeries) we have not currently encountered any software that doesn't work.

Since the majority of the processors that support Linux on the iSeries have HMT capability, it is expected that kernels used on the iSeries will be built with SMP. Table 3 shows the /proc/cpuinfo file on a single processor partition with HMT enabled.

```
processor       : 0
cpu             : I-star
clock           : 500.04Mhz
time base       : 500.04MHz
i-cache         : 128
d-cache         : 128
revision        : 0.1
bogomips        : 470.72

processor       : 1
cpu             : I-star
clock           : 500.04Mhz
time base       : 500.04MHz
i-cache         : 128
d-cache         : 128
revision        : 0.1
bogomips        : 470.72

total bogomips  : 940.94
machine         : iSeries Logical Partition
```

Table 3: iSeries /proc/cpuinfo with HMT

7 Being Well Behaved

Linux running in an iSeries logical partition is restricted by the processor state from direct control of many hardware functions. As mentioned earlier, it is imperative that an operating system running in a logical partition not, either accidentally or maliciously, be able to affect another partition. A consequence of this is that many operations, such as setting up the Hardware Page Table, that are performed directly in the other flavours of the PPC architecture are performed by calling the hypervisor in the iSeries implementation.

The hypervisor requires that a partition demonstrate "aliveness" by either making hypervisor calls, or responding to interrupts (which goes through the hypervisor.) A partition that does not demonstrate aliveness for 500ms will be terminated by the hypervisor. During development there were a few scenarios where the Linux kernel would have interrupts disabled for too long and be terminated by the hypervisor. Since the hypervisor preserves the registers at the time the partition is terminated, [3] it turned out to be an excellent debugging mechanism for problems that otherwise would have been very hard to isolate (hanging with interrupts disabled.) The mechanism whereby the hypervisor determines partition aliveness is termed the "heartbeat" mechanism. The hypervisor concludes the partition is dead if it does not detect heartbeats.

When the hypervisor preempts a shared processor running with HMT there are some interesting deadlocks. One scenario that was encountered involved two processors competing for a spinlock with interrupts disabled. The first processor got the spinlock and invoked a hypervisor call. Since the processor

[3] Interesting note: how do you terminate a processor that is running with interrupts disabled? The hypervisor invalidates the hardware page table for the partition causing a Data Storage Interrupt (DSI) which will be vectored through the hypervisor code.

```
+------------------------- Processor Type ------------------------+
|  Use the arrow keys to navigate this window or press the hotkey of |
|  the item you wish to select followed by the <SPACE BAR>. Press    |
|  <?> for additional information about this option.                 |
| +---------------------------------------------------------------+ |
| |                     ( ) 6xx/7xx/7400/8260                     | |
| |                     ( ) 4xx                                   | |
| |                     ( ) POWER3                                | |
| |                     ( ) POWER4                                | |
| |                     ( ) 8xx                                   | |
| |                     (X) ISERIES_LPAR                          | |
| +---------------------------------------------------------------+ |
+---------------------------------------------------------------------+
|                       <Select>      < Help >                        |
+---------------------------------------------------------------------+
```

Table 2: menuconfig Processor Types

had reached the end of it's timeslice, the hypervisor attempted to interrupt the second thread. Unfortunately the second thread was waiting on the spinlock with interrupts disabled. Eventually the hypervisor concluded that the second thread was dead and terminated the machine. The fix for this scenario was "soft disabling" interrupts (see below.)

8 What Changed

There are a number of PPC files specific to the various architectures. One of the main files handles entry to the kernel via exceptions, system calls, etc. This is an assembler file named "head.S" (or some variation on that. A fragment of the Makefile in arch/ppc/kernel is shown in Table 4.

Table 5 shows a mapping of some other files which are common in the other PPC flavours but have specific iSeries versions.

9 Control Blocks

In order to set up a well managed interface between the hypervisor and the operating system in the logical partition, an architected set of control blocks is built into the kernel. Some of the information in these blocks is examined by the hypervisor before

```
ifeq (\$(CONFIG_4xx),y)
  KHEAD := head_4xx.o
else
  ifeq (\$(CONFIG_8xx),y)
    KHEAD := head_8xx.o
  else
    ifeq (\$(CONFIG_PPC_ISERIES), y)
   KHEAD := iSeries_head.o
    else
        KHEAD := head.o
    endif
  endif
endif
```

Table 4: head.S configuration

```
setup.c        -> iSeries_setup.c
hashtable.c    -> iSeries_hashtable.c
dma.c          -> iSeries_dma.c
```

Table 5: iSeries files

doing any initialization of the logical partition, and certainly before the kernel begins execution. For example, one of the blocks indicates how the initial load area should be mapped in memory (remember that the kernel always executes with address translation active) and how many pages to map.

The root of all control blocks is the "HvReleaseData" block. The hypervisor looks for an offset to that block at offset 0x20 in the kernel. The format of the HvReleaseData block as architected, includes flags indicating whether this OS supports:

- Tags active or Tags Inactive
- 32 or 64-bit addressing
- Processor sharing
- HMT
- Minimum supported Hypervisor version

There are two other significant architected control blocks that are used to exchange information between Linux and the hypervisor. There is one "NACA" (Node Architected Control Area) [4] for the partition, that contains information such as the logical partition number for this partition. Additionally it contains the interrupt vector with the addresses of the routines that the hypervisor will invoke for various types of exceptions.

There is one control block for each processor in the partition called the "PACA" (Processor Architected Control Area). The biggest use of this data structure is as a save location during interrupt processing. Special purpose register 1 (SPRG1) of each processor always points to the virtual address of the PACA associated with that processor. Upon an interrupt, the value stored in SPRG1 is read, converted to a real address, and then used as a base pointer to the PACA where a small amount of processor state is stored while relocation remains disabled and a kernel stack is not yet available. Additionally there are indicators for whether deferred interrupts have occurred (the hypervisor may defer interrupts that occur while the hypervisor itself is running.)

The NACA and PACAs are anchored (via a couple of levels of indirection) off the HvReleaseData block.

[4]The terms NACA and PACA come from the internal design of OS/400, and do not exist in the 32 bit PPC code other than the iSeries sections. They are, however, working their way into the 64-bit version because there is a need for a system wide control block, and processor specific control blocks and there didn't seem like a good reason to change the names

Other than locating and initializing these blocks, they do not drive significant changes into the PPC code.

9.1 Memory Management

In an iSeries partition, the operating system has no direct access to the hardware page table. The iSeries hypervisor manages the hardware page table, and is directed by the operating system in the partition. The hypervisor establishes addressibility for the first 64 MB of memory at 0xC0000000 by building a hashed page table and setting segment register 12.

The partition memory is not physically contiguous, nor necessarily addressable with a 32-bit real address. Obviously in a system with 32 partitions each allocated a gigabyte or more of memory, many addresses are high. Also, because of the way memory is allocated to partitions, and because of the fact that OS/400 partitions support dynamic addition and removal of memory, there are no guarantees about memory continuity.

The hypervisor provides functions which the kernel can use to discover the layout of memory. The iSeries LPAR specific code in the kernel builds a table that maps contiguous pseudo-real addresses starting at zero to the actual physical addresses owned by this partition. In 32-bit mode it is restricted to no more than 768 MB currently.

The iSeries system may have very large memories (up to 128 GB) and a partition may get memory in "chunks" that may be anywhere in the 2^{52} real address space. The chunks are 256K in size. To map this to the memory model Linux expects, the iSeries specific code builds a translation table to translate what Linux thinks are "physical" addresses to the actual real addresses. This allows it appear to Linux that there is contiguous memory starting at physical address zero while in fact this could be far from the truth.

To avoid confusion, let the words physical and real address apply to the Linux addresses while the term "absolute address" refers to the actual hardware real address.

iSeries_setup contains a routine which gets information from the Hypervisor to determine the ab-

solute addresses of the memory that has been assigned to the partition. It builds a table used to translate Linux's physical addresses to these absolute addresses. Absolute addresses are needed when communicating with the hypervisor (e.g. to build HPT entries). This table is called the mschunks table.

The bulk of the Linux PPC code then deals in "physical" addresses, and only the actual HPT mapping code does the translation to absolute addresses.

The load area is mapped at physical address 0, and the memory mapped by the hypervisor before the kernel is first invoked is mapped sequentially after that.

Since the PowerPC hardware page table cannot (in general) contain all required address translations, sometimes entries must be cast out to make room for new entries. The requirement to handle page faults with *relocation on* further requires that some entries (those required to handle page faults themselves) never be cast out. These special hardware page table entries are known as "bolted" entries.

1. Fault handler code. This code is identified either by its physical address (the first four pages of physical memory), or through use of the compiler directive __bolted.

2. Task structure and kernel stacks (one exists for each process)

3. Linux page tables (one exists for each process)

4. msChunks array. This data structure is used to convert Linux logical addresses to hardware physical addresses when building HPT entries.

5. NACA

6. PACA array. This object must be bolted because it might be used to store some data during the interrupt processing.

All of the above, except the kernel stacks and Linux page tables, are fixed at boot time and are bolted using their kernel (0xC000...) addresses.

See Table 6 for a boot log showing some of the memory mapping messages.

9.2 Interrupts

The PPC architecture has a number of special purpose registers. Specifically, SRR0 and SRR1 are named "Machine Status Save/Restore Registers" and are set when interrupts occur. The PPC architecture has no concept of automatically saving things on the stack on interrupts. SRR0 normally contains the next instruction address, and SRR1 contains the machine status. There are four special purpose software registers (SPRG0 to SPRG3) for use by privileged software.

When Linux interrupt handlers get control, the hypervisor has already saved SRR0 and SRR1 into the PACA control block shared between the hypervisor and Linux. The values in the actual SRR0 and SRR1 are not valid. The hypervisor reserves SPRG0 for its own use, which requires a change in the use of the the way the SPRG registers from the rest of the PPC architecture. The definitions are:

Register	iSeries Architecture
SPRG0	reserved for hypervisor
SPRG1	addr of PACA
SPRG2	temp - used to save gpr
SPRG3	Linux thread

Table 7: Special register mapping

It is frequently possible for an interrupt to occur either while a hypervisor system call is active, or (on a partition sharing a processor with another operating system) while the partition does not own the processor. In these cases the hypervisor will turn on a bit in the PACA indicating that a deferred interrupt has occurred. The kernel is responsible for checking these bits and running the appropriate interrupt vectors.

It is extremely significant to note that the *only* asynchronous interrupts that are processed on the iSeries are decrementer interrupts and Inter Processor Interrupts (IPIs.) Obviously there are lots of synchronous exceptions (such as accessing an invalid address, or floating point errors) that get vectored through the interrupt code.

All other I/O related events are delivered as LP Events (see Table 9.4 for a description of LP events.)

```
Mapping load area - physical addr = 0, absolute addr = 0000000012000000
Load area size 32768K
HPT absolute addr = 0000000026000000, size = 2048K
D-cache line size = 128  (log = 7)
I-cache line size = 128  (log = 7)
<5>mf.c: iSeries Linux LPAR Machine Facilities initialized
Total memory: 67108864 bytes
Progress: [0300] - MMU:hash init
Progress: [0105] - hash:enter
iSeries_hashinit: added 8192 hptes to existing mapping
Progress: [0205] - hash:done
Progress: [0301] - MMU:mapin
Linux version 2.4.30419UI (boutcher@TERRIER) (gcc version 2.95.2-5 19991024 (
Progress: [3eab] - setup_arch: enter
Progress: [3eab] - setup_arch: bootmem
Max  logical processors = 32
Max physical processors = 16
Processor frequency = 500.04
Time base frequency = 500.04
Processor version = 360001
```

Table 6: iSeries boot log

9.2.1 Soft Disable

To support shared processors on iSeries LPAR and the iSeries hypervisor heartbeat mechanism, the kernel only "soft" disables interrupts. External interrupts and decrementer interrupts remain enabled from the hardware's point of view even when Linux is disabled. If Linux is soft disabled, the external and decrementer interrupt handlers merely mark (in the processor's PACA) that an interrupt occurred and then return to the interrupted code. Only when interrupts are subsequently soft enabled, are the interrupt service routines driven.

9.3 System Calls from System Calls

There is one and only one system call instruction in the PowerPC architecture. That one instruction, however, is used to make two different transitions. User state code wanting to call the kernel loads GPR0 with the system call function it is invoking. In order to differentiate Linux (and OS/400) system calls from hypervisor calls, all hypervisor calls load -1 in GPR0 and store the hypervisor function in GPR3. The hypervisor will only accept system calls from privileged state preventing user mode code from trying to invoke hypervisor routines.

Whenever the system call instruction is executed the hypervisor will be invoked. If GPR0 contains -1 and the call came from privileged state, the hypervisor considers the call targeted at itself. If GPR0 contains some other value, or the call came from user state, the call is passed to the system call code in the Linux kernel.

9.4 Logical Partition Events

The hypervisor provides a mechanism for communicating between partitions. This mechanism is a queue of events, known as "LP Events" anchored off the PACA. In general, each partition has only one queue, so the queue pointer in the first PACA will be set and the pointer in all the other PACAs will be NULL. This means that all LP Event processing will be done by the first processor.

LP Events are intended to be small and fast. They are limited to 256 bytes. There are both "acknowledged" events, where the hypervisor guarantees that an acknowledgment will be returned for every event sent, and "unacknowledged" events that are the equivalent of UDP spray and pray.

The LP Event queue is key to the I/O structure of

Linux on an iSeries.

For data larger than 256 bytes, DMA addresses are sent and a DMA operation is done between partitions. The DMA infrastructure is used, however under the covers the hypervisor essentially does a memcpy.

10 Virtual Devices

Configuring an OS/400 logical partition is a complex operation. OS/400 expects boot disks, console devices, etc. all arranged in the correct IOP/IOA structure. A goal of Linux on iSeries was to keep the configuration extremely simple.

In order to do this, Linux on iSeries supports "Virtual" I/O where the I/O devices are provided by OS/400 and do not actually exist as hardware managed by Linux. This means that the minimum configuration for an iSeries partition is 64MB of memory and 1/10 of a processor. OS/400 then provides the following virtual devices to Linux:

- Virtual console
- Virtual disk
- Virtual Ethernet
- Virtual CD
- Virtual tape

All of these virtual devices are implemented with a Linux device driver that exchanges LP Events with an OS/400 partition. The LPAR configuration supports identifying one partition as "hosting" another. A hypervisor call exists so the Linux partition can request the identity of its hosting partition. Virtual I/O events are then sent to that partition.

Virtual Console ends up connecting to a terminal or telnet session connected to a port on OS/400. An interesting twist on this is that Linux initialized the `TERM` environment variable to `linux`, which can confuse the telnet client (which thinks it's talking something like VT100.) This can be worked around by setting a kernel parameter like `TERM=vt100`.

The virtual disk ends up being very much like a loopback file system file, where the disk file is managed by OS/400.

Virtual tape and virtual CD allow a path to the real tape and CD devices attached to OS/400. CD can be shared by many users concurrently, while tape is single-user (i.e. only OS/400 or only one Linux partition.)

Finally, the OS/400 LPAR environment supports up to 16 virtual ethernet segments that can be connected to any combination of logical partitions. The OS/400 display for configuring virtual ethernets is shown in table 8.

10.0.1 DMA Space

Given the discontiguous, 64-bit nature of the iSeries physical address space, it is necessary to map 32-bit PCI DMA addresses to 64-bit "real" (remembering the definition of "real" in an earlier section) addresses. On PowerPC, the translation entries are known as TCEs (Translation Control Entries). The management of TCEs is implemented within the pci_alloc_consistent, pci_free_consistent (and similar pci functions). The deprecated virt_to_bus, virt_to_phy functions are not implemented in Linux for iSeries.

The hypervisor does not allow direct access to the TCE table, therefore hypervisor calls must be made to add and delete TCE table entries.

For scatter/gather lists, pci_map_sg/pci_unmap_sg allow multiple buffers to be allocated to a contiguous range in DMA space as long as buffers fall on page boundaries. Since most buffers used by disk I/O are 4K blocks, which is the same as the page size, significant numbers of pages can be collapsed into single DMA space entries.

10.1 PCI I/O Space

MMIO address are manufactured addresses that don't really exists in Linux. They are mapped to the device in the Linux iSeries I/O support. The PCI I/O space is managed by the Hypervisor and only devices (PCI I/O space) that are assigned to the Linux partition are accessible. Direct memcpy functions will not work, however the memcpy_toio

```
                    Work with Virtual LAN Configuration
                                                              System:   ISERIESA
 Type options press Enter.
   2=Change

     Par          -------------------Virtual LAN Identifiers-------------------
 Opt ID  Name      0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
  _   0  PRIMARY   1   1   .   .   .   .   .   .   .   .   .   .   .   .   .   .
  _   1  LINUX1    1   1   .   .   .   .   .   .   .   .   .   .   .   .   .   .
  _   2  LINUX2    1   .   1   .   .   .   .   .   .   .   .   .   .   .   .   .
  _   3  LINUX3    1   .   1   .   .   .   .   .   .   .   .   .   .   .   .   .
  _   4  LINUX4    1   .   1   1   .   .   .   .   .   .   .   .   .   .   .   .
  _   5  LINUX5    1   .   .   1   .   .   .   .   .   .   .   .   .   .   .   .


'1' Indicates LAN in use.  '.' Indicates LAN not in use.
F3=Exit                              F9=Show all partitions
 F11=Display communication options   F12=Cancel
```

Table 8: iSeries Virtual LAN Configuration

and memcpy_fromio functions are supported. This allows the hypervisor to check all accesses and provide the degree of isolation required on that platform. In addition, this approach allows the hypervisor to manage some of the error recovery when access to certain I/O devices fails. This does result in some performance overhead, however robustness requirements were the top priority on this platform. One again, it is critical that devices owned by one partition not be visible or accessible from another partition.

One consequence of this scheme is that the ioremap call does not work as it does on some other architectures. ioremap maps a PCI address to virtual memory. On the iSeries system the 32 bit PCI addresses are usable on calls such as `readb()`, but are not usable directly. It is interesting to note the defines in `vga.h` which will not work:

```
#define vga_readb(x) (*(x))
#define vga_writeb(x,y) (*(y) = (x))
```

11 PCI Device Support

In the 32 bit iSeries kernel PCI device drivers are supported only after recompiling. This is due to some of the changes above. Table 9 shows some of the definitions in io.h.

Note that in the non-iSeries case the readb() etc. interfaces resolve to inline macros, whereas on the iSeries these are calls to routines.

The iSeries system formally supports a selected set of PCI adapters. In the first release, IBM will formally support a selected set of LAN and SCSI adapters under Linux on iSeries.

Part III

Summary

12 OS400 Changes to support Linux

Surprisingly few changes were made to OS/400 to support running in a partition. Due to the strongly architected Hypervisor system call interfaces that already existed, the operating system structure was ready to support another operating system in a partition.

```
#if defined(CONFIG\_PPC\_ISERIES) && defined(CONFIG\_PCI)
#define readb(addr)      iSeries\_Readb((u32*)(addr))
#define readw(addr)      iSeries\_Readw((u32*)(addr))
#define readl(addr)      iSeries\_Readl((u32*)(addr))
#define writeb(data, addr)   iSeries\_Writeb(data,(u32*)(addr))
#define writew(data, addr)   iSeries\_Writew(data,(u32*)(addr))
#define writel(data, addr)   iSeries\_Writel(data,(u32*)(addr))
#else
#define readb(addr) in\_8((volatile u8 *)(addr))
#define writeb(b,addr) out\_8((volatile u8 *)(addr), (b))
#define readw(addr) in\_le16((volatile u16 *)(addr))
#define readl(addr) in\_le32((volatile u32 *)(addr))
#define writew(b,addr) out\_le16((volatile u16 *)(addr),(b))
#define writel(b,addr) out\_le32((volatile u32 *)(addr),(b))
#endif  /* CONFIG\_PPC\_ISERIES && defined(CONFIG\_PCI) */
```

Table 9: io.h

Obviously some changes were made to the system configuration (both the internal support as well as the user interfaces) to allow for something other than OS/400 to run in a partition.

Additionally, support for virtual I/O was added to OS/400.

Part IV

Status

All the changes to the kernel to support the iSeries are working their way into the kernel. As documentation typically follows function, the following web pages may or may not have anything really interesting on them yet:

`http://www.ibm.com/iSeries/linux`
`http://www.iSeriesLinux.com/`

At the time of writing this paper, the kernel patches including the iSeries changes are at:

`http://oss.software.ibm.com/`
`developerworks/opensource/`
`linux/projects/ppc/`

The code should be working its way through the general community site

`http://penguinppc.org/`

and fsmlabs:

`http://www.fsmlabs.com/linuxppcbk.html`

13 Acknowledgments

The iSeries Linux team at IBM included:
Bill Armstrong
Troy Armstrong
Dave Boutcher
Keith Cooper
Mike Corrigan
Jeff Haumont
Wayne Holm
Bob Holthorf
Brian King
Kyle Lucke
Naresh Nayar
Greg Nordstrom
John Scales
Jeff Scheel
Al Trautman

References

[Borkenhagen] J.M. Borkenhagen, R.J. Eickemeyer, R.N. Kalla, S.R. Kunkel, *A multithreaded PowerPC processor for commercial servers*, IBM Journal of Research and Development, Volume 14, Number 6, November 2000 p. 885-898.

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

[Soltis] Frank Soltis, *Inside the AS/400*, Duke Press, 1997

14 Trademarks