Twin-Linux: Running independent Linux Kernels simultaneously on separate cores of a multicore system

Adhiraj Joshi
LinSysSoft Technologies
adhiraj@linsyssoft.com

Swapnil Pimpale
LinSysSoft Technologies
swapnilp@linsyssoft.com

Mandar Naik
LinSysSoft Technologies
mandarn@linsyssoft.com

Swapnil Rathi
LinSysSoft Technologies
swapnilr@linsyssoft.com

Kiran Pawar
LinSysSoft Technologies
kiranp@linsyssoft.com

Abstract

There are three classes of common consumer and enterprise computing - Server, Interactive and Real-Time. These are characterized respectively by the need to obtain highest throughput, sustained responsiveness, and hard real-time guarantees. These are contradictory requirements hence it's not possible to implement an operating system to achieve all these goals. Most operating systems are designed towards serving only one of these classes and try to do justice to the other two classes to a reasonable extent.

We demonstrate a technique to overcome this limitation when a single hardware box is required to fulfill multiple of these computing classes. We propose to run different copies of kernels simultaneously on different cores of a multi-core system and provide synchronization between the kernels using IPIs (Inter Processor Interrupts) and common memory. Our solution enables users to run multiple operating systems each one the best for its class of computing. For ex., using our idea we can configure a quad core system with 2 cores dedicated for server class computing (database processing), 1 core for UI applications and remaining 1 core for real-time applications.

This idea has been used in the past, primarily on non-x86 processors and custom designed hardware. Our proposal opens the doors of this idea to the off-the shelf hardware resources. We present Twin-Linux, an implementation of this scenario for 2 processing units using Intel-Core-2-Duo system. This idea finds applications in - Filers,Intelligent Switches, Graphics Processing Engines, where different types of functions are performed in a pipelined manner.

1 Introduction

Consider an application that requires huge data computations and responsiveness simultaneously. In this case, an operating system with server kind of computing will only be able to handle data processing part and it lacks in responsiveness. while realtime system will be able to handle responsiveness but throughput will be very low. Hence there is a need to provide different operating system environment within the same hardware box.

We have implemented this concept on a Intel Core 2 Duo architecture. In current scenario (On a Intel Core 2 Duo system), there are two cores (processors) on a single chip, a SMP Kernel and a single copy of RAM. In normal case, a single copy of kernel boots up on core 2 duo machines in SMP (Symmetric Multi Processing) mode. At the boot time one of the cores boots the Linux Kernel and other keeps spinning till the Kernel boots up. The booting core is termed as the BSP(Bootstrap Processor) while the other cores are termed as the APs(Application Processor). After the booting process scheduler assigns the task to both cores in parallel so that there could be simultaneous execution of different threads on respective cores.

By replicating kernel code and by making performance boundaries explicit, Twin-Linux removes the kernel as a bottleneck to scaling up performance in large multiprocessor systems. Each kernel loaded on the cores is composed of a scheduler, a memory manager and code to coordinate communication between other kernels.

The rest of the paper is organized as follows: Section 2 provides background of SMP systems. Details of processor classification, ACPI tables and IPI messages are covered in section 2. Section 3 covers the motivation behind the project and the innovation of Twin-Linux. Section 4 covers the Twin-Linux design. It covers the implementation details and the algorithms used to locate the MADT and IPI communication. It provides an insight into the issues involved in loading the kernel on the respective cores. A concise description of the applications and marketability of the project is included in Section 5. Finally section 6 summarizes the conclusions of the project and the last section lists the References.

2 Features of SMP Systems

2.1 Processor Classification

The Intel Specification classifies Processors into two types: the bootstrap processor (BSP) and the application processors (AP). The BSP is chosen by the hardware or by the BIOS in conjunction with the hardware. The BSP is responsible for initializing the system and for booting the operating system.

The BSP executes the BIOS's boot-strap code to configure the APIC environment, sets up system-wide data structures, and starts and initializes the APs. When the BSP and APs are initialized, the BSP then begins executing the operating-system initialization code. Following a power-up or reset, the APs complete a minimal self-configuration, then wait for a startup signal (a SIPI message) from the BSP processor. Upon receiving a SIPI message, an AP executes the BIOS AP configuration code, which ends with the AP being placed in halt state. APs are activated only after the operating system is up and running. Once the MP operating system is up and running, the BSP functions as an AP.

2.2 ACPI Tables

For a multiprocessor system, the BIOS performs the following functions :

- Pass configuration information to the operating system that identifies all processors and other multiprocessing components of the system.
- Initialize all processors and the rest of the multiprocessing components to a known state.

The BIOS fills the ACPI system description tables and hands over the control to the Boot loader. In a multiprocessor system, multiple local and I/O APIC units operate together as a single entity, communicating with one another over the ICC bus. The APIC units are collectively responsible for delivering interrupts from interrupt sources to interrupt destinations throughout the multiprocessor system.

The local APIC units also provide interprocessor interrupts (IPIs), which allow any processor to interrupt any other processor or set of processors. There are several types of IPIs. Among them, the INIT IPI and the STARTUP IPI are specifically designed for system startup and shutdown. Each local APIC has a Local Unit ID Register and each I/O APIC has an I/O Unit ID Register. The ID serves as a physical name for each APIC unit. It is used by software to specify destination information for I/O interrupts and interprocessor interrupts, and is also used internally for accessing the ICC bus.

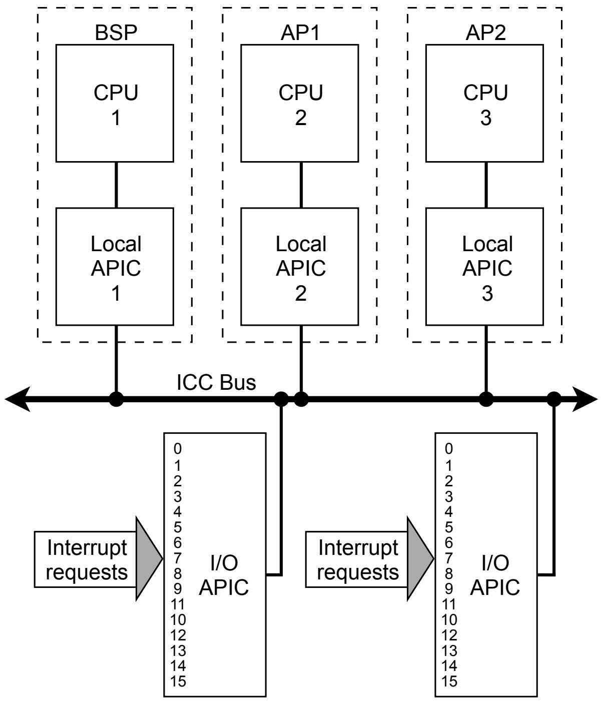


Figure 1: SMP System with APIC Configuration

2.3 Using INIT IPIs

The primary local APIC facility for issuing IPIs is the interrupt command register (ICR). INIT IPI is an Interprocessor Interrupt with trigger mode set to level and delivery mode set to "101" (bits 8 to 10 of the ICR).

The ICR consists of the following fields :

- Vector: The vector number of the Interrupt being sent.
- Delivery Mode: Specifies the type of IPI to be sent. This field is also know as the IPI message type field.

  101 – INIT IPI message to all the local APICs in the system to set their arbitration IDs to the values of their APIC Ids, the level flag must be set to 0 and trigger mode flag to 1.

  110 – Start Up IPI Sends a special "start-up" IPI (called a SIPI) to the target processor or processors. The vector typically points to a start-up routine that is part of the BIOS boot-strap code
- Destination Mode:
- Delivery Status (Read Only)
- Level
- Trigger Mode
- Destination Shorthand
- Destination

An INIT IPI is an IPI that has its delivery mode set to RESET. Upon receiving an INIT IPI, a local APIC causes an INIT at its processor. The processor resets its state, except that caches, floating point unit, and write buffers are not cleared. Then the processor starts executing from a fixed location, which is the reset vector location. To cause the processor to jump to a different location, the INIT IPI must be used as part of a warm-reset. By putting an appropriate pointer in the warm-reset vector, setting the shutdown code to 0Ah, then causing an INIT, the BIOS (or the operating system) can cause the current processor to jump immediately to any location.

2.4 Using Startup-IPIs

STARTUP IPIs are used with systems based on Intel processors with local APIC versions of 1.x or higher. These local APICs recognize the STARTUP IPI, which is an APIC Interprocessor Interrupt with trigger mode set to edge and delivery mode set to "110" (bits 8 through 10 of the ICR).

The STARTUP IPI causes the target processor to start executing in Real Mode from address 000VV000h, where VV is an 8-bit vector that is part of the IPI message. Startup vectors are limited to a 4-kilobyte page boundary in the first megabyte of the address space.

For an operating system to use a STARTUP IPI to wake up an AP, the address of the AP initialization routine (or of a branch to that routine) must be in the form of 000VV000h. Sending a STARTUP IPI with VV as its vector causes the AP to jump immediately to and begin executing the operating system's AP initialization routine.

3 Twin-Linux Applied

Consider a network services application. We describe the performance benefits that can be realized from the utilization of multiple processing cores for the application.

Functional pipelining is a technique that sub-divides application software into multiple sequential stages and assigns these stages to dedicated execution cores. Pipelining can increase locality of reference since each execution core runs a subset of the entire application, potentially increasing the cache hit rate.

In case of a 2-core system, we can have one core dealing with the network I/O requests that will process them partially and the other core will handle all the disk I/Os and remaining processing. Thus the first kernel will handle the networking part where it accepts requests from other systems, processes them partially for the other core and then the other kernel takes care of the storage part and processes those requests completely (as shown in the Figure 2). Distributing network processing to multiple cores can be achieved using multiple network interfaces (NICs) on a system, or affinitizing interrupts of NICs to different cores. This approach can be generalized to apply to other types of network connections between devices such as IPSec flows or IP header compression contexts. Since there is typically little or no locality of reference between different traffic flows like these, reducing the number of different traffic flows processed by each core can improve cache efficiency. Each execution core services a subset of the flows which confines memory accesses to a smaller set of memory addresses and potentially increases cache efficiency.

3.1 Innovation

Multi-core Processors have been adopted in various computer systems from commodity machines to embedded systems. To utilize these multi-core machines more efficiently, a promising way is to run multiple applications on one machine or share resources among different users. In such a situation, instead of using a single operating system, we can run multiple operating systems each one the best for its class of computing. This benefits the applications which demand different operating system environments.

This approach is the first of its kind and is most suitable for computation-intensive and responsive applications. This idea can be easily scaled for quad core and eight core architectures with minimal modification.

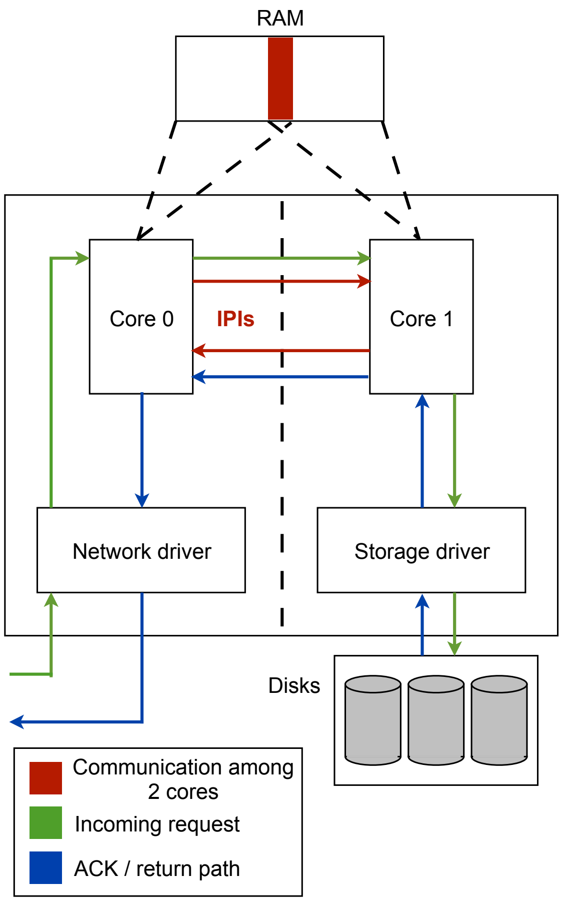


Figure 2: Twin-Linux applied to Filers

4 Design

4.1 Implementation Details

In the current scenario, The BIOS (Basic Input/Output System) selects the Bootstrap Processor (BSP) from a set of processors and the rest are Application Processors. The BIOS, initially, puts the APs in halted state, so that they do not try to execute the same BIOS code as the BSP. The BSP performs all the booting sequence and loads the Operating System after the POST (Power On Self Test). When the control is handed over to the boot loader, GRUB (GRand Unified Boot loader) in our case, only the BSP is active whereas the APs are in a halted condition with interrupts disabled. This means that the AP's local APICs are passively monitoring the APIC bus and will react only to INIT or STARTUP interprocessor interrupts (IPIs).

In our project, the GRUB code is modified in such a way that it will bring up all the APs unlike the normal scenario where only the BSP is up and the APs are in a halted state. Thus we provide SMP support to GRUB making it SMP compatible.

Our project involves the following steps::

4.1.1 Locating and Parsing the MADT (Multiple APIC Description Table)

The MADT is one of the ACPI (Advanced Configuration and Power Interface) tables which furnish information about Multiple Processors.

After locating the Root System Description Pointer (RSDP) structure, we locate the Root System Description Table (RSDT) or the Extended Root System Description Table (XSDT) using the physical system address supplied in the RSDP. The MADT is then located by walking through the RSDT. The MADT contains the

- Header
- Local APIC Address
- List of APIC Structures - This list contains all of the I/O APIC, Local APIC, Interrupt Source Override, Non-Maskable Interrupt Source, Local APIC NMI Source, Local APIC Address Override structures needed to support the platform. The APIC structures are searched for identifying the number of processors and their respective Local APIC IDs.

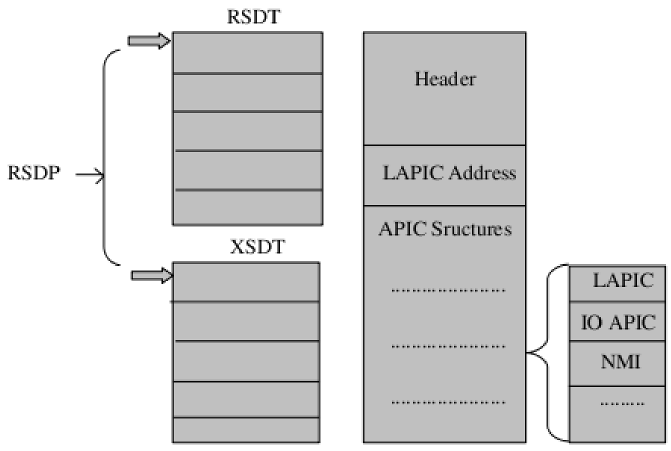
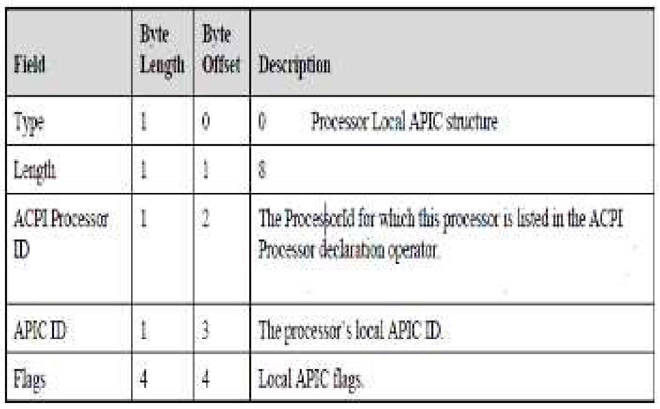


Figure 3: Overview of ACPI Tables

We obtain the Processor ID and the Processor Local APIC ID from the Processor Local APIC Structure which has the following format:

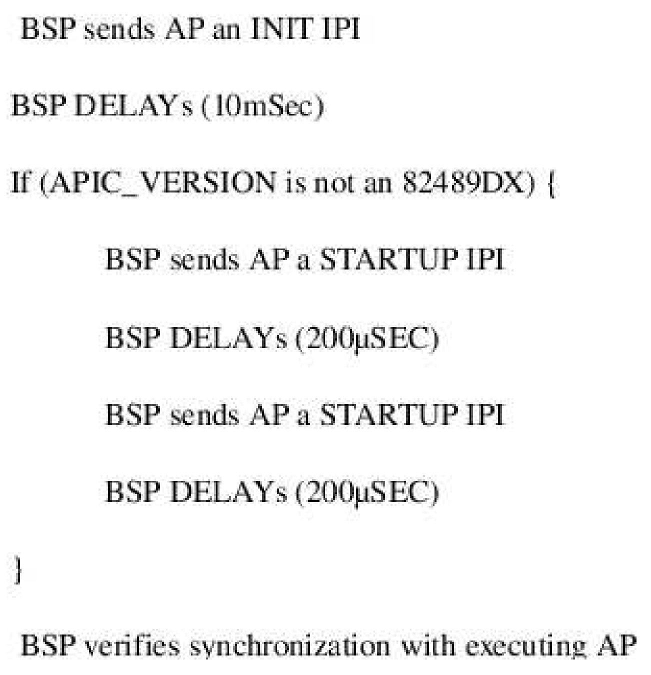


Field	Byte Length	Byte Offset	Description
Type	1	0	0 Processor Local APIC structure
Length	1	1	8
ACPI Processor ID	1	2	The ProcessorId for which this processor is listed in the ACPI Processor declaration operator.
APIC ID	1	3	The processor's local APIC ID.
Flags	4	4	Local APIC flags.

Figure 4: Processor Local APIC structure

4.1.2 Bringing up the APs

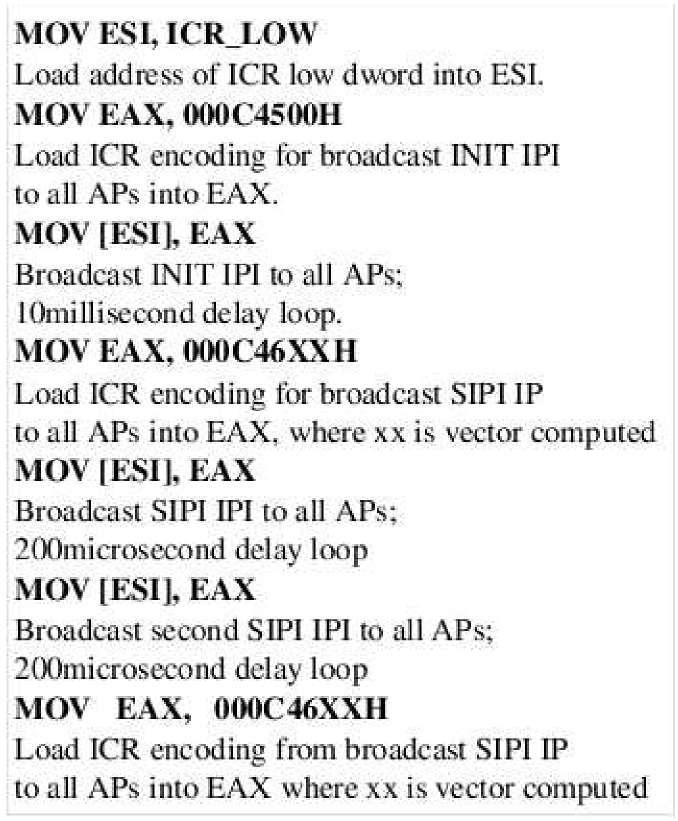
The APs will be activated by sending the IPIs using the local APIC IDs of the APs. We follow the "Universal Algorithm" for awaking the Application Processors mentioned in the Intel Multiprocessor Specification Version 1.4.



```
BSP sends AP an INIT IPI

BSP DELAYs (10mSec)

If (APIC_VERSION is not an 82489DX) {

        BSP sends AP a STARTUP IPI

        BSP DELAYs (200µSEC)

        BSP sends AP a STARTUP IPI

        BSP DELAYs (200µSEC)

}

BSP verifies synchronization with executing AP
```

Figure 5: Universal Algorithm

The following pseudo code Broadcasts an INIT-SIPI-SIPI IPI sequence to wake up and initialize the APs:



```
MOV ESI, ICR_LOW
Load address of ICR low dword into ESI.
MOV EAX, 000C4500H
Load ICR encoding for broadcast INIT IPI
to all APs into EAX.
MOV [ESI], EAX
Broadcast INIT IPI to all APs;
10millisecond delay loop.
MOV EAX, 000C46XXH
Load ICR encoding for broadcast SIPI IP
to all APs into EAX, where xx is vector computed
MOV [ESI], EAX
Broadcast SIPI IPI to all APs;
200microsecond delay loop
MOV [ESI], EAX
Broadcast second SIPI IPI to all APs;
200microsecond delay loop
MOV   EAX,   000C46XXH
Load ICR encoding from broadcast SIPI IP
to all APs into EAX where xx is vector computed
```

Figure 6: INIT-SIPI-SIPI IPI sequence

4.1.3 Loading Kernel on the APs

After bringing up all the APs, independent copy of the kernel will be loaded on each of them. Before loading of the kernel on the individual cores, following issues are taken care of:

1. Division of RAM:

   The RAM memory will be divided symmetrically according to the number of cores available, in our case, two. Thus, For a Intel Core 2 Duo processor running at 2.66 GHz with 2 GB RAM , Each kernel will be given 1 GB RAM memory and some predefined shared memory will be reserved for IPI communication. The similar approach can be followed for multi-core systems with some extensions.

2. Disabling SMP support:

   The SMP support needs to be disabled before loading of the kernels otherwise the kernel would also attempt to wake up the application processors.

3. Sharing the Hardware:

   One kernel will mask all PCI devices and other kernel will mask all the PCI-X devices. Both kernels will get one network adapter each (one PCI and other gets PCI-E cards) so that both are connected to the network.

4. User access to the OS:

   The video RAM will be divided into two halves so that the one kernel displays messages in the upper half and the other one uses the lower half for display.

5. Communication between Application processors:

   Communication between APs will be done through IPIs (Inter-Processor Interrupts) and some predefined common memory.

5 Applications

The project can be deployed in Linux environment for Intel core 2 duo architecture to efficiently utilize both the cores simultaneously.

Examples of systems where this method works are - filers (storage devices), SCSI targets, and graphics processing engines.

Intelligent switches: Intelligent switches do the job of a regular switch and filtering the data being transmitted. Filtering of data could be for identifying security threats or masquerading. This requires a combination of two classes of computing - real time computing for switch functionality (control plane) and server computing for filtering (data plane). Building an efficient switch using a single operating system kernel running on multiple cores could be a challenging task. The resultant switch may either waste computing resources or may not provide adequate responsiveness to connected devices. This situation can be improved with our proposal to run an operating system suitable for control plane on a single core while the rest of the cores run another operating system suitable for heavy server class computing in data plane.

6 Road Ahead

6.1 Enhancements

In future this idea can be implemented for architectures other than x-86 and for different boot loaders. This idea can be easily scaled for quad core and eight core architectures with minimal modification.

6.2 Conclusion

We suggested Twin-Linux an approach of running independent Linux kernels on multiple cores simultaneously. This approach opens the doors of mixed kind of computing to the x86 and open-source community. It enables users to run applications that require different operating system environments. It also provides separation of multiple environments for users. In addition, it reduces contention in operating systems kernels by reducing the synchronization costs.

It extensively takes advantage of the abundance of cores of multi-core systems. This project is a working prototype for systems which demand a combination of data-processing, real-time processing and UI processing. It also provides SMP support to GRUB.

References

[1] Intel's MP specification Version 1.4 [Online] Available `http://www.intel.com/design/archives/processors/pro/docs/242016.htm`

[2] Intel's ACPI specification Revision 4.0 [Online] Available `http://acpi.info/spec40.htm`

[3] Intel's 64 and IA-32 Architectures Software developer manual Volume 3A [Online] Available `http://www.intel.com/products/processor/manuals/`

[4] Intel Architecture Software Developers Manual Volume2: Instruction Set Reference [Online] Available `http://developer.intel.com/design/pentiumii/manuals/243191.htm`

[5] Prof. Godfrey C. Muganda, North Central College, *Intro to GNU Assembly Language on Intel Processors* [Online] Available `scr.csc.noctrl.edu/courses/csc220/asm/gasmanual.pdf`

[6] Danel P. Bovet, Marco Cesati, *Understanding The Linux Kernel*

[7] Robert Love, *Linux Kernel Development*

Proceedings of the Linux Symposium

July 13th–16th, 2010
Ottawa, Ontario
Canada