

LOWER LEVEL ARCHITECTURE OF THE SOMBRERO SINGLE ADDRESS SPACE DISTRIBUTED OPERATING SYSTEM

Donald S. Miller, Donald B. White, Alan C. Skousen, Rossen Tcherepov
Arizona State University - Department of Computer Science and Engineering
Tempe, AZ, USA
{donald.miller, dbwhite, alan.skousen, rossen.tcherepov@asu.edu}

ABSTRACT

Sombrero is an object-oriented single address space operating system whose virtual address space can be distributed over multiple nodes. This paper presents recent additions to the design and implementation of the lower level architecture of Sombrero. This design exploits Sombrero's object-oriented development model to provide an implementation of a lower-level architecture for the Alpha platform. Software representations of the hardware system's major components have been implemented. The existing middle level architectural modules now use the lower-level services. These include board and processor level hardware interfaces as well as interrupt processing, device driver and protocol component architectures. We contrast differences in the ways these things are designed and operate in a single address space environment with the way they are done in a conventional multiple address space operating system. Features covered include the use of passive servers, simple efficient thread switching and threads blocking within interrupts. Passive and active device drivers are described and the rationale for each mode given. The performance of the Sombrero protocol stack was closely examined with attention to its passive service architecture and thread switching method. The current Sombrero Prototype is described.

KEY WORDS

Single address space operating systems, operating systems, distributed operating systems, object-oriented systems, distributed shared memory

1. Introduction and Motivation

Sombrero [1] [2] [3] [4] [5] is a very large single address space operating system (SASOS) that is intended to be extensible to a multiple node system consisting of a set of homogeneous workstations, servers and LANs. In a process-oriented operating system, the process boundary inhibits processes from addressing data or executing code in another processes' virtual address (VA) space. However, in a single address space operating system, neither threads nor programs nor the access rights associated with these programs are bound by a reused VA space. Only a single address space exists and VAs have the same meaning anywhere on a local area network. This leads to important reductions in the complexity of application and system software and consequent reductions

in the costs of software development [6] and increased dependability of OS modules. Sombrero is designed to have the appearance of a single large multithreaded process distributed over many nodes. This paper addresses the design and implementation of the lower level Sombrero architecture.

2. Lower Level Architecture

The fundamental concepts of Sombrero: its single address space and protection mechanisms are described in [2]. The middle level Sombrero architecture consisting of such things as protection domains, memory objects, program class objects and instantiated program objects are described in [3]. Sombrero is a native operating system built directly on the processor in order to better take advantage of single address space features. The lower level architecture consists of a collection of Sombrero services representing the components of the hardware environment. The general idea has been to exploit Sombrero's object-oriented nature to represent hardware components as software objects and implement these abstract objects with object classes and their instantiations.

The elements in this collection are briefly described here. In these descriptions a class refers to a program class object (PCO). The Sombrero compilation process creates a PCO. An instance refers to an instantiated program object (IPO). The boot loader creates the IPO from the PCO and instantiated memory objects (IMOs), the read/write portion of an instantiation of a program obtained from its file during system startup. A service refers to a callable entry point into the IPO. A service performs some processing for the calling thread. The relationships of the classes in the lower level architecture are shown graphically in Figure 1. Driver objects are described in a section 4.

2.1 DEC164SX Mainboard

The Sombrero target system uses an AlphaPC 164SX mainboard. This board is described in [8]. Within Sombrero, an instance the DEC164SX class represents this mainboard. This instance is an abstraction of the inter-chip wiring. This class contains two services. The first maps a device on the PCI bus to a system interrupt number. This reflects the wiring between the PCI slot and the 21174 core logic chip. The second service maps a system interrupt number to system interrupt priority level. This reflects the

interrupt routing established by the AlphaBIOS and the wiring between the processor and the core system logic. The interrupt priority level associated with an interrupt request is called the targeted interrupt priority level.

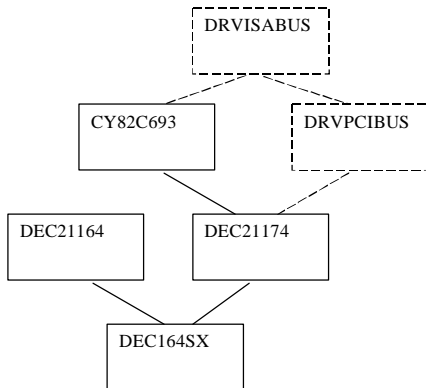


Figure 1: Lower Level Architecture

2.2 21164PC Processor

The Sombrero target system uses a 21164PC processor. This processor is described in [9]. Within Sombrero, an instance of the DEC21164 class represents this processor. This instance provides access to the processor's configuration and status registers. Five services are provided that read and write processor registers including the instruction and decode unit control and status register and the memory address translation unit control and status register.

2.3 System Core Logic

The Sombrero target system uses the DEC 21174 core logic chip. This chip is described in [10]. Within Sombrero, an instance of the DEC21174 class represents the system core logic. The system core logic includes the system interrupt controller, PCI bus controller, the memory controller and the system clock. Services are provided that initialize and start a periodic system timer, enable and disable system interrupt request numbers, return the number of the last industry standard architecture (ISA) bus interrupt request, and to configure the action that the core logic will take upon a fault during access to the PCI configuration address space. The Sombrero middle level interrupt handling module and the PCI bus driver use these services.

2.4 ISA Bus Controller

The Sombrero target system hardware provides an ISA bus through use of a PCI to ISA bus bridge chip, the CY82C693U [11]. This chip includes a pair of cascaded, 8259A compatible, programmable interrupt controllers. Within Sombrero, an instance of the CY82C693 class represents the ISA bus controller. Services are provided that initialize the programmable interrupt controllers, enable and disable their interrupts and issue a specific interrupt acknowledge command to the appropriate programmable interrupt controller

3. Interrupt Processing Architecture

In Sombrero a thread can block in a registered General Protection Domain (GPD) and provide the context for handling the interrupt for which it is waiting. Return from interrupt is a context switch *back to the original user thread*. This allows user threads to block directly in device handlers supporting the concept of passive services. The Sombrero thread scheduler uses this approach to service the system clock interrupt and there is a passive Sombrero protocol service in which user threads block in the network card driver waiting to service both transmit and receive side interrupts.

This architecture establishes a control structure within which interrupt threads execute. It is partly written in C and partly in PALCode (DEC Privileged Architecture Library Code), which runs in a privileged mode that facilitates low-level hardware support functions such as MMU control, interrupt and exception handling and power-up initialization. While device drivers are confined within a GPD, a poorly written driver can still affect negatively overall system behavior. The following sections describe more fully the Sombrero interrupt processing architecture followed by an evaluation of its effectiveness.

3.1 Interrupt as Synchronization Object

Sombrero treats interrupts similar to a synchronization object such as a semaphore. While the analogy is not exact, threads wait on interrupts similar to the manner in which threads wait on semaphores. When the interrupt occurs, the system schedules the first waiting thread similar to what happens as a result of a signal on a semaphore. Interrupt processing must allow for interrupt sharing, raising the interrupt priority level (IPL), and raising the thread priority level. These are features not normally required in semaphore management.

For each triple of system interrupt request number, targeted interrupt priority level and general protection domain, Sombrero creates an instance of an interrupt vector data structure. This data structure provides the data items necessary to manage the threads waiting for the interrupt. A middle level architecture interrupt handling module provides the services to add and delete entries within the interrupt vector matrix and based on these entries computes the priority at which the system runs so that the interrupt the thread is awaiting is enabled.

3.2 Low Level Interrupt Service Support

There are three primary services: Block Interrupt Thread, Schedule Interrupt Thread and Unblock Interrupt Thread. Device drivers use Block Interrupt Thread to request one of four actions described below. Schedule Interrupt Thread is used to select and schedule an interrupt thread. The system thread scheduler uses Unblock Interrupt Thread to schedule a thread that is blocked waiting for interrupt and has an expired timer. When it blocks, an interrupt thread may specify a maximum time that it is to remain blocked.

The Schedule Interrupt Thread service is internal to the PALCode. The processor transfers control to this service when it recognizes an interrupt. This service validates that the current interrupt priority level is lower than the target priority level of the interrupt. It then examines the interrupt vector matrix to find the appropriate entry. If the entry contains a blocked thread, the current interrupt priority level is raised to the targeted interrupt priority of this interrupt and the thread priority is raised to prevent the thread from being preempted while the system executes at this interrupt priority level.

An interrupt thread calls the Block Interrupt Thread service to change the current interrupt priority level, to lower its thread priority, to wait for an interrupt and to request that the system schedule another thread to service an interrupt. Using thread promotion an interrupt thread can request that the current interrupt priority level be raised to the level associated with the interrupt that the thread services. This is used to prevent races that could otherwise occur. Second, an interrupt thread can request that it be suspended to wait for an interrupt. The thread's context is saved and the thread's thread control block is appended to the list of waiting threads for the appropriate entry in the interrupt vector matrix. The processor's current interrupt priority level is lowered if necessary to unmask the interrupt on which the thread is waiting. Third, an interrupt thread can indicate that it did not service the interrupt and that another thread, if waiting, should be given the opportunity to service the interrupt. Finally, an interrupt thread can indicate that it has completed servicing the interrupt. At this point, both the thread priority level and the current interrupt priority level can be lowered. The thread continues to execute but can now perform "bottom-half" processing without masking interrupts.

3.3 Interrupt Thread Models

The Sombrero interrupt processing architecture supports two models for device driver organization, an active service model and a passive service model. In the active service model, the device driver during system initialization creates a dedicated thread(s) to service device interrupts. User threads interact with the interrupt thread through input and output queues. Semaphores may be used to block user threads while waiting for the interrupt thread to complete device interactions. The serial port and real-time clock drivers, described in following sections, use this model. In the passive service model, the calling user thread becomes the interrupt thread as it enters the device driver system module. If no user threads are reading or writing the device, then device interrupts may be disabled or ignored. The Sombrero protocol stack and the network card driver, described in a following section, use this model. The selection of the appropriate model depends on device characteristics and the processing environment. Device characteristics found to be important were status reporting and buffering. The processing environment conditions found to be important were the presence or absence of an activating thread and whether

the service was to be blocking or non-blocking. A detailed analysis of the suitability of each model for particular devices and drivers can be found in [4].

3.4 Evaluation

The Sombrero interrupt processing architecture is feasible. Several device drivers were implemented using the architecture. While the Sombrero interrupt architecture differs from more conventional operating systems such as Linux, the architecture itself presents no obstacles to driver implementation. Implementation issues come up with respect to interrupt latency, thread to IRQ Mapping, shared interrupts and multiplexed interrupts. The details are discussed in [4].

3.5 Interrupt Threads

Servicing interrupts within a thread context is not unique to Sombrero. The use of kernel threads to handle interrupts in the Solaris 2 kernel is described in [12]. The Solaris 2 kernel preallocates partly initialized kernel threads, called interrupt threads. The kernel then converts an interrupt into an executing instance of one of these interrupt threads allowing a single synchronization model to be used throughout the kernel. Providing the interrupt handler with a thread context also permits the full use of kernel functions, even those that can potentially block the caller. This removes a traditional constraint on the logic of interrupt handlers. Sombrero's interrupt threads provide the same functionality while removing the Solaris 2 constraint limiting interrupt threads to kernel threads. This functionality is possible because no context switch is needed for threads to go from user to kernel address space. Most importantly since in Sombrero any thread can be an interrupt thread, the interrupt thread can possess the complete application context. Further comparison can be found in [4].

4. Device Driver Architecture

The Sombrero device driver architecture builds on the lower level architecture services and the middle level interrupt processing architecture to support the development of drivers for devices attached to a Sombrero target system's PCI and ISA buses. The relationships of the classes in the device driver architecture are shown graphically in Figure 2. This architecture includes infrastructure objects that represent the PCI bus and ISA bus and device objects representing the network interface card, the real time clock and a serial port.

4.1 PCI Bus

Within Sombrero, an instance of the DRVPCIBUS class represents the PCI bus. This object is the initial owner of the virtual addresses corresponding to the PCI configuration and input-output address spaces. It is a passive model driver. During system initialization, this object enumerates the devices on the PCI bus and validates input-output address ranges assigned to the devices by the

AlphaBIOS. It has services that allows a device driver to locate a device if present on the PCI bus and transfer ownership of the device's PCI address ranges and interrupt number to the caller's GPD and that provide driver read and write access to PCI configuration address space and PCI input-output address space.

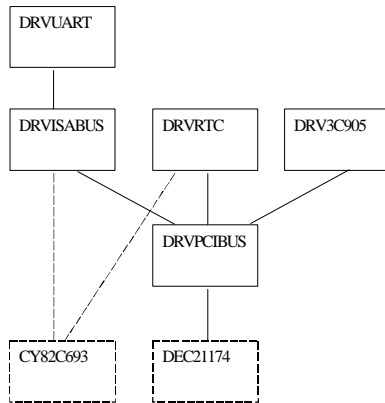


Figure2: Device Driver Architecture

4.2 Industry Standard Architecture Bus

An instance of the DRVISABUS class represents the industry standard architecture (ISA) bus. This class provides services to support development of drivers for devices on the ISA bus. This driver uses the services of the instances of the CY82C693 and DEC21174 classes described previously to provide a high-level ISA bus abstraction. This driver is a passive service that demultiplexes the ISA interrupts multiplexed over system interrupt seven. It provides services to register and unregister a calling GPD as the owner of the specified ISA bus IRQ number and to enable a calling GPD to wait for an interrupt.

4.3 Network Card Driver

Within Sombrero, an instance of the class DRV3C905 provides a passive model driver for the 3Com 3C905 network interface card (NIC). The driver uses the PCI bus driver services. During construction of the instance, the driver creates two rings of buffers: one for transmit and one for receive. These rings are organized as download (from system memory to the network interface card memory) and upload (from network interface card memory to system memory) lists. The 3C905 uses these lists to transfer data to and from the 3C905's internal transmit and receive buffers. The driver provides a send service that waits for a free ring buffer if necessary and copies the contents of a data buffer chain into it and a receive service that waits for a full ring buffer if necessary and copies the contents of the ring buffer into a data buffer chain. The driver also includes a service that returns the Ethernet address of the network card associated with this instance of the driver.

4.4 Real Time Clock

An instance of the DRVRTC class represents the real-time or time-of-day clock. The clock hardware resides on the CY82C693U chip. This is an example of an active service. During system initialization, the driver creates an interrupt thread that configures the clock to generate an end-of-update interrupt occurring at one-second intervals. The thread then loops forever, blocking and waiting for the clock interrupt, then reading from the clock chip into local storage and then blocking again. This driver could have been easily implemented as a passive service. A consequence would be that some calls to the get_time service would be delayed. The time is not available from the chip during timer rollover. During these periods, the calling thread would need to wait for the time to become available. The implementation as an active service allows the caller of the get_time service to obtain the last reported time of day without delay. The driver provides a service that copies the current time from local storage to the caller's storage.

4.5 Serial Port

An instance of the DRVUART class represents the serial communication ports whose hardware resides in a chip on the motherboard. The driver is implemented following the active service model. During system initialization, the driver creates two interrupt threads, one for each serial port. A serial port driver thread loops servicing UART interrupts. There are circular transmit and receive buffers. Calling threads may be blocked on semaphores. The interrupt threads signal the semaphore when space or data is available. Raw and cooked modes of operation are provided.

The decision to implement active services resulted from considerations during data output. Serial devices are slow. Writing to a serial port should not routinely block the user thread. The serial port has a limited output buffer (16 byte FIFO) and requires the processor to refill the FIFO to sustain output. The solution adopted was to allocate a relatively large circular output buffer. The write service normally just copies the user buffer into the output buffer, starts the data transfer and returns. The user thread is blocked only when there is insufficient space in the output buffer. In that case, the user thread waits on a semaphore for output buffer space to be available. The interrupt thread signals the semaphore when space is available. The receive side for read operations operates similarly. There are three services: an ioctl service to configure the driver and serial port and read and write services to transfer characters from and to the UART.

5. Protocol Component Architecture

The original Sombrero protocol stack was presented in [3]. This paper describes the refined protocol stack shown in Figure 3 along with its evaluation described in [4]. The stack has five in-line layer components and two supporting components, each of which is represented by a Sombrero class. The general functionality of these layers is similar to

the protocol stack layers of many modern operating systems. We concentrate on the issues that come up as a result of single address space operation and in particular those related to passive servers. The DRV3C905 class representing the NIC device driver at the physical layer is described in section 4.

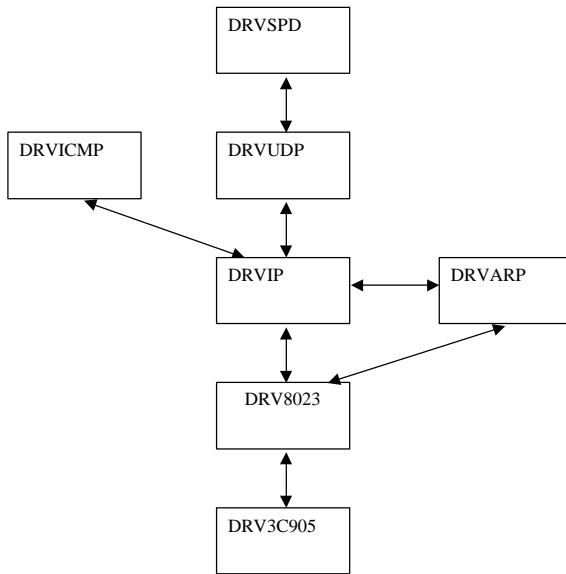


Figure 3: Sombrero Protocol Stack

5.1 Link Layer Protocol

An instance of the class DRV8023 provides the link layer protocol. It is a passive model driver. The link protocol is a simple implementation of the Ethernet packet framing protocol following RFC 894. A sending service adds framing and calls the DRV3C905Send service. A receive service examines the frame from the DRV3C905Receive service and, based on whether it's the one it is waiting for or not, either returns to its higher-level caller or switches to another waiting thread. A MAC address service returns the local Ethernet address to the caller.

5.2 Address Resolution Protocol

An instance of the DRVARP class provides support for the Address Resolution Protocol (ARP). It is an active model driver. During protocol stack initialization, an ARP daemon thread is created. This thread loops forever calling the link layer receive service to receive ARP protocol packets. If a received ARP packet requests the Ethernet address of the Sombrero system as identified by its IP address, the ARP thread sends an ARP reply packet containing the system's Ethernet and IP addresses. There is also a MACLookup service that accepts an IP address from a local caller and returns the associated Ethernet address.

5.3 Internet Protocol

An instance of the DRVIP class provides a minimal set of internet protocol (IP) services. An IPIn service adds a packet header and calls the link layer receive service for a packet and an IPOut service constructs an IP header and

calls the link layer send service to send the packet to its destination and returns to the caller.

5.4 Internet Control Message Protocol

An instance of the DRVICMP class provides internet control message protocol (ICMP) support. It is an active model driver. During protocol stack initialization, an ICMP thread is created. This thread loops calling the IPIn to receive ICMP packets. If the packet is an ICMP echo request, the thread allocates a buffer and buffer descriptor. It formats the buffer as an ICMP echo response packet and calls the IPOut service. Otherwise, the thread discards the input packet.

5.5 User Datagram Protocol

An instance of the DRVUDP class provides basic user datagram protocol (UDP) support. It is a passive model driver. A UDPOut service adds a datagram header and calls the IPOut lower layer service to send the datagram to its destination and returns to the caller. A UDPIn service removes the datagram header and compares the caller's destination UDP port number to that in the received datagram header. If they match, the service copies the source IP address and the source UDP port number to the caller's storage and then returns to the caller. If they do not match, the service switches to another waiting thread.

5.6 Sombrero Protocol Driver

An instance of the class DRVSPD passive model driver provides Sombrero Protocol Driver (SPD) services. This is a Sombrero specific protocol. It is used between the Sombrero host and target systems. In a distributed Sombrero configuration, it is used between Sombrero target systems. The Sombrero Protocol Driver provides services to send messages to a host, wait for messages from a host with and without a timeout, send data to a host and receive data from a host. Key features of the message protocol are a transparent transfer of messages up to 1024 octets, a stop-and-wait protocol and an acknowledgement timer to trigger retransmission. A burst protocol supports the data transmissions. This provides 8 megabyte bursts of up to 8192 fragments containing up to 1024 octets for the transfer of large VA regions. A retransmit-N protocol and an acknowledgement bitmap following a burst are used.

5.7 Sombrero Protocol Stack Evaluation

The Sombrero prototypes have provided the opportunity to examine in operation the correctness and performance of features made available to the Sombrero protocol component architecture by a single address space – particularly the passive server model, the opportunity for reduced context switches and reduced data movement. As could be expected in an early prototype of an unconventional research operating system, in addition to the mostly correct and reasonable performance features, some unexpected logical design and implementation problems were observed. The most serious of these is due to passive server thread operations in the receive

operations of the protocol stack. It is possible for a receiving thread to not be able to get control at an appropriate layer before a desired message is discarded. This can lead to equilibrium conditions during which message transfers fail to progress. A purely active model protocol stack does not have this problem since the situation of multiple threads wanting to receive a message does not occur. To fix this it will be necessary to provide waiting threads with more information than is required in active model protocol stacks. Another observed problem is that the receive side of the protocol stack showed degraded performance at the IP and Link layers, again due to the existence of multiple threads in the passive thread model. There are two threads that can be present at these layers because of the existence of the ARP and ICMP protocols. Only one can hold the layer lock and the other must wait. This leads to increased thread switching and data movement. It also became obvious that a more efficient data movement policy is possible in the single address space than that currently used in Sombbrero where each thread has its own associated buffer chain in addition to a given layer's common buffer. Finally the existence of multiple threads each of which determines its own buffer size can cause buffer overflow and data leakage between threads. Design and implementation solutions to these problems are in progress.

6. Related Work

Most contemporary research SASOSs have been designed to run on stock RISC 64-bit processors. These include Opal [13] and Mungi [14]. This means additional software communication protocols are required for protection. These operating systems also run on top of multiple address space operating systems, Mach and L4 respectively, further precluding them from taking full advantage of single address space properties. Typically capabilities are used, requiring operations in one or more additional namespaces. SASOSs with protection and memory management hardware targeted at single address space operation include iSeries [15], and Sombbrero. Special Hardware support typically enables or significantly facilitates the protection mechanism and the granularity of access. In Sombbrero it also enables implicit domain switching. The iSeries does not make the full set of single address space properties available to applications and does not distribute the virtual address space over multiple nodes. More information on the Sombbrero project can be found at [16].

References

- [1] Alan Skousen and Donald Miller, "Operating System Structure and Processor Architecture for a Large Distributed Single Address Space", *IASTED 10th International Parallel and Distributed Computing and Systems Conference Proceedings*, October 1998, pages 631-634.
- [2] Alan Skousen and Donald Miller, "Using a Single Address Space Operating System for Distributed Computing and High Performance", *18th IEEE International Performance, Computing and Communications Conference*, February 1999.
- [3] Alan C. Skousen, *SOMBRERO: Implementation of a Single Address Space Paradigm for Distributed Computing Exhibiting Reduced Complexity*, Ph.D. Dissertation, Computer Science and Engineering Department, Arizona State University, August 2002.
- [4] Donald B. White, *Implementation of a Lower Level Architecture for the Sombbrero Single Address Space Operating System*, MS Thesis, Computer Science and Engineering Department, Arizona State University, December 2005.
- [5] Donald S. Miller, Alan C. Skousen and Milind Patil, "Distributed Scheduling for the Sombbrero Single Address Space Distributed Operating System", *PDPTA'06 - The 2005 International Conference on Parallel and Distributed Processing Techniques and Applications*, June 2006.
- [6] Ron Feigen, Alan Skousen and Donald Miller, "Reduction of Software Development Costs under the Sombbrero Distributed Single Address Space Operating System", *The 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2002)*, June 2002.
- [7] Alan C. Skousen, *Sombbrero: A Very Large Single Address Space Distributed Operating System*, MS Thesis, Arizona State University, 1994.
- [8] Compaq Computer Corporation. Alpha PC 164SX Motherboard. Technical Reference Manual, EC-R57EB-TE, 1998.
- [9] Digital Equipment Corporation. Digital Semiconductor Alpha 21164PC Microprocessor. Hardware Reference Manual, EC-R2W0A-TE, September 1997.
- [10] Digital Equipment Corporation. Digital Semiconductor 21174 Core Logic Chip. Technical Reference Manual (preliminary), EC-R12GC-TE, 1997.
- [11] Cypress Semiconductor Corporation. hyperCache / Stand-Alone PCI Peripheral Controller with USB. Technical Manual (preliminary), CY82C693U, 1997.
- [12] S. Kleiman and J. Eykholt. Interrupts as threads. *Operating Systems Review* 29(2): 21-26, Apr. 1995.
- [13] Chase, J. S., Levy, H. M., Feeley, M. J., and Lazowska, E. D., "Sharing and Protection in a Single-Address-Space Operating System", *ACM Transactions on Computer Systems*, Vol. 12, No. 4, November 1994, pp. 271-307.
- [14] Heiser, G., Elphinstone, K., Vochtelloo, J., Russell, S., and Liedtke, J., "The Mungi Single-Address Space Operating System", *Software-Practice and Experience*, VOL. 28(9), July 1998, pp. 901-928.
- [15] Soltis, F.G. "Fortress Rochester, the Inside Story of the IBM iSeries", *NEWS/400 Books division of Penton Technology Media*, 2001.
- [16] Sombbrero Web Site, <http://www.eas.asu.edu/~sasos/>