Using a Single Address Space Operating System for Distributed Computing and High Performance

Alan Skousen Donald Miller
Computer Science and Engineering Department
Arizona State University, Tempe, Arizona 85287, USA
E-mail: {alan.skousen,donald.miller}@asu.edu

Abstract

Recent 64-bit microprocessors have made a huge 18.4 quintillion byte address space potentially available to programs. This has led to the design of Operating Systems that provide a single virtual address space in which all code and data reside in and that spans all levels of storage and all nodes of a distributed system. These operating systems, called SASOSs, have characteristics that can be used to support synchronization and coherency in a distributed system in ways that provide an improved program development environment and higher performance than that available from conventional operating systems. Sombrero, our SASOS design, makes use of its hardware support for object-grained protection, separate thread related protection domains and implicit protection domain crossing to provide synchronization and coherency support for distributed object copy set management not available in SASOSs built on stock processors. Its design, which provides direct system level support for object oriented programming includes a number of system architectural features targeted specifically at modern distributed computing environments.

1. Introduction

When provided with changes to processor protection and memory management architecture to support its abstractions, a SASOS can provide a better substrate for programming and performance in a distributed system by assigning unique CPU enforced policy to each and every address in the single distributed address space.

Sombrero provides support for distributed object copy set management not available in conventional process oriented operating systems or in SASOSs built on stock processors. Its design, which in effect, provides direct system level support for object oriented programming, includes provision for user-extendible consistency semantics. It contains a mechanism for migration/replication of kernel control structures and support for transparent and implicit access to remote data and services. Sombrero provides additional support for the use of passive servers in client/server computing systems. Sombrero's large distributed single address space has access to neighboring computers in a way that is optimized to support a scalable cluster of homogenous machines targeted at close knit operations too large for a single node. Examples are combined CAD/CAM and Process Management systems, DDBMSs and a variety of other client/server systems. The Sombrero prototype is currently being implemented [1].

2. Single address spaces and SASOSs

The recent advent of 64-bit RISC processors has made much greater virtual address spaces available to applications and the operating system. This has led to a reexamination of the premises of the conventional process-oriented operating system model. In this model, which has evolved from a main memory oriented view of address spaces, virtual addresses are reused and every process has its own copy of a virtual address space supported by page-oriented protection and there is a separate file system namespace for persistent data. The fundamental question is whether a different model might supply an improved program development environment and increased performance. As a 64-bit virtual address space is large enough to provide addresses for all nodes of a modest network for "a long time" [2], the design has built-in extensions specifically targeted at programming support and performance in a distributed system.

2.1. Characteristics of a single address space

In a single address space there is one persistent network-wide set of virtual addresses that can be used to uniquely name every data object in a computer system. Its characteristics are: 1) Virtual addresses can be permanently and uniquely bound to all data and code

objects. This means virtual addresses can serve as unique names and the virtual address space can serve as the only namespace. 2) The virtual address namespace spans all levels of the storage hierarchy on every node. This means all physical storage can be viewed as a hierarchy of caches for the contents of persistent virtual addresses. This contrasts with a view of sets of physical containers, e.g., RAM, disk, whose elements are used to assign short-term references for the duration of a process. 3) The single large namespace is manipulated directly by the CPU and access to it is controlled directly by memory and protection management hardware. This means the CPU can directly enforce access policy on all objects defined in the system as it manipulates virtual addresses. This is especially helpful for the frequent case when policy is cached in the CPU. For a more complete treatment of SASOS advantages and the paradigm shift that occurs in going from multiple namespaces to a single network-wide namespace, see [3].

2.2. Process-oriented operating systems

In process-oriented operating systems such as UNIX, NT, MVS and Mach, each process runs in it's own VA space making VAs context dependent. Protection is provided by the underlying memory management hardware. Process boundaries encapsulate and bind temporary VA spaces, execution state, objects, protection domains and principal activities. Persistent storage is in a separate namespace. This leads to frequent use of message passing for control transfer, data exchange and synchronization and forces servers to be active in the sense that they must be executed by their own threads. IPC using shared memory requires prior setup and mapping. Data migration is difficult, necessitating creation of a new namespace for distributed data (DSM). Distinct node namespaces for code make task migration difficult.

2.3. Motivation for building a SASOS

Processors with a 64-bit architecture are now widely available although current models provide pin-level support for less than 64 bits of VA space (e.g., DEC Alpha 43 bits, MIPS R10000 44 bits). Another development has been the increasing size and complexity of modern software, leading to increased software engineering efforts to reduce the costs of program development. Simpler mechanisms that support sharing and communication between large programs and databases are needed. These include common naming mechanisms and system level support for Object Oriented Design and Programming. It is generally agreed that, for a variety of reasons, within about 10 years performance increases derived from reducing chip feature sizes will begin to decline [4]. Increased performance will have to come from architectural changes such as increased parallelism [5] and shifting to single address space operation. The major reason for building a SASOS is that the very large common namespace has the potential to simplify programming and increase performance in single node and homogenous distributed system clusters. The tradeoffs are the costs associated with shifting to new hardware (it has been pointed out that only a two-fold or greater improvement in performance makes a switch to incompatible hardware worthwhile [6]) and to a new design paradigm. There have been a number of SASOS designs with and without additional protection hardware support. Those without additional hardware include Opal [2] [7], Mungi [8], Arius [9] and Grasshopper [10]. Those with modified protection hardware include AS/400 [11], Monads [12] and Sombrero [13]. See [14] [15] for comparisons between the features of hardware-supported SASOSs and those built on stock RISC processors and between Sombrero and other hardware-supported SASOSs.

3. Sombrero design overview

3.1. Basic Abstractions

Table 1 lists the basic Sombrero abstractions.

Memory Object º Characterized by *Range* - a Distinct Variable Sized Contiguous Allocation of the Virtual Address Space - contains *Memory Regions*, sets of Virtual Addresses within a Range with Non-Overlapping Protection Attributes.
Protection Domain º Set of Memory Objects and Protection Domains Reachable by a Thread executing within it and their Associated Access Permissions. There are General and Carrier Protection Domains (GPDs and CPDs).
GPD: Code, Data and other GPDs accessible to all threads within it
CPD: Data and GPDs privately accessible to an Individual Thread
Thread º State of a Computation Represented by Current or most Relevant CPU Register State (PC, SP, General Registers, etc.), Reachable Carrier Protection Domain Memory, and Status Information.
Principal º Unique Protected Identity that Represents a User or a System Service - binds Resources and Activities to an Owning User.

Table 1. Sombrero Basic Abstractions

3.2. Hardware design functional requirements

To evaluate whether every memory access can be allowed, protection hardware must be CPU-resident. Sombrero includes a design of this hardware called a Range Protection Look-aside Buffer (RPLB). Ideally, CPU hardware to support protection in a single address space should allow every CPU instruction that accesses memory from an authorized protection domain to be executed while preventing unauthorized access. In addition, no special instructions should be required for a thread to traverse protection domains or to extend access to previously unaccessed memory objects; rather access should be implicit reflecting higher-level policy decisions. To allow this implicit protected access it is necessary to cache memory access and domain switching

authorizations within the CPU in a way that the protection checks can be integrated into the instruction pipeline. The cache should contain VA-related protection information and have a pair of registers containing virtual addresses identifying the current general protection domain and a carrier protection domain associated with the currently executing thread. The cache needs to store triplets of the form:

<protection domain tag, resource accessed, access rights>

where the protection domain tag and resource accessed, which can be either a memory region or another protection domain, are represented by virtual addresses and the access rights are {*rwxs*} where *s* means a domain switch is allowed. The hardware functionality provided by the Sombrero RPLB is summarized in Table 2. This functionality is a considerable extension of that provided by the PLB presented in [16].

1. Separation of address translation and protection functions
2. Hardware caching of allowed protection domain crossings
3. Protection domains for threads distinct from the general protection context
4. Implicit domain crossing using ordinary instructions
5. Protection definition for variable granularity object sizes

Table 2. Hardware Functionality implemented by the RPLB.

The RPLB design has been synthesized and its performance evaluated using Mentor, Synopsys and Cascade VHDL tools. Performance has been shown to be feasible relative to 500MHz Alpha 21164 timings. Details of this synthesis can be found in [17]. The complete logical design of the Sombrero RPLB is given in [18].

3.3. Software protection data structures design

Kernel-resident data structures, cache the data needed to convert between user-level policy information in the form of resources, access rights and users, e.g., files, {rwx} privileges, user ids and resource ownership, and the CPU-level representation of these things in terms of protection domains, virtual addresses and access privileges. These data structures called Protection and Resource Access Lists (PRALs) logically contain two types of simple lists interlinked to provide fast resource access and efficient rights revocation. The PRALs co-locate the same fundamental protection information held in classical capability lists and ACLs. A more detailed description of the PRAL design is contained in [18]. A software simulation of the PRAL and a comparison of Sombrero RPLB + PRAL vs. Alpha NT TLB + Page Tables performance is presented in [19]. For larger memory object sizes, RPLB performance is shown to be significantly better than TLB performance.

4. Sombrero System Architecture

As our understanding of operating system structure for a single address space increased, it became clear that a SASOS should have a completely flat but modular system structure. Taking advantage of the characteristics of a single address space allows us to design the Sombrero operating system as a set of peer level services. Figure 1, presents a simplified sketch of our early view to Sombrero's structure (the figure does not clearly indicate the orthogonality of privilege level and protection domains and the fact that modules other than the Sombrero Services can, if authorized, have access to privileged instructions). Hardware caching of protection domain object regions, hardware caching of allowable domain crossings and hardware-supported private thread protection domains are crucial to this structure. They allow Sombrero service requests by applications and upcalls by executive services to be implemented by ordinary procedure calls.

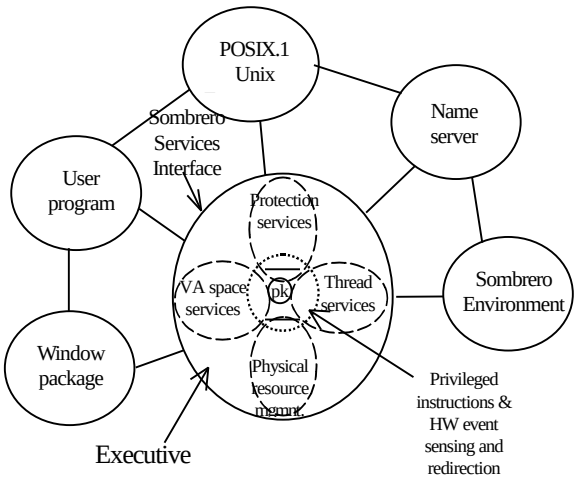


Figure 1. Sombrero structure including executive, sombrero Services Interface, operating system environments, user programs, kernel mode protected operations and pico-kernel.

Communication between the executive and user modules is the same as that between user programs. System calls implemented with trap instructions and the context switches associated with kernel intervention as occur in process-oriented operating systems are not required except to access privileged instructions. Privilege level (the one controlled by the kernel/user bit) is orthogonal to protection domain definition, although typically only executive modules may need to execute in kernel mode. Executive services run at user privilege level except when kernel mode is necessary to execute privileged instructions or access sensitive memory areas. The architecture allows all but the basic services shown in Figure 1 to be provided entirely by user servers. User services include authorization, user identification and

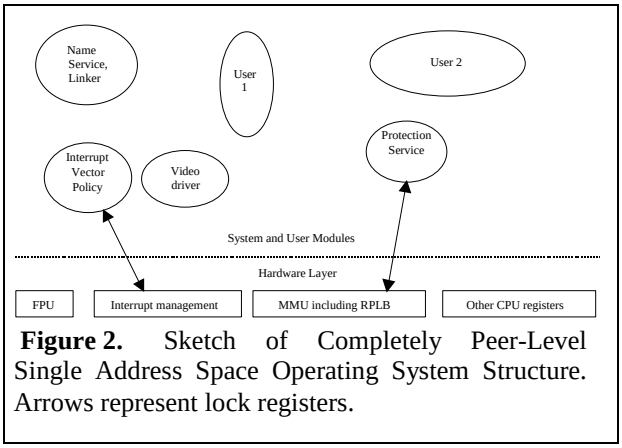


Figure 2. Sketch of Completely Peer-Level Single Address Space Operating System Structure. Arrows represent lock registers.

ownership establishment, and resource naming. In this architecture, the executive can be viewed as a set of peer-level persistent objects. The very small kernel handles redirections for exceptions and traps and is also entered to allow access to privileged instructions for such purposes as context switching, TLB and RPLB entry insertion and deletion and node power up/down. See [14] for a more extended treatment. The Sombrero SSI is described in [20].

Our current design (see Figure 2) has no central kernel and distributes the services it typically performs among the executive services. Lock registers are used via privileged instructions to limit access of CPU resources to designated protection domains that provide the associated executive services. This removes the orthogonal protection currently supplied by kernel/user bits replacing it with the more general mechanism that applies to all protection domains. In Figure 2, a sketch of part of such a structure, each module exists at the same CPU privilege level but is guarded by access policy, i.e., who can access what, as determined by protection information cached in the PRALs and RPLB. For compatibility with available hardware our current implementation makes use of kernel mode to limit access to CPU resources to designated executive protection domains.

The Sombrero Services executive modules are instantiations of object classes that are implemented as general protection domains (GPDs) as are all Sombrero programs. These services' methods can be extended and overridden by other services as established by a path in the name-service for the object instantiation linker. The name-service provides symbolic representation for memory objects, protection domains and threads as well as a naming hierarchy for establishing conventional directories and linker information. The linker information appears as decorated entry point names for executables and pointers to data for other memory objects. Conventional APIs employing software traps are not used within Sombrero as these impose a non-extensible hierarchy to system structure.

5. Distributed System Design Overview

In Sombrero the global VA space is partitioned among 'owning' nodes with subsequent refinements as nodes enter and leave a cluster via a fissioning and merging technique [21]. Each node allocates memory from its own exclusive portion of the VA space. This means that there is a 'home' node for every virtual address range. Network routers use this unique address range to route network packets based on the destination memory address of the message instead of using IP numbers (another namespace). When memory objects, protection domains and threads are first created, the control blocks used to manage them are given virtual addresses (names) from the home node partition. If the memory object, protection domain or thread is migrated or replicated to another node, a surrogate control block is created at that node with a virtual address from the new node's VA partition. This surrogate contains the virtual address name of the original control block. Every copy of a control block has three token bits that indicate whether read, write and execute privileges are enabled for the memory object, protection domain or thread on its node. These can be used, for example, to indicate whether the read or write privileges for a given node's copy of the memory object are in effect, whether a protection domain on a node can be entered or whether a thread or its clone is active on the node. In order to provide consistency management to support the caching and replication present in a clustered computing system, every control block also has a pointer to an associated 'token tracking' data structure. This structure contains pointers to token tracking structures on other nodes for the same or related objects that are used for copy-set and address space management. Finally, every node has one or more user authorization agents and methods associated with memory object control blocks (MOCBs) that implement distributed system consistency policy for memory objects. Exactly when a token is set to enable or disable a privilege is a policy decision controlled by the methods specified by the owner of the memory object. Sombrero MOCB default methods enforce sequential consistency. Other consistency model policies, such as causal or release, can be implemented by user extensions overriding the MOCB base class methods. The initial privilege to access an object, control a thread or enter a protection domain is obtained from a user authorizing agent by the corresponding Sombrero executive service. The migration and replication of kernel objects using surrogates make the network's multiple computers appear as a single virtual NUMA multiprocessor. Additional information on Sombrero distributed system design can be found in [13] [21] [22]. Sombrero support for distributed parallel computation and distributed system fault tolerance can be found in [15]. A more complete treatment of the

protection, resource access and control transfer issues that must be dealt with in the design of a communication system can be found in [14].

5.1. Kernel Replication via Surrogates

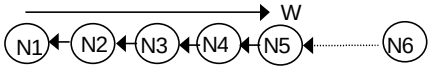
Sombrero kernel control structures can be replicated across the network. Unlike page distribution that copies pages to the copy-set, kernel replication builds local representations of permissions found globally in the copy-set as needed. Local representations of control structures, which originated elsewhere, are called *surrogates* and have the same *name* field entry as the original control structure (the VA on the originating node) but have a different VA for the local control block. This has several advantages in the Sombrero environment. The edges of Token Tracking graphs have nodes as vertices and edges that connect the nodes. Each edge is actually a pointer to a replicated surrogates' VA node. This allows network routing to forward graph messages not only to the proper node based on the local surrogate VA, but also to the exact control structure. The local hardware also uses the local surrogate VA in the RPLB to perform object loads. These two optimizations reduce the need to search the kernel structures (their VAs are always the local structure name) and allow a compact representation for the various control blocks by avoiding fragmentation of locally allocated physical space. The drawback is that the relatively few instances of resource access by the global name field VA name on a foreign node will have to search the protection access lists there to find a match rather than using the VA name as a pointer. Since in many cases, i.e., on the originating node, the local and global VA names will be the same this tradeoff seems worthwhile.

5.2. Distributed object copy-set management

Object grained protection hardware directly supports object regions rather than pages as the base unit of network distribution. Per-object copy-set management replaces the per-page copy-set management found in process-oriented systems and SASOSs built on stock hardware because TLB per-page access rights are no longer required to determine protection. Network data transmission granularity is therefore determined by access policy rather than by operating system virtual address translation mechanisms. Even more important, because the hardware enforced protection triples (PD, VA, access rights) are invariant and available throughout the network-wide virtual address space, every virtual address, and every memory object containing it, is tied to a specific access policy. Once established, access policy, i.e., who can access what and when, is implicit and transparently distributed across the network by the protection hardware resident on every node. Local and remote protection faults support object access policy in much the same way as local and remote page faults support memory sharing in DSM systems. And as in DSM, the demand-paged virtual memory management already present on every node automatically generates the object region address faults needed. Unlike DSM, Object-region granularity enables simple scaleable, per-object data structures and pointers to be used. Token tracking structures contain the list pointers of simple network-wide graphs used to locate the most recent copy of an object's modified page, and control block tokens are used to enable access. Only the data structures in use at a node need be present at it.

Two directed acyclic graphs that Sombrero uses to manage the consistency of cached and replicated objects, the *Last Known Writer Graph* and the *Modified Page Cache Graph* are shown in Figures 3, 4 and 5. They are used to locate the most recent copy of a particular page of an object for a reader or a writer. In overview they are generalized graphs for supporting various types of consistency management.

They are simple and self-pruning (see Figures 4 and 5). When a node forwards a token to another node it stores a pointer to the other node in the object's Last Known Writer Graph to assist in locating the token should it need it again. When a node receives a token it stores a pointer to the sender in the object's Modified Page Cache Graph to assist in locating the most recent version of a page. These pointers are stored in local token tracking data structures on the sending and receiving nodes,



The Modified Page Cache Graph: N5 currently holds the write enabling token W which originated at N1. N6 is attempting to read the Memory object.

Figure 3. Modified Page Cache Graph

respectively. Nodes wanting pages from a particular memory object locate the pages by following the Last Known Writer Graph to find the current write token holder. The most recent version of the page will be the first occurrence of the page found by following the Modified Page Cache Graph beginning at the current write token holder. Performance is enhanced because recently altered pages will be in a Modified Page Cache



Figure 4. Modified Page Cache Graph pruning. The Modified Page Cache Graph intersecting itself in A prunes itself in B when write enabling token W is transferred from N5 to N3. Modified Pages in N5 and N4 are copied to N3 with those from N5 taking precedence.

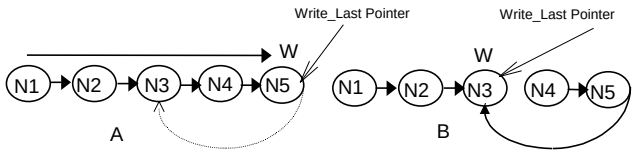


Figure 5. Last Known Write Graph pruning. The Last Known Writer Graph before (A) and after (B) pruning.

Graph node close to the 'starting' node (the one at the end of the Last Known Writer Graph) while page fault-based lazy updating leaves less often modified pages nearer the inception node of the graphs. Pages are moved only when they are needed. Consistency is guaranteed because when a page is read, the reader will access the most recent version first.

Two additional graphs, a *Copy-Set Graph* and a *Parent Graph* contain the information to define, control and provide some fault tolerance for the copy-set of a memory object. For a distributed MOCB the Copy-Set Graph, shown in Figure 6, is a tree rooted at the owner node. Each additional vertex corresponds to a node at

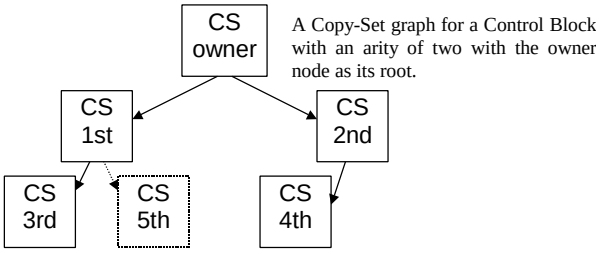


Figure 6. A Copy-Set Graph. Four nodes are currently in the copy-set. A fifth node, when added, will continue to fill out the balanced graph. The first node was the first to make a request, and so on. There is no other ordering.

which there is a copy of the MOCB. Additions to the tree occur when any node first requests access to a memory object. The Copy-Set Graph pointer of a MOCB on a given node is a vector with arity elements that point to the descendants of that node in the Copy-Set Graph tree. The Copy-Set graph is used to notify all nodes holding a given MOCB of important changes pertaining to it such as obsolete pages and node requests for exclusive access. The Parent Graph contains the pointers necessary to traverse a Copy-Set Graph from the leaves back to the root.

5.3. Distributed Protection Domain Copy-Set Management

Tracking of the control blocks for GPDs and CPDs called General Protection Domain Control Blocks (GCBs) and Thread Protection Domain Control Blocks (TCBs) in general requires simpler graphs than those used for MOCBs. GCBs and TCBs have two basic semantics: replication or unique. Unique semantics mean that only one active copy of a token for a control block can exist, and hence the control block can be active on only one node in the network. Replication means any number of token copies, one per node, called clones, can enable access with the access rights for the GCBs or TCBs as a whole being the sum of the rights of the GCBs or TCBs on the separate nodes. MOCB semantics will still prevail for memory access but multiple GCBs or TCBs operating on different nodes act in parallel and exchange data when useful to do so. GCBs and TCBs use the Copy-Set graph to perform actions on all of the clones for some operations such as determining if access permission exists to some MOCB. This mechanism is useful, for example, to support Distributed Parallel Computation [23] [24]. Sombrero has some default rules governing how and when MOCBs, GCBs and TCBs are migrated and mechanisms for user programs to override how migration occurs. This topic is discussed further in [21].

5.4. Client/Server and other Clustered Workstation and Distributed applications

The common global address space allows true thread migration in a SASOS. The Thread stack, thread private static data, thread ID, and other thread state, e.g., run status and priority, and general register contents can all be copied into the same virtual addresses and CPU general registers on the destination node. The thread can be immediately restarted and any missing data can be migrated to the thread. The performance of Distributed Shared Memory is also considerably improved in all SASOSs because mappings between multiple VA namespaces are unnecessary. The ability of threads to migrate in a SASOS enables servers to be constructed as passive entities. They can exist in a separate protection domain on another node and, in a hardware supported SASOS, be entered and activated by remote clients' threads via ordinary procedure calls. The existence of passive servers has a far-reaching affect on client/server performance because it simplifies service migration and replication via caching. These reduce the likelihood of a server becoming a bottleneck and also provide improved performance from local operations. Hardware support for a separate CPD and GPD adds flexibility to server design. Servers can be designed to either retain client information within their own GPD or to operate on this information when it is introduced as part of the calling client's CPD. In the former case, a copy of the server GPD would migrate to the client node. In the latter case, the thread and its messages and other initialization data contained in a carrier protection domain can migrate to the server's node or server code can be cached on the client node for efficiency. Two distributed system applications that can

benefit from SASOS operation are Object-Oriented DDBMSs (OODDBMSs) and real-time systems. OODDBMSs are a widely used and currently costly. OODDBMSs can profit from the common unique and permanent naming scheme, communications support for numerous large program modules and the sharing and protection support for a huge distributed database. Multi-node soft real-time systems such as those used for multimedia applications gain from the predictability of reduced real-time operations and multinode real-time applications used in industrial fabrication and also those within satellites gain because communication is faster using a shared address space substrate.

References

[1] Skousen, A. and Miller, D., "The Sombrero Distributed Single Address Space Operating System Project", *2nd USENIX Windows NT Symposium*, August 1998, p. 168.

[2] 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.

[3] Skousen, A. and Miller, D., "Operating System Structure and Processor Architecture for a Large Distributed Single Address Space", *Proceedings of PDCS'98: 10th International Conference on Parallel and Distributed Computing Systems*, October 1998.

[4] Pountain, D., "Amending Moore's Law", *Byte*, March 1998.

[5] Lo, J., Eggers, S., Emer, J., Levy, H., Stamm, R., and Tullsen, D., "Converting Thread-Level Parallelism to Instruction-Level Parallelism via Simultaneous MultiThreading", *ACM Transactions on Computing Systems*, August 1997.

[6] Yu, A., "The Future of Microprocessors", IEEE Micro, December 1996.

[7] Chase, J. "An Operating System Structure for Wide-Address Architectures", *Tech. Report 95-08-06*, Department of Computer Science, University of Washington, August 1995, Ph.D. Dissertation.

[8] Heiser, G., Elphinstone, K., Vochteloo, 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.

[9] Wilkinson, T. and Murray, K., "Evaluation of a Distributed Single Address Space Operating System", *IEEE Proceedings of the 16th ICDCS*, 1996.

[10] Dearle, A., di Bona, R., Farrow, J., Henskens, F., Lindström, A., Rosenberg, J. and Vaughn, F., "Grasshopper: An Orthogonally Persistent Operating System", *Computing Systems*, 7(3), pp. 289-312, Summer 1994.

[11] Soltis, F. G., *Inside the AS/400*, Duke Press, 1995.

[12] Rosenberg, J., and. Abramson, D., "MONADS-PC - A Capability-Based Workstation to Support Software Engineering", *Proceedings of the Eighteenth Annual Hawaii International Conference on System Sciences*, pages 515-522, IEEE, 1985.

[13] Skousen, A. C., "Sombrero: A Very Large Single Address Space Distributed Operating System", Department of Computer Science and Engineering, Arizona State University, MS Thesis, December 1994.

[14] Skousen, A. and Miller, D., "Operating System Structure and Processor Architecture for a Large Distributed Single Address Space", Computer Science and Engineering Department, Arizona State University, *TR-98-005*, August 1998 (an extended version of [3]).

[15] Skousen, A, and Miller, D., "Using a Distributed Single Address Space Operating System to Support Modern Cluster Computing", Proceedings of *32nd Hawaii International Conference on System Sciences, HICSS-32*, January 1999.

[16] Koldinger, E. J., Chase, J. S., and Eggers, S. J., "Architectural Support for Single Address Space Operating Systems", *Proceedings of 5th International Conference on Architectural Support for Programming Languages and Operating Systems, ACM SIGOPS Operating Systems Review*, October 1992.

[17] Torla, M. J., "RPLB Design Project Report", MCS Project, Computer Science and Engineering Department, Arizona State University, July 1997.

[18] Skousen, A. C. and Miller, D. S., "Resource Access and Protection in Sombrero: Protection Model, Software Protection Data Structures and Hardware Range Protection Lookaside Buffer, ASU 64-bit OS Group WP 2, version f, revised January 1995", Computer Science and Engineering Department, Arizona State University, *TR-96-013*, May 1996.

[19] Khatri, S., "Protection Structures in the Sombrero Operating System", MS Thesis, December 1997.

[20] Carnes, M., "Sombrero System Interface", MCS Project, Computer Science and Engineering Department, Arizona State University, December 1996.

[21] Skousen, A. C. and Miller, D. S., "Implementing a Single Very Large Address Space across Multiple Nodes: Memory Partitioning, Protection Domain Migration, Kernel Replication, Consistency and Fault Tolerance, ASU 64-bit OS Group WP 10, version e, revised January 1996", Computer Science and Engineering Department, Arizona State University, *TR-96-021*, May 1996.

[22] Skousen, A. and Miller, D., "Using a Single Address Space Operating System to Support Client/Server Distributed Computing", Computer Science and Engineering Department, Arizona State University, *TR-97-019*, July 1997.

[23] Dasgupta, P., "Parallel Processing with Windows NT Networks", *Proceedings of the USENIX Windows NT Workshop*, August 1997, pp. 113-122.

[24] Mclaughlin, D., Sardesai, S. and Dasgupta, P., "Calypso NT: Reliable Efficient Parallel Processing on Windows NT Networks, *Technical Report, TR-97-001*, Department of Computer Science and Engineering, Arizona State University, 1997.