Project Summary

The principal objective of this research is to demonstrate the advantages of an operating system supported by specially designed CPU-resident protection hardware to distribute a single virtual address space in order to reduce the complexity of distributed programming, and additionally, though less important, to improve performance. Secondary objectives include evaluating the advantages of using this operating system model to provide: a simplified but protected platform for embedded and real time implementations; reduced security provability issues for classified environments as a product of reduced complexity; improvements to distributed control system management, since every remote device in the network has a unique address and can be treated as a local device, and migration of software resources to users since program context is not bound to a particular site.

A prototype research operating system, running natively on top of a 64-bit RISC processor containing emulated CPU hardware for the protection model, is operational on test bed systems. The operating system and emulated hardware utilize the properties of a distributed single address space to provide application programs an object oriented network-wide view of system resources. It has a multithreaded design that provides scalable, low cost inter-task communication and concurrency control across protection domains on a network of homogenous machines. Operating system service modules and user applications co-exist in a simple flat peer-level organization of protection domains. It implements a *single* distributed shared environment for all system and user programs in a very large persistent virtual address space.

The method to be employed in this research is to extend and evaluate the existing prototype to determine and measure the gains in reduced program complexity that are made from moving the sharing, thread migration and other features of a distributed computing environment directly into the operating system instead of building it as middle-ware on top of a multiple address space operating system. Prior to the completion of the bare-bones prototype, a number of simulations were performed using a best guess of the system services interface. The necessary pieces need to be added to the prototype to enable verification of complexity reduction and such features as thread migration, passive servers, distributed persistence, load balancing and fault tolerance. These will be demonstrated by constructing simple reduced complexity applications such as a basic client/server distributed database management system, a secure distributed authentication service and a protected embedded application. The operating system will also be ported to IA-64 based computers which supply better support for 64-bit single address space operation than the Alpha processors (that have become obsolete) in the current prototype. This modern hardware will also allow the involvement of additional IA-64 owners to collaborate on continuing development. The IA-64 appears to be the logical successor to the Alpha.

The proposed activity is significant to the advancement of knowledge in the operating system, computer architecture and computer network fields. It defines a set of hardware and software design changes needed to make the paradigm shift from process-oriented multiple address space operating systems, using middle-ware to provide a distributed processing environment, to single address space operating systems, using native default distributed behavior, that are desirable and practical. It can lead to new ways of designing operating systems which would lead to better computing systems in the future. In particular this research can demonstrate a way to reduce application program complexity in a distributed system and thereby to substantially reduce software development costs for many systems. In many circumstances it may also provide better performance since the reduced complexity implies fewer instructions to carry out a task.

The intellectual merit of this activity is to revisit the generally accepted multiple address space paradigm for operating systems for deploying distributed computing environments. This approach now makes sense because of the alternatives provided by the availability of powerful processors that support a 64-bit address space. Most previous research efforts in this area have been directed at using single address spaces on individual computers and have run on top of multiple address space operating systems or micro-kernels. These have not demonstrated a sufficient advantage to overcome the current practice of using middleware over commodity operating systems and hardware to provide distributed computing since they were themselves middleware approaches. That approach results in complex solutions that are ultimately more costly to develop and maintain and are less efficient. The approach presented in this proposal can demonstrate the advantages of folding the complexity of inter-computer communication into default operating system behavior and allowing the low-level computing context for individual threads to become distributed. This in turn can demonstrate the advantages of a distributed single address space paradigm and generate more interest in this approach.

The outcome of this research will be publication of the results of this research and proposals to government and private industry for implementation of commercial and military versions of this OS paradigm. It should also provide a convincing argument to silicon chipmakers to invest in the on-chip protection hardware needed to fully implement a practical system.

Operating System Design to Reduce the Complexity of Distributed Systems

1 INTRODUCTION

Conventional process-oriented operating systems provide a separate virtual address namespace for every process and different namespaces for RAM and secondary store. Current distributed system applications either provide their own client/server algorithms, or more typically run on a layer of user level middleware that provides one, to map memory between address spaces, communicate via messages, migrate control and data between address spaces, and replicate, distribute and synchronize code and data across the network. This software adds considerable complexity to user visible software making it more costly to develop and maintain. It also reduces performance. Additional complexity is introduced because primary and secondary store are provided with different namespaces.

Much of the reason for separate virtual address spaces in conventional operating systems stems from the fact that the available 16 and 32 bit virtual addresses did not provide a large enough namespace for multiple applications to coexist within. It was necessary to reuse the 64KB or 4GB virtual addresses in individual processes, binding different values to them in each process. The appearance of 64-bit processors in the last decade has eliminated this problem for a large class of distributed applications running on clusters of workstations.

A proposed and often discussed solution to this problem is using these powerful 64-bit processors to provide a single very large 64-bit virtual address space for deploying distributed computing environments. Most previous research efforts in this area such as Opal [1] and Mungi [2] have been directed at using single address spaces on individual computers and have run on top of multiple address space operating systems or micro-kernels. These have not demonstrated a sufficient advantage to overcome the current practice of using middleware over commodity operating systems and hardware to provide distributed computing. They could not since they are themselves middleware. The commercially viable IBM System 36/38/AS400/iSeries is the exception and it is not distributed [3].

In this proposal we present an approach which premises the availability of protection hardware designed to take advantage of single address space properties and includes a design and implementation of an operating system to run on it that can demonstrate the advantages of the distributed single address space paradigm. This design folds the complexity of inter-computer communication into default operating system behavior and allows the low-level computing context for individual threads to become distributed. An operating system prototype implementing many of these ideas, called Sombrero, is operational. It includes a design and an emulation of the premised hardware. We are asking NSF to support the continuation of this project.

The target computers and most of our development systems run on DEC Alpha 21164-based computers. These were chosen because at the time we began prototyping it was the most suitable based on its 64-bit characteristics, available systems and hardware and software support. Little of this is true anymore. The company developing it has stated that the latest version is the last one that will be issued. IA-64 processors with true 64-bit virtual address spaces and significant single address space support in the form of protection key registers with domain granularity protection are available. And considerable software support including a 64-bit compiler is also available for them. To demonstrate the viability of the operating system design we are proposing, it is essential that we port it to a platform that provides protection within the instruction pipeline and to systems that are generally available and supported. The modern platform will also enable others who have available hardware to collaborate with us. This proposal includes a request to NSF for funding to purchase the needed IA-64 based systems.

The plan of work, in overview, is to spend most of the proposed 2¼ year timeframe extending and augmenting internal Sombrero OS classes, completing the Sombrero system services interface and running distributed and embedded applications on this system to corroborate functionality and measure code complexity. For example, we will complete the interface necessary to run the sample DDBMS systems we ran on a simulated Sombrero interface and re-implement these to run on the prototype. In addition we plan to restructure our current compilation/system generation/booting sequence to reduce and simplify the number of steps and to allow separation of kernel compilation, loading and linking from application compilation, loading and linking. We will extend the already existing debugging and disassembling tools to provide more information on network problems and runtime tracing tools. By the time this proposal period has started we expect to have the Sombrero prototype to the point where graduate students with reasonable operating system internals skills can be added to the development team without significantly slowing down its progress to get them up to speed.

The proposed activity is significant to the advancement of knowledge in the operating system, computer architecture and computer network fields. It defines a set of hardware and software design changes needed to make the paradigm shift from process-oriented multiple address space operating systems, using middle-ware to provide a distributed processing environment, to single distributed address space operating systems, using default distributed behavior, that are desirable and practical. It also defines a more flexible protection model for embedded and real-time systems that should make these systems easier to design and maintain. It can lead to new ways of designing operating systems which would lead to better computing systems in the future. In particular this research can demonstrate a way to reduce application program complexity in a distributed system and thereby to substantially reduce software development costs for many systems. Although less of a goal, this kind of operating system can cut overhead in many system services.

Since we last proposed to NSF for assistance on this project in 1998, a bare-bones native single address space operating system prototype has been constructed. It is being developed on three sets of 3-5 computer systems which include a W2K host computer for editing, remote debugging and user interface, an Alpha based Windows NT development computer for editing and compiling the Boot Loader and PALCode, an Alpha based Red Hat Linux 7.1 development computer for compiling the Sombrero executable modules and one or more target Alpha PCs that run Sombrero. Custom tools and services have been developed to run on the W2K host for communication, remote building and debugging, and the GCC compiler process has been augmented to modify intermediate output so that it is suitable for a single address space. Much of the Sombrero infrastructure is currently in place and it runs simple and multithreaded applications. The key pieces we are currently developing include a pager and virtual disk to provide persistence, distributing Sombrero to a second node and simplifications to the compile/boot sequence. These are expected to be complete by May 2004. We have run a number of simulations on distributed scheduling, reduction of software complexity and distributed consistency management using estimates of the expected Sombrero services interface. This work has been the subject of three MS theses and three conference papers. The Sombrero middle-level architecture was defined and an initial prototype implemented as a Ph.D. dissertation [4].

After an initial Arizona State University research initiation grant that provided the original research stations, the work on this project has been unfunded and the new prototype stations described above have been purchased with our own funds. We have relied on graduate students supported by industrial employment or consulting and teaching assistants for development and on the time available to the PI after fulfilling other teaching, research and service commitments at ASU. Prototype development has gone slowly because of the complexity of modern hardware and the software that initializes and runs directly on it and most of all because of the lack of funds to pay people skilled enough to work on the initial prototype. By May 2004, the prototype should be to a point where graduate students with fewer skills in these areas than the PI, postdoctoral associate and the industrial employee/MS student currently working on the Sombrero project can be productively added to the project. Most of the support from NSF requested for this project is in the form of salaries for the PI, postdoctoral associate and research assistants/associates. A relatively small amount is needed to purchase equipment to upgrade the initial prototype stations at ASU and for IA-64 based computers.

Our motivation for this research is that, as mentioned above, it is targeted at doing something that can be generally useful (and in an area we know a lot about) and that working on a system that starts from fundamentally different premises in the basic areas of memory management and protection is an interesting, exciting and potentially rewarding project.

2 SOFTWARE COMPLEXITY

Reduction in the complexity of user and system software is the major premise of our effort. Operating in a multi-address space environment substantially increases the cost of designing, implementing and maintaining applications over that which occurs in a single address space. This high cost comes about because multiple address spaces increase the complexity of the software that implements communication, sharing, resource access and process control.

The Sombrero single address space hardware/software architecture reduces application complexity by simplifying the global view of the programming model. It provides powerful system abstractions to developers that make many system implementation details transparent and allows programmers to concentrate on their applications. For example in [5] we compared a simple client/server file copy program written for a process-oriented multiple address space operating system with the same program written under Sombrero. The Sombrero file copy consisted of three

lines of code plus a few definitions while the process-oriented version contained dozens of lines of code and in addition referenced many more lines of file access and manipulation code via function calls. These also would not be in a single address space version.

In process-oriented operating systems for both distributed and single node applications, either the application itself or the application plus a layer of user level middleware provides a client-server set of programs to map memory between address spaces (DSM), communicate via messages, migrate control and data between address spaces, and replicate, distribute and synchronize all of this. Additional complexity is introduced because primary and secondary store are provided by different namespaces.

Table 1. Metrics Captured From Sample Database Application

	Halstead Program Effort	Halstead Bug Prediction	McCabe	LOC	Basic COCOMO
Windows					
Phase 1	2,637,658	6.36	85	852	2.03
Phase 2	4,287,457	8.80	121	1348	3.28
Phase 3	4,420,421	8.98	127	1420	3.46
Phase 4	6,204,183	11.26	145	1612	3.96
DSM[1]	4,987,458	9.73	132	1481	3.62
Sombrero					
Phase 1	1,025,712	3.39	67	639	1.50
Phase 2	1,084,757	3.52	65	647	1.52
Phase 3	1,135,836	3.63	68	719	1.70
Phase 4	1,135,836	3.63	68	719	1.70

1.This is the sample application under Windows 2000 with distributed shared memory.

In [5] we developed programs for a simple distributed data base management system under Windows 2000 and a simulated representation of the Sombrero system services interface (Sombrero was not operational at the time). We coded four increasingly functional versions of the application, starting with a single client and server, then adding multiple clients, next adding multithreading to the server and finally, adding multiple servers with replication and load balancing. We ran the first version on a single PC system and the rest on a 2 PC system to check for correct operation. We collected software complexity data using industry standard metrics: Lines of Code, Halstead (vocabulary) and McCabe (complexity) for each version under each operating system. The factors of 2:1 and 3:1 reduction in complexity (see Table 1) – depending on the metric and application version - were much larger than we anticipated and very large by software engineering standards.
Examination of the code showed that the complexity of mapped memory management, messaging and IPC, thread management, control transfer and server architecture and data distribution and load balancing were all reduced by taking advantage of single address space properties. In particular, memory management under W2K required mapping the user database and associated indices into the client's VA space and a memory mapping page handler to ensure the proper pages from disk were mapped into virtual memory. A windowing mechanism and pointer swizzling were necessary. Much of this was not transparent to the application programmer. In Sombrero the VA space is persistent and includes the backing store of the local node and all peer nodes. Additional mapping mechanisms are unnecessary. Pointers remain constant and swizzling is unnecessary.

To support communication between processes the sample DBS had to pass messages in order to move data from one namespace to another. In Windows 2000 this required the creation of a communication class to handle creating TCP/IP client and server ports, establishing IPC, message read, message write and close connection. In addition the reading and writing operations require the sending and receiving applications to agree on a message format. Mechanisms to pack and unpack messages and marshal and unmarshall arguments must be written. Also additional data transfer, buffering and binding software was needed. In Sombrero all processes share the same VA space and there is no need to pass messages. Either one process signals the other that data is ready, or the client accesses the server's methods, in a manner not unlike a local procedure call.

The Sombrero design facilitates the implementation of passive server objects. These are class instantiations that persist but do not have their own threads of execution. The client supplies a thread to a passive server when it accesses a server method. The implementation enables the client to access the server via an ordinary procedure call. In terms of control transfer, the client thread migrates to the passive server and 'animates' it. The client may or may not reside on the same node as the server and any required migration of service or thread across nodes is transparent to the developer and the application. Passive servers are one mechanism used in Sombrero to eliminate message passing. Multithreaded Windows 2000 versions of the simple DDBMS system required a thread dispatcher in the server to create worker threads. Sombrero did not need this.

Load balancing in the Windows 2000 DBS was implemented in the client by distributing its read requests, round robin, to a predefined pool of available servers. Data consistency under replication was implemented in the DBS server. Each server forwarded open, close and write requests received from a client to a single neighbor server, which, in turn forwarded the request to its neighbor, etc., resulting in all operations that modify the state of the database being executed on all copies of the database. By default, all applications are distributed in Sombrero. The Sombrero scheduler, unless instructed to do otherwise, considers neighboring resources in the network when determining where a thread of execution should run and transparently migrates (replicates) the data to the node running the thread or migrates the thread to the node with the data, based on its knowledge of current system state. Scheduling, data consistency and load balancing are transparent to the application and developer [6].

The results described above must be repeated on an operational distributed Sombrero system to determine whether the simulated interface represented what could be practicably provided in a robust, persistent and fault tolerant way with reasonable performance. We will measure and assess complexity as we did in our simulations. Performance will be estimated by backing out costs incurred from RPLB emulation. We are asking NSF for support to complete this objective.

More details on the Sombrero approach to complexity reduction can be found in [7].

3 SECONDARY OBJECTIVES

The principal objective of this research is to demonstrate the advantages of an operating system supported by specially designed CPU-resident protection hardware to distribute a single virtual address space in order to reduce the complexity of distributed programming, and additionally, though less important, to improve performance. Secondary objectives include evaluating the advantages of using this operating system model to provide:

1) A simplified but protected platform for embedded and real-time implementations.
2) Reduced security provability issues for classified environments as a product of reduced complexity.
3) Improvements to distributed control system management, since every remote device in the network has a unique address and can be treated as a local device.
4) Migration of software resources to users since program context is not bound to a particular site.

3.1 Real Time and Embedded Systems

The Sombrero operating system software and protection hardware can also be useful in the design of single board embedded systems and real-time systems. These systems typically would like the protection provided by virtual memory systems but can not use it because of lack of a disk or insufficient RAM or the unacceptability of the unpredictable delays page faults impose.

This proposal presents an approach in which the desired more general protection model is provided in the single address spaces these embedded and real time systems run. The decoupling of protection and address translation present in the Sombrero architecture implicitly provides protection when it is needed. The complexity of embedded and real time software should be reduced because of the elimination of locks and other workarounds conventionally used to provide the necessary protection and also because of the association of threads with protection domains. As in distributed systems the reduced complexity should decrease development and maintenance costs. It is well known that these systems are particularly prone to nasty bugs because of the lack of adequate protection. We are asking NSF for support in prototyping our ideas for improving protection in embedded and real-time systems.

3.2 Reduced Security Provability Issues

Given the reduced complexity of Sombrero programs and the peer level modular design of system and user programs it seems reasonable that the accessibility of services and data can be limited to simple protected interfaces. It is possible these dedicated protection domain interfaces are more provably secure than that found with a hierarchically designed protection mechanism that exposes all secure information at the same level to unneeded services and programs that also occupy the same protection level. We would like to study this potential aspect of Sombrero after the primary objective has been accomplished.

3.3 Improvements To Distributed Control System Management

Since a device driver represents every device on every node in a Sombrero network and every program has a unique address in the distributed network the distributed virtual address space can appear to an application to be a single large computer. With this perspective it seems reasonable that control system design can benefit especially as new hardware is added or removed from an existing production system or modifications are made on existing equipment. Existing production systems tend to be eclectic with difficult interfaces. A control system built around a distributed virtual address space would simplify access issues and modifications.

3.4 Migration Of Software Resources To Users

Since program context is not bound to a particular site in Sombrero a thread context stored on one computer in a Sombrero distributed virtual address space is valid if it resumes execution on another computer in the same VA space. This would freely permit an executing context to follow a user or device allowing simple mobility and best response time as programs migrate to the user's nearest computer. This combined with control system management should provide some interesting military applications.

4 SOMBRERO SYSTEM ARCHITECTURE

Sombrero system architecture is designed around the concept of a set of independently protected service modules that provide their services through entry points. Unlike legacy systems, there are no privilege levels, e.g., kernel mode and user mode. However, Sombrero does make use of PALCode executing almost entirely in the physical address space to provide some services. This affords a gray area with regard to the kernelless design in which not only is the RPLB emulated but also context switches, interrupt vectoring, wait queue management, CPU resource locking of privileged PAL vectors to system modules, and other register like services. (See Figure 1)

4.1 Object Oriented System Modules

Abstractly, the system modules appear as classic object oriented entities with methods and an encapsulated protected content. Access to the methods is determined by access policy and can be used to create access hierarchies implementing object oriented inheritance features. Modern kernel based operating systems are also typically hierarchical internally (NT), or even monolithic (UNIX), but have a flat defined API from the user point of view that is implemented directly or indirectly as a system trap. This limits user programs to services exposed by the API, which is a desirable but inflexible feature.

The Sombrero design is more flexible, for example, for real time requirements since base class system entry points can become available through access policy changes. A real time system or a driver might need, for example, to bypass system accounting features to allocate a block of memory in order to meet time constraints if access time is deemed more important than accounting. In a kernel based system, the real time requirement would either have to be built into the kernel (loadable module), exposing the integrity of the kernel when new code is introduced, or a new system trap would have to be supported by the vendor, requiring a system release, perhaps undesirably exposing lower level access to user access in non-real time systems. In Sombrero, the administrator who manages system policy can expose the memory allocator by creating a permission to access the low-level memory allocation entry point. No system code modifications are required. Perhaps more importantly however, is the ability of designers to build enforceable protected hierarchies using the system modules as base classes. By this means, system presentations for special users (such as a Windows or UNIX API) can readily be built with completely protected hierarchies.

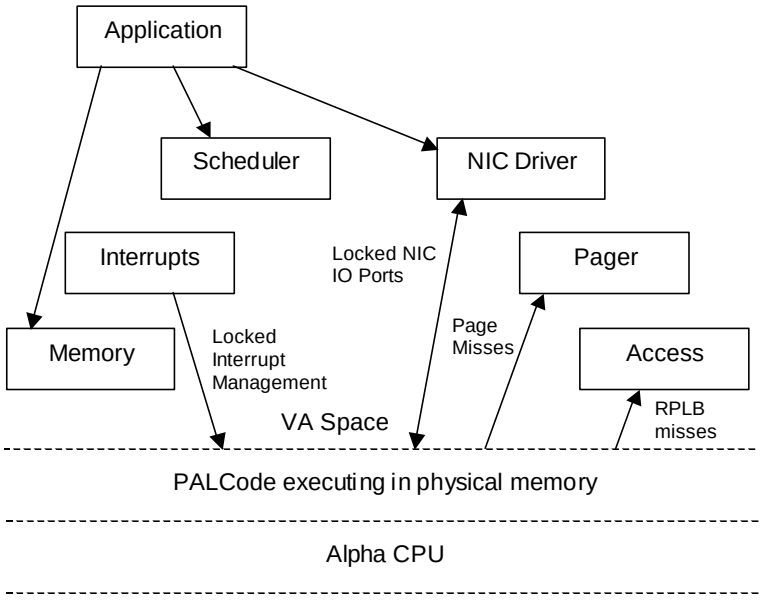


Figure 1: Sombrero Architecture showing some services and relationships to PALCode and Hardware. Arrows indicate direction of initiated access.

4.2 System Protection

Protection in Sombrero is designed to be functionally transparent and ubiquitous. Transparent because protected module entry points are entered implicitly rather than requiring an explicit protection domain switch, and ubiquitous because *every* memory access is checked against the system PRAL via the RPLB to insure that the correct access policy is enforced. Most SASOS designs rely on capabilities. Software implementations of capabilities require explicit application of the capability in order to perform a protection domain switch. Sombrero records access policy, including protection domain switches, in the PRAL and at a lower cache level, in the RPLB, where the CPU has access to it. This allows the CPU to initiate a domain switch when it detects the need and permission is available. It is also costless since it is done in parallel with program execution except on an RPLB miss. This strategy also contributes to reduction in program complexity for Sombrero[1]. The theory and logical design of the RPLB can be found in [8] and [9].

Data structures that support the protection mechanism includes the General Protection Domain (GPD) control block, the Carrier Protection Domain (CPD a.k.a. thread protection domain) control block, Memory Object (MO) control block, and Accesses Descriptors (ADs). The control blocks, that provide information to the system about protection domains and memory objects, are all interconnected by the Access Descriptors. This interconnection constitutes the Protection and Resource Access List (PRAL). Additional details are available in Appendix C.

A protection domain is the set of all code and data that is reachable by a thread of execution in its current context as well as the set of protection domain switches available to the thread. Protection domains in Sombrero are actually the union of a GPD and a CPD. The GPD is the same as the protection domain definition, but can be exited by the thread for another GPD. The CPD is data and allowed switches, but not code that remain associated with the thread when it switches to a new GPD. By this means, access to memory such as the thread stack can be shuttled between GPDs without needing to have an AD for every occurrence. For this and other advantages of the CPD, see [8][9]10].

4.3 Token Tracking

The proposed set of algorithms collectively labeled Token Tracking manage a set of graphs that are used to locate updated pages belonging to memory objects and maintain data consistency over the network. Token tracking

[1] It should be noted that capabilities require the addition of at least one more namespace, the management of which increases program complexity.

also effectively distributes the access policy stored in the PRAL between requesting nodes using control block surrogates. Theory and figures for token tracking can be found in [9], [11] and [12].

4.4 Pager

The Sombrero pager, unlike pagers usually found in a MASOS, is designed to present the same view to all programs running on a Sombrero network. The pager depends on a consistency service to obtain the correct pages from other pagers on the net. Pages are presented on demand on a page fault after an attempted access has cleared the protection mechanism. Like other pagers, it has a component to support virtual paging in RAM as well as backing to a persistent store.

Since Sombrero is implemented in a 64-bit address space, the pager cannot use traditional page tables indexed by the VA without consuming a significant portion of RAM. Instead, the existing implementation uses an inverted page table and a hashing technique to index into the page table. By this means, the RAM consumed by the pager always remains proportional to the RAM available on the current node.

As mentioned before, in Sombrero there is only one view supported by the pager for all programs and so only one set of page translations exist per node. In contrast, a MASOS must support a distinct set of translations for each process. As a consequence of the single set of translations found in Sombrero, it is feasible that, unlike a MASOS, a hardware accelerator (e.g., a Content Addressable Memory, (CAM)) on the system memory bus, relieving the CPU of all but table updates, can support the single set of translations. This should bring the translation performance of even an inverted page table up to par with that of MASOS processors serving smaller address spaces.
Another effect of the Sombrero design is the perception of the role of RAM in the memory hierarchy. MASOS designs tend to be RAM-Centric or focused on moving data and programs into and out of RAM as mentioned earlier with respect to the need to preserve data between program executions. Sombrero allows RAM to be treated more like a level in the cache hierarchy enabling well understood cache schemes to be employed in its support.

4.5 Persistence

VA space persistence in Sombrero is a core issue that affects the degree to which system and program complexity may be reduced from that of the MASOS paradigm. The Sombrero design proposes to support the persistence of the VA space for the life of the data it stores, relieving the program designer of the requirement to convert data to a differently organized non-volatile secondary storage, such as a file system, for long-term availability. However, the VA space is intuitively identified as the most vulnerable part of the Sombrero design. A VA space is vulnerable to corruption and so is any other computer resource on any computer. Although the experiments with distributed persistence in Sombrero have not been completed, there is an argument that can be put forward in support of the Sombrero design to address the apparent vulnerability. First, a few observations need to be made.

VA space persistence is an essential contributor to reduced complexity in Sombrero and so it is valuable even if it is vulnerable. Past experience suggests that, in general, computer systems become corrupt and need to be rebooted. Forms of corruption include unhandled program exceptions, power failures, and equipment failure. Corruption seeded early in a computational effort by human error or bugs propagates for several backup cycles until it is noticed but recent copies of data will now only reseed the problem. Many forms of corruption exist in current designs, so what happens to the data in Sombrero when a failure occurs? Is it worse? Of course some loss occurs. All of these problems exist to some degree in every computer system. However, users have learned to coexist with them, and techniques to cope with them get better over time.

The traditional MASOS line of defense against a corrupted operating system is the division between the persistent store, typically a hard drive, and the more corruptible volatile store in RAM where the operating system runs. If there is a failure, just press the reset button and everything is rebuilt from the disk. Using this line of defense gives a fallback position to what is typically considered a more reliable device: the hard drive or even additional levels of backup. This fallback position is the comfort zone in which we have learned to coexist with our computers. The elective absence of that fallback position in Sombrero (it is still available if you want it, but program complexity increases) makes people feel uncomfortable, but the reality is that none of the above mentioned system corruptions are erased by the fallback strategy, only mitigated. Disk drives still fail. Tapes and CDs become unreadable. Rebooting won't cure bad data stored on the disk.

Here is the argument: If reliability techniques have been developed in the MASOS paradigm, workable techniques to coexist with computers in the Sombrero paradigm can also be developed. These techniques can be found if maintaining the reduced complexity and the accompanying cost reduction for development offered by Sombrero makes it worthwhile to do so.

Reliability techniques in the works for Sombrero include the protection model featuring the RPLB, system snapshots for single nodes, fault tolerance based on replication over the network using RAID like techniques, snapshots of consistent data sets with intermediate logging and others. The major aspect of these efforts is that basic reliability is a function of the operating system rather than the application. Programs will still need to guard against their own bugs.

The several preceding design issues discussed have been the guiding influence for the top down part of Sombrero design. The bottom up influence has been the available platform and how it can be utilized. No formal specification document was developed for the middle level architecture, instead, this being an experiment, each problem and its solution has been handled in order and applied. This was done to gain experience with the expectation that a formal specification of the operating system can be developed for an implementation of Sombrero in the next iteration of the design. The objective of research into the Sombrero persistent store is to establish the best means to provide reliable persistence at different levels of scale. The strategies differ as the distributed space includes a larger set of computers or greater latencies in the network due to distance and network bandwidth.

5 RELATED WORK

Distributed systems are currently implemented over commodity operating systems as middle ware. This introduces complexity that appears in user applications and in underlying middleware support that makes development and maintenance more costly [13][14][15][16]. With the advent of 64 bit and eventually larger processors a number of researchers have investigated the Single Address Space paradigm only to implement these ideas as middle ware as well.

The IBM iSeries [3] is the only SASOS in widespread use. It has a Single Address Space Operating System comprised of a low level piece, the System Licensed Internal Code (SLIC) and a higher level piece OS/ iSeries and has a well defined interface, called the Machine Interface (MI) between them. It runs on Power PC 64-bit RISCs. There are probably over 100,000 iSeries systems in commercial use. The system architecture is targeted at database management-type usage. The iSeries has a great deal of special purpose hardware. All of memory has additional tag bits to differentiate between protected pointers and ordinary data. Secondary store has tag fields for the same purpose. The Power PC architecture used in the iSeries has additional instructions for tag management and other purposes. OS/ iSeries supports a single large address space on each node but this does not appear to span a network of machines, even though a principal class of users are client/server systems. The internal design is object-oriented, as is the MI available to the outside world. Protected pointers provide most of the protection for user level programs and data. Capabilities are used internally. The iSeries MI is a technology independent interface that converts the underlying 64-bit single address space to an externally available high performance object-oriented single 64-bit address space especially suited for database management systems. The full set of single address space properties is not available to applications. The iSeries is a very successful industrial strength product, demonstrating the feasibility of a SASOS in its targeted application area.

Most contemporary research SASOSs have been designed to run on stock RISC 64-bit processors. These include Opal [1], Mungi [2], Arius and Grasshopper. This means additional software communication protocols are required for protection. 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 [3], Monads and Sombrero [17]. Special Hardware support typically enables or significantly facilitates the protection mechanism and the granularity of access. In Sombrero it also enables implicit domain switching.

6 CONTRIBUTIONS

Specific contributions [4] made by this research include the development of ideas that have been suggested or modeled elsewhere as well as original material. These contributions arise from the implementation of the Sombrero architecture and include the following items:

?? Protection Model – The protection model used in Sombrero is based on the legacy protection model found in TLBs and the extensions suggested by Koldinger [1992] for the PLB in a single address space. A proposed protection model called the RPLB, which is further extended, compared to the PLB, to provide object based rather than page based protection using region masks. This has been implemented in emulation and supported in system software (See appendix B).

?? Carrier Protection Domain – A protection domain associated with the thread of execution. A proposal that the state associated with a thread of execution be extended to allow it to access its own data store no matter what GPD it is executing in. This is important in trusted relationships to reduce the overhead associated with giving a thread access to its stack or other buffers regardless of the GPD association. CPDs are not allowed to have execute access except for associated GPD switches.

?? Domain Switching – Domain switching is a well-understood concept but is usually performed explicitly. A domain switch in Sombrero is implicit and usually costless and is implemented in the RPLB. Use of an implied domain switch is a major contributor to program simplification in Sombrero.

?? Kernelless Architecture – Making use of the new protection model to provide the boundary between system and user allows a simpler model for system design. Sombrero implements a non-existent or nearly non-existent kernel. The absolute need for a kernel or something like it such as PAL Code has yet to be determined, but it is already apparent that the kernel can be greatly simplified in a SASOS. Sombrero is built around the kernelless architecture.

?? Binding Hardware Resources to Protection Domains – Sombrero uses a strategy of binding CPU and other hardware resources to individual protection domains to provide the protection normally provided by a kernel. This is enabled by the availability of protection domains directly supported by the hardware.

?? Programmable Hierarchies – By using the Object model, including inheritance and access to base classes, a system model that can be molded via policy management to suit a particular environment such as real time or embedded systems vs. a general computing environment is available.

?? Reduced Complexity – Compared to traditional MASOS systems and even proposed SASOS systems, The Sombrero design has carefully tried to take advantage of the single name space to fold complexities that are normally addressed in the program model into the default behavior of the operating system. No doubt more such opportunities remain to be found, but this effort has been largely successful at reducing complexity.

?? Entry/Return Point Mechanism – Sombrero has a program entry/return point mechanism that supports both a local instantiation context switch as well as a protection domain switch. The entry/return point is designed to be unaware of its participation in a GPD switch, relying on the *implied* GPD switch mechanism, so that it can operate the same in interdomain or intradomain services requests. This generic behavior contributes to reduced complexity.

?? Tail switch – Sombrero implements the Tail Switch, an implied switch that provides an automatic return switch to a protection domain from a service in another protection domain. Tail Switches are pushed at entry points only when an actual GPD switch occurs, remaining transparent to the program. Tail Switches are popped at return points with the same advantage of transparency as GPD switches.

?? Compiler Support – Sombrero has compiler support to allow the C++ program language model to be used to program access to external services via entry/return points.

?? Semaphores and Locks – Semaphores and locks take advantage of the single address space. These do not need to be registered in the operating system to be globally available.

?? Passive System Services – Instead of passing data between threads passive services are animated directly by user threads. This avoids both the thread communication protocol and a thread context switch. Passive services are not new to Sombrero but Sombrero makes more extensive use of the concept by allowing system services to be passive subject to trust relationships.

?? Blocking Interrupts – Sombrero implements blocking interrupts that allow threads to block in a registered GPD and provide the context for handling the interrupt. 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.

?? Distributed Consistency in a SASOS – Sombrero has a mechanism to manage consistency and distribute data across the network on an object based rather than a page based granularity. Pages still provide the basic granularity of storage, but pages are not individually tracked rather the objects associated with the pages. Referred to as token tracking.

?? Surrogate Pointers – Sombrero uses surrogates for system control blocks that allow network routing to be based on the address of the pointers to the surrogates. This is how the protection and access system modules distribute the virtual address space.

7 PROJECT TIMELINE

The sequence of events in our proposed research include:

Year One -

?? Initial refinement of system tools such as the debugger
?? Development of additional hardware drivers
?? Introduction of metric devices into the system modules to profile exact behavior
?? Begin acquisition if IA-64 hardware
?? Continue experiments with consistency management and replication for fault tolerance
?? Implementation of sample applications to measure complexity reduction
?? Begin port to IA-64 hardware

Year Two -

?? Paper on the effectiveness of complexity reduction and potential refinements
?? Paper on issues in distributed address space persistence
?? Addition of security features and comparative study to current designs
?? Experiments with thread migration and user mobility on small network
?? Complete IA-64 port
?? Make Sombrero available to larger community for scaled testing
?? Propose concepts to Industry with performance evaluations
?? Publish papers on Security and mobility

Many of these steps will be in parallel as additional talent is added to the project.

APPENDIX A
DISTRIBUTED SYSTEM CONSISTENCY MANAGEMENT

Applications running on process-oriented systems typically require an additional layer of middleware support to provide distributed consistency management in the form of a Distributed Shared Memory (DSM) package[15] or a runtime system providing remote object invocation[14]. These approaches scale poorly, have limited transparency and provide little flexibility in addition to considerable performance problems.

Sombrero uses the properties of a single address space to make distributed consistency management transparent, scalable and flexible.

The DSM in process-oriented operating systems in general [16], uses a distributed namespace to translate between distinct underlying VA namespaces on the various nodes. The usual VA-to-Physical address translations occur between these distinct VA namespaces and their underlying physical memory and swap files on the various nodes of a DSM system. A translation between the local VA and the DSM distributed namespace is needed for every inter-nodal message passing, RPC, and page shuttling occurrence. The DSM VA must be again translated to the VA on the receiving node.

The Sombrero design contains a distributed system of user-definable consistency semantics on a per-object resource by resource basis called Token Tracking[12]. Token Tracking refers to services that enforce consistency semantics between copies of objects on different nodes. It allows distributed resource management to occur at a very low level, the level at which resources are created. Object grained protection hardware directly supports object regions rather than pages as the base unit of network distribution. It also enables per-object copy-set management to replace the per-page copy-set management found in process-oriented systems. 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 (protection domain, 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 conventional DSM systems. And as in DSM, the demand-paged virtual memory management 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 [11]. Only the data structures in use at a node need be present at it. In Sombrero, there is no middleware needed to do page shuttling, namespace translation and message passing. Additionally, the application program developer need not have knowledge of either individual pages or the network and consistency semantics policy can be defined by the user and tailored to the application. No special handling is needed by the application to control or monitor distribution of a memory object. The Copy Set is automatically managed by Sombrero. Similarly, the Paging Service is built into Sombrero and no additional paging or communications services, such as those commonly found in current software-DSM middleware, are needed to implement distributed shared memory. Only the information that is needed locally is stored locally and the Consistency Service makes communications calls directly to the affected nodes not relying on broadcasts. The Communications Service can use virtual addresses for routing. Since all execution shares the one address space in Sombrero, and, since that address space is distributed, all memory in Sombrero is intrinsically Distributed Shared Memory.

When we were evaluating this design approach, because the Sombrero prototype was not available, a simulation using these algorithms and data structures to provide sequential consistency and a multiple reader/multiple writer consistency across a distributed system was developed under Windows 2000 to show implementation feasibility and to observe the dynamic behavior of the consistency algorithms. [12].

As part of the work under this grant, we propose to add the consistency manager and its interactions with the protection, communication and memory managers to the distributed Sombrero prototype to check out its correctness, logical design, scalability and performance and its robustness under node restarts.

APPENDIX B
SOMBRERO HARDWARE ARCHITECTURE OVERVIEW

This section summarizes the Sombrero distributed operating system design from the hardware aspect.

Model

The basic abstractions in Sombrero [8] are memory object, protection domain, thread and principal. The basic storage abstraction is the ***Memory Object***. All resources are represented and named by memory objects. A memory object is characterized by its range. *Memory ranges* are distinct, variable sized, contiguous allocations of the virtual address space. Memory ranges contain *memory regions*, which are possibly noncontiguous sets of virtual addresses within a range with one or more non-overlapping protection attributes from the set {rwx}. The basic protection abstraction is the ***Protection Domain***. It is an abstract entity describing the set of memory objects and protection domains that are reachable by a thread operating within it and their associated access permissions. There are two kinds of protection domains, *General Protection Domains* (GPDs) and *Carrier Protection Domains* (CPDs). GPDs contain the code, data and other GPDs accessible to all of the threads executing within them and CPDs contain the data and the GPDs privately accessible to an individual thread. At any instant the effective protection domain is the union of a GPD and a CPD. A ***thread*** is the unit of execution represented by the current or most recent relevant register state of the CPU including the contents of the instruction pointer, stack pointer, general registers, reachable CPD memory and some additional status information. A ***principal*** is a unique protected identity that represents a user or a system service. It binds the resources, activities and access privileges carried out on behalf of programs to an owning user and is used to manage authorization of system resources.

Hardware Design

As discussed earlier, it is necessary to add protection hardware to support a resource access model that makes full use of the computing potential of a single address space. To evaluate whether every memory access can be allowed, this hardware must be CPU-resident. This is because it must utilize data emitted by the CPU during the execution of every instruction that accesses memory. Unless this is done, explicit time-consuming software communication protocols must be used at run-time to achieve the same ends. The Sombrero design includes a design of this hardware called a Region Protection Lookaside Buffer or RPLB.

Functional Requirements

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 Sombrero RPLB is a protection cache that provides these features. To do this requires a cache for protection information and 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:

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.

<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 [18].

Logical Design

Figures 2 and 3 give an overview of RPLB intra-domain operation. The unit of protected resource access in Sombrero is a memory region. A region is a possibly noncontiguous collection of bytes with fixed access privileges from the set {rwx}. Its virtual addresses can be defined by a Match Mask and a Don't Care Mask as shown in Figure 2.

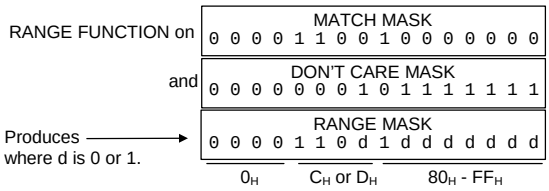


Figure 2. RPLB Range Mask defining a region consisting of virtual addresses from $0C80_H$-$0CFF_H$ and $0D80_H$-$0DFF_{H.}$

Sombrero associates a protection domain called the Carrier Protection Domain (CPD) with the thread being executed and a General Protection Domain (GPD) with the code and data the thread is executing. The current values of these, actually the virtual addresses of the control blocks that store data about them, are stored in two high speed base registers, CPDBR and GPDBR as shown at the bottom of Figure 3.

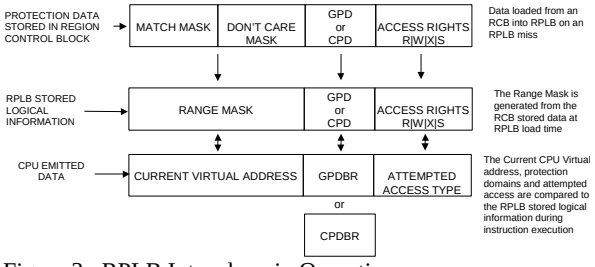


Figure 3. RPLB Intra-domain Operation.

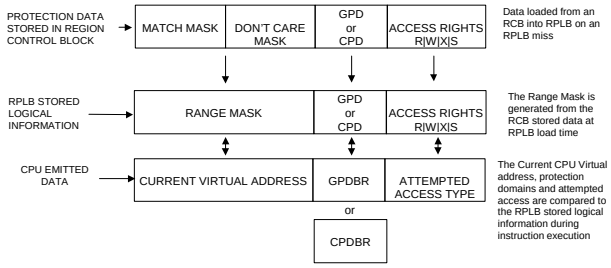


Figure 4. RPLB Cross-domain Operation.

An RPLB entry for part or all of a region is shown at the top of Figure 3. The top set of fields is loaded into the RPLB on a miss. When the RPLB is loaded, combinational logic generates a Range Mask that defines the set of valid addresses stored in the RPLB. CPU emitted data is compared against the RPLB stored logical information on every memory access to determine whether the access is allowed.

Figure 4 gives an overview of RPLB inter-domain operation. To determine whether a new domain can be entered the RPLB needs to store the virtual address that is being accessed, the protection domain attempting to make the access and a flag {s} indicating that this entry specifies an attempted cross domain switch rather than normal program operation. In addition, it must provide the virtual address that is the name of the new General Protection Domain for storage into the GPDBR on a valid access.

The top set of fields is loaded into the RPLB on a miss. The CPU emitted data is compared against the RPLB stored information to determine whether the domain crossing is allowed. The RPLB associatively compares each entry against the CPU emitted data. For allowed accesses the new GPD name is stored in the GPDBR and this is followed by an intra-domain access attempt. More information on RPLB operation can be found in [9] [8] [19].

RPLB VLSI Synthesis and Hardware Emulation

The RPLB design was described and proof of concept verified in VHDL (VHSIC Hardware Description Language), using the Mentor design tools available in the ASU ULSI Design Laboratory [20]. Subsequently, synthesis, the process of converting a language description (in this case VHDL) into a schematic or netlist description, was performed using Mentor's Autologic II synthesis tool and Synopsys Design Compiler and the Cascade Design Automation static timing analyzer, Tactic, performed timing analysis [21]. Timing analysis indicated RPLB operation would be competitive with DEC Alpha TLB operation. More details can be found in [21]. In the prototype, DEC Alpha PALCode has been has been extended to emulate the RPLB, which is triggered by misses in the TLB. More on this can be found in [22].

APPENDIX C
ADDITIONAL ARCHITECTURE DETAILS

Software Protection Data Structures

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. One type, called Protection Domain Access Lists, define a protection domain. Each list is headed by a control block representing a GPD or a CPD and contains Access Descriptors that indicate the resources and protection domains accessible to the GPD or CPD. These lists contain the same fundamental protection information held by classical capability lists. The second type, called Resource Access Control Lists, contain the list of valid accessors for a resource or a protection domain. A control block representing a memory object or a GPD heads each list. These lists are classical ACLs. In fact, because PRAL lists could be quite long and because the memory objects represented vary in size they are implemented as binary search tables or hash tables. PRAL contents are updated implicitly on a protection fault or on user request via Sombrero Systems Service routines. The attributes of a single address space facilitate the co-location of protection triples. The RPLB and PRAL designs make use of this providing basic support for the operating system structure described in the next section.

Sombrero System Structure

This section provides a high level overview of Sombrero system structure. 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. This enables one to take advantage of the characteristics of a single address space to design an operating system as a set of peer level services.

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 performed in process oriented systems by system calls implemented by trap instructions, and upcalls by executive services to be implemented by ordinary procedure calls.

Sombrero has no central kernel and distributes the services it typically performs among the executive services. Lock registers are used to limit access of CPU resources via privileged instructions 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 1, 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 in Sombrero as these impose a non-extensible hierarchy to system structure.

1. Memory object, thread and protection domain migration supported by distributed surrogate kernel data structures enabled by tokens
2. Transparent and implicit consistency and concurrency policy - every VA is tied to a specific policy
3. Granularity of data transmission determined by policy
4. Copy-set management uses local data/minimizes broadcasts
5. Network routers use virtual addresses

Table 3. Sombrero Distributed System Features

Distributed System Design Overview

In Sombrero the global VA space is initially partitioned among 'owning' nodes with subsequent refinements as nodes enter and leave a cluster via a fissioning and merging technique [SKMi99a] [SkMi99b] [SkMi96b]. 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 routing software uses this unique address range to route network packets based on the destination memory address of the message. 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 MOCB. 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 an authorizing agent by the corresponding Sombrero system service. The migration and replication of kernel objects using surrogates make the network's multiple computers appear as a single virtual non-uniform memory access multiprocessor. Additional information on Sombrero distributed system design can be found in [23] and [9]. Table 3 lists the Sombrero Distributed System Features.

References Cited

[1] 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.

[2] 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.

[3] Soltis, F.G. "Fortress Rochester, the Inside Story of the IBM iSeries", NEWS/400 Books division of Penton Technology Media, 2001.

[4] Skousen, A.C., "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

[5] Feigen, R., Skousen, A.C., and Miller, D., "Reduction of Software Development Costs under the Sombrero Distributed Single Address Space Operating System", *The 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2002)*, June 2002.

[6] Patil, M. "Distributed Scheduling in Sombrero, A Single Address Space Distributed Operating System". MS Thesis, Computer Science and Engineering Department, Arizona State University, December 1999.

[7] Feigan, R. "Reduction of Software Development Costs under Sombrero, A Single Address Space Distributed Operating System". MS Thesis, Computer Science and Engineering Department, Arizona State University, December 2001.

[8] 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*, Computer Science and Engineering Department, Arizona State University, TR-95-013, May 1996.

[9] 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.

[10] Khatri, S., "Protection Structures in Sombrero, A Very Large, Single Address Space, Distributed Operating System", MS Thesis, November 1997.

[11] 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*, Computer Science and Engineering Department, Arizona State University, TR-95-021, May 1996.

[12] Olson, J. "Distributed Consistency Management in Sombrero, A Single Address Space Distributed Operating System", MS Thesis, Computer Science and Engineering Department, Arizona State University, August 2002.

[13] OSF DCE 1.2.2 Application Development Guide - Core Components, The Open Group 1997.

[14] Wollrath, A., Riggs, R. and Waldo, J.: "A Distributed Object Model for the Java System", Computing Systems, vol. 9, no.4, pp. 265-290, Fall 1996.

[15] Protic, J., Tomasevic, M., and Milutinovic, V.: "Distributed Shared Memory, Concepts and Systems.", IEEE Computer Society Press. 1998.

[16] Tanenbaum, A.S., Van Steen, M. "Distributed Systems, Principles and Paradigms", Prentice Hall, 2002.

[17] 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.

[18] 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.

[19] Skousen, A. and Miller, D., "Hardware/Software Design in a Distributed Single Address Space", Computer Science and Engineering Department, Arizona State University, TR-97-018, July 1997.

[20] Ganesan, R., Sukavanam, V., and Torla, M., "Range Protection Lookaside Buffer, VHDL Synthesis Report", Term Report CSE 518 VLSI Synthesis using HDLs, December 1995.

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

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

[23] [SkMi97a]Skousen, A. and Miller, D., "Hardware/Software Design in a Distributed Single Address Space", Computer Science and Engineering Department, Arizona State University, TR-97-018, July 1997.