Project Summary

The objective of this research is to demonstrate the user-friendly programming environment and high performance that can be provided by a single address space operating system (SASOS) that is supported by CPU-resident protection hardware. A research prototype SASOS running on top of an emulation of a 64-bit RISC processor with additional CPU hardware for protection will be completed and evaluated. Recent large-address space architecture operating systems built on stock 64-bit RISC processors have demonstrated feasibility of the approach as well as comparable performance to that of contemporary process-oriented operating systems. They have not, however, produced the kinds of gains required to overcome the disadvantages of shifting to this paradigm. A basic assumption of this research project is that additional protection hardware within the CPU is required to produce these kinds of gains. In addition, the advantages must apply both to single nodes and to clustered workstations on a LAN.

It is well known that single address space properties can be utilized to provide a more efficient substrate for data exchange and synchronization between programs, elimination of the file system for persistent storage and support for execution migration across protection domains and machines. These can result in improvements in the programmability and performance of tasks that need to work together and access shared databases. A major premise of this research is that adding protection hardware to a 64-bit RISC processor provides two additional underlying features. First, changes to the hardware enable simple implicit communication and sharing mechanisms to replace inefficient and cumbersome software workarounds. Second, the architecture of the operating system executive itself can be changed to a collection of modular peer-level interacting protection domains. These can interact with user programs as additional peers via ordinary procedure calls and their services modified by run-time user requests. Hardware support for these domains enables them to provide system services in a way that directly supports an object-oriented applications program development environment.

The method to be employed in this research is to complete the emulation, currently underway, of a 64-bit Alpha processor with CPU-resident hardware. In the prototype, DEC Alpha PALCode has been extended to emulate the CPU-resident protection hardware. Misses in the Alpha TLB trigger it. A research prototype single address space operating system will be completed. It will run as a specialized NT process. The various elements of the design will then be subjected to a systematic performance and functional evaluation. This includes examining the characteristics of the additional hardware and whether the protection and resource access data structures and the modularity of the operating system provide ease of programming and improved performance on a single node and, most importantly, in a distributed environment.

The proposed activity is significant to the advancement of knowledge in the operating system and computer architecture fields. It defines the necessary hardware and software design changes needed to make the paradigm shift from process-oriented multiple address space operating systems to single address space operating systems desirable and practical. Combining a 64-bit SASOS kernel and a hardware simulation of additional protection hardware of the type proposed here has not been tried. Features that are not available in conventional process-oriented operating systems or in SASOSs designed to run on stock 64-bit processors are directly supported. The proposed hardware modifications to the CPU are architectural changes - the type that will become increasingly important in the next decade when the rate of CMOS CPU performance improvements based on smaller line and spacing widths is expected to drop. Practical provision of a high performance simplified programming environment over all levels of the memory hierarchy and across all nodes of a LAN will enhance the operation and reduce the development costs of distributed systems. These enhancements apply to client/server systems and workstation clusters for CAD and other multi-database applications, to course-grained distributed parallel processing environments and to soft real-time systems that support multimedia applications and manufacturing.

A Hardware Supported Single Address Space Operating System Prototype

1 INTRODUCTION

The objective of this research is to demonstrate user-friendly programming environment and the high performance that can be provided by a single address space operating system (SASOS) that is supported by CPU-resident protection hardware. Recent large-address space architecture operating systems built on stock 64-bit RISC processors have demonstrated the feasibility of the approach as well as comparable performance to that of contemporary process-oriented operating systems. They have not, however, produced the kinds of gains required to overcome the disadvantages of shifting to this paradigm. A basic assumption of this research project is that additional protection hardware within the CPU is required to produce these kinds of gains. In addition, the advantages must apply both to single nodes and to clustered workstations on a LAN.

It is well known that single address space properties can be utilized to provide a more efficient substrate for data exchange and synchronization between programs, elimination of the file system for persistent storage and support for execution migration across protection domains and machines. These can result in improvements in the programmability and performance of tasks that need to work together and access shared databases. A major premise of this research is that adding protection hardware to a 64-bit RISC processor provides two additional underlying features. First, changes to the hardware enable simple implicit communication and sharing mechanisms to replace inefficient and cumbersome software workarounds. Second, the architecture of the operating system executive itself can be changed to a collection of modular peer-level interacting protection domains. These can interact with user programs as additional peers via ordinary procedure calls and their services modified by run-time user requests. Hardware support for these domains enables them to provide system services in a way that directly supports an object-oriented applications program development environment.

It is our view that the combined CPU hardware/executive software design proposed in this project will make the paradigm shift from process-oriented multiple address space operating systems to single address space operating systems desirable and practical by providing an efficient cost effective distributed object oriented single system view to a small to large cluster of homogenous workstations. Combining a 64-bit SASOS kernel and a hardware emulation of additional protection hardware of the type proposed here has not been tried. Features that are not available in conventional process-oriented operating systems or in SASOSs designed to run on stock 64-bit processors are directly supported. The proposed hardware modifications to the CPU are architectural changes capable of increasing processor functionality and performance without significantly increasing the component count. Practical provision of a high performance simplified programming environment over all levels of the memory hierarchy and across all nodes of a LAN can enhance the operation and reduce the development costs of distributed systems. These enhancements apply to client/server systems and workstation clusters for CAD and other multi-database applications, to course-grained distributed parallel processing environments and to soft real-time systems used in manufacturing and multimedia applications.

This proposal is for funds to continue our on-going research into the structure and design of single address space operating systems. The present state of our project, begun in 1994, is:

- We have determined that, to fully realize the potential of a SASOS, it must be supported by protection hardware contained within the processor and we have a design for this hardware and for a kernel and executive components for a distributed SASOS that would run on it.
- We have obtained a development platform of two Alpha 21164 workstations running Windows NT and have nearly complete implementations of the protection model and of an emulator of a 21164 containing SASOS protection hardware. We reprogrammed the DEC Alpha PALCode to make the entire address space of the DEC Alpha available to our experiments and have written a program to generate SASOS programs for VC++ object and code files and a primitive debugger.
- We have run simulations that indicate that the protection hardware we propose is likely to operate at speeds that will enable it to check access rights within a contemporary RISC processor pipeline stage.
- We have run simulations indicating that the software protection data structures that cache higher level resource access policy and handle protection hardware misses will compete with page tables in process-oriented systems under expected conditions.
- A System Services Interface has been defined.
- Most of the underlying token enabling mechanism to be used for the support of consistency across multiple nodes has been designed.

- We expect that by May 1999, the requested start date for this proposal, we will have completed the emulator and first implementations of the SASOS's key executive and user level modules and run some initial validation experiments.

Our reason for pursuing this research is a firm conviction that operating systems and computer hardware designed to fully utilize the properties of a very large single address space have the potential to perform better than the legacy and research systems currently available, particularly for client/server applications. We believe that performance improvements for these applications can exceed the twofold or greater improvement in system performance [Yu96] that are necessary for microprocessor manufacturers to produce CPUs that are not binary compatible with earlier processors. Even more importantly, we believe that we can also show that fully using the properties of a single address space, via hardware support, enables simplifications to the programming of applications and the middleware and environment subsystems they reside on that significantly reduce development time and cost. These reduced development times apply chiefly to client/server and parallel processing applications in homogenous clustered workstation environments although they also apply to local node programming.

In our view this research falls into a number of operating systems and kernel design areas of interest to the NSF Operating Systems and Compilers Program. The design contains operating system structures for large-address space architectures; supports efficient communication across protection domains and network; provides architectural support for reducing OS overhead and latency in message communication; provides OS level support for distributed shared memory management. It contains a distributed system architecture with support for scalable services, distributed object access and management and fault-tolerance. It also provides system level mechanisms for parallel programming in distributed systems and a single-system image view of a cluster of computers and techniques for fault tolerance. The research also directly addresses issues in parallel and distributed file systems design, parallel I/O systems, load balancing and data migration.

The following section outlines the significance and advantages of a hardware-supported SASOS and the reasons they are not available in current process oriented systems and research SASOSs built on stock 64-bit RISC processors. Section 3 examines the key design features of our SASOS, called Sombrero, and its hardware support, called a Range Protection Look-aside Buffer (RPLB). This background enables a detailed comparison in Section 4 of design features of the two most extensively documented SASOSs that assume use of stock RISC processors with the design proposed here which does not make this assumption. Section 5 discusses related work on hardware-supported SASOSs and CPU-resident protection hardware and the relationship of this research to them. Section 6 is the project plan, including what we have done so far and our research plans for evaluating Sombrero during the project period.

2 MOTIVATION

This research represents a fundamental change in design strategy from that used in process-oriented systems and is significantly different from current SASOS efforts using stock RISC processors and those using hardware support. This impacts all of the OS and kernel design areas of interest to NSF mentioned in section 1. The more technical reasons for this are presented starting in section 2.1.

Designing a SASOS and protection hardware that is able to more fully use the properties of a single address space leads to a general simplification that we believe will demonstrate a 30% to 50% reduction in design complexity as well as a significant increase in performance for the cross domain, cross platform areas listed above. This occurs because an estimated 30% of the code needed for conventional communication between devices and nodes is used to support transitions between different views for disk and RAM and across the network between clients and servers, for example in database systems [Atk*83]. Since a SASOS has only one view, the single VA space, much of this is eliminated.

Object and thread migration become much simpler since thread state and object pointers remain valid on all nodes and many services become distributed by normal caching effects. The Sombrero design researches an improved fundamental platform for general computing in single node and small to large clusters of computers that work together under a single system image in a way similar to a NUMA multiprocessor. The design assumes a homogenous cluster of computers.

Our research is based on the following set of assumptions. Section 6 contains our research plan to verify them.

Multiple views to data such as that used to convert data from files to usable memory objects and back to files increases program size and complexity as well as execution time.

Client server models that must overcome process boundaries using forms of IPC other than shared memory in order to communicate increase program size and complexity.

A distributed single address space model in which pointers and state remain valid when moved from node to node and in which the address space is persistent can provide a reliable model on which to design client/server programs that have the simplicity of programs designed to operate entirely within the memory of a single process.

This simplification will result in code reduction not only at the application level but also at intermediate run-time system levels and to the system services that, for example, provide paged access to the network.

Reduced code requirements will result in better performance from the hardware since fewer instructions are executed.

Simpler programs are cheaper to design and easier to debug. Efficiency and cost reduction for design and programming are the bottom line goals of many of the NSF OS and kernel design areas listed in section 1. It is our belief that they can be better achieved by providing a simpler substrate on which to design rather than by adding incremental changes and complexity to the process model, which was designed to overcome past hardware limitations. The real contribution of this research is that we have designed ways to cache the mechanisms that support a protected single address space in ways that allow a SASOS to perform at the levels one intuitively expects.

There are additional features in the Sombrero model. Sombrero distributes the address space as a set of paged objects (a feature of the RPLB) rather than managing copy-sets for each page. This model is more scalable and decentralized since every object can be serviced independently. The Sombrero design also takes considerable advantage of caching mechanisms that can reduce client/server communication cost for the frequent case to that of a common subroutine call. This is not possible in the process model. Another feature that the single address space affords is that in bulk communications on the network the exact location in a node's memory where data will be stored can always be known in advance. This enables Sombrero to bypass the buffering mechanisms normally found in the IP stack to perform direct page to page copies for more efficient network access. Another Sombrero feature is that the garbage collection problem is not difficult. This is because Sombrero does not use capabilities as independent entities, rather they are held in a protection matrix that allows rescinding resource access at any time. Thus the dangling pointer problem does not exist as it does with protected pointers or capability based protection designs. With respect to reliability, we believe that fault tolerance is no greater a problem for a hardware-supported SASOS than for most computer systems. For example for Sombrero all of the storage devices beginning at the CPU, 1st level cache, 2nd level cache, RAM and the disk/network storage can be regarded as levels of cache. By adding a fault tolerant level of cache after the disk/network level, important copies of consistent data sets can be copied with intermediate transactions logged just as is done commonly now in logged file systems such as NT's NSFS.

If the Sombrero research project gives the appearance of a much greater challenge or that it requires more resources than are available to us were this proposal funded, this is because we are working on a single core change rather than many independent incremental changes as is normally done. The research plan does not include a thorough investigation of all aspects of our SASOS design by the end of funding. However it includes enough experiments and validations to fundamentally demonstrate whether the design offers a credible strategy that better addresses many of the operating system and kernel design areas of interest to the NSF Operating Systems and Compilers Program.

2.1 Single address spaces and single address space operating systems

The recent advent of 64-bit RISC processors such as the DEC Alpha 21164 and MIPS have made much greater virtual address spaces available to applications and the operating system though none of them as yet provide a full 64 bit address space. This has led to a reexamination of the premises of the conventional process-oriented operating system model in which 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 issue is how to structure an operating system so that it supports a simple programming model and high performance in a computer system that supports a very large address space where address space conservation is no longer a driving issue. One such model consists of a *single* network-wide address space in which all code and data reside, and an operating system designed specifically to run in this environment.

2.2 Characteristics of a single address space

Single address spaces have a different set of properties than that available in the multiple address spaces supported by conventional process-oriented operating systems such as UNIX [Vah96] and NT [Cus92]. 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. 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 virtual addresses.

3. The virtual address 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 where policy is cached within the CPU.

2.3 Advantages that can be provided by a SASOS

- Using the attributes of a single address space a SASOS can be designed to provide the following key features:
- Address translations remain the same for all programs.
- Threads free to travel throughout the virtual address space with no changes in the environment in which they are running except for protection context.
- Network-wide Communication using shared memory that requires no prior or additional setup. It becomes a transparent program feature.
- Internal pointers and pointers into other objects that remain the same across all levels of storage and between all programs - marshalling, flattening and dynamic linking are not needed.
- Persistence without use of a separate file system.
- Protection by restricting what a computation is allowed to access rather than what it is allowed to address. This essentially reduces managing IPC to managing protection.

2.4 Essence of the paradigm shift

A paradigm has been defined as a "constellation of beliefs, values, techniques and so on shared by the members of a given community that implicitly define the legitimate problems and methods of a research field for succeeding generations of practitioners" [Kuh96]. A real, important and often not fully understood paradigm shift occurs when going from the multiple virtual address spaces of process-oriented systems to the single address space of SASOSs. This is brought about by the fact that in a single address space the set of virtual addresses (VAs) can be used as a system-wide namespace. A SASOS names all of its resources using a *single* namespace where the names are unique VAs. Each VA is permanently bound to a byte of a data object. Combining this with the fact that the VA namespace is directly operated upon by the CPU means that all computations can use the system hardware to *directly* implement access policy on network-wide resources. All objects can be identified, located and accessed using the same VA name wherever they are and whenever they are used. So the paradigm shift is:

Going from multiple namespaces in which object names can vary and are determined by the logical structures they are associated with, e.g., process, file or record, or by the physical container that holds them, e.g., cache, RAM or disk

Going to a single network-wide namespace in which object names are invariant and are determined by the unique virtual addresses used to access them

2.5 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 principals. Migration of execution across protection domains or machine boundaries without an address space or thread context switch is essentially impossible. 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.

2.6 SASOSs with and without changed protection hardware support

Most contemporary SASOSs have been designed to run on stock RISC 64-bit processors. These include Opal [Cha*94], Mungi [He*98], 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 AS/400 [Sol96], Monads and Sombrero [SkMi98b]. Hardware support enables or significantly facilitates:

- Orthogonality between obtaining system services and entry into privileged supervisor mode
- A flat modular system architecture - OS services, environment servers and user programs accessible to each other via ordinary procedure calls
- Direct support for OOD and OOP - an object class can be implemented directly as a protection domain and a server as an instantiation of the class; base classes are extendable via user overrides.
- Object granularity independent of page granularity
- Simple scaleable per-object copy set replication and consistency in a distributed system
- Access authorization, resource accounting and name services outside of the executive
- Clean separation of OS support for protection implementation and user definition of protection policy

2.7 Motivation for building a HW-Supported SASOS

The increasing size and complexity of modern software, has led to increased software engineering efforts to reduce the costs of program development. A HW-supported SASOS provides simpler mechanisms for the support of sharing and communication between large programs and databases. These include common naming mechanisms and system level support for Object Oriented Design and Programming. Application areas that can benefit are Object-Oriented DDBMSs, multiple database/multiple programming model applications (e.g., CAD/CAM) and distributed parallel computations. Soft real-time systems used to support multi-media applications and in manufacturing applications can benefit significantly from the increased predictability of reduced delay times and reduced communication costs.

Changing tradeoffs resulting from new and improved technologies also favor HW-supported SASOS usage. These include the increasing disparity between CPU and RAM cycle times in combination with the feasibility of large virtual first and second level on-chip caches, and the decreasing ratio of inter-node access time to local disk access time combined with the ease of distributing the VA space. In addition, it is generally agreed, for a number of reasons, that within about 10 years performance increases derived from reducing chip feature sizes will begin to decline [Pou98]. Increased performance will have to come from architectural changes such as shifting to single address space operation.

Hardware support enables implicit protection domain crossing at all levels, simplifying programming and increasing performance. Changes to protection hardware require changes to current processor architectures and system software, pre-existing software will not be able to take full advantage of the SASOS architecture and new software may not run on the old hardware.

3 DESIGN OF A HARDWARE SUPPORTED SASOS

This section presents our design of a hardware-supported SASOS. It is designed to make fuller use of the potentials of a single address space. The operating system is called Sombrero and the CPU-resident protection hardware is called a Range Protection Lookaside Buffer (RPLB). The design is targeted at single nodes and distributed systems consisting of homogenous workstations.

3.1 Model

The basic abstractions in Sombrero 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.

3.2 Hardware Design

As explained in section 2.6, 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, the RPLB.

3.2.1 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:

<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 1. This functionality is a considerable extension of that provided by the PLB presented in [Ko*92].

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 1. Hardware Functionality implemented by the RPLB.

3.2.2 Logical Design

Figures 1 and 2 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 1.

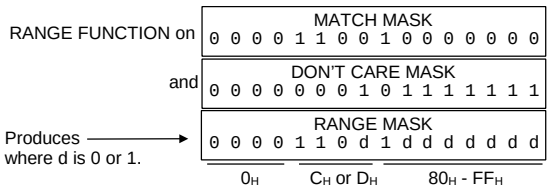


Figure 1. RPLB Range Mask defining a region consisting of virtual addresses from $0C80_H$-$0CFF_H$ and $0D80_H$-$0DFF_H$.

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 2.

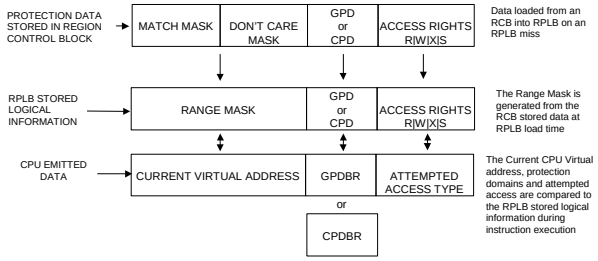


Figure 2. RPLB Intra-domain Operation.

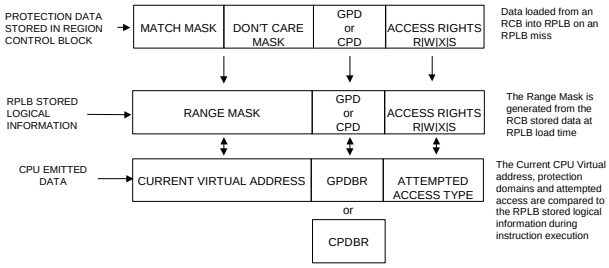


Figure 3. RPLB Cross-domain Operation.

An RPLB entry for part or all of a region is shown at the top of Figure 2. 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 3 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 [Sko94] [SkMi96a] [SkMi97a].

3.2.3 RPLB VLSI Synthesis

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 [Ga*95]. 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 [Tor97]. The objective was to design and synthesize a block able to provide memory protection within the time constraints of the DEC Alpha pipeline, and to do so in a 500 MHz, 3.3/2.5 V, 0.35 µm circuit. The synthesized RPLB (Figure 1) consists of two major blocks: the Read/Write interface through which entries are written and a K-entry (K=8 in this synthesis) FIFO buffer which also performs the matching function and implements a least-recently-used (LRU) replacement algorithm. Additionally, there is a latch UPDOut that stores the output of the FIFO when a valid domain switch occurs.

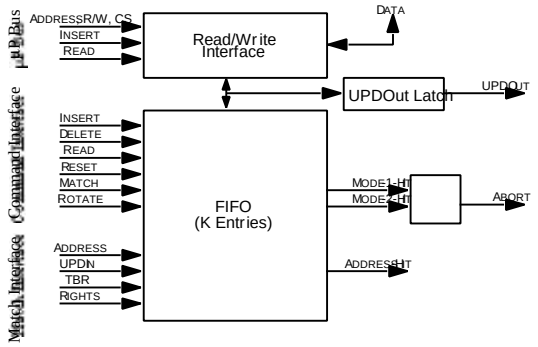


Figure 4. RPLB architecture synthesized with Synopsis Design Compiler using 0.35-micron library.

Eighteen different synthesis recipes were run on a single FIFO location to determine worst case single location size and timing and a number of runs were made to determine timing for the entire RPLB. Worst FIFO area observed was approximately 1.27 mm2 as compared to 208 mm2 for an Alpha 21164 manufactured to a 0.35 micron process indicating that an RPLB for k=64 would probably not increase Alpha chip size significantly. Timing analysis indicates a worst case speed of operation of the synthesized RPLB of 120 MHz for the entire FIFO and 250 MHz for a single FIFO location. Although this does not meet the timing requirements of a state-of-the-art 500 MHz Alpha microprocessor, there are architectural and technological reasons to believe that the synthesis indicates inclusion of an RPLB within a RISC CPU is well within the capabilities of today's silicon fabrication technology. The most important of these were that the synthesis used an LRU replacement algorithm as contrasted to the Not Last Used algorithm used by DEC in the Alpha TLB [Dig97]; a generic RPLB was synthesized rather than three separate RPLBs designed specifically for protection of instruction; data and domain switching accesses, and CAD tools, library elements and custom techniques available to DEC [Bo*95] were not available to the synthesis. More details can be found in [Tor97].

3.2.4 Hardware Emulation

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 [SkMi98a].

3.3 Software Protection Data Structures

3.3.1 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. 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.

3.3.2 Simulation

Performance of the Sombrero protection model has been evaluated by simulating the average cost of memory access in Sombrero and in a typical, process-oriented operating system, Alpha-NT. Cache access traces from the Internet were used to drive the simulated hardware caches and software data structures, the Sombrero RPLB and PRALs and the Alpha TLB and Alpha-NT page tables. Performance statistics including cache miss rates and miss rate penalties versus the number of protection domains were collected. By necessity, the numbers obtained were approximate, but are useful to observe and characterize behavior differences between the two systems. Simulation results showed that while the RPLB miss penalty is likely to be twice as high as the TLB miss penalty, the RPLB miss rate is likely to be lower than the TLB miss rate under many common scenarios. This is true, for instance, when the memory objects accessed are likely to be larger than a page. In these situations performance, as measured by the product of miss rate and miss penalty appears to be roughly comparable with the RPLB performing better as the average accessed memory object becomes larger. This offers a solution for the page table problems associated with larger address spaces. Details on the PRAL simulation can be found in [Kha97].

3.4 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. Figure 6, presents two simplified[1] sketches of our early view to Sombrero's structure including the modular executive and its relationship to the hardware and the interface provided by the Sombrero executive to supported environments, the Sombrero Services Interface (SSI). The Sombrero executive supports explicit definition of user protection policy via service requests and implicit implementation of protection via protection faults.

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.

The flat rather than hierarchical and layered structure for all modules, including those within and those outside of the executive services allows in-calls and out-calls implemented by procedure calls to replace system calls and up-calls.

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 onecontrolled 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 6 to be provided entirely by user servers. User services include authorization, user identification and 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

[1] The figure does not show 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.

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 [SkMi98a] and [SkMi99] for a more extended treatment.

The executive supports authorization for protection policy and system service requests without maintaining any intermediate state for the Sombrero Service clients in order to protect its data structures from corruption. It is able to do this by exploiting the mechanism used in performing cross-domain control transfers to and from passive code in a hardware-protected single address space[2]

In addition, trusted user servers such as a fileserver can safely access protected objects for different principals by combining the hardware protection support for separate calling thread protection domains (that of the principals) and a trusted general protection domain (fileserver code)[3].

Our current design is an updating to the design described above. It 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 7, 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.

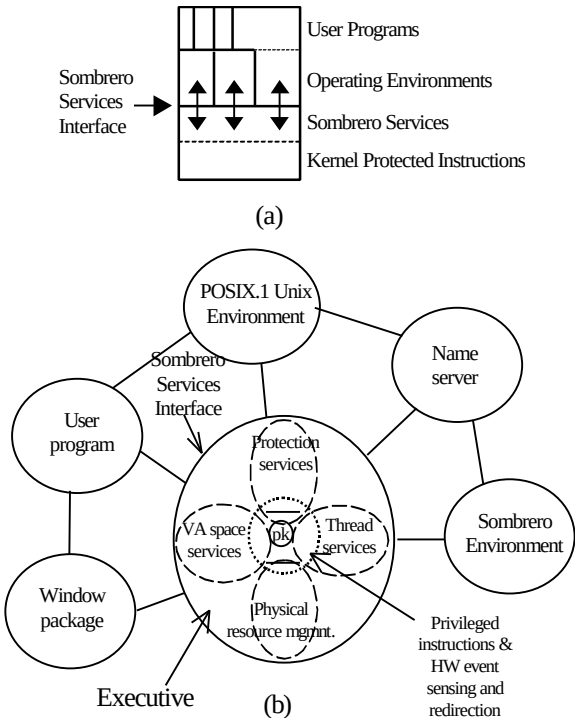


Figure 6. Sombrero structure including executive, Sombrero Services Interface, operating system environments, user programs, kernel mode protected operations and pico-kernel.
(a) Traditional layered view.
(b) Peer-level domain view.

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

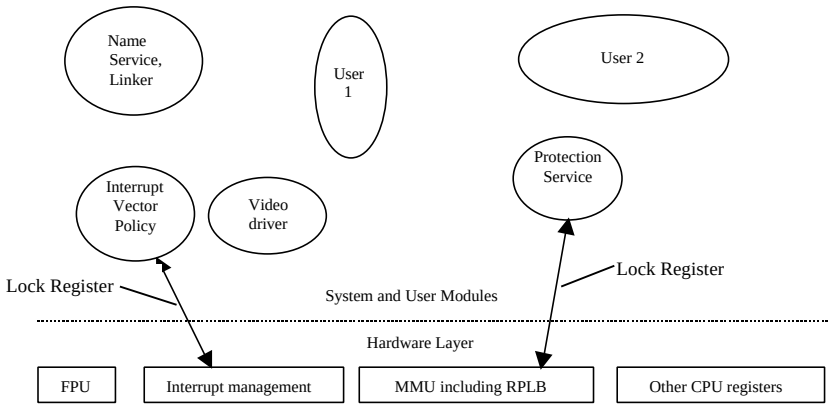


Figure 7. Sketch of Completely Peer-Level Single Address Space Operating System Structure.

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.

[2] This is because there is no memory map change in a single address space and no protection domain change for thread-private data in Sombrero because of its hardware support for a thread related protection domain.

[3] This is possible because the called code determines how the caller's data is handled and where it is stored. It can be implemented by loads and stores in code common to the two protection domains and data objects accessible independently to the two protection domains.

3.4.1 Sombrero Services

The interface to Sombrero Services (SSI) is a simple object-oriented interface that provides a standard set of methods to access the fundamental Sombrero Services. The design makes most of the full power of the underlying model available while making it unnecessary to be familiar with the arcane details of the implementation. The SSI, shown in Figure 6, defines five classes for which there are object instantiations. These are for memory objects, threads, protection domains, principals (principal objects represent owning entities, users, or system services) and nodes (node objects represent physical computing resources). There is also a common super-class to support authorization mechanisms. Sombrero Services convert user service requests to insertion/deletion operations on the PRALs and searching/updating of the contents of PRAL control blocks. Also included is a stub/gateway function mechanism used to transfer control to user method overrides such as those in authorizing agents on the occurrence of events such as a PRAL miss during resource access following an RPLB fault.

While any program can use it, a more likely scenario is that the SSI will be used by a higher level interface that gives more familiar facilities to its users, for example current legacy operating system environments such as Windows NT Win32 [Ric95] or UNIX POSIX.1 [IEE90] as shown in Figure 6. These environments could add SASOS features to their facilities as they chose. SSI call functionality includes that of Opal [Cha95] and SSI calls are sufficiently complete to implement the functionality provided by Solaris 2.4 [Sun94] system calls. The SSI has been described in the form of C++ class definitions with a C interface to the underlying structures. The details are contained in [Car96].

3.5 Distributed System Design

All SASOSs provide the potential for increased utilization of network resources. Most of these advantages derive simply from distributing the advantages of a persistent global namespace over a LAN. These include client/server performance increases from using passive servers, more efficient DSM from reduced mapping operations and improvements to distributed parallel computation that derive from increased thread mobility. Supporting a SASOS with changed CPU protection hardware provides network-wide advantages to cluster computing beyond those available to SASOSs that run on stock processors. This section describes the distributed system design of Sombrero and the improvements to program development environment and performance made possible by, or greatly facilitated by, its additional CPU-based protection hardware.

3.5.1 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 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 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 executive service as indicated in Section 3.4. The migration and replication of kernel objects using surrogates make the network’s multiple computers appear as a single virtual multiprocessor. Additional information on Sombrero distributed system design can be found in [SkMi97] and [Sko94].

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 2. Sombrero Distributed System Features

Table 2 lists the Sombrero Distributed System Features.

3.5.2 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 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. 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 required at a node need be present at it. More information can be found in [SkMi99b].

3.5.3 Distributed Parallel Computing

Applications that perform course-grained parallel computation over a client/server system benefit from SASOSs with additional CPU protection hardware and the above-mentioned support for user-extensible semantics policy managers. One application of this is support for distributed parallel processing environments such as that provided in the Calypso research project [Das97]. Separate thread carrier protection domains directly support the kind of thread cloning needed in Concurrent Read Exclusive Write (CREW) semantics. These semantics, essentially a form of weak consistency semantics with a single synchronizing variable, are very useful in an environment where multiple nodes act as CPU servers for a single client node. CREW semantics allow concurrent multiple writes to the same and to different parts of the same memory objects on different nodes, controlled by global synchronization primitives and 'smart' application programmers [Sar97], [Mc*97].

3.5.4 Client/Server and other Clustered Workstation 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 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 makes possible service migration and replication via caching that reduces the likelihood of a server becoming a bottleneck and also provides 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.

3.5.5 Distributed System Fault Tolerance

In a single address space every level of storage on every node can be viewed as a cache for the virtual address space. Sombrero takes advantage of this cache hierarchy to place an archival fault tolerant layer at the bottom of the hierarchy. This layer is used for check-pointing portions of the virtual address database that are in a consistent state. Fault tolerance in a single address space is no harder than (and no easier than) that in process-oriented systems with a separate file system namespace for persistent storage. Similar logging and private workspace mechanisms apply.

4 EFFECTS OF PROTECTION HARDWARE ON SASOS FEATURES

Protection, resource access and control transfer are the underlying issues that must be dealt with in the design of a communication mechanism, This section examines these issues, contrasting approaches contrasting approaches in SASOSs built on stock RISC processors and Sombrero.

4.1 Protection

Protection, whether for resource access or control transfer is always defined by a logical triple:

<accessing protection domain, accessed resource, type of access>

These triples can be managed by access control list and/or capability list mechanisms. Process-oriented systems bind protection to pages with page tables whose entries can be cached in a TLB. The page tables and TLB define the per-process page translation/protection map in effect at a given time. Stock processors are designed to support this operation. Every SASOS built on a conventional processor is forced to use these protection mechanisms. Protection hardware specifically designed for use by a SASOS can bind protection directly to data objects by protecting the VAs that comprise them. It can also define a second protection domain associated with a computation's thread and distinct from the environment it is executing in. The next two sections present in greater detail the resource access and control transfer mechanisms available in Sombrero to support communication in a cluster of computers. Where appropriate, we compare these against mechanisms that perform the same functions in Opal [Cha95] and Mungi [He*98], which run on conventional 64-bit RISC processors. The key point is that changed CPU protection hardware provides resource access and control transfer directly without requiring additional software protocols. These protocols complicate programming and reduce the transparency of distributed system communication.

4.2 Resource Access

In Sombrero, resources are named and access is controlled by co-located protection triples contained in the PRAL and cached in the RPLB. In SASOSs without additional protection hardware resources are named and password capabilities ultimately backed by page tables and a TLB provide access control. Capabilities are used to explicitly attach segments or to perform specific operations on objects. The capabilities can be presented implicitly or via a mechanism using published capabilities that enables segments to be attached transparently on address faults reducing the need for most applications to deal with them. However, presenting capabilities requires a software protocol at runtime. This protocol always involves another namespace, in which the resource owner can validate the capabilities presented by the accessor. This is what the passwords are used for. They are essentially a protected sparse namespace used to corroborate protection triples. No matter how they are implemented, these software protocols add program complexity and overhead at one or more levels. Co-located CPU-resident protection triples, such as those used in Sombrero, can be used directly as pre-authorized capabilities improving performance and simplifying programming. Software protocols always involve serial operations and additional instructions. Access control using an RPLB can be done in "parallel", i.e., the required runtime protection check is made during the execution of the instruction performing the memory access. No additional instructions are required. In a distributed single address space, programming and performance benefits in resource access extend across the entire cluster, simplifying remote data sharing and data migration.

4.3 Control Transfer

Software protocols are also used for control transfer in SASOSs built on stock RISC processors. To protect a calling client's data on a domain crossing, the capability lists used for resource access are grouped in a different way to change a client's protection domain so that it includes objects accessible via a capability list in the server domain plus a selected subset of the objects accessible in the client's domain. These are called protection domain extensions (PDXs). PortalIDs are also used to control transfers into another protection domain with capability presentation and protection operations performed serially by objects called proxies on the calling side and guards on the called side. By caching allowed protection domain crossings in hardware as is done in Sombrero it is possible to remove almost all the run time overhead associated with protection domain crossings. The 45-121 machine cycles on a RISC platform for well-designed one-way IPC [Lie95], [Li*97] can be reduced to close to the *one* machine cycle required for a procedure call. CPU hardware defining a separate CPD allows data copying based on trust relationships to be performed independently of domain switches. A new CPD associated with the thread containing copies of arguments and data sets needed by the called program that require protection can be created, followed by an ordinary procedure call to effect the general protection domain switch. Copies need not be made if protection is not required. If they are required, given that the carrier protection domain can be reused later, the number of setups is sharply reduced.[4] This achieves in hardware what a PDX achieves in software. The RPLB removes the need for portalIDs (an additional namespace) used in Opal because the cache of VAs at which domain crossing is permitted provides protection for individual entry point addresses. When protection is page-based, only the address of the page can be used this way. A separate hardware-supported protection domain map for the private data of a thread of execution, as provided in Sombrero, replaces the simplifying assumption of orthogonality of execution and protection that underlies the proxy/guard/portalID mechanism, and argument passing at domain crossings can be faster and simpler. Capability-based systems essentially build workarounds on top of the pre-existing substrate in order to make it suitable for use in a single address space. These factors always reduce performance and complicate the programming environment, increasing program development costs. CPU protection hardware oriented towards operation in a single address space *changes* the underlying substrate to make it more suitable for use in this

[4] It is roughly proportional to the number of traveling threads rather than the product of traveling threads and modules entered.

environment. Simplification of the programming environment and performance improvements for control transfer under a distributed SASOS extend across a network of clustered computers simplifying task migration and service access.

5 RELATED WORK ON HARDWARE-SUPPORTED SASOSs AND CPU-RESIDENT PROTECTION HARDWARE

Two SASOSs that include CPU hardware to provide protection, AS/400 and MONADS, have been implemented in recent times. In addition we briefly discuss three CPU-resident protection mechanism designs. One, the PA-RISC architecture has been implemented and is in widespread use. The other two, are a PLB and Guarded pointers. The PLB has not been implemented and Guarded Pointers is currently under implementation.

5.1 AS/400

The IBM AS/400 [Sol96] 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/400 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 AS/400 systems in commercial use. The system architecture is targeted at database management-type usage. The AS/400 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 AS/400 has additional instructions for tag management and other purposes. OS/400 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 AS/400 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 AS/400 is a very successful industrial strength product, demonstrating the feasibility of a SASOS in its targeted application area.

5.2 MONADS

MONADS [Ros92] [Ro*85] is a research SASOS now only being actively pursued at Ulm Universitaat in Germany. Prior to that it was developed at three universities in Australia, most recently, the University of Sydney. It has extensive special purpose hardware support including a segmentation mechanism that supports capabilities. As callers must present a capability to cross domains that needs to be validated for the type of access and converted to an entry point, it does not appear that a single instruction suffices for a domain switch. MONADS' hardware, however, is streamlined to support fast implicit capability-based domain switching. MONADS served as a substrate for persistent object management system design. MONADS supports a network wide single level store and a flat 60-bit paged single virtual address. The MONADS hardware predates modern 64-bit RISC technology. Researchers at the University of Sydney are currently working on the Grasshopper orthogonal persistent object operating system, which is targeted to run on stock 64-bit RISC processors [De*94]

5.3 CPU-Resident Protection Hardware

HP's PA-RISC architecture [Lee89] [Wi*92] provides hardware support for protection in a single address space. Access rights are tied to pages and their virtual addresses, however, making it difficult and sometimes impossible to concurrently assign different access rights for the same pages to multiple protection domains. The Sombrero RPLB extends the functionality of the PLB [Ko*92] by caching allowable domain crossings, caching thread state independently and enabling object-based protection granularity. Guarded pointers [Ca*94] are a hardware technique to implement capability-based addressing. Guarded pointers, although overcoming much of the speed disadvantages associated with capabilities, ultimately support a segmented address space and do not provide the generality of a flat wide address space.

6 RESEARCH PLAN

The research into the protection and distribution of the SASOS proposed in the previous sections has been underway since 1994. To date there are designs and nearly complete implementations for the protection model, hardware RPLB, PRAL lists and data structures, the system services interface, a program to generate Sombrero programs from VC++ object and code files and a primitive debugger. By reprogramming the DEC Alpha PALCode we have made the entire address space of the DEC Alpha available to our experiments. A hardware design and simulation of the RPLB and a simulation of RPLB miss handling have been completed with encouraging results. Simulation of the

distributed features of Sombrero is underway. There have also been 3 refereed conference publications. Between now and August 2001, the ASU 64-bit OS research group which currently includes 1 Ph.D. candidate, 4 MS students and one faculty member will complete the implementation of the Sombrero operating system as an extension of the Windows NT 4.0 operating system using hardware emulated in DEC Alpha PALCode. The research group will characterize the features of Sombrero regarding network wide and single node performance improvements and simplification of the client/server design model.

Our research plan includes completing the research prototype implementation and program design and experiments to validate our premises that a SASOS supported by additional CPU-resident hardware will produce an easier to use programming model and higher performance valuable to extending client/server and parallel processing models to a network cluster. The program design and experiments will be targeted at:

1. Verifying our simulation results indicating that the RPLB will perform object level protection at a performance cost as low or lower than a TLB under a real load. Our simulation results to date indicate that as the average object accessed becomes much larger than a page the RPLB should perform far better than a TLB. The translation buffer will be assumed to reside on the main memory bus as a hardware-supported CAM since for a SASOS only a single inverted page table is required for each node. The performance measurements will be obtained by measuring the cost to emulate the hardware (the CPU provides these metrics) and subtracting these from actual performance measurements. These will be compared with similar programs written for Windows NT.

2. Verifying that our protection domains operate as expected and that the switch mechanism operates in a cost-effective manner by measuring client/server access with and without the active switch mechanism. There should be some overhead for infrequently accessed services but the frequent case should show little measurable overhead. The carrier protection domain will be tested to determine its role in reducing the proliferation of inter-domain switches and establishing client/server trust relationships.

3. Measuring the growth and management of the PRAL data and investigating ways to make it more efficient. Experiments will be run to determine ways to make it resilient from damage due to system failures and to make it efficient to search on RPLB misses.

4. Determining if Sombrero programs are usually much simpler to design than programs requiring a file service and IPC techniques and if these programs actually perform better as well. Various kinds of client/server programs to run both in Sombrero and in Windows NT will be designed. Unless performance results are greater than 10% it is possible that other kinds of overhead (such as excessive paging) will mask any improvements. Savings of program development time of 10% will however be considered very significant.

5. If time permits, evaluation of compiler techniques to take advantage of transparent protection domain switching between clients and servers. Also exploration of the management of various levels of client/server trust relationships and how compilers can be employed to implement them. We have already done some of this with good results using the assembler source output from the VC++ compiler and the unlinked object files in COFF format.

6. Exploration of the effects of distributing the client/server programs designed in step 4 on the various Sombrero network distribution algorithms. Network latency effects on performance improvements will be measured. Based on these experiments, we will also make projections on the scalability of these network algorithms using on the cost of the distribution of the needed data structures. Scalability considerations include memory consumption and additional latency.

7. Exploration of operating system and application design by refining system services. Since Sombrero service models and applications programs are classical instantiated objects, ways to provide extended features to already existing services by using them as base classes for extended services will be investigated.

8. Investigation of consistency management schemes for updating distributed objects.

9. Examination of the current Sombrero strategy targeted at overcoming the false sharing problem for large distributed memory objects during concurrent access.

10. Examining data migration and load balancing by modeling parallel processing algorithms used in the Calypso project [**] at ASU and comparing performance with that of similar results obtained from Calypso programs run on NT workstations.

11. Determining the overall resiliency of the network and the effect of failures. We will investigate the fault tolerance mechanism describe in section 3.5.5.

By designing a whole operating environment we can produce interactions not available to researches of more limited scope built on less suitable platforms. It is necessary to use NT to run the Sombrero pager and provide I/O to limit the work to specific interests. Expected completion of items 1-4 or 5 is the 1999-2000 academic year and parts 6-11

by August 2001. The CSE Department of ASU was selected by Microsoft as one of a group of university departments to have access to the Windows NT source code and supporting binaries for research purposes. The NT 4.0 source code is on hand from Microsoft. The proposed research is the primary research project of the principal investigator, Donald Miller, and the dissertation topic of Alan Skousen – which is currently in progress.

References Cited

[Atk*83] Atkinson, M. P., Bailey, P., Chisholm, K. J., Cockshott, W. P., and Morrison, R., "An Approach to Persistent Programming", *The Computer Journal*, (26) (4), pp. 360-365, Nov. 1983.

[Bo*95] Bowhill, W., Bell, S., Cooper, E., et al, Circuit Implementation of a 300-MHz 64-bit Second-generation, CMOS Alpha CPU", *Digital Technical Journal*, Volume 7, Number 1, Special Edition, 1995.

[Car96] Carnes, M., "Sombrero System Interface", MCS Project, Computer Science and Engineering Department, Arizona State University, December 1996.

[Ca*94] Carter, N., Keckler, .S. and Dally, W., "Hardware Support for fast Capability-Based Addressing", *Proc. of Sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VI)*, October 1994, pp. 319-327.

[Ch*94] 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.

[Cha95] 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).

[Cus92] Custer, H., *Inside Windows NT*, Microsoft Press, 1992.

[Das97] Dasgupta, P., "Parallel Processing with Windows NT Networks", *Proceedings of the USENIX Windows NT Workshop*, August 1997, pp. 113-122.

[Dig97] Digital Equipment Corporation, *Digital Semiconductor 21164 Alpha Microprocessor Hardware Reference Manual*, Order Number: EC-QP99B-TE, January 1997.

[De*94] "Grasshopper: An Orthogonally Persistent Operating System", Dearle, A., di Bona, R., Farrow, J., Henskens, F., Lindström, A., Rosenberg, J. and Vaughn, F., *Computing Systems*, 7(3), pp. 289-312, Summer 94.

[Ga*95] Ganesan, R., Sukavanam, V., and Torla, M., "Range Protection Lookaside Buffer, VHDL Synthesis Report", Term Report CSE 518 VLSI Synthesis using HDLs, December 1995.

[He*98] 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.

[IEE90] Institute of Electrical and Electronic Engineers, *Information Technology-Portable Operating System Interface (POSIX) Part 1: System Application Program Interface (API) [C language]*, 1003.1-1990, IEEE, December 1990.

[Kha97] Khatri, S., "Protection Structures in Sombrero, A Very Large, Single Address Space, Distributed Operating System", MS Thesis, November 1997.

[Ko*92] 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.

[Kuh96] *The Structure of Scientific Revolutions, 3rd ed.*, Kuhnes, T., University of Chicago Press, 1996.

[Lee89] Lee, R. B. "Precision Architecture" *IEEE Computer*, pages 78-91, January 1989.

[Lie95]Liedtke, J., "On Micro-Kernel Construction", *Proceedings of the 15th SOSP, Operating Systems Review*, Vol. 29, No. 5, December 1995, pp. 191-200.

[Li*96] Liedtke, J., Elphinstone, K., Schönberg, S., Härtig, H., Heiser, G., Islam, N. and Jaeger, T., "Achieved IPC Performance (Still the Foundation for Extensibility)", *Sixth Workshop on Hot Topics in Operating Systems*, May 1997.

[Mc*97] 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.

[Pou98] Pountain, D., "Amending Moore's Law", *Byte*, March 1998.

Ric95] Richter, J., *Advanced Windows, The Developer's Guide to the Win32 API for Windows NT 3.5 and Windows 95*, Microsoft Press, 1995.

[Ros92] Rosenberg, J., "Architectural and Operating System Support for Orthogonal Persistence", *Computing Systems*, Vol. 5, No. 3, summer 1992, pp. 305-335.

[Ro*85] 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.

[Sar97]Sardesai, S., "Chime: A Versatile Distributed Parallel Processing System", Department of Computer Science and Engineering, Arizona State University, *Ph.D. Dissertation*, May 1997.

[Sk*96] Skousen, A. C. Miller, D. S. and Feigen, R. G. "The Sombrero Operating System An Operating System For A Distributed Single Very Large Address Space General Introduction", *ASU 64-bit OS Group WP 0*, Computer Science and Engineering Department, Arizona State University, TR-96-005, April 1996.

[SkMi96a]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.

[SkMi96b]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.

[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.

[SkMi98a]Skousen, A and Miller D., "The Sombrero Distributed Single Address Space Operating System Project", *Second USENIX Windows NT Symposium*, August 1998, p. 168.

[SkMi98b] 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.

[SkMi99a] 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.

[SkMi99b] Skousen, A. and Miller, D., "Using a Single Address Space Operating System for Distributed Computing and High Performance", *18th IEEE International Performance, Computing, and Communications Conference (IPCCC'99) Proceedings*, February 1999. (submitted).

[Sko94] 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.

[Sol96] Soltis, F. G., *Inside the AS/400*, Duke Press, 1995.

[Sun94] Sun Microsystems Inc., *Solaris 2.4 Manual Pages*, 1994.

[Tor97] Torla, M. J., "RPLB Design Project Report", MCS Project, Computer Science and Engineering Department, Arizona State University, July 1997.

[Vah96] Vahalia, U., *UNIX Internals The New Frontier*, Prentice-Hall, 1996.

[Wi*92] Wilkes, J. and Sears, B., "A comparison of Protection Lookaside Buffers and the PA-RISC protection architecture", Computer Systems Laboratory, Hewlett-Packard Laboratories, *Technical Report HPL-92-55*, March 1992.