Operating System Structure and Processor Architecture for a Large Distributed Single Address Space

ALAN SKOUSEN DONALD MILLER
Computer Science and Engineering Department
Arizona State University, Tempe, Arizona 85287, USA
E-mail: {alan.skousen,donald.miller}@asu.edu WWW: http://www.eas.asu.edu/~sasos

ABSTRACT

Recent 64-bit microprocessors have made a huge 18.4 quintillion byte address space potentially available to applications and the operating system. Because current process-oriented operating systems and their underlying page-based protection model were designed to conserve virtual addresses by reusing them, this four billion-fold increase in address space has led to investigations of alternative models. The fundamental issue is how to structure an operating system so that it provides a simple program development environment 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 approach is to establish a *single* network-wide address space in which all code and data reside, and then design an operating system specifically to run in this environment. This paper presents a model and the design of a Single Address Space Operating System (SASOS) called Sombrero. It includes a design of proposed changes to CPU protection and memory management architecture to support its abstractions.

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 question is whether another model might supply an improved program development environment and increased performance. 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.

1.1 Characteristics of a single address space

In a single address space there is one persistent network-wide set of virtual addresses that can be used to uniquely name every data object. 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.

1.2 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.

1.3 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". A 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 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 determined by the unique virtual addresses used to access them

1.4 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.

1.5 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 [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 AS/400 [3], Monads and Sombrero [4]. 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

1.6 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. Real-Time systems can benefit from reduced and more predictable program communication times.

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 [5]. 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 and pre-existing software will not be able to take full advantage of the SASOS architecture.

2 SOMBRERO DESIGN OVERVIEW

2.1 Basic Abstractions

Table 1 lists the basic Sombrero abstractions.

2.2 Hardware Design Functional Requirements

To evaluate whether every memory access can be allowed, protection hardware must be CPU-resident. Sombrero includes a design of this hardware called a Range Protection Lookaside Buffer (RPLB). The design

Memory Object º Characterized by *Range* - a Distinct Variable Sized Contiguous Allocation of the Virtual Address Space - contains *Memory Regions*, sets of Virtual Addresses within a Range with Non-Overlapping Protection Attributes.

Protection Domain º Set of Memory Objects and Protection Domains Reachable by a Thread executing within it and their Associated Access Permissions. There are General and Carrier Protection Domains (GPDs and CPDs).

GPD: Code, Data and other GPDs accessible to all threads within it
CPD: Data and GPDs privately accessible to an Individual Thread

Thread º State of a Computation Represented by Current or most Relevant CPU Register State (PC, SP, General Registers, etc.), Reachable Carrier Protection Domain Memory, and Status Information.

Principal º Unique Protected Identity that Represents a User or a System Service - binds Resources and Activities to an Owning User.

Table 1. Sombrero Basic Abstractions

is targeted at single nodes and distributed systems consisting of homogenous workstations

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's 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. 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 Sombrero RPLB is a protection cache that provides these features. The hardware functionality provided by the Sombrero RPLB is summarized in Table 2.

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 Useful for the Support of a Single Address Space Operating System

The RPLB design has been synthesized and its performance evaluated using Mentor, Synopsys and Cascade VHDL tools. Performance has been shown to be feasible relative to 500MHz Alpha 21164 timings. Details of this synthesis can be found in [6]. The complete logical design of the Sombrero RPLB is given in [7].

2.3 Software Protection Data Structures Design

Kernel-resident data structures, cache the data needed to convert between user-level policy information in the form of resources, access rights and users, e.g., files, {rwx} privileges, user ids and resource ownership, and the CPU-level representation of these things in terms of protection domains, virtual addresses and access privileges. These data structures called Protection and Resource Access Lists (PRALs) logically contain two types of simple lists interlinked to provide fast resource access and efficient rights revocation. 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 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. A more detailed description of the PRAL design can be found in [7]. A software simulation of the PRAL and a comparison of Sombrero RPLB + PRAL vs. Alpha NT TLB + Page Tables performance is presented in [8].

2.4 Operating System Structure

Figure 1 presents a simplified view of 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. Executive modules are instantiations of object classes that are implemented as Sombrero protection domains. Their methods can be extended and overriden by user modules. 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. There is a

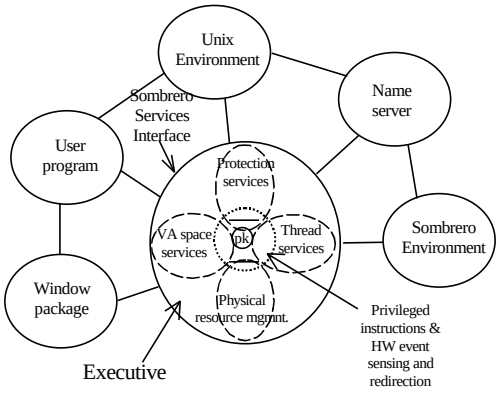


Figure 1. Sombrero System structure.

flat rather than hierarchical and layered structure for all modules, including those within and those outside of the executive services. In-calls and out-calls 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 one controlled by the kernel/user bit) is orthogonal to protection domain definition, although typically only executive modules may need to execute in kernel mode. Executive services run at user privilege level except when kernel mode is necessary to execute privileged instructions or access sensitive memory areas.

2.5 Distributed System Design Overview

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 beyond those available to SASOSs that run on stock processors. Table 3 summarizes the Sombrero distributed system features [9] and the advantages they provide.

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

3 CURRENT STATUS

The Sombrero prototype is being developed on two Alpha 21164 boxes running NT. NT Alpha PALCode has been modified to specialize an NT process to include the full address range. Additional PALCode modifications handle TLB misses and support an emulated RPLB. The prototype is expected to be operational by December 1998. Additional information on prototype design, comparisons between features of Sombrero and other SASOSs with and without changed hardware support can be found in [10] (an extended version of this paper), [11] and [12], and at http://www.eas.asu.edu/~sasos.

4 CONCLUSIONS

SASOSs increase the available choices for structuring the operating system and applications, for sharing, protecting and storing data, and for communication between programs. Economic and technological tradeoffs will increasingly favor single address space operation and SASOSs within ten years. It is necessary to make changes to CPU-resident protection and memory management hardware in order to design a SASOS that makes the paradigm shift viable. Advantages obtained from hardware support include improved program development environment, higher performance, better support for distributed applications and a better match to the needs of real-time systems. CPU-resident protection hardware and a SASOS that runs on it can be implemented now.

References

[1] J. S. Chase, et al, "Sharing and Protection in a Single-Address-Space Operating System", *ACM TOCS*, 12(4), November 1994, 271-307.

[2] G. Heiser, et al, “Implementation and Performance of the Mungi Single-Address Space Operating System”, *UNSW-CSE-TR-9704*, Department of Computer Systems, The University of New South Wales, June 1997.

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

[4] A. C. Skousen, “Sombrero: A Very Large Single Address Space Distributed Operating System", Computer Science and Engineering Department, ASU, MS Thesis, December 1994.

[5] D. Pountain, “Amending Moore’s Law”, *Byte*, March 1998.

[6] M. J. Torla, “RPLB Design Project Report”, MCS Project, CSE Dept., ASU, July 1997.

[7] A. C. Skousen, D. S. Miller, "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, ASU, *TR-96-013*, May 1996.

[8] S. Khatri, “Protection Structures in the Sombrero Operating System”, CSE Dept., ASU, MS Thesis, December 1997.

[9] A. C. Skousen, and D. S. Miller, "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”, CSE Dept., ASU, *TR-96-021*, May 1996.

[10] A. Skousen and D. Miller, “Operating System Structure and Processor Architecture for a Large Distributed Single Address Space”, CSE Dept., ASU, *TR-98-005*, August 1998.

[11] A. Skousen and D. Miller, “Support for Caching and Replication in a Distributed Single Address Space Operating System”, CSE Dept., ASU, *TR-98-006*, August 1998, (submitted to HICSS-32, January 1999.)

[12] A. Skousen and D. Miller, “Using a Distributed Single Address Space Operating System to Support Modern Cluster Computing”, CSE Dept., ASU, *TR-98-007*, August 1998, (submitted to HICSS-32, January 1999.)