Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures

And

Labels and Event Processes in the Asbestos Operating System

Presented by Petko Nikolov
9/22/09
Mach

• Problem
  • OS portability suffers due to diff. memory structures

• Solution
  • Portable, multiprocessor OS – Mach
  • Few assumptions about memory hardware
    – Just recover from page faults
Mach VM

• Supports:
  • Large, sparse virtual address spaces
  • Copy-on-write virtual copy operations
  • Copy-on-write and read-write memory sharing
  • Memory mapped files
  • User-provided backing store objects and pagers
Mach Design

- Task
- Thread
- Port
- Message
- Memory object
VM Operations

• A task can:
  • Allocate a region of VM on a page boundary
  • Deallocate a region of VM
  • Set the protection status of a region
  • Specify the inheritance of a region
  • Create and manage a memory object
Implementation

- 4 basic memory management data structures:
  - Resident page table
  - Address map
  - Memory object
  - Pmap

- Machine dependent vs independent
Resident Memory

• Physical memory – cache for virtual memory objects

• Physical page entries linked into:
  • Memory object list
  • Memory allocation queues
  • object/offset hash bucket
Address Maps

- Doubly-linked list of address map entries
- Map range of virtual addresses to area in virtual object
  - Contiguous
- Efficient for most frequent operations:
  - Page fault lookups
  - Copy/protection operations on address ranges
  - Allocation/deallocation of address ranges
Memory Objects

- Repository for data, indexed by byte
  - Resembles a UNIX file
- Reference counters allow garbage collection
- Pager – memory object managing task
  - Handles page faults, page-out requests outside of kernel
Sharing Memory

- Copy-on-write
  - Shadow objects
  - Remembers modified pages
- Read/write sharing
  - Memory object not appropriate for this
  - Must use sharing maps
Object Tree

- Must prevent large chains of shadow objects
  - Utilize GC for shadow objects
- Unnecessary chains occurs during heavy paging
  - Cannot be detected easily
- Complex locking rules
pmap

- Management of physical address maps
  - Only machine-dependent module
  - Implement page-level operations
  - Ensure hardware map is operational
  - Need not keep track of all currently valid mappings
- Machine-independent parts are the driving force of Mach VM operations
Porting Mach VM

- Code for VM originally ran on VAX machines
- IBM RT PC
  - Approx. 3 weeks for pmap module
- Sequent Balance
  - 5 weeks – bootable system
- Sun 3, Encore MultiMAX
### Performance of Mach VM Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Mach</th>
<th>UNIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero fill 1K (RT PC)</td>
<td>.45ms</td>
<td>.58ms</td>
</tr>
<tr>
<td>zero fill 1K(uVAX II)</td>
<td>.58ms</td>
<td>1.2ms</td>
</tr>
<tr>
<td>zero fill 1K(SUN 3/160)</td>
<td>.23ms</td>
<td></td>
</tr>
<tr>
<td>fork 256K (RT PC)</td>
<td>41ms</td>
<td>145ms</td>
</tr>
<tr>
<td>fork 256K (uVAX II)</td>
<td>59ms</td>
<td>220ms</td>
</tr>
<tr>
<td>fork 256K (SUN 3/160)</td>
<td>89ms</td>
<td></td>
</tr>
<tr>
<td>read 2.5M file(VAX 8200)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>first time</td>
<td>5.2/11sec</td>
<td>5.0/11sec</td>
</tr>
<tr>
<td>second time</td>
<td>1.2/1.4sec</td>
<td>5.0/11sec</td>
</tr>
<tr>
<td>read 50K file (VAX 8200)</td>
<td>(system/elapsed sec)</td>
<td></td>
</tr>
<tr>
<td>first time</td>
<td>.2/.3sec</td>
<td>.2/.5sec</td>
</tr>
<tr>
<td>second time</td>
<td>.1/.1sec</td>
<td>.2/.2sec</td>
</tr>
</tbody>
</table>

**Table 7-1:**

The cost of various measures of virtual memory performance for Mach, ACIS 4.2a, SunOS 3.2, and 4.3bsd UNIX.
Summary

- Sophisticated, hardware-independent VM system possible
- Can achieve good performance in some cases
Asbestos

Labels and Event Processing in the Asbestos Operating System

With slides borrowed from SOSP 2005 Asbestos presentation
Asbestos Outline

● Why is it needed?
● Other models
  ● Virtual machines
● Asbestos OS
  ● Labels
  ● Event processes
● Asbestos OKWS
● Performance
The Problem

- Web servers have exploitable software flaws
  - SQL injection, buffer overrun
- Private information leaked
  - Credit card #'s, SS #'s
  - All data potentially exposed due to single flaw
- Lack of isolation of user data
- Unconstrained information flow
Virtual Machine Isolation

Kernel

/submit_order.cgi

Alice
123 Main St.
4275-8204-4009-7915

VMM

Kernel

/submit_order.cgi

Bob
456 Elm St.
5829-7640-4607-1273
Problem with VM Isolation

- Course-grained sharing/isolation
- Heavy on resources
- Clumsy way to handle problem
  - Requires separate instance of OS for each label
  - Should really have support for this in OS
Information Flow Control Systems

● Conventional multi-level security
  • Kernel-enforced information flow control across processes
  • A handful of *levels* and *compartments*: “secret, nuclear”
  • Inflexible, administrator-established policies
  • Central authority, no privilege delegation

● Language-enforced information flow (Jif)
  • Applications can define flexible policies at compile time
  • Enforced within one process

● Asbestos
  • Applications can define flexible policies
  • Kernel-enforced across all processes
Approaches

Within a process

Across processes

Policy defined by:

Application

Kernel

POSIX.6 standard defines a subject to be an active entity that can cause information to flow between controlled objects. The POSIX.6 standard further specifies that since processes are the only such interface-visible element of both the POSIX.1 and POSIX.6 standards, processes are the only subjects treated in POSIX.6 MAC. Objects are defined by POSIX.6 as the interface-visible data containers, i.e., entities that receive or contain data to which MAC is applied. POSIX.6 specifies that objects are files (this includes regular files, directories, FIFO-special files, and unnamed pipes), and processes (in cases where a process is the target of some request by another process).

POSIX.6 also specifies that each subject and object shall have a MAC label associated with it at all times.

The POSIX.6 standard does not define a mandatory access control policy perse, but does define the restrictions for access based upon the comparison of the MAC label associated with the subject and the MAC label associated with the object. The first general restriction states that unprivileged processes (subjects) cannot cause information labeled at some MAC label (L1) to become accessible to processes at MAC label (L2) unless L2 dominates L1 (see Section 4.6.2 for the definition of "dominates"). This restriction is further defined with regard to accessing files and other processes. The restrictions placed on file manipulation (reading, writing, creating, etc.) are those that are generally accepted when implementing a MAC policy:

1. to read a file, the label of the process must dominate the label of the file.
2. to write to a file, the label of the process must be dominated by the label of the file (The POSIX.6 standard specifies that dominance equals equivalence - if the labels are equal, then each is considered to be dominant to the other).

For example, a user who is running a process at Secret should not be allowed to read a file with a label of Top-Secret. Conversely, a user who is running a process with a label of Secret should not be allowed to write to a file with a label of Confidential.

The POSIX.6 restriction for assigning labels to newly created files is that the new file must have a label that is dominant to the label of the subject, although the POSIX.6 interfaces only allow the label to be equal to that of the process creating the new object. This restriction forces implementations to not allow processes to create files at a "lower" label.

For example, a process with a label of Top-Secret should not be allowed to create a file with a label of Secret. There are analogous restrictions on object access when the object is a process as mentioned above.

Conventional MLS

Asbestos

Top-Secret
Asbestos Goal

Asbestos should support efficient, unprivileged, and large-scale server applications whose application-defined users are isolated from one another by the operating system, according to application policy.
Asbestos Goal

- Large-scale
  - Changing population of thousands
- Efficient
  - Cache user data, while keeping it isolated
- Unpriviliged
  - Minimum privilege required
- Application defines notion of user
- Isolation of users' data
- Application policy
  - Application-defined, OS-enforced
Asbestos Overview

- IPC similar to that of Mach
  - Messages sent to ports
  - Asynchronous, unreliable
- Asbestos labels
  - Track, limit flow of information
- Event processes
  - Efficiently support/isolate many concurrent users
Compartments

• Contamination / label type
  • Mike's data, Michele's data, Peter's business data
• Created by application
  • Creator process can delegate rights
Labels

- Each process has send and receive label
  - Send label track current contamination
  - Receive label tracks max contamination (clearance)
- Rules enforced when messages are sent
- Contamination of receiver updated
Basic Example

User
Kernel

Alice's ahttpd  Bob's ahttpd  cgi script  Backend DB

Send Label
Recv Label

[Diagram showing network flow and label placement]
Basic Example

Alice's ahttpd

Bob's ahttpd

cgi script

Backend DB

Send Label

Recv Label
Rule 1: The kernel contaminates the message with all of the sender's contamination.
Rule 2: The kernel validates that the destination has clearance to receive the contamination of the message.
Basic Example

Rule 3:
At delivery, the destination takes on the contamination of the message
Basic Example

User
Kernel

Alice's ahttpd
Bob's ahttpd
cgi script

Backend DB

Send Label
Recv Label
Implementing Clearance Checks

- How does the clearance check work?
- Labels form a lattice
- Partial ordering
  - Sender's send label must be less than or equal to the destination's receive label
- Send label updated with a least upper bound operator
Limiting Bug Impact

User
Kernel

Alice's ahttpd
Bob's ahttpd

cgi script

Send Label
Recv Label

Backend DB
Limiting Bug Impact

User
Kernel

Alice's ahttpd
Bob's ahttpd
cgi script

Send Label
Recv Label

Backend DB
Limiting Bug Impact

- User
- Kernel

Alice's ahttpd → Bob's ahttpd → cgi script → Backend DB

Send Label: Biohazard (Red) → Biohazard (Blue) → Biohazard (Red)
Recv Label: Biohazard (Red) → Biohazard (Blue) → Biohazard (Red)
Limiting Bug Impact

User
Kernel

Alice's ahttpd

Bob's ahttpd

cgi script

Send Label

Recv Label
Limiting Bug Impact

User
Kernel
Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB

Send Label
Recv Label
Limiting Bug Impact

User
Kernel

Alice's ahttpd
Bob's ahttpd

cgi script

Backend DB

Send Label
Recv Label

Limiting Bug Impact
Application Defined Policies

- Where did the compartments come from?
- How did the labels get set the way they are?
- In traditional multi-level security systems, the system operator does these things.
- Asbestos labels provide a decentralized and unprivileged method to set these initial conditions.
Compartment Creation

User

Kernel

Alice's ahttpd

Bob's ahttpd

cgi script

Backend DB

Send Label

Recv Label
Compartment Creation

Alice's ahttpd → password → Bob's ahttpd → cgi script → Backend DB

Send Label
Recv Label
Compartment Creation

User

Kernel

Alice's ahttpd

Bob's ahttpd

cgi script

Backend DB

Send Label

Recv Label

password
Any process that creates a compartment gets privilege with respect to that compartment:
- Declassify data
- Grant clearance
- Delegate privilege
Declassify
Receive

Alice's ahttpd

Bob's ahttpd

cgi script

Backend DB

User

Kernel

Send Label

Recv Label
Optional Labels

- Process can attach optional (discretionary) labels to messages
  - $C_S$ – Contaminate Send
  - $D_R$ – Declassify Receive
  - $D_S$ – Declassify Send
  - $V$ – Verify
Declassify receive grants clearance for a compartment to another process.

Send Label

Recv Label
The kernel checks that processes have the privilege needed to grant clearance.
Declassify
Receive

User
Kernel

Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB

Send Label
Recv Label

\[ D_R = \]

\[ D_R = \]
Declassify
Receive

User
Kernel

Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB

$D_R = \text{biohazard}$

Send Label
Recv Label
Declassify
Receive

User

Kernel

Alice's
ahttpd

Bob's
ahttpd

cgi script

Backend
DB

Send
Label

Recv
Label

$D_R =$
Declassify
Receive

User
Kernel

Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB

Send Label
Recv Label

$D_R = \text{biohazard}$
Declassify
Receive

User
Kernel

Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB

Send Label
Recv Label
Contaminate
Send

User
Kernel

Alice's ahttpd

Bob's ahttpd

cgi script

Backend DB

$C_S = \text{biohazard symbol}$

Send Label

Recv Label
Contaminate
Send

No privilege needed for $C_S$ – it can only add processes to a compartment.
Contaminate
Send

User
Kernel

Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB

Send Label

Recv Label

$C_S =$
Contaminate
Send

User
Kernel

Alice's
ahttpd

Bob's
ahttpd

cgi script

Backend
DB

$C_S =$ 

Send Label

Recv Label

58
Contaminate
Send

User
Kernel

Alice's ahttpd

Bob's ahttpd

cgi script

Backend DB

Send Label

Recv Label

\[ C_S = \]
Contaminate

Send

User
Kernel

Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB

\( C_s = \)

Send Label
Recv Label
CGI Setup

User
Kernel

Alice's ahttpd
Bob's ahttpd

Kernel

Send Label
Recv Label

D_R =
Bob Setup

User
Kernel

Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB

Send Label
Recv Label
Bob Setup

User
Kernel

Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB

Application Trust

Send Label

Recv Label
Label Implementation

- Contamination & Privilege = Label level (*, 0-3)

\[ \{A^*, B^3, 1\} \]

- A & B are compartment names
- Trailing 1 = Neutral in all other compartments
Declassification

- Information flow control keeps users data completely disjoint
- Alice wants to export some of her data, like her profile
  - But all her data is in her compartment
- How can she safely declassify her data?
- Alice must trust all processes that can do so
- To minimize declassification bugs, we build declassifiers as simple, single purpose programs
The process must have privilege for the compartment to use both $D_S$ and $D_R$. 

**Alice's ahttpd**

**Bob's ahttpd**

**Alice's profile declassifier**

**Backend DB**

$D_S =$ 

$D_R =$ 

Send Label

Recv Label
Declassification

User
Kernel
Alice's ahttpd
Bob's ahttpd
Alice's profile declassifier
Backend DB

Send Label
Recv Label

User
Kernel
Alice's ahttpd
Bob's ahttpd
Alice's profile declassifier
Backend DB
Declassification

User
Kernel

Alice's
ahttpd

Bob's
ahttpd

Alice's profile
declassifier

Backend
DB

profile

Send
Label

Recv
Label

69
Declassification

User
Kernel

Alice's ahttpd
Bob's ahttpd
Alice's profile declassifier
Backend DB

Send Label
Recv Label
profile
Since the process is privileged in Alice's compartment, it doesn't get contaminated.
Other Label Features

- Verify label on messages
  - Allows a process to prove it has labels at specific levels
- Integrity tracking
  - Enabled by level 0
- Different default level for send & receive labels
  - Enables interesting isolation policies
Preventing Contamination

• Ports
  • Associated with receive label
  • Verification imposed by receiver
  • Deny decontamination of receive labels beyond certain point
  • Receiver can grant rights to processes to send
  • Prevents arbitrary processes from sending to it
Combating Process Over-Contamination

● One process per user per service
  ● Lots of heavy weight context switches
  ● Lots of memory

● Combine processes to get one process per service?
  ● Become too contaminated to function
  ● Or too privileged

● Many processes are similar
● Programming style help?
Event Loop

```c
while (1) {
    event = get_next_event();
    user = lookup_user(event);
    if (user not yet seen)
        user.state = create_state();
    process_event(event, user);
}
```

- State isolated to data structures
- Stack not used from event to event
- Execution state has nice preemption points
Event Process Abstraction

```c
ep_checkpoint(&msg);
if (!state.initialized) {
    initialize_state(&state);
    state.reply = new_port();
}
process_message(&msg, &state);
ep_yield();  // revert to checkpointed memory
```

- Fork memory state for each new session
- Memory isolation is the same as fork
- Small differences anticipated, stored efficiently (diff)
- Event loop allows shared execution state
  - Allows light weight context switches
Web Server Architecture

netd

demux

ahttpd-idd

db-proxy

Database

worker1

... workerN
Experimental Setup – Memory

- How much memory do event processes use?
- Shopping cart application
  - Session state stored in event process
  - One event process per user

- Active session – Adding an item to the shopping cart

- Cached session – Deciding if you really want an item

Click!

Hmm

/shopping_cart.cgi
Event Processes Conserve Memory

- Includes user and kernel memory
- Not too many active sessions on a large website

![Graph showing memory usage vs. number of sessions]

- 9.48 pages/session for active sessions
- 1.45 pages/session for cached sessions
Experimental Setup – Throughput

- Simple character generation service
  - Not interested in application overhead
  - One event process per session (user)
- Compare to Apache & Mod-Apache
  - Varied concurrency to get best case performance

- Apache
  - Service runs as a CGI script
  - Connections are isolated into processes
  - Processes are not isolated or jailed on the system

- Mod-Apache
  - Service runs inside Apache process
- For 16 sessions, 150% of Apache
- For 10,000 session, 75% of Apache
# Latency

<table>
<thead>
<tr>
<th>Server</th>
<th>Latency (µs)</th>
<th>Median</th>
<th>90th Percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mod-Apache</td>
<td>999</td>
<td></td>
<td>1,015</td>
</tr>
<tr>
<td>Apache</td>
<td>3,374</td>
<td></td>
<td>5,262</td>
</tr>
<tr>
<td>OKWS, 1 session</td>
<td>1,875</td>
<td></td>
<td>2,384</td>
</tr>
<tr>
<td>OKWS, 1000 sessions</td>
<td>3,414</td>
<td></td>
<td>6,767</td>
</tr>
</tbody>
</table>

**Figure 8:** The median and 90th percentile latencies of requests to various server configurations.
Label Cost Linear in Label Size

- Label cost starts small but outstrips OKWS cost around 6500 sessions
- Declassifiers label size $O(#\text{sessions})$
Conclusion

- Asbestos labels make MAC more practical
  - Labels provide decentralized compartment creation & privilege
  - Event processes avoid accumulation of contamination
- The OK web server on Asbestos
  - Performs comparably to Apache
  - Provides better security properties than Apache