Virtual Memory: Mach and Asbestos

Presented by Hakim Weatherspoon
Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures
Richard Rashid, Avadis Tevanian, Michael Young, David Golub, Robert Baron, David Black, William Bolosky, and Jonathan Chew

• Richard Rashid
  – Lead developer of Mach
  – Microsoft Research

• William Bolosky
  – Microsoft Research
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
Takeaway

• Hardware-independent virtual memory (VM) is not only possible, but can be elegant
  – Hardware dependent structures contained to pmap
  – VM functionality can be delegated to user process
  – Mach works with uniprocessors, multiprocessors, One- and two- level page tables, and inverted page tables

• Lessons/Flaws
  – Macrobenchmark performance missing
  – Performance revisited over next 10+ years
Mach Virtual Memory

• 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 Abstractions

- **Task**
  - Basic unit of resource allocation
  - Virtual address space, communication capabilities

- **Thread**
  - Basic unit of computation

- **Port**
  - Communication channel for IPC

- **Message**
  - May contain port capabilities, pointers

- **Memory Object**
Virtual Memory 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 Map

- 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 Virtual Memory

• 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
Multiprocessor Issues

• TLB Consistency
  – Force interrupts to all CPU’s
  – Wait until timer interrupt
  – Temporarily allow inconsistency
## Performance

### 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>.27ms</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>5.2/11sec</td>
<td>5.0/11sec</td>
</tr>
<tr>
<td>(system/elapsed sec)</td>
<td>1.2/1.4sec</td>
<td>5.0/11sec</td>
</tr>
<tr>
<td>read 50K file (VAX 8200)</td>
<td>.2/.3sec</td>
<td>.2/.5sec</td>
</tr>
<tr>
<td>(system/elapsed sec)</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.
Perspective

• Achieved Goals
  – Sophisticated, hardware-independent VM system possible
  – Can achieve good (microbenchmark) performance

• Lessons/Flaws
  – Macrobenchmark performance missing
  – Performance revisited over next 10+ years
Labels and Event Processes in the Asbestos Operating System

Petros Efstathopoulos, Maxwell Krohn, Steve VanDeBogart, Cliff Frey, David Ziegler, Eddie Kohler, David Mazières, Frans Kaashoek, Robert Morris

- Frans Kaashoek and Robert Morris
  - MIT Faculty. Creators of Chord.
  - Academic father and grandfather to other authors and many more

- Maxwell Krohn
  - Creator of OK Cupid dating Service (ugrad @ Harvard)
  - Creator SFSLite and OK Web Server

- David Mazières
  - Stanford Faculty
  - Creator of SFS and libasync

- Eddie Kohler
  - UCLA Faculty
  - Creator of Click Modular Router

- Rest were students at MIT or UCLA
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
The Problem

• If Bob compromises the system, he can access Alice's data
The Problem

- If Bob compromises the system, he can access Alice's data.

**Kernel**

/submit_order.cgi

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

Bob
456 Elm St.
5829-7640-4607-1273
The Problem

- If Bob compromises the system, he can access Alice's data
The Problem

- If Bob compromises the system, he can access Alice's data
The Goal: User Isolation

• Bob should not be able to access Alice's data without Alice's permission
  – Alice and Bob’s data is isolated

• Complications
  – Even if there are bugs in the applications
  – Alice's data may travel through several processes

• To isolate, must prevent inappropriate data flow

• Application designer defines inappropriate
Virtual Machine Isolation

Kernel

/submit_order.cgi
Alice
123 Main St.
4275-8204-4009-7915

Kernel

/submit_order.cgi
Bob
456 Elm St.
5829-7640-4607-1273
Virtual Machine Tradeoffs

+ Strict partitioning of off-the-shelf software

+ But…
  – Coarse-grained sharing
  – Resource challenges

• Isolation should be an OS feature
Desired Behavior

Kernel

/submit_order.cgi

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

Bob
456 Elm St.
5829-7640-4607-1273
Desired Behavior

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

/submit_order.cgi

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

Bob
456 Elm St.
5829-7640-4607-1273
Desired Behavior

Kernel

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

Bob
456 Elm St.
5829-7640-4607-1273

/submit_order.cgi
Information Flow Control

• Information flow control solves this kind of problem
Information Flow Control

Label data with its owner (contaminate with respect to its owner)

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

Bob
456 Elm St.
5829-7640-4607-1273
Information Flow Control

Keep track of who the connection is for

Kernel

/submit_order.cgi

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

Bob
456 Elm St.
5829-7640-4607-1273
Information Flow Control

Kernel

/submit_order.cgi

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

Bob
456 Elm St.
5829-7640-4607-1273
Information Flow Control

Track the information as it moves around the operating system

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

Bob
456 Elm St.
5829-7640-4607-1273
Information Flow Control

Kernel

Base access control decisions on labels

/submit_order.cgi

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

Bob
456 Elm St.
5829-7640-4607-1273
Approaches:
Information Flow Control Systems

Within a process
Across processes

Policy defined by:
Application

Determining MAC Access
The functionality provided by the interfaces to support MAC is used to
determine the access of objects by subjects. The 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.

Kernel

Conventional MLS

Jif

Asbestos

Top-Secret
Aproaches: 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
Asbestos Goals

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

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

• Large-scale
  – Changing population of thousands

• Efficient
  – Cache user data, while keeping it isolated

• Unprivileged
  – 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
Asbestos Compartments

• Contamination / label type
  – Mike's data, Michele's data, Peter's business data
  – Example had two compartments: Alice & Bob

• Created by application
  – Creator process can delegate rights
  – Kernel enforces compartment policy
Asbestos 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
Asbestos Labels

- Application can create compartments without privilege
  - Application created users are isolated with the same mechanism as login users
  - Applications can easily sub-divide privilege
- Applications can delegate rights for compartments
  - Decentralized declassification like Jif
- Applications can choose different policies
  - Mandatory Access Control
  - Discretionary Access Control
  - Capabilities
  - More...
Basic Label Example

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

User

Kernel

Send Label

Recv Label
Basic Label Example

Alice's ahttpd
Bob's ahttpd
cgi script

Backend DB

User
Kernel
Send Label
Recv Label
Basic Label Example

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

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

User
Kernel
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

\[ V \]

\[ V \]

\[ V \]
Limiting Bug Impact

Alice's ahttpd

Bob's ahttpd

cgi script

Backend DB

User

Kernel

Send Label

Recv Label
Limiting Bug Impact

Alice's ahttpd

Bob's ahttpd

cgi script

Backend DB

User

Kernel

Send Label

Recv Label
Limiting Bug Impact
Limiting Bug Impact

Alice's ahttpd

Bob's ahttpd

cgi script

Backend DB

User

Kernel

Send Label

Recv Label
Limiting Bug Impact

Alice's ahttpd
Bob's ahttpd
cgi script

User
Kernel
Send Label
Recv Label

Backend DB
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
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.
Declassify

The kernel checks that processes have the privilege needed to grant clearance.

The diagram illustrates the flow of information between Alice's `ahttpd` and Bob's `ahttpd` through the `Backend DB`. The kernel checks the necessary privileges for clearance.

Concepts mentioned:
- `Alice's ahttpd`
- `Bob's ahttpd`
- `Kernel`
- `User`
- `Send Label`
- `Recv Label`
- `Backend DB`
Declassify Receive

Alice's ahttpd

Bob's ahttpd

cgi script

Backend DB

User

Kernel

Send Label

Recv Label

$D_R =$
Declassify
Receive

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

User
Kernel
Send Label
Recv Label

$D_R = \text{biological hazard}$
Declassify
Receive

User

Kernel

Send Label

Recv Label

Alice's ahttpd

Bob's ahttpd

cgi script

Backend DB

D_R =
Declassify Receive

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

User
Kernel
Send Label
Recv Label
Contaminate
Send

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

User
Kernel
Send Label
Recv Label

$C_S =$
Contaminate

Send

Alice's ahttpd

Bob's ahttpd

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

Backend DB

User

Kernel

Send Label

Recv Label

$C_S$ =
Contaminate
Send

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

User
Kernel
Send Label
Recv Label

C_S =
Contaminate
Send
Contaminate
Send

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

User -> Alice's ahttpd

Kernel

Send Label

C_S =

Recv Label
Contaminate
Send

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

User
Kernel
Send Label
Recv Label

C_s =
CGI Setup

User
Kernel
Send Label
Recv Label

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

$D_R =$
Bob Setup

Alice's ahttpd

Bob's ahttpd

cgi script

Backend DB

User

Kernel

Send Label

Recv Label

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
  - Including those that haven't been created yet

- Label representation linear in # 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 process 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 $D_R$. 

\[ D_S = \] 

\[ D_R = \]
Declassification

Alice's ahttpd

Bob's ahttpd

Alice's profile declassifier

Backend DB

User

Kernel

Send Label

Recv Label

Label

profile
Since the process is privileged in Alice's compartment, it doesn't get contaminated.

Alice's profile declassifier

Backend DB
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

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
Event Processes Abstraction

- Event process isolate state
  - Used so that each event process is only contaminated by one user
  - One process per service with one event process per user

- Even at 10,000 event processes, state is stored efficiently

- Little additional programmer overhead because event processes fit into event driven programming style
Performance Hypotheses

• Is the memory overhead from event processes mild, even at 10,000 sessions?

• Despite better security properties, is the performance of the OK web server on Asbestos comparable with Apache?

• Does the per connection kernel overhead increase at most linearly with the number of sessions?
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
Event Processes Conserve Memory

- Includes user and kernel memory
- Not too many active sessions on a large website
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
  – i.e. did not fork a worker process
- For 16 sessions, 150% of Apache
- For 10,000 session, 75% of Apache
## Latency

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

- Throughput benchmark
- DB performance fixed

- Label cost starts small but outstrips OKWS cost around 6500 sessions

- Declassifiers label size $O(\#\text{sessions})$
Future Work

- Minimizing label costs
- Easing programmability
- Label persistence
- More applications
Perspective

• Asbestos labels make MAC (mandatory access control) tractable
  – 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

• Lessons/Flaws
  – Increased cached sessions decrease performance
  – Label checking scales linearly with number of labels
  • “at least not quadratic or exponential”!
Next Time

• Read and write review:
  – *Exokernel: an operating system architecture for application-level resource management*, Dawson R. Engler, M. Frans Kaashoek, and James O'Toole, Jr. 15th ACM symposium on Operating systems principles (SOSP), December 1995, pages 251—266
Next Time

• Read and write review:

• Project Proposal
  – Return comments later today

• Project Survey Paper due next Friday

• Check website for updated schedule