Cute Tricks with Virtual Memory

(and why they don't work)

CS 614 9/7/06
by Ari Rabkin

A short history of VM

- Memory used to be quite limited.
- Use secondary storage to emulate.
- Either by *swapping* out whole processes, or by *paging* out individual pages.
- Old technology -- done in the 60s.
- Program is oblivious.

How paging works

- Divide RAM into fixed size (4k?) pages.
- Hardware typically has map from virtual address to physical page. (Page table).
- Each page table entry also has valid bits, mode bits, etc.

Paging, continued

- Hardware translates each mem. operand from virt. to phys. (Much too slow otherwise)
- Uses a *translation lookaside buffer* (TLB) that's typically a fast associative cache.
- Details of page tables, TLB vary from architecture to arch.

4

Key questions

- How architecture-independent can an OS's VM system be?
- What else can we do with VM?
  - VM hardware gives the OS complete mediation of app memory access -- very powerful!
- Who gets to do it?

Flash forward...

- Modern systems aren't RAM constrained: So what do we do with all the VM hardware?
- Lot of possible uses explored, e.g., memory mapping files. See Appel and Li for details.
  - Examples: Garbage collection, guard pages, etc etc.
- What is VM, fundamentally?

What is VM **really**?

- Virtual memory isolates process memory
- Separation is key tool for security
  - Can poke holes in interprocess barriers to allow IO or interprocess communication.
  - Shared memory IPC, remapping to send messages, etc.

The Mach Vision

- Want to build the next generation Unix:
  - More portable, more flexible
- Accent system showed you could use memory mapping for fast IPC.
- Why not move most of OS into user space, just use kernel for (memory-based) IPC?

The mach vision

- IPC via *messages* sent through *ports* to *threads*
- Fork needs to be cheap, so want to use copy-on-write extensively
- Push paging into userspace, for flexibility.
- Also, want to reduce size, cost of pagetables

mach VM system

- Want to minimize arch. dependence, and keep tables small.
- But page tables are architecture-specific.
- Solution: keep machine indep. data structures, machine-dep. structures are purely a cache.
- Authoritative copy is maintained by machine-independent parts of Mach.

Handling forks

- Unix typically uses lots of small processes, created via fork. Want to make this fast.
- Typically, every process needs its own page table.
- Page tables are expensive and bulky; how to reduce overhead?

Structures

- A few key data structures
  - Per-process Address maps to define regions (small!)
  - Machine-independent (inverted*)* page table (one per system)
  - Memory objects to hold actual backing store

Mach's interface

- Exposes powerful low-level interface.

vm_allocate(target_task,address,size,anywhere)
Allocate and fill with zeros new virtual memory either anywhere or at a specified address.
vm_copy(target_task,source_address,count,dest_address)
Virtually copy a range of memory from one address to another.
vm_deallocate(target_task,address,size)
Deallocate a range of addresses, i.e. make them no longer valid.
vm_inherit(target_task,address,size,new_inheritance)
Set the inheritance attribute of an address range.
vm_protect(target_task,address,size,set_maximum,new_protection)
Set the protection attribute of an address range.
vm_read(target_task,address,size,data,data_count)
Read the contents of a region of a task's address space.
vm_regions(target_task,address,size,elements,elements_count)
Return description of specified region of task's address space.
vm_statistics(target_task,vm_stats)
Return statistics about the use of memory by target_task.
vm_write(target_task,address,count,data,data_count)
Write the contents of a region of a task's address space.

Lessons

- Really is possible to build a largely machine-independent VM system.
- Radically improves portability.
- Very flexible: pagers can live in user space
- Can even make it fairly fast!

How can this be fast?

- Often, arch's page tables aren't quite right for the OS.
- Better to use machine-indep format that's really right, and then can be flexible in how arch. pages tables are used.
- Who designs better data structures: Programmers or chip designers?

But is it really fast?

- If Mach is so fast, how come nobody uses it?
- VM system isn't the whole OS: messaging hurts a lot.
- Security checks on messages are expensive: most of the expense of IPC is the check. (See the L3/L4 papers for how to make IPC fast)
- Mach didn't make it fast. Too much overhead on the critical path.

Hardware changes

- Hardware has changed since Mach
- CPUs getting faster, faster than RAM. Caching critical.
- Bigger (more important) caches
  - Microkernels don't play nice here: dump cache on the [frequent] context switches.
  - See Chen and Bershad 93

L4 syscall results

System	Time	Cycles
Linux	1.68 µs	223
L^4Linux	3.95 µs	526
L^4Linux (trampoline)	5.66 µs	753
MkLinux in-kernel	15.41 µs	2050
MkLinux user	110.60 µs	14710

Conclusion: Cost of syscall is from Mach, not microkernels generally.

The End of The µ-Kernels

- Linus Torvalds: “Essentially, trying to make microkernels portable is a waste of time. It's like building an exceptionally fast car and putting square tires on it. The idea of abstracting away the one thing that must be blindingly fast--the kernel--is inherently counter-productive.” [http://www.oreilly.com/catalog/opensources/book/linus.html]

Why Asbestos?

- Servers typically touch data for many users.
- Bugs may allow user A to read data from part of server talking to user B.
- Want to have outside enforcement of isolation of different user’s data.
- Normal processes too heavyweight; need something lighter.

What is Asbestos?

- Research OS, designed to have data flow labels
- Strong compartmentalization to limit damage from user-level compromises.
- Fires don’t spread
- Key question: how much does it all cost?

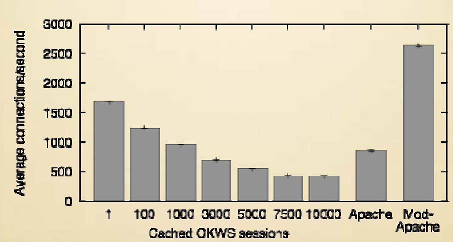
Event Processes

- Idea: have event processes where different instances share most of address space, but not everything.
- Just a few pages separate--for user-specific data.
- OS alters pagetables for just those pages on context switch

What does it cost?

- Not so cheap.
- Competitive with comparatively few cached events.
  - Note: cached events are essentially expired.
- But much better security

Asbestos Throughput



Asbestos Latency

Server	Latency (μs) Median	90th Percentile
Mod-Apache	999	1,015
Apache	3,374	5,262
OKWS, 1 session	1,875	2,384
OKWS, 1000 sessions	3,414	6,767

Lessons from Asbestos

- Can use VM to protect data in fine-grained way. Performance is respectable, particularly for an unoptimized system.
  - Sometimes high protection is more important than throughput
- Need to organize apps to keep protected data contiguous and on heap. Otherwise is costly.