Hot Topics — Cool Systems
Hot Topics — Cool Systems

Cool System: EROS [SSF99]

IDEA: FAST, SECURE, RELIABLE OS
Hot Topics — Cool Systems

Cool System: EROS [SSF99]

**IDEA:** fast, secure, reliable OS

Features:

- segregated capabilities,
- single-level store,
- persistence (via checkpointing),
- fast,
- mandatory access control,
- formal proof of confinement [SW00].
Hot Topics — Cool Systems

Cool System: EROS [SSF99]

**IDEA:** FAST, SECURE, RELIABLE OS

Features:

- segregated capabilities,
- single-level store,
- persistence (via checkpointing),
- fast,
- mandatory access control,
- formal proof of confinement [SW00].

EROS is a re-design of KeyKOS [BFF+92].
Clists form page table
eros access rights management

- Limit propagation and support revocation of rights by:
EROS ACCESS RIGHTS MANAGEMENT

• Limit propagation and support revocation of rights by:

  ★ “weak capabilities”:

    ➔ reading/writing any cap via a weak cap makes it R/O and weak
    ➔ can obtain transitive read-only access
EROS ACCESS RIGHTS MANAGEMENT

- Limit propagation and support revocation of rights by:
  - “weak capabilities”:
    - reading/writing any cap via a weak cap makes it R/O and weak
    - can obtain transitive read-only access
  - indirection:
    - Reference monitor (similar to L4 chief mediates cap transfer
    - inserts forwarding objects to capabilities
    - implements security policy
    - on change of policy can revoke caps by revoking forwarding object
## EROS PERFORMANCE

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Linux-Normalized</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipe Latency</td>
<td>5.66 μs</td>
<td>32.3%</td>
</tr>
<tr>
<td></td>
<td>8.34 μs</td>
<td></td>
</tr>
<tr>
<td>Pipe Bandwidth</td>
<td>281 MB/s</td>
<td>8.07%</td>
</tr>
<tr>
<td></td>
<td>260 MB/s</td>
<td></td>
</tr>
<tr>
<td>Create Process</td>
<td>0.664 ms</td>
<td>65.3%</td>
</tr>
<tr>
<td></td>
<td>1.92 ms</td>
<td></td>
</tr>
<tr>
<td>Ctxt Switch</td>
<td>1.19 μs</td>
<td>5.5%</td>
</tr>
<tr>
<td></td>
<td>1.26 μs</td>
<td></td>
</tr>
<tr>
<td>Grow Heap</td>
<td>20.42 μs</td>
<td>35.7%</td>
</tr>
<tr>
<td></td>
<td>31.74 μs</td>
<td></td>
</tr>
<tr>
<td>Page Fault</td>
<td>8.67 μs</td>
<td>99.5%</td>
</tr>
<tr>
<td></td>
<td>687 μs</td>
<td></td>
</tr>
<tr>
<td>Trivial Syscall</td>
<td>1.6 μs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.7 μs</td>
<td>-128%</td>
</tr>
</tbody>
</table>

Note: No data on cost of indirection for MAC.
Cool Stuff: Soft Timers [AD99]

**Problem: Want High-Resolution Timers**

E.g.: Timeouts in L4, prefer high resolution.

- Kernel has wakeup queues
- Kernel sets timer interrupt at certain frequency.
- At each interrupt (tick) checks wakeup queue.
**PROBLEM: WANT HIGH-RESOLUTION TIMERS**

E.g.: Timeouts in L4, prefer high resolution.

- Kernel has wakeup queues
- Kernel sets timer interrupt at certain frequency.
- At each interrupt (tick) checks wakeup queue.
- Handling interrupts is costly.
- Handling interrupt pollutes cache.
- Cannot have very high frequency timer ticks.

What can we do?
IDEA: SOFT TIMERS RATHER THAN HARDWARE INTERRUPT

• Check wakeup queues when cost of handling with them is low.
• This is generally the case when in the kernel already.
• Do it at a time where minimal state needs to be saved.
**Idea: Soft Timers Rather Than Hardware Interrupt**

- Check wakeup queues when cost of handling with them is low.

- This is generally the case when in the kernel already.

- Do it at a time where minimal state needs to be saved.

- **Suitable times:**
  - just before returning from system call,
  - at the end of exception handling,
  - at the end of interrupt handling,
  - in idle loop.
LIMITATIONS OF SOFT TIMERS:

- No absolute, only \textit{probabilistic} accuracy.
- Actual accuracy depends on frequency of trigger events.
- Can enforce absolute upper bound on delay via timer tick.
The Return of the Dumb Terminal: SLIM [SLN99]

**Problem:** High cost-of-ownership of PCs

- PCs are expensive to maintain/administer as individual machines.
- Would be cool to have the power of a big iron occasionally.
  - There’s still (or again?) some attraction in big central servers.
Ideas: Stateless thin-client architecture

- Stateless (other than frame buffer) cheap terminals (graphic displays),
- server sends 2D bitmaps,
- connected via 100Mbps ethernet.
**IDEA: STATELESS THIN-CLIENT ARCHITECTURE**

- Stateless (other than frame buffer) cheap terminals (graphic displays),
- server sends 2D bitmaps,
- connected via 100Mbps ethernet.

Supported by very simple, low-level display protocol (5 commands):

1. SET: set pixels in rectangle to literal value
2. BITMAP: fill rectangle with foreground, background colour
3. FILL: set all pixels to fixed value
4. COPY: copy rectangle
5. CSCS: colour-space conversion
**IDEA: STATELESS THIN-CLIENT ARCHITECTURE**

- Stateless (other than frame buffer) cheap terminals (graphic displays),
- server sends 2D bitmaps,
- connected via 100Mbps ethernet.

Supported by very simple, low-level display protocol (5 commands):

1. SET: set pixels in rectangle to literal value
2. BITMAP: fill rectangle with foreground, background colour
3. FILL: set all pixels to fixed value
4. COPY: copy rectangle
5. CSCS: colour-space conversion

Benchmarked GUI apps: photoshop, netscape, framemaker, MPEG, quake
How to Roll Your Own Tagged TLB and Caches
How to Roll Your Own Tagged TLB and Caches

CONTEXT SWITCH WITH UNTAGGED TLB:

→ load translation table base register to replace page table
→ flush instruction TLB
→ flush data TLB
→ flush instruction cache (if virtual)
→ flush data cache (if virtual)
⇒ High overhead
SIMULATING TLB TAGS BY SEGMENTATION:

Segmented virtual address

<table>
<thead>
<tr>
<th>Seg#</th>
<th>Page#</th>
<th>Offset</th>
</tr>
</thead>
</table>

Flat virtual address

<table>
<thead>
<tr>
<th>Page#</th>
<th>Offset</th>
</tr>
</thead>
</table>
SEGMENT REGISTERS AS ASID TAGS:

- Fill all segment registers with process’ ASID
SEGMENT REGISTERS AS ASID TAGS:

- Fill all segment registers with process’ ASID
- Problem if flat VAS is only 32-bit (Pentium)
SEGMENT REGISTERS AS ASID TAGS:

- Fill all segment registers with process’ ASID
- Problem if flat VAS is only 32-bit (Pentium)
- Partial solution [Lie95]:
  - split AS into large (3GB) and small area
  - small area split into smaller (128MB) slots
  - put small processes into a slot
  - can do fast context switches between one large and all small processes
ON NON-SEGMENTED ARCHITECTURE (STRONGARM)

• StrongARM SA-1100 is a high-performance low-power processor

• Designed for embedded applications

• Some less attractive features (for multitasking):
  ➔ virtually indexed, virtually tagged caches,
  ➔ untagged, hardware-loaded TLB.
ARM Page Tables

- Top-level PT entry can:
  - map a 1MB \textit{section};
  - protection with section granularity,
  - point to a second-level PT.

- Second-level PT entry can:
  - map a 64kB \textit{large} page,
  - map a 4kb \textit{small} page.

Protection with 1/4 page granularity.

- Each page or section is tagged with a \textit{domain id}
ARM DOMAINS

Additional access-control feature:

- 16 different domains,
- domain access control register (DACR) defines accessibility for each:
  - no access
  - access according to page permission bits
  - access irrespective of page permission bits
**How to Avoid Flushing?**

- Can avoid flushes if *no overlap of address spaces*
- How????
HOW TO AVOID FLUSHING?

• Can avoid flushes if *no overlap of address spaces*

• How????

• Two-part approach [WH00]:
  ➔ delay flushes as long as possible
  ➔ avoid overlaps as much as possible
HOW TO AVOID FLUSHING?

- Can avoid flushes if *no overlap of address spaces*
- How????

- Two-part approach [WH00]:
  - delay flushes as long as possible
    - *in microkernel*
  - avoid overlaps as much as possible
    - *in server*
**Delaying Flushes: Idea**

- Don’t change hardware page tables.

- Have translation table base register point to a cache of top-level page table entries (CPD).

- Use domain IDs as address-space tag.

- Flush TLB entries and caches when replacing CPD entry.
CACHING PAGE DIRECTORY

PD0

LPT00

LPT01

PD1

LPT10

LPT11

... ... ... ...

... ... ... ...

CPD

copy
CONTEXT SWITCH:

- Set software PT pointer to reference new context’s page table
- Load DACR to:
  - “access according to page permission bits” for new context’s domain,
  - “no access” for other domains
- Go!

No flushes required as long as have separate domain per context.
CACHE REPLACEMENT

- Domains ensure that CPD entries belonging to wrong context raise *domain fault* exception.

- Handled by:
  - replacing CPD entry
  - flushing TLBs and caches
  - flush only in case of actual collision.

- Also need to flush if recycling domains.
AVOIDING OVERLAP

• Server to allocate process data & stack at unique addresses
AVOIDING OVERLAP

- Server to allocate process data & stack at unique addresses
  ⇒ mini-SASOS (single-address-space OS)
AVOIDING OVERLAP

- Server to allocate process data & stack at unique addresses
  ⇒ mini-SASOS (single-address-space OS)

- Need to use separate domains for shared (text). segments

- Aliasing still a problem.


[Lie95] Jochen Liedtke. Improved address-space switching on Pentium processors by transparently multiplexing user


[WH00] Adam Wiggins and Gernot Heiser. Fast address-space