μ-Kernel Construction (8)

Small Address Spaces – IPC

Special Optimization for Untagged TLBs
Untagged TLB Context-Switch Costs

- Enter/exit kernel
  - 486 ... PIII: 40 ... 200 cycles
  - Pentium 4: 150 ... 200 cycles
- Switch thread
  - 486 ... PIII: ≈ 10 cycles
  - Pentium 4: ≈ 10 cycles
- Switch address space
  - Flush TLB
    - 486 ... PIII: ≈ 50 ... 80 cycles
    - Pentium 4: ≈ 230 ... 250 cycles
  - Refill TLB
    - 486 ... PIII: 6 ... 96 TLB refills, 15 ... 40 cycles/refill, ≈ 100 ... 4000 cycles
    - Pentium 4: 6 ... 192 TLB refills, 15 ... 500 cycles/refill, ≈ 100 ... 96000 cycles
  - Refill L1 caches
    - Pentium 4: 12 K Tracecache, 8 K Data cache, 15 ... 25 cycles/refill, ≈ 100 ... 16000 cycles
Even when calling a thread with a very small TLB working set

- Thread A frequently calls thread B
- Working sets
  - Thread A: 4 different sets of 10 pages between B-calls
  - Thread B: always the same 5 pages
Untagged TLB Context-Switch Costs

- Even when calling a thread with a very small TLB working set
  - Thread A frequently calls thread B
  - Working sets
    - Thread A: 4 different sets of 10 pages between B-calls
    - Thread B: always the same 5 pages

[cycles]
- 8 IPCS (w/o AS costs): 1440
- 60 TLB misses: 900 ... 30000
- 8 TLB flushes: 400 ... 2000

Untagged TLB total: 2740 ... 33440
Untagged TLB Context-Switch Costs

- Even when calling a thread with a very small TLB working set
  - Thread A frequently calls thread B
  - Working sets
    - Thread A: 4 different sets of 10 pages between B-calls
    - Thread B: always the same 5 pages

[cycles]
- 8 IPCs (w/o AS costs): 1440
- 60 TLB misses: 900 ... 30000
- 8 TLB flushes: 400 ... 2000

Untagged TLB total: 2740 ... 33440

Tagged TLB total: ≈ 1500

We get “TLB hits” starting at the second iteration, assuming a sufficiently large TLB.
Small Address Spaces on x86

How to emulate tagged TLBs?
Address Spaces

A

B

C

CS, DS

Kcode, Kdata, Physmem
Address Spaces

kernel CS, DS

user CS, DS

A

B

C

Kcode, Kdata, Physem
Small Address Spaces

CS, DS for A, B, C

A

B

C

Kcode, Kdata, Physmem
Small Address Spaces

CS, DS for A, B, C,

for X,

Y

Kcode, Kdata, Physmem

A

B

C
Small Address Spaces
Address Space (AS) – Hardware AS (HwAS)
Long-IPC Implementation Revisited

<table>
<thead>
<tr>
<th>Source / Destination</th>
<th>Method</th>
<th>Source offset</th>
<th>Dest offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large / Large</td>
<td>temp mapping switch HwAS</td>
<td>0</td>
<td>Kernel communication area</td>
</tr>
</tbody>
</table>

Temporary mapping area alias kernel communication area
## Long-IPC Implementation Revisited

<table>
<thead>
<tr>
<th>Source / Destination</th>
<th>Method</th>
<th>Source offset</th>
<th>Dest offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large / Large</td>
<td>temp mapping switch HwAS</td>
<td>0</td>
<td>Kernel com area</td>
</tr>
<tr>
<td>Large / Small</td>
<td>direct - no HwAS switch -</td>
<td>0</td>
<td>dest AS base</td>
</tr>
</tbody>
</table>

![Diagram showing the mapping of source and destination offsets](diagram.png)
Long-IPC Implementation Revisited

<table>
<thead>
<tr>
<th>Source / Destination</th>
<th>Method</th>
<th>Source offset</th>
<th>Dest offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large / Large</td>
<td>temp mapping switch HwAS</td>
<td>0</td>
<td>Kernel com area</td>
</tr>
<tr>
<td>Large / Small</td>
<td>direct - no HwAS switch -</td>
<td>0</td>
<td>dest AS base</td>
</tr>
<tr>
<td>Small / Small</td>
<td>direct - no HwAS switch -</td>
<td>source AS base</td>
<td>dest AS base</td>
</tr>
</tbody>
</table>

Diagram showing layers with arrows indicating the direction of data flow between source and destination.
### Long-IPC Implementation Revisited

<table>
<thead>
<tr>
<th>Source / Destination</th>
<th>Method</th>
<th>Source offset</th>
<th>Dest offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large / Large</td>
<td>temp mapping switch HwAS</td>
<td>0</td>
<td>Kernel com area</td>
</tr>
<tr>
<td>Large / Small</td>
<td>direct - no HwAS switch -</td>
<td>0</td>
<td>dest AS base</td>
</tr>
<tr>
<td>Small / Small</td>
<td>direct - no HwAS switch -</td>
<td>source AS base</td>
<td>dest AS base</td>
</tr>
<tr>
<td>Large / Same</td>
<td>direct - no HwAS switch -</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

![Diagram](image)
## Long-IPC Implementation Revisited

<table>
<thead>
<tr>
<th>Source / Destination</th>
<th>Method</th>
<th>Source offset</th>
<th>Dest offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large / Large</td>
<td>temp mapping switch HwAS</td>
<td>0</td>
<td>Kernel com area</td>
</tr>
<tr>
<td>Large / Small</td>
<td>direct - no HwAS switch -</td>
<td>0</td>
<td>dest AS base</td>
</tr>
<tr>
<td>Small / Small</td>
<td>direct - no HwAS switch -</td>
<td>source AS base</td>
<td>dest AS base</td>
</tr>
<tr>
<td>Large / Same</td>
<td>direct - no HwAS switch -</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Small / Same</td>
<td>direct - no HwAS switch -</td>
<td>source AS base</td>
<td>source AS base</td>
</tr>
</tbody>
</table>
## Long-IPC Implementation Revisited

<table>
<thead>
<tr>
<th>Source / Destination</th>
<th>Method</th>
<th>Source offset</th>
<th>Dest offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large / Large</td>
<td>temp mapping switch HwAS</td>
<td>0</td>
<td>Kernel comm area</td>
</tr>
<tr>
<td>Large / Small</td>
<td>direct - no HwAS switch -</td>
<td>0</td>
<td>dest AS base</td>
</tr>
<tr>
<td>Small / Small</td>
<td>direct - no HwAS switch -</td>
<td>source AS base</td>
<td>dest AS base</td>
</tr>
<tr>
<td>Large / Same</td>
<td>direct - no HwAS switch -</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Small / Same</td>
<td>direct - no HwAS switch -</td>
<td>source AS base</td>
<td>source AS base</td>
</tr>
<tr>
<td>Small / Large = Current HwAS</td>
<td>direct - no HwAS switch -</td>
<td>source AS base</td>
<td>0</td>
</tr>
</tbody>
</table>
## Long-IPC Implementation Revisited

<table>
<thead>
<tr>
<th>Source / Destination</th>
<th>Method</th>
<th>Source offset</th>
<th>Dest offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large / Large</td>
<td>temp mapping switch HwAS</td>
<td>0</td>
<td>Kernel com area</td>
</tr>
<tr>
<td>Large / Small</td>
<td>direct - no HwAS switch -</td>
<td>0</td>
<td>dest AS base</td>
</tr>
<tr>
<td>Small / Small</td>
<td>direct - no HwAS switch -</td>
<td>source AS base</td>
<td>dest AS base</td>
</tr>
<tr>
<td>Large / Same</td>
<td>direct - no HwAS switch -</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Small / Same</td>
<td>direct - no HwAS switch -</td>
<td>source AS base</td>
<td>source AS base</td>
</tr>
<tr>
<td>Small / Large = Current HwAS</td>
<td>direct - no HwAS switch -</td>
<td>source AS base</td>
<td>0</td>
</tr>
<tr>
<td>Small / Large ≠ Current HwAS</td>
<td>temp mapping switch HwAS</td>
<td>source AS base</td>
<td>Kernel com area</td>
</tr>
</tbody>
</table>
## Long-IPC Implementation Revisited

<table>
<thead>
<tr>
<th>Source / Destination</th>
<th>Method</th>
<th>Source Offset</th>
<th>Dest Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large / Large</td>
<td>temp mapping switch HwAS</td>
<td>0</td>
<td>Kernel com area</td>
</tr>
<tr>
<td>Large / Small</td>
<td>direct - no HwAS switch -</td>
<td>0</td>
<td>dest AS base</td>
</tr>
<tr>
<td>Small / Small</td>
<td>direct - no HwAS switch -</td>
<td>source AS base</td>
<td>dest AS base</td>
</tr>
<tr>
<td>Large / Same</td>
<td>direct - no HwAS switch -</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Small / Same</td>
<td>direct - no HwAS switch -</td>
<td>source AS base</td>
<td>source AS base</td>
</tr>
<tr>
<td>Small / Large ≠ Current HwAS</td>
<td>direct - no HwAS switch -</td>
<td>source AS base</td>
<td>0</td>
</tr>
<tr>
<td>Small / Large ≠ Current HwAS</td>
<td>temp mapping switch HwAS then direct</td>
<td>source AS base</td>
<td>Kernel com area — or — 0</td>
</tr>
</tbody>
</table>
Long-IPC Implementation Revisited

- A global bit in page table entry indicates that TLB entry should not be flushed on TLB flushes
  - Used for not flushing kernel entries

- or -

- first switch HwAS
then direct

- or -
Kerncom ar0

© 2009 Universität Karlsruhe, System Architecture Group
A **global bit** in page table entry indicates that TLB entry should not be flushed on TLB flushes

- Used for not flushing kernel entries

---

**No “global bit”: All TLB entries flushed**
Long-IPC Implementation Revisited

- A **global bit** in page table entry indicates that TLB entry should not be flushed on TLB flushes
  - Used for not flushing kernel entries

With “global bit”: Non-global TLB entries flushed
A **global bit** in page table entry indicates that TLB entry should not be flushed on TLB flushes
- Used for not flushing kernel entries
- Small spaces are global over all hardware address spaces
  - Mark small spaces’ PTEs as global

### Table: Temp mapping

<table>
<thead>
<tr>
<th>Source/C</th>
<th>Dest/C</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>Large</td>
<td>0</td>
</tr>
<tr>
<td>Small</td>
<td>Small</td>
<td>0</td>
</tr>
<tr>
<td>Large</td>
<td>Large</td>
<td>0</td>
</tr>
<tr>
<td>Small</td>
<td>Large</td>
<td>0</td>
</tr>
<tr>
<td>Large</td>
<td>Large</td>
<td>0</td>
</tr>
</tbody>
</table>

### Diagram: With “global bit”:
Non-global TLB entries flushed
Long-IPC Implementation Revisited

With “global PTE entries”, TLB misses only in dest space.
Without global pages, TLB misses in source and dest.

TLB misses on com area and in dest space (after the switch).

Small / Large ≠≠ ≠≠ Current HwAS

With mapping switch HwAS or –
first switch HwAS then direct

source AS base
Kernel com area – or – 0
Leave thread:

\[
\text{if } \text{mytcb.partner} \neq \text{nilthread} \text{ then} \\
\text{myPDE.TMarea} := \text{nil} ; \\
\text{if } \text{dest AS} = \text{my AS} \text{ then} \\
\text{flush TLB}
\]
Evicting the Temporary Mapping Area

- We must evict the temporary mapping from the TLB when switching from ... to ...
- Depending on whether small spaces use the temporary mapping / or not at all

<table>
<thead>
<tr>
<th>from \ to</th>
<th>Small AS</th>
<th>Active large AS</th>
<th>Inactive large AS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small AS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Large AS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1: assuming the temp. mapping is invalidated when switching to the small space
2: to prevent T3 from using T1’s mapping area after T1 (large AS A) → T2 (small AS B) → T3 (large AS A), possible due to 1
Evicting the Temporary Mapping Area

- We must evict the temp. mapping from the TLB when switching from ... to ...
- Depending on whether small spaces use the temp. mapping / or not at all

<table>
<thead>
<tr>
<th>from \ to</th>
<th>Small AS</th>
<th>Active large AS</th>
<th>Inactive large AS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small AS</td>
<td>Yes / No</td>
<td>Yes / No(^1)</td>
<td>No / No</td>
</tr>
<tr>
<td>Large AS</td>
<td>Yes / Yes(^2)</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

1: assuming the temp. mapping is invalidated when switching to the small space
2: to prevent T3 from using T1’s mapping area after T1 (large AS A) → T2 (small AS B) → T3 (large AS A), possible due to 1
Leave thread:
\[
\text{if } \text{mytcb.partner} \neq \text{nilthread} \text{ then } \\
\text{myPDE.TMarea := nil ;} \\
\text{if } (\text{dest is small or} \\
\text{dest HwAS = curr HwAS}) \text{ then} \\
\text{flush TLB}
\]
Temporary Mapping Revisited

Leave thread:

if mytcb.partner ≠ nilthread then
    myPDE.TMarea := nil ;

if (dest is small or
     dest HwAS = curr HwAS) and
not curr is small
then
    flush TLB

Assuming small spaces never use the temp. mapping area.
Thread Switching Revisited

- A sends to B, executes in HwAS B
  - A is preempted or PF in A
  - Thread switch from A to X
Thread Switching Revisited

- **A** sends to **B**, executes in HwAS **B**
  - **A** is preempted or PF in **A**
  - Thread switch from **A** to **X**
  - Switch back from **X** to **A**
Thread Switching Revisited

- A sends to B, executes in HwAS B
  - A is preempted or PF in A
  - Thread switch from A to X
  - Switch back from X to A
Thread Switching Revisited

- **A** sends to **B**, executes in HwAS **B**
  - **A** is preempted or PF in **A**
  - Thread switch from **A** to **X**
  - Switch back from **X** to **A**
  - Preemption or PF resumes in **A**, but **A** now executes in HwAS **X**
Thread Switching Revisited

A Solution

- A sends to B, executes in HwAS B
  - A is preempted or PF in A
    - Mark A “in partner space”
Thread Switching Revisited

A Solution

- A sends to B, executes in HwAS B
  - A is preempted or PF in A
    - Mark A "in partner space"
  - Thread switch from A to X

In partner space
Thread Switching Revisited
A Solution

- A sends to B, executes in HwAS B
- A is preempted or PF in A
  - Mark A “in partner space”
- Thread switch from A to X
- Switch back from X to A
  - Also switch HwAS to HwAS B
Small Address Spaces and Fast System Calls

Getting around automatic segment register reloading
Fast System Calls

- Optimized instructions
  - sysenter/sysexit (Intel)
  - syscall/sysret (AMD)

- Faster than software interrupts
  - Can avoid certain checks
  - Unconditionally reload segment registers (page based protection)

<table>
<thead>
<tr>
<th></th>
<th>450 MHz PIII</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int / Iret</td>
<td>280</td>
</tr>
<tr>
<td>Sysenter / Sysexit</td>
<td>50</td>
</tr>
</tbody>
</table>
Fast System Calls

- Optimized instructions
  - sysenter/sysexit (Intel)
  - syscall/sysret (AMD)

- Faster than software interrupts
  - Can avoid certain checks
  - Unconditionally reload segment registers (page based protection)

<table>
<thead>
<tr>
<th></th>
<th>450 MHz PIII</th>
<th>1.5 GHz P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int / Iret</td>
<td>280</td>
<td>1600</td>
</tr>
<tr>
<td>Sysenter / Sysexit</td>
<td>50</td>
<td>140</td>
</tr>
</tbody>
</table>

Problematic for small spaces (segment based protection)
Fast System Calls
Automatic Segment Register Reloading

CS, DS for A

A

X

Y

Kcode, Kdata, Physmem

B

C
Fast System Calls
Automatic Segment Register Reloading

sysenter

kernel CS, DS

A

B

C

Kcode, Kdata, Physmem
Fast System Calls
Automatic Segment Register Reloading

sysexit

user CS, DS

A

X

Y

Kcode, Kdata, Phymem

B

C

Task A now has full access to small space area!
Automatic Segment Register Reloading Solution – In-Kernel Trampoline

Kernel_entry:

perform system call

User_function:

sysenter

%cs = kernel_cs
%ss = kernel_ds
%ds = kernel_ds

Application

Kernel
Automatic Segment Register Reloading Solution – In-Kernel Trampoline

Kernel_entry:
perform system call
sysexit

Trampoline:
mov $user_ds, %ss
mov $user_ds, %ds
ret

User_function:
%cs = kernel_cs
%ss = kernel_ds
%ds = kernel_ds

Application
Pops off %cs and %eip from user stack (%esp)
Automatic Segment Register Reloading Solution – In-Kernel Trampoline

User_function:

sysenter

Kernel_entry:

perform system call

sysexit

Trampoline:

mov $user_ds, %ss
mov $user_ds, %ds
lret

What if %esp points to small space memory?

Works since we set %ss before accessing stack.
Automatic Segment Register Reloading Solution – In-Kernel Trampoline

User_function:

What if interrupts arrive or lret instruction raises exceptions (e.g., page-fault on stack access)?

- Exception gets handled
- Return to faulting instruction (lret)
  - Via iret (no nested fast syscall)
  - Restores kernel %cs, user %ss and %ds
- Continue to completion
Outlook on IA32 + TLB Tagging
AMD Opteron

- “Flush Filter”, Patent 6,604,187
  - Automatic ASN tagging
- Lookup cache, storing ptab lookup chain
- Tag TLB as “to-be-flushed”
  - If entry in cache gets modified (mem snooping)
  - If cache overflows
- Selective flushing on CR3 reloads

- Pistachio IPC: 750 cycles → 240 cycles