μ-Kernel Construction (5)

IPC Implementation
IPC Importance
General IPC Algorithm

- Validate parameters
- Locate target thread
  - Return error if unavailable
- Transfer message
  - Untyped items (short IPC)
  - Typed items (long IPC)
- Schedule target thread
  - Switch address space as necessary
- Wait for IPC (reply/next request)
IPC Implementation

Short IPC
Short IPC (uniprocessor)

- System-call pre (disable IRQs)
- Identify dest thread and check
  - Same chief / no IPC redirection?
  - Ready-to-receive?
- Analyze message and transfer
  - Short IPC $\Rightarrow$ no action required
- Switch to dest thread & address space
- System-call post

The critical path
Short IPC (uniprocessor) "call"

- System-call pre (disable IRQs)
- Identify dest thread and check
  - Same chief / no IPC redirection?
  - Ready-to-receive?
- Analyze message and transfer
  - Short IPC $\Rightarrow$ no action required
- Switch to dest thread & address space
- System-call post

Running

Wait to receive

Running

Wait to receive
Short IPC (uniprocessor) "send" (eagerly)

- System-call pre (disable IRQs)
- Identify dest thread and check
  - Same chief / no IPC redirection?
  - Ready-to-receive?
- Analyze message and transfer
  - Short IPC $\Rightarrow$ no action required
- Switch to dest thread & address space
- System-call post
Short IPC (uniprocessor) “send” (lazily)

- System-call pre (disable IRQs)
- Identify dest thread and check
  - Same chief / no IPC redirection?
  - Ready-to-receive?
- Analyze message and transfer
  - Short IPC \(\rightarrow\) no action required
- Switch to dest thread & address space
- System-call post

running → wait to receive

wait to receive → running
IPC

Kernel Stacks and TCBs

%ESP0

Kernel memory

EAX
ECX
EDX
EBX
ESI
EDI
EBP
ESP
EFLAGS
EIP
CS
ES
SS
FS
DS
GS
IPC via sysenter/sysexit

- Real register use
  - EAX: dest. TID $\Rightarrow$ sender TID
  - ECX: timeouts $\Rightarrow$ user IP (sysexit)
  - EDX: receive TID $\Rightarrow$ user SP (sysexit)
  - EBX: (scratch) $\Rightarrow$ MR$_1$
  - EBP: (scratch) $\Rightarrow$ MR$_2$
  - ESI: MR$_0$ [only unchanged register]
  - EDI: UTCB(sender) $\Rightarrow$ UTCB(receiver)
Implementation Goal

- Most frequent kernel op: Short IPC
  - Thousands of invocations per second

- Performance is critical
  - Structure IPC for speed
  - Structure entire kernel to support fast IPC

- What affects performance?
  - Cache line misses
  - TLB misses
  - Memory references
  - Pipe stalls and flushes
  - Instruction scheduling
Fast Path

- Optimize for common cases
  - Write in assembler
  - Non-critical paths written in C++
    - But still fast as possible

- Avoid high-level language overhead
  - Function call state preservation
  - Incompatible code optimizations

- We want every cycle possible!
  - At least sometimes …
IPC Requirements for Fast Path

- Untyped message
- Single runnable thread after IPC
  - Must be valid call-like IPC
    - Send phase
      - Target is already waiting
    - Receive phase
      - Sender is not ready to couple, causing us to block
  - Switch threads, originator blocks
- No receive timeout
  - Send timeout can be ignored: receiver is waiting
  - Xfer timeouts do not apply for untyped messages
Memory is Forbidden

- Memory references are slow
  - Avoid in IPC
    - E.g., use lazy scheduling
  - Avoid in common case
    - E.g., (xfer) timeouts

- Microkernel should minimize artifacts
  - Cache pollution
  - TLB pollution
  - Memory bus
Optimized Memory

Also hard-wire TLB entries for kernel code and data.

- stack
- thread state
  - &UTCB
  - cpu ID
  - thread ID

Single TLB entry

TCB state, grouped by cache lines
Walking/Modifying a linked list has a TLB footprint.
Lazy Scheduling

- Do not update the scheduling lists
  - Blocked sender remains in ready list
    - Check real thread state when dispatching
    - May be released before being scheduled ➞ avoids list manipulations
  - Unblocked receiver not added to ready list
    - Appended to ready list at end of timeslice
    - May block before ➞ avoids list manipulations
Avoid Table Lookups

\[ \text{virtual TCB area} \]

\[ \text{TCB} = \text{TCB\_area} + ((\text{thread\_no} \gg x) \& \text{TCB\_size\_mask}) \]
Validate Thread ID

Are the thread IDs equal?
Branch Elimination

```
slow = ~receiver->thread_state | ((timeouts ^ 0x400) & 0xffff) | sender->resources | receiver->resources;

if (0 != slow)
    enter_slow_path()
```

- Reduces branch prediction footprint
- Avoids mispredictions, stalls, and flushes
- Slightly increases latency for slow path
TCB Resources

- One bit per resource
- Fast path checks entire word
  - If not 0, jump to resource handlers

Resources bitfield

Copy area

Debug registers

© 2009 Universität Karlsruhe, System Architecture Group
Slow and Fast

user mode → IPC wait via slow path → user mode

user mode → IPC send via fast path → user mode

user mode → IPC wait via fast path → user mode

user mode → IPC send via fast path → user mode
Consistent State

- Cooperative thread scheduling in kernel
- TCB in consistent state for IPC wait
  - IPC restores user mode context
    - Avoids cycles for restoring kernel context
    - Fast path can activate slow path TCB

Problem?

Can't use fast path for kernel threads.
How often do kernel threads use IPC?
How to efficiently detect kernel threads?

⇒ Use resource bit!
IBM PowerPC 750, 500 MHz, 32 registers

- Many cycles wasted on pipe flushes for privileged instructions.
- Up to 10 physical registers
- Virtual register copy loop
Short IPC Performance (2)

AMD Opteron 242,
1.6 GHz
IPC Implementation

Long IPC
Long IPC (uniprocessor)

- System-call pre (disable IRQs)
- Identify dest thread and check
  - Same chief / no IPC redirection?
  - Ready-to-receive?
- Analyze message and transfer
  - Long/map:
    - Preemptions possible!
      (end of timeslice, device interrupt…)
    - Pagefaults possible!
      (in source and dest address space)
  - transfer message -
  - Switch to dest thread & address space
  - System-call post
**Long IPC (uniprocessor)**

- System-call pre (disable IRQs)
- Identify dest thread and check
  - Same chief / no IPC redirection?
  - Ready-to-receive?
- Analyze message and transfer
  - **Long/map:**
    - Lock both partners
    - **- transfer message -**
    - Unlock both partners
    - Switch to dest thread & address space
    - System-call post

**Preemptions possible!**
(end of timeslice, device interrupt…)

**Pagefaults possible!**
(in source and dest address space)
Long IPC (uniprocessor)

- System-call pre (disable IRQs)
- Identify dest thread and check
  - Same chief / no IPC redirection?
  - Ready-to-receive?
- Analyze message and transfer
  - Long/map:
    - Lock both partners
    - Enable IRQs
    - **transfer message**
    - Disable IRQs
    - Unlock both partners
  - Switch to dest thread & address space
  - System-call post

Preemptions possible!
(end of timeslice, device interrupt...)

Pagefaults possible!
(in source and dest address space)
Long IPC (uniprocessor)

- System-call pre (disable IRQs)
- Identify dest thread and check
  - Same chief / no IPC redirection?
  - Ready-to-receive?
- Analyze message and transfer
  - Long/map:
    - Lock both partners
    - Enable IRQs
    - **transfer message**
    - Disable IRQs
    - Unlock both partners
- Switch to dest thread & address space
- System-call post
String IPC / memcpy

- Why?
  - Trust
  - Granularity
  - Synchronous ("atomic") transfer
Copy In – Copy Out

- Copy into kernel buffer
Copy In – Copy Out

- Copy into kernel buffer
- Switch spaces
Copy In – Copy Out

- Copy into kernel buffer
- Switch spaces
- Copy out of kernel buffer

- Costs for \( n \) words
  - \( 2 \times 2n \) r/w operations
  - \( 3 \times n/8 \) cache lines
    - \( 1 \times n/8 \) cache misses (small \( n \))
    - \( 4 \times n/8 \) cache misses (large \( n \))
Temporary Mapping
Temporary Mapping

- Select dest area (2x4 MB)
Temporary Mapping

- Select dest area (2x4 MB)
- Map into source AS (kernel)
Temporary Mapping

- Select dest area (2x4 MB)
- Map into source AS (kernel)
- Copy data
Temporary Mapping

- Select dest area (2x4 MB)
- Map into source AS (kernel)
- Copy data
- Switch to dest space
Temporary Mapping

- Copy page directory entry (PDE) from dest
  - Addresses in temporary mapping area are resolved using dest’s page table
Temporary Mapping

- Problems
  - Multiple threads per AS
  - Mappings might change while message is copied

- How long to keep PTE?
- What about TLB?
Temporary Mapping

- When switching threads during IPC
Temporary Mapping

- When switching threads during IPC
  - Invalidate PTE
  - Flush TLB
Temporary Mapping

- When returning to a thread during IPC
  - Raises pagefault on copy area
  - (Re)copy PDE from receiver
Temporary Mapping

Reestablishing temp mapping requires storing **partner id** and **dest area address** in the sender’s TCB.

Note: Receiver’s page mappings might have changed!
Temporary Mapping

Start temp mapping:
mytcb.partner := partner;
mytcb.waddr := dest 8M area base;
myPDE.TMarea := destPDE.destarea.

Leave thread:
\[
\text{if } \text{mytcb.waddr} \neq \text{nil} \text{ then } \\
\text{myPDE.TMarea} := \text{nil} ; \\
\text{if dest AS = my AS then } \\
\text{flush TLB ;}
\]

Close temp mapping:
\[
\text{mytcb.waddr} := \text{nil} ; \\
\text{myPDE.TMarea} := \text{nil} .
\]

Optimization only:
Avoids second TLB flush if subsequent thread switch would flush the TLB anyway.

why?
Temporary Mapping

- **Alternative method:**

  Requires separation of TLB flush
  - and
  - load PT root

  Does therefore not work reasonably on x86.

  Load PT root implicitly includes TLB flush on x86.

---

Leave thread:

```plaintext
if mytcb.waddr ≠ nil then
  myPDE.TMarea := nil;
  flush TLB;
  TLB flushed := true
fi.
```

Thread switch:

```plaintext
... if TLB just flushed
  then TLB flushed := false
  else flush TLB
fi;
PT root := ...
```
Temporary Mapping

- Page Fault Resolution:
Temporary Mapping

- Page Fault Resolution:
Temporary Mapping

- Page Fault Resolution:
Temporary Mapping

- Page Fault Resolution:

  TM area PF:
  
  \[
  \text{if } \text{myPDE.TMarea} = \text{destPDE.destarea} \text{ then} \\
  \text{tunnel to (partner)} ; \\
  \text{access dest area} ; \\
  \text{tunnel to (my)} \\
  \text{fi} ; \\
  \text{myPDE.TMarea := destPDE.destarea} .
  \]
Temporary Mapping

- SMP
Temporary Mapping

- SMP
  - TM area per processor
Temporary Mapping

- SMP
  - TM area per processor
  - Page table per processor

CPU1

CPU2

CPU1 AS

CPU2 AS
Cost Estimates for Copying \( n \) Words

<table>
<thead>
<tr>
<th></th>
<th>Copy in - copy out</th>
<th>Temporary mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>R/W operations</strong></td>
<td>( 2 \times 2n )</td>
<td>( 2n )</td>
</tr>
<tr>
<td><strong>Cache lines</strong></td>
<td>( 3 \times n/8 )</td>
<td>( 2 \times n/8 )</td>
</tr>
<tr>
<td><strong>Small n overhead cache misses</strong></td>
<td>( n/8 )</td>
<td>0</td>
</tr>
<tr>
<td><strong>Large n cache misses</strong></td>
<td>( 5 \times n/8 )</td>
<td>( 3 \times n/8 )</td>
</tr>
<tr>
<td><strong>Overhead TLB misses</strong></td>
<td>2</td>
<td>( n/ ) (words per page)</td>
</tr>
<tr>
<td><strong>Startup instructions</strong></td>
<td>0</td>
<td>50</td>
</tr>
</tbody>
</table>

(assuming 8 words/cache line)
486 IPC Cost

- **Mach**: Copy in/out
- **L4**: Temp. mapping

![Graph showing IPC costs for Mach and L4](image-url)

- **Mach**: Copy in/out
- **L4**: Temp. mapping
- **L4 + cache flush**: Shows increased cost with cache flush
- **L4**: Basic copy operation
- **raw copy**: Lowest cost option

- **msg len** (message length) on the x-axis
- **[µs]** on the y-axis