SHARED ADDRESS TRANSLATION REVISITED

Xiaowan Dong
Sandhya Dwarkadas
Alan L. Cox

University of Rochester
University of Rochester
Rice University
Limitations of Current Shared Memory Management

• Physical memory sharing is common

• However, address translation is private per process
  • page tables and Translation Lookaside Buffer (TLB) entries

• Potential for duplicate translation information (as much as 58% on Android)
  • Scalability problem: O(# of processes)
  • Inefficient utilization of shared caches
Previous Work

- Previous work shares page tables for applications handling *large amounts of contiguous data*
  - E.g., PostgreSQL database systems

- Limitations:
  - Overlook code at smaller granularity (such as *shared libraries*).
  - Ignore duplication in the TLB.

- New opportunities on Android, where shared libraries are used intensively.
Android Process Creation Model

All applications share the same physical and virtual addresses for the preloaded libraries.
Goal: Shared Address Translation: Page Tables and TLB Entries

• Sharing address translation for the zygote-preloaded shared libraries

• Implemented at the OS level with existing hardware support
  • Mostly machine-independent

• Benefits
  • Reduce soft page faults
  • Improve cache and TLB performance
Impact of Shared Libraries on Instruction Footprint

- Number of shared libraries per application:
  - Loaded: 88 to 107 (zygote-preloaded: 88)
  - Invoked: 24 to 68 (zygote-preloaded: 21 to 46)
Shared Library Instruction Footprint Intersection

- **Considerable overlap** in the shared library code accessed across different applications
  - 46% of total inst pages accessed are in common for each pair of applications
  - Zygote-preloaded: 38%

The % of inst footprint overlapped:
- Laya Music Player: 91%
- Adobe Reader: 85%
- MX Player: 72%
SHARING ADDRESS
TRANSLATION
Sharing Page Tables

• The ARM architecture defines a two-level hierarchical page table

• L2 page table pages are shared at fork time between the zygote and its child processes
  • Supports private writable memory regions

• Shared page table pages and physical pages should both be managed in a copy-on-write (COW) manner
Maintaining Shared Page Tables

• A shared page table page needs to be unshared (COWed) in the following cases:

  • Page fault with write access
  • A process creates, destroys, or modifies a memory region within the range of a shared page table page
  • A process tries to free a shared page table page

• Modification to any memory region will lose the entire shared page table page
  • Mapping the page table entries of the code segment and data segment of a shared library into different page table pages
Sharing TLB Entries

- **Global bit**
  - We set the global bit in the page table entries of the zygote-preloaded shared libraries’ code segments
  - Overrides Address Space Identifier (ASID) in TLB

- **Domain protection model of 32-bit ARM**
  - Prevents processes not forked from the zygote from accessing the shared global TLB entries
    - E.g., system services and daemons
Leveraging the domain protection model

Domain 1
User Space

Domain 2
Kernel Space

Domain 3
Zygote-preloaded shared libraries

Non-zygote processes

Zygote-like processes

00: No access permission
01: Based on permission bits listed in the TLB entry

DACR

Memory Abort Handler

Global bit

Domain field

VPN
ASID
1
0011
Permission bits

TLB

Trap into kernel

Check fault status register

Flush all TLB entries with the faulting address
EVALUATION
Evaluation Platforms

- Nexus 7 (2012)
  - 1.2GHz Nvidia Tegra 3 processor with four ARM Cortex-A9 cores
  - A private 2-level TLB
    - I/D micro TLB (*flushed over context switch*)
    - 128-entry main TLB
  - 32KB/32KB L1 cache (I/D)
  - 1MB shared L2 cache

- Android KitKat 4.4.4 OS
  - New android runtime (ART)

- Benchmarks:
  - Most popular application in each category on Google Play Store
Zygote Fork

• Sharing page table improves execution time of a zygote fork by **2.1x**

• **Trade-off between cost of fork and # of page faults experienced by child processes**
  • Sharing page table is the best of both worlds

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Execution Cycles (x 10^6)</th>
<th># of PTPs allocated</th>
<th># of PTEs copied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stock Android</td>
<td>2.9</td>
<td>38</td>
<td>3,900</td>
</tr>
<tr>
<td>Copied PTEs</td>
<td>4.6</td>
<td>51</td>
<td>9,800</td>
</tr>
<tr>
<td>Shared PTPs</td>
<td>1.4</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>
• Every application follows the same launch procedure before it loads its application-specific Java classes

• Launch time improved by 7% (10% with 2MB alignment)
  • 94% fewer page faults for creating PTEs that map shared code and data
  • 15% reduction in L1 Icache stall cycles
  • 68 % less page table page allocation
Over The Course of Execution

38% fewer Page faults for creating PTEs that map shared code and data on average (maximum 78%)

35% fewer page table pages allocated (maximum 58%)
Android IPC Performance

- Inter-process communication (IPC) is common on Android
- Developed microbenchmark using Android IPC binder mechanism
- Inst main TLB stall cycles are reduced by:
  - Client: 36%
  - Server: 19%
Conclusion

• Android presents opportunities for shared library address translation sharing

• We eliminated the duplication of address translation on Android

• Android’s application launch, steady-state, and context switch efficiency are improved
  • Speed up a zygote fork by 2.1x
  • Improve application launch by 10%

• Our shared address translation infrastructure should be portable to other platforms
Large Pages Are Inefficient for Zygote-preloaded Shared Libraries

- Using large pages (64KB page for example) will waste physical memory compared to 4KB base pages:
  - 2.6x memory consumption on average
  - 94% more memory consumption for the union set

- Linux does not support the use of large pages for code

- Our design can complement large pages
  - 64KB page on ARM also requires 2-level page table as 4KB page does

CDF of # of 4KB pages untouched within a 64KB page of zygote-preloaded shared libraries
Sharing TLB

zygote
exec ↓
Task_struct.
zygote = 1

mmap the code segment of a shared library

fork ↓
Task_struct.
zygote_like = 1

Page fault on a zygote-preloaded shared library

Task_struct.
zygote = 1 or
zygote_like = 1?

yes

Vma.
global = 1?

yes

Vma.global = 1

Set global bit in PTE

Global bit is used for kernel pages in stock Linux
Sharing Page Table at Fork

Virtual memory area (VMA): a memory region

Parent’s addr space
- vma1
- vma2
- vma3

Child’s addr space
- vma1
- vma2
- vma3

- L1 PTP
  - L1 PTE1
  - L1 PTE2
  - L1 PTE3

- L2 PTP
  - L2 PTE1
  - L2 PTE2
  - L2 PTE3

L2 PTP is shared?
- No
- Write-protect every writable L2 PTE

If ARM supports write protection in L1 PTE as x86, we can avoid write-protecting every L2 PTE