The Cost of IPC

An Architectural Analysis

I. Gelado J. Cabezas L. Vilanova N. Navarro

Universitat Politècnica de Catalunya
Departament d'Arquitectura de Computadors
{igelado,jcabezas,vilanova,nacho}@ac.upc.edu

Presented by:
Roberto Gioiosa

Funded by:
Ministry of Science and Technology of Spain
HiPEAC
SARC

Introduction

- Operating Systems should provide:
  - Security
  - Process isolation
  - Resource multiplexation
- OSes based on μ-kernels were proposed in the 80s
- μ-kernels try to solve some of the major concerns on current systems:
  - Security
  - Fault tolerance
- But μ-kernels have costs:
  - Impose a more intensive use of *Inter-Process Communication* (IPC)
  - IPC is costly on current architectures

Background and Motivation

Why μ-kernels are so desirable?

- Improve security and fault tolerance by:
  - Unloading most of the code to user-level $\rightarrow$ Smaller *TCB*
  - User-level code more easily respawned when it is detected to fail
  - Processes are a good unit of isolation
- Naturally stimulate modularity, and ease software development
- Provide mechanisms without imposing policies
  $\rightarrow$ Different OS strategies coexisting in a unique system

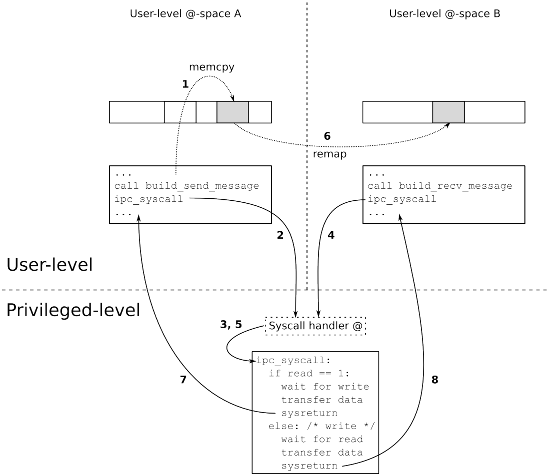
But why are they not so widely used?

- Applications use cross address-space IPC (through *system-calls*)
- Liedtke showed in '94 that IPC performance was driven by:
  - Privilege-level switch
  - Address space switch
  - Message data copy

We reevaluated these costs for machines from PII to Core2Duo

Background and Motivation

IPC Implementation



IPC costs come from:

- Privilege-level switch: *ipc_syscall*
- Address space switch: *sysreturn*
- Message data copy: *memcpy*

Capability-based OSes have nice properties, but stress the use of IPC.

Architectural Analysis

- Privilege-level switches
  - System calls require a privilege level switch
  - A privilege-level switch is triggered in an architecture-dependant way
  - Different privileged instructions could be mixed in the pipeline
- Address space switches
  - Each process has its own virtual address space
  - Kernel switches address space on every IPC between two processes
    → *Small spaces* avoid address space switch (TLB and cache flush) in some special cases (independently devised by L4 and EROS)
  - L1 cache can be addressed either *physically* or *virtually*
- Memory copy
  - Long IPC messages require the kernel to copy data from the sender's memory into a receiver's buffer (short messages are inlined into GPR)
  - A single memory copy is enough by temporarily mapping the receiver's buffer into the sender's address space
  - Several techniques exist to speed-up the copy itself

Architectural Analysis

- Privilege-level switches
  - System calls require a privilege level switch
  - A privilege-level switch is triggered in an architecture-dependant way
  - Different privileged instructions could be mixed in the pipeline
- Address space switches
  - Each process has its own virtual address space
  - Kernel switches address space on every IPC between two processes

    → *Small spaces* avoid address space switch (TLB and cache flush) in some special cases (independently devised by L4 and EROS)
  - L1 cache can be addressed either *physically* or *virtually*
- Memory copy
  - Long IPC messages require the kernel to copy data from the sender's memory into a receiver's buffer (short messages are inlined into GPR)
  - A single memory copy is enough by temporarily mapping the receiver's buffer into the sender's address space
  - Several techniques exist to speed-up the copy itself

Architectural Analysis

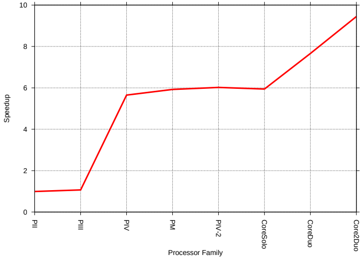
- Privilege-level switches
  - System calls require a privilege level switch
  - A privilege-level switch is triggered in an architecture-dependant way
  - Different privileged instructions could be mixed in the pipeline
- Address space switches
  - Each process has its own virtual address space
  - Kernel switches address space on every IPC between two processes
    → *Small spaces* avoid address space switch (TLB and cache flush) in some special cases (independently devised by L4 and EROS)
  - L1 cache can be addressed either *physically* or *virtually*
- Memory copy
  - Long IPC messages require the kernel to copy data from the sender's memory into a receiver's buffer (short messages are inlined into GPR)
  - A single memory copy is enough by temporarily mapping the receiver's buffer into the sender's address space
  - Several techniques exist to speed-up the copy itself

Evaluation

Experimental Environment

- Bare-metal measurements: implemented a minimal kernel and a set of user-level applications
- Costs are normalized to the SPECint2006 speedup on each machine (comparative slowdown)
- Three experiment configurations, depending on actions taken before test:

hot	:	no special action
data cold	:	flush L1 data cache
cold	:	flush L1 caches (i+d)



Code	CPU	Clock	Stages	Width	L1$ (i+d)	L2$	Year
PII	Deschutes	400MHz	14	4	16+16KiB	512KiB	1998
PIII	Katmai	450MHz	14	4	16+16KiB	512KiB	1999
PIV	Presscot 3,0E	3.00GHz	31	3	12+16KiB	1MiB	2004
PM	Dothan 740	1.6GHz	∼14	3	32+32KiB	2MiB	2005
PIV-2	Cedar Mill 631	3.00GHz	31	3	12+16KiB	2MiB	2006
CoreSolo	Yonah T1300	1.66GHz	12	4	32+32KiB	2MiB	2006
CoreDuo	Yonah T2600	2.16GHz	12	4	32+32KiB	2MiB	2006
Core2Duo	Conroe E6600	2.4GHz	14	4	32+32KiB	4MiB	2006

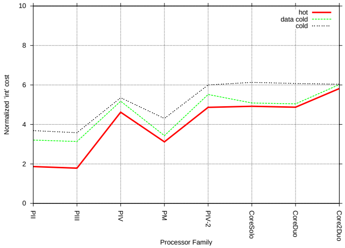
PII, PIII L2$: Off-chip caches working at 50% of CPU-speed
PIV, PIV-2: Use a Trace Cache

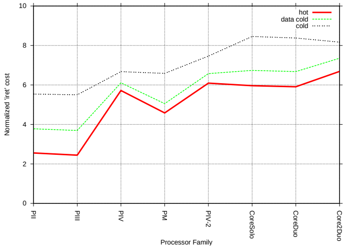
Evaluation

Privilege Level Switches

`int / iret`

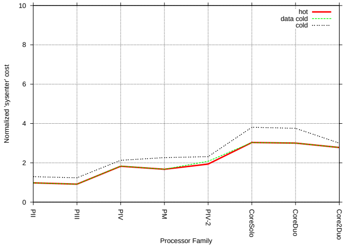
—*PII-PIII*: On *cold* and *data cold*, L2 cache is outside the chip
—*PIV, PIV-2, CoreSolo, CoreDuo*: Besides having deeper pipeline, PIVs have higher clock frequency
—*CoreDuo, Core2Duo*: Pipeline stages increased more than clock speed

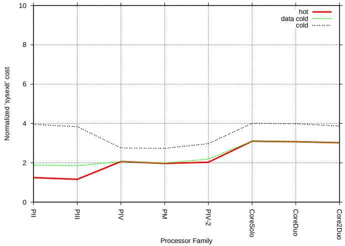




`sysenter / sysexit`

+*All*: Use registers instead of memory





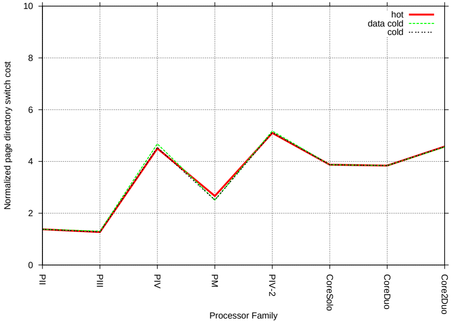
Overall

—*All*: Increased cost on every family, relative to pipeline depth and width
—*All*: *cold* is related to L1$ state
—*PII,PIII*: Our timestamping function on *exit* misses outside the chip (L2$)

Evaluation

Caches and TLBs

− *All*: Cost is related to pipeline depth and width
− *PIV*,*PIV-2*: Trace cache is flushed
+*PM*: Lower cost than Core processors with deeper pipeline (less FUs)
+*All*: Cost is independent of L1D$ state



Kernel pages are pinned (*global* bit)

Evaluation

Memory Copy

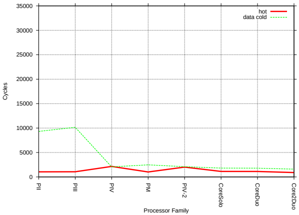
- 4, 8 and 16KiB are typical sizes read from network or disk
- ICache does not affect this tests
- `cpuid` inserts a barrier to wait for memory copies
- Results are kept in real CPU cycles

Overall

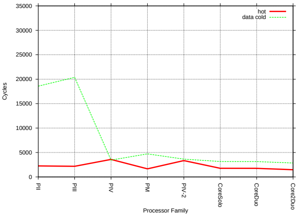
– *PII,PIII*: *Data cold* access to small off-chip L2$
– *PIV,PIV-2*: Higher cache latency
– *PIV,PIV-2*: We speculate *data cold* performs like *hot* because of the prefetching implemented in the chip

16Kib

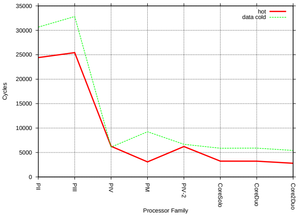
– *PII-PIV,PIV-2*: Message does not fit in L1$



4KiB



8KiB



16KiB

Conclusions (1/2)

- IPC is a critical part of μ-kernels (which provide greater security and fault tolerance), and even more on capability-based OSes
- Despite algorithmic optimizations, IPC heavily relies on architecture support for privilege and address space switches
- IPC-related mechanisms costs do not improve at the same pace of the horsepower in every new microarchitecture:
  - Privilege level switch
    - `int` / `iret`: evolution of the relative slowdown up to 6x
    - `sysenter` / `sysexit`: evolution of the relative slowdown up to 2x
  - Address space switch
    - Cost is driven by the number of in-flight instructions
    - Imposes later costs on flushed cache and TLB entries
    - Evolution of the relative slowdown up to 3x
  - Memory copy
    - Memory copies have been improved due to bigger caches
    - The only ways to improve are bigger L1$, smaller latencies and data prefetching

Conclusions (2/2)

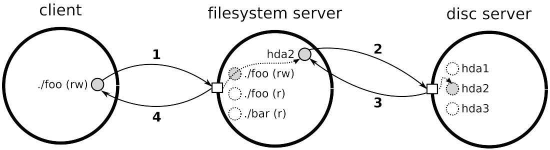
- The IPC performance is very architecture-dependant (we tested IA-32)
- The need of IPC performance depends on the demanding application (SPEC is CPU-bound)
- General applications' performance is still far from being determined by system calls, but they are approaching

Thanks

Questions?

Capability-based OSes

Capability : Kernel-protected token whose possession gives the holder the ability to invoke an operation (with a given set of permissions) on an object implemented by either another process or the kernel ⟶ **One or more IPCs**



- All process communication goes through capabilities
- Provide strong security (process needs to hold a capability to invoke)
- Capabilities can be revoked
- Enforce the *Principle of Least Privilege* (programmers are lazy)
- No need for the super-user "backdoors"
- Can be mathematically reasoned about
  (by analyzing a process' transitive access to other processes)