The SNIA NVM Programming Model: Latest Developments and Challenges

Andy Rudoff, Intel Corporation
“Programming Model” – Four meanings (at least)
Programming Model: SW Interface to HW
Result:
Persistent Memory hardware accessed like memory (cache coherent).
Described by ACPI on x86.
Programming Model: Instruction Set Architecture

- MOV
- CLWB
- persistent domain

Memory Subsystem

L3

Core
L1 L1
L2

Core
L1 L1
L2

NVDIMM
Programming Model: Instruction Set Architecture

Result:
Stores flushed from CPU cache, globally-visible ➔ Persistent (on x86)
Programming Model: Exposing to Applications
SNIA NVM Programming Model
Memory-Mapped Files

- Application
  - Standard File API
  - Load/Store

- PM-Aware File System
  - MMU Mappings

- NVDIMM
  - No Page Cache
  - Result: Direct Access (DAX)
Programming Model: The Programmer Experience

Result:
Safer, less error-prone, idiomatic in common languages
What’s Next for the SNIA NVM Programming Model?
Active Work

- More details on flushing to persistence
  - Includes flushing to remote persistence
- Continue to refine the error model
- Transactions
- APIs?
Why is “Flushing” a Focus?
Memory-Mapped Files

- Standard Flush
- `msync()`
- `FlushViewOfFile()`
- `FlushFileBuffers()`
Memory-Mapped Files

30-year-old mechanism:
Just as error-prone today
as it was 30 years ago

Standard Flush
msync()
FlushViewOfFile()
FlushFileBuffers()
Memory-Mapped Files

Standard Flush

\texttt{msync()}

\texttt{FlushViewOfFile()}

\texttt{FlushFileBuffers()}

30-year-old mechanism: Just as error-prone today as it was 30 years ago

Not Atomic
Java PersistentSortedMap

PersistentSortedMap employees = new PersistentSortedMap();

... employees.put(id, data);

No flush calls.
Transactional.
Java library handles it all.

See “pilot” project at: https://github.com/pmem/pcj
What Lies Between:

Memory-Mapped Files

Standard Flush

msync()
FlushViewOfFile()
FlushFileBuffers()

High-Level Language Support

PersistentSortedMap
employees.put(id, data);
NVM Libraries: pmem.io
C/C++ on Linux and Windows

- Open Source
  - [http://pmem.io](http://pmem.io)
  - libpmem
  - libpmemobj
  - libpmemblk
  - libpmemlog
  - libvmem

NVM Libraries

- User Space
- Kernel Space
- Application
- Load/Store
- Standard File API
- PM-Aware File System
- MMU Mappings
- NVDIMM
## Types of Store Barriers

<table>
<thead>
<tr>
<th>Barrier Type</th>
<th>Current Status</th>
</tr>
</thead>
</table>
| **Standard API**    | Fully specified  
|                     | Fully supported:  
|                     |   • Linux (ext4, XFS)  
|                     |   • Windows (NTFS)       |
| **Optimized Flush** | Specified, but evolving (ask when safe)  
|                     |   • Linux: **unsafe** except Device DAX  
|                     |     • (and new file systems)  
|                     |   • Windows: safe         |
| **Remote Flush**    | Proposals under discussion  
|                     | (works today with extra round trip)                                    |
| **Deep Flush**      | Upcoming Specification                                           |
| **Transactions**    | Built on above via libraries and languages  
|                     | **Much more language support to do**                                |
Summary

SNIA NVM Programming Model
- Both Windows and Linux support basic model
- Evolving to meet ongoing needs of community
- [http://snia.org/nvmp](http://snia.org/nvmp)

Interesting work ongoing on remote PM
- Expect specs this year in SNIA & other forums

Continued refinement of the SNIA model in TWG
- Flush semantics
- Error semantics
- Transactions, and more…

Ongoing community work on libraries & languages
- [https://github.com/pmem/nvml](https://github.com/pmem/nvml)
- [http://pmem.io](http://pmem.io)