LPC’24 X86 Micro-conference

Revisit XSAVE

Lessons from 20 Years of Processor Context State Management

Chang Bae, Intel

LINUX PLUMBERS CONFERENCE | Vienna, Austria
Sept. 18-20, 2024

Questions

- What is the XSAVE architecture?
- How has it performed over time?
- Why consider an alternative?

Questions

- What is the XSAVE architecture?
  - What is its approach?
  - How was it adopted by Linux?
  - How has the architecture evolved?
- How has it performed over time?
- Why consider an alternative?

XSAVE Architecture

- Monolithic approach to context management
  - A generic way to save/restore extended states
  - Primary use case: *context switching*
  - Memory layout:
    - Extension to the format used by FXSAVE
    - Defined by hardware (XSAVE format)

XSAVE Architecture (cont.)

- XSAVE format adopted as part of ABI
  - Applied in in signal and core-dump/ptrace frames
  - Arguments (historically) [discussion]:
    - Offsets are fixed and discoverable for the layout
    - No need for a separate descriptor for the layout
  - New extended states:
    - Must be managed by XSAVE to be included in the frame
    - Otherwise, the kernel fills the states according to the XSAVE format

XSAVE Architecture (cont.)

- Optimizations
  - Performance optimizations (hardware-driven):
    - Skip saving initial states: INIT optimization
    - Save only modified states: Modified optimization
  - Size optimizations:
    - Save selected states and compact the buffer
    - Dynamically expand the buffer only when detecting first state usage

XSAVE and Feature Adoption History

Years	XSAVE Variants	Features
2023		CET: Control-flow Enforcement Technology
2022	Compact for guest kernel (XSAVEC)	PASID: Process Address Space Identifiers
2021	Dynamic states (XFD)	AMX: Advanced Matrix Extensions
2020	Supervisor states	LBR: Last Branch Record
2016	Compaction+optimization (XSAVES)	PKRU: Protection Key Feature
2014		AVX-512: Advanced Vector Extensions 512 MPX: Memory Protection Extensions
2010	Optimization (XSAVEOPT)	
2009		AVX: Advanced Vector Extensions
2008	Introduction (XSAVE)	
1999	Predecessor (FXSAVE)	SSE: Streamline SIMD Extension

Questions

- What is the XSAVE architecture?
- How has it performed over time?
  - Is the approach still effective?
  - Has it scaled efficiently?
  - Are the optimizations still relevant?
- Why consider an alternative?

Cases Against Monolithic Design

- Protection Key Features (PKRU)
  - Need to keep the current state always valid
  - switch_to() and flush_thread() write the value eagerly [patch]
  - Thus, separately managed in a dedicated storage [series]
- Supervisor States
  - Need to read/modify the state from the XSAVE buffer
  - This retrieval can be costly to find the exact location in XSAVE buffer due to the compaction logic

Cases Against Monolithic Design (cont.)

- Mitigation for Supervisor States
  - CET: Control-flow Enforcement Technology
    - Instead of retrieving, restore the state directly for modify [patch]
- However, managing separately would simplify these operations

- Takeaway: This monolithic switching is not always beneficial

Cases Against XSAVE format as ABI

- Inefficient ABI format
  - The context layout is fixed and universal across tasks
  - This model was viable until disruptive new states emerged
  - Some new states are large but not always in use, leading to inefficiencies
- Mitigations
  - Selective expansion through permission-based usage control
  - Alternatively, consider a new ABI format, more flexible ABI format

- Takeaway: The static ABI format is inefficient for dynamic usages

Review Hardware-Driven Optimizations

- Fragile ‘Modified’ optimization
  - Hardware-driven optimization
  - Modified optimization is effective for consecutive context saves

Questions

- What is the XSAVE architecture?
- How has it performed over time?
- Why consider an alternative?
  - What should be the key considerations moving forward?

Summary of Retrospection

- Monolithic Approach vs Heterogenous State Nature
  - Some features require state switches more frequently than scheduler
- Uniform and Unified Storage Complexity
  - Retrieving states for inactive tasks is fragile and costy
- Inflexible Context Layout
  - Too static for dynamic state usage models
- Hardware-Driven Optimizations
  - Reliance on a single buffer model

Closing: Consideration of Alternatives

- Minimum Architectural Requirements
  - Flexibility to save/restore individual states independently
  - Allow the kernel to define the context layout format
- Challenges
  - ABI: Transition away from the XSAVE format, introducing a new, software-defined format
  - Significant rework might be required to shift from monolithic state management to a disaggregated state model

Discussions