When Poll is better than Interrupt

FAST’12
Summary

• Future SSD devices will show short I/O response time
  – Thanks to high-performance next-generation NVM
• Traditional I/O path based on “interrupt” mechanism can be ineffective on such fast NVM SSDs
  – This actually works well for rotation devices or even NAND SSDs
  – This is referred to as “Asynchronous I/O completion” model
• Synchronous I/O completion model based on “polling” can be more effective on such fast NVM SSDs
• This paper presents empirical comparisons of two models
• This paper analyzes the latencies
Asynchronous I/O Completion

(1) I/O thread running on front-end submits “I/O” to kernel
(2) Kernel thread submits “device command” to SSD
  – After submitting “device command”, kernel thread releases CPU
  – Other user threads can utilize the free CPU during SSD process
(3) SSD notifies kernel the completion via “interrupt”
(4) Kernel thread processes completion and wakes up I/O thread
Synchronous I/O Completion

1. I/O thread running on front-end submits “I/O” to kernel
2. Kernel thread submits “device command” to SSD
   - After submitting “device command”, it polls the completion
   - During the polling, kernel thread holds on CPU resource
   - No other user/kernel thread cannot use CPU
3. Once completion is detected, I/O thread resumes
Experimental Setup

• DRAM-based prototype storage device
  – A fast future SSD model
  – PCIe (ver.2, x8 lanes) bus
  – NVMe interface

• Linux OS
  – Block device driver
Single I/O Latency

- **C-state**: when CPU enters low-power mode during SSD processing
- Average latency of a 512B/4KB single I/O
- Hardware device part includes “interrupt delivery latency”
- Operating system part excludes “overlapping time H/W device”
- **Synchronous model shows better latency than asynchronous one**
  - Note that H/W device time is very short
Latency Breakdown

- **Asynchronous model**
  - \( T_d \) (4.1us), \( T_a \) (4.9us), \( T_b \) (1.4us), \( T_u \) (2.7us)
  - **OS cost** = \( T_a + T_b = 6.3 \text{us} \)

- **Synchronous model**
  - \( T_d \) (2.9us), **OS cost** (4.4us)
More on Asynchronous Model

• (1) **Interrupt overhead**
  – Interrupt brings extra works, increasing CPU utilization
  – H/W interrupt has the highest priority
  – *Synchronous model does not incur interrupt*

• (2) **Cache & TLB pollution**
  – Task scheduler eagerly finds any runnable thread
  – 2.7us (8000 cycles) is enough to pollute cache by other user threads
  – *Synchronous model continuously maintain the context*

• (3) **CPU power-state complication**
  – Modern processor may enter “power-saving state” when not loaded
  – Transition among states incur additional latency (2us)
  – *Synchronous model prevents CPU from entering other state*