Asynchronous Page Fault and SDEI Virtualization for ARM64

Gavin Shan <gshan@redhat.com>
KVM Forum 2021

Overview

- Motivation
- Current Status
- Requirements
- Async PF on ARM64
- SDEI Virtualization
- Performance
- Conclusions
- Review & Community Support

Motivation

- Asynchronous Page Fault (Async PF) improves the guest's parallelism significantly, by rescheduling other processes for execution in the guest while the host resolves stage-2 page fault.
- The guest's performance benefits from the improved parallelism by Async PF during post-copy live migration.
- Async PF may serve other purposes, like relaying FUSE's errors from host to guest.

Current Status

- Async PF is introduced to X86 initially. (https://www.linux-kvm.org/images/a/ac/2010-forum-Async-page-faults.pdf)

- Virtiofsd uses Async PF to relay FUSE’s errors from host to guest on X86.

- Async PF is supported on S390, but not available on ARM64 yet.

Requirements

- Async PF depends on two notifications: page-not-present and page-ready.
- Page-not-present notification is synchronously sent from host to guest before the stage-2 page fault is to be resolved. The guest reschedules other processes for execution while the stage-2 page fault is being resolved on the host.
- Page-ready notification is asynchronously sent from host to guest after the stage-2 page fault is resolved on the host. The guest reschedules the previous faulting process for execution.
- The data block, shared between host and guest, is updated to differentiate the notifications and identify the specific Async PF by a unique token.
- Configuration and live migration are supported by using the control data block.

Async PF on ARM64

- Page Fault vector (14) is used to deliver page-not-present notification on X86. The same mechanism is unavailable on ARM64 due to the limited space in ESR_EL1.

- SDEI (Software Delegated Exception Interface) is leveraged to deliver the page-not-present notification synchronously on ARM64.

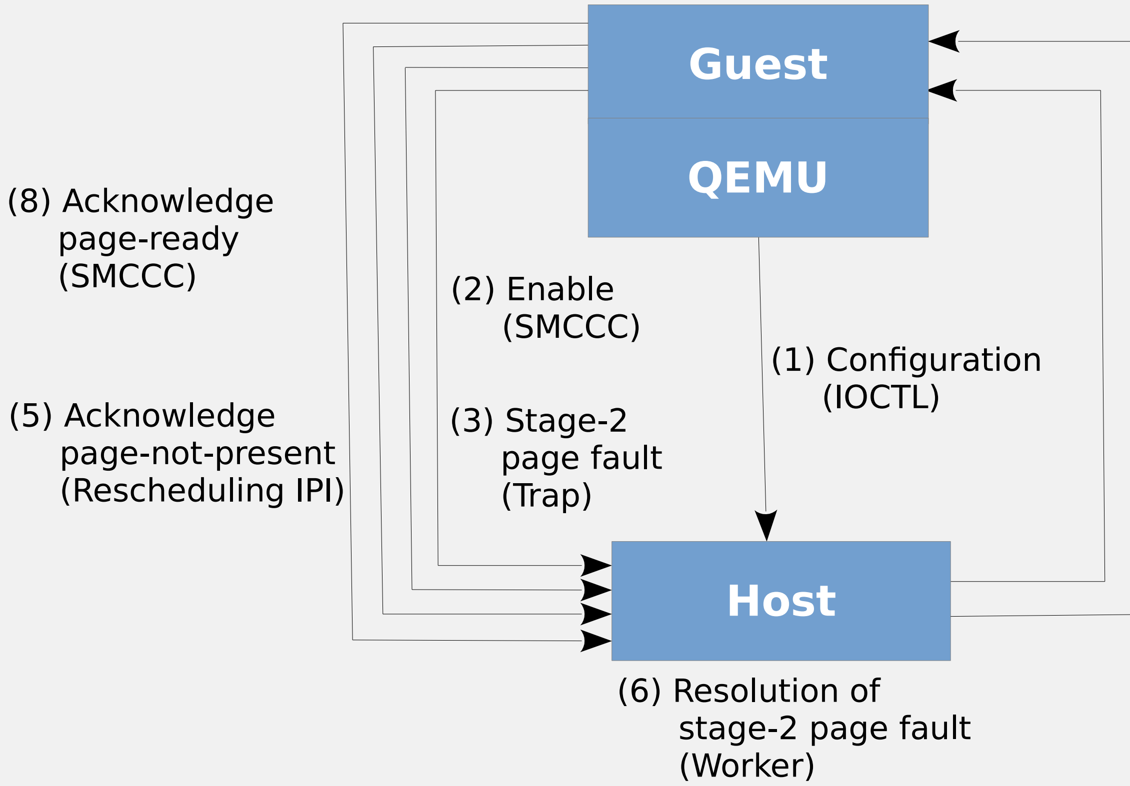
- Private Peripheral Interrupt (PPI) is used to deliver the page-ready notification in asynchronous fashion, similar to X86's implementation where vector 243 is used.

- The shared data block is updated on ARM64 when the notifications are delivered and acknowledged, exactly same to the implementation on X86.

- The control data block is accessed by SMCCC (Secure Monitor Call Calling Convention) to configure Async PF on ARM64, but emulated MSRs are reserved for it on X86.

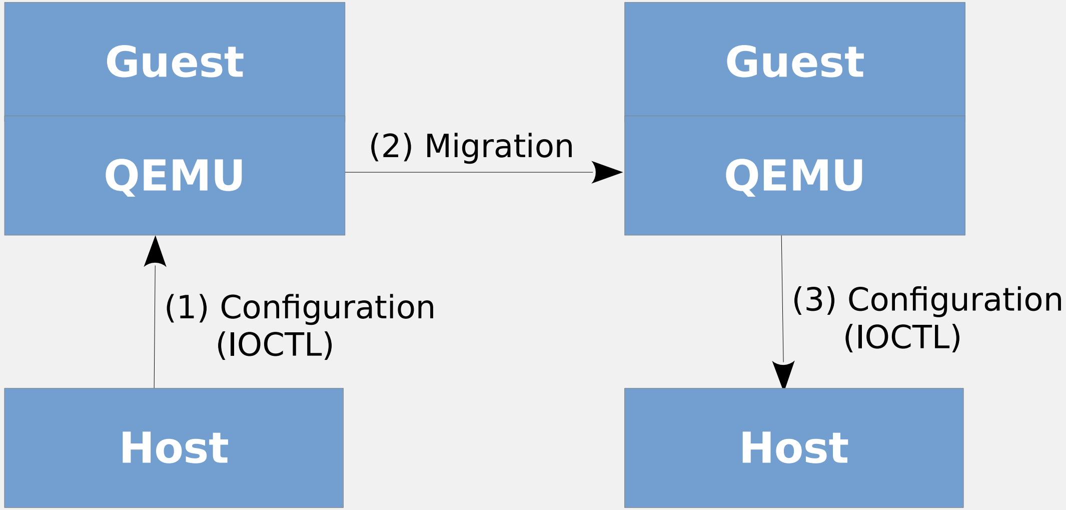
- The control data block is also accessed by IOCTL commands from userspace to support live migration on ARM64.

Async PF on ARM64



The control data block is updated when Async PF is configured, while the shared data block is updated when page-not-present and page-ready are delivered and acknowledged.

Migration of Async PF



The state of Async PF, residing in the control data block, is retrieved on source VM and restored on target VM through IOCTL interface.

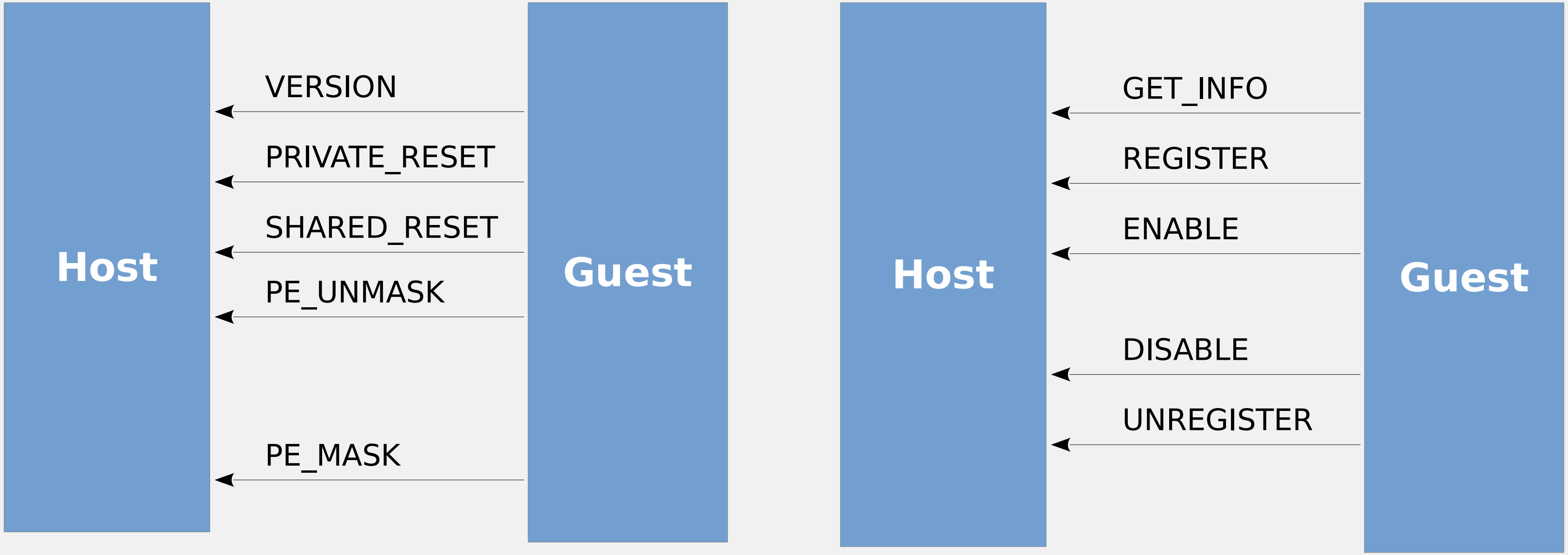
SDEI

- Abbreviation of Software Delegated Exception Interface, defined by DEN0054A (https://developer.arm.com/documentation/den0054/latest). It provides a mechanism for registering and servicing system events from hypervisor. The interface is offered by hypervisor to guest OS.
- The service is delivered with SDEI event, identified by unique event number. The SDEI event is delivered to guest OS immediately regardless of guest's state. It is not maskable by irq_disable(), similar to x86's NMI in this regard.
- SDEI events are classified into shared and private events. The shared event is owned by multiple PEs (Processing Elements) and delivered to one of them. The private event is only visible and owned by one PE.
- The private SDEI event and interrupt (PPI) are used by Async PF for page-not-present and page-ready notification delivery.

SDEI and SMCCC

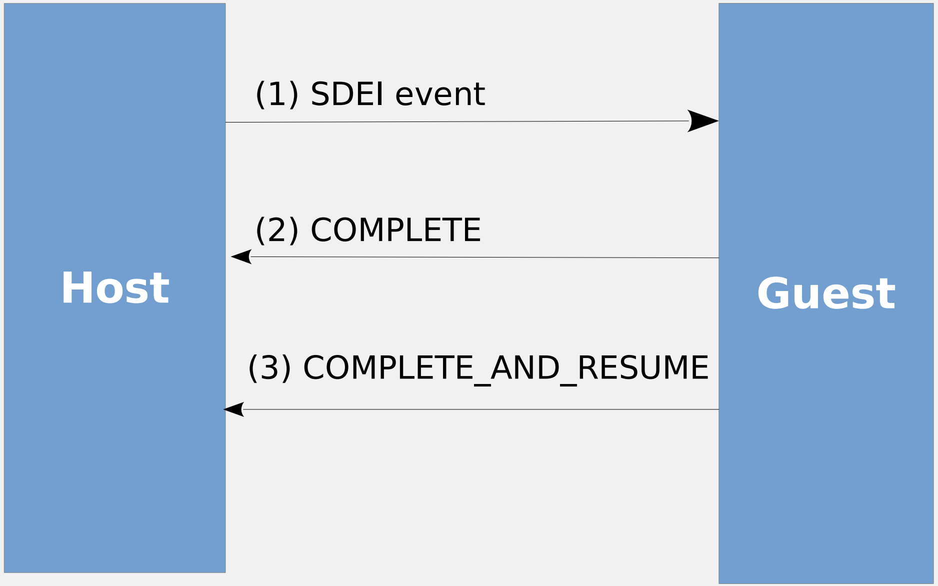
- SMCCC (Secure Monitor Call Calling Convention), defined by DEN0028D. (https://developer.arm.com/documentation/den0028/latest)
- Define a common calling mechanism to be used with the Secure Monitor Call (SMC) and Hypervisor Call (HVC) instructions.
- The HVC instruction is used to generate a synchronous exception, which is handled by a hypervisor running in EL2. The arguments and return values are passed in X0-X17 registers. The service is identified by the argument in X0.
- SDEI falls in the category of SMCCC's Standard Service Calls whose function ID is 0x4.

SDEI Interface



All these calls in control path are delivered through SMCCC.

SDEI Interface



(1) The following registers are saved and then the registered handler is invoked in guest.

x0 – x17, PC, PState

(2) COMPLETE and COMPLETE_AND_RESUME issued through SMCCC, notify the received SDEI event has been handled.

(3) COMPLETE_AND_RESUME schedules the pending interrupts immediately, while COMPLETE does not.

Performance of Heavy Swapin

The test program writes to all available memory while calculation thread might be running in parallel.

vCPU: 1 Memory: 1024MB   cgroupv2.limit: 512MB
Command: testsuite test async_pf -l 1 [-t] -q
Asynchronous page faults: 55000

Time-	Calculation-	Time+	Calculation+	Output
13.214s		14.072s		+6.4%
13.329s		14.150s		+6.1%
13.433s		14.159s		+5.4%
13.535s		14.222s		+5.0%
13.553s		14.328s		+5.7%
24.016s	1806m	15.294s	1042m	-36.3%
24.264s	1826m	15.542s	1040m	-35.9%
24.579s	1835m	15.616s	1033m	-36.4%
27.230s	2257m	15.738s	1084m	-42.2%
27.236s	2278m	15.862s	1071m	-41.7%

~5% more time to finish the job due to the overhead introduced by Async PF.

~40% less time to finish the job, significant improvement in terms of parallelism or interactivity.

Performance of Live Migration

Performance in post-copy live migration scenario.

vCPU: 1 Memory: 1024MB   cgroupv2.limit: unlimited
Command: testsuite test async_pf -l 50 [-t] -q
Migrate.total_time: ~1.6s

Time-	Calculation-	Time+	Calculation+	Output	
8.910s		8.610s		-3.3%	
8.965s		8.620s		-3.8%	
9.125s		8.659s		-5.1%	
9.217s		8.685s		-5.7%	
9.281s		8.685s		-6.4%	
19.483s	1581m	19.346s	1657m	-0.7%	+67.1%
19.838s	1611m	19.976s	1711m	+0.7%	+68.3%
19.976s	1630m	20.183s	1733m	+1.0%	+65.9%
20.441s	1707m	20.193s	1742m	-1.2%	+41.7%
20.900s	1763m	20.439s	1781m	-2.2%	+42.1%

~3% to ~6% less time needed to finish the job because the improved interactivity by Async PF helps to decrease the page dirty rate and post-copy requests.

The time used to finish the job is comparative, but ~41% to ~68% more calculation capability (speed) is offered during the live migration by the improved interactivity from Async PF.

Performance of Live Migration

The post-copy request count is dropped from 1065 to 782 because of the improved interactivity by Async PF. The calculation capacity (speed) is improved by ~70% during the period of live migration.

	Time-	Time+	Time-	Time+
Memory write:	9.151s	8.856s	20.400s	20.439s
Calculation:			1684m	1781m
total time:	1687ms	1762ms	1672ms	1664ms
downtime:	10ms	11ms	10ms	10 ms
setup:	3ms	3ms	3ms	4 ms
transferred ram:	914646KB	916202KB	914682KB	916214KB
throughput:	4449.92mbps	4267.47mbps	4490.09mbps	4522.03mbps
dirty sync count:	2	2	2	2
page size:	4KB	4KB	4 KB	4KB
pages-per-second:	153009	153495	152873	159403
post-copy request count:	1065	782	243	321

Conclusions

- Async PF is significantly beneficial to the guest’s parallelism or interactivity. ~40% improvement of the parallelism in the heavy swapin scenario.

- It is also tremendously beneficial (41% to 68%) to the guest’s performance in the period of post-copy live migration.

Review & Community Support

- Updated patchset is available online

  https://lkml.org/lkml/2021/8/14/415 # Support SDEI Virtualization (v4)
  https://lkml.org/lkml/2021/8/14/443 # Support Asynchronous Page Fault (v4)

  https://github.com/gwshan/linux # kvm/arm64_sdei
  https://github.com/gwshan/linux # kvm/arm64_apf
  https://github.com/gwshan/qemu # kvm/arm64_sdei
  https://github.com/gwshan/qemu # kvm/arm64_apf

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHat

youtube.com/user/RedHatVideos