

# Long live asynchronous page fault!

KVM FORUM 2020

Vitaly Kuznetsov <[vkuznets@redhat.com](mailto:vkuznets@redhat.com)>

Vivek Goyal <[vgoyal@redhat.com](mailto:vgoyal@redhat.com)>

# About us

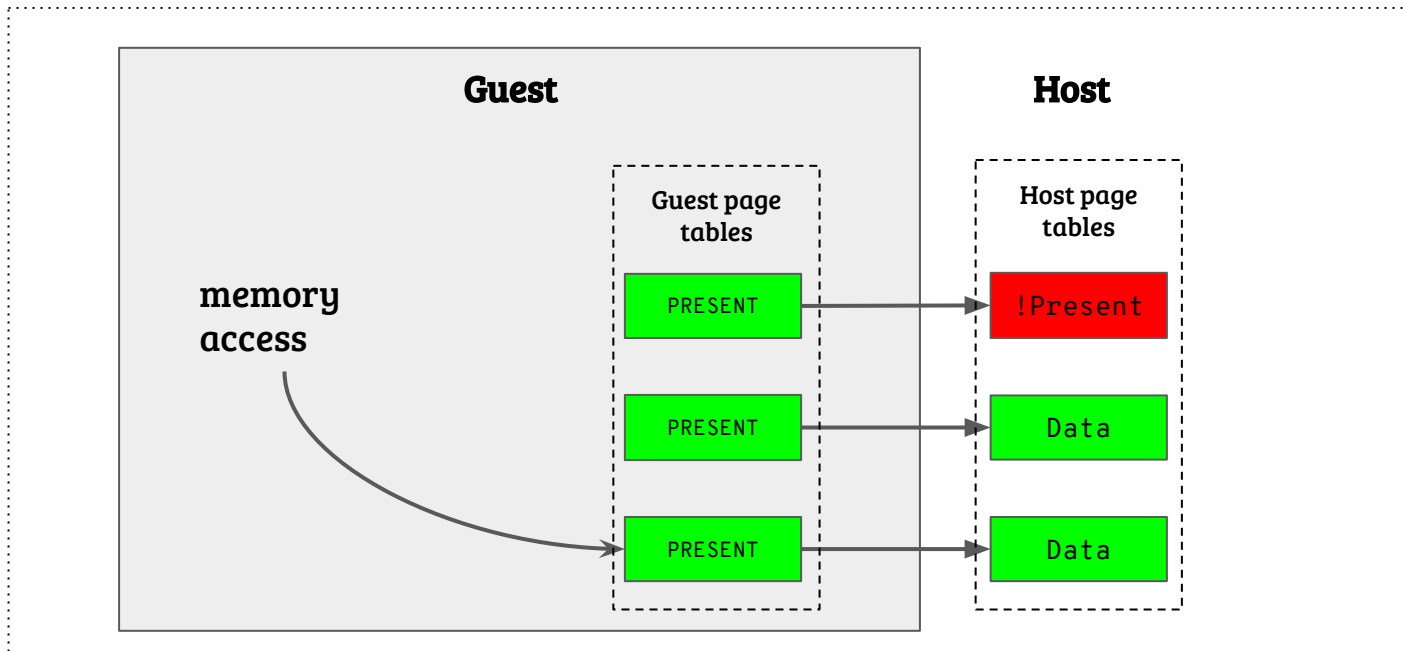
## Vitaly Kuznetsov

- **KVM contributor and reviewer**
- **Areas of interest:**
  - **PV features**
  - **Hyper-V emulation, Windows guests**
  - **Nesting**

## Vivek Goyal

- **Linux Kernel Developer**
- **Contributions in virtiofs, overlays, kexec/kdump, block layer.**

## Which #PFs are we interested in?



## What normally happens in these circumstances?

1. **Guest vCPU is blocked**
2. **Page fault condition is (hopefully) resolved on the host, e.g. the missing page is brought back from swap.**
3. **Guest vCPU resumes execution**


## What normally happens in these circumstances?

1. Guest vCPU is blocked
2. **Page fault condition is (hopefully) resolved on the host, e.g. the missing page is brought back from swap. → *This can take really long, but the physical CPU may actually be idle.***
3. Guest vCPU resumes execution

## Can we still do something useful on the vCPU while we are waiting for the page to be brought from storage?

Two options:

- Let the guest know about the situation so it can run *some other processes* in the meantime → **asynchronous page fault**.
- Synthetically “halt” guest vCPU and hope for rescheduling interrupt to arrive → **synthetic APF halt**.



# Asynchronous page fault: initial implementation (2010)

# Asynchronous page fault: implementation details (original, 2010)

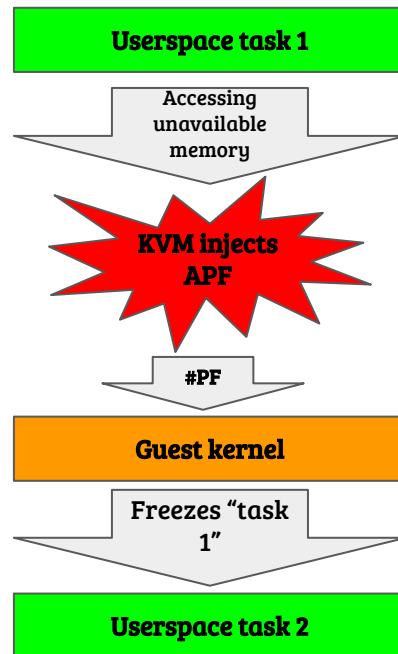
- **Two types of events:**
  - “Page not present”
  - “Page ready”
- **#PF exception is used to deliver the event**
- **Page gets an associated ‘token’ which is passed through CR2**
- **Guest/host shared data structure (“APF reason”) which indicates:**
  - **The fact that the #PF is asynchronous**
  - **Type of the event (page not present, page ready)**



# Asynchronous page fault: how does it work

## “Page not present” flow

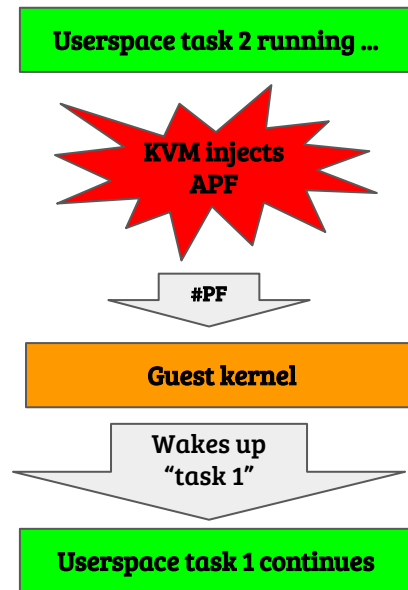
1. A process inside the guest is trying to access a page which is swapped out on the host
2. KVM queues the request to bring the page, assigns a token, sets a flag in the shared data structure indicating ‘page not present’ event.
3. KVM injects #PF to the guest
4. Guest kernel analyzes shared data structure and gets the token from CR2
5. The process which caused APF is being frozen until the corresponding ‘page ready’ even arrives.
6. Guest kernel switches to some other process



# Asynchronous page fault: how does it work

## “Page ready” flow

1. ... eventually the page which previously caused a “page not present” event is swapped in on the host.
2. KVM interrupts the guest, sets a flag in the shared data structure indicating ‘page ready’ event.
3. #PF is being injected to the guest
4. Guest kernel analyzes shared data structure and gets the token from CR2
5. The process which caused APF is being unfrozen, it can now be scheduled again.



## Asynchronous page fault: extensions

- **“Send always” mode**
  - **Allows delivery of APF “page not present” events in CPL=0**
    - **Process causing APF can be the guest kernel itself (CONFIG\_PREEMPT)**
- **Asynchronous page fault delivery in guest mode**
  - **Allows delivery of APF events as PF VMEXITs when the guest is running a nested guest.**

## Asynchronous page fault: design flaws

*Documentation/virt/kvm/msr.rst:*

**Guest must not enable interrupt before the reason is read, or it may be overwritten by another APF. Since APF uses the same exception vector as regular page fault guest must reset the reason to 0 before it does something that can generate normal page fault.**

But can this really be guaranteed?

## Example of a faulty scenario

=> Asynchronous page fault event is injected (#PF)

=> NMI before APF handler reads CR2 and clears APF 'reason'

=> NMI handler accesses user memory and gets real #PF

=> CR2 gets clobbered, ...

=> APF event can't be handled

## Example of a faulty scenario

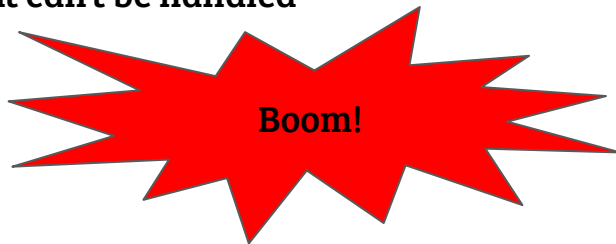
=> Asynchronous page fault event is injected

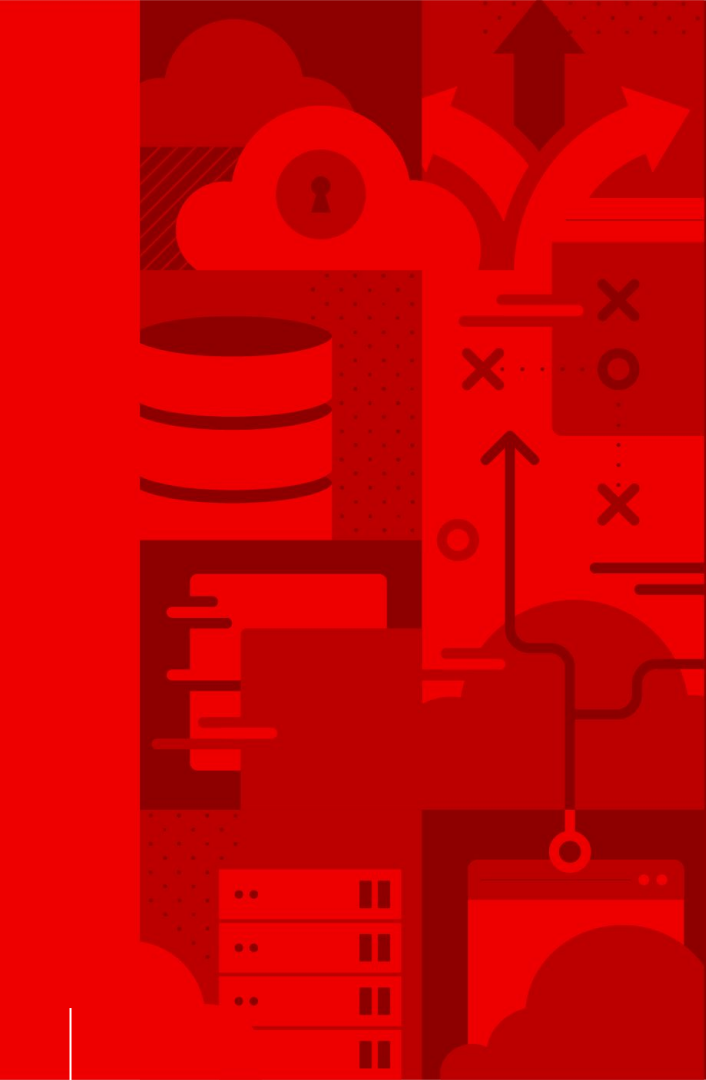
=> NMI before APF handler reads CR2 and clears APF 'reason'

=> NMI handler accesses user memory and gets real #PF

=> CR2 gets clobbered, ...

=> APF event can't be handled





# Asynchronous page fault: updated implementation (2020)

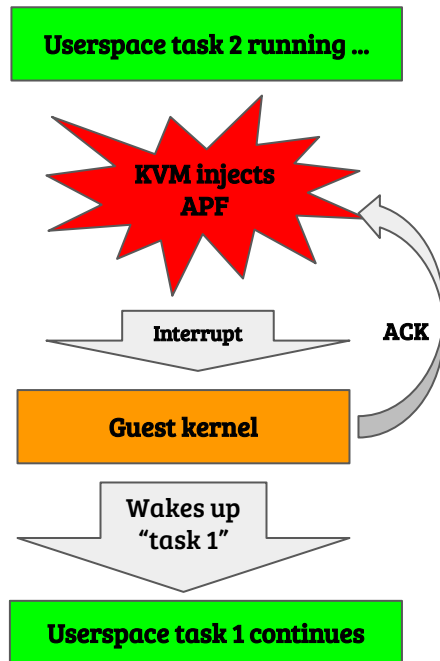
## Asynchronous page fault: updated implementation

- **“Send always” mode for “page not present” was completely deprecated and disabled.**
- **“Page ready” event delivery switched to using interrupts instead of #PF**
  - **Shared data structure was extended with token information (as CR2 is not being used).**
  - **Guest kernel has to support the new delivery mode or APF mechanism is not enabled**
    - **KVM was switched in Linux-5.8, QEMU-5.2 is required**
    - **Guest kernel enablement in Linux-5.9**
- **Plans to switch “page not present” events to #VE/#MC/some other exception (interrupt?).**



## Asynchronous page fault: Updated “Page ready” flow

1. ... eventually the page which previously caused a “page not present” event is swapped in on the host.
2. KVM interrupts the guest and checks if the token slot in the shared data structure is free.
  - In case it is not the event can't be delivered.
3. KVM fills shared data structure with token information
4. An interrupt is injected to the guest, it doesn't have to be handled immediately.
5. In the interrupt handler guest kernel analyzes shared data structure and gets the token from it
6. The process which caused APF is being unfrozen, it can now be scheduled again.
7. Guest frees token slot and indicates “end of message” to KVM so more events can be delivered.





# KVM Page Fault Error Handling

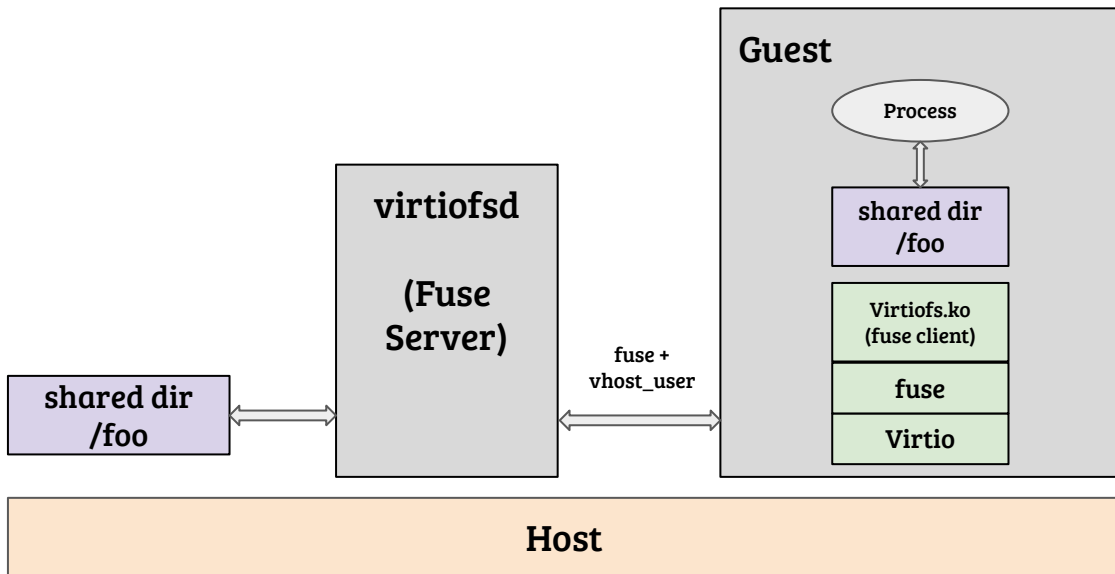
## What if KVM can't resolve page fault successfully

# What if KVM can't resolve page fault successfully

## When can that happen

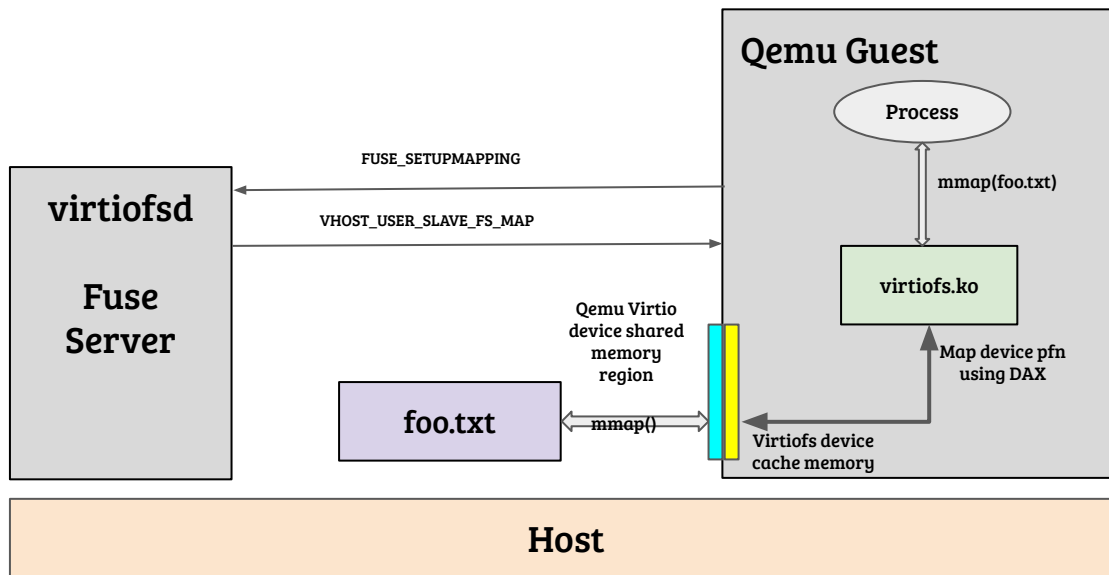
**What if KVM can't resolve page fault successfully**  
**When does that happen**  
**Virtiofs + DAX**

## What is virtiofs



- A pass through file system
- Allows sharing a directory tree on host with guest

## What about virtiofs + DAX



- Host page cache page is directly mapped into guest process
- Qemu does mmap(foo.txt) in device cache region
- virtiofs.ko maps cache region pfn into process using DAX

**What if file foo.txt gets truncated on host and guest does  
a load/store operation from/to truncated page**

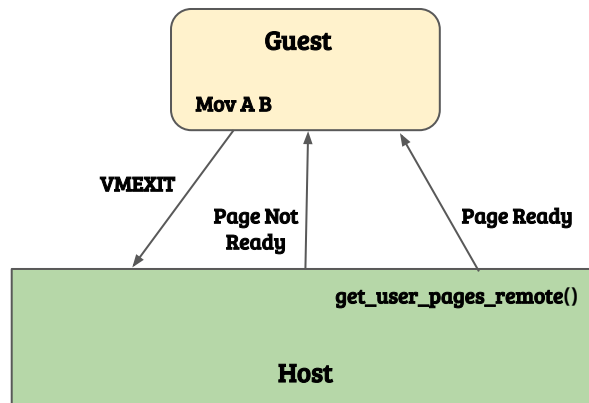


**What if file foo.txt gets truncated on host and guest does a load/store operation from/to truncated page**

- **Sync page fault, exit to qemu with -EFAULT**
- **Async page fault, loop infinitely**

## KVM async page fault error handling

- KVM async page fault code ignores error and always injects “**Page Ready**” event
- Guest retries memory access and kvm injects “**Page Ready**” again.
- And this continues in a loop infinitely
- Need a mechanism to report errors to guest



## Desired Behavior

- **Uniform behavior for sync/async page faults**
- **Posted a patch to fix it**
  - <https://lore.kernel.org/kvm/20200720211359.GF502563@redhat.com/>
- **Normally on host a process gets SIGBUS in such scenarios**
- **Need a way to detect error on host, inject error in guest and send SIGBUS to process**
- **If guest kernel was accessing truncated page, return from memcpy to user space**
  - **Exception Table Fixup**

## Various Proposals

- **Use asynchronous page fault “page ready” interrupt to report error**
  - Works only if “page not present” event was sent
  - Now disabled if `cpl == 0`
  - Can't handle guest kernel access of truncated page
  - Error reporting dependent on async page fault mechanism. Does not work for synchronous faults.
  - Posted patches here
    - <https://lore.kernel.org/kvm/20200616214847.24482-1-vgoyal@redhat.com/>

## Various Proposals

- **Use Machine Check Exceptions**
  - **Generated synchronously for load operation**
  - **On real hardware, stores will not generate synchronous MCE**
  - **Need error reporting both on load and store**
  - **Most of the error handling code written with assumption that only loads will generate synchronous MCE**
  - **There was resistance to create separate code path in MCE handler**

## Various Proposals

- **Use #VE**
  - Available on Intel (x86). What about other arches?
  - Can be used to report “Page Not Present” events as well, hence eliminating #PF races.
  - Somebody is yet to post patches.

## Various Proposals

- **Handle at virtiofs level**
  - **Implement some sort of file lease**
  - **File truncate takes write lease, breaks existing leases and unmaps pages from all other guests**
  - **Truncation proceeds and now guest should not run into the issue of mapped pfn missing**
  - **Does not work if shared directory is modified on host directly. Does not go through virtiofsd.**

# Thank you!



[linkedin.com/company/red-hat](https://www.linkedin.com/company/red-hat)



[youtube.com/user/RedHatVideos](https://www.youtube.com/user/RedHatVideos)



[facebook.com/redhatinc](https://www.facebook.com/redhatinc)



[twitter.com/RedHat](https://twitter.com/RedHat)