Userfaultfd: Post-copy VM migration and beyond

Mike Rapoport
<rppt@linux.vnet.ibm.com>

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements No 645402 and No 688386

Overview

- Userfaultfd is a mechanism for user-space paging and memory tracking
  - Anonymous
  - Hugetlbfs
  - Shared memory
- File descriptor with `ioctl()`'s for control
- `[e]poll()` and `read()` for notifications
- Page fault events (since 4.3)
- Events for virtual memory layout changes (since 4.11)

Use cases

- Userfaultfd-Missing
  - VMs and containers post-copy migration
  - Shared memory robustness
  - Host enforcement for VM memory ballooning
- Userfaultfd-Write-Protect (WIP)
  - Efficient replacement of mprotect + SIGBUS
  - Memory snapshotting
  - Garbage collectors
  - Distributed shared memory

Migration: why?

- Spectacular
- Statefull long-running application with no downtime
  - Hardware upgrades
  - Software upgrades requiring boot
- Load balancing

Migration: how?

- Very simple
  - Save state on source
  - Copy state to destination
  - Restore state on destination
- Memory is the heaviest part
  - Pre-copy vs post-copy

Migration flows

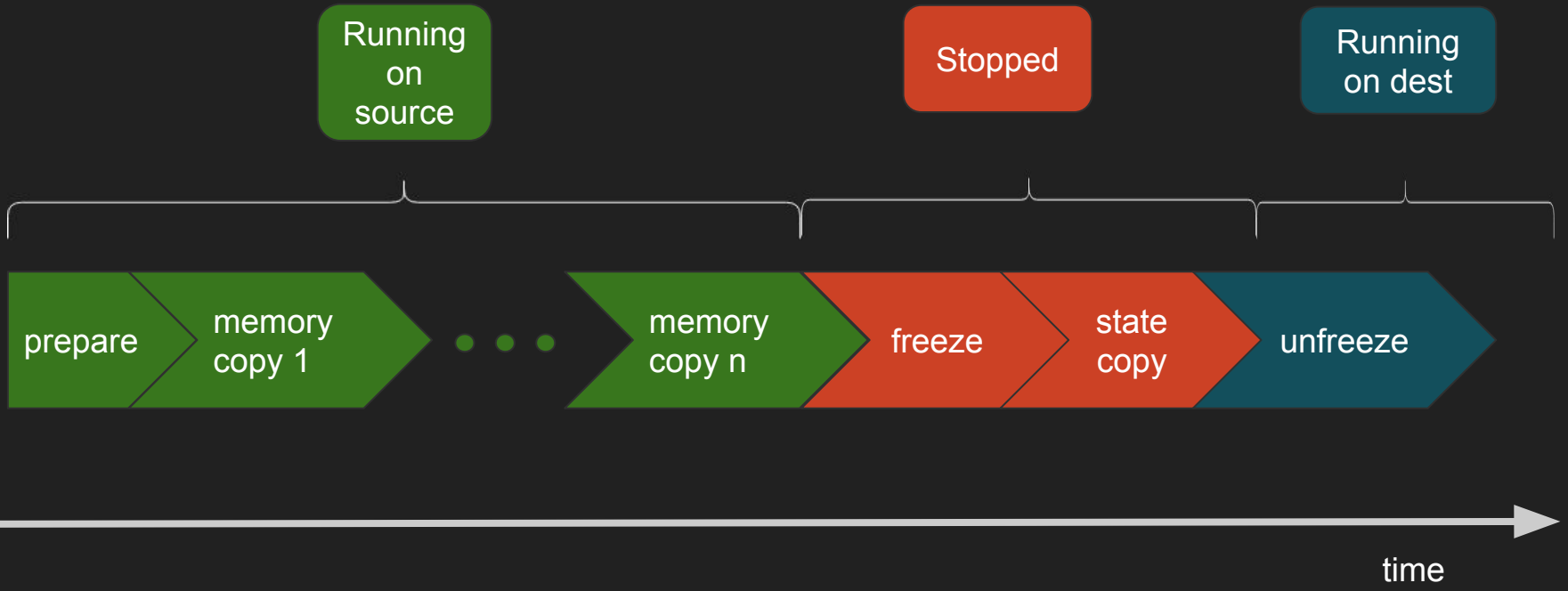
Pre-copy

- Track memory, copy inactive part
- Freeze on source
- Copy state and remaining memory
- Unfreeze on destination

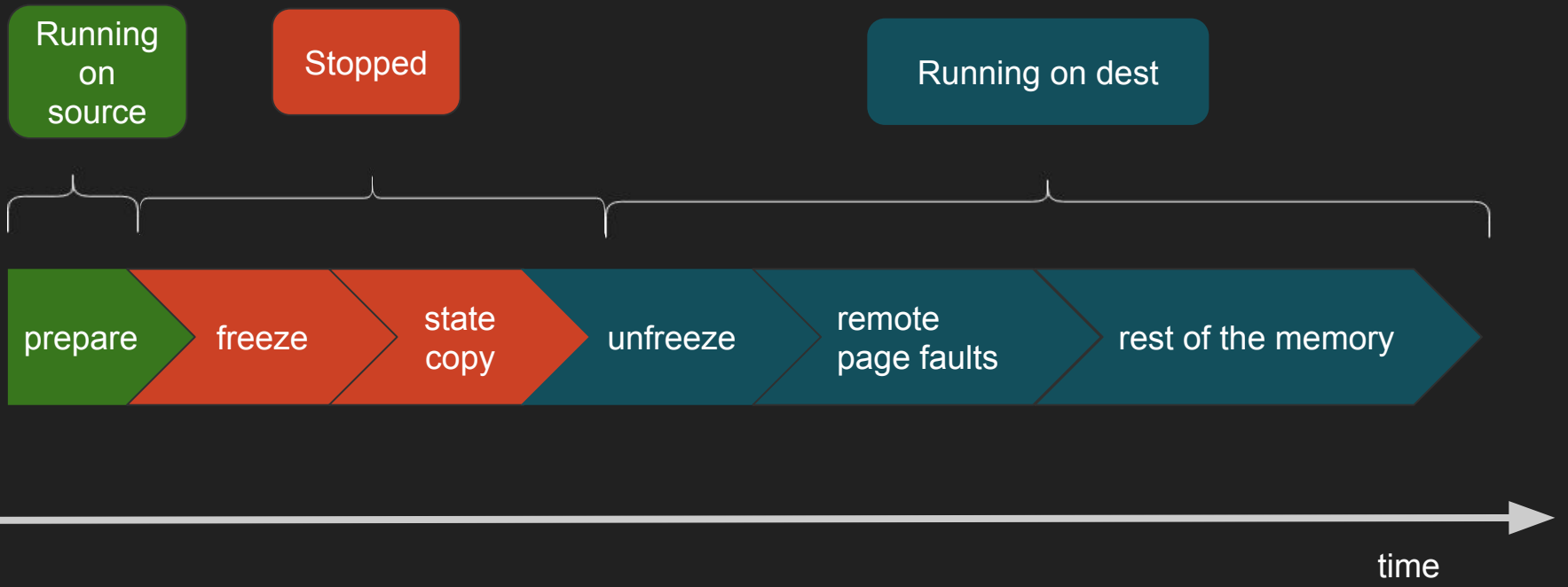
Post-copy

- Freeze on source
- Copy state **except** memory
- Enable “remote swap”
- Unfreeze on destination
- Bring memory on demand

Pre-copy


Running on source
Stopped
Running on dest
prepare
memory copy 1
memory copy n
freeze
state copy
unfreeze
time

Post-copy


Running on source
Stopped
Running on dest
prepare
freeze
state copy
unfreeze
remote page faults
rest of the memory
time

Pre-copy vs post-copy

Pre-Copy

+ Less vulnerable to node failures
+ High performance in “UP” state
- Longer downtime
- Might diverge

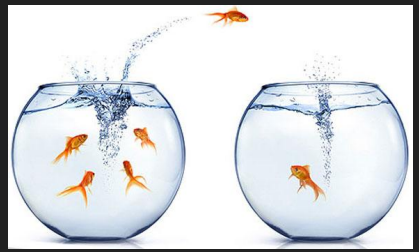
Post-Copy

- More vulnerable to node failures
- Slowdown after migration
+ Shorter downtime
+ Predictable downtime

https://youtu.be/lo2JJ2KWrlA

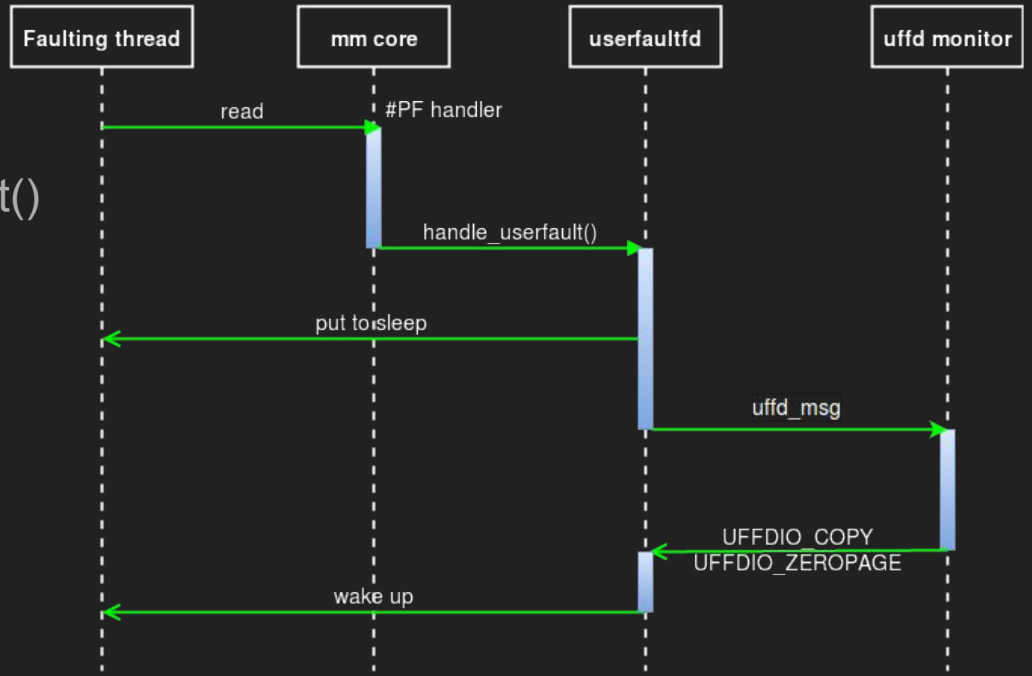
VM post-copy migration

- Guest memory is a part of QEMU address space
- Combine pre- and post-copy
- Straightforward flow
  - Start a thread for for user fault handling
  - Register guest memory areas with userfaultfd
  - Guest page fault causes `UFFD_EVENT_PAGEFAULT`
    - Request the page from source
    - copy/zero guest memory upon response
  - Fetch non-faulting pages in the background



Page fault processing

- Read not-present memory
- #PF handler calls handle_userfault()
- handle_userfault():
  - Suspend the faulting thread
  - Notify userfaultfd monitor
- monitor thread:
  - Resolve the pagefault
    - UFFDIO_{COPY,ZEROPAGE}
  - Wake up the faulting thread



CRIU + post-copy migration

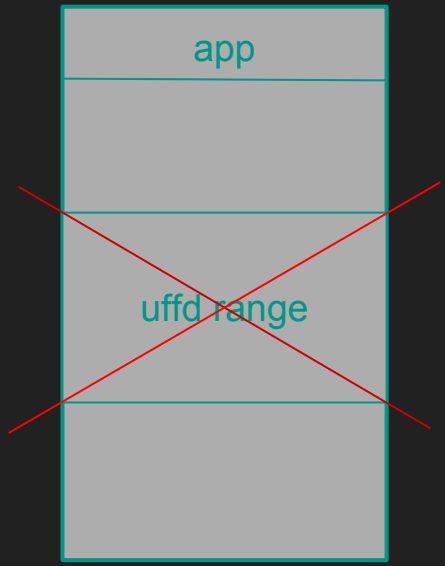
- Different address spaces
  - Restore controller
  - Restored processes
- Basic flow similar to VMs
  - Start a daemon for user fault handling
  - Register restored process areas with userfaultfd
    - Might be quite a few uffds
  - Handle page faults
  - Fetch non-faulting memory in the background
- BUT

Non-cooperative userfaultfd

- Processes change mappings on the flight
  - `madvise(DONTNEED, FREE, REMOVE)`
  - `munmap()`
  - `mremap()`
- Processes share memory
  - `fork()`
- Processes die
  - `exit()`

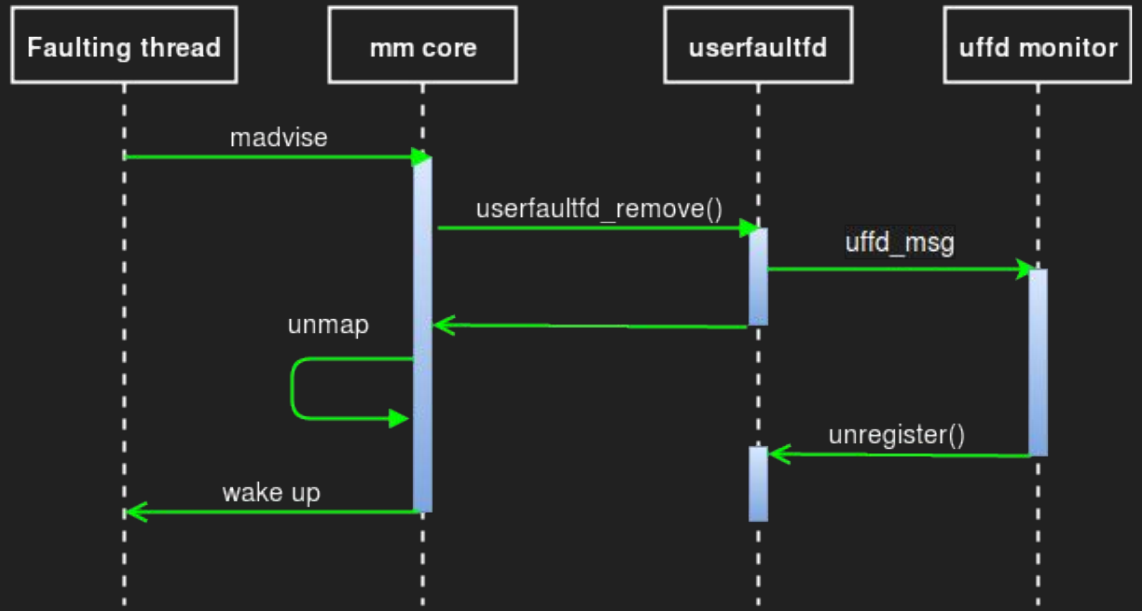
madvise(DONTNEED, FREE, REMOVE)

- Application requests to free pages
- Expects to read zeros
- Userfaultfd should not intervene



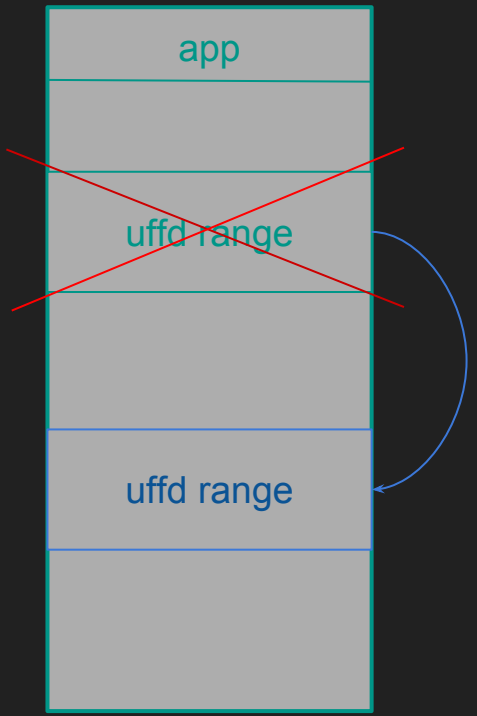
madvise(DONTNEED, FREE, REMOVE)

- madvise(REMOVE)
  - userfaultfd_remove()
  - Notify monitor
- Monitor:
  - userfaultfd_unregister()



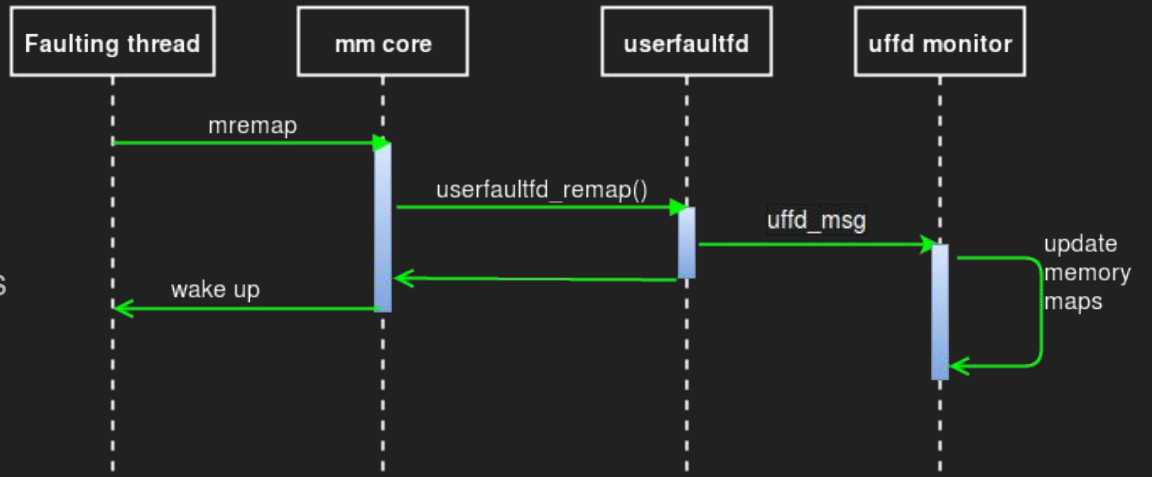
mremap()

- Application moves virtual range
- Expects data at new addresses
- Userfaultfd monitor should “remap” as well



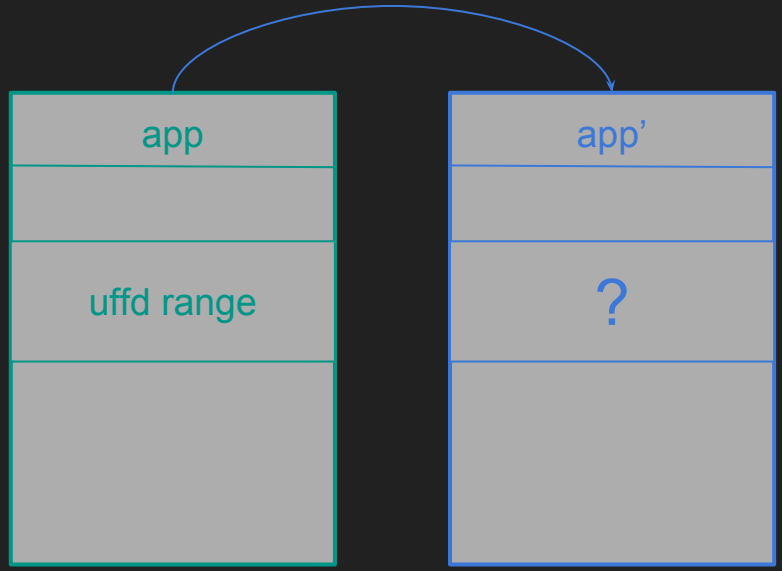
mremap()

- mremap()
  - userfaultfd_remap()
  - Notify monitor
- Monitor:
  - Update internal memory maps



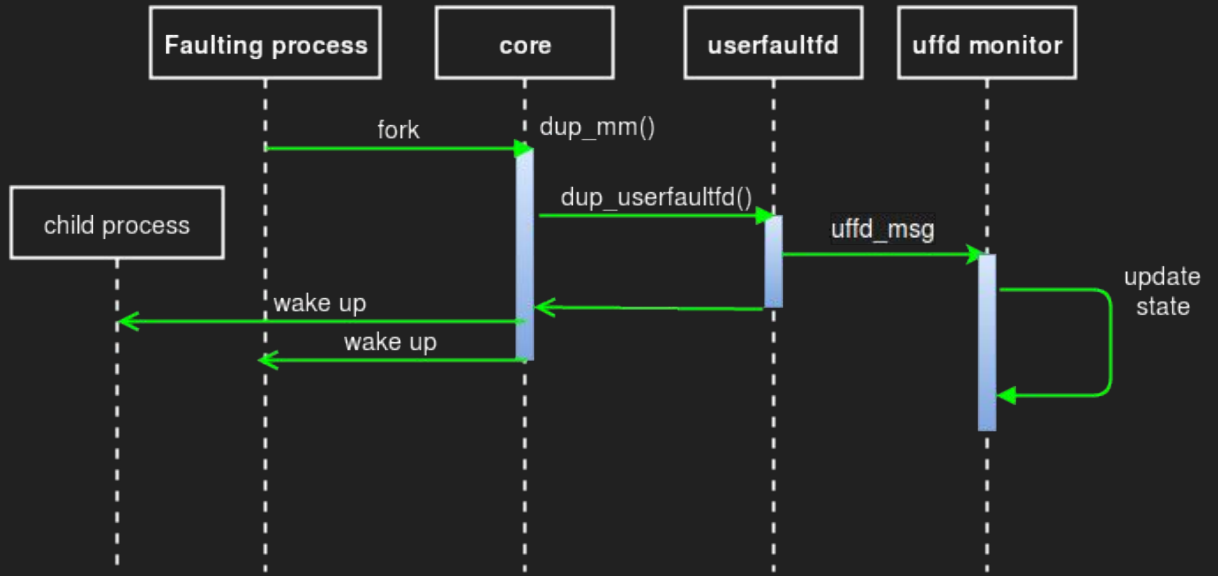
fork()

- Address space is duplicated
- Child expects to see data
- Userfaultfd monitor has to fill it



fork()

- fork()
  - dup_userfaultfd()
  - Notify monitor
- Monitor:
  - Update internal state



exit()

- Process exits during memory copy
- Userfaultfd monitor should stop filling the memory
  - Attempted `UFFDIO_EVENT_EXIT`, was dropped because of implementation complexity
  - Exit detection is based on `-ESRCH` returned by `UFFDIO_COPY/UFFDIO_ZEROPAGE`

Limitations

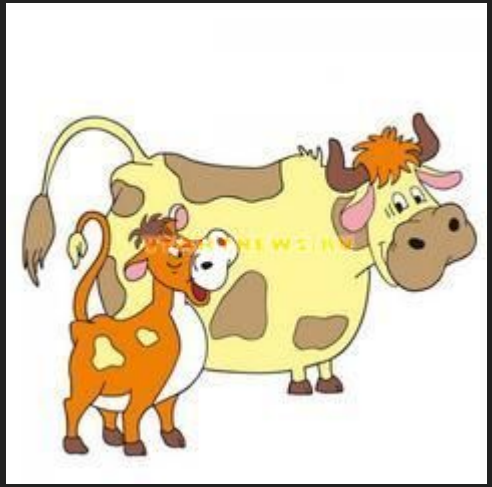
- No **UFFDIO_ZEROPAGE** for hugetlbfs
- Missing notification for ***fallocate*(PUNCH_HOLE | KEEP_SIZE)**
- Synchronous events are required for multi-threaded monitor apps
- Userfaultfd and COW
  - Same page may be shared by several processes. Now we create a copy for each.
  - COW-ed page may be “lazily” restored, need API for that
- Userfaultfd nesting/chaining

Current development

- Add synchronous events
  - http://www.spinics.net/lists/linux-mm/msg127093.html
- Add notification for ***fallocate*(PUNCH_HOLE | KEEP_SIZE)**
  - **Stuck at the moment**

Future: userfaultfd and COW

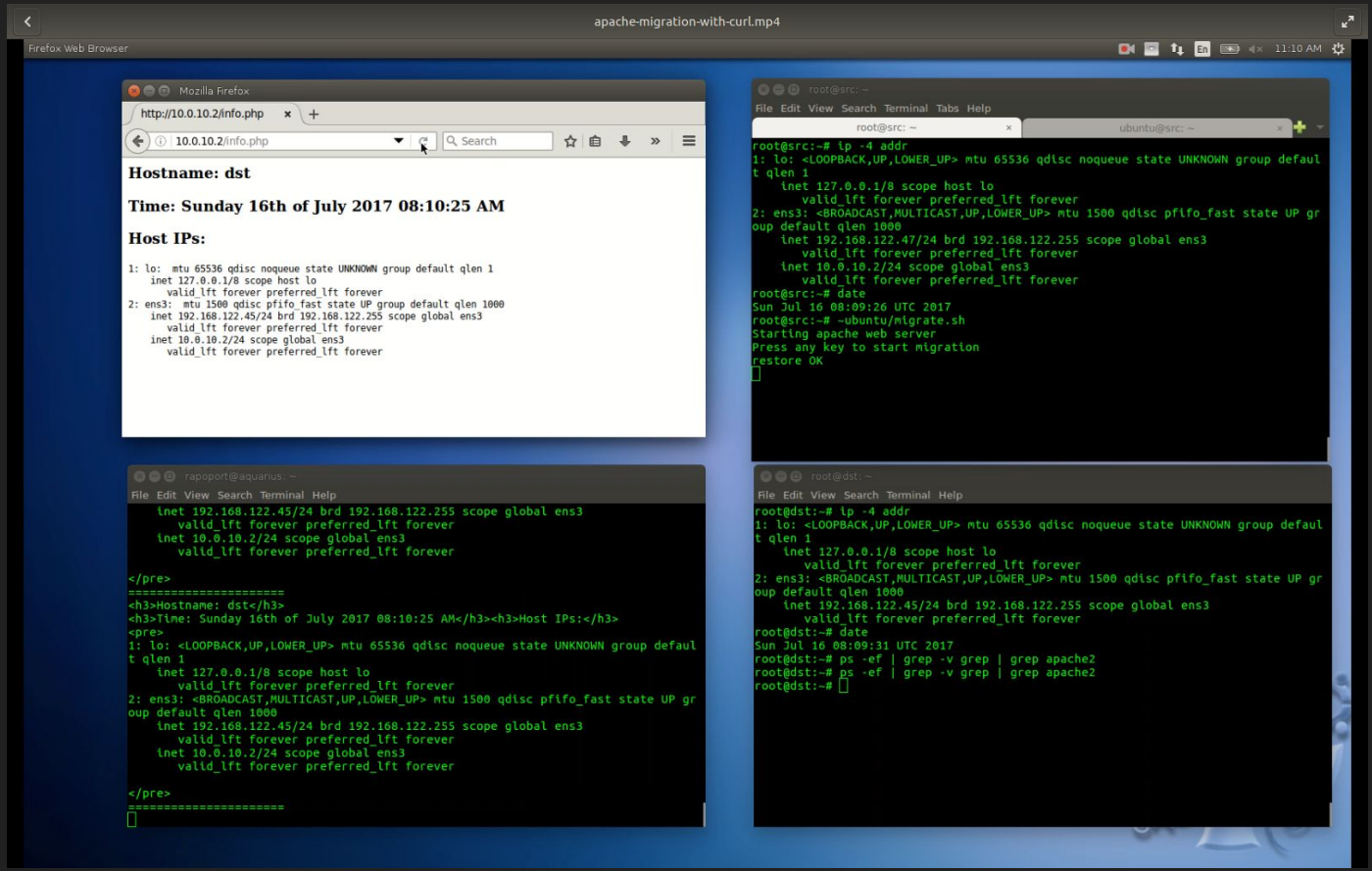
- Enable userfaultfd for file-backed VMAs
  - Special COW-only mode?
  - UFFDIO_EVENT_PAGEFAULT_COW?
- Add API for COW-ing pages
  - UFFDIO_COPY + map to several address spaces
  - Process page fault in kernel and copy from page cache



Future: userfaultfd nesting/chaining

- Notifications about userfaultfd_{register,unregister}
- Ability to “pass” page fault to the next userfaultfd context
- Userfaultfd context prioritization

Demo


apache-migration-with-curl.mp4
Firefox Web Browser
11:10 AM
Mozilla Firefox
http://10.0.10.2/info.php
10.0.10.2/info.php
Search
Hostname: dst
Time: Sunday 16th of July 2017 08:10:25 AM
Host IPs:
1: lo: mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
2: ens3: mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
inet 192.168.122.45/24 brd 192.168.122.255 scope global ens3
valid_lft forever preferred_lft forever
inet 10.0.10.2/24 scope global ens3
valid_lft forever preferred_lft forever
root@src: ~
File Edit View Search Terminal Tabs Help
root@src: ~
ubuntu@src: ~
root@src:~# ip -4 addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
inet 192.168.122.47/24 brd 192.168.122.255 scope global ens3
valid_lft forever preferred_lft forever
inet 10.0.10.2/24 scope global ens3
valid_lft forever preferred_lft forever
root@src:~# date
Sun Jul 16 08:09:26 UTC 2017
root@src:~# ~ubuntu/migrate.sh
Starting apache web server
Press any key to start migration
restore OK
rapoport@aquarius: ~
File Edit View Search Terminal Help
inet 192.168.122.45/24 brd 192.168.122.255 scope global ens3
valid_lft forever preferred_lft forever
inet 10.0.10.2/24 scope global ens3
valid_lft forever preferred_lft forever
</pre>
==========
<h3>Hostname: dst</h3>
<h3>Time: Sunday 16th of July 2017 08:10:25 AM</h3><h3>Host IPs:</h3>
<pre>
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
inet 192.168.122.45/24 brd 192.168.122.255 scope global ens3
valid_lft forever preferred_lft forever
inet 10.0.10.2/24 scope global ens3
valid_lft forever preferred_lft forever
</pre>
==========
root@dst: ~
File Edit View Search Terminal Help
root@dst:~# ip -4 addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
inet 192.168.122.45/24 brd 192.168.122.255 scope global ens3
valid_lft forever preferred_lft forever
root@dst:~# date
Sun Jul 16 08:09:31 UTC 2017
root@dst:~# ps -ef | grep -v grep | grep apache2
root@dst:~# ps -ef | grep -v grep | grep apache2
root@dst:~#

References

https://www.kernel.org/doc/Documentation/vm/userfaultfd.txt

http://man7.org/linux/man-pages/man2/userfaultfd.2.html

https://schd.ws/hosted_files/lcccna2016/c4/userfaultfd.pdf

http://wiki.qemu.org/Features/PostCopyLiveMigration

https://criu.org/Lazy_migration

https://medium.com/@MartinCracauer/generational-garbage-collection-write-barriers-write-protection-and-userfaultfd-2-8b0e796b8f7f

Thank you!

Initialization

- Create userfault file descriptor

```
uffd = syscall(__NR_userfaultfd, O_CLOEXEC | O_NONBLOCK);
```

- Perform API handshake
  - Query supported features
  - Enable/disable notifications for non page fault events

```
ioctl(uffd, UFFDIO_API, &uffdio_api);
```

- Register memory ranges

```
userfaultfd_register(uffd, UFFDIO_REGISTER, &uffdio_register);
```

Simple userfaultfd monitor

```
pollfd[0].fd = uffd;

while (poll(pollfd, 1, -1) > 0) {
    read(uffd, &uffd_msg, sizeof(uffd_msg));

    switch (uffd_msg.event) {
    case UFFD_EVENT_PAGEFAULT:
        page_fault_event(&msg);
        break;
    case UFFD_EVENT_FORK:
        fork_event(&msg);
        break;
    ...
    default:
        unknown_event();
        break;
}
```

User-space page fault handler

```
page_fault_event(struct uffdio_msg *msg)
{
    struct uffdio_copy uffdio_copy = { 0 };
    unsigned long address, len;
    void *page_data;

    address = msg->arg.pagefault.address;
    get_data(address, &page_data, &len);

    uffdio_copy.src = (unsigned long) page_data;
    uffdio_copy.dst = address;
    uffdio_copy.len = len;

    ioctl(uffd, UFFDIO_COPY, &uffdio_copy);
}
```