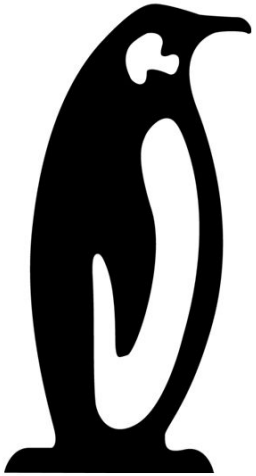


Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024

1G page support for guest_memfd

and how to support CoCo VM use cases effectively

Vishal Annapurve, Ackerley Tng

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

Goals of 1G page support in guest_memfd

- Provide physically contiguous 1G pages
  - Mapping 1G pages is out of scope

- Avoid double-allocation with CoCo VMs when backing shared and private memory ranges using 1G physically contiguous memory.

Option for source of 1G pages: HugeTLB

- Implemented in RFC [1]
- Refactor HugeTLB to extract allocator component
- Pro: Graceful transition from HugeTLBfs to guest_memfd
  - Near term: Allows co-tenancy of HugeTLBfs and guest_memfd backed VMs
    - No need to give up memory savings from HugeTLBfs Vmemmap Optimization (HVO)
- Pro: Can provide iterative steps toward a new future allocator
- Con: Dependency on HugeTLB
  - Unexplored: Managing userspace-visible changes e.g. HugeTLB's free_hugepages

[1] RFC: https://lore.kernel.org/all/cover.1726009989.git.ackerleytng@google.com/T/

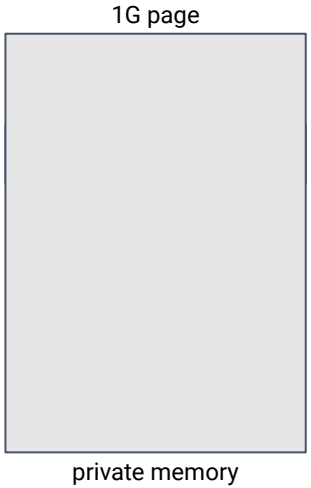
Option: Contiguous Memory Allocator (CMA)

- Port some HugeTLB features to be applied on CMA
  - e.g. 1G page pools, HugeTLB Vmemmap Optimization (HVO)
- Pro: Clean slate
- Con: Rebuilding/duplicating HugeTLB features

Current guest_memfd model causes double allocation

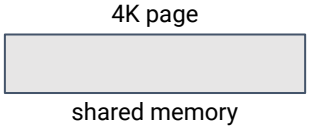
- guest_memfd only supports backing private memory
  - Need separate memory store to back shared memory
- Guests can convert memory at 4K granularity
  - During conversion, userspace VMM needs to unback private guest memory backing.
  - If private memory is backed by 1G pages, its subranges can't be unbacked.

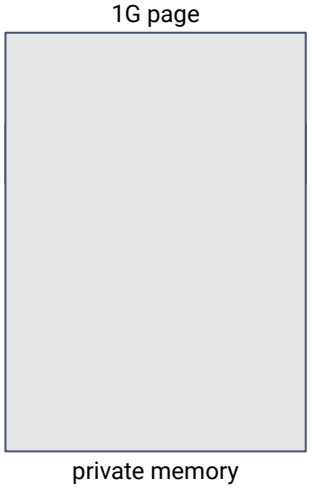
Double-allocation problem, illustrated



Double-allocation problem, illustrated

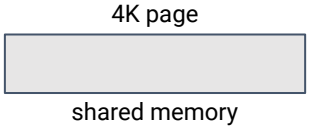
Guest requests to convert 4K page for shared memory

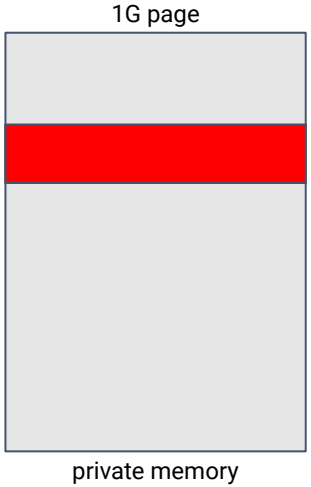




Double-allocation problem, illustrated

Guest requests to convert 4K page for shared memory





Punching out a sub-hugepage doesn't free memory

Solution for double-allocation (RFC [1])

- Allow mmap() and fault in only shared ranges (built on Fuad's series [2])
- Invariant: private ranges should not be faultable by userspace
  - Split 1G pages so only shared memory ranges can be faulted in. [3]
  - Reconstruct 1G pages back when entire hugepage range is private

[1] RFC: https://lore.kernel.org/all/cover.1726009989.git.ackerleytng@google.com/T/

[2] Fuad's series: https://lore.kernel.org/all/20240801090117.3841080-1-tabba@google.com/T/

[3] Discussed in Linux MM Alignment Session: https://lore.kernel.org/all/20240712232937.2861788-1-ackerleytng@google.com/

Issue: Hugepage reconstruction blocked on active users

- guest_memfd needs to return hugepages to the allocator on cleanup
- Subpage ranges may still be in use when inode cleanup happens
  - Hence cannot wait for safe_refcount like in Elliot's patch series [1]
- Option:
  - Guest_memfd marks shared page as a special type of page.
    - guest_memfd drops all the refcounts on truncation (or conversion to private).
    - Core-mm invokes a callback on such special pages when folio_put hits a refcount of 0.
    - guest_memfd reconstructs huge pages through this callback.
- Suggestions?

[1] Elliot's series: https://lore.kernel.org/all/20240829-guest-memfd-lib-v2-0-b9afc1ff3656@quicinc.com/T/

Issue: Elevated refcounts on private hugepage ranges

- Example scenario: Split of private hugepage on conversion of subpage range to shared
- Extra refcounts can be grabbed by KVM/arch subsystem.
- Option: Agree to following policy?
  - Guest memfd owns all long-term refcounts on private memory
  - Any short-term refcounts distributed outside guest_memfd should be protected by folio locks.

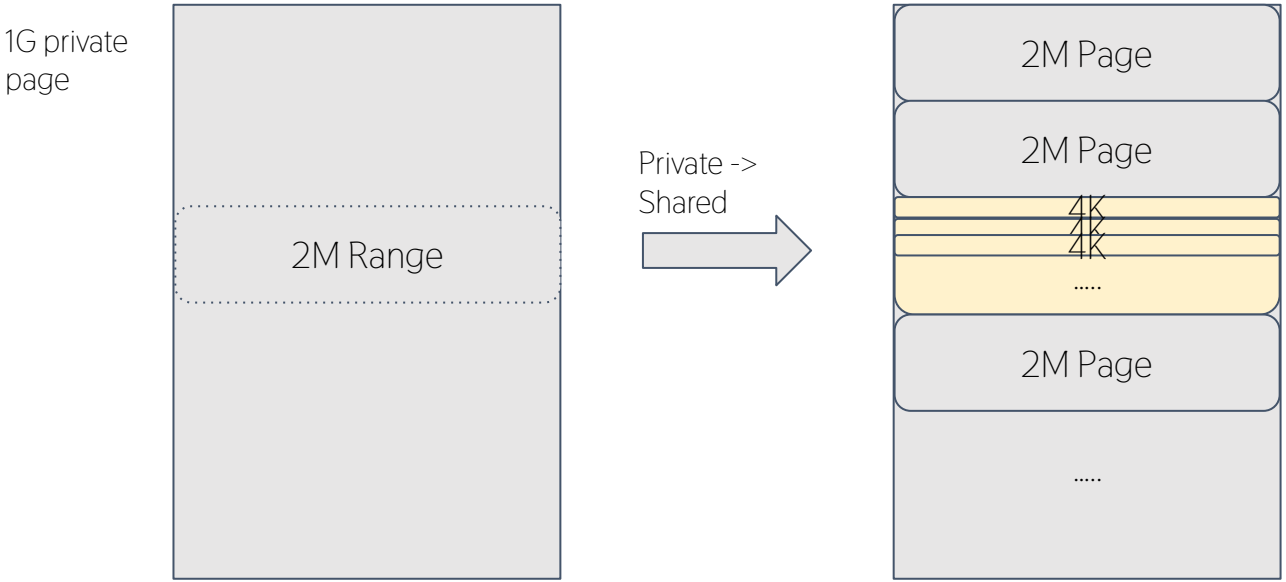
[1] Elliot's series: https://lore.kernel.org/all/20240829-guest-memfd-lib-v2-0-b9afc1ff3656@quicinc.com/T/

Issue: Elevated refcounts on shared hugepage ranges

- Example scenario: Split of shared hugepage on conversion of subpage range to private
- Extra refcounts can be grabbed by userspace/kernel.
- Options on the next slides

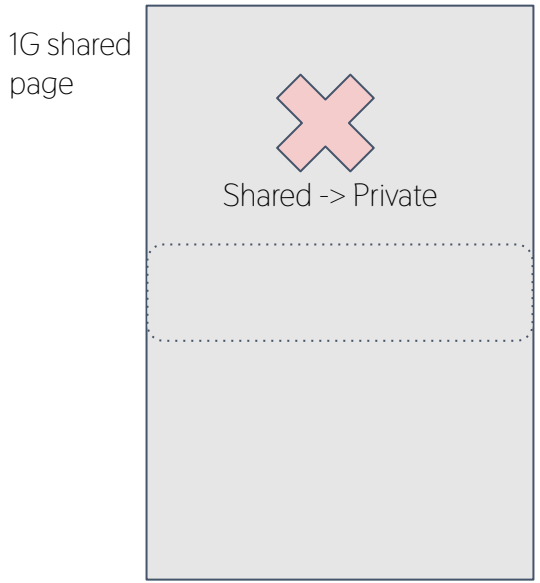
Option: Always split shared memory to 4K granularity (RFC)

- Hugepage is always split to 4k granularity if any subrange is shared



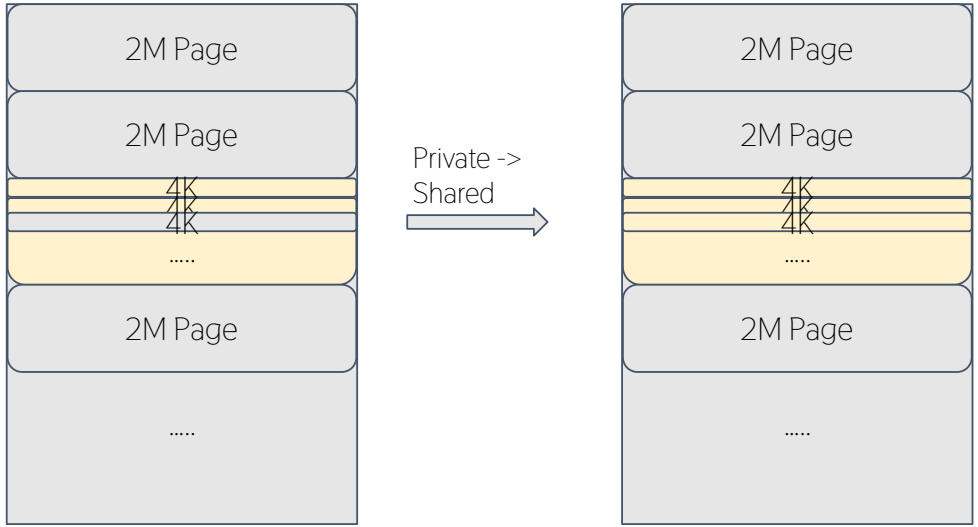
Option: Disallow splitting of shared hugepages

- No splitting means no need to handle refcounts
- Implication: Guests need to convert exact ranges back to private which were marked as shared before.



Option: Disallow splitting/reconstruction of shared hugepages

- No splitting means no need to handle refcounts
- Implication: Guests need to convert exact ranges back to private which were marked as shared before.
- If the last private subpage is converted to shared, don't merge, so that splitting is never required



Thanks!

Backup slides follow

Why 1G pages in guest_memfd?

- VM performance
  - Increase TLB hit rate and reduce page walks on TLB miss
  - Improved IO performance
- Memory savings of ~1.6% from HugeTLB Vmemmap Optimization (HVO)
- Bring guest_memfd to parity with existing VMs that use HugeTLB pages for backing memory

Conversion flow

1. Userspace uses KVM_SET_MEMORY_ATTRIBUTES ioctl to request conversion of range
2. If page is shared, guest_memfd unmaps entire hugepage range
    a. If page is private, only requested shared range is unmapped from guest-to-host page tables
3. At fault time, page is split and mapped back as with new shared/private status

What does HugeTLB refactoring involve?

- Broadly involves separating the HugeTLB allocator away from the rest of HugeTLB
  - More modularity
  - No functionality change intended
  - Likely step towards HugeTLB's integration into core-mm
- guest_memfd will use just the allocator component of HugeTLB, not including the complex parts of HugeTLB like
  - Userspace reservations (resv_map)
  - Shared PMD mappings
  - Special page walkers

HugeTLB features and refactoring

HugeTLB allocator	HugeTLB	Other supporting components
Actual huge page allocator	Userspace reservations (resv_map)	Vmemmap Optimization (HVO)
Hstate reservations/accounting	mmap(MAP_HUGETLB)	HugeTLB cgroup accounting
Subpools	Special hugetlb_fault()	mem cgroup accounting
Parsing kernel cmdline	Shared PMD mappings	
Reporting (/sys/kernel/mm/hugepages)	Userspace page walker	
Post-boot adjustment of # hugepages	**HugeTLBfs**	
Surplus HugeTLB page allocator	memfd_create(MFD_HUGETLB)	

HugeTLB features and refactoring

HugeTLB allocator	HugeTLB	Other supporting components
Actual huge page allocator	Userspace reservations (resv_map)	Vmemmap Optimization (HVO)
Hstate reservations/accounting	mmap(MAP_HUGETLB)	HugeTLB cgroup accounting
Subpools	Special hugetlb_fault()	mem cgroup accounting
Parsing kernel cmdline	Shared PMD mappings	
Reporting (/sys/kernel/mm/hugepages)	Userspace page walker	
Post-boot adjustment of # hugepages	**HugeTLBfs**	
Surplus HugeTLB page allocator	memfd_create(MFD_HUGETLB)	

Used by guest_memfd

What features need to be ported from HugeTLB?

- Improved allocation guarantees
  - Per NUMA node pool of huge pages
  - Subpools per guest_memfd
- Memory savings
  - HugeTLB Vmemmap Optimization
- Configuration/reporting features
  - Configuration of number of pages available (and per NUMA node) at and after host boot
  - Reporting of memory usage/availability statistics at runtime

What will the refactored interface look like?

- Allocator provides these functions
  - reserve(node, page_size, num_pages) => opaque handle
  - allocate(handle, mempolicy, page_size) => folio
  - split(handle, folio, target_page_size) => void
  - reconstruct(handle, first_folio, nr_pages) => void
  - free(handle, folio) => void
  - error(handle, folio) => void
  - unreserve(handle) => void
- Interface will allow allocator to be replaced

guest_memfd allocator interface

- Let userspace choose allocator
  - HugeTLB, or
  - Other allocator for other types of memory, like Nvidia's Extended GPU Memory (EGM)
- _ops structure with these hooks
  - reserve(node, page_size, num_pages) => opaque handle
  - allocate(handle, mempolicy, page_size) => folio
  - split(handle, folio, target_page_size) => void
  - reconstruct(handle, first_folio, nr_pages) => void
  - free(handle, folio) => void
  - error(handle, folio) => void
  - unreserve(handle) => void

RFC: Preventing Races (kvm_gmem_)

semaphore	should_set_attributes	get_pfn	fault	allocate	punch_hole	error_folio
filemap_invalidate_lock	write lock		shared (read) lock	shared (read) lock	write lock	
hugetlb_fault_mutex_lock	taken	taken	taken	taken	taken	
KVM_MMU_LOCK	taken	taken				