CHINA LINUX KERNEL
中国Linux内核开发者大会

vivo

# mmap_lock和anon_vma lock contention优化


OPPO 底层软件工程师
李杨欧文

CHINA LINUX KERNEL
中国Linux内核开发者大会

vivo

# 目录

CONTENTS



01

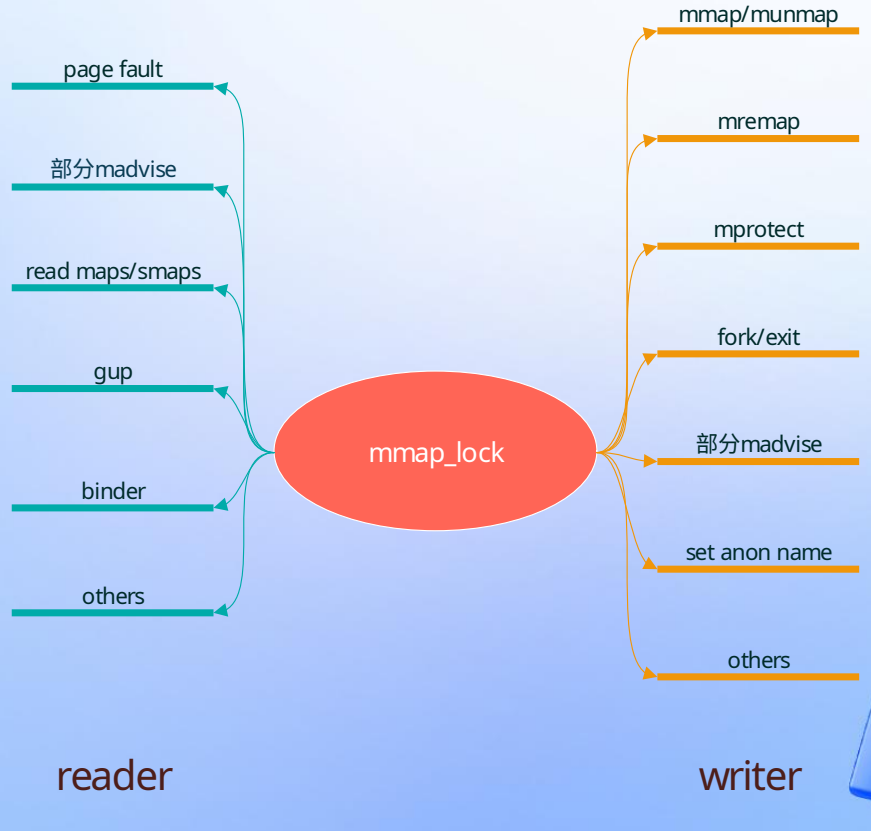
# 内存锁介绍

# 内存锁介绍



## mmap_lock

- mmap_lock（在较老的内核中称为 mmap_sem），存在于每个进程的内存描述符（struct mm_struct）中
- 核心作用：保护进程的虚拟内存区域（VMA）元数据的一致性
- 锁粒度大，持锁路径很多且热点
- 读写锁，只要存在写者，很容易产生严重的竞争，出现优先级翻转问题




# 内存锁介绍

社区mmap_lock竞争问题的优化思路

- 投机性缺页（speculative page-fault，SPF）

  采用“乐观锁”策略，处理缺页异常时先不拿mmap_lock，缺页最后验证vma在此期间是否被修改。如果不幸被修改，则取消前面的工作，回退到获取mmap_lock的原路径。

- per-vma lock

  将粗粒度的 mmap_lock 拆分为更细粒度的锁，为每个 VMA 对象分配一个独立的锁。这样，当多个线程操作不同的 VMA 时，它们可以并行执行，而不会相互阻塞。

  a. 读操作（如缺页异常）：仅获取该 VMA 的读锁（vm_lock），而不是整个 mmap_lock。

  b. 写操作：仍然需要获取整个 mmap_lock 写锁，修改某个特定 VMA 时，需要获取该 VMA 的写锁

# 内存锁介绍

anon_vma root lock

- 位于struct anon_vma中 anon_vma->rwsem
- 经过fork/split，anon_vma形成树形结构，总是获取root锁 anon_vma->root->rwsem
- 匿名页反向映射的读写锁，保护anon_vma之间的关联关系
  读锁场景：查找反向映射，如内存回收、内存迁移
  写锁场景：建立/销毁反向映射关系，如fork、exit、vma split
- anon_vma root lock的竞争会跨进程，可能同时需要获取mmap_lock，让竞争和依赖关系变得更加复杂，造成优先级翻转

02

# madvise中的mmap_lock竞争优化

# madvise中的mmap_lock竞争优化

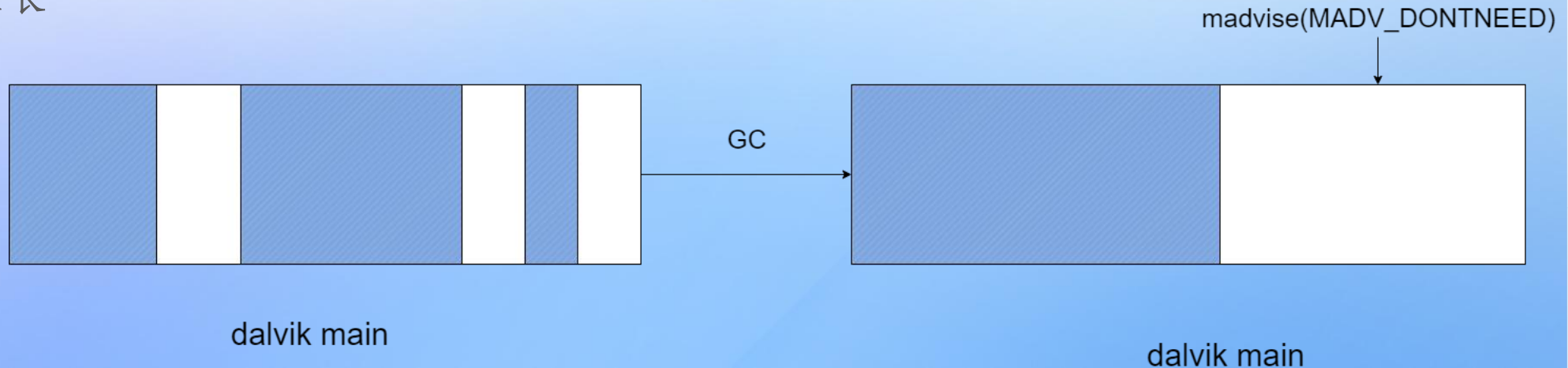
背景知识

java堆和native libc库会通过madvise来动态管理内存

以java 垃圾回收(Garbage Collection, GC)为例

安卓app的java堆位于名为dalvik-main的巨大vma中，会很频繁的进行GC行为，调用MADV_DONTNEED释放内存。

- 全程持mmap_lock读锁
- vma很大，临界区长
- 优先级低

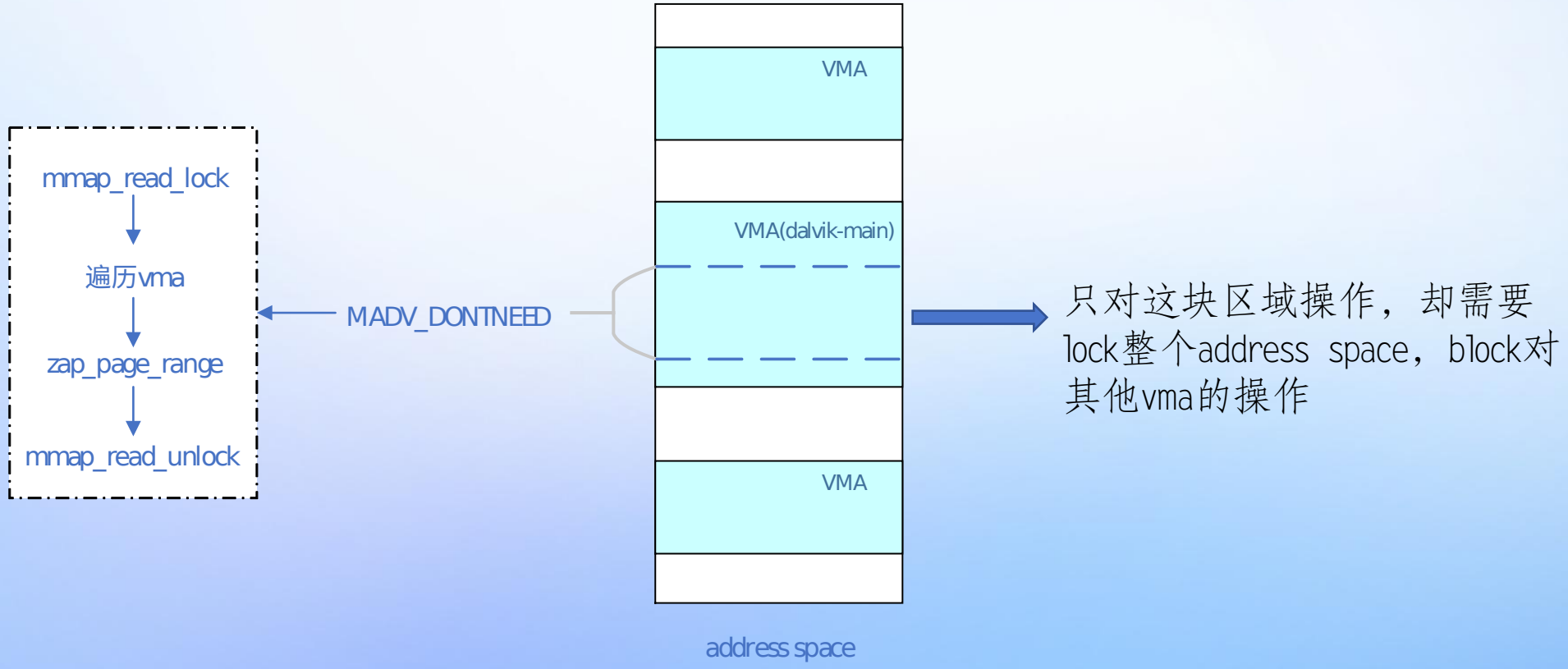


# madvise中的mmap_lock竞争优化

- GC线程的madvise很容易与UI线程发生竞争
- GC优先级往往偏低，出现优先级翻转
- 结果：UI线程卡住，用户感到明显卡顿

# madvise中的mmap_lock竞争优化

优化方案：大锁拆小锁

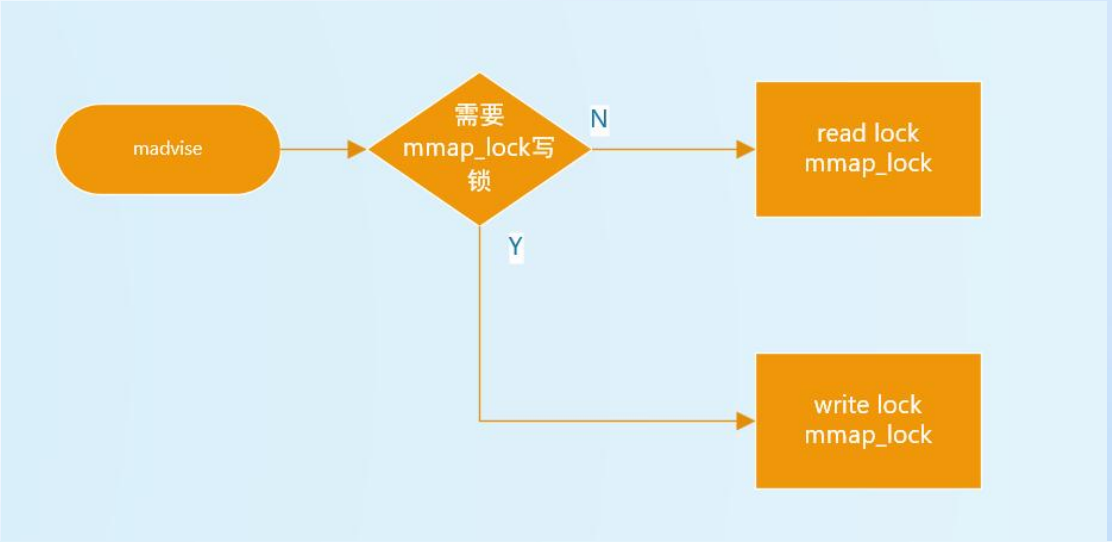


# madvise中的mmap_lock竞争优化

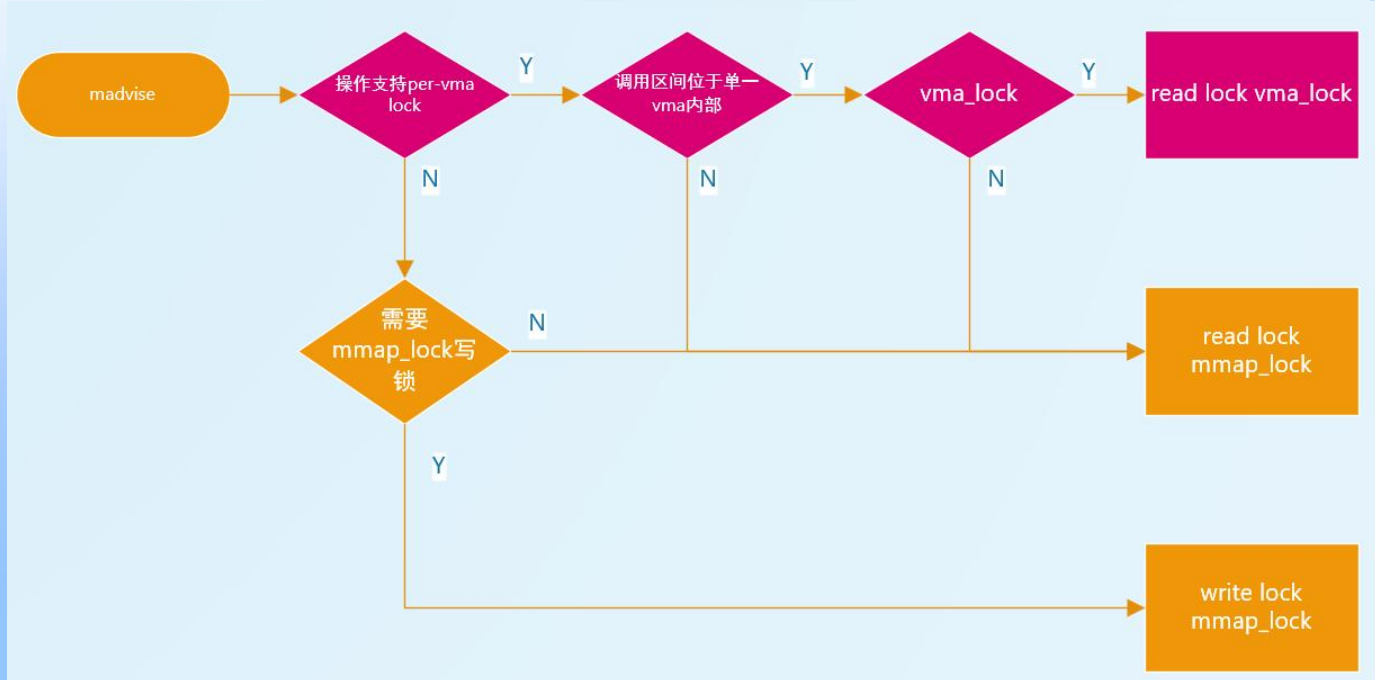


优化方案：大锁拆小锁

用per-vma lock替代mmap_lock




1. 绝大多数madvise操作不会修改vma字段，只需要vma保持稳定，可用per-vma lock替代。当前已支持MADV_DONTNEED和MADV_FREE
2. 只适用madvise区间位于单一vma内的情况
3. lock vma可能失败，需要fallback到read-lock mmap_lock



# madvise中的mmap_lock竞争优化



优化方案：大锁拆小锁

用per-vma lock替代mmap_lock

社区链接：

mm: use per_vma lock for MADV_DONTNEED

mm: madvise: use per_vma lock for MADV_FREE

```
1694  /*
1695   * Any behaviour which results in changes to the vma->vm_flags needs to
1696   * take mmap_lock for writing. Others, which simply traverse vmas, need
1697   * to only take it for reading.
1698   */
1699  static enum madvise_lock_mode get_lock_mode(struct madvise_behavior *madv_behavior)
1700  {
1701          if (is_memory_failure(madv_behavior))
1702                  return MADVISE_NO_LOCK;
1703
1704          switch (madv_behavior->behavior) {
1705          case MADV_REMOVE:
1706          case MADV_WILLNEED:
1707          case MADV_COLD:
1708          case MADV_PAGEOUT:
1709          case MADV_POPULATE_READ:
1710          case MADV_POPULATE_WRITE:
1711          case MADV_COLLAPSE:
1712          case MADV_GUARD_INSTALL:
1713          case MADV_GUARD_REMOVE:
1714                  return MADVISE_MMAP_READ_LOCK;
1715          case MADV_DONTNEED:
1716          case MADV_DONTNEED_LOCKED:
1717          case MADV_FREE:
1718                  return MADVISE_VMA_READ_LOCK;
1719          default:
1720                  return MADVISE_MMAP_WRITE_LOCK;
1721          }
1722  }
1723
```

MADV_DONTNEED以及MADV_FREE已支持 per-vma lock，可以覆盖android设备大多数问题。

收益：Android模拟用户场景测试中，99.5%的madvise成功使用到per-vma lock，主要来自native libc的动态内存管理以及java GC。

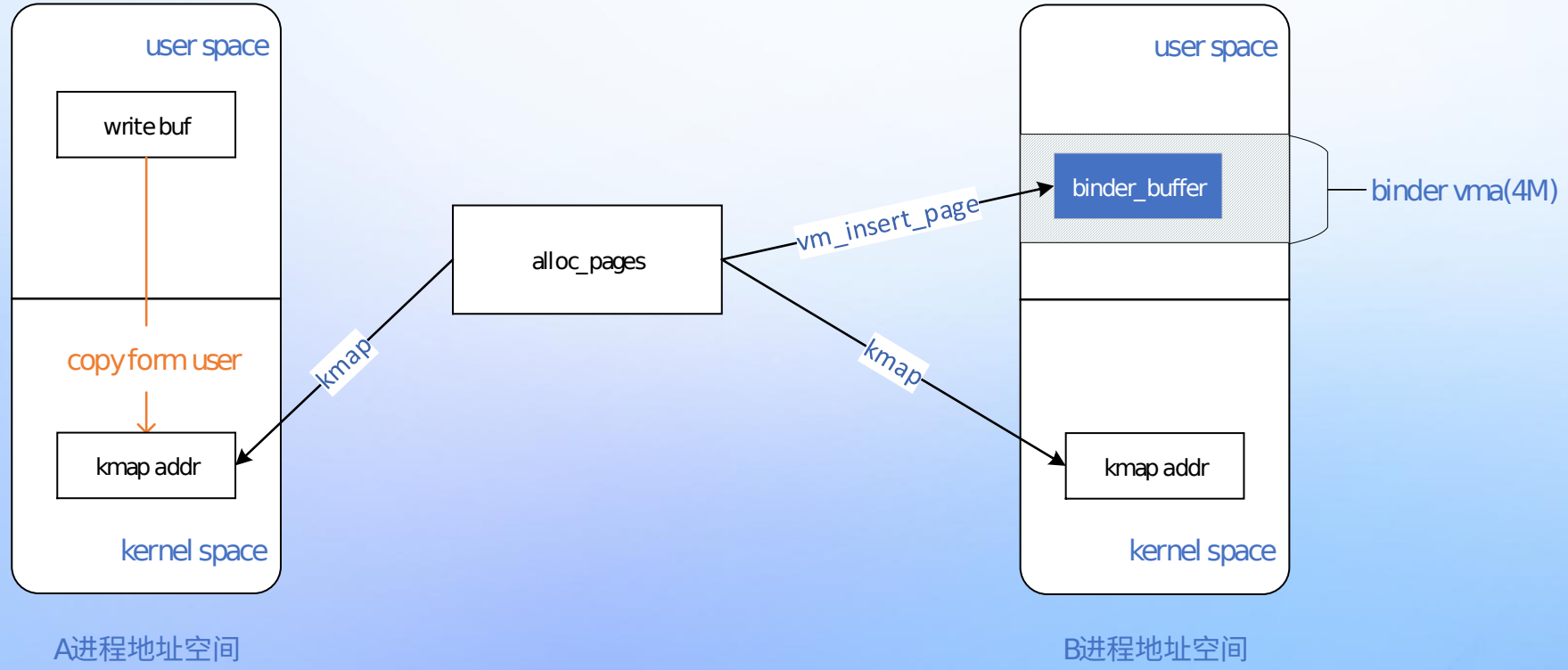
03

# binder驱动中的mmap_lock竞争优化

# binder驱动中的mmap_lock竞争优化

binder是android中重要的IPC通信机制

高性能：相较与传统的IPC，只需要一次内存拷贝




A进程发binder到B进程

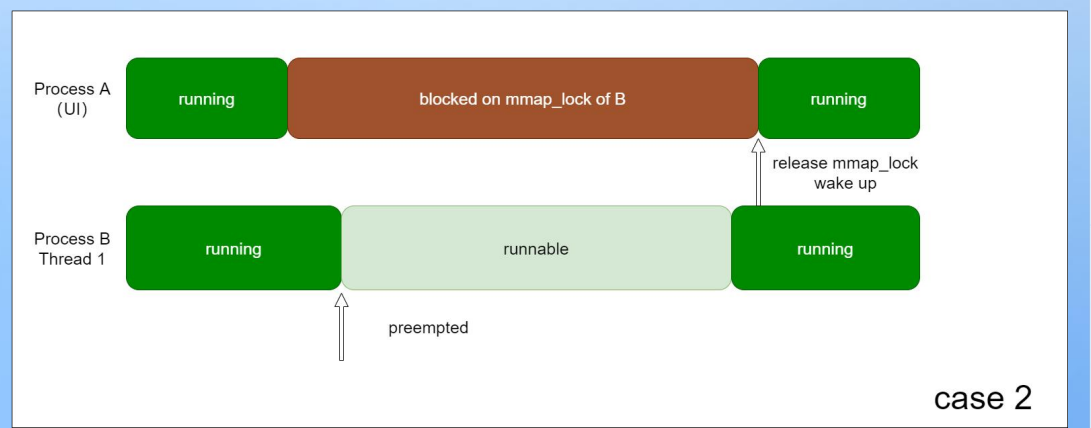
1. B进程mmap分配一块binder_vma，此时未分配物理内存，只有虚拟地址
2. A进程通过ioctl进入binder驱动
3. 在B进程的binder_vma中寻找一块合适的binder_buffer
4. 查看binder_buffer对应的物理内存是否分配。如果没有，则分配page，并映射到B进程binder_vma（此过程需要获取B的mmap_lock）
5. kmap将物理地址映射的内核空间地址，copy_from_user将A进程用户空间buffer拷贝到内核空间
6. 唤醒B进程，处理数据

# binder驱动中的mmap_lock竞争优化



binder中的mmap_lock竞争问题

- Android中大量使用binder，在UI或者交互相关线程中，若binder call阻塞，用户感觉到明显卡顿，体验差
- 发送端获取binder接收端进程的mmap_lock，对端状态不可控
  1. 接收端线程间可能竞争激烈
  2. 接收端可能存在调度资源不足，出现优先级翻转

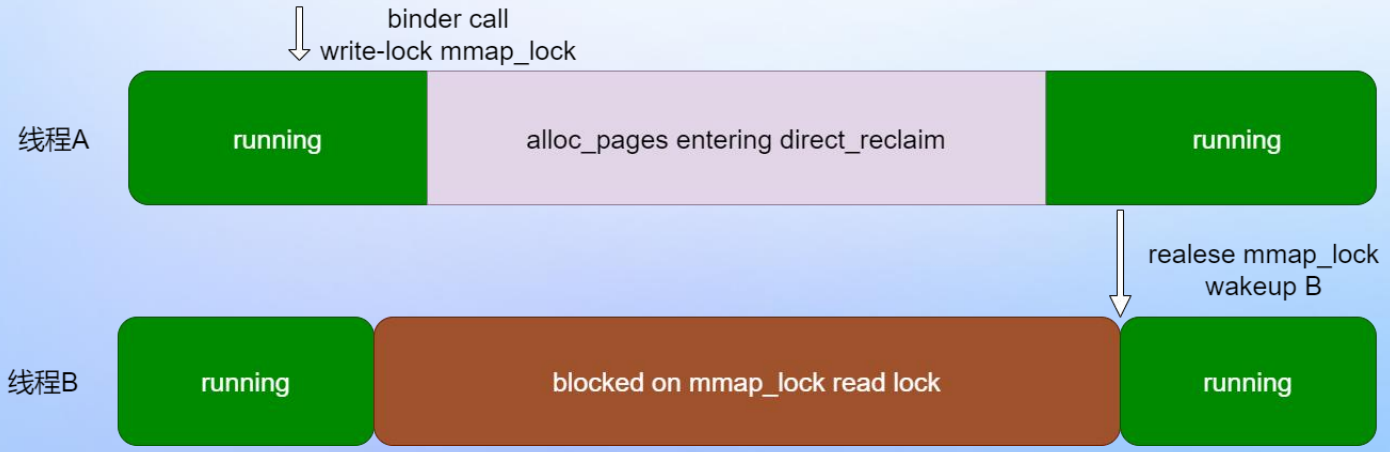



# binder驱动中的mmap_lock竞争优化



binder中的mmap_lock竞争问题

- binder中持mmap_lock锁，导致其他更关键的线程拿不到锁
  1. write-lock mmap_lock，使读写锁回退到互斥锁，影响并发
  2. binder临界区中会分配内存，内存紧张时耗时不可控



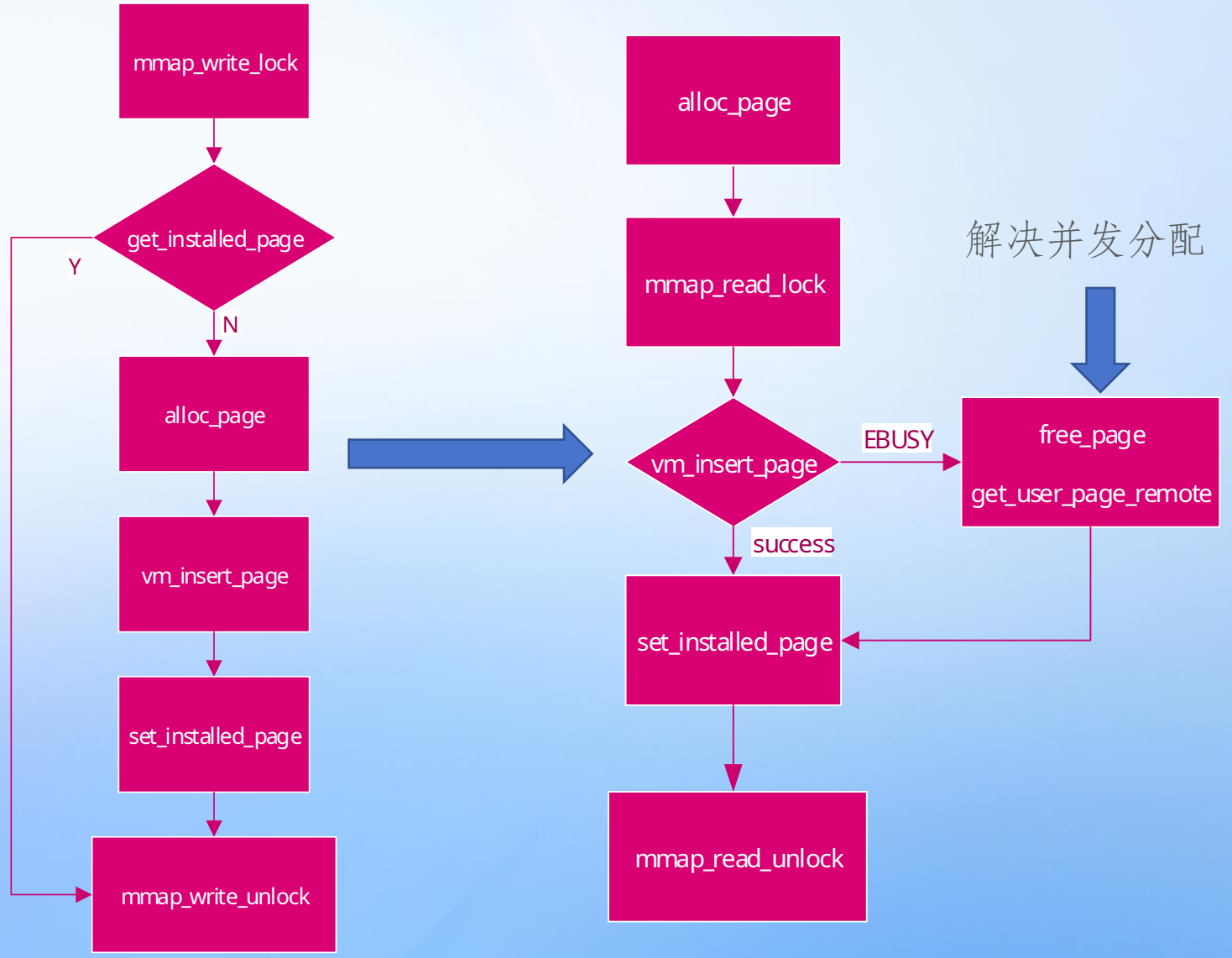
# binder驱动中的mmap_lock竞争优化

binder中的mmap_lock竞争问题

优化1:

内存分配移出临界区，写锁改读锁

有效缓解mmap_lock锁竞争，未完全解决，依然需要read-lock mmap_lock

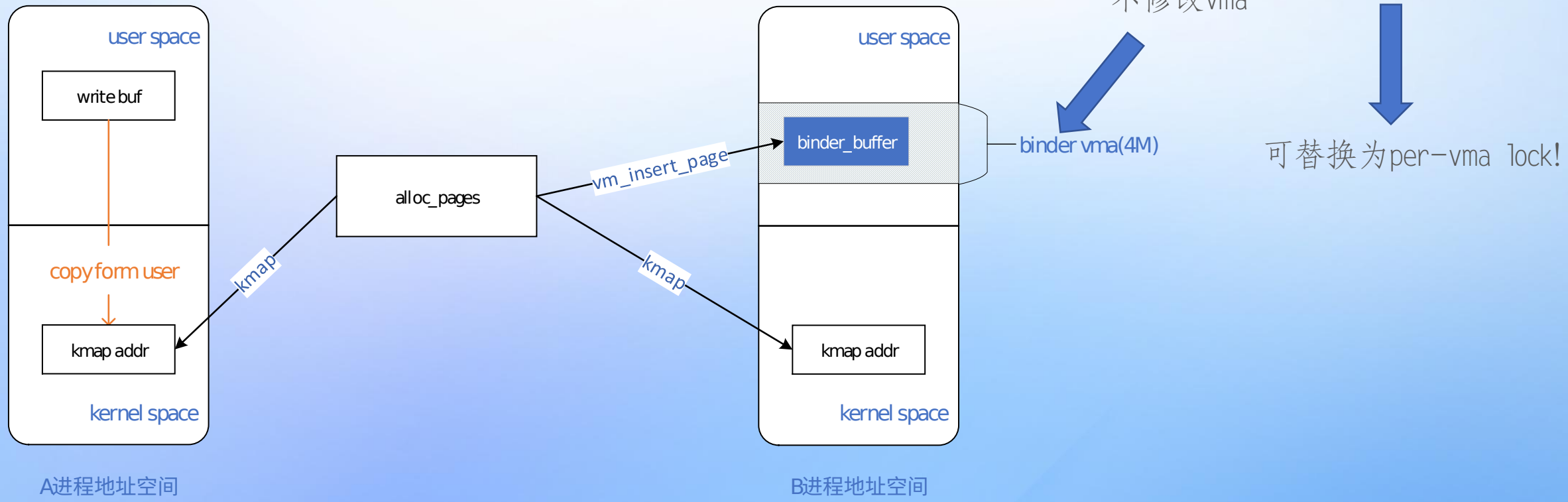


# binder驱动中的mmap_lock竞争优化

binder中的mmap_lock竞争问题

优化2:

per-vma lock替换mmap_lock

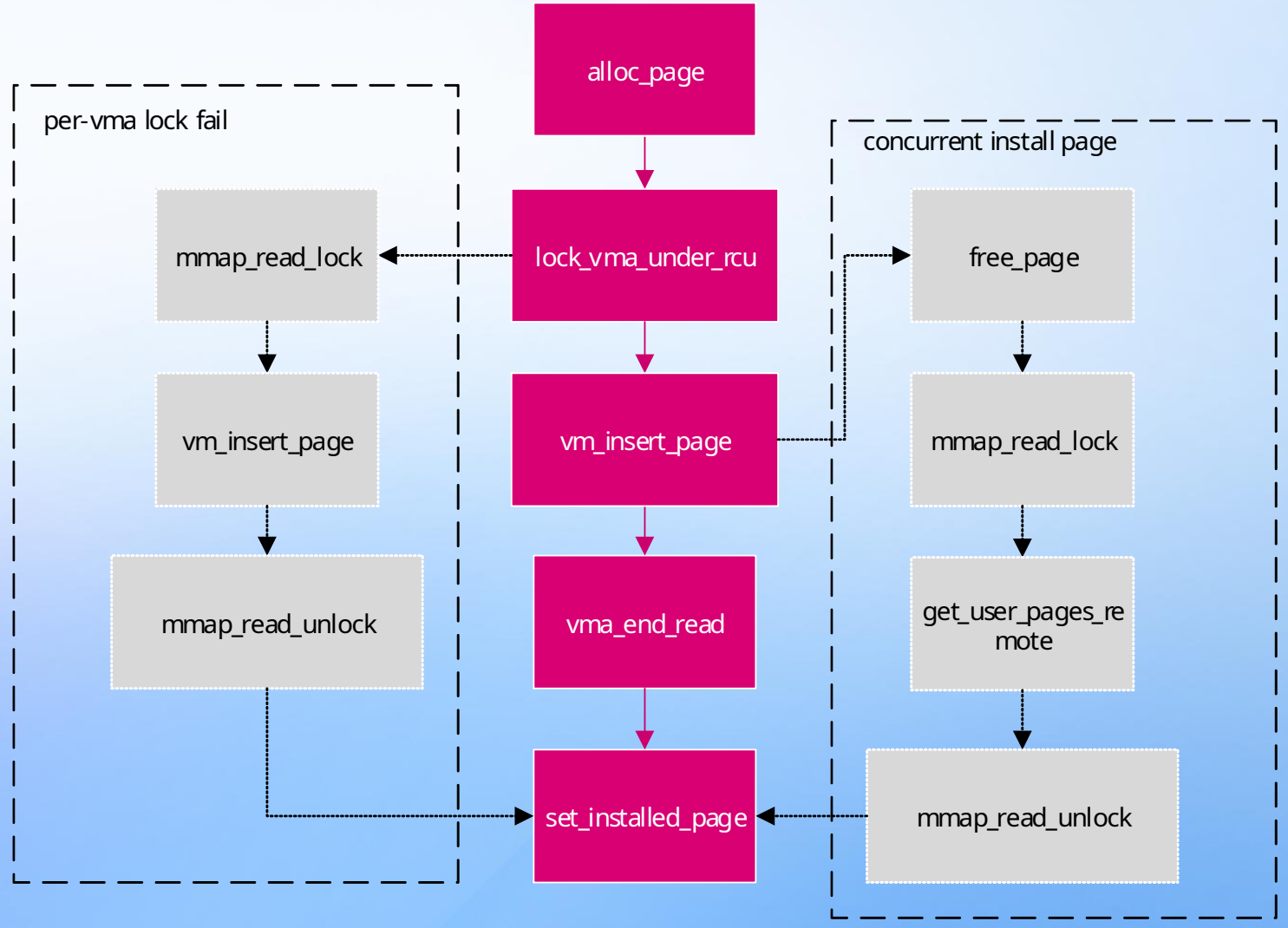


# binder驱动中的mmap_lock竞争优化

binder中的mmap_lock竞争问题

优化2:

per-vma lock替换mmap_lock



# binder驱动中的mmap_lock竞争优化

binder中的mmap_lock竞争问题

社区链接：

优化1:内存分配移出临界区，写锁改读锁

binder: concurrent page installation

优化2:per-vma lock替换mmap_lock

binder: use per-vma lock in page installation

04

# anon_vma root锁竞争问题探讨

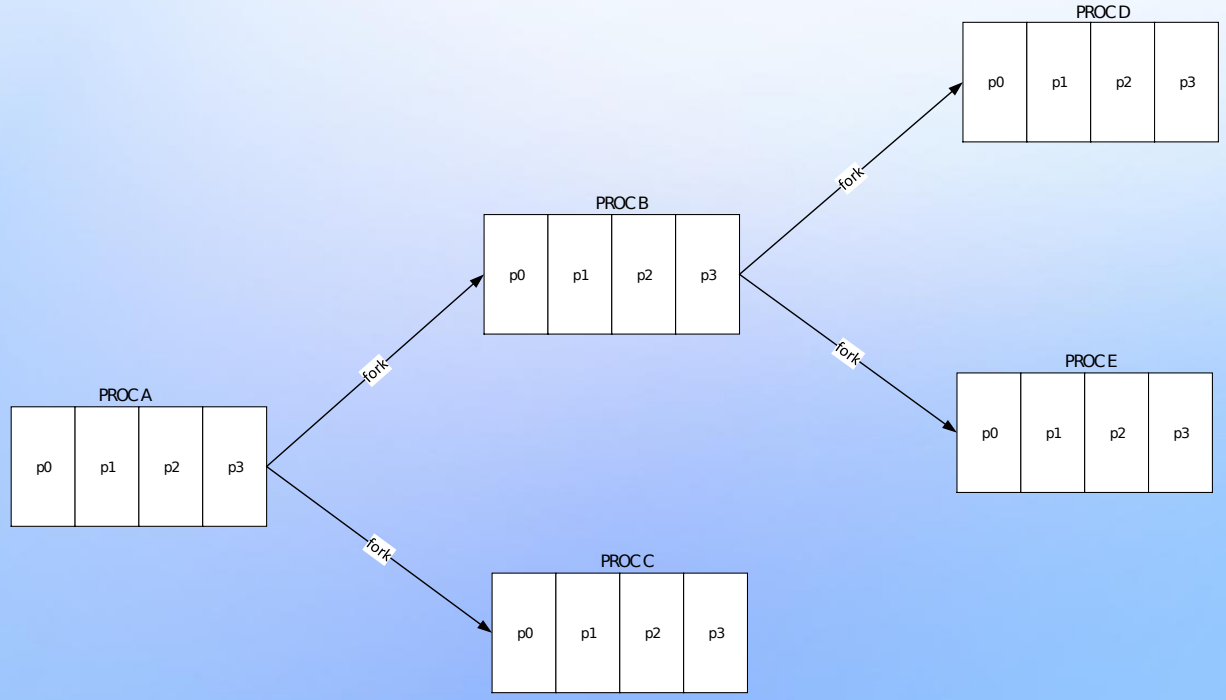
# anon_vma root锁竞争问题探讨

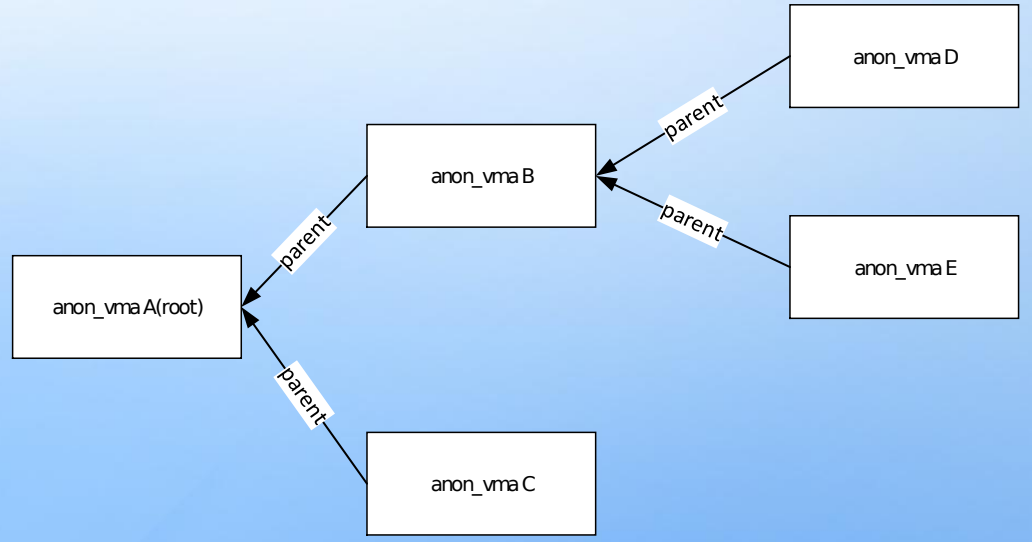
## 问题描述

对匿名页反向映射相关的修改与访问，均要获取anon_vma->root->rwsem锁

读锁：各个反向映射过程，如内存回收、内存迁移

写锁：建立/销毁反向映射关系，如fork、exit、vma split/merge



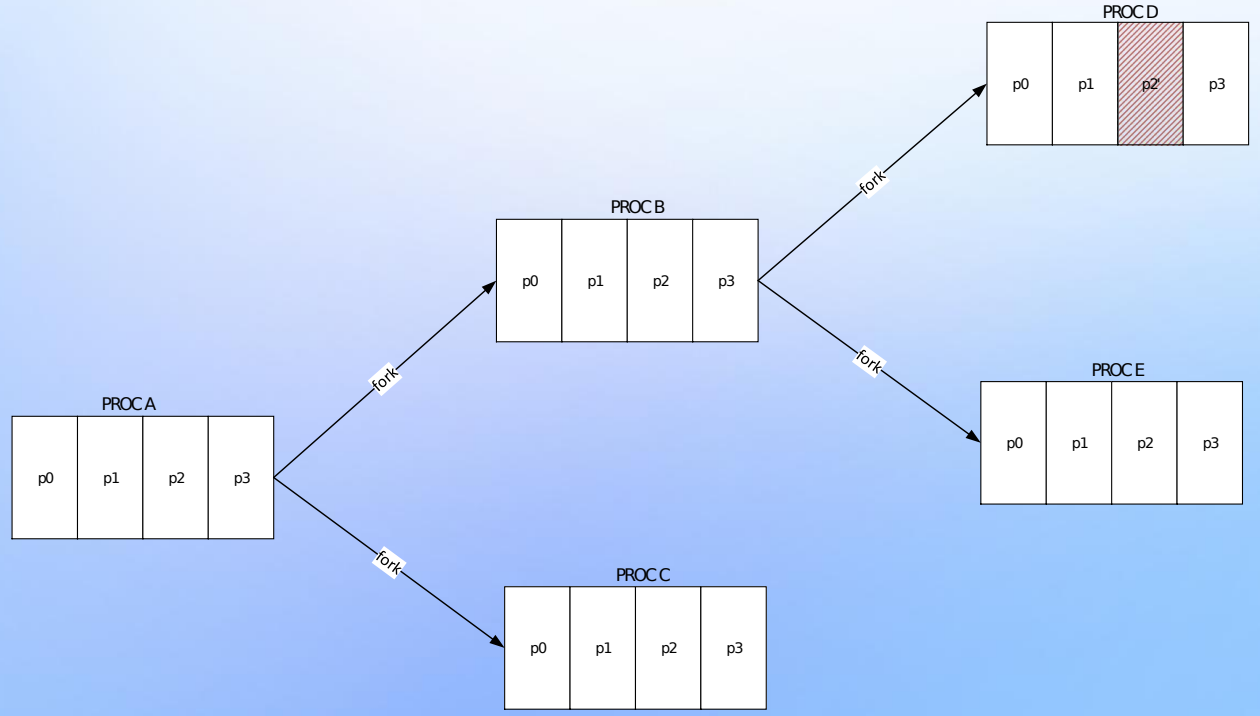



# anon_vma root锁竞争问题探讨

问题描述

对anon_vma相关的修改与访问，均要获取anon_vma->root->rwsem锁

如果发生COW，会怎样?




反向映射会出现两种情况

1. 如果rmap_walk老页P2，以A的anon_vma开始反向映射，找到ABCDE，依然会经过进程D的vma

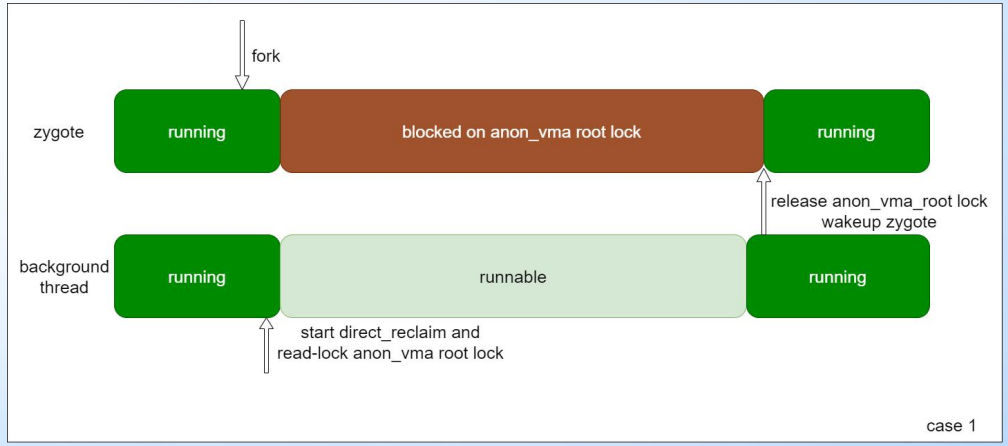
2. 如果rmap_walk新页P2'，以D的anon_vma开始走反向映射，只会找到D

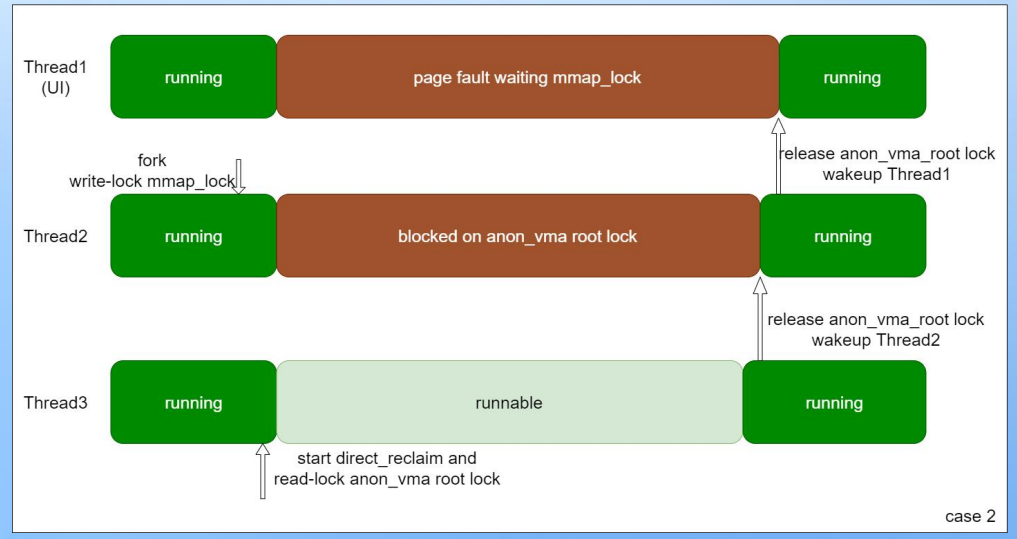
由于anon_vma的继承关系，不论是1或2，都需要read-lock A的anon_vma->rwsem，因为A-E的anon_vma->root始终是A。即使vma上所有page都完成COW，结果仍然不变

# anon_vma root锁竞争问题探讨

Android中所有app进程都由zygote fork出来，app再fork子进程，形成一棵庞大的anon_vma tree，共用anon_vma->root->rwsem，很容易出现锁竞争。

典型问题：
1. 内存回收和fork/exit产生竞争，导致进程创建或退出慢
2. anon_vma root lock往往与mmap_lock嵌套，引发更复杂的竞争问题

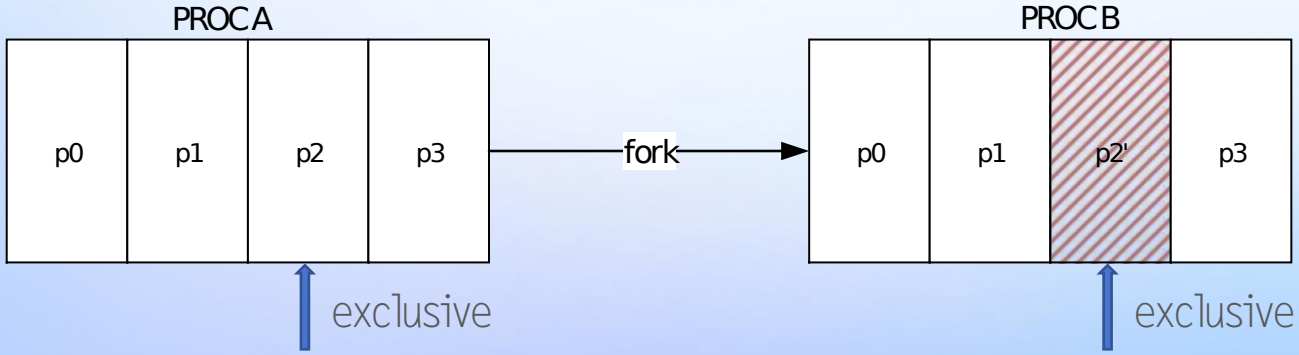


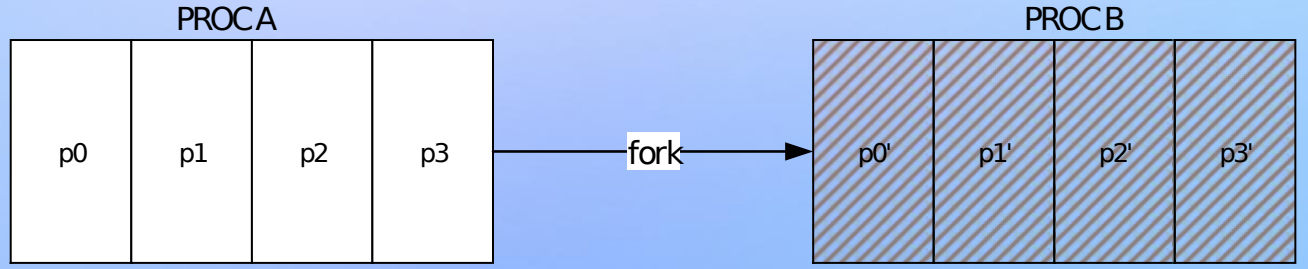
# anon_vma root锁竞争问题探讨

可能的优化思路

- 绝大多数匿名页是独占的（Android典型场景约95%），这类页面的反向映射依然会遍历rmap tree，lock root anon_vma，这实际上是不必要的。
- 针对独占非共享匿名页，引入per-anon_vma lock，避免竞争root lock大锁




B发生COW，P2和P2'变成独占page，反向映射无需遍历，可使用per-anon_vma lock，替代anon_vma root锁
**依赖folio被lock**，rmap期间保持独占




如果vma所有page全部发生COW，可直接脱离rmap tree。
难点：如何确认vma上所有page都发生COW

[DISCUSSION] anon_vma root lock contention and per anon_vma lock

CONTENTS



# Q&A

# THANKS