ArceOS 下线程安全的 Mimalloc 补充报告

完成情况说明

在之前的报告中，尝试了不使用 Mimalloc 完整算法来实现无锁的多线程支持，但是并没有成功。在接下来的数周，我们尝试参考 Mimalloc 完整算法的 C 语言版本 实现对应的 Rust 语言版本，但是实现细节与工作量超出我们最初的预期（最新版本 Mimalloc 核心代码超过5k 行），最终没能在期限前写完（已实现的部分在 `mimalloc_multithread_allocator` 约 3k 行）

Mimalloc 实现无锁多线程分析

基本设计

相比多数内存分配器实现了粗粒度的缓存，mimalloc 的特点是将缓存划分到各个线程中，内存分配都通过线程本地缓存，最大程度避免了内存分配时线程间的锁竞争。一个线程缓存包含 heap 和 segment 两部分，其中 heap 面向应用，而 segment 则负责向 kernel 申请大块内存供 heap 使用。某个线程申请的内存由本线程的 heap 管理，但是可能会由其他线程申请释放而导致多线程竞态，mimalloc 使用了特殊的创新设计来解决这一点。

heap 结构

page 是一组大小固定的 block 序列，而 heap 以 page 为单位来管理。mimalloc 对于每种 block 大小分类管理 page，每种 block 大小维护 page 的空闲列表 (free list / page queue)。为了加速查找有空闲 block 的 page，mimalloc 还设计了 full page queue 将填满的页放入。但产生了这样一个问题，当跨线程将一个满的页内释放，需要放回 page queue 很难无锁实现。Mimalloc 对此设计了 delayed_free 缓存队列，跨线程释放时先加入缓存队列，再由原线程定期处理。

Mimalloc 对 free list 进行了分片：供内存分配的 free list，本地线程释放的 local-free list，其它线程释放的 thread-free list。这样 thread-free 分片可以使得本线程与其它线程的 free 隔离，减少竞争的概率。而 local-free list 分片则提供了一种**心跳机制**，在 free list 用完之后从 local-free, thread-free, delay_free 中回收 block（如果不使用 local-free 则可能导致 free list 用完的时间间隔不确定）。thread_free 使用 CAS 解决竞争。

segment 结构

segment 是一块连续的大内存，以 64KB 为单位划分为 slice，在 heap 需要新 page 时拿走连续的 slice span。segment　同样按照不同的 slice span 大小分类管理，申请 page 时会选择尽量小的 slice span，并将多余的部分变为新的 slice span。

未来可行的方向

实际在实现 rust 版本的过程中，为了处理底层的大量内存操作而不得不在代码各处引入 unsafe，这似乎背离了 rust 内存安全的初衷。或许更好的方式是将 mimalloc 的 c 语言版本编译好后打包在 rust 中使用。实际上目前已经有了相关的实现 mimalloc - crates.io: Rust Package Registry