对Mimalloc在ArceOS下线程安全实现的一些思考

邵志航

2024

思考：Mimalloc如何实现线程安全

- Thread-local heaps
- Lock-free data structures
- Deferred reclamation
- Batch Processing

“mimalloc 中的页归属于线程局部的堆，其分配是本地完成的，因此无锁是理所当然的。但是，对于一个对象来说，任意线程都可以释放它。为了避免 free() 操作持锁，mimalloc 再一次为每页面切分 thread free list，并且把其他线程释放的对象都使用原子操作放到这个链表中。”

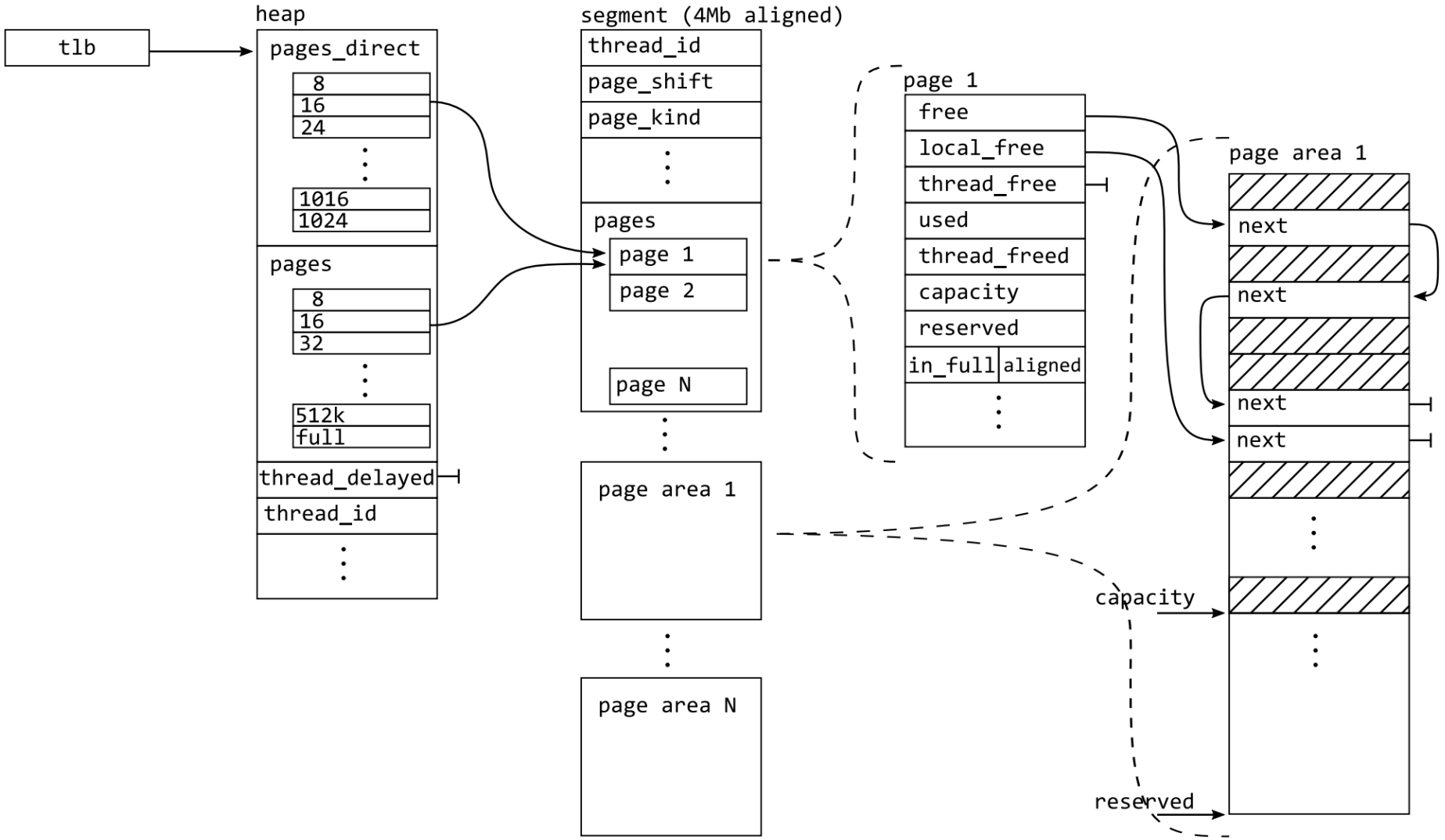
ini 代码解读 复制代码

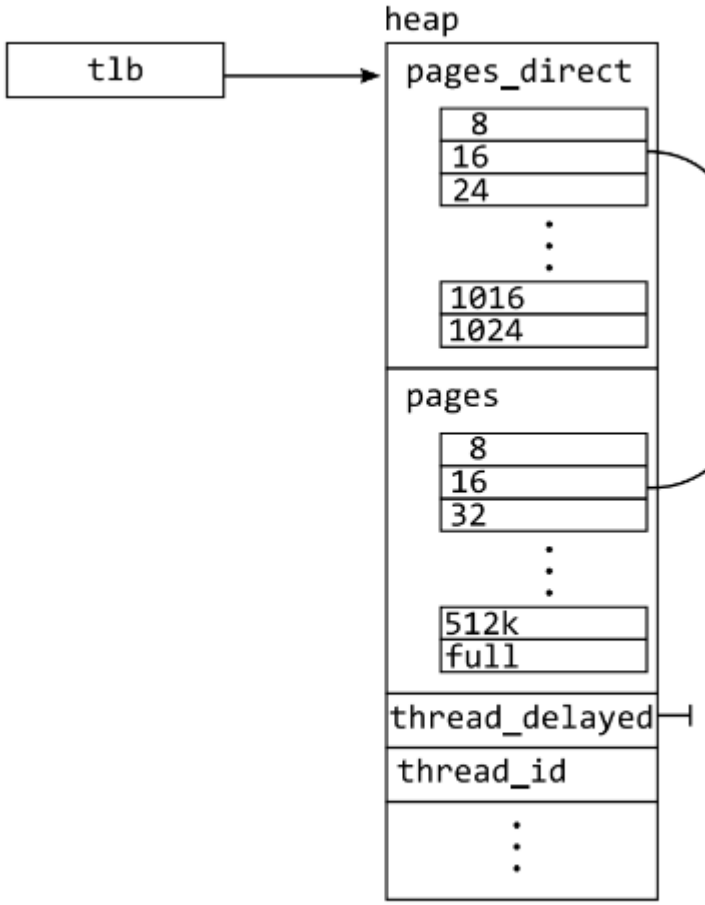
```
void free( void* p ) {
  //找到p所在的segment
  segment_t* segment = (segment_t*)((uintptr_t)p & ~(4*MB));
  if (segment==NULL) return;
  //找到p所在的page
  page_t* page = &segment->pages[(p - segment) >> segment->page_shift];
  block_t* block = (block_t*)p;

  if (thread_id() == segment->thread_id) { // free的是本线程分配的内存（local free）
    block->next = page->local_free;
    page->local_free = block;
    page->used--;
    if (page->used - page->thread_freed == 0) page_free(page);
  }
  else { // free的是其他线程分配的内存（non-local free）
    atomic_push( &page->thread_free, block);
    atomic_incr( &page->thread_freed );
  }
}
```

```c
// Free a block
// Fast path written carefully to prevent register spilling on the stack
void mi_free(void* p) mi_attr_noexcept
{
  mi_segment_t* const segment = mi_checked_ptr_segment(p,"mi_free");
  if mi_unlikely(segment==NULL) return;

  const bool is_local = (_mi_prim_thread_id() == mi_atomic_load_relaxed(&segment->thread_id));
  mi_page_t* const page = _mi_segment_page_of(segment, p);

  if mi_likely(is_local) {                      // thread-local free?
    if mi_likely(page->flags.full_aligned == 0) { // and it is not a full page (full pages need to move from the full bin), nor has aligned blocks (aligned blocks need to be unaligned)
      // thread-local, aligned, and not a full page
      mi_block_t* const block = (mi_block_t*)p;
      mi_free_block_local(page, block, true /* track stats */, false /* no need to check if the page is full */);
    }
    else {
      // page is full or contains (inner) aligned blocks; use generic path
      mi_free_generic_local(page, segment, p);
    }
  }
  else {
    // not thread-local; use generic path
    mi_free_generic_mt(page, segment, p);
  }
}
```

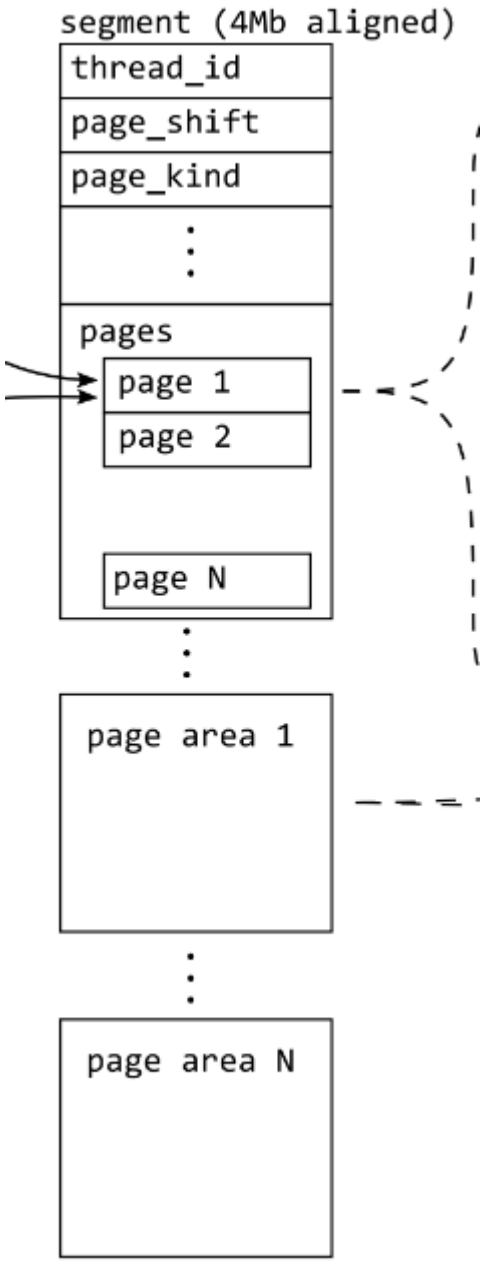
ArceOS下已有的工作：

```rust
pub struct GlobalAllocator {
    basic_alloc: Mutex<BasicAllocator<0>>,
    buddy_alloc: Mutex<BuddyByteAllocator>,
    slab_alloc: Mutex<SlabByteAllocator>,
    tlsf_c_alloc: Mutex<TLSFCAllocator>,
    tlsf_rust_alloc: Mutex<TLSFAllocator>,
    mi_alloc: Mutex<MiAllocator>,
    alloc_type: AllocType,
    heap_arddress: usize,
    heap_size: usize,
}

const PAGE_SIZE: usize = 1 << 22; // need 4MB aligned (prepared for mimalloc)
const HEAP_SIZE: usize = 1 << 26; // 512MB
/// The memory heap in this test
static mut HEAP: [usize; HEAP_SIZE + PAGE_SIZE] = [0; HEAP_SIZE + PAGE_SIZE];

/// The global allocator to test alloc in user mode.
/// The alloc mode supported: system_alloc, basic_alloc, buddy_alloc,
/// slab_alloc, tlsf_c_alloc and tlsf_rust_alloc.
/// The basic_alloc can choose strategy: first_fit, best_fit and worst_fit.
impl GlobalAllocator {
    pub const fn new() -> Self {
        Self {
            basic_alloc: Mutex::new(BasicAllocator::new()),
            buddy_alloc: Mutex::new(BuddyByteAllocator::new()),
            slab_alloc: Mutex::new(SlabByteAllocator::new()),
            tlsf_c_alloc: Mutex::new(TLSFCAllocator::new()),
            tlsf_rust_alloc: Mutex::new(TLSFAllocator::new()),
            mi_alloc: Mutex::new(MiAllocator::new()),
            alloc_type: AllocType::SystemAlloc,
            heap_arddress: 0,
            heap_size: 0,
        }
    }
```

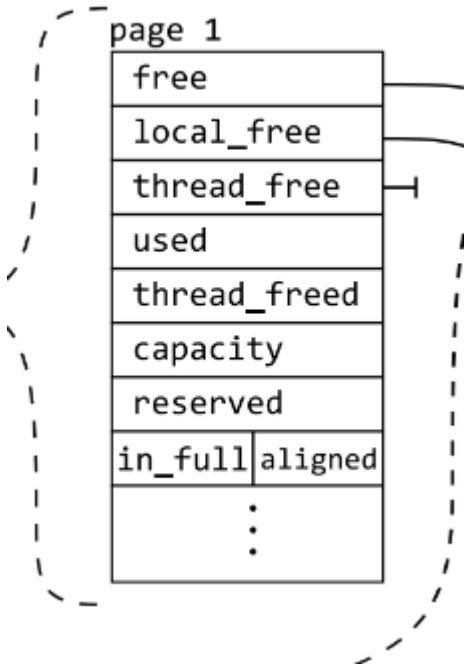

tlb
heap
pages_direct
8
16
24
1016
1024
pages
8
16
32
512k
full
thread_delayed
thread_id
segment (4Mb aligned)
thread_id
page_shift
page_kind
pages
page 1
page 2
page N
page area 1
page area N
page 1
free
local_free
thread_free
used
thread_freed
capacity
reserved
in_full
aligned
page area 1
next
next
next
next
capacity
reserved




```rust
/// mimalloc的heap结构
pub struct MiHeap {
    // page链表
    pub pages: [PagePointer; TOT_QUEUE_NUM],
}
```




```rust
pub struct Segment {
    // 把mi_heap藏在第一个段的开头
    pub mi_heap: MiHeap,
    // page种类
    pub page_kind: PageKind,
    // 段的大小
    pub size: usize,
    // 包含多少个page
    pub num_pages: usize,
    // 每个page的头结构
    pub pages: [Page; MAX_PAGE_PER_SEGMEGT],
    // padding，使空间对齐到8192
    pub padding: [usize; 434],
    // 接下来就是每个page的实际空间，注意第一个page会小一些
}
```




```rust
/// mimalloc的一个page控制头
#[derive(Clone, Copy)]
pub struct Page {
    // 块大小
    pub block_size: usize,
    // free链表
    pub free_list: BlockPointer,
    // page开始地址
    pub begin_addr: usize,
    // page结束地址
    pub end_addr: usize,
    // 尚未分配过的地址起点
    pub capacity: usize,
    // page链表中的上一项
    pub prev_page: PagePointer,
    // page链表中的下一项
    pub next_page: PagePointer,
    // 剩余块数
    pub free_blocks_num: usize,
}
```

TODO：

- 将C语言实现的Mimalloc官方仓库接入arceos
  - https://github.com/microsoft/mimalloc
- 实现Rust语言、支持多线程的Mimalloc
- 增加更多测试多线程的测例

接入C语言实现的Mimalloc库，可参考：

```rust
fn main() {
    let out_dir: String = env::var(key: "OUT_DIR").unwrap();

    Command::new(program: "cc").args(&["tests/test.c", "-O3","-c", "-fPIC", "-o"]) &mut Command
        .arg(&format!("{}/test.o", out_dir)) &mut Command
        .status().unwrap();

    Command::new(program: "ar").args(&["crus", "libtest.a", "test.o"]) &mut Command
        .current_dir(&Path::new(&out_dir)) &mut Command
        .status().unwrap();

    println!("cargo:rustc-link-search=native={}", out_dir);
    println!("cargo:rustc-link-lib=static=test");
}
```

```rust
use std::ffi::c_ulonglong;
#[link(name = "tlsf")]
extern {
    pub fn tlsf_create_with_pool(mem: c_ulonglong, bytes: c_ulonglong) -> c_ulonglong;
    pub fn tlsf_add_pool(tlsf: c_ulonglong, mem: c_ulonglong, bytes: c_ulonglong) -> c_ulonglong;

    pub fn tlsf_malloc(tlsf: c_ulonglong, bytes: c_ulonglong) -> c_ulonglong;//申请一段内存
    pub fn tlsf_memalign(tlsf: c_ulonglong, align: c_ulonglong, bytes: c_ulonglong) -> c_ulonglong;//申请一段内存，要求对齐到align
    pub fn tlsf_free(tlsf: c_ulonglong, ptr: c_ulonglong);//回收
}
```

增加更多测试多线程的用例，计划移植：
https://github.com/daanx/mimalloc-bench/blob/master/bench/glibc-bench/bench-malloc-thread.c

References：

1. https://github.com/daanx/mimalloc-bench
2. https://github.com/microsoft/mimalloc
3. https://www.bluepuni.com/archives/paper-reading-mimalloc-free-list-sharding-in-action
4. mimalloc-tr-v1.pdf
5. https://www.cnblogs.com/Five100Miles/p/12169392.html

Thanks