ArceOS中Mimalloc多线程实现最终报告

邵志航

2024

从中期报告之后的四周做了什么样的工作?

- Week 5: 利用已有接口接入了一个新的测例并运行。
- Week 6-8: 阅读ArceOS与mimalloc源码，并积极尝试加入多线程支持。

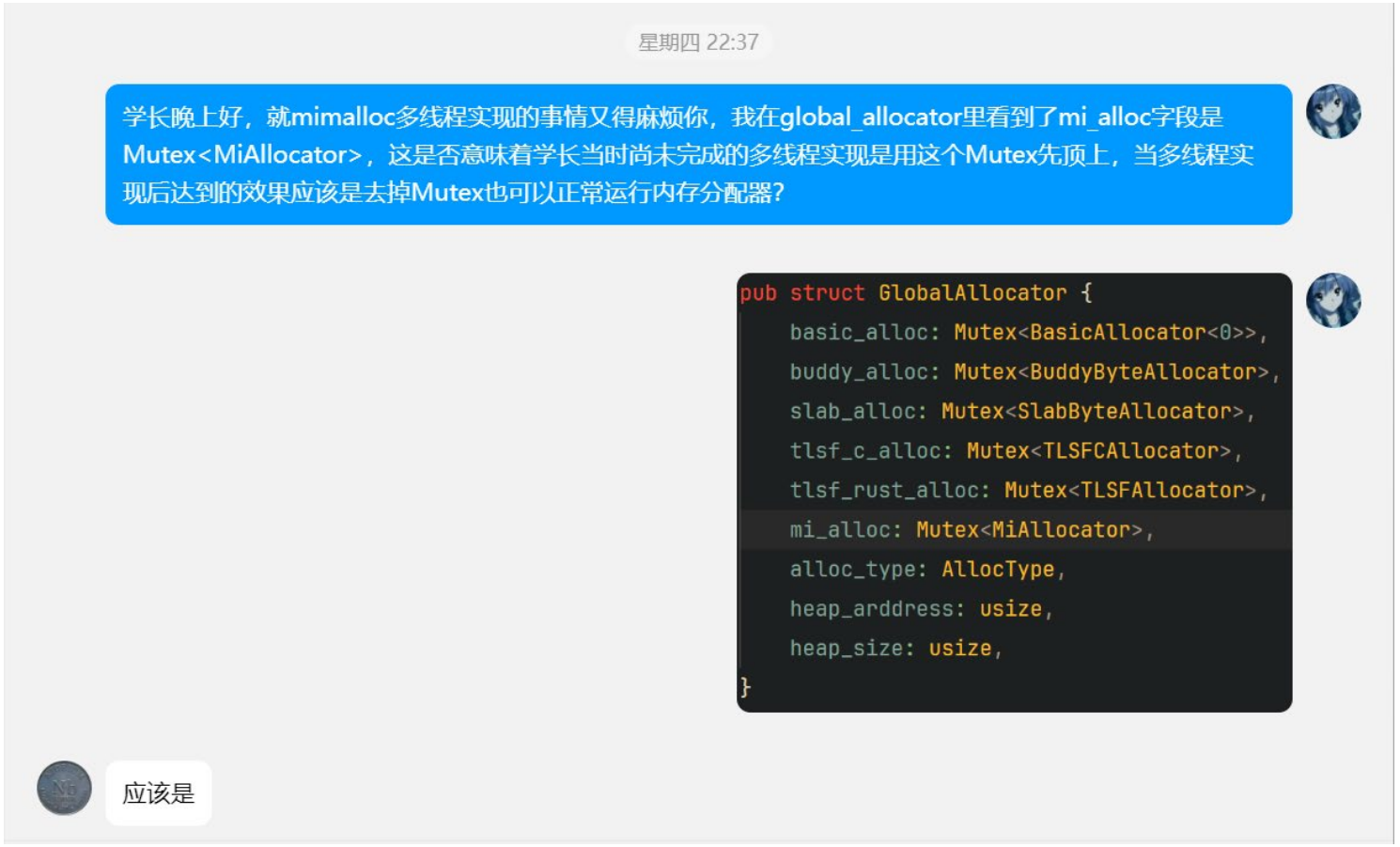
先说结论：
就目前为止，我们仍找不到一个好的方法实现无锁多线程支持。
但是也许有思路，只是仍需要时间去打磨和实现。

Week 5: 利用已有接口接入了一个新的测例并运行。

```rust
/// the start function of the glibc_bench_test
pub fn glibc_bench_test_start(
    cb1: CallBackMalloc,
    cb2: CallBackMallocAligned,
    cb3: CallBackFree,
);
/// the start function of the glibc_bench_test_simple
pub fn glibc_bench_simple_test_start(
    cb1: CallBackMalloc,
    cb2: CallBackMallocAligned,
    cb3: CallBackFree,
);
/// the start function of the multi_thread_c_test
pub fn multi_thread_c_test_start(
    cb1: CallBackMalloc,
    cb2: CallBackMallocAligned,
    cb3: CallBackFree,
);
```

```rust
pub fn glibc_bench_simple_test() {
    println!("Glibc bench simple test begin...");
    let t0 = std::time::Instant::now();
    allocator_test::glibc_bench_simple_test();
    let t1 = std::time::Instant::now();
    println!("time: {:#?}", t1 - t0);
    println!("Glibc bench simple test OK!");
}
```

Week 6-8:
阅读ArceOS与mimalloc源码，并积极尝试加入多线程支持。



```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: MiAllocator,
    //mi_alloc: Mutex<MiAllocator>,
    alloc_type: AllocType,
    heap_arddress: usize,
    heap_size: usize,
}
```

```
running 1 test
mi alloc test:
Basic alloc test begin...
time: 246.45466ms
Basic alloc test OK!
Mi alloc test begin...
time: 41.782906ms
Mi alloc test OK!
Align alloc test begin...
time: 10.252158ms
Align alloc test OK!
Malloc large test begin...
time: 8.466428ms
Malloc large test OK!
Glibc bench test begin...
time: 4.770976691s
Glibc bench test OK!
Glibc bench simple test begin...
time: 12.394596604s
Glibc bench simple test OK!
Multi thread memory allocation test begin.
panicked at crates/allocator/tests/global_allocator.rs:161:48:

thread panicked while processing panic. aborting.
error: test failed, to rerun pass `-p allocator --test test_main`

Caused by:
  process didn't exit successfully: `/kerneldebug/arceos/target/release/deps/test_main-577fffbd2af5da6c --nocapture` (signal: 6, SIGABRT: process abort signal)
root@ab148eb3a638:/kerneldebug/arceos#
```

```c
struct mi_heap_s {
  mi_tld_t*             tld;
  _Atomic(mi_block_t*)  thread_delayed_free;
  mi_threadid_t         thread_id;                           // thread this heap belongs too
  mi_arena_id_t         arena_id;                            // arena id if the heap belongs to a specific arena (or 0)
  mi_random_ctx_t       random;                              // random number context used for secure allocation
  size_t                page_count;                          // total number of pages in the `pages` queues.
  size_t                page_retired_min;                    // smallest retired index (retired pages are fully free, but still
in the page queues)
  size_t                page_retired_max;                    // largest retired index into the `pages` array.
  mi_heap_t*            next;                                // list of heaps per thread
  bool                  no_reclaim;                          // `true` if this heap should not reclaim abandoned pages
  uint8_t               tag;                                 // custom tag, can be used for separating heaps based on the object
types
  mi_page_t*            pages_free_direct[MI_PAGES_DIRECT];  // optimize: array where every entry points a page with possibly
free blocks in the corresponding queue for that size.
  mi_page_queue_t       pages[MI_BIN_FULL + 1];              // queue of pages for each size class (or "bin")
};
```

堆块具体结构的定义，其中堆块含有多个页，这符合先前实现里的定义。

```rust
/// mimalloc的heap结构
pub struct MiHeap {
    // page链表
    pub pages: [PagePointer; TOT_QUEUE_NUM],
    // thread id
    pub thread_id: usize,
    // thread delayed free
    pub thread_delayed_free: AtomicBlockPtr,
}
```

```c
typedef struct mi_page_s {
  // "owned" by the segment
  uint8_t               segment_idx;       // index in the segment `pages` array, `page == &segment->pages[page->segment_idx]`
  uint8_t               segment_in_use:1;  // `true` if the segment allocated this page
  uint8_t               is_committed:1;    // `true` if the page virtual memory is committed
  uint8_t               is_zero_init:1;    // `true` if the page was initially zero initialized
  uint8_t               is_huge:1;         // `true` if the page is in a huge segment

  // layout like this to optimize access in `mi_malloc` and `mi_free`
  uint16_t              capacity;          // number of blocks committed, must be the first field, see `segment.c:page_clear`
  uint16_t              reserved;          // number of blocks reserved in memory
  mi_page_flags_t       flags;             // `in_full` and `has_aligned` flags (8 bits)
  uint8_t               free_is_zero:1;    // `true` if the blocks in the free list are zero initialized
  uint8_t               retire_expire:7;   // expiration count for retired blocks

  mi_block_t*           free;              // list of available free blocks (`malloc` allocates from this list)
  mi_block_t*           local_free;        // list of deferred free blocks by this thread (migrates to `free`)
  uint16_t              used;              // number of blocks in use (including blocks in `thread_free`)
  uint8_t               block_size_shift;  // if not zero, then `(1 << block_size_shift) == block_size` (only used for fast path
in `free.c:_mi_page_ptr_unalign`)
  uint8_t               heap_tag;          // tag of the owning heap, used to separate heaps by object type
                                           // padding
  size_t                block_size;        // size available in each block (always `>0`)
  uint8_t*              page_start;        // start of the page area containing the blocks

  #if (MI_ENCODE_FREELIST || MI_PADDING)
  uintptr_t             keys[2];           // two random keys to encode the free lists (see `_mi_block_next`) or padding canary
  #endif

  _Atomic(mi_thread_free_t) xthread_free;  // list of deferred free blocks freed by other threads
  _Atomic(uintptr_t)        xheap;
```

```rust
/// mimalloc的一个page控制头
#[derive(Clone)]
pub struct Page {
    // 块大小
    pub block_size: usize,
    // free链表
    pub free_list: BlockPointer,
    // local free list
    pub local_free_list: BlockPointer,
    // thread free list
    pub thread_free_list: AtomicBlockPtr,
    // 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,
}
```

```c
typedef struct mi_segment_s {
  // constant fields
  mi_memid_t           memid;            // memory id to track provenance
  bool                 allow_decommit;
  bool                 allow_purge;
  size_t               segment_size;     // for huge pages this may be different from `MI_SEGMENT_SIZE`
  mi_subproc_t*        subproc;          // segment belongs to sub process

  // segment fields
  struct mi_segment_s* next;             // must be the first (non-constant) segment field  -- see `segment.c:segment_init`
  struct mi_segment_s* prev;
  bool                 was_reclaimed;    // true if it was reclaimed (used to limit on-free reclamation)

  size_t               abandoned;        // abandoned pages (i.e. the original owning thread stopped) (`abandoned <= used`)
  size_t               abandoned_visits; // count how often this segment is visited for reclaiming (to force reclaim if it is
too long)

  size_t               used;             // count of pages in use (`used <= capacity`)
  size_t               capacity;         // count of available pages (`#free + used`)
  size_t               segment_info_size;// space we are using from the first page for segment meta-data and possible guard
pages.
  uintptr_t            cookie;           // verify addresses in secure mode: `_mi_ptr_cookie(segment) == segment->cookie`

  struct mi_segment_s* abandoned_os_next; // only used for abandoned segments outside arena's, and only if
`mi_option_visit_abandoned` is enabled
  struct mi_segment_s* abandoned_os_prev;

  // layout like this to optimize access in `mi_free`
  _Atomic(mi_threadid_t) thread_id;      // unique id of the thread owning this segment
  size_t               page_shift;       // `1 << page_shift` == the page sizes == `page->block_size * page->reserved` (unless
the first page, then `-segment_info_size`).
  mi_page_kind_t       page_kind;        // kind of pages: small, medium, large, or huge
  mi_page_t            pages[1];         // up to `MI_SMALL_PAGES_PER_SEGMENT` pages
} mi_segment_t;
```

```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],
    /// thread_id
    pub thread_id: usize,
    // padding，使空间对齐到8192
    pub padding: [usize; 431],
    // 接下来就是每个page的实际空间，注意第一个page会小一些
}
```

```
running 1 test
mi alloc test:
Basic alloc test begin...
time: 2.829709758s
Basic alloc test OK!
Mi alloc test begin...
time: 10.936271344s
Mi alloc test OK!
Align alloc test begin...
thread 'test_start' panicked at crates/allocator_test/src/align_test.rs:33:13:
align not correct.
stack backtrace:
   0: rust_begin_unwind
             at /rustc/ed04567ba1d5956d1080fb8121caa005ce059e12/library/std/src/panicking.rs:665:5
   1: core::panicking::panic_fmt
             at /rustc/ed04567ba1d5956d1080fb8121caa005ce059e12/library/core/src/panicking.rs:74:14
   2: allocator_test::align_test::align_test
             at /kerneldebug/arceos/crates/allocator_test/src/align_test.rs:33:13
   3: test_main::align_test
             at ./tests/test_main.rs:83:5
   4: test_main::mi_alloc_test
             at ./tests/test_main.rs:395:5
   5: test_main::test_start
             at ./tests/test_main.rs:418:5
   6: test_main::test_start::{{closure}}
             at ./tests/test_main.rs:409:16
   7: core::ops::function::FnOnce::call_once
             at /rustc/ed04567ba1d5956d1080fb8121caa005ce059e12/library/core/src/ops/function.rs:250:5
   8: core::ops::function::FnOnce::call_once
             at /rustc/ed04567ba1d5956d1080fb8121caa005ce059e12/library/core/src/ops/function.rs:250:5
note: Some details are omitted, run with `RUST_BACKTRACE=full` for a verbose backtrace.
error: test failed, to rerun pass `-p allocator --test test_main`

Caused by:
  process didn't exit successfully: `/kerneldebug/arceos/target/debug/deps/test_main-046ad87eb4743645 --nocapture` (signal: 11, SIGSEGV: invalid memory reference)
```

```
error[E0204]: the trait `core::marker::Copy` cannot be implemented for this type
  --> crates/mimalloc_allocator/src/data.rs:62:17
   |
62 | #[derive(Clone, Copy)]
   |                 ^^^^
...
71 |     pub thread_free_list: AtomicPtr<BlockPointer>,
   |     ------------------------------------------- this field does not implement `core::marker::Copy`
   |
   = note: this error originates in the derive macro `Copy` (in Nightly builds, run with -Z macro-backtrace for more info)

error[E0277]: the trait bound `AtomicPtr<data::BlockPointer>: Clone` is not satisfied
   --> crates/mimalloc_allocator/src/data.rs:71:5
    |
62  | #[derive(Clone, Copy)]
    |          ----- in this derive macro expansion
...
71  |     pub thread_free_list: AtomicPtr<BlockPointer>,
    |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ the trait `Clone` is not implemented for `AtomicPtr<data::BlockPointer>`
    |
note: required by a bound in `AssertParamIsClone`
   --> /root/.rustup/toolchains/nightly-x86_64-unknown-linux-gnu/lib/rustlib/src/rust/library/core/src/clone.rs:198:34
    |
198 | pub struct AssertParamIsClone<T: Clone + ?Sized> {
    |                                  ^^^^^ required by this bound in `AssertParamIsClone`
    = note: this error originates in the derive macro `Clone` (in Nightly builds, run with -Z macro-backtrace for more info)

Some errors have detailed explanations: E0204, E0277.
For more information about an error, try `rustc --explain E0204`.
error: could not compile `mimalloc_allocator` (lib) due to 2 previous errors
```

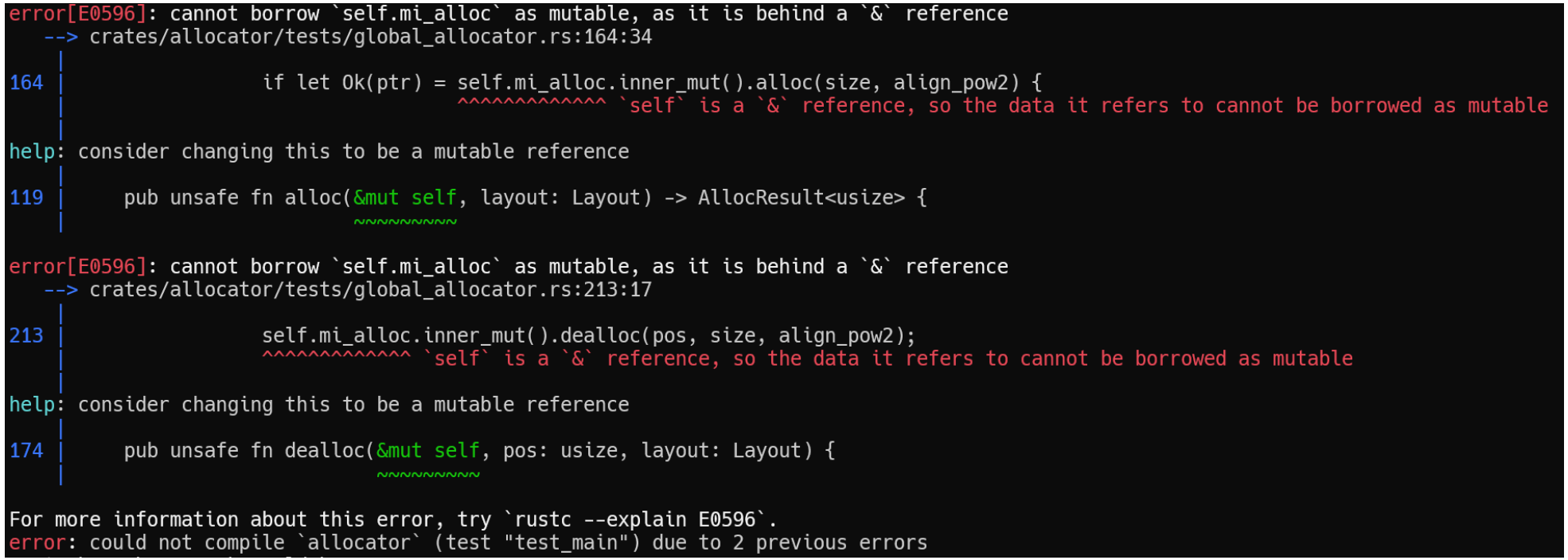
```rust
pub struct AtomicBlockPtr {
    pub ptr: AtomicPtr<BlockPointer>,
}

impl AtomicBlockPtr {
    fn new(ptr: *mut BlockPointer) -> Self {
        AtomicBlockPtr {
            ptr: AtomicPtr::new(ptr)
        }
    }
}

impl Clone for AtomicBlockPtr {
    fn clone(&self) -> Self {
        let atomic_ptr_clone = self.ptr.load(Ordering::SeqCst);
        let new_atomic_ptr = AtomicPtr::new(atomic_ptr_clone);

        AtomicBlockPtr {
            ptr: AtomicPtr::new(atomic_ptr_clone)
        }

    }
}
```


error[E0596]: cannot borrow `self.mi_alloc` as mutable, as it is behind a `&` reference
--> crates/allocator/tests/global_allocator.rs:164:34
164 | if let Ok(ptr) = self.mi_alloc.inner_mut().alloc(size, align_pow2) {
^^^^^^^^^^^^^^ `self` is a `&` reference, so the data it refers to cannot be borrowed as mutable
help: consider changing this to be a mutable reference
119 | pub unsafe fn alloc(&mut self, layout: Layout) -> AllocResult<usize> {
~~~~~~~~~~
error[E0596]: cannot borrow `self.mi_alloc` as mutable, as it is behind a `&` reference
--> crates/allocator/tests/global_allocator.rs:213:17
213 | self.mi_alloc.inner_mut().dealloc(pos, size, align_pow2);
^^^^^^^^^^^^^^ `self` is a `&` reference, so the data it refers to cannot be borrowed as mutable
help: consider changing this to be a mutable reference
174 | pub unsafe fn dealloc(&mut self, pos: usize, layout: Layout) {
~~~~~~~~~~
For more information about this error, try `rustc --explain E0596`.
error: could not compile `allocator` (test "test_main") due to 2 previous errors

几个解决方案：
1.RefCell<T>：提供内部可变性。

但是这样的方法行不通。因为会有Already borrowed: BorrowMutError。
这是很正常的事情，因为RefCell本身就不保证线程安全。

2.Miallocator的多线程改造。

需要参考RefCell进行操作，但是又不能引入锁。

```rust
type BorrowFlag = AtomicIsize;
/// A byte-granularity memory allocator based on the [mimalloc_allocator] written by rust
pub struct MiAllocator {
    //Atomic...
    borrow: BorrowFlag,
    data: UnsafeCell<MiAllocatorInner>,
}

pub struct MiAllocatorInner {
    inner: Option<Heap>,
}

impl MiAllocator {
    /// Creates a new empty `TLSFAllocator`.
    pub const fn new() -> Self {
        Self {
            borrow: AtomicIsize::new(0),
            data: UnsafeCell::new(MiAllocatorInner::new())
        }
    }

    pub fn inner_mut(&mut self) -> &mut MiAllocatorInner {
        self.data.get_mut()
    }

    pub fn inner(&self) -> &MiAllocatorInner {
            let ptr = self.data.get();
            let reference: &MiAllocatorInner = unsafe {
                &*ptr
            };
            reference
    }
}
```

未来的工作：
1.至少先完善数据结构，能够运行起多线程测例。
2.现有增加的mimalloc内部字段如何使用仍需要探索。

有鉴于此，也许这需要另外的八周去完成，也许需要更长的时间。

Thanks