Теория и практика многопоточного программирования

Неганов Алексей

Московский физико-технический институт (национальный исследовательский университет)
Кафедра теоретической и прикладной информатики

Москва 2020

RMW-операции: ABA

```
struct node {
      struct node *next;
}

static struct node *top = NULL;

void push(struct node *n) {
      do {
            struct node *t = top;
            n->next = t;
      } while (!CAS(&top, t, n));
}

void struct node *pop(void) {
      struct node *next;
      do {
            struct node *t = top;
            if (t == NULL)
                  return NULL;
            next = t->next;
      } while (!CAS(&top,t,next));
      return t;
}
```

```
Thread #1:                     Thread #2:

struct node *a = pop();        struct node *b = pop();
                               pop();
                               push(b);
```

Tagged pointers

```
// linux/include/linux/rbtree.h

struct rb_node {
    unsigned long  __rb_parent_color;
    struct rb_node *rb_right;
    struct rb_node *rb_left;
} __attribute__((aligned(sizeof(long))));

#define rb_parent(r)   ((struct rb_node *)((r)->__rb_parent_color & ~3))
```

Рисунок 1 – Простой пример использования свободных бит в указателе

Tagged pointers

```
// boost/lockfree/detail/tagged_ptr_ptrcompression.hpp

template <class T>
class tagged_ptr
{
    typedef boost::uint64_t compressed_ptr_t;
public:
    typedef boost::uint16_t tag_t;
private:
    union cast_unit {
        compressed_ptr_t value;
        tag_t tag[4];
    };
    static const int tag_index = 3;
    static const compressed_ptr_t ptr_mask = 0xffffffffffffUL; //(1L<<48L)-1;
    static T* extract_ptr(volatile compressed_ptr_t const & i) {
        return (T*)(i & ptr_mask);
    }
    static tag_t extract_tag(volatile compressed_ptr_t const & i) {
        cast_unit cu;
        cu.value = i;
        return cu.tag[tag_index];
    }
```

Рисунок 2 – Пример использования старших 16 бит

Tagged pointers: пример использования

```
template <typename T> struct node {
    tagged_ptr next;
    T data;
} ;

template <typename T> class MSQueue {
   tagged_ptr<T> volatile m_Head;
   tagged_ptr<T> volatile m_Tail;

   FreeList m_FreeList;
public:
   MSQueue() {
     m_Head.ptr = m_Tail.ptr = new node();
   }
};
```

Tagged pointers: пример использования

```
void enqueue( T const& value ) {
E1: node * pNode = m_FreeList.newNode();
E2: pNode->data = value;
E3: pNode->next.ptr = nullptr;
E4: for (;;) {
E5:   tagged_ptr<T> tail = m_Tail;
E6:   tagged_ptr<T> next = tail.ptr->next;
E7:   if tail == m_Tail {
E8:      if next.ptr == nullptr {
E9:         if CAS(&tail.ptr->next, next, tagged_ptr<T>(node, next.tag+1)) {
E10:          break; // success!
            }
E11:     } else {
E12:        CAS(&m_Tail, tail, tagged_ptr<T>(next.ptr, tail.tag+1)); // tail is not the last
         }
      }
    } // end loop
E13: CAS(&m_Tail, tail, tagged_ptr<T>(pNode, tail.tag+1)); // tail to inserted
}
```

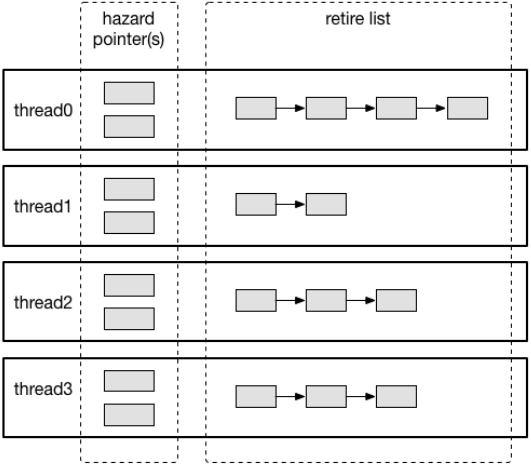
Tagged pointers: пример использования

```
bool dequeue( T& dest ) {
D1:  for (;;) {
D2:    tagged_ptr<T> head = m_Head;
D3:    tagged_ptr<T> tail = m_Tail;
D4:    tagged_ptr<T> next = head->next;
D5:    if ( head == m_Head ) {
D6:       if ( head.ptr == tail.ptr ) {
D7:         if (next.ptr == nullptr )
D8:              return false; // queue is empty
D9:         CAS(&m_Tail, tail, tagged_ptr<T>(next.ptr, tail.tag+1>)); // tail to last
D10:      } else {
D11:        dest = next.ptr->data;
D12:        if (CAS(&m_Head, head, tagged_ptr<T>(next.ptr, head.tag+1))
D13:           break // success
          }
       }
     } // end of loop
D14: m_FreeList.add(head.ptr);
D15: return true;
}
```

Epoch-based reclamation

```
static atomic<unsigned int> m_nGlobalEpoch := 1 ;

const EPOCH_COUNT = 3 ;

struct ThreadEpoch {
    unsigned int m_nThreadEpoch ;
    // deferred for reclamation
    List<void *> m_arrRetired[ EPOCH_COUNT ] ;

    ThreadEpoch(): m_nThreadEpoch(1) {}

    void enter() {
       if ( m_nThreadEpoch <= m_nGlobalEpoch )
          m_nThreadEpoch = m_nGlobalEpoch + 1 ;
    }

    void exit() {
       if ( each thread epoch is greater than m_nGlobalEpoch ) {
          ++m_nGlobalEpoch ;

          // for each thread free elements
          m_arrRetired[ (m_nGlobalEpoch - 1) % EPOCH_COUNT ]
       }
    }
} ;
```

Hazard pointers



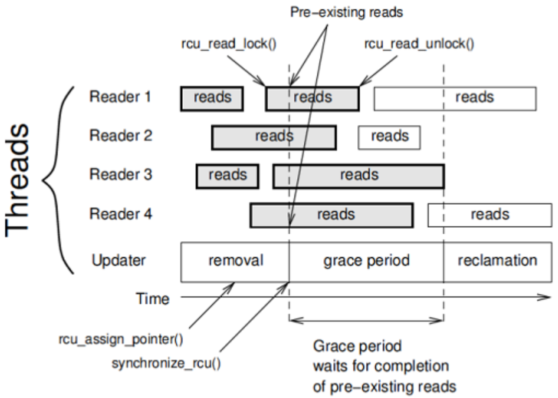
`https://github.com/concurrencykit/ck/blob/master/src/ck_hp.c`

Hazard pointers

```
// per-thread HP array
// only owner can write, everybody can read
void *HP[HP_NUM];

unsigned dcount = 0; // dlist used
void* dlist[BATCH_SIZE]; // to delete

void RetireNode(void *node) {
  dlist[dcount++] = node;
  if (dcount == BATCH_SIZE)
     Scan();
}

void Scan() {
   unsigned p=0;
   void * plist[HP_NUM*THR_NUM];

   // Stage 1: collect all HPs
   for (unsigned t=0; t < THR_NUM; ++t) {
      void **pHPThread = get_thread_data(t)->HP;
      for (unsigned i = 0; i < HP_NUM; ++i) {
         void *hptr = pHPThread[i];
         if ( hptr != nullptr )
            plist[p++] = hptr;
      }
   }

   // Stage 2: sort HPs (prepare the search)
   sort(plist);

   // Stage 3: delete all non-hazard items
   unsigned new_dcount = 0;
   void *new_dlist[HP_NUM*THR_NUM];
   for ( i = 0; i < BATCH_SIZE; ++i ) {
      if ( binary_search(dlist[i], plist))
         new_dlist[new_dcount++] = dlist[i];
      else
         free(dlist[i]); // non-hazard
   }

   // Stage 4: reinitialize dlist
   for (i = 0; i < new_dcount; ++i )
      dlist[i] = new_dlist[i];
   dcount = new_dcount;
}
```

RCU

```
rcu_read_lock( ... );
stuff = find_the_stuff(args);
do_something_with(args);
rcu_read_unlock();
```



RCU

```
void rcu_read_lock(void) {}

void rcu_read_unlock(void) {}

void call_rcu(void (*callback) (void *), void *arg) {
    // add callback/arg pair to a list
}

void synchronize_rcu(void) {
    int cpu, ncpus = 0;

    for_each_cpu(cpu)
        schedule_current_task_to(cpu);

        for each entry in the call_rcu list
            entry->callback (entry->arg);
}
```

User-space RCU

- Quiescent-State-Based Reclamation RCU
- General-Purpose URCU
- RCU via Signal Handling