CSE 511: Operating Systems Design

Lectures 24

Lock-free data structures

Exercise

- Modify the lock-free stack implementation using ABA tags
  - Can use something like:

```
struct aba_ptr_s {
    void *ptr;
    size_t tag;
};

aba_ptr_new = { .ptr = new_ptr, .tag = aba_ptr_old.tag + 1 };

CAS(&aba_ptr_global, aba_ptr_old, aba_ptr_new)
```

- Use two stacks
  - free_stack contains deallocated/free nodes
  - alloc_stack contains allocated nodes (normal stack)

Memory Reclamation

- Still **unsafe** to return memory from free_stack back to the OS when only using the ABA tags but can recycle past nodes
  - Why?
- Special approaches exist to solve the problem:
  - Epoch-based reclamation
  - Hazard pointers
  - Hazard eras (combines the above two schemes)
  - etc

Memory Reclamation

- Will use **epoch-based reclamation**

  Enclose the code which access shared memory (it is very similar to lock() and unlock() but does not block)

  start_op(thread_id); // Creates a thread reservation

  ...

  end_op(thread_id); // Clears a thread reservation

  Instead of free(), call **retire**(). free() will be called automatically when it is safe to do so.

Michael-Scott's Lock-Free Queue

```
struct node_s { // Queue Element
    struct node_s *next; // Next
    void *obj; // Associated Object
};

typedef struct node_s node_t;

// Head and Tail pointers
// Create one sentinel node initially
node_t* head = malloc(sizeof(node_t);
node_t* tail = head;
head->next = NULL;
```

Michael-Scott's Lock-Free Queue

```
void enqueue(void *obj) {
    node_t *node = malloc(sizeof(node_t);
    node->obj = obj;
    node->next = NULL;
    while (true) {
        node_t *t = LOAD(tail);
        node_t *next = t->next;
        if (t == LOAD(tail)) {
            if (next == NULL) {
                if (CAS(&t->next, next, node)) break;
            } else {
                CAS(&tail, t, next); // Help to move tail
            }
        }
    }
    CAS(&tail, t, next); // Try to move tail (one time)
}
```

Michael-Scott's Lock-Free Queue

```
void *dequeue() {
    while (true) {
        node_t *h = LOAD(head);
        node_t *t = LOAD(tail);
        node_t *next = h->next;
    I   if (h == LOAD(head)) {
            if (h == t) {
                if (next == NULL) return NULL; // Empty
                CAS(&tail, t, next);
            } else {
                void *obj = next->value; // Read before the update
                if (CAS(&head, h, next))
                    return obj; // Return the object
            }
        }
    }
}
```