CSE 511: Operating Systems Design

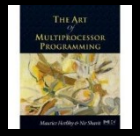
Lectures 23

CAS

Lock-free data structures

Compare-and-Set

```
bool compareAndSet(long *globalVal, long oldValue,
      long newValue)
{
   /* Done atomically (as a single instruction)! */
   long prior = *globalVal;
   if (prior == oldValue) {
      *globalVal = newValue;
      return true;
   } else {
      return false;
   }
}
```



Treiber's Lock-Free Stack

```
struct node_s { // Stack Element
    struct node_s *next; // Next
    void *obj; // Associated Object
};
struct node_s* stack = NULL; // Top of the stack
```

Treiber’s Lock-Free Stack

```
void push(void *obj)
{
    struct node_s *node = malloc(sizeof(struct node_s));
    node->obj = obj;
    do {
        node->next = LOAD(stack);
    } while (!CAS(&stack, node->next, node));
}
```

Treiber's Lock-Free Stack

```
void *pop()
{
    void *obj = NULL;
    while (true) {
        struct node_s *node = LOAD(stack);
        if (node == NULL) break;
        if (CAS(&stack, node, node->next)) {
            obj = node->obj;
            // Free node (can still be used by other threads)
            break;
        }
    }
    return obj;
}
```

ABA Problem

- Multiple threads may access elements simultaneously
  - Deallocation of an element by one thread should not result in memory access violations by other threads that access the same element
- To solve the ABA problem with CAS, each pointer is associated with a unique tag which is incremented each time a corresponding pointer is updated
  - Need to update 16 bytes rather than 8 bytes
    - 8 bytes for the pointer
    - 8 bytes for the tag
  - A special instruction ***cmpxchg16b*** exists in x86-64

Exercise

- Modify the above implementation using ABA tags
- 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

Michael-Scott's Lock-Free Queue

```
struct node_s { // Queue Element
    struct node_s *next; // Next
    void *obj; // Associated Object
};
// Head and Tail pointers (one sentinel node)
struct node_s* head = malloc(sizeof(struct node_s);
struct node_s* tail = head;
head->next = NULL;
```