【OS】Day36

Class	Operating System: Three Easy Pieces
Date	@February 15, 2022

【Ch29】Lock-based Data Structure(2)

Scaling Linked Lists

One technique that researchers have explored to enable more concurrency within a list is something called hand-over-hand locking.

Instead of having a single lock for the entire list, we instead add a lock per node of the list. When traversing the list, the code first grabs the next node's lock and then releases the current node's lock.

However, this approach in practice is hard to make such a structure faster than the simple single lock approach, as the overheads of acquiring and releasing locks for each node of a list traversal is prohibitive.

29.3 Concurrent Queues

```
1  typedef struct __node_t {
2      int                 value;
3      struct __node_t     *next;
4  } node_t;
5
6  typedef struct __queue_t {
7      node_t              *head;
8      node_t              *tail;
9      pthread_mutex_t     head_lock, tail_lock;
10 } queue_t;
11
12 void Queue_Init(queue_t *q) {
13     node_t *tmp = malloc(sizeof(node_t));
14     tmp->next = NULL;
15     q->head = q->tail = tmp;
16     pthread_mutex_init(&q->head_lock, NULL);
17     pthread_mutex_init(&q->tail_lock, NULL);
18 }
19
20 void Queue_Enqueue(queue_t *q, int value) {
21     node_t *tmp = malloc(sizeof(node_t));
22     assert(tmp != NULL);
23     tmp->value = value;
24     tmp->next  = NULL;
25
26     pthread_mutex_lock(&q->tail_lock);
27     q->tail->next = tmp;
28     q->tail = tmp;
29     pthread_mutex_unlock(&q->tail_lock);
30 }
31
32 int Queue_Dequeue(queue_t *q, int *value) {
33     pthread_mutex_lock(&q->head_lock);
34     node_t *tmp = q->head;
35     node_t *new_head = tmp->next;
36     if (new_head == NULL) {
37         pthread_mutex_unlock(&q->head_lock);
38         return -1; // queue was empty
39     }
40     *value = new_head->value;
41     q->head = new_head;
42     pthread_mutex_unlock(&q->head_lock);
43     free(tmp);
44     return 0;
45 }
```

Figure 29.9: **Michael and Scott Concurrent Queue**

In this data structure, there are two locks, one for the head of the queue, and one for the tail. The goal of these two locks is to enable concurrency of enqueue and dequeue operations.

In the common case, the enqueue routine will only access the tail lock, and dequeue only the head lock.

29.4 Concurrent Hash Table

```
1  #define BUCKETS (101)
2
3  typedef struct __hash_t {
4      list_t lists[BUCKETS];
5  } hash_t;
6
7  void Hash_Init(hash_t *H) {
8      int i;
9      for (i = 0; i < BUCKETS; i++)
10         List_Init(&H->lists[i]);
11 }
12
13 int Hash_Insert(hash_t *H, int key) {
14     return List_Insert(&H->lists[key % BUCKETS], key);
15 }
16
17 int Hash_Lookup(hash_t *H, int key) {
18     return List_Lookup(&H->lists[key % BUCKETS], key);
19 }
```

Figure 29.10: **A Concurrent Hash Table**