Spin-Lock with Atomic operations

Concurrent Programming

Introduction

- Atomic Operations
- Warming up using simple atomic instruction
  - fetch and add
- Implementing Spin-lock
  - with TAS
  - with TTAS
  - CLH Lock (do it yourself)

Atomic Operations

- Atomic operations provide instructions that execute ***atomically*** without interruption

- A processor can simultaneously read a location and write it in the same bus operation

https://gcc.gnu.org/onlinedocs/gcc-4.4.4/gcc/Atomic-Builtins.html

Fetch-and-Add

```
__sync_fetch_and_add(type *ptr, type v)
```

- Atomically, adds the value of *v* to the **ptr* and returns previous value of **ptr*
- Full memory barrier is created when this function is invoked
- Compare performance with the mutex practice in lab2

Fetch-and-Add

```
 1 #include <stdio.h>
 2 #include <pthread.h>
 3
 4 #define NUM_THREAD      10
 5 #define NUM_INCREMENT   1000000
 6
 7 bool flag[NUM_THREAD * NUM_INCREMENT];
 8
 9 int cnt_global = 0;
10
11 void* ThreadFunc(void* arg) {
12     int ticket = 0;
13     for (int i = 0; i < NUM_INCREMENT; i++) {
14         ticket = __sync_fetch_and_add(&cnt_global, 1);
15         flag[ticket] = true;
16     }
17 }
```

Fetch-and-Add

```
19 int main(void) {
20     pthread_t threads[NUM_THREAD];
21
22     int i;
23     for (i = 0; i < NUM_THREAD; i++) {
24         if (pthread_create(&threads[i], 0, ThreadFunc, NULL) < 0) {
25             return 0;
26         }
27     }
28     for (i = 0; i < NUM_THREAD; i++) {
29         pthread_join(threads[i], NULL);
30     }
31     for (i = 0; i < NUM_THREAD * NUM_INCREMENT; i++) {
32         if (flag[i] == false) {
33             printf("ERROR!!\n");
34             break;
35         }
36     }
37     if (i == NUM_THREAD * NUM_INCREMENT) {
38         printf("ALL FLAGS ON!\n");
39     }
40     printf("global count: %d\n", cnt_global);
41
42     return 0;
43 }
```

Fetch-and-Add

```
mrbin2002@ubuntu:~/TA/Multicore/lab10$ time ./prac_fetch_and_add
ALL FLAGS ON!
global count: 10000000

real    0m0.306s
user    0m1.004s
sys     0m0.068s
```

```
mrbin2002@ubuntu:~/TA/Multicore/lab2$ time ./prac_mutex
thread 140555014108928, local count: 1000000
thread 140555005716224, local count: 1000000
thread 140554997323520, local count: 1000000
thread 140554988930816, local count: 1000000
thread 140554980538112, local count: 1000000
thread 140554972145408, local count: 1000000
thread 140554963752704, local count: 1000000
thread 140554955360000, local count: 1000000
thread 140554946967296, local count: 1000000
thread 140554938574592, local count: 1000000
global count: 10000000

real    0m1.124s
user    0m1.464s
sys     0m2.848s
```

lab 2 – result of mutex practice

Test-and-Set

```
__sync_test_and_set(type *ptr, type v)
```

- Atomically, writes *v* into **ptr* and returns previous value of **ptr*
- An acquire memory barrier is created when this function is invoked

Test-and-Set Lock

```
 1 #include <stdio.h>
 2 #include <pthread.h>
 3
 4 #define NUM_THREAD      8
 5 #define NUM_WORK        1000000
 6
 7 int cnt_global;
 8 int gap[128]; // to allocate cnt_global & object_tas in different cache line
 9 int object_tas;
10
11 void lock(int* lock_object) {
12     while (__sync_lock_test_and_set(lock_object, 1) == 1) {}
13 }
14
15 void unlock(int* lock_object) {
16     *lock_object = 0;
17     __sync_synchronize();
18 }
19
20 void* Work(void* args) {
21     for (int i = 0; i < NUM_WORK; i++) {
22         lock(&object_tas);
23         cnt_global++;
24         unlock(&object_tas);
25     }
26 }
```

Test-and-Set Lock

```
28 int main(void) {
29     pthread_t threads[NUM_THREAD];
30
31     for (long i = 0; i < NUM_THREAD; i++) {
32         pthread_create(&threads[i], 0, Work, 0);
33     }
34     for (int i = 0; i < NUM_THREAD; i++) {
35         pthread_join(threads[i], 0);
36     }
37     printf("cnt_global: %d\n", cnt_global);
38 }
```

```
mrbin2002@ubuntu:~/TA/Multicore/lab10$ time ./prac_taslock
cnt_global: 8000000

real    0m2.102s
user    0m8.304s
sys     0m0.000s
```

[result of TAS lock]

Test-and-Test-and-Set Lock

```
 1 #include <stdio.h>
 2 #include <pthread.h>
 3
 4 #define NUM_THREAD      8
 5 #define NUM_WORK        1000000
 6
 7 int cnt_global;
 8 int gap[128]; // to allocate cnt_global & object_tas in different cache line
 9 int object_ttas;
10
11 void lock(int* lock_object) {
12     while (1) {
13         while (*lock_object == 1) {}
14         if (__sync_lock_test_and_set(lock_object, 1) == 0) {
15             break;
16         }
17     }
18 }
19
20 void unlock(int* lock_object) {
21     *lock_object = 0;
22     __sync_synchronize();
23 }
```

Test-and-Test-and-Set Lock

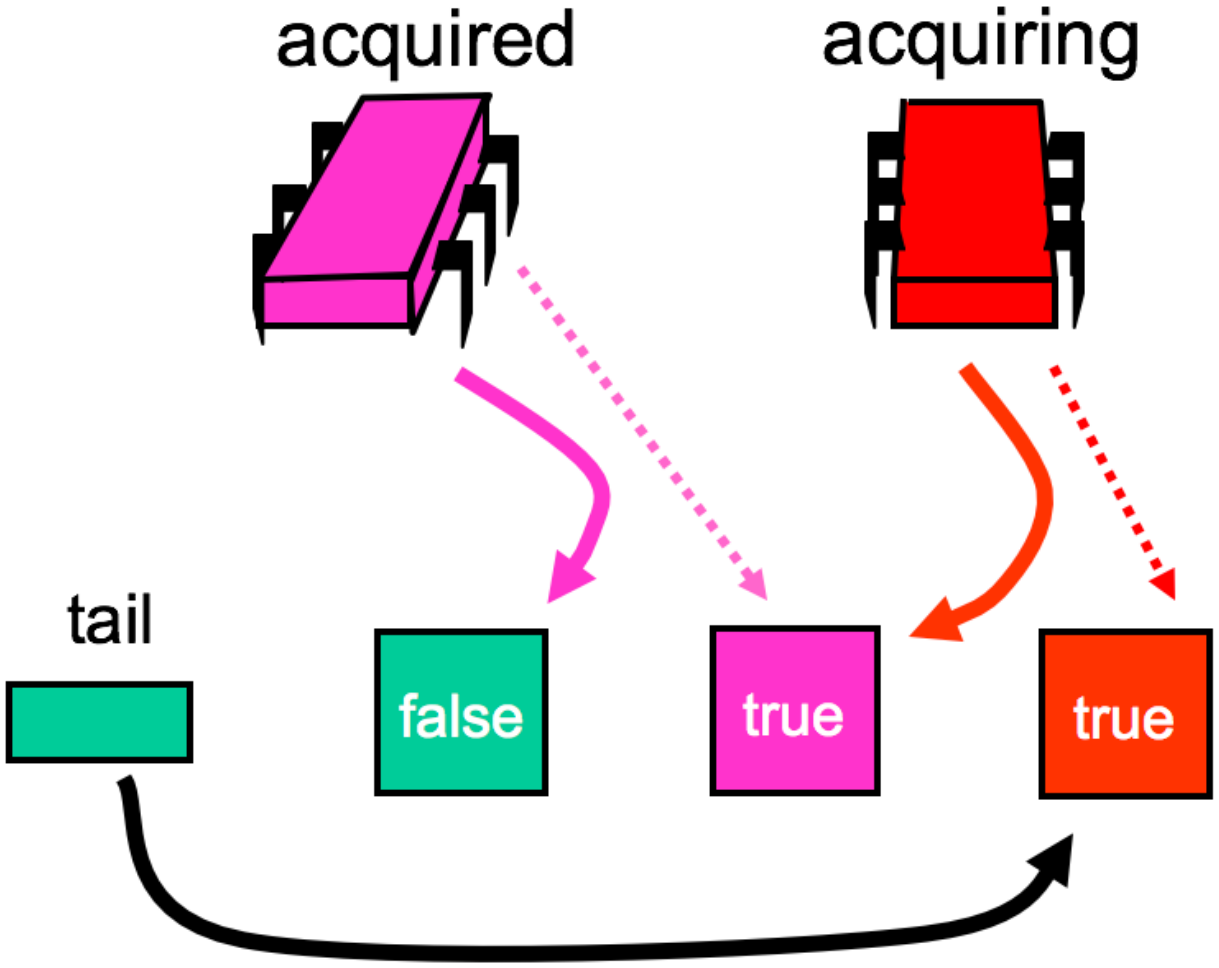
```
25 void* Work(void* args) {
26     for (int i = 0; i < NUM_WORK; i++) {
27         lock(&object_ttas);
28         cnt_global++;
29         unlock(&object_ttas);
30     }
31 }
32
33 int main(void) {
34     pthread_t threads[NUM_THREAD];
35
36     for (long i = 0; i < NUM_THREAD; i++) {
37         pthread_create(&threads[i], 0, Work, 0);
38     }
39     for (int i = 0; i < NUM_THREAD; i++) {
40         pthread_join(threads[i], 0);
41     }
42     printf("cnt_global: %d\n", cnt_global);
43 }
```

```
mrbin2002@ubuntu:~/TA/Multicore/lab10$ time ./prac_ttaslock
cnt_global: 8000000

real    0m1.576s
user    0m6.188s
sys     0m0.012s
```

[result of TTAS lock]

CLH Lock (do it yourself)



Thank You