Spin Locks

Concurrent Programming

Introduction

- Test-And-SetAtomic Operation
- Implementing Spin-lock
  - with TAS
  - with TTAS
  - CLH Lock (make 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

Test-and-Set

```
__sync_lock_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

[prac_taslock.cpp]

```
 1 #include <stdio.h>
 2 #include <pthread.h>
 3
 4 #define NUM_THREAD      16
 5 #define NUM_WORK        1000000
 6
 7 int cnt_global;
 8 /* to allocate cnt_global & object_tas in different cache lines */
 9 int gap[128];
10 int object_tas;
11
12 void lock(int* lock_object) {
13     while (__sync_lock_test_and_set(lock_object, 1) == 1)  {}
14 }
15
16 void unlock(int* lock_object) {
17     *lock_object = 0;
18     __sync_synchronize();
19 }
20
21 void* thread_work(void* args) {
22     for (int i = 0; i < NUM_WORK; i++) {
23         lock(&object_tas);
24         cnt_global++;
25         unlock(&object_tas);
26     }
27 }
```

Test-and-Set Lock

```
29 int main(void) {
30     pthread_t threads[NUM_THREAD];
31
32     for (int i = 0; i < NUM_THREAD; i++) {
33         pthread_create(&threads[i], NULL, thread_work, NULL);
34     }
35     for (int i = 0; i < NUM_THREAD; i++) {
36         pthread_join(threads[i], NULL);
37     }
38     printf("cnt_global: %d\n", cnt_global);
39
40     return 0;
41 }
```

```
$ g++ -o prac_taslock prac_taslock.cpp -lpthread
```

```
jongbin@multicore-96:~/TA/Multicore/lab10$ time ./prac_taslock
cnt_global: 16000000

real    0m9.279s
user    2m14.808s
sys     0m0.000s
```

[result of TAS lock]

Test-and-Test-and-Set Lock

[prac_ttaslock.cpp]

```cpp
#include <stdio.h>
#include <pthread.h>

#define NUM_THREAD      16
#define NUM_WORK        1000000

int cnt_global;
/* to allocate cnt_global & object_tas in different cache lines */
int gap[128];
int object_ttas;

void lock(int* lock_object) {
    while (1) {
        while (*lock_object == 1) {}
        if (__sync_lock_test_and_set(lock_object, 1) == 0)
            break;
    }
}

void unlock(int* lock_object) {
    *lock_object = 0;
    __sync_synchronize();
}
```

Test-and-Test-and-Set Lock

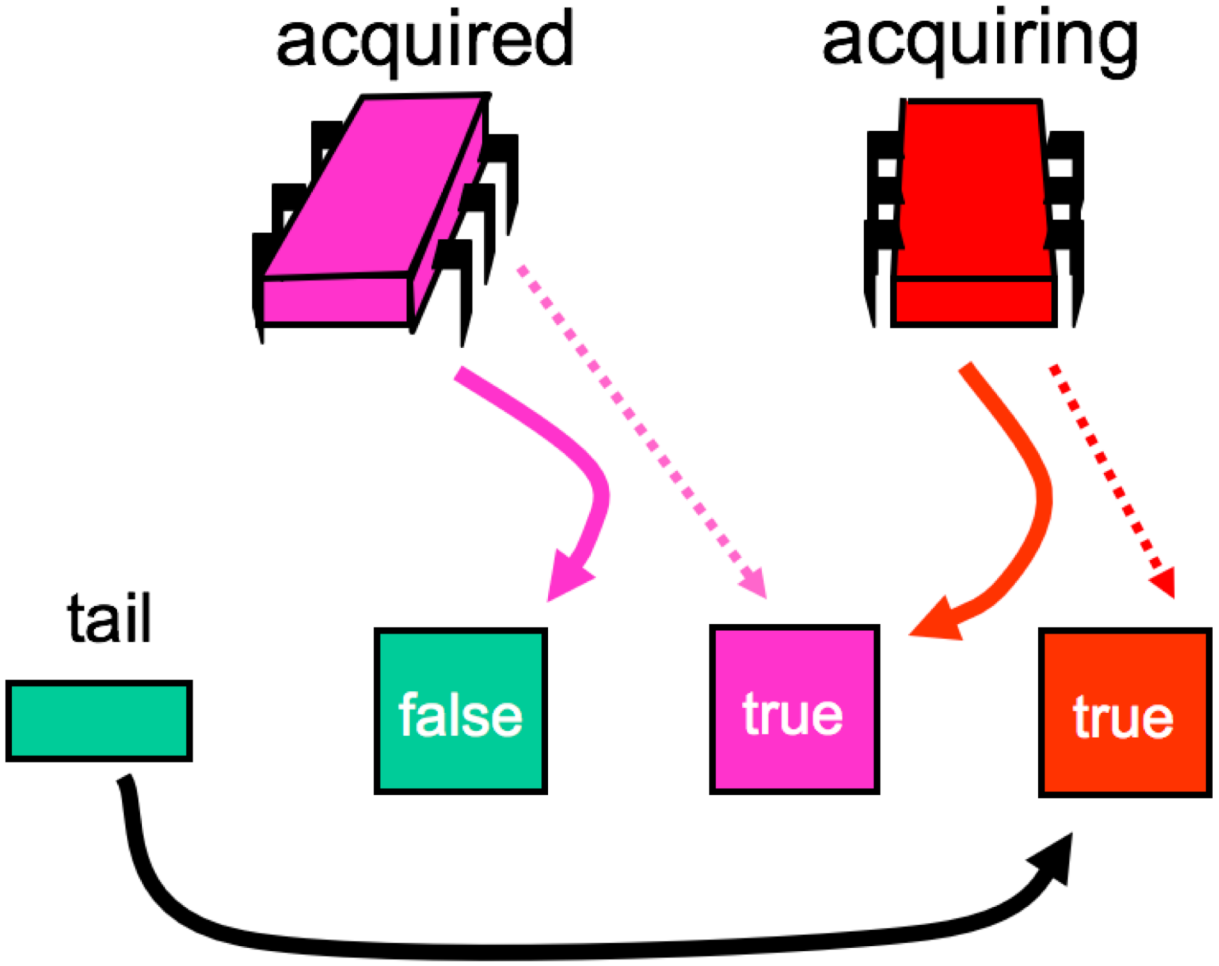
```
25 void* thread_work(void* args) {
26     for (int i = 0; i < NUM_WORK; i++) {
27         lock(&object_ttas);
28         cnt_global++;
29         unlock(&object_ttas);
30     }
31 }
32
33 int main(void) {
34     pthread_t threads[NUM_THREAD];
35
36     for (int i = 0; i < NUM_THREAD; i++) {
37         pthread_create(&threads[i], NULL, thread_work, NULL);
38     }
39     for (int i = 0; i < NUM_THREAD; i++) {
40         pthread_join(threads[i], NULL);
41     }
42     printf("cnt_global: %d\n", cnt_global);
43
44     return 0;
45 }
```

```
$ g++ -o prac_ttaslock prac_ttaslock.cpp -lpthread
```

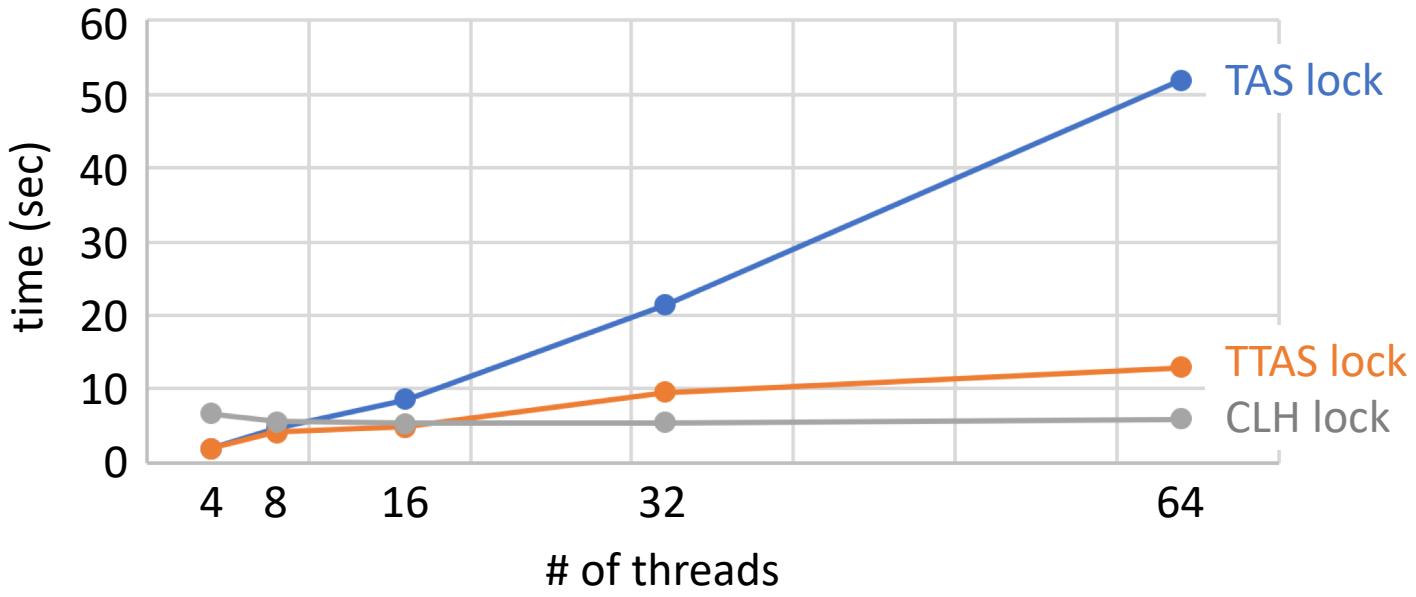
```
jongbin@multicore-96:~/TA/Multicore/lab10$ time ./prac_ttaslock
cnt_global: 16000000

real    0m6.676s
user    1m34.420s
sys     0m0.008s
```

[result of TTAS lock]

CLH Lock (do it yourself)



Experimental Result



# of CPU: 4 sockets x 24 cores = 96 cores

Number of total lock&unlock : 16,000,000

Thank You