OSTEP Chapter 28

ECE 3600, Fall 2022

Table of Contents

1. Locks
2. Simple Flag
3. Test-and-Set
4. Other Hardware Primitives
5. Exercises
6. Q2
7. Q4
8. Q5
9. Q6

1. Locks

mutex = mutual exclusion, only one thread can hold the lock at any time

```
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
...
Pthread_mutex_lock(&lock); // wrapper; exits on failure
balance = balance + 1;
Pthread_mutex_unlock(&lock);
```

How to implement?

Issues: Correctness (even on multiprocessors), Performance (time overhead), Fairness (no starve)

2. Simple Flag

```
1  typedef struct __lock_t { int flag; } lock_t;
2
3  void init(lock_t *mutex) {
4      // 0 -> lock is available, 1 -> held
5      mutex->flag = 0;
6  }
7
8  void lock(lock_t *mutex) {
9      while (mutex->flag == 1)  // TEST the flag
10         ; // spin-wait (do nothing)
11     mutex->flag = 1;          // now SET it!
12 }
13
14 void unlock(lock_t *mutex) {
15     mutex->flag = 0;
16 }
```

Figure 28.1: **First Attempt: A Simple Flag**

Correct: no

Assume flag=0 to begin:

Thread 1	**Thread 2**
call `lock()`	
while (flag == 1)	
interrupt: switch to Thread 2	
	call `lock()`
	while (flag == 1)
	flag = 1;
	interrupt: switch to Thread 1
flag = 1; // set flag to 1 (too!)	

Figure 28.2: **Trace: No Mutual Exclusion**

3. Test-and-Set

Hardware test-and-set instruction (atomic exchange):

```
int TestAndSet(int *ptr, int new);
```

returns the old value and simultaneously updates to the new value.

```
1  typedef struct __lock_t {
2      int flag;
3  } lock_t;
4
5  void init(lock_t *lock) {
6      // 0: lock is available, 1: lock is held
7      lock->flag = 0;
8  }
9
10 void lock(lock_t *lock) {
11     while (TestAndSet(&lock->flag, 1) == 1)
12         ; // spin-wait (do nothing)
13 }
14
15 void unlock(lock_t *lock) {
16     lock->flag = 0;
17 }
```

Figure 28.3: **A Simple Spin Lock Using Test-and-set**

Correct: yes; Performance: bad (spinning); Fairness: no

Fix performance --> yield; Fix fairness --> queue

4. Other Hardware Primitives

Compare-and-swap:

```
int CompareAndSwap(int *ptr, int expected, int new);
```

returns the old value and simultaneously updates to the new value if old == expected.

Fetch-and-add:

```
int FetchAndAdd(int *ptr);
```

returns the old value and simultaneously adds 1 and stores the new value.

Can be used to implement ticket lock and ensure fairness.

```
1  typedef struct __lock_t {
2      int ticket;
3      int turn;
4  } lock_t;
5
6  void lock_init(lock_t *lock) {
7      lock->ticket = 0;
8      lock->turn   = 0;
9  }
10
11 void lock(lock_t *lock) {
12     int myturn = FetchAndAdd(&lock->ticket);
13     while (lock->turn != myturn)
14         ; // spin
15  }
16
17 void unlock(lock_t *lock) {
18     lock->turn = lock->turn + 1;
19 }
```

Figure 28.7: **Ticket Locks**

5. Exercises

See the book for exercises using x86.py

```
$ cat flag.s
.var flag
.var count

.main
.top

.acquire
mov  flag, %ax      # get flag
test $0, %ax        # if we get 0 back: lock is free!
jne  .acquire       # if not, try again
mov  $1, flag       # store 1 into flag

# critical section
mov  count, %ax     # get the value at the address
add  $1, %ax        # increment it
mov  %ax, count     # store it back

# release lock
mov  $0, flag       # clear the flag now

# see if we're still looping
sub  $1, %bx
test $0, %bx
jgt .top

halt
```

6. Q2

```
$ python ./x86.py -p flag.s -R ax,bx -a bx=2 -M flag,count -c

 flag count      ax    bx          Thread 0                Thread 1

    0     0       0     2
    0     0       0     2   1000 mov  flag, %ax
    0     0       0     2   1001 test $0, %ax
    0     0       0     2   1002 jne  .acquire
    1     0       0     2   1003 mov  $1, flag
    1     0       0     2   1004 mov  count, %ax
    1     0       1     2   1005 add  $1, %ax
    1     1       1     2   1006 mov  %ax, count
    0     1       1     2   1007 mov  $0, flag
    0     1       1     1   1008 sub  $1, %bx
    0     1       1     1   1009 test $0, %bx
    0     1       1     1   1010 jgt .top
    0     1       0     1   1000 mov  flag, %ax
    0     1       0     1   1001 test $0, %ax
    0     1       0     1   1002 jne  .acquire
    1     1       0     1   1003 mov  $1, flag
    1     1       1     1   1004 mov  count, %ax
    1     1       2     1   1005 add  $1, %ax
    1     2       2     1   1006 mov  %ax, count
    0     2       2     1   1007 mov  $0, flag
    0     2       2     0   1008 sub  $1, %bx
    0     2       2     0   1009 test $0, %bx
    0     2       2     0   1010 jgt .top
    0     2       2     0   1011 halt
    0     2       0     2   ----- Halt;Switch -----  ----- Halt;Switch -----
    0     2       0     2                           1000 mov  flag, %ax
    0     2       0     2                           1001 test $0, %ax
    0     2       0     2                           1002 jne  .acquire
    1     2       0     2                           1003 mov  $1, flag
    1     2       2     2                           1004 mov  count, %ax
    1     2       3     2                           1005 add  $1, %ax
    1     3       3     2                           1006 mov  %ax, count
    0     3       3     2                           1007 mov  $0, flag
    0     3       3     1                           1008 sub  $1, %bx
    0     3       3     1                           1009 test $0, %bx
    0     3       3     1                           1010 jgt .top
    0     3       0     1                           1000 mov  flag, %ax
    0     3       0     1                           1001 test $0, %ax
    0     3       0     1                           1002 jne  .acquire
    1     3       0     1                           1003 mov  $1, flag
    1     3       3     1                           1004 mov  count, %ax
    1     3       4     1                           1005 add  $1, %ax
    1     4       4     1                           1006 mov  %ax, count
    0     4       4     1                           1007 mov  $0, flag
    0     4       4     0                           1008 sub  $1, %bx
    0     4       4     0                           1009 test $0, %bx
    0     4       4     0                           1010 jgt .top
    0     4       4     0                           1011 halt
```

7. Q4

```
$ python ./x86.py -p flag.s -R ax,bx -a bx=2 -M flag,count -c -i 6
```

flag	count	ax	bx	Thread 0	Thread 1
0	0	0	2		
0	0	0	2	1000 mov flag, %ax	
0	0	0	2	1001 test $0, %ax	
0	0	0	2	1002 jne .acquire	
1	0	0	2	1003 mov $1, flag	
1	0	0	2	1004 mov count, %ax	
1	0	1	2	1005 add $1, %ax	
1	0	0	2	------ Interrupt ------	------ Interrupt ------
1	0	1	2		1000 mov flag, %ax
1	0	1	2		1001 test $0, %ax
1	0	1	2		1002 jne .acquire
1	0	1	2		1000 mov flag, %ax
1	0	1	2		1001 test $0, %ax
1	0	1	2		1002 jne .acquire
1	0	1	2	------ Interrupt ------	------ Interrupt ------
1	1	1	2	1006 mov %ax, count	
0	1	1	2	1007 mov $0, flag	
0	1	1	1	1008 sub $1, %bx	
0	1	1	1	1009 test $0, %bx	
0	1	1	1	1010 jgt .top	
0	1	0	1	1000 mov flag, %ax	
0	1	1	2	------ Interrupt ------	------ Interrupt ------
0	1	0	2		1000 mov flag, %ax
0	1	0	2		1001 test $0, %ax
0	1	0	2		1002 jne .acquire
1	1	0	2		1003 mov $1, flag
1	1	1	2		1004 mov count, %ax
1	1	2	2		1005 add $1, %ax
1	1	0	1	------ Interrupt ------	------ Interrupt ------

flag	count	ax	bx	Thread 0	Thread 1
1	1	0	1	1001 test $0, %ax	
1	1	0	1	1002 jne .acquire	
1	1	0	1	1003 mov $1, flag	
1	1	1	1	1004 mov count, %ax	
1	1	2	1	1005 add $1, %ax	
1	2	2	1	1006 mov %ax, count	
1	2	2	2	------ Interrupt ------	------ Interrupt ------
0	2	2	2		1006 mov %ax, count
0	2	2	1		1007 mov $0, flag
0	2	2	1		1008 sub $1, %bx
0	2	2	1		1009 test $0, %bx
0	2	0	1		1010 jgt .top
0	2	2	1	------ Interrupt ------	------ Interrupt ------
0	2	2	0	1007 mov $0, flag	
0	2	2	0	1008 sub $1, %bx	
0	2	2	0	1009 test $0, %bx	
0	2	2	0	1010 jgt .top	
0	2	2	0	1011 halt	
0	2	0	1	----- Halt;Switch -----	----- Halt;Switch -----
0	2	0	1		1001 test $0, %ax
0	2	0	1	------ Interrupt ------	------ Interrupt ------
0	2	0	1		1002 jne .acquire
1	2	0	1		1003 mov $1, flag
1	2	2	1		1004 mov count, %ax
1	2	3	1		1005 add $1, %ax
1	3	3	1		1006 mov %ax, count
0	3	3	1		1007 mov $0, flag
0	3	3	1	------ Interrupt ------	------ Interrupt ------
0	3	3	0		1008 sub $1, %bx
0	3	3	0		1009 test $0, %bx
0	3	3	0		1010 jgt .top
0	3	3	0		1011 halt

8. Q5

```
$ cat test-and-set.s
.var mutex
.var count

.main
.top

.acquire
mov  $1, %ax
xchg %ax, mutex     # atomic swap of 1 and mutex
test $0, %ax        # if we get 0 back: lock is free!
jne  .acquire       # if not, try again

# critical section
mov  count, %ax     # get the value at the address
add  $1, %ax        # increment it
mov  %ax, count     # store it back

# release lock
mov  $0, mutex

# see if we're still looping
sub  $1, %bx
test $0, %bx
jgt .top

halt
```

9. Q6

```
$ python ./x86.py -p test-and-set.s -R ax,bx -a bx=2 -M mutex,count -c -i 6
```

```
mutex count      ax    bx          Thread 0                Thread 1
    0     0       0     2
    0     0       1     2   1000 mov  $1, %ax
    1     0       0     2   1001 xchg %ax, mutex
    1     0       0     2   1002 test $0, %ax
    1     0       0     2   1003 jne  .acquire
    1     0       0     2   1004 mov  count, %ax
    1     0       1     2   1005 add  $1, %ax
    1     0       0     2   ------ Interrupt ------  ------ Interrupt ------
    1     0       1     2                           1000 mov  $1, %ax
    1     0       1     2                           1001 xchg %ax, mutex
    1     0       1     2                           1002 test $0, %ax
    1     0       1     2                           1003 jne  .acquire
    1     0       1     2                           1000 mov  $1, %ax
    1     0       1     2                           1001 xchg %ax, mutex
    1     0       1     2   ------ Interrupt ------  ------ Interrupt ------
    1     1       1     2   1006 mov  %ax, count
    0     1       1     2   1007 mov  $0, mutex
    0     1       1     1   1008 sub  $1, %bx
    0     1       1     1   1009 test $0, %bx
    0     1       1     1   1010 jgt .top
    0     1       1     1   1000 mov  $1, %ax
    0     1       1     2   ------ Interrupt ------  ------ Interrupt ------
    0     1       1     2                           1002 test $0, %ax
    0     1       1     2                           1003 jne  .acquire
    0     1       1     2                           1000 mov  $1, %ax
    1     1       0     2                           1001 xchg %ax, mutex
    1     1       0     2                           1002 test $0, %ax
    1     1       0     2                           1003 jne  .acquire
    1     1       1     1   ------ Interrupt ------  ------ Interrupt ------
    1     1       1     1   1001 xchg %ax, mutex
    1     1       1     1   1002 test $0, %ax
    1     1       1     1   1003 jne  .acquire
    1     1       1     1   1000 mov  $1, %ax
    1     1       1     1   1001 xchg %ax, mutex
    1     1       1     1   1002 test $0, %ax
    1     1       0     2   ------ Interrupt ------  ------ Interrupt ------
```

```
mutex count      ax    bx          Thread 0                Thread 1
    1     1       1     2
    1     1       2     2                           1004 mov  count, %ax
    1     2       2     2                           1005 add  $1, %ax
    0     2       2     2                           1006 mov  %ax, count
    0     2       2     1                           1007 mov  $0, mutex
    0     2       2     1                           1008 sub  $1, %bx
    0     2       2     1                           1009 test $0, %bx
    0     2       1     1   ------ Interrupt ------  ------ Interrupt ------
    0     2       1     1   1003 jne  .acquire
    0     2       1     1   1000 mov  $1, %ax
    1     2       0     1   1001 xchg %ax, mutex
    1     2       0     1   1002 test $0, %ax
    1     2       0     1   1003 jne  .acquire
    1     2       2     1   1004 mov  count, %ax
    1     2       2     1   ------ Interrupt ------  ------ Interrupt ------
    1     2       2     1                           1010 jgt .top
    1     2       1     1                           1000 mov  $1, %ax
    1     2       1     1                           1001 xchg %ax, mutex
    1     2       1     1                           1002 test $0, %ax
    1     2       1     1                           1003 jne  .acquire
    1     2       1     1                           1000 mov  $1, %ax
    1     2       2     1   ------ Interrupt ------  ------ Interrupt ------
    1     2       3     1   1005 add  $1, %ax
    1     3       3     1   1006 mov  %ax, count
    0     3       3     1   1007 mov  $0, mutex
    0     3       3     0   1008 sub  $1, %bx
    0     3       3     0   1009 test $0, %bx
    0     3       3     0   1010 jgt .top
    0     3       1     1   ------ Interrupt ------  ------ Interrupt ------
    1     3       0     1                           1001 xchg %ax, mutex
    1     3       0     1                           1002 test $0, %ax
    1     3       0     1                           1003 jne  .acquire
    1     3       3     1                           1004 mov  count, %ax
    1     3       4     1                           1005 add  $1, %ax
    1     4       4     1                           1006 mov  %ax, count
    1     4       3     0   ------ Interrupt ------  ------ Interrupt ------
    1     4       3     0   1011 halt
    1     4       4     1   ----- Halt;Switch -----  ----- Halt;Switch -----
    0     4       4     1                           1007 mov  $0, mutex
    0     4       4     0                           1008 sub  $1, %bx
    0     4       4     0                           1009 test $0, %bx
    0     4       4     0                           1010 jgt .top
    0     4       4     0                           1011 halt
```