

lock

\$\*Z#

```
balance = balance + 1;
```

```
1  lock_t mutex; // some globally-allocated lock 'mutex'
2  ...
3  lock(&mutex);
4  balance = balance + 1;
5  unlock(&mutex);
```

mutex

available    unlocked    free  
held

lock()    unlock()

lock()

mutex    lock()

unlock()

lock()

lock variable

acquired    locked

owner

lock()

\$\*Z\$ BfZdM\$V

POSIX

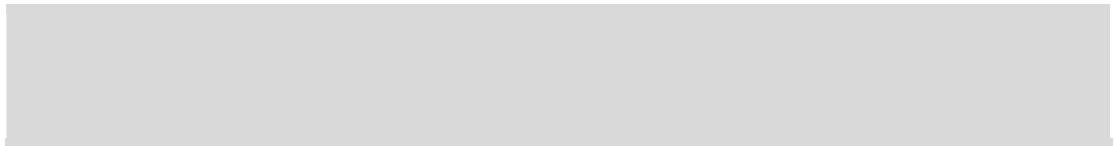
mutex

POSIX

```
1  pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
2
3  Pthread_mutex_lock(&lock);    // wrapper for pthread_mutex_lock()
4  balance = balance + 1;
5  Pthread_mutex_unlock(&lock);
```

POSIX lock unlock

\$\*Z%



\$\*Z&

mutual exclusion

fairness

starve

performance

CPU

CPU

\$\*Z

```

1  void lock() {
2      DisableInterrupts();
3  }
4  void unlock() {
5      EnableInterrupts();
6  }

```

lock()

lock()

CPU

CPU

CPU

			DEKKER	PETERSON	
20	60	Dijkstra			Theodorus Jozef Dekker
					Dekker
					Dekker
algorithm	load	store			
Peterson		Dekker	[P81]	load	store
	Peterson	Peterson	algorithm		
flag	turn				
<pre> int flag[2]; int turn;  void init() {     flag[0] = flag[1] = 0;    // 1-&gt;thread wants to grab lock     turn = 0;                // whose turn? (thread 0 or 1?) } void lock() {     flag[self] = 1;          // self: thread ID of caller     turn = 1 - self;        // make it other thread's turn     while ((flag[1-self] == 1) &amp;&amp; (turn == 1 - self))         ; // spin-wait } void unlock() {     flag[self] = 0;          // simply undo your intent } </pre>					

\$\*ž(

	20	60	Burroughs B5000[M82]	
		CPU		
			test-and-set instruction	atomic
exchange		test-and-set		
	28.1			
	lock()		1	1
		unlock()		1

```

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 }

```

28.1

spin-wait)                      unlock()                      lock()                      while                      while

28.1

flag=0

\$\*Z#

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!)	

1

spin-waiting

\$\*Z)

SPARC

ldstb load/store unsigned byte /

x86 xchg

atomic exchange

test-and-set

C

```

1 int TestAndSet(int *old_ptr, int new) {
2     int old = *old_ptr; // fetch old value at old_ptr
3     *old_ptr = new;     // store 'new' into old_ptr
4     return old;         // return the old value
5 }

```

old\_ptr

new

atomically

spin lock

28.2

flag 0

TestAndSet(flag, 1)

0

lock()

while

flag 1

unlock() flag 0

```

1 typedef struct lock_t {
2     int flag;
3 } lock_t;
4
5 void init(lock_t *lock) {
6     // 0 indicates that lock is available, 1 that it 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 }

```

28.2

flag 1

lock()

TestAndSet(flag, 1)

1

TestAndSet()

1

flag

0

TestAndSet()

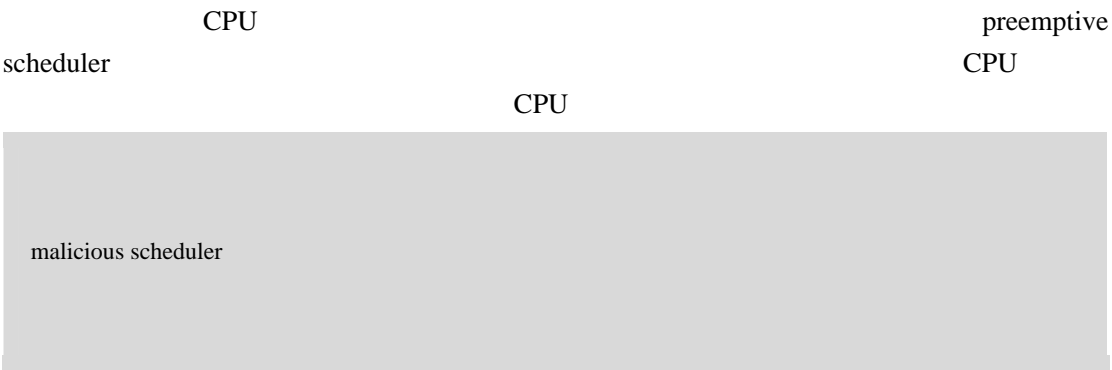
0

1

test

set

spin lock

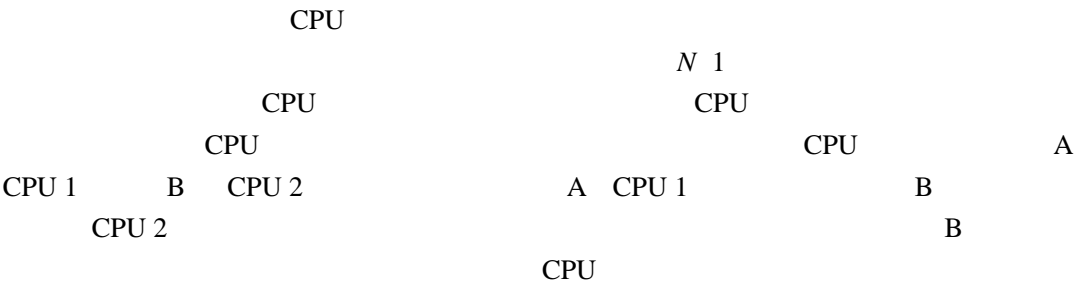


$\$^* \check{Z}^*$

correctness

fairness

performance



$\$^* \check{Z}^+$

x86

compare-and-exchange

28.3

SPARC

compare-and-swap

C

```
1  int CompareAndSwap(int *ptr, int expected, int new) {
2      int actual = *ptr;
3      if (actual == expected)
4          *ptr = new;
5      return actual;
6  }
```

28.3

ptr

expected

ptr

```
lock()

1 void lock(lock_t *lock) {
2     while (CompareAndSwap(&lock->flag, 0, 1) == 1)
3         ; // spin
4 }
```

0

1

C

x86

[S05])

```
1 char CompareAndSwap(int *ptr, int old, int new) {
2     unsigned char ret;
3
4     // Note that sete sets a 'byte' not the word
5     __asm__ __volatile__ (
6         " lock\n"
7         " cmpxchgl %2,%1\n"
8         " sete %0\n"
9         : "=q" (ret), "=m" (*ptr)
10        : "r" (new), "m" (*ptr), "a" (old)
11        : "memory");
12    return ret;
13 }
```

wait-free synchronization [H91]

\$\*Ž#'

load-linked

28.4

C

store-conditional

Alpha

PowerPC

MIPS

ARM

[H93]

[W09]

```
1 int LoadLinked(int *ptr) {
2     return *ptr;
3 }
4
5 int StoreConditional(int *ptr, int value) {
6     if (no one has updated *ptr since the LoadLinked to this address) {
```



```

7         *ptr = value;
8         return 1; // success!
9     } else {
10        return 0; // failed to update
11    }
12 }

```

28.4

store-conditional

1 ptr

value 0

28.5

lock() 0

1

```

1 void lock(lock_t *lock) {
2     while (1) {
3         while (LoadLinked(&lock->flag) == 1)
4             ; // spin until it's zero
5         if (StoreConditional(&lock->flag, 1) == 1)
6             return; // if set-it-to-1 was a success: all done
7                     // otherwise: try it all over again
8     }
9 }
10
11 void unlock(lock_t *lock) {
12     lock->flag = 0;
13 }

```

28.5 LL/SC

Hugh Lauer

[L81]

Lauer &amp; Law

lock()

lock

0

0

1

David Capel

```

1 void lock(lock_t *lock) {
2     while (LoadLinked(&lock->flag) || !StoreConditional(&lock->flag, 1))
3         ; // spin
4 }

```

\$\*Z##

fetch-and-add

C

ticket

Mellor-Crummey

Michael Scott[MS91]

28.6

lock

unlock

```

1 int FetchAndAdd(int *ptr) {
2     int old = *ptr;
3     *ptr = old + 1;
4     return old;
5 }
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     FetchAndAdd(&lock->turn);
19 }

```

28.6 ticket

ticket turn

ticket

turn

myturn

lock-&gt;turn

myturn

== turn

unlock

turn

ticket

 $\$^* \tilde{Z} \# \$$ 

1 0  
 1 0  
 1  $N$   
 $N - 1$

CPU

 $\$^* \tilde{Z} \# \%$ 

ticket

CPU

Al Davis

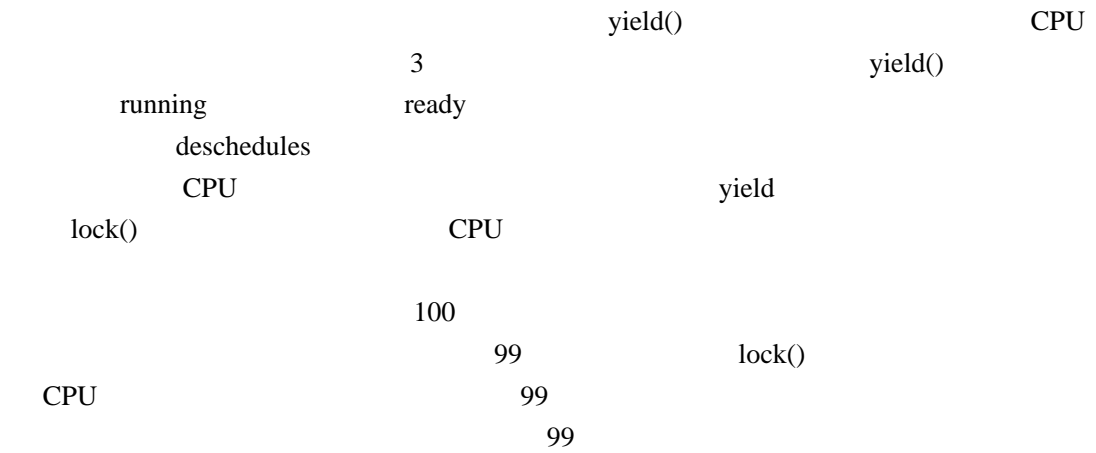
[D91] 28.7

```

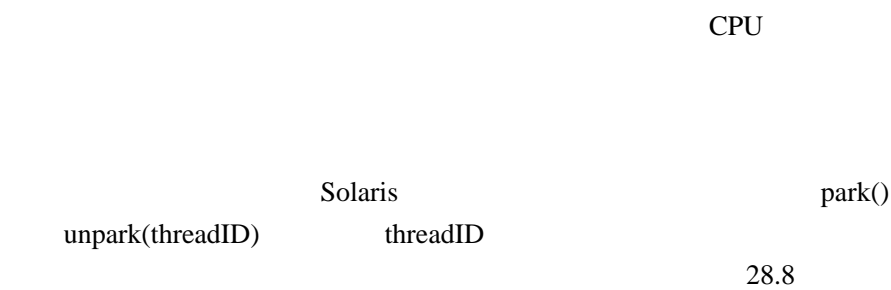
1  void init() {
2      flag = 0;
3  }
4
5  void lock() {
6      while (TestAndSet(&flag, 1) == 1)
7          yield(); // give up the CPU
8  }
9
10 void unlock() {
11     flag = 0;
12 }

```

28.7



\$\*~#&



```

1  typedef struct  lock_t {
2      int flag;
3      int guard;
4      queue_t *q;
5  } lock_t;
6
7  void lock_init(lock_t *m) {
8      m->flag = 0;
9      m->guard = 0;
10     queue_init(m->q);
11 }
12
13 void lock(lock_t *m) {
14     while (TestAndSet(&m->guard, 1) == 1)
15         ; //acquire guard lock by spinning

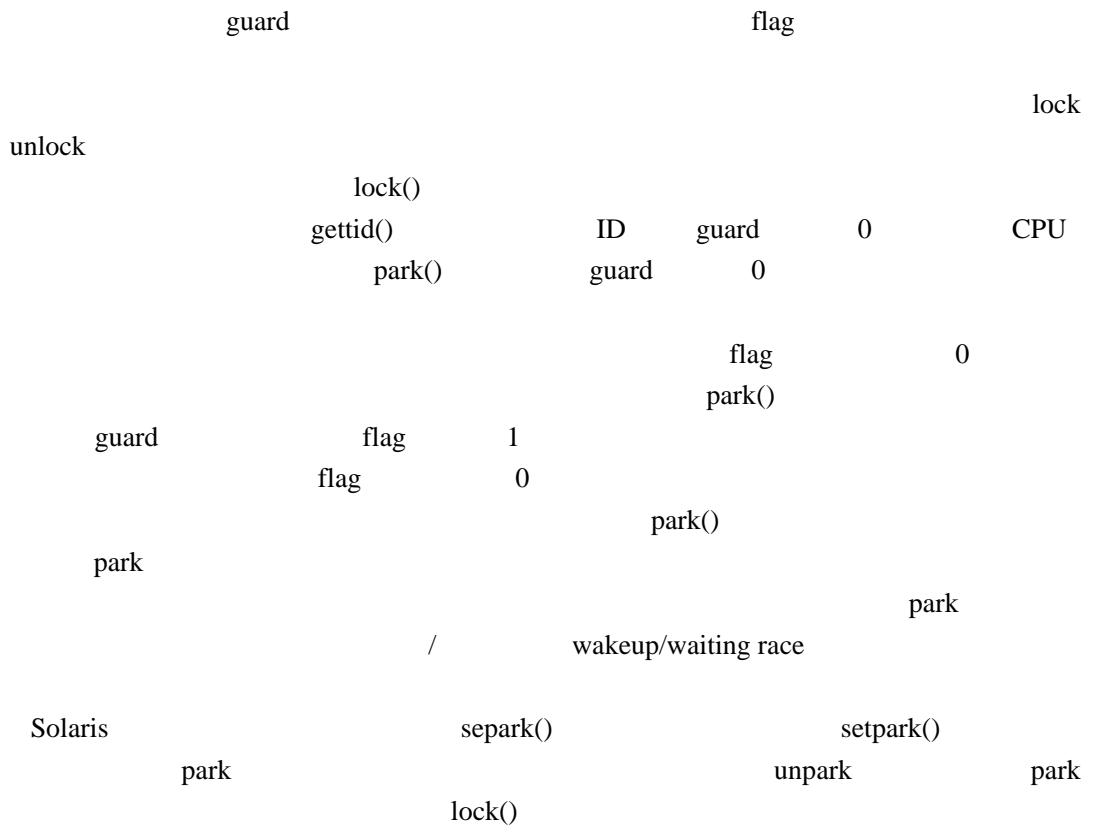
```

```

16     if (m->flag == 0) {
17         m->flag = 1; // lock is acquired
18         m->guard = 0;
19     } else {
20         queue_add(m->q, gettid());
21         m->guard = 0;
22         park();
23     }
24 }
25
26 void unlock(lock_t *m) {
27     while (TestAndSet(&m->guard, 1) == 1)
28         ; //acquire guard lock by spinning
29     if (queue_empty(m->q))
30         m->flag = 0; // let go of lock; no one wants it
31     else
32         unpark(queue_remove(m->q)); // hold lock (for next thread!)
33     m->guard = 0;
34 }

```

28.8



```

1  queue_add(m->q, gettid());
2  setpark(); // new code
3  m->guard = 0;

```

guard

\$\*Z#

	Linux	futex	Solaris
futex			
futex			
		futex_wait(address, expected)	address
expected			futex_wake(address)

## 28.9 Linux

```

1  void mutex_lock (int *mutex) {
2      int v;
3      /* Bit 31 was clear, we got the mutex (this is the fastpath) */
4      if (atomic_bit_test_set (mutex, 31) == 0)
5          return;
6      atomic_increment (mutex);
7      while (1) {
8          if (atomic_bit_test_set (mutex, 31) == 0) {
9              atomic_decrement (mutex);
10             return;
11         }
12         /* We have to wait now. First make sure the futex value
13            we are monitoring is truly negative (i.e. locked). */
14         v = *mutex;
15         if (v >= 0)
16             continue;
17         futex_wait (mutex, v);
18     }
19 }
20
21 void mutex_unlock (int *mutex) {
22     /* Adding 0x80000000 to the counter results in 0 if and only if
23        there are not other interested threads */
24     if (atomic_add_zero (mutex, 0x80000000))
25         return;
26
27     /* There are other threads waiting for this mutex,
28        wake one of them up. */
29     futex_wake (mutex);

```

28.9 Linux futex

nptl      gnu   libc      [L09]      lowlevellock.h

\$\*~#(

Linux      20  
60      Dahm   [M82]      two-phase lock

Linux  
futex

hybrid

\$\*~#)

Solaris      park()      unpark()      Linux      futex  
Solaris      Linux

[L09   S09]   David

[D+13]

[D91]   Just Win, Baby: Al Davis and His Raiders   Glenn Dickey, Harcourt 1991

Al Davis      Just Win

[D+13]   Everything You Always Wanted to Know about Synchronization but Were Afraid to Ask

Tudor David, Rachid Guerraoui, Vasileios Trigonakis

SOSP 13, Nemacon Woodlands Resort, Pennsylvania, November 2013

[D68] Cooperating sequential processes Edsger W. Dijkstra, 1968

Dijkstra

Dekker

[H93] MIPS R4000 Microprocessor User's Manual Joe Heinrich, Prentice-Hall, June 1993

[H91] Wait-free Synchronization Maurice Herlihy

ACM Transactions on Programming Languages and Systems (TOPLAS) Volume 13, Issue 1, January 1991

[L81] Observations on the Development of an Operating System Hugh Lauer

SOSP '81, Pacific Grove, California, December 1981

Pilot

PC

[L09] glibc 2.9 (include Linux pthreads implementation)

nptl

Linux

pthread

[M82] The Architecture of the Burroughs B5000 20 Years Later and Still Ahead of the Times? Alastair J.W. Mayer, 1982

RDLK

RDLK

Dave Dahm

Burroughs

Buzz Locks

Dahm Locks

[MS91] Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors John M. Mellor-Crummey and M. L. Scott

ACM TOCS, Volume 9, Issue 1, February 1991

[P81] Myths About the Mutual Exclusion Problem

G. L. Peterson

Information Processing Letters, 12(3), pages 115-116, 1981

Peterson

[S05] Guide to porting from Solaris to Linux on x86 Ajay Sood, April 29, 2005

[S09] OpenSolaris Thread Library

Oracle

Sun

Mike Swift

[W09] Load-Link, Store-Conditional



ll/sc	MIPS	ldrex/strex	ARM	6
-------	------	-------------	-----	---

[WG00] The SPARC Architecture Manual: Version 9 David L. Weaver and Tom Germond, September 2000  
SPARC International, San Jose, California

Sparc

x86.py

# README

1	-p	flag.s	x86.py
2		flag.s	
		-M	-R
			-c
3	-a	%bx	-a bx = 2    bx =
4		bx	-i
5		test-and-set.s	xchg
6			-i
		CPU	
7	-P		
8		peterson.s	Person
9		-i	
10		-P	
11		ticket.s	ticket

---

12		-a bx=1000, bx=1000	1000
13			
14	yield.s	yield	CPU
		OS	
		test-and-set.s	yield.s
15	test-and-test-and-set.s		test-and-set.s