


TCSS 422: OPERATING SYSTEMS

Linux Thread API, Lock Implementations, Lock-based data structures,



Wes J. Lloyd
School of Engineering and Technology
University of Washington - Tacoma

April 25, 2024 TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington Tacoma

1

OBJECTIVES – 4/25

- **Questions from 4/23**
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 26
- Assignment 1 - Due Tue May 7
- Quiz 1 (Due Thur Apr 25) – Quiz 2 (Due Tue April 30)
- Chapter 27: Linux Thread API
 - pthread_cond_wait/_signal/_broadcast
- Chapter 28: Locks
 - Introduction, Lock Granularity
 - Spin Locks, Test and Set, Compare and Swap
- Chapter 29: Lock Based Data Structures
 - Approximate Counter (Sloppy Counter)
 - Concurrent Structures: Linked List, Queue, Hash Table

April 25, 2024 TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma L10.2

2

ONLINE DAILY FEEDBACK SURVEY

- Daily Feedback Quiz in Canvas – Available After Each Class
- Extra credit available for completing surveys **ON TIME**
- Tuesday surveys: due by ~ Wed @ 11:59p
- Thursday surveys: due ~ Mon @ 11:59p

TCSS 422 A > Assignments

Spring 2021

Home

Announcements

Zoom

Syllabus

Assignments

Discussions

Search for Assignment

Upcoming Assignments

TCSS 422 - Online Daily Feedback Survey - 4/1
Available until Apr 5 at 11:59pm | Due Apr 5 at 10pm | -/1 pts

Quiz 0 - C background survey

April 25, 2024

TCSS422: Computer Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma

L10.3

3

TCSS 422 - Online Daily Feedback Survey - 4/1

Quiz Instructions

Question 1 0.5 pts

On a scale of 1 to 10, please classify your perspective on material covered in today's class:

1	2	3	4	5	6	7	8	9	10
Mostly Review To Me				Equal New and Review					Mostly New to Me

Question 2 0.5 pts

Please rate the pace of today's class:

1	2	3	4	5	6	7	8	9	10
Slow				Just Right					Fast

April 25, 2024

TCSS422: Computer Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma

L10.4

4

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (31 respondents):
 - 1-mostly review, 5-equal new/review, 10-mostly new
 - **Average - 6.45** (↓ - previous **6.58**)

- Please rate the pace of today's class:
 - 1-slow, 5-just right, 10-fast
 - **Average - 5.06** (↓ - previous **5.10**)

April 25, 2024	TCSS422: Computer Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.5
----------------	--	-------

5

FEEDBACK FROM 4/23

- **Why does `pthread_mutex_lock()` function call return an integer which is assigned to variable `rc`?**

```
int rc = pthread_mutex_init(&lock, NULL);  
assert(rc == 0); // always check success!
```

 - We capture the function's return code into int rc

- **Why does `rc` have to be asserted that it is equal to 0 right after the initialization?**
 - The assert function throws an error (and stops the program) when rc is non-zero
 - Receiving a non-zero return indicates a critical error, and the program should likely not continue to run
 - Calling `assert()` is optional and not required

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.6
----------------	---	-------

6

OBJECTIVES – 4/25

- Questions from 4/23
- **C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 26**
- Assignment 1 - Due Tue May 7
- Quiz 1 (Due Thur Apr 25) – Quiz 2 (Due Tue April 30)
- Chapter 27: Linux Thread API
 - pthread_cond_wait/_signal/_broadcast
- Chapter 28: Locks
 - Introduction, Lock Granularity
 - Spin Locks, Test and Set, Compare and Swap
- Chapter 29: Lock Based Data Structures
 - Approximate Counter (Sloppy Counter)
 - Concurrent Structures: Linked List, Queue, Hash Table

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.7
----------------	---	-------

7

OBJECTIVES – 4/25

- Questions from 4/23
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 26
- **Assignment 1 - Due Tue May 7**
- Quiz 1 (Due Thur Apr 25) – Quiz 2 (Due Tue April 30)
- Chapter 27: Linux Thread API
 - pthread_cond_wait/_signal/_broadcast
- Chapter 28: Locks
 - Introduction, Lock Granularity
 - Spin Locks, Test and Set, Compare and Swap
- Chapter 29: Lock Based Data Structures
 - Approximate Counter (Sloppy Counter)
 - Concurrent Structures: Linked List, Queue, Hash Table

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.8
----------------	---	-------

8

OBJECTIVES – 4/25

- Questions from 4/23
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 26
- Assignment 1 - Due Tue May 7
- **Quiz 1 (Due Thur Apr 25) – Quiz 2 (Due Tue April 30)**
- Chapter 27: Linux Thread API
 - pthread_cond_wait/_signal/_broadcast
- Chapter 28: Locks
 - Introduction, Lock Granularity
 - Spin Locks, Test and Set, Compare and Swap
- Chapter 29: Lock Based Data Structures
 - Approximate Counter (Sloppy Counter)
 - Concurrent Structures: Linked List, Queue, Hash Table

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.9
----------------	---	-------

9

QUIZ 1

- Active reading on Chapter 9 – Proportional Share Schedulers
- Posted in Canvas
- Due Thursday April 25th at 11:59pm
- Link:
- https://faculty.washington.edu/wlloyd/courses/tcss422/quiz/TCSS422_s2024_quiz_1.pdf

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.10
----------------	---	--------

10

QUIZ 2

- Canvas Quiz – Practice CPU Scheduling Problems
- Posted in Canvas
- Unlimited attempts permitted
- Provides CPU scheduling practice problems
 - FIFO, SJF, STCF, RR, MLFQ (Ch. 7 & 8)
- Multiple choice and fill-in the blank
- Quiz automatically scored by Canvas
 - Please report any grading problems

- Due Tuesday April 30th at 11:59pm

- Link:
- <https://canvas.uw.edu/courses/1728244/quizzes/2030525>

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.11
----------------	---	--------

11


OBJECTIVES – 4/25

- Questions from 4/23
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 26
- Assignment 1 - Due Tue May 7
- Quiz 1 (Due Thur Apr 25) – Quiz 2 (Due Tue April 30)
- Chapter 27: Linux Thread API
 - pthread_cond_wait/_signal/_broadcast
- Chapter 28: Locks
 - **Introduction**, Lock Granularity
 - Spin Locks, Test and Set, Compare and Swap
- Chapter 29: Lock Based Data Structures
 - Approximate Counter (Sloppy Counter)
 - Concurrent Structures: Linked List, Queue, Hash Table

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.12
----------------	---	--------

12


CHAPTER 28 – LOCKS



April 25, 2024 TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma L10.1
3

13

LOCKS



- Ensure critical section(s) are executed atomically-as a *unit*
 - Only one thread is allowed to execute a critical section at any given time
 - Ensures the code snippets are “mutually exclusive”
- Protect a global counter:

```
balance = balance + 1;
```
- A “critical section”:

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

April 25, 2024 TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma L10.14

14

LOCKS - 2

- Lock variables are called “MUTEX”
 - Short for mutual exclusion (that’s what they guarantee)

- Lock variables store the state of the lock

- States
 - **Locked** (acquired or held)
 - **Unlocked** (available or free)

- Only 1 thread can hold a lock

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.15
----------------	---	--------

15

LOCKS - 3

- `pthread_mutex_lock(&lock)`
 - Try to acquire lock
 - If lock is free, calling thread will acquire the lock
 - Thread with lock enters critical section
 - Thread “owns” the lock

- No other thread can acquire the lock before the owner releases it.

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.16
----------------	---	--------

16

OBJECTIVES – 4/25

- Questions from 4/23
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 26
- Assignment 1 - Due Tue May 7
- Quiz 1 (Due Thur Apr 25) – Quiz 2 (Due Tue April 30)
- Chapter 27: Linux Thread API
 - pthread_cond_wait/_signal/_broadcast
- Chapter 28: Locks
 - Introduction, **Lock Granularity**
 - Spin Locks, Test and Set, Compare and Swap
- Chapter 29: Lock Based Data Structures
 - Approximate Counter (Sloppy Counter)
 - Concurrent Structures: Linked List, Queue, Hash Table

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.17
----------------	---	--------

17

LOCKS - 4

- Program can have many mutex (lock) variables to “serialize” many critical sections
- Locks are also used to protect data structures
 - Prevent multiple threads from changing the same data simultaneously
 - Programmer can make sections of code “granular”
 - ***Fine grained*** – means just one grain of sand at a time through an hour glass
 - Similar to relational database transactions
 - DB transactions prevent multiple users from modifying a table, row, field


April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.18
----------------	---	--------

18

FINE GRAINED?

■ *Is this code a good example of “fine grained parallelism”?*

```
pthread_mutex_lock(&lock);  
a = b++;  
b = a * c;  
*d = a + b +c;  
FILE * fp = fopen ("file.txt", "r");  
fscanf(fp, "%s %s %s %d", str1, str2, str3, &e);  
ListNode *node = mylist->head;  
Int i=0  
while (node) {  
    node->title = str1;  
    node->subheading = str2;  
    node->desc = str3;  
    node->end = *e;  
    node = node->next;  
    i++  
}  
e = e - i;  
pthread_mutex_unlock(&lock);
```




April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.19
----------------	---	--------

19

FINE GRAINED PARALLELISM

```
pthread_mutex_lock(&lock_a);  
pthread_mutex_lock(&lock_b);  
a = b++;  
pthread_mutex_unlock(&lock_b);  
pthread_mutex_unlock(&lock_a);  
  
pthread_mutex_lock(&lock_b);  
b = a * c;  
pthread_mutex_unlock(&lock_b);  
  
pthread_mutex_lock(&lock_d);  
*d = a + b +c;  
pthread_mutex_unlock(&lock_d);  
  
FILE * fp = fopen ("file.txt", "r");  
pthread_mutex_lock(&lock_e);  
fscanf(fp, "%s %s %s %d", str1, str2, str3, &e);  
pthread_mutex_unlock(&lock_e);  
  
ListNode *node = mylist->head;  
int i=0 . . .
```



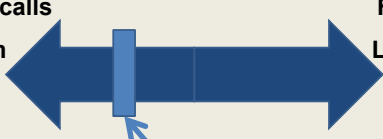
April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.20
----------------	---	--------

20

LOCK GRANULARITY TRADE-OFF SPACE

FINE-GRAINED

Many Lock (kernel) calls
More overhead from excessive locking
More parallelism
Higher code complexity & debugging



COARSE-GRAINED

Few Lock (kernel) calls
Low overhead from minimal locking
Less parallelism
Low code complexity & simpler debugging

Every program implementation lies someplace along the trade-off space...

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L4.21
----------------	---	-------

21


EVALUATING LOCK IMPLEMENTATIONS

- **Correctness**
 - Does the lock work?
 - Are critical sections mutually exclusive? (atomic-as a unit?)

- **Fairness**
 - Do all threads that compete for a lock have a fair chance of acquiring it?

- **Overhead**

What makes a good lock?



April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.22
----------------	---	--------

22

BUILDING LOCKS

- Locks require hardware support
 - To minimize overhead, ensure fairness and correctness
 - Special “atomic-as a *unit*” instructions to support lock implementation
 - Atomic-as a *unit* exchange instruction
 - XCHG
 - Compare and exchange instruction
 - CMPXCHG
 - CMPXCHG8B
 - CMPXCHG16B

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.23
----------------	---	--------

23

HISTORICAL IMPLEMENTATION

- To implement mutual exclusion
 - Disable interrupts upon entering critical sections

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

- Any thread could disable system-wide interrupt
 - What if lock is never released?
- On a multiprocessor processor each CPU has its own interrupts
 - Do we disable interrupts for all cores simultaneously?
- While interrupts are disabled, they could be lost
 - If not queued...

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.24
----------------	---	--------

24

OBJECTIVES – 4/25


- Questions from 4/23
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 26
- Assignment 1 - Due Tue May 7
- Quiz 1 (Due Thur Apr 25) – Quiz 2 (Due Tue April 30)
- Chapter 27: Linux Thread API
 - pthread_cond_wait/_signal/_broadcast
- Chapter 28: Locks
 - Introduction, Lock Granularity
 - **Spin Locks** Test and Set, Compare and Swap
- Chapter 29: Lock Based Data Structures
 - Approximate Counter (Sloppy Counter)
 - Concurrent Structures: Linked List, Queue, Hash Table

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.25
----------------	---	--------

25

BASIC SPIN LOCK IMPLEMENTATION

- Demonstration of lock implementation using C code
- C code is compiled to assembly, instructions are not atomic
- Idea is to imagine “what if” the lock code were atomic



- Is this lock implementation:
(1)Correct?
(2)Fair?
(3)Performant?

```
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 }
```

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.26
----------------	---	--------

26

BASIC SPIN LOCK: CORRECT?

- If both threads can run at the same time, then correctness requires luck... (e.g. *basic spin lock is incorrect*)

Thread1	Thread2
<pre>call lock() while (flag == 1) interrupt: switch to Thread 2</pre>	<pre>call lock() while (flag == 1) flag = 1; interrupt: switch to Thread 1</pre>
<pre>flag = 1; // set flag to 1 (too!)</pre>	

- Here both threads have “acquired” the lock simultaneously

April 25, 2024TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - TacomaL10.27

27

BASIC SPIN LOCK: PERFORMANCE ?

```
void lock(Lock_t *mutex)
{
  while (mutex->flag == 1); // while lock is unavailable, wait...
  mutex->flag = 1;
}
```

- What is wrong with while(<cond>); ?
- Spin-waiting wastes time actively waiting for another thread
- while (1); will “peg” a CPU core at 100%
 - Continuously loops, and evaluates mutex->flag value...
 - If multiple threads wait for the CPU, more CPU capacity is wasted
 - Generates heat...

April 25, 2024TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - TacomaL10.28

28

OBJECTIVES – 4/25

- Questions from 4/23
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 26
- Assignment 1 - Due Tue May 7
- Quiz 1 (Due Thur Apr 25) – Quiz 2 (Due Tue April 30)
- Chapter 27: Linux Thread API
 - pthread_cond_wait/_signal/_broadcast
- Chapter 28: Locks
 - Introduction, Lock Granularity
 - Spin Locks **Test and Set**, Compare and Swap
- Chapter 29: Lock Based Data Structures
 - Approximate Counter (Sloppy Counter)
 - Concurrent Structures: Linked List, Queue, Hash Table

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.29
----------------	---	--------

29

TEST-AND-SET INSTRUCTION

- Hardware support required for working locks
- Book presents pseudo code of C implementation for TEST-AND-SET instruction that needs to be atomic
 - TEST-and-SET checks old value improving on basic spin lock
 - TEST-and-SET returns the old value so it can be checked
 - Comparison is made in the caller
 - Assumption is the TEST-AND-SET routine runs atomically on the CPU
 - Here is the C-pseudo code:

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

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.30
----------------	---	--------

30

TEST-AND-SET - 2

- lock() method checks that TestAndSet doesn't return 1
- If TestAndSet returns 1:
 - This indicates someone else has the lock

```
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,
7      // 1 that it is held
8      lock->flag = 0;
9  }
10
11 void lock(lock_t *lock) {
12     while (TestAndSet(&lock->flag, 1) == 1)
13         ; // spin-wait
14 }
15
16 void unlock(lock_t *lock) {
17     lock->flag = 0;
18 }
```

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.31
----------------	---	--------

31

SPIN LOCK EVALUATION

- **Correctness:**
 - Spin locks with atomic Test-and-Set:
Critical sections won't be executed simultaneously by (2) threads
- **Fairness:**
 - No fairness guarantee. Once a thread has a lock, nothing forces it to relinquish it... lock distribution is random
- **Performance:**
 - Spin locks perform "busy waiting"
 - Spin locks are best for short periods of waiting (< 1 time quantum)
 - Performance is slow when multiple threads share a CPU
 - Especially if "spinning" for long periods

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.32
----------------	---	--------

32

OBJECTIVES – 4/25

- Questions from 4/23
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 26
- Assignment 1 - Due Tue May 7
- Quiz 1 (Due Thur Apr 25) – Quiz 2 (Due Tue April 30)
- Chapter 27: Linux Thread API
 - pthread_cond_wait/_signal/_broadcast
- Chapter 28: Locks
 - Introduction, Lock Granularity
 - Spin Locks, Test and Set, **Compare and Swap**
- Chapter 29: Lock Based Data Structures
 - Approximate Counter (Sloppy Counter)
 - Concurrent Structures: Linked List, Queue, Hash Table

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.33
----------------	---	--------

33

COMPARE AND SWAP

- Checks that the lock variable has the expected value **FIRST**, before changing its value
 - If so, make assignment
 - Return value at location
- Adds a comparison to TestAndSet method
 - Textbook presents C pseudo code
 - Assumption is that the compare-and-swap method runs **atomically**
- Useful for wait-free synchronization
 - Supports implementation of shared data structures which can be updated atomically (*as a unit*) using Hardware support: x86 CompareAndSwap instructions
 - Shared data structure updates become “wait-free”
 - Upcoming in Chapter 32

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.34
----------------	---	--------

34

COMPARE AND SWAP

- Compare and Swap

```
1 int CompareAndSwap(int *ptr, int expected, int new) {
2   int actual = *ptr;
3   if (actual == expected)
4     *ptr = new;
5   return actual;
```
- Spin I **C implementation on 1-core VM:
Count is correct, no deadlock**

```
3   ; // spin
4 }
```
- x86 CPU provides “`cmpxchg1`” compare-and-exchange instructions
 - `cmpxchg8b`
 - `cmpxchg16b`

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.35
----------------	---	--------

35

When implementing locks in a high-level language (e.g. C), what is missing that prevents implementation of CORRECT locks?

- Shared state variable
- Condition variables
- ATOMIC instructions
- Fairness
- None of the above

Start the presentation to see live content. For screen share software, share the entire screen. Get help at poller.com/app

36

“LOCK BUILDING” CPU INSTRUCTIONS ON ARM PROCESSORS

- Cooperative instructions used together to support synchronization on RISC systems
- No support on x86 processors
 - Supported by RISC: Alpha, PowerPC, ARM
- Load-linked (LL)
 - Loads value into register
 - Same as typical load
 - Used as a mechanism to track competition
- Store-conditional (SC)
 - Performs “mutually exclusive” store
 - Allows only one thread to store value

April 25, 2024

TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma

L10.37

37

LL/SC LOCK

```
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 }
```

- LL instruction loads pointer value (ptr)
- SC only stores if the load link pointer has not changed
- Requires HW support
 - C code is psuedo code

April 25, 2024

TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma

L10.38

38

LL/SC LOCK - 2

```
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 }
```

■ Two instruction lock

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.39
----------------	---	--------

39

WE WILL RETURN AT 5:10PM

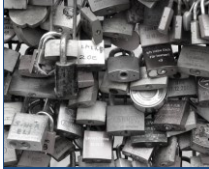


April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.40
----------------	---	--------

40

<h2 style="text-align: center;">OBJECTIVES – 4/25</h2>		
<ul style="list-style-type: none">■ Questions from 4/23■ C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 26■ Assignment 1 - Due Tue May 7■ Quiz 1 (Due Thur Apr 25) – Quiz 2 (Due Tue April 30)■ Chapter 27: Linux Thread API<ul style="list-style-type: none">▪ pthread_cond_wait/_signal/_broadcast■ Chapter 28: Locks<ul style="list-style-type: none">▪ Introduction, Lock Granularity▪ Spin Locks, Test and Set, Compare and Swap■ Chapter 29: Lock Based Data Structures<ul style="list-style-type: none">▪ Approximate Counter (Sloppy Counter)▪ Concurrent Structures: Linked List, Queue, Hash Table		
April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.41

41

<h1>CHAPTER 29 – LOCK BASED DATA STRUCTURES</h1>		
April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.42

42

LOCK-BASED CONCURRENT DATA STRUCTURES

- Adding locks to data structures make them **thread safe**.
- Considerations:
 - Correctness
 - Performance
 - Lock granularity

April 25, 2024

TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma

L10.43

43

COUNTER STRUCTURE W/O LOCK

- Synchronization weary --- not thread safe

```
1     typedef struct __counter_t {
2         int value;
3     } counter_t;
4
5     void init(counter_t *c) {
6         c->value = 0;
7     }
8
9     void increment(counter_t *c) {
10        c->value++;
11    }
12
13    void decrement(counter_t *c) {
14        c->value--;
15    }
16
17    int get(counter_t *c) {
18        return c->value;
19    }
```

April 25, 2024

TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma

L10.44

44

CONCURRENT COUNTER

```
1  typedef struct __counter_t {
2      int value;
3      pthread_lock_t lock;
4  } counter_t;
5
6  void init(counter_t *c) {
7      c->value = 0;
8      Pthread_mutex_init(&c->lock, NULL);
9  }
10
11 void increment(counter_t *c) {
12     Pthread_mutex_lock(&c->lock);
13     c->value++;
14     Pthread_mutex_unlock(&c->lock);
15 }
16
```

- Add lock to the counter
- Require lock to change data

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.45
----------------	---	--------

45

CONCURRENT COUNTER - 2

- Decrease counter
- Get value

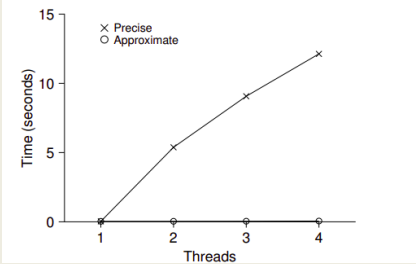
```
(Cont.)
17 void decrement(counter_t *c) {
18     Pthread_mutex_lock(&c->lock);
19     c->value--;
20     Pthread_mutex_unlock(&c->lock);
21 }
22
23 int get(counter_t *c) {
24     Pthread_mutex_lock(&c->lock);
25     int rc = c->value;
26     Pthread_mutex_unlock(&c->lock);
27     return rc;
28 }
```

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.46
----------------	---	--------

46

CONCURRENT COUNTERS - PERFORMANCE

- Concurrent counter is considered a “precise counter”
- iMac: four core Intel 2.7 GHz i5 CPU
- Each thread increments counter 1,000,000 times



Threads	Precise (seconds)	Approximate (seconds)
1	0	0
2	5	0
3	9	0
4	12	0

Precise counter scales poorly.

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.47
----------------	---	--------

47

PERFECT SCALING

- Achieve (N) performance gain with (N) additional resources
- Throughput:
 - Transactions per second (tps)
 - 1 core
 - N = 100 tps
 - 10 cores (x10)
 - N = 1000 tps (x10)
- ***Is parallel counting with a shared counter an embarrassingly parallel problem?***

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.48
----------------	---	--------

48

OBJECTIVES – 4/25

- Questions from 4/23
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 26
- Assignment 1 - Due Tue May 7
- Quiz 1 (Due Thur Apr 25) – Quiz 2 (Due Tue April 30)
- Chapter 27: Linux Thread API
 - pthread_cond_wait/_signal/_broadcast
- Chapter 28: Locks
 - Introduction, Lock Granularity
 - Spin Locks, Test and Set, Compare and Swap
- Chapter 29: Lock Based Data Structures
 - **Approximate Counter (Sloppy Counter)**
 - Concurrent Structures: Linked List, Queue, Hash Table

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.49
----------------	---	--------

49

APPROXIMATE (SLOPPY) COUNTER

- Provides single logical shared counter
 - Implemented using local counters for each ~CPU core
 - 4 CPU cores = 4 local counters & 1 global counter
 - Local counters are synchronized via local locks
 - Global counter is updated periodically
 - Global counter has lock to protect global counter value
 - Update threshold (S) – *referred to as sloppiness threshold*:
How often to push local values to global counter
 - Small (S): more updates, more overhead
 - Large (S): fewer updates, more performant, less synchronized
- Why this implementation?
Why do we want counters local to each CPU Core?

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.50
----------------	---	--------

50

APPROXIMATE COUNTER – MAIN POINTS

- Idea of the Approximate Counter is to **RELAX** the synchronization requirement for counting
 - Instead of synchronizing global count variable each time:
`counter=counter+1`
 - Synchronization occurs only every so often:
 e.g. *every 1000 counts*
- Relaxing the synchronization requirement **drastically** reduces locking API overhead by trading-off split-second accuracy of the counter
- Approximate counter: trade-off accuracy for speed
 - It's approximate because it's not so accurate (until the end)

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.51
----------------	---	--------

51

APPROXIMATE COUNTER - 2

- Update threshold (S) = 5
- Synchronized across four CPU cores
- Threads update local CPU counters

Time	L ₁	L ₂	L ₃	L ₄	G
0	0	0	0	0	0
1	0	0	1	1	0
2	1	0	2	1	0
3	2	0	3	1	0
4	3	0	3	2	0
5	4	1	3	3	0
6	5 → 0	1	3	4	5 (from L ₁)
7	0	2	4	5 → 0	10 (from L ₄)

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.52
----------------	---	--------

52

THRESHOLD VALUE S

- Consider 4 threads increment a counter 1000000 times each
- Low S → What is the consequence?
- High S → What is the consequence?

Approximation Factor (S)	Time (seconds)
1	11.5
2	5.5
4	3.0
8	1.8
16	1.1
32	0.7
64	0.45
128	0.3
256	0.2
512	0.15
1024	0.1

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.53
----------------	---	--------

53

APPROXIMATE COUNTER - EXAMPLE

- Example implementation – `sloppybasic.c`
- Also with CPU affinity

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.54
----------------	---	--------

54

When poll is active, respond at pollev.com/wesleylloyd641
Text **WESLEYLLOYD641** to **22333** once to join

W Which of the following is NOT a problem as a result of having a low S-value for the approximate counter (Sloppy Counter) threshold?

- The counter overhead is very high.
- The counter implementation performs a very large number of LOCK/UNLOCK API calls.
- The global counter value is highly accurate.
- The counter performs very few local to global counter updates.
- None of the above

Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

55

OBJECTIVES – 4/25

- Questions from 4/23
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 26
- Assignment 1 - Due Tue May 7
- Quiz 1 (Due Thur Apr 25) – Quiz 2 (Due Tue April 30)
- Chapter 27: Linux Thread API
 - pthread_cond_wait/_signal/_broadcast
- Chapter 28: Locks
 - Introduction, Lock Granularity
 - Spin Locks, Test and Set, Compare and Swap
- Chapter 29: Lock Based Data Structures
 - Sloppy Counter
 - **Concurrent Structures: Linked List**, Queue, Hash Table

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.56
----------------	---	--------

56

CONCURRENT LINKED LIST - 1

- Simplification - only basic list operations shown
- Structs and initialization:

```
1 // basic node structure
2 typedef struct __node_t {
3     int key;
4     struct __node_t *next;
5 } node_t;
6
7 // basic list structure (one used per list)
8 typedef struct __list_t {
9     node_t *head;
10    pthread_mutex_t lock;
11 } list_t;
12
13 void List_Init(list_t *L) {
14     L->head = NULL;
15     pthread_mutex_init(&L->lock, NULL);
16 }
17
18 (Cont.)
```

April 25, 2024

TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma

L10.57

57

CONCURRENT LINKED LIST - 2

- Insert - adds item to list
- Everything is critical!
 - There are two unlocks

```
(Cont.)
18 int List_Insert(list_t *L, int key) {
19     pthread_mutex_lock(&L->lock);
20     node_t *new = malloc(sizeof(node_t));
21     if (new == NULL) {
22         perror("malloc");
23         pthread_mutex_unlock(&L->lock);
24         return -1; // fail
25     }
26     new->key = key;
27     new->next = L->head;
28     L->head = new;
29     pthread_mutex_unlock(&L->lock);
30     return 0; // success
31 }
32
33 (Cont.)
```

April 25, 2024

TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma

L10.58

58

CONCURRENT LINKED LIST - 3

- Lookup – checks list for existence of item with key
- Once again everything is critical
 - Note - there are also two unlocks

```
(Cont.)
32
33  int List_Lookup(list_t *L, int key) {
34      pthread_mutex_lock(&L->lock);
35      node_t *curr = L->head;
36      while (curr) {
37          if (curr->key == key) {
38              pthread_mutex_unlock(&L->lock);
39              return 0; // success
40          }
41          curr = curr->next;
42      }
43      pthread_mutex_unlock(&L->lock);
44      return -1; // failure
45  }
```

April 25, 2024

TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma

L10.59

59

CONCURRENT LINKED LIST

- First Implementation:
 - Lock **everything** inside Insert() and Lookup()
 - If malloc() fails lock must be released
 - Research has shown “**exception-based control flow**” to be error prone
 - 40% of Linux OS bugs occur in rarely taken code paths
 - Unlocking in an exception handler is considered a poor coding practice
 - There is nothing specifically wrong with this example however
- Second Implementation ...

April 25, 2024

TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma

L10.60

60

CCL – SECOND IMPLEMENTATION

▪ Init and Insert

```
1 void List_Init(list_t *L) {
2     L->head = NULL;
3     pthread_mutex_init(&L->lock, NULL);
4 }
5
6 void List_Insert(list_t *L, int key) {
7     // synchronization not needed
8     node_t *new = malloc(sizeof(node_t));
9     if (new == NULL) {
10        perror("malloc");
11        return;
12    }
13    new->key = key;
14
15    // just lock critical section
16    pthread_mutex_lock(&L->lock);
17    new->next = L->head;
18    L->head = new;
19    pthread_mutex_unlock(&L->lock);
20 }
21
```

April 25, 2024

TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma

L10.61

61

CCL – SECOND IMPLEMENTATION - 2

▪ Lookup

```
(Cont.)
22 int List_Lookup(list_t *L, int key) {
23     int rv = -1;
24     pthread_mutex_lock(&L->lock);
25     node_t *curr = L->head;
26     while (curr) {
27         if (curr->key == key) {
28             rv = 0;
29             break;
30         }
31         curr = curr->next;
32     }
33     pthread_mutex_unlock(&L->lock);
34     return rv; // now both success and failure
35 }
```

April 25, 2024


TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma

L10.62

62

CONCURRENT LINKED LIST PERFORMANCE

- Using a single lock for entire list is not very performant
- Users must “wait” in line for a single lock to access/modify any item
- Hand-over-hand-locking (lock coupling)
 - Introduce a lock for each node of a list
 - Traversal involves handing over previous node’s lock, acquiring the next node’s lock...
 - Improves lock granularity
 - Degrades traversal performance
- Consider hybrid approach
 - Fewer locks, but more than 1
 - Best lock-to-node distribution?



April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.63
----------------	---	--------

63

OBJECTIVES – 4/25

- Questions from 4/23
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 26
- Assignment 1 - Due Tue May 7
- Quiz 1 (Due Thur Apr 25) – Quiz 2 (Due Tue April 30)
- Chapter 27: Linux Thread API
 - pthread_cond_wait/_signal/_broadcast
- Chapter 28: Locks
 - Introduction, Lock Granularity
 - Spin Locks, Test and Set, Compare and Swap
- Chapter 29: Lock Based Data Structures
 - Sloppy Counter
 - Concurrent Structures: Linked List, Queue Hash Table

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.64
----------------	---	--------

64

MICHAEL AND SCOTT CONCURRENT QUEUES

- Improvement beyond a single master lock for a queue (FIFO)
- Two locks:
 - One for the **head** of the queue
 - One for the **tail**
- Synchronize enqueue and dequeue operations

- Add a dummy node
 - Allocated in the queue initialization routine
 - Supports separation of head and tail operations

- Items can be added and removed by separate threads at the same time

April 25, 2024

TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma

L10.65

65

CONCURRENT QUEUE

- Remove from queue

```
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 headLock;
10     pthread_mutex_t tailLock;
11 } queue_t;
12
13 void Queue_Init(queue_t *q) {
14     node_t *tmp = malloc(sizeof(node_t));
15     tmp->next = NULL;
16     q->head = q->tail = tmp;
17     pthread_mutex_init(&q->headLock, NULL);
18     pthread_mutex_init(&q->tailLock, NULL);
19 }
20
(Cont.)
```

April 25, 2024

TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma

L10.66

66

CONCURRENT QUEUE - 2

- Add to queue

```
(Cont.)
21 void Queue_Enqueue(queue_t *q, int value) {
22     node_t *tmp = malloc(sizeof(node_t));
23     assert(tmp != NULL);
24
25     tmp->value = value;
26     tmp->next = NULL;
27
28     pthread_mutex_lock(&q->tailLock);
29     q->tail->next = tmp;
30     q->tail = tmp;
31     pthread_mutex_unlock(&q->tailLock);
32 }
(Cont.)
```

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.67
----------------	---	--------

67

OBJECTIVES – 4/25

- Questions from 4/23
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 26
- Assignment 1 - Due Tue May 7
- Quiz 1 (Due Thur Apr 25) – Quiz 2 (Due Tue April 30)
- Chapter 27: Linux Thread API
 - pthread_cond_wait/_signal/_broadcast
- Chapter 28: Locks
 - Introduction, Lock Granularity
 - Spin Locks, Test and Set, Compare and Swap
- Chapter 29: Lock Based Data Structures
 - Sloppy Counter
 - Concurrent Structures: Linked List, Queue, **Hash Table**

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.68
----------------	---	--------

68

CONCURRENT HASH TABLE

- Consider a simple hash table
 - Fixed (static) size
 - Hash maps to a bucket
 - Bucket is implemented using a concurrent linked list
 - One lock per hash (bucket)
 - Hash bucket is a linked lists

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.69
----------------	---	--------

69

INSERT PERFORMANCE – CONCURRENT HASH TABLE

- Four threads – 10,000 to 50,000 inserts
 - iMac with four-core Intel 2.7 GHz CPU

Inserts (Thousands)	Simple Concurrent List (seconds)	Concurrent Hash Table (seconds)
10	~1.0	~0.5
20	~2.5	~0.5
30	~4.5	~0.5
40	~7.5	~0.5
50	~11.0	~0.5

The simple concurrent hash table scales magnificently.

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.70
----------------	---	--------

70

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
14     int Hash_Insert(hash_t *H, int key) {
15         int bucket = key % BUCKETS;
16         return List_Insert(&H->lists[bucket], key);
17     }
18
19     int Hash_Lookup(hash_t *H, int key) {
20         int bucket = key % BUCKETS;
21         return List_Lookup(&H->lists[bucket], key);
22     }
```

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.71
----------------	---	--------

71

Which is a major advantage of using concurrent data structures in your programs?

- Locks are encapsulated within data structure code ensuring thread safety.
- Lock granularity tradeoff already optimized inside data structurew
- Multiple threads can more easily share data
- All of the above
- None of the above

Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

72


LOCK-FREE DATA STRUCTURES

- Lock-free data structures in Java
- `java.util.concurrent.atomic` package
- Classes:
 - `AtomicBoolean`
 - `AtomicInteger`
 - `AtomicIntegerArray`
 - `AtomicIntegerFieldUpdater`
 - `AtomicLong`
 - `AtomicLongArray`
 - `AtomicLongFieldUpdater`
 - `AtomicReference`
- See: <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/concurrent/atomic/package-summary.html>

April 25, 2024	TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma	L10.73
----------------	---	--------

73

QUESTIONS



74