


# TCSS 422: OPERATING SYSTEMS

## Lock-based data structures, Midterm review



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

November 2, 2021      TCSS422: Operating Systems [Fall 2021]  
School of Engineering and Technology, University of Washington      Tacoma

1

## OBJECTIVES – 11/2

- **Questions from 10/28**
- Assignment 0 Update
- Assignment 1 – Nov 12
- Quiz 1 (Due Tue Nov 2) – Quiz 2 (Due Thur Nov 4)
- Chapter 28: Locks: RISC atomic lock instructions
- Chapter 29: Lock Based Data Structures
  - Approximate Counter (Sloppy Counter)
  - Concurrent Structures: Linked List, Queue, Hash Table
- 2<sup>nd</sup> hour: Midterm Review
  - Practice Questions

November 2, 2021      TCSS422: Operating Systems [Fall 2021]  
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

November 2, 2021
TCSS422: Computer Operating Systems [Fall 2021]  
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			

November 2, 2021
TCSS422: Computer Operating Systems [Fall 2021]  
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 (28 respondents):
  - 1-mostly review, 5-equal new/review, 10-mostly new
  - **Average - 6.50** (↓ - previous 6.66)
  
- Please rate the pace of today's class:
  - 1-slow, 5-just right, 10-fast
  - **Average - 5.48** (no change - previous 5.48)

November 2, 2021	TCSS422: Computer Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.5
------------------	--	-------

5

## FEEDBACK

- **Sample problem #4 from TCSS 422 CPU Scheduler Examples**
  
- Issue with timing graph found
- New solution posted..

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.6
------------------	---	-------

6

Jackson deploys a 3-level MLFQ scheduler. The time slice is 1 for high priority jobs, 2 for medium priority, and 4 for low priority. This MLFQ scheduler performs a Priority Boost every 6 timer units. When the priority boost fires, the current job is preempted, and the next scheduled job is run in round-robin order.

Job	Arrival Time	Job Length
A	T=0	4 <del>3</del> 0
B	T=0	16 <del>15</del> <del>14</del> 5
C	T=0	8 <del>7</del> 1

(11 points) Show a scheduling graph for the MLFQ scheduler for the jobs above. Draw vertical lines for key events and be sure to label the X-axis times as in the example. Please draw clearly. An unreadable graph will loose points.

7

## OBJECTIVES - 11/2

- Questions from 10/28
- **Assignment 0 Update**
- Assignment 1 - Nov 12
- Quiz 1 (Due Tue Nov 2) - Quiz 2 (Due Thur Nov 4)
- Chapter 28: Locks: RISC atomic lock instructions
- Chapter 29: Lock Based Data Structures
  - Approximate Counter (Sloppy Counter)
  - Concurrent Structures: Linked List, Queue, Hash Table
- 2<sup>nd</sup> hour: Midterm Review
  - Practice Questions

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.8
------------------	---	-------

8

## OBJECTIVES – 11/2

- Questions from 10/28
- Assignment 0 Update
- **Assignment 1 – Nov 12**
- Quiz 1 (Due Tue Nov 2) – Quiz 2 (Due Thur Nov 4)
- Chapter 28: Locks: RISC atomic lock instructions
- Chapter 29: Lock Based Data Structures
  - Approximate Counter (Sloppy Counter)
  - Concurrent Structures: Linked List, Queue, Hash Table
- 2<sup>nd</sup> hour: Midterm Review
  - Practice Questions

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.9
------------------	---	-------

9

## OBJECTIVES – 11/2

- Questions from 10/28
- Assignment 0 Update
- Assignment 1 – Nov 12
- **Quiz 1 (Due Tue Nov 2) – Quiz 2 (Due Thur Nov 4)**
- Chapter 28: Locks: RISC atomic lock instructions
- Chapter 29: Lock Based Data Structures
  - Approximate Counter (Sloppy Counter)
  - Concurrent Structures: Linked List, Queue, Hash Table
- 2<sup>nd</sup> hour: Midterm Review
  - Practice Questions

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.10
------------------	---	--------

10

## QUIZ 1

- Active reading on Chapter 9 – Proportional Share Schedulers
- Posted in Canvas
- Due Tuesday Nov 2<sup>nd</sup> at 11:59pm
- Grace period til Thursday Nov 4<sup>th</sup> at 11:59 \*\* AM \*\*
- Late submissions til Saturday Nov 6<sup>th</sup> at 11:59pm
- Link:
- [http://faculty.washington.edu/wlloyd/courses/tcss422/TCSS422\\_s2021\\_quiz\\_1.pdf](http://faculty.washington.edu/wlloyd/courses/tcss422/TCSS422_s2021_quiz_1.pdf)

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.11
------------------	---	--------

11

## QUIZ 2 - CPU SCHEDULING ALGORITHMS

- Quiz posted on Canvas
- Due Thursday Nov 4 @ 11:59p
- Provides CPU scheduling practice problems
  - FIFO, SJF, STCF, RR, MLFQ (Ch. 7 & 8)
- Unlimited attempts allowed
- Multiple choice and fill-in the blank
- Quiz automatically scored by Canvas
  - Please report any grading problems

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.12
------------------	---	--------

12

## OBJECTIVES – 11/2

- Questions from 10/28
- Assignment 0 Update
- Assignment 1 – Nov 12
- Quiz 1 (Due Tue Nov 2) – Quiz 2 (Due Thur Nov 4)
- **Chapter 28: Locks: RISC atomic lock instructions**
- Chapter 29: Lock Based Data Structures
  - Approximate Counter (Sloppy Counter)
  - Concurrent Structures: Linked List, Queue, Hash Table
- 2<sup>nd</sup> hour: Midterm Review
  - Practice Questions

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.13
------------------	---	--------

13

### 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 [polllev.com/app](https://polllev.com/app)

14

## TWO MORE “LOCK BUILDING” CPU INSTRUCTIONS

- Cooperative atomic instructions used together to support synchronization on RISC systems
- Instructions provided as opposed to: XCHG, CMPXCHG(8B,16B)
- 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

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.15
------------------	---	--------

15

## 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

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.16
------------------	---	--------

16



## 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

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.17
------------------	---	--------

17

## OBJECTIVES - 11/2

- Questions from 10/28
- Assignment 0 Update
- Assignment 1 - Nov 12
- Quiz 1 (Due Tue Nov 2) - Quiz 2 (Due Thur Nov 4)
- Chapter 28: Locks: RISC atomic lock instructions
- **Chapter 29: Lock Based Data Structures**
  - Approximate Counter (Sloppy Counter)
  - Concurrent Structures: Linked List, Queue, Hash Table
- 2<sup>nd</sup> hour: Midterm Review
  - Practice Questions

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.18
------------------	---	--------

18

# CHAPTER 29 – LOCK BASED DATA STRUCTURES

November 2, 2021

TCSS422: Operating Systems [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L10.19

19

## LOCK-BASED CONCURRENT DATA STRUCTURES

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

November 2, 2021

TCSS422: Operating Systems [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L10.20

20

## 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 }
```

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.21
------------------	---	--------

21

## 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

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.22
------------------	---	--------

22

## 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 }
```

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.23
------------------	---	--------

23

## 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.**

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.24
------------------	---	--------

24

## 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?***

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.25
------------------	---	--------

25

## OBJECTIVES – 11/2

- Questions from 10/28
- Assignment 0 Update
- Assignment 1 – Nov 12
- Quiz 1 (Due Tue Nov 2) – Quiz 2 (Due Thur Nov 4)
- Chapter 28: Locks: RISC atomic lock instructions
- Chapter 29: Lock Based Data Structures
  - **Approximate Counter (Sloppy Counter)**
    - Concurrent Structures: Linked List, Queue, Hash Table
- 2<sup>nd</sup> hour: Midterm Review
  - Practice Questions

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.26
------------------	---	--------

26

## 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?

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.27
------------------	---	--------

27

## 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)

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.28
------------------	---	--------

28

## APPROXIMATE COUNTER - 2

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

Time	L <sub>1</sub>	L <sub>2</sub>	L <sub>3</sub>	L <sub>4</sub>	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 <sub>1</sub> )
7	0	2	4	5 → 0	10 (from L <sub>4</sub> )

November 2, 2021
TCSS422: Operating Systems [Fall 2021]  
 School of Engineering and Technology, University of Washington - Tacoma
L10.29

29

## 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	12
2	6
4	3
8	1.5
16	0.75
32	0.375
64	0.1875
128	0.09375
256	0.046875
512	0.0234375
1024	0.01171875

November 2, 2021
TCSS422: Operating Systems [Fall 2021]  
 School of Engineering and Technology, University of Washington - Tacoma
L10.30

30

# APPROXIMATE COUNTER - EXAMPLE

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

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.31
------------------	---	--------

31

🌐 When poll is active, respond at [pollev.com/wesleylloyd641](https://pollev.com/wesleylloyd641)  
📧 Text **WESLEYLLOYD641** to **22333** once to join

## 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](https://pollev.com/app)

32



## OBJECTIVES - 11/2

- Questions from 10/28
- Assignment 0 Update
- Assignment 1 - Nov 12
- Quiz 1 (Due Tue Nov 2) - Quiz 2 (Due Thur Nov 4)
- Chapter 28: Locks: RISC atomic lock instructions
- Chapter 29: Lock Based Data Structures
  - Sloppy Counter
  - **Concurrent Structures: Linked List,** Queue, Hash Table
- 2<sup>nd</sup> hour: Midterm Review
  - Practice Questions

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.33
------------------	---	--------

33

## 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.)
```

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.34
------------------	---	--------

34

## 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  }
(Cont.)
```

November 2, 2021

TCSS422: Operating Systems [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L10.35

35

## 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  }
```

November 2, 2021

TCSS422: Operating Systems [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L10.36

36

## 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 ...**

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.37
------------------	---	--------

37

## 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
```

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.38
------------------	---	--------

38

## 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  }
```

November 2, 2021

TCSS422: Operating Systems [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L10.39

39

## 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?



November 2, 2021

TCSS422: Operating Systems [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L10.40

40

## OBJECTIVES – 11/2

- Questions from 10/28
- Assignment 0 Update
- Assignment 1 – Nov 12
- Quiz 1 (Due Tue Nov 2) – Quiz 2 (Due Thur Nov 4)
- Chapter 28: Locks: RISC atomic lock instructions
- Chapter 29: Lock Based Data Structures
  - Sloppy Counter
  - Concurrent Structures: Linked List, **Queue** Hash Table
- 2<sup>nd</sup> hour: Midterm Review
  - Practice Questions

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.41
------------------	---	--------

41

## 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

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.42
------------------	---	--------

42

# 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.)
```

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.43
------------------	---	--------

43

# 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.)
```

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.44
------------------	---	--------

44

## OBJECTIVES – 11/2

- Questions from 10/28
- Assignment 0 Update
- Assignment 1 – Nov 12
- Quiz 1 (Due Tue Nov 2) – Quiz 2 (Due Thur Nov 4)
- Chapter 28: Locks: RISC atomic lock instructions
- Chapter 29: Lock Based Data Structures
  - Sloppy Counter
  - Concurrent Structures: Linked List, Queue, **Hash Table**
- 2<sup>nd</sup> hour: Midterm Review
  - Practice Questions

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.45
------------------	---	--------

45

## 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

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.46
------------------	---	--------

46

## 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	~0.1
20	~2.5	~0.1
30	~4.5	~0.1
40	~7.5	~0.1
50	~11.5	~0.1

**The simple concurrent hash table scales magnificently.**

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.47
------------------	---	--------

47

## 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     }
```

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.48
------------------	---	--------

48



## 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 [polllev.com/app](https://polllev.com/app)

49

## 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>

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.50
------------------	---	--------

50

**WE WILL RETURN AT  
2:40PM**



November 2, 2021 TCSS422: Operating Systems [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma L10.5  
1

51

**OBJECTIVES – 11/2**

- Questions from 10/28
- Assignment 0 Update
- Assignment 1 – Nov 12
- Quiz 1 (Due Tue Nov 2) – Quiz 2 (Due Thur Nov 4)
- Chapter 28: Locks: RISC atomic lock instructions
- Chapter 29: Lock Based Data Structures
  - Sloppy Counter
  - Concurrent Structures: Linked List, **Queue**, Hash Table
- **2<sup>nd</sup> hour: Midterm Review**
  - Practice Questions

November 2, 2021 TCSS422: Operating Systems [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma L10.52

52



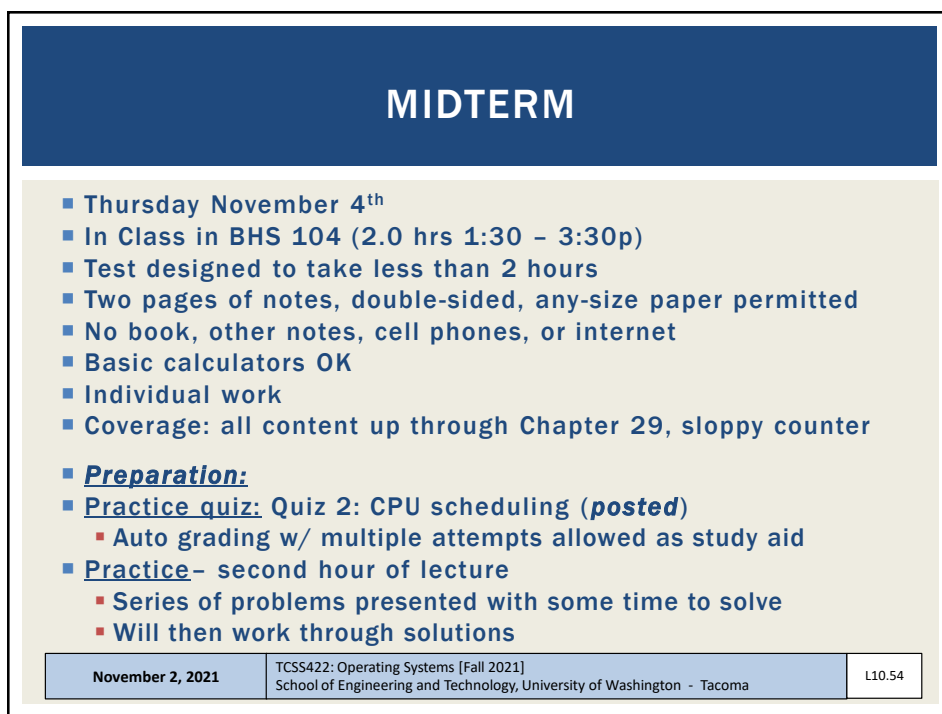
A slide with a blue background. In the top right corner, there is a graphic of glowing question marks. The text 'MIDTERM REVIEW' is centered in large white letters. At the bottom, there is a footer with course information and a slide number.

November 2, 2021

TCSS422: Operating Systems [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L10.5  
3

53



A slide with a dark blue header containing the word 'MIDTERM'. Below the header is a list of details for the exam, including date, time, location, and preparation instructions. A footer at the bottom contains course information and a slide number.

## MIDTERM

- Thursday November 4<sup>th</sup>
- In Class in BHS 104 (2.0 hrs 1:30 - 3:30p)
- Test designed to take less than 2 hours
- Two pages of notes, double-sided, any-size paper permitted
- No book, other notes, cell phones, or internet
- Basic calculators OK
- Individual work
- Coverage: all content up through Chapter 29, sloppy counter
- **Preparation:**
- **Practice quiz:** Quiz 2: CPU scheduling (*posted*)
  - Auto grading w/ multiple attempts allowed as study aid
- **Practice-** second hour of lecture
  - Series of problems presented with some time to solve
  - Will then work through solutions

November 2, 2021

TCSS422: Operating Systems [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L10.54

54

## FIFO EXAMPLE

- Operation of CPU schedulers can be visualized with timing graphs.
- The graph below depicts a FIFO scheduler where three jobs arrive in the sequence A, B, C, where job A runs for 10 time slices, job B for 5 time slices, and job C for 10 time slices.

**FIFO** | AAAAAAAAAABBBBBBBBBBBBBBBBBB

0                      10      15                      25

November 2, 2021
TCSS422: Operating Systems [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L10.55

55

## Q1- SHORTEST JOB FIRST (SJF) SCHEDULER

- Draw a scheduling graph for the SJF scheduler without preemption for the following jobs. Draw vertical lines for key events and be sure to label the X-axis times as in the example.

Job	Arrival Time	Job Length
A	T=0	25
B	T=5	10
C	T=10	15

**SJF**

0

November 2, 2021
TCSS422: Operating Systems [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L10.56

56

Q1 – SJF - 2

What is the response time (RT) and turnaround time (TT) for jobs A, B, and C?

RT Job A: \_\_\_\_\_ TT Job A: \_\_\_\_\_

RT Job B: \_\_\_\_\_ TT Job B: \_\_\_\_\_

RT Job C: \_\_\_\_\_ TT Job C: \_\_\_\_\_

What is the average response time for all jobs? \_\_\_\_\_

What is the average turnaround time for all jobs? \_\_\_\_\_

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.57
------------------	---	--------

57

Q2 – SHORTEST TIME TO COMPLETION FIRST (STCF) SCHEDULER

Draw a scheduling graph for the STCF scheduler with preemption for the following jobs.  
 Draw vertical lines for key events and be sure to label the X-axis times as in the example.

Job	Arrival Time	Job Length
A	T=0	25
B	T=5	10
C	T=10	15

CPU

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.58
------------------	---	--------

58

## Q2 – STCF - 2

- What is the response time (RT) and turnaround time (TT) for jobs A, B, and C?  
  
RT Job A: \_\_\_\_\_                      TT Job A: \_\_\_\_\_  
  
RT Job B: \_\_\_\_\_                      TT Job B: \_\_\_\_\_  
  
RT Job C: \_\_\_\_\_                      TT Job C: \_\_\_\_\_
  
- What is the average response time for all jobs? \_\_\_\_\_
  
- What is the average turnaround time for all jobs? \_\_\_\_\_

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.59
------------------	---	--------

59

## Q3 - OPERATING SYSTEM APIs

1. Provide a definition for what is a blocking API call
  
2. Provide a definition for a non-blocking API call
  
3. Provide an example of a blocking API call.  
Consider APIs used to manage processes and/or threads.
  
4. Provide an example of a non-blocking API call.  
Consider APIs used to manage processes and/or threads.

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.60
------------------	---	--------

60

## Q4 – OPERATING SYSTEM APIs - II

1. When implementing memory synchronization for a multi-threaded program list one **advantage** of combining the use of a condition variable with a lock variable via the Linux C thread API calls: `pthread_mutex_lock()` and `pthread_cond_wait()`

2. When implementing memory synchronization for a multi-threaded program using locks, list one **disadvantage** of using blocking thread API calls such as the Linux C thread API calls for: `pthread_mutex_lock()` and `pthread_cond_wait()`

3. List (2) factors that cause Linux blocking API calls to introduce **overhead** into programs:

November 2, 2021

TCSS422: Operating Systems [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L10.61

61

## Q5 – PERFECT MULTITASKING OPERATING SYSTEM

In a perfect-multi-tasking operating system, every process of the same priority will always receive exactly  $1/n^{\text{th}}$  of the available CPU time. Important CPU improvements for multi-tasking include: (1) fast context switching to enable jobs to be swapped in-and-out of the CPU very quickly, and (2) the use of a timer interrupt to preempt running jobs without the user voluntarily yielding the CPU. These innovations have enabled major improvements towards achieving a coveted "Perfect Multi-Tasking System".

List and describe two challenges that remain complicating the full realization of a Perfect Multi-Tasking Operating System. In other words, what makes it very difficult for all jobs (for example, 10 jobs) of the same priority to receive **EXACTLY** the same runtime on the CPU? Your description must explain why the challenge is a problem for achieving perfect multi-tasking.

November 2, 2021

TCSS422: Operating Systems [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L10.62

62

## Q6 – ROUND-ROBIN SCHEDULER

Show a scheduling graph for a Round-Robin (RR) scheduler with job preemption where newly arriving jobs will immediately run. Assume a time slice of 3 timer units. Draw vertical lines for key events and be sure to label the X-axis times as in the example.

Job	Arrival Time	Job Length
A	T=0	25
B	T=5	10
C	T=10	15

November 2, 2021

TCSS422: Operating Systems [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L10.63

63

## Q6 – RR SCHEDULER - 2

Using the graph, from time t=10 until all jobs complete at t=50, evaluate Jain's Fairness Index:

Jain's fairness index is expressed as:

$$\mathcal{J}(x_1, x_2, \dots, x_n) = \frac{(\sum_{i=1}^n x_i)^2}{n \cdot \sum_{i=1}^n x_i^2}$$

Where n is the number of jobs, and  $x_i$  is the time share of each process Jain's fairness index=1 for best case fairness, and  $1/n$  for worst case fairness.

For the time window from t=10 to t=50, what percentage of the CPU time is allocated to each of the jobs A, B, and C?

Job A: \_\_\_\_\_                      Job B: \_\_\_\_\_                      Job C: \_\_\_\_\_

With these values, calculate Jain's fairness index from t=10 to t=50.

November 2, 2021

TCSS422: Operating Systems [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L10.64

64



## Q7 – SLOPPY COUNTER

Below is a tradeoff space graph similar to those we've shown in class. Based on the sloppy counter threshold (S), add numbers on the **left** or **right** side of the graph for each of the following tradeoffs:

1. High number of Global Updates	2. High Performance
3. High Overhead	4. High Accuracy
5. Low number of Global Updates	6. Low Performance
7. Low Overhead	8. Low Accuracy

Low sloppy threshold (S) High sloppy threshold (S)

|-----|

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.65
------------------	---	--------

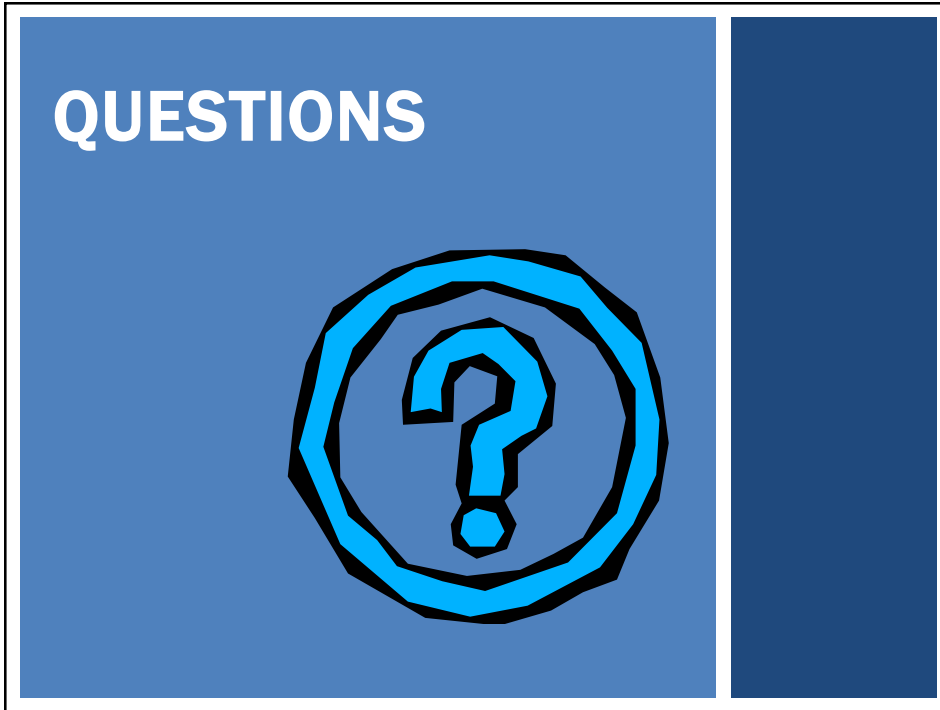
65

## MULTI-LEVEL FEEDBACK QUEUE

- Review the bonus lecture for scheduling examples including several Multi-level-feedback-queue problems (MLFQ)
  
- <https://tinyurl.com/cxtau9zw>

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.66
------------------	---	--------

66



67