


# TCSS 422: OPERATING SYSTEMS

## Lock-based data structures, Midterm Review

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



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

1

# OBJECTIVES – 4/30

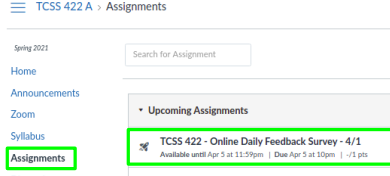
- **Questions from 4/27**
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 26
- Assignment 1 - Due Tue May 7
- Quiz 1 (Due Thur Apr 25) – Quiz 2 (Due Tue Apr 30)
- Chapter 28: Locks
- Chapter 29: Lock Based Data Structures
  - Approximate Counter (Sloppy Counter)
  - Concurrent Structures: Linked List, Queue, Hash Table
- Practice Midterm – 2<sup>nd</sup> hour

April 30, 2024
TCSS422: Operating Systems [Spring 2024]  
 School of Engineering and Technology, University of Washington - Tacoma
L11.2

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



April 30, 2024
TCSS422: Computer Operating Systems [Spring 2024]  
 School of Engineering and Technology, University of Washington - Tacoma
L11.3

3

# TCS422 - 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 30, 2024
TCSS422: Computer Operating Systems [Spring 2024]  
 School of Engineering and Technology, University of Washington - Tacoma
L11.4

4

# MATERIAL / PACE

- Please classify your perspective on material covered in today's class (29 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.98 (↑ - previous 6.45)**
- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.52 (↑ - previous 5.10)**

April 30, 2024
TCSS422: Computer Operating Systems [Spring 2024]  
 School of Engineering and Technology, University of Washington - Tacoma
L11.5

5

# FEEDBACK FROM 4/25

- **Which APIs are user calls, and which are kernel calls?**
- From Chapter 29:
  - pthread\_create()
  - pthread\_join()
  - pthread\_mutex\_lock()
  - pthread\_mutex\_unlock()
  - pthread\_mutex\_trylock()
  - pthread\_mutex\_timelock()
  - pthread\_cond\_wait()
  - pthread\_cond\_wait\_signal()
  - pthread\_cond\_wait\_broadcast()

April 30, 2024
TCSS422: Operating Systems [Spring 2024]  
 School of Engineering and Technology, University of Washington - Tacoma
L11.6

6

### OBJECTIVES – 4/30

- Questions from 4/27
- 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 Apr 30)
- Chapter 28: Locks
- Chapter 29: Lock Based Data Structures
  - Approximate Counter (Sloppy Counter)
  - Concurrent Structures: Linked List, Queue, Hash Table
- Practice Midterm – 2<sup>nd</sup> hour

April 30, 2024 TCSS422: Operating Systems [Spring 2024]  
School of Engineering and Technology, University of Washington - Tacoma L11.7

7

### OBJECTIVES – 4/30

- Questions from 4/27
- 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 Apr 30)
- Chapter 28: Locks
- Chapter 29: Lock Based Data Structures
  - Approximate Counter (Sloppy Counter)
  - Concurrent Structures: Linked List, Queue, Hash Table
- Practice Midterm – 2<sup>nd</sup> hour

April 30, 2024 TCSS422: Operating Systems [Spring 2024]  
School of Engineering and Technology, University of Washington - Tacoma L11.8

8

### OBJECTIVES – 4/30

- Questions from 4/27
- 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 Apr 30)**
- Chapter 28: Locks
- Chapter 29: Lock Based Data Structures
  - Approximate Counter (Sloppy Counter)
  - Concurrent Structures: Linked List, Queue, Hash Table
- Practice Midterm – 2<sup>nd</sup> hour

April 30, 2024 TCSS422: Operating Systems [Spring 2024]  
School of Engineering and Technology, University of Washington - Tacoma L11.9

9


### 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 30<sup>th</sup> at 11:59pm
- Link:  
<https://canvas.uw.edu/courses/1728244/quizzes/2030525>

April 30, 2024 TCSS422: Operating Systems [Spring 2024]  
School of Engineering and Technology, University of Washington - Tacoma L11.10

10

## CHAPTER 29 – LOCK BASED DATA STRUCTURES



April 30, 2024 TCSS422: Operating Systems [Spring 2024]  
School of Engineering and Technology, University of Washington - Tacoma L11.11

11

### OBJECTIVES – 4/30

- Questions from 4/27
- 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 Apr 30)
- Chapter 28: Locks
- Chapter 29: Lock Based Data Structures
  - Approximate Counter (Sloppy Counter)**
  - Concurrent Structures: Linked List, Queue, Hash Table
- Practice Midterm – 2<sup>nd</sup> hour

April 30, 2024 TCSS422: Operating Systems [Spring 2024]  
School of Engineering and Technology, University of Washington - Tacoma L11.12

12

### 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 30, 2024
TCCS422: Operating Systems (Spring 2024)  
School of Engineering and Technology, University of Washington - Tacoma
L11.13

13

### 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 30, 2024
TCCS422: Operating Systems (Spring 2024)  
School of Engineering and Technology, University of Washington - Tacoma
L11.14

14

### 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> )

April 30, 2024
TCCS422: Operating Systems (Spring 2024)  
School of Engineering and Technology, University of Washington - Tacoma
L11.15

15

### THRESHOLD VALUE S

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

April 30, 2024
TCCS422: Operating Systems (Spring 2024)  
School of Engineering and Technology, University of Washington - Tacoma
L11.16

16

### APPROXIMATE COUNTER - EXAMPLE

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

April 30, 2024
TCCS422: Operating Systems (Spring 2024)  
School of Engineering and Technology, University of Washington - Tacoma
L11.17

17

When poll is active, respond at [polllev.com/wesleylloyd641](https://polllev.com/wesleylloyd641)  
 Text **WESLEYLOYD641** 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 softwares, share the entire screen. Get help at [polllev.com/app](https://polllev.com/app)

18

## OBJECTIVES – 4/30

- Questions from 4/27
- 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 Apr 30)
- Chapter 28: Locks
- Chapter 29: Lock Based Data Structures
  - Sloppy Counter
  - **Concurrent Structures: Linked List**, Queue, Hash Table
- Practice Midterm – 2<sup>nd</sup> hour

April 30, 2024
TCCS422: Operating Systems (Spring 2024)  
School of Engineering and Technology, University of Washington - Tacoma
L11.19

19

## 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 30, 2024
TCCS422: Operating Systems (Spring 2024)  
School of Engineering and Technology, University of Washington - Tacoma
L11.20

20

## CONCURRENT LINKED LIST - 2

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

```

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 30, 2024
TCCS422: Operating Systems (Spring 2024)  
School of Engineering and Technology, University of Washington - Tacoma
L11.21

21

## CONCURRENT LINKED LIST - 3

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

```

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

April 30, 2024
TCCS422: Operating Systems (Spring 2024)  
School of Engineering and Technology, University of Washington - Tacoma
L11.22

22

## 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 30, 2024
TCCS422: Operating Systems (Spring 2024)  
School of Engineering and Technology, University of Washington - Tacoma
L11.23

23

## 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
22 (Cont.)
    
```

April 30, 2024
TCCS422: Operating Systems (Spring 2024)  
School of Engineering and Technology, University of Washington - Tacoma
L11.24

24

## 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 30, 2024
TCSS422: Operating Systems [Spring 2024]  
School of Engineering and Technology, University of Washington - Tacoma
L11.25

25

## 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 30, 2024
TCSS422: Operating Systems [Spring 2024]  
School of Engineering and Technology, University of Washington - Tacoma
L11.26

26

## OBJECTIVES – 4/30

- Questions from 4/27
- 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 Apr 30)
- Chapter 28: Locks
- Chapter 29: Lock Based Data Structures
  - Sloppy Counter
  - Concurrent Structures: Linked List, Queue, Hash Table
- Practice Midterm – 2<sup>nd</sup> hour

April 30, 2024
TCSS422: Operating Systems [Spring 2024]  
School of Engineering and Technology, University of Washington - Tacoma
L11.27

27

## 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 30, 2024
TCSS422: Operating Systems [Spring 2024]  
School of Engineering and Technology, University of Washington - Tacoma
L11.28

28

## 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 30, 2024
TCSS422: Operating Systems [Spring 2024]  
School of Engineering and Technology, University of Washington - Tacoma
L11.29

29

## 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 30, 2024
TCSS422: Operating Systems [Spring 2024]  
School of Engineering and Technology, University of Washington - Tacoma
L11.30

30

## OBJECTIVES – 4/30

- Questions from 4/27
- 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 Apr 30)
- Chapter 28: Locks
- Chapter 29: Lock Based Data Structures
  - Sloppy Counter
  - Concurrent Structures: Linked List, Queue, **Hash Table**
- Practice Midterm – 2<sup>nd</sup> hour

April 30, 2024
TCSS422: Operating Systems [Spring 2024]  
School of Engineering and Technology, University of Washington - Tacoma
L11.31

31

## 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 30, 2024
TCSS422: Operating Systems [Spring 2024]  
School of Engineering and Technology, University of Washington - Tacoma
L11.32

32

## 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	~1.5	~1.0
20	~3.0	~1.5
30	~5.0	~2.0
40	~8.0	~2.5
50	~12.0	~3.0

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

April 30, 2024
TCSS422: Operating Systems [Spring 2024]  
School of Engineering and Technology, University of Washington - Tacoma
L11.33

33

## 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 30, 2024
TCSS422: Operating Systems [Spring 2024]  
School of Engineering and Technology, University of Washington - Tacoma
L11.34

34

### 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 structure

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

35

## 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 30, 2024
TCSS422: Operating Systems [Spring 2024]  
School of Engineering and Technology, University of Washington - Tacoma
L11.36

36

**WE WILL RETURN AT  
5:00PM**



April 30, 2024 TCSS422: Operating Systems [Spring 2024]  
 School of Engineering and Technology, University of Washington - Tacoma L11.37

37

**OBJECTIVES – 4/30**

- Questions from 4/27
- 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 Apr 30)
- Chapter 28: Locks
- Chapter 29: Lock Based Data Structures
  - Sloppy Counter
  - Concurrent Structures: Linked List, Queue, Hash Table
- **Practice Midterm – 2<sup>nd</sup> hour**

April 30, 2024 TCSS422: Operating Systems [Spring 2024]  
 School of Engineering and Technology, University of Washington - Tacoma L11.38

38

**MIDTERM  
REVIEW**



April 30, 2024 TCSS422: Operating Systems [Spring 2024]  
 School of Engineering and Technology, University of Washington - Tacoma L11.39

39

**MIDTERM**

- Thursday May 2<sup>nd</sup>
- In Class in BHS 106 (2.0 hrs 3:40 – 5:40p)
- 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
- **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

April 30, 2024 TCSS422: Operating Systems [Spring 2024]  
 School of Engineering and Technology, University of Washington - Tacoma L11.40

40

**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** | AAAAAAAAAAABBBBBCCCCCCCC

0                      10      15                      25

April 30, 2024 TCSS422: Operating Systems [Spring 2024]  
 School of Engineering and Technology, University of Washington - Tacoma L11.41

41

**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

April 30, 2024 TCSS422: Operating Systems [Spring 2024]  
 School of Engineering and Technology, University of Washington - Tacoma L11.42

42

### 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? \_\_\_\_\_

April 30, 2024
TCSS422: Operating Systems (Spring 2024)  
School of Engineering and Technology, University of Washington - Tacoma
L11.43

43

### 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

April 30, 2024
TCSS422: Operating Systems (Spring 2024)  
School of Engineering and Technology, University of Washington - Tacoma
L11.44

44

### 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? \_\_\_\_\_

April 30, 2024
TCSS422: Operating Systems (Spring 2024)  
School of Engineering and Technology, University of Washington - Tacoma
L11.45

45

### 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.

April 30, 2024
TCSS422: Operating Systems (Spring 2024)  
School of Engineering and Technology, University of Washington - Tacoma
L11.46

46

### 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:

April 30, 2024
TCSS422: Operating Systems (Spring 2024)  
School of Engineering and Technology, University of Washington - Tacoma
L11.47

47

### 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.

April 30, 2024
TCSS422: Operating Systems (Spring 2024)  
School of Engineering and Technology, University of Washington - Tacoma
L11.48

48



[Empty slide content]

April 30, 2024 TCSS422: Operating Systems (Spring 2024) School of Engineering and Technology, University of Washington - Tacoma L11.4 9

49

### 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

RR

April 30, 2024 TCSS422: Operating Systems (Spring 2024) School of Engineering and Technology, University of Washington - Tacoma L11.50

50

### 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.

April 30, 2024 TCSS422: Operating Systems (Spring 2024) School of Engineering and Technology, University of Washington - Tacoma L11.51

51

### Q6 - II

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

April 30, 2024 TCSS422: Operating Systems (Spring 2024) School of Engineering and Technology, University of Washington - Tacoma L11.5 2

52

### 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)

April 30, 2024 TCSS422: Operating Systems (Spring 2024) School of Engineering and Technology, University of Washington - Tacoma L11.53

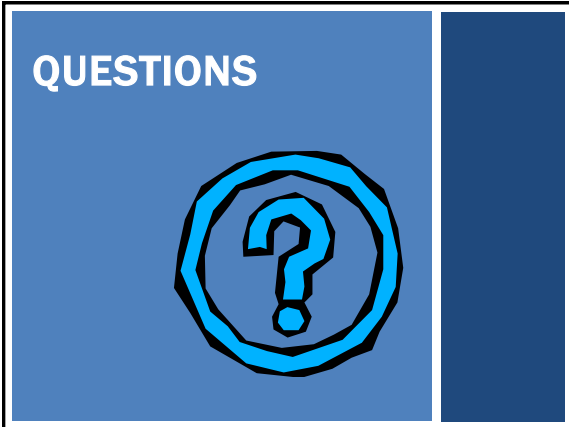
53

### MULTI-LEVEL FEEDBACK QUEUE

- Review the bonus lecture for scheduling examples including several Multi-level-feedback-queue problems (MLFQ)
- <https://tinyurl.com/422s24-practice>

April 30, 2024 TCSS422: Operating Systems (Spring 2024) School of Engineering and Technology, University of Washington - Tacoma L11.54

54



69