TCSS 422: OPERATING SYSTEMS

Lock-based data structures, Midterm Review

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

May 6, 2025 TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

1

OBJECTIVES – 5/6

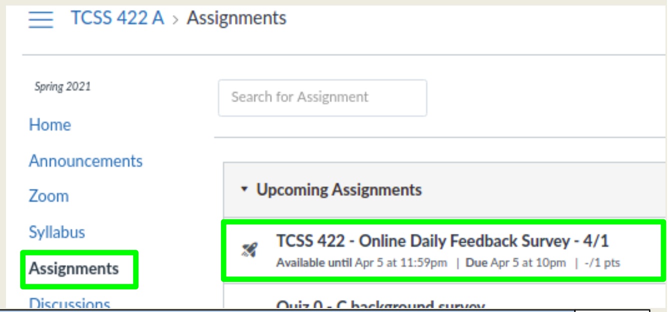
- Questions from 5/1
- C Tutorial - Pointers, Strings, Exec in C - Close May 4 AOE
- Assignment 1 - Due Tue May 13 AOE
- Quiz 1 (Close May 5 AOE) – Quiz 2 (Due Tue May 6 AOE)
- Chapter 28: Locks
- Chapter 29: Lock Based Data Structures
  - Approximate Counter (Sloppy Counter)
  - Concurrent Structures: Linked List, Queue, Hash Table
- Practice Midterm – 2nd hour

May 6, 2025 TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma L11.2

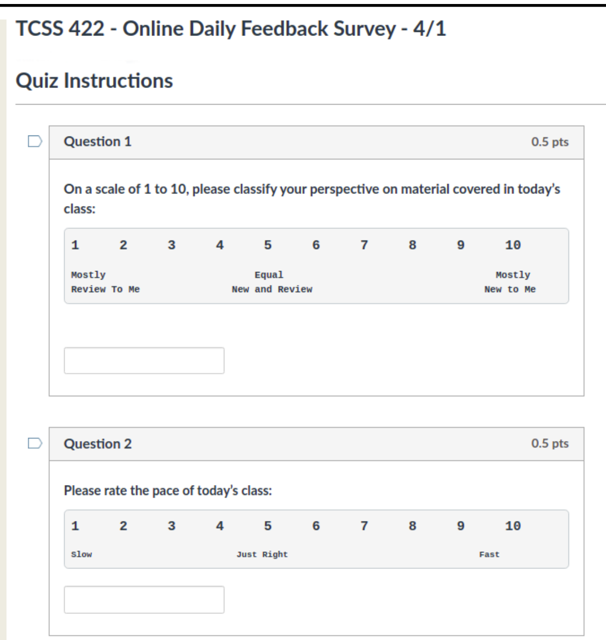
2

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



3



4

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (46 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.70 (↑ - previous 5.73)**

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.24 (↑ - previous 4.80)**

FEEDBACK FROM 5/1

- ***How do we guarantee in "test and set" that the set value is different from the original value?***

```
void lock(lock_t *lock) {
    while (TestAndSet(&lock->flag, 1) == 1)
        ;        // spin-wait
}
```

- Atomic TestAndSet() is called within lock()
- The output from TestAndSet is inspected
- Only if the returned 'old' value from TestAndSet() is ZERO, do we acquire the lock

TEST AND SET

- C pseudo code

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

- Chat GPT can provide the assembly code for x86 "TestAndSet"

```
mov eax, 1
lock xchg eax, [lock_var] ; Atomically set [lock_var] to 1, get old value in eax
```

- 1 is loaded into eax register
- 'lock' forces the xchg instruction to be atomic
- xchg swaps the values eax ←→ [lock_var]
- Old lock_var is in eax, can be checked

X86 ASSEMBLY LOCK

- Chat GPT can provide the assembly spinlock using xchg

```
acquire_lock:
    mov eax, 1                ; Value to set
.spin:
    lock xchg eax, [lock_var] ; Atomically swap eax and lock_var
    test eax, eax             ; Was the previous value 0?
    jnz .spin                 ; If not, spin (someone else has the lock)
    ret
```

- Notice the use of a 'goto' =)
  - jnz is a conditional jump (or goto)

REVIEW QUESTION

- ***Which APIs are non-blocking API calls?***
- From Chapter 27:
  - pthread_create()
  - pthread_join()
  - pthread_mutex_lock()
  - pthread_mutex_unlock()
  - pthread_mutex_trylock()
  - pthread_mutex_timelock()
  - pthread_cond_wait()
  - pthread_cond_signal()
  - pthread_cond_broadcast()

OBJECTIVES – 5/6

- Questions from 5/1
- C Tutorial - Pointers, Strings, Exec in C - Close May 4 AOE
- Assignment 1 - Due Tue May 13 AOE
- Quiz 1 (Close May 5 AOE) – Quiz 2 (Due Tue May 6 AOE)
- Chapter 28: Locks
- Chapter 29: Lock Based Data Structures
  - Approximate Counter (Sloppy Counter)
  - Concurrent Structures: Linked List, Queue, Hash Table
- Practice Midterm – 2nd hour

OBJECTIVES – 5/6

- Questions from 5/1
- C Tutorial - Pointers, Strings, Exec in C - Close May 4 AOE
- **Assignment 1 - Due Tue May 13 AOE**
- Quiz 1 (Close May 5 AOE) – Quiz 2 (Due Tue May 6 AOE)
- Chapter 28: Locks
- Chapter 29: Lock Based Data Structures
  - Approximate Counter (Sloppy Counter)
  - Concurrent Structures: Linked List, Queue, Hash Table
- Practice Midterm – 2nd hour

11

OBJECTIVES – 5/6

- Questions from 5/1
- C Tutorial - Pointers, Strings, Exec in C - Close May 4 AOE
- Assignment 1 - Due Tue May 13 AOE
- **Quiz 1 (Close May 5 AOE) – Quiz 2 (Due Tue May 6 AOE)**
- Chapter 28: Locks
- Chapter 29: Lock Based Data Structures
  - Approximate Counter (Sloppy Counter)
  - Concurrent Structures: Linked List, Queue, Hash Table
- Practice Midterm – 2nd hour

12

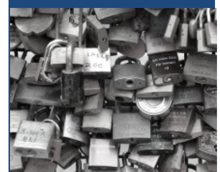
QUIZ 2

- 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 May 6th AOE
- Link:
- https://canvas.uw.edu/courses/1809484/assignments/10329061

CATCH UP FROM LECTURE 10

- Switch to Lecture 10 Slides
- Slides L10.44 to L10.50
  (Chapter 29 –Lock Based Data Structures)

CHAPTER 29 – LOCK BASED DATA STRUCTURES



May 6, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L11.15

15

OBJECTIVES – 5/6

- Questions from 5/1
- C Tutorial - Pointers, Strings, Exec in C - Close May 4 AOE
- Assignment 1 - Due Tue May 13 AOE
- Quiz 1 (Close May 5 AOE) – Quiz 2 (Due Tue May 6 AOE)
- Chapter 28: Locks
- Chapter 29: Lock Based Data Structures
  - **Approximate Counter (Sloppy Counter)**
  - Concurrent Structures: Linked List, Queue, Hash Table
- Practice Midterm – 2^{nd} hour

May 6, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L11.16

16

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?

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)

APPROXIMATE COUNTER - 2

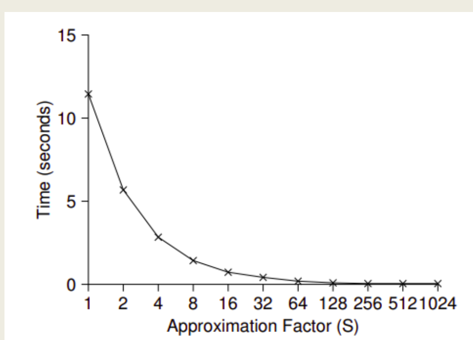
- Update threshold (*S*) = 5
- Synchronized across four CPU cores
- Threads update local CPU counters

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)

19

THRESHOLD VALUE S

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



20

APPROXIMATE COUNTER - EXAMPLE

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

When poll is active, respond at **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**

OBJECTIVES – 5/6

- Questions from 5/1
- C Tutorial - Pointers, Strings, Exec in C - Close May 4 AOE
- Assignment 1 - Due Tue May 13 AOE
- Quiz 1 (Close May 5 AOE) – Quiz 2 (Due Tue May 6 AOE)
- Chapter 28: Locks
- Chapter 29: Lock Based Data Structures
  - Sloppy Counter
  - **Concurrent Structures: Linked List**, Queue, Hash Table
- Practice Midterm – 2^{nd} hour

CONCURRENT LINKED LIST - 1

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

```
// basic node structure
typedef struct __node_t {
        int key;
        struct __node_t *next;
} node_t;

// basic list structure (one used per list)
typedef struct __list_t {
        node_t *head;
        pthread_mutex_t lock;
} list_t;

void List_Init(list_t *L) {
        L->head = NULL;
        pthread_mutex_init(&L->lock, NULL);
}

(Cont.)
```

CONCURRENT LINKED LIST - 2

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

```
(Cont.)
int List_Insert(list_t *L, int key) {
        pthread_mutex_lock(&L->lock);
        node_t *new = malloc(sizeof(node_t));
        if (new == NULL) {
                perror("malloc");
                pthread_mutex_unlock(&L->lock);
        return -1; // fail }
        new->key = key;
        new->next = L->head;
        L->head = new;
        pthread_mutex_unlock(&L->lock);
        return 0; // success
}
(Cont.)
```

May 6, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L11.25

25

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.)

int List_Lookup(list_t *L, int key) {
        pthread_mutex_lock(&L->lock);
        node_t *curr = L->head;
        while (curr) {
                if (curr->key == key) {
                        pthread_mutex_unlock(&L->lock);
                        return 0; // success
                }
                curr = curr->next;
        }
        pthread_mutex_unlock(&L->lock);
        return -1; // failure
}
```

May 6, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L11.26

26

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 …

CCL – SECOND IMPLEMENTATION

- Init and Insert

```
void List_Init(list_t *L) {
        L->head = NULL;
        pthread_mutex_init(&L->lock, NULL);
}

void List_Insert(list_t *L, int key) {
        // synchronization not needed
        node_t *new = malloc(sizeof(node_t));
        if (new == NULL) {
                perror("malloc");
                return;
        }
        new->key = key;

        // just lock critical section
        pthread_mutex_lock(&L->lock);
        new->next = L->head;
        L->head = new;
        pthread_mutex_unlock(&L->lock);
}
```

CCL – SECOND IMPLEMENTATION - 2

- Lookup

```
(Cont.)
int List_Lookup(list_t *L, int key) {
        int rv = -1;
        pthread_mutex_lock(&L->lock);
        node_t *curr = L->head;
        while (curr) {
                if (curr->key == key) {
                        rv = 0;
                        break;
                }
                curr = curr->next;
        }
        pthread_mutex_unlock(&L->lock);
        return rv; // now both success and failure
}
```


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?

OBJECTIVES – 5/6

- Questions from 5/1
- C Tutorial - Pointers, Strings, Exec in C - Close May 4 AOE
- Assignment 1 - Due Tue May 13 AOE
- Quiz 1 (Close May 5 AOE) – Quiz 2 (Due Tue May 6 AOE)
- Chapter 28: Locks
- Chapter 29: Lock Based Data Structures
  - Sloppy Counter
  - Concurrent Structures: Linked List, **Queue**, Hash Table
- Practice Midterm – 2nd hour

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

CONCURRENT QUEUE

- Remove from queue

```
typedef struct __node_t {
        int value;
        struct __node_t *next;
} node_t;

typedef struct __queue_t {
        node_t *head;
        node_t *tail;
        pthread_mutex_t headLock;
        pthread_mutex_t tailLock;
} queue_t;

void Queue_Init(queue_t *q) {
        node_t *tmp = malloc(sizeof(node_t));
        tmp->next = NULL;
        q->head = q->tail = tmp;
        pthread_mutex_init(&q->headLock, NULL);
        pthread_mutex_init(&q->tailLock, NULL);
}

(Cont.)
```


CONCURRENT QUEUE - 2

- Add to queue

```
(Cont.)
void Queue_Enqueue(queue_t *q, int value) {
        node_t *tmp = malloc(sizeof(node_t));
        assert(tmp != NULL);

        tmp->value = value;
        tmp->next = NULL;

        pthread_mutex_lock(&q->tailLock);
        q->tail->next = tmp;
        q->tail = tmp;
        pthread_mutex_unlock(&q->tailLock);
}
(Cont.)
```

OBJECTIVES – 5/6

- Questions from 5/1
- C Tutorial - Pointers, Strings, Exec in C - Close May 4 AOE
- Assignment 1 - Due Tue May 13 AOE
- Quiz 1 (Close May 5 AOE) – Quiz 2 (Due Tue May 6 AOE)
- Chapter 28: Locks
- Chapter 29: Lock Based Data Structures
  - Sloppy Counter
  - Concurrent Structures: Linked List, Queue, **Hash Table**
- Practice Midterm – 2nd hour

May 6, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L11.35

35

CONCURRENT HASH TABLE

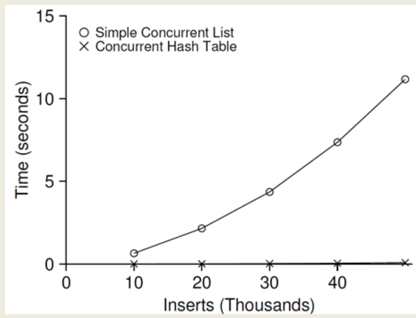
- Consider a simple hash table
  - Fixed (static) size
  - Hash maps to a bucket
    - Bucket is implemented using a concurrent linked list
    - One lock per hash (bucket)
    - Hash bucket is a linked lists

May 6, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L11.36

36

INSERT PERFORMANCE – CONCURRENT HASH TABLE

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



The simple concurrent hash table scales magnificently.

CONCURRENT HASH TABLE

```
#define BUCKETS (101)

typedef struct __hash_t {
        list_t lists[BUCKETS];
} hash_t;

void Hash_Init(hash_t *H) {
        int i;
        for (i = 0; i < BUCKETS; i++) {
                List_Init(&H->lists[i]);
        }
}

int Hash_Insert(hash_t *H, int key) {
        int bucket = key % BUCKETS;
        return List_Insert(&H->lists[bucket], key);
}

int Hash_Lookup(hash_t *H, int key) {
        int bucket = key % BUCKETS;
        return List_Lookup(&H->lists[bucket], key);
}
```

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

39

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

40

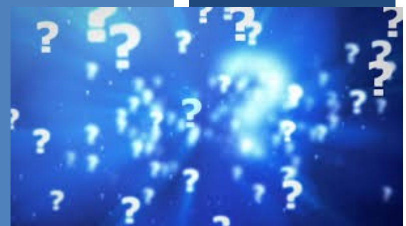
WE WILL RETURN AT 5:14PM

41

OBJECTIVES – 5/6

- Questions from 5/1
- C Tutorial - Pointers, Strings, Exec in C - Close May 4 AOE
- Assignment 1 - Due Tue May 13 AOE
- Quiz 1 (Close May 5 AOE) – Quiz 2 (Due Tue May 6 AOE)
- Chapter 28: Locks
- Chapter 29: Lock Based Data Structures
  - Sloppy Counter
  - Concurrent Structures: Linked List, Queue, Hash Table
- **Practice Midterm – 2nd hour**

42



MIDTERM REVIEW

May 6, 2025 TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma L11.43

43

MIDTERM

- Thursday May 8th
- Meet in BHS 106 (2.0 hrs 3:40 – 5:40p)
- Test designed to take less than 2 hours
- Three pages of notes, double-sided, any-size paper permitted
- No book, other notes, cell phones, or internet
- Basic calculators OK
- Individual work
- Coverage: up through Chapter 29

- ***Preparation:***
- Practice quiz: Quiz 2: CPU scheduling (***posted***)
  - Auto grading w/ multiple attempts allowed as study aid
- Practice Midterm Questions– second hour of lecture
  - Series of problems presented with some time to solve
  - Will then work through solutions

May 6, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L11.44

44

FIFO EXAMPLE

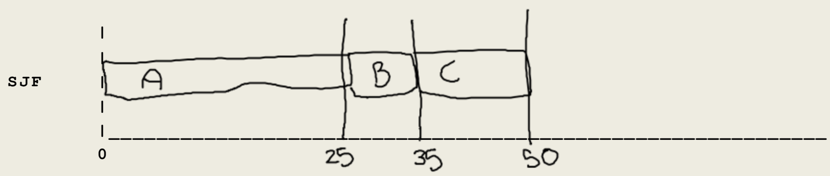
- Operation of CPU schedulers can be visualized with timing graphs.
- The graph below depicts a FIFO scheduler where three jobs arrive in the sequence A, B, C, where job A runs for 10 time slices, job B for 5 time slices, and job C for 10 time slices.

```
     |
FIFO |AAAAAAAAAABBBBBCCCCCCCCCC
     |______________________________________
     0          10   15       25
```


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



Q1 – SJF - 2

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

RT Job A: 0

TT Job A: 25

RT Job B: 25 - 5 = 20

TT Job B: 35 - 5 = 30

RT Job C: 35 - 10 = 25

TT Job C: 50 - 10 = 40

What is the average response time for all jobs? $\frac{45}{3} = 15$

What is the average turnaround time for all jobs? $\frac{95}{3} = 31.6\bar{6}$

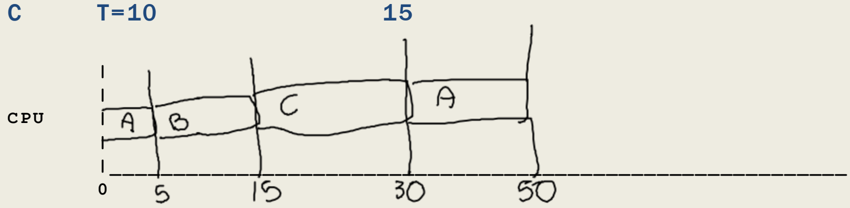
47

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~~ 20
B	T=5	~~10~~ ~~5~~ 0
C	T=10	15



48

Q2 – STCF - 2

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

RT Job A: 0

TT Job A: 50

RT Job B: 0

TT Job B: 15-5 =10

RT Job C: 15-10 = 5

TT Job C: 30-10 = 20

- What is the average response time for all jobs? $\frac{5}{3} = 1.\overline{66}$
- What is the average turnaround time for all jobs? $\frac{80}{3} = 26.\overline{66}$

Q3 - OPERATING SYSTEM APIs

1. Provide a definition for what is a blocking API call

   Kernel API that causes the process to block and wait for an operation to finish

2. Provide a definition for a non-blocking API call

   Kernel API that does not involve having process switch from running to blocked

3. Provide an example of a blocking API call.

   Consider APIs used to manage processes and/or threads.

   wait() pthread_mutex_lock()

4. Provide an example of a non-blocking API call.

   Consider APIs used to manage processes and/or threads.

   pthread_create() fork() execvp()

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()`

condition variables use a FIFO queue to guarantee the order that threads will receive the lock
(*) FAIRNESS

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()`

DEADLOCK overhead complexity - NEED A STATE VARIABLE

3. List (2) factors that cause Linux blocking API calls to introduce **overhead** into programs: W/ FINE-GRAINED LOCKING HIGH OVERHEAD
ALL KERNELS FEATURE - the system must context switch FROM USER MODE TO KERNEL MODE

May 6, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L11.51

51

Q5 – PERFECT MULTITASKING OPERATING SYSTEM

In a perfect-multi-tasking operating system, every process of the same priority will always receive exactly 1/nth 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.

May 6, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L11.52

52

JOB ACCOUNTING TAKES RESOURCES

CONTEXT SWITCHING INVOLVES O/H

SCHEDULING ALGORITHMS MUST BE PERFECTLY FAIR

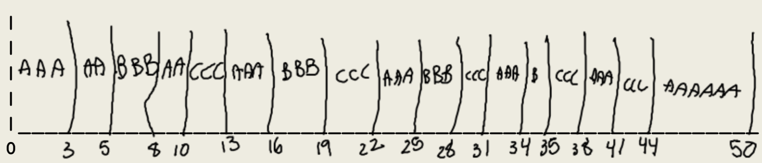
53

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



54

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:

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

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: .45 18/40 Job B: .175 7/40 Job C: .375 15/40

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

55

Q6 - II

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

.45
.175
.375

$3 \cdot [(.45)^2 + (.175)^2 + (.375)^2]$

$3\ [.2025 + .030625 + .140625]$

$3 \times [.37375]$

1.12125

$\frac{1}{1.12125} = .89186$

56

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) — 1 3 4 6

High sloppy threshold (S) — 2 5 7 8

May 6, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L11.57

57

Q8 – ROUND ROBIN SCHEDULER

- Consider a round-robin (RR) scheduler where upon new job arrival, the scheduler does not perform a context switch, but places the new job at the back of the RR queue.
  When the current job finishes its time quantum, a context switch is performed to execute the next job in the RR queue.
- The RR-scheduler has a 3-sec time slice.
- Draw a scheduling graph for the following jobs.

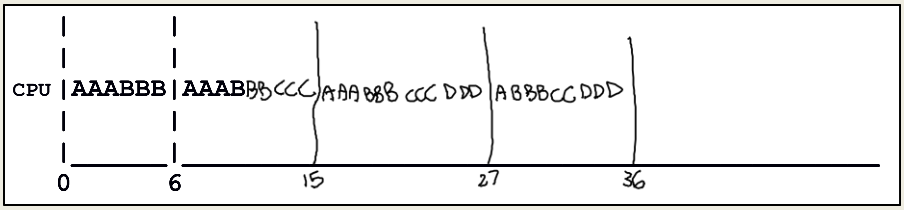
Job	Arrival Time	Job Length
A	T=0	10 seconds
B	T=2	12 seconds
C	T=10	8 seconds
D	T=16	6 seconds

May 6, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L11.58

58

Job	Arrival Time	Job Length
A	T=0	10 seconds
B	T=2	12 seconds
C	T=10	8 seconds
D	T=16	6 seconds

Starting Graph: ABCD



59

Job	Arrival Time	Job Length
A	T=0	10 seconds
B	T=2	12 seconds
C	T=10	8 seconds
D	T=16	6 seconds

Starting Graph:

```
    |      |
    |      |
CPU |AAABBB|AAAB
    |      |
    |______|________________________________________
    0      6
```

Solution:

```
    |      |         |            |         |
    |      |         |            |         |
CPU |AAABBB|AAABBBCCC|AAABBBCCCDDD|ABBBCCDDD|
    |      |         |            |         |
    |______|_________|____________|_________|__________
    0      6         15           27        36
```


60

Q8 PART B – ROUND ROBIN SCHEDULER

- Using the graph, calculate the turnaround time for each job, and the average turnaround time
- TT Job A: __28__ AVG TT: __100/4 = 25__
- TT Job B: __31-2 = 29__
- TT Job C: __33-10 = 23__
- TT Job D: __36-16 = 20__
- Calculate the response time for each job, and the average response time
- RT Job A: __0__ AVG RT: __11/4 = 2.75__
- RT Job B: __3-2 = 1__
- RT Job C: __12-10 = 2__
- RT Job D: __24-16 = 8__

61

Q8 PART B – ROUND ROBIN SCHEDULER

- Using the graph, calculate the turnaround time for each job, and the average turnaround time
- TT Job A: _28-0=28 AVG TT: 100/4=25_
- TT Job B: _31-2=29
- TT Job C: 33-10=23
- TT Job D: 36-16=20
- Calculate the response time for each job, and the average response time
- RT Job A: ___0____ AVG RT: 11/~~8=3.06~~ 4 = 2.75
- RT Job B: _3-2 = 1_
- RT Job C: 12-10=2_
- RT Job D: 24-16=8_

62

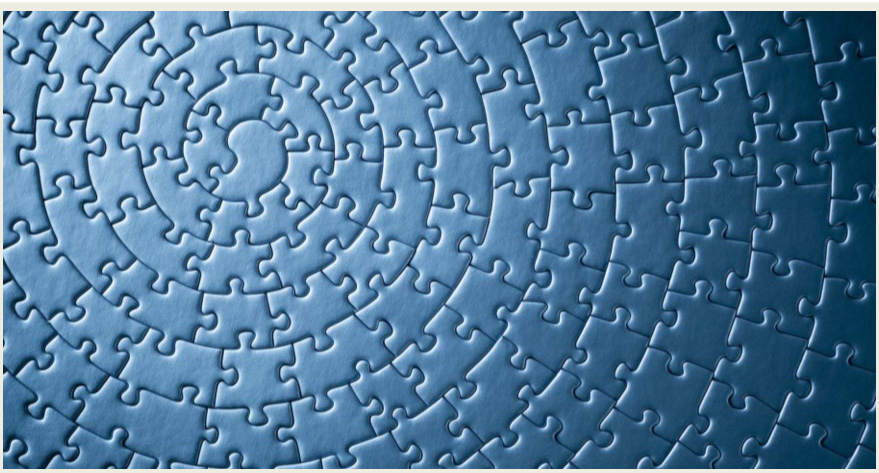
MULTI-LEVEL FEEDBACK QUEUE

- Review the bonus lecture for scheduling examples including several Multi-level-feedback-queue problems (MLFQ)
- Shortcut to Zoom recording of practice session:
- **https://tinyurl.com/422s25-practice**

May 6, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L11.63

63

SOLUTIONS



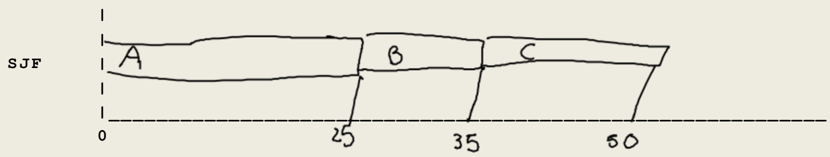
May 6, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L11.64

64

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



65

Q1 – SJF - 2

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

RT Job A: 0

TT Job A: 25

RT Job B: 25 - 5 = 20

TT Job B: 35 - 5 = 30

RT Job C: 35 - 10 = 25

TT Job C: 50 - 10 = 40

What is the average response time for all jobs? $\frac{0+20+25}{3} = \frac{45}{3} = 15$

What is the average turnaround time for all jobs? $\frac{25+30+40}{3} = \frac{95}{3}$ = 31.66

66

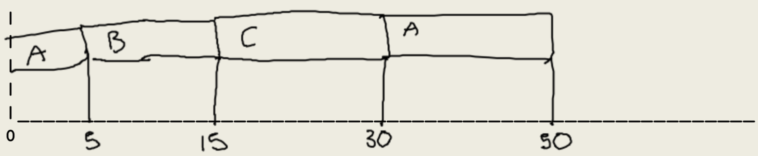
Q2 – SHORTEST TIME TO COMPLETION FIRST (STCF) SCHEDULER

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~~ 20
B	T=5	~~10~~ ~~5~~ 0
C	T=10	15

CPU



Q2 – STCF - 2

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

RT Job A: 0

RT Job B: 0

RT Job C: 15 − 10 = 5

TT Job A: 50

TT Job B: 15 − 5 = 10

TT Job C: 30 − 10 = 20

- What is the average response time for all jobs? $\frac{0+0+5}{3} = \frac{5}{3}$ = 1.66

- What is the average turnaround time for all jobs? $\frac{50+10+20}{3} = \frac{80}{3}$ → 26.66

Q3 - OPERATING SYSTEM APIs

1. Provide a definition for what is a blocking API call

AN API Call that suspends the CALLING THREAD TO WAIT FOR A system interrupt TO FIRE WHEN AN EVENT OCCURS. Calling Thread Goes to sleep. RUNNING → BLOCKED

2. Provide a definition for a non-blocking API call

AN API CALL that DOES NOT SUSPEND the CALLING Thread, but returns quickly AND DOES NOT WAIT FOR AN INTERRUPT TO OCCUR (OR EVENT)

3. Provide an example of a blocking API call.
Consider APIs used to manage processes and/or threads. Others?

pthread_mutex_lock() wait() waitpid()

4. Provide an example of a non-blocking API call. OTHERS?
Consider APIs used to manage processes and/or threads.

pthread_mutex_trylock() fork()

69

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()`

The combination ensures the order that blocked threads waiting for the lock will be woken up and given access to the lock. Threads wait in FIFO order.

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()`

- w/ pthread_mutex_lock the lock may never become available resulting in DEADLOCK
- DETAIL: must introduce and check state variable - more API calls

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

- blocking APIs must trap + context switch and be run in kernel mode introducing more overhead
- WITH FINE-GRAINED LOCKING, MANY CALLS TO LOCK APIS TO synchronize critical sections will increase overhead

70

Q5 – PERFECT MULTITASKING OPERATING SYSTEM

In a perfect-multi-tasking operating system, every process of the same priority will always receive exactly 1/nth 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.

71

2 challenges for perfect MULTITASING

- JOBS ARRIVE AT DIFFERENT TIMES AND RUN FOR DIFFERENT Lengths MAKING IT MORE DIFFICULT TO PERFECTLY BALANCE Runtime FOR JOBS IN the same PRIORITY QUEUE
- JOB ACCOUNTING (TRACKING TIME) INVOLVES overhead – MEASUREMENTS may NOT be PRECISE (VRUNTIME)
- context switching: # of context switches + overhead of a c/s may LEAD TO Inconsistencies IN JOB RUNTIME

72

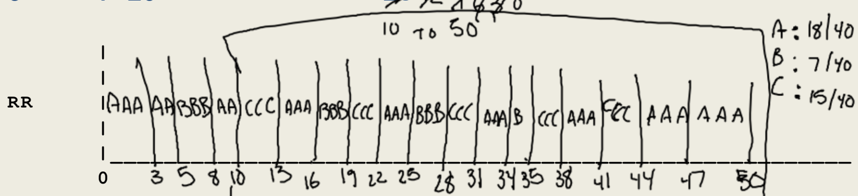
Q6 – ROUND-ROBIN SCHEDULER

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.

ARRIVING JOBS ARE ADDED TO THE BACK OF THE RUNQUEUE AND THE JOB PTR WILL JUMP TO THE MOST RECENTLY ADDED JOB – AND CONTINUES IN RR FASHION

Job	Arrival Time	Job Length
A	T=0	~~25~~ 20 18 15 12 9 6
B	T=5	~~10~~ 7 4 1 0
C	T=10	~~15~~ 12 9 6 3 0

RUNQUEUE: ABC



73

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:

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

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: 18/40 = .45 Job B: 7/40 = .175 Job C: 15/40 = .375

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

74

Q6 - II

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

worst case $\frac{1}{3} = .333$

perfect 1

$(.45 + .175 + .375) = (1)^2 = 1$

$n \cdot \sum_{i=1}^{n} x_i^2 \rightarrow 3 \cdot ((.45)^2 + (.175)^2 + (.375)^2)$

$3 \cdot (.2025 + .030625 + .140625)$

$3 \cdot (.37375)$

1.12125

$\rightarrow \frac{1}{1.12125}$

$\rightarrow .8918617$

$\sim 89.2\%$

May 6, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L11.75

75

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) — 1364

High sloppy threshold (S) — 5728

May 6, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L11.76

76

QUESTIONS

77