Homework 8

Graded

Student

Ashley Leora Cain

Total Points

55 / 100 pts

Question 1

Cache Tracing **1** / 25 pts

1.1 **Cache Trace 1** **0** / 8 pts

+ 8 pts Correct

Hit/Miss

+ 4 pts Marks conflict miss

+ 1 pt Marks cold/compulsory miss (previously referenced)

+ 0 pts Marked capacity miss (the cache is not full) or hit

Modified Cache

+ 2 pts Marks C0

+ 2 pts Marks correct cache for incorrect index (see below)

+ 0 pts Marks any other cache or None

Modified Index

+ 2 pts Marks Index 1

✔ **+ 0 pts** Marks any other index or None

+ 0 pts Incorrect

1.2 **Cache Trace 2** **0** / 8 pts

+ 8 pts Correct

Hit/Miss

+ 4 pts Marks hit

✔ **+ 0 pts** Marks any miss

Modified Cache

+ 2 pts Marks None (no cache is modified on a hit)

+ 1 pt Marks C1 (the correct cache), even though it is not modified

+ 0 pts Marks any other cache

Modified Index

+ 2 pts Marks None

+ 1 pt Marks Index 3 (the correct index), even though it is not modified

+ 0 pts Marks any other index

+ 0 pts Incorrect

1.3 **Cache Trace 3** **0** / 8 pts

+ 8 pts Correct

Hit/Miss

+ 4 pts Marks cold/compulsory miss

+ 1 pt Marks any other miss

+ 0 pts Marks hit

Modified Cache

+ 2 pts Marks C3

+ 2 pts Marks correct cache for incorrect index (see below)

+ 0 pts Marks any other cache or None

Modified Index

+ 2 pts Marks Index 3

✔ **+ 0 pts** Marks any other index or None

+ 0 pts Incorrect

1.4 **(no title)** **1** / 1 pt

✔ **+ 1 pt** Correct (16)

+ 0 pts Incorrect

Question 2

Conditional Variable **24** / 24 pts

2.1 **Before Step 3** **8** / 8 pts

✔ **– 0 pts** Correct

m: NA
c: T1 m

– 0 pts Correct

– 4 pts Any thread holding Mutex

– 4 pts Incorrect condition variable (wrong thread, also T2, etc.)

– 8 pts Incorrect

2.2 **Before Step 5** **8** / 8 pts

✔ **– 0 pts** Correct

- m: T2, T1
- c: NA

– 0 pts Correct

Mutex

– 2 pts Incorrect order in mutex queue (T2 then T1)

– 2 pts Missing one thread in the condition variable queue

– 4 pts Incorrect mutex queue (beyond or all of the above cases; i.e. do not take off more than 4 points for mutex)

– 4 pts Incorrect condition variable (any thread in the queue)

– 8 pts Incorrect

2.3 **After All Steps** **8** / 8 pts

✔ **– 0 pts** Correct

- m: T1
- c: NA

– 0 pts Correct

– 4 pts Incorrect mutex

– 4 pts Incorrect condition variable (anyone in the queue)

– 8 pts Incorrect

Question 3

User vs. Kernel Level Threads **30** / 30 pts

3.1 **User > Kernel** **15** / 15 pts

✔ **+ 15 pts** Correct

+ 0 pts Click here to replace this description.

3.2 **Kernel > User** **15** / 15 pts

✔ **+ 15 pts** Correct

+ 0 pts no answer

Question 4

Multi-threaded Code Debugging **0** / 21 pts

4.1 **Multi-threaded Code Debugging** **0** / 21 pts

+ 7 pts Incorrect condition variable signaled in generator()

+ 7 pts consumer() dequeues before checking condition

+ 7 pts If statement in generator() should be a while loop

+ 7 pts Wait statements do not pass in a lock

+ 7 pts Entire functions wrapped in locks

✔ **– 21 pts** Wrong answer / incorrect

Q1 Cache Tracing
25 Points

You are given a 4-way set associative cache. Each cache set has four cache entries, valid bits, and "Least Recently Used (LRU)", which holds a sequence of caches ordered based on how recently were they used. For example, an entry of

$$C1 \rightarrow C2 \rightarrow C3 \rightarrow C0$$

indicates $C0$ as the least recently used cache entry.

In the tables below, an entry denotes the memory address whose data is stored at that location, not the data itself. Additionally, assume the entry in the "Valid" column for an index is the value of the valid bit for each entry at that index. An "O" means each entry in the index is valid, and an "X" means each entry in the index is invalid.

Q1.1 Cache Trace 1
8 Points

Given the current state of the cache below, what will happen if the CPU accesses the memory location of **41**? Assume that location 41 **has** been previously referenced in the cache.

Index	C0	C1	C2	C3	Valid	LRU
0	4	--	36	28	X	--
1	33	5	9	45	O	$C2 \rightarrow C3 \rightarrow C1 \rightarrow C0$
2	14	38	30	18	O	$C0 \rightarrow C2 \rightarrow C1 \rightarrow C3$
3	39	11	7	27	O	$C1 \rightarrow C3 \rightarrow C0 \rightarrow C2$

Hit/Miss & Type:

- ○ Hit
- ○ Miss - Cold/Compulsory
- ○ Miss - Capacity
- ◉ Miss - Conflict

Modified Cache Entry (Choose cache & its index)

Cache

- ○ C0
- ◉ C1
- ○ C2
- ○ C3
- ○ None

Index

- (•) Index 0
- ( ) Index 1
- ( ) Index 2
- ( ) Index 3
- ( ) None

Q1.2 Cache Trace 2
8 Points

Given the current state of the cache below, what will happen if the CPU accesses the memory location of **11**? Assume that location 11 **has** been previously referenced in the cache.

Index	C0	C1	C2	C3	Valid	LRU
0	12	16	0	8	O	$C1 \rightarrow C3 \rightarrow C2 \rightarrow C0$
1	1	13	21	9	O	$C2 \rightarrow C1 \rightarrow C0 \rightarrow C3$
2	-	-	-	-	X	----
3	3	11	7	19	O	$C1 \rightarrow C2 \rightarrow C3 \rightarrow C0$

Hit/Miss & Type:

- (●) Hit
- ( ) Miss - Cold/Compulsory
- ( ) Miss - Capacity
- ( ) Miss - Conflict

Modified Cache Entry (Select the cache & its index)

Cache

- ( ) C0
- (●) C1
- ( ) C2
- ( ) C3
- ( ) None

Index

- ○ Index 0
- ○ Index 1
- ○ Index 2
- ◉ Index 3
- ○ None

Q1.3 Cache Trace 3
8 Points

Given the current state of the cache below, what will happen if the CPU accesses the memory location of **27**? Assume that location 27 **has NOT** been previously referenced in the cache.

Index	C0	C1	C2	C3	Valid	LRU
0	32	4	16	8	O	$C2 \rightarrow C3 \rightarrow C1 \rightarrow C0$
1	5	13	33	41	O	$C3 \rightarrow C1 \rightarrow C2 \rightarrow C0$
2	18	14	26	22	O	$C0 \rightarrow C1 \rightarrow C3 \rightarrow C2$
3	35	43	15	7	O	$C1 \rightarrow C0 \rightarrow C2 \rightarrow C3$

Hit/Miss & Type:

- ○ Hit
- ○ Miss - Cold/Compulsory
- ○ Miss - Capacity
- ◉ Miss - Conflict

Modified Cache Entry (Choose cache & its index)

Cache

- ◉ C0
- ○ C1
- ○ C2
- ○ C3
- ○ None

Index

- [x] Index 0
- [ ] Index 1
- [ ] Index 2
- [ ] Index 3
- [ ] None

Q1.4
1 Point

How many bits are required to keep track of the ordering of 8 way set associative?

- [ ] 40320
- [ ] 256
- [x] 16
- [ ] 8

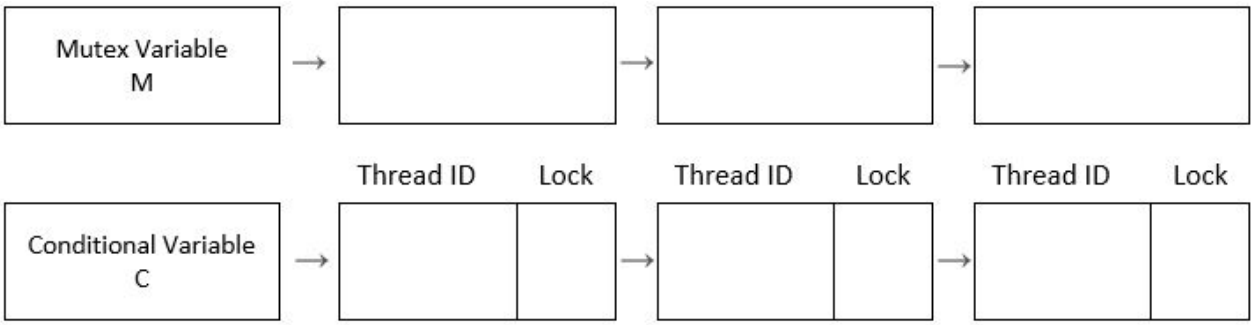
Q2 Conditional Variable

24 Points

Given a mutex lock m, and a conditional variable c, the following events happen in the order of occurrence stated below:

1. $T1$ executes mutex-lock(m).
2. $T1$ executes cond-wait(c, m).
3. $T2$ executes mutex-lock(m).
4. $T2$ executes cond-signal(c).
5. $T2$ executes mutex-unlock(m)

Fill in the boxes below. If there is no thread waiting for a lock or a signal, leave the boxes blank.



Q2.1 Before Step 3
8 Points

Write down the state of the two waiting queues after steps 1 and 2 are completed.

For the mutex variable m, write your answers as a comma-separated list with the first entry being the thread that currently has the lock. For example, if T1 has the lock and T2 is waiting for the lock, you should answer "T1, T2". If a queue is empty, write down "NA"

For the conditional variable c, write a comma-separated list of threads waiting in order that they arrived. For instance, if T1 and T2 are waiting for mutex lock m, you should answer "T1 m, T2 m." If a queue is empty, write down "NA".

Mutex variable m

NA

Conditional variable c

T1 m

Q2.2 Before Step 5
8 Points

Write down the state of the two waiting queues **before** step 5. Steps 1, 2, 3, and 4 have completed.

For the mutex variable m, write your answers as a comma-separated list with the first entry being the thread that currently has the lock. For example, if T1 has the lock and T2 is waiting for the lock, you should answer "T1, T2." If a queue is empty, write down "NA"

For the conditional variable c, write a comma-separated list of threads waiting in order that they arrived. For instance, if T1 and T2 are waiting for mutex lock m, you should answer "T1 m, T2 m." If a queue is empty, write down "NA".

Mutex variable m

T2, T1

Conditional variable c

NA

Q2.3 After All Steps
8 Points

Write down the state of the two waiting queues after all the steps are completed.

For the mutex variable m, write your answers as a comma-separated list with the first entry being the thread that currently has the lock. For example, if T1 has the lock and T2 is waiting for the lock, you should answer "T1, T2" If a queue is empty, write down "NA"

For the conditional variable c, write a comma-separated list of threads waiting in order that they arrived. If a queue is empty, write down "NA".

Mutex variable m

T1

Conditional variable c

NA

Q3 User vs. Kernel Level Threads

30 Points

Q3.1 User > Kernel

15 Points

Describe a situation in which you would want to use user-level threading over kernel-level, and explain why your example makes sense.

Since user level threads incur a minimal direct and indirect cost for context switching, and since a thread level scheduler is customizable for specific applications, the use of user level threading can be helpful in executing programs that may be heavy in context switches, comparable to making a procedure call. This cheap thread switching is a result of the user-level not involving the operating system, and instead the thread scheduler picks another thread to run within the same process in cases of a different blocked thread.

Q3.2 Kernel > User

15 Points

Describe a situation in which you would want to use kernel-level threading over user-level, and explain why your answer makes sense.

Device specific functions are examples of times when you would want to use a kernel thread. Kernel threads are a unit of scheduling and can be carried out independent from any user-level processes. Since certain functions and scheduling functionality can be happening outside of the user level, it is important that these procedures are still able to continue, regardless of user behavior and knowledge.

Q4 Multi-threaded Code Debugging

21 Points

Take a look at the following code snippet:

```
int QUEUE_IS_EMPTY;
int QUEUE_IS_FULL;

void generator(){
    Object thing = generate();
    pthread_mutex_lock(&qlock);
    if (QUEUE_IS_FULL){
        pthread_cond_wait(&queue_not_full);
    }
    enqueue(thing);
    pthread_cond_signal(&queue_not_full);
    pthread_mutex_unlock(&qlock);
}

void consumer(){
    pthread_mutex_lock(&qlock);
    Object thing = dequeue();
    consume(thing);
    while (QUEUE_IS_EMPTY){
        pthread_cond_wait(&queue_not_empty);
    }
    pthread_cond_signal(&queue_not_full);
    pthread_mutex_unlock(&qlock);
}
```

This code generates objects in `generator()`, and consumes them in `consumer()`.

Objects are placed into a queue to be consumed, and this queue has a limited size. `enqueue()` adds an item to the queue, and `dequeue()` removes one. An item may not be enqueued if the queue is full, or dequeued if the queue is empty. You may assume that `QUEUE_IS_EMPTY` and `QUEUE_IS_FULL` always correctly indicate whether the queue is empty, full, or neither, even if they are not updated in these methods.

Q4.1 Multi-threaded Code Debugging
21 Points

There are three multithreading related errors in this code that could result in deadlocking or other undesirable behavior. What are they?

1. Deadlock when while loop triggered for QUEUE_IS_EMPTY in consumer() and will never end since the queue_not_empty condition is never triggered to exit the while
2. mutex at start causes deadlock
3. consumer( ) immediately consumes the object created and will never signal queue_not_full in order for the generator to unlock qlock after the wait condition is fulfilled