Homework 8

● Graded

Student

Vidit Dharmendra Pokharna

Total Points

100 / 100 pts

Question 1

Page Coloring **16** / 16 pts

1.1 **Overlapping Bits** **10** / 10 pts

✔ **– 0 pts** Correct: 7

– 0 pts Correct

– 2.5 pts Incorrectly calculated number of bits for page offset (Correct: P = 10)

– 2.5 pts Used wrong block size for calculation (Correct: 2^8 bytes or B = 8)

– 2.5 pts Used wrong cache size for calculation (Correct: 2^{20} bytes or C = 20)

– 2.5 pts Incorrectly calculated number of index bits based on previous assumptions (Ideally: I = 9)

– 4 pts Incorrectly calculated number of required bits for page coloring (Ideally: B + I - P = 7)

– 10 pts Missing/Incorrect

1.2 **Overlap in Cache Address** **6** / 6 pts

✔ **+ 6 pts** Correct

+ 0 pts Incorrect

1.3 **Work (Optional)** **0** / 0 pts

✔ **+ 0 pts** Correct

Question 2

Test-and-Set **6** / 6 pts

✔ **+ 6 pts** Correct

+ 0 pts Incorrect

Question 3

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

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

✔ **+ 15 pts** Correct

+ 7.5 pts Identifies a valid example, e.g. single processor programming

+ 7.5 pts Gives a valid explanation, e.g. user-level threading is faster due to a much lighter context switch overhead when switching to another thread

+ 0 pts Incorrect/Blank

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

✔ **+ 15 pts** Correct

+ 7.5 pts Identifies a valid example, e.g. multiprocessor programming, many blocking operations

+ 7.5 pts Gives a valid explanation, e.g. kernel allows threads from a process to be scheduled on different processors, provide hardware concurrency

+ 0 pts Incorrect/Blank

Question 4

Conditional Variable **24** / 24 pts

4.1 **Before Step 3** **8** / 8 pts

✔ **– 0 pts** Correct

m: T2, T1
c: NA

– 0 pts Correct

Mutex

– 2 pts Wrong order of mutex queue (T1, T2)

– 4 pts Incorrect mutex queue (beyond and including the above case; i.e. do not take off more than 4 points for mutex)

– 4 pts Any thread in condition variable queue

– 8 pts Incorrect

4.2 **Before Step 5** **8** / 8 pts

✔ **– 0 pts** Correct

- m: T1
- c: NA

– 0 pts Correct

– 4 pts Incorrect mutex

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

– 8 pts Incorrect

4.3 **After All Steps** **8** / 8 pts

✔ **– 0 pts** Correct

- m: NA
- c: T1 m

– 0 pts Correct

– 4 pts Any thread holding the mutex

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

– 8 pts Incorrect

Question 5

Multi-threaded Code Debugging **24** / 24 pts

5.1 **Multi-threaded Code Debugging** **24** / 24 pts

- ✔ **+ 8 pts** Incorrect condition variable signaled in generator()
- ✔ **+ 8 pts** consumer() dequeues before checking condition
- ✔ **+ 8 pts** If statement in generator() should be a while loop
- **+ 0 pts** Wrong answer / incorrect

Q1 Page Coloring
16 Points

Page coloring is used to make sure that a few least significant bits of the virtual page number (VPN) and physical frame number (PFN) remain unchanged during address translation.

Imagine the following memory hierarchy:

- 64-bit virtual address
- 32-bit physical address
- Virtually-indexed, physically-tagged, 8-way set associative cache
- Page size of 1 KB
- Memory is byte-addressable
- Total Cache Size of 1 MB
- Cache block size of 256 bytes

Assume K = 1024 and M = 1024 * 1024.

Q1.1 Overlapping Bits
10 Points

How many of the least significant bits of the VPN must remain unchanged in the VPN-PFN translation?

7

Q1.2 Overlap in Cache Address
6 Points

Where in the cache address are the overlapping bits present? (End describes positions touching MSB, and beginning describes positions touching LSB)

- ○ End of the tag
- ○ Beginning of the index
- ○ Middle of the index
- ◉ End of the index
- ○ Beginning of the offset

Q1.3 Work (Optional)
0 Points

If you would like partial credit in case of an incorrect answer on the previous parts, show your work in the field below or attach it as a file:

No files uploaded

1.1:
10 bits offset (1 KB page size), 54 bit VPN (64 - 10 = 54)
b = log_2(256) = 8
L = 2^20 / 2^8 / 8 = 2^12 / 2^3 = 2 ^ 9
n = log_2(2^9) = 9
t = 64 - (9 + 8) = 47
54 - 47 = 7 bits

Q2 Test-and-Set

6 Points

In order to support synchronization between multiple processes, why do we need a test-and-set (T&S) instruction instead of just using existing load, store, and branch-if-zero instructions?

- ( ) processor speed
- ( ) efficiency
- (•) atomicity
- ( ) energy saving
- ( ) reduction in CPI

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.

Consider a scenario such as multimedia processing, which requires frequent context switching between different threads. In this case, using kernel-level threads can result in substantial overhead due to the frequent transitions between user and kernel modes for each context switch, leading to increased system resource consumption.
User-level threads, however, can be more efficient in managing these lightweight tasks. They are managed entirely by the application and the user-level thread library.

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.

Process P1 comprises two kernel-level threads, whereas process P2 consists of two user-level threads. In the event that one thread within P1 encounters a block, the second thread remains unaffected. Conversely, if one thread within P2 is blocked, perhaps due to an input/output operation, the entire process P2, including the second thread, becomes blocked.
In cases of blockage, kernel-level threads are designed to operate independently of one another. Thus, when one kernel-level thread encounters an obstacle, other threads within the same process can continue executing without impediment.

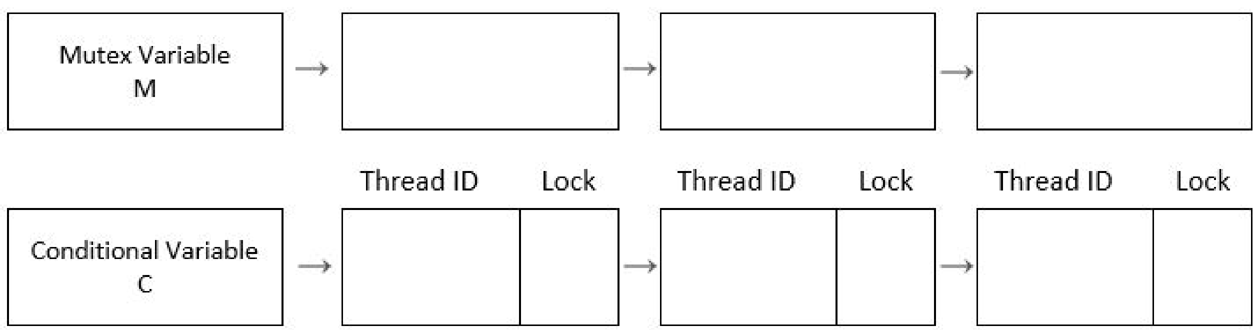
Q4 Conditional Variable

24 Points

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

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

Fill in the boxes as stated in the instructions below.



Q4.1 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

T2, T1

Conditional variable c

NA

Q4.2 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

T1

Conditional variable c

NA

Q4.3 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

NA

Conditional variable c

T1 m

Q5 Multi-threaded Code Debugging
24 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, &qlock);
    }
    enqueue(thing);
    pthread_cond_signal(&queue_not_full);
    pthread_mutex_unlock(&qlock);
}

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

Q5.1 Multi-threaded Code Debugging
24 Points

There are at least three logical errors in this code that could result in deadlocking or other undesirable behaviors. Identify three errors.

Note: Assume all locks and condition variables have been initialized correctly.

1. In the generator function, when checking if the queue is full, it should be while (QUEUE_IS_FULL) instead of if (QUEUE_IS_FULL). This change ensures that the thread rechecks the condition.

2. In the generator function, there is no purpose of pthread_cond_signal(&queue_not_full) as we just enqueued after finishing the while loop with the condition that queue is full. Thus, after all this, the queue would just be full again, meaning this line of code serves no purpose.

3. In the consumer function, we should shift the first three lines of code after the while loop because it can cause an error if the queue is empty, as we cannot dequeue from an empty queue. It would be purposeful if we moved it after the loop as we are certain the queue is not empty after the loop.