Please print in pen:
Waterloo Student ID Number:

WatIAM/Quest Login Userid:

UNIVERSITY OF WATERLOO

Examination
Midterm
Winter 2020
ECE 459

Times: Tuesday 2020-02-25 at 18:00 to 19:00 (6 to 7PM)
Duration: 1 hour (60 minutes)
Exam ID: 4455938
Sections: ECE 459 LEC 001
Instructors: Jeff Zarnett

Open Book

Candidates may bring any reasonable aids. Open book, open notes. Calculators without communication capability permitted.

Instructions:

1. This exam is open book, open notes, calculators with no communication capability permitted.
2. Turn off all communication devices. Communication devices must be stored with your personal items for the duration of the exam. Taking a communication device to a washroom break during this examination is not allowed and will be considered an academic offence.
3. Place all bags at the front or side of the examination room, or beneath your table, so they are inaccessible.
4. There are five (5) questions. Not all are equally difficult.
5. If space to answer a question is insufficient, you can use the back of the last page. Be sure to clearly indicate that your answer is continued on that page; on that page indicate which answer is being continued.
6. The exam lasts **60** minutes and there are 45 marks.
7. Verify that your name and student ID number is on the cover page.
8. If you feel like you need to ask a question, know that the most likely answer is “Read the Question”. No questions are permitted. If you find that a question requires clarification, proceed by clearly stating any reasonable assumptions necessary to complete the question. If your assumptions are reasonable, they will be taken into account during grading.
9. Do not fail this city.
10. After reading and understanding the instructions, sign your name in the space provided below.

Signature

Please initial:

1 Short Answer [15 Marks total]

Answer these questions using at most three sentences. Each question is worth 3 points.

(a) Is a single-threaded program guaranteed not to have race conditions? Explain your answer.

(b) The idea behind Google Stadia is that the game runs on a server in a data centre and your gameplay is sent like a video to your screen in your home. Obviously, latency will be an issue. Could Google use speculative execution to solve this for single-player games? Explain your answer.

(c) Under the MESI protocol, is it valid for CPU2 to issue a write to a location that is not in its cache, but is in the Exclusive state in the cache of CPU4? Explain.

(d) (Order of magnitude estimation, `computers-are-fast.github.io`) Given a SQLite table with 10 million rows (as in the website), how many times faster is a search if the table has an index as compared to when there is no index?

(e) What problem might helgrind report with this code and how would you fix it?

```
void transfer_funds( account* sender, account* recipient, double amount) {
  pthread_mutex_lock( sender->lock );
  pthread_mutex_lock( recipient->lock );
  sender->balance -= amount;
  recipient->balance += amount;
  pthread_mutex_unlock( sender->lock );
  pthread_mutex_unlock( recipient->lock );
}
```


Please initial:

2 Thread Pools [10 marks]

Consider the following test harness code (it runs unit tests). Given an array of tests (which take no arguments and return a `bool`; see the function pointer definition below). The previous coop student wrote the following code. This is inefficient because it creates and destroys a new thread for each test. Thus, if your program has 30 000 tests, the overhead costs of creating and destroying threads are paid every single time.

You know that can make this code more efficient using a *thread pool*. Rewrite the code to use a thread pool pattern. You may add or remove synchronization constructs as you see fit. Your code should not have any race conditions or memory leaks.

```
typedef bool (*test_fn)( ); /* Test function pointer */
int failed_tests = 0;
sem_t next;
sem_t done;
pthread_mutex_t lock;

void init()  {
    sem_init( &next, 0, threads );
    sem_init( &done, 0, 0 );
    pthread_mutex_init( &lock, NULL );
}

void* executor( void* arg ) {
    test_fn f = (test_fn) arg;
    bool success = f();
    pthread_mutex_lock( &lock );
    if ( !success ) {
      failed_tests++;
    }
    completed++;
    if ( completed == total_tests ) {
      sem_post( &done );
    }
    pthread_mutex_unlock( &lock );
    sem_post( &next );
}
```

```
void cleanup() {
    sem_destroy( &next );
    sem_destroy( &done );
    pthread_mutex_destroy( &lock );
}

int run_tests_parallel( test_fn * tests, int num_tests,
     int threads ) {
    init();
    total_tests = num_tests;

    for ( int i = 0; i < num_tests; i++ ) {
        pthread_t t;
        sem_wait( &next )
        pthread_create( &t, NULL, executor, tests[i] );
        pthread_detach( t );
    }
    sem_wait( &done );
    cleanup();
    return failed_tests;
}
```

Rewrite the code below.

Please initial:

3 OpenMP Tasks [8 marks]

Consider the following code that is used to calculate the golden ratio. Apply OpenMP tasks to parallelize this code. You can assume that you do not need to implement a cutoff to get a reasonable speedup in this calculation.

```
double gr( int n ) {

    if ( n >= 2 ) {

        double f, g;


        f = gr( n-1 );




        g = gr( n-1 );


        return ( ( f + 5 ) / g ) / 2;

    } else {

        return 2.0;


    }
}

int main( int argc, char *argv[] ) {
    if ( argc != 2 ) {
        printf( "%s: requires an argument > 10\n", argv[0] );
        return EXIT_FAILURE;
    }
    int n = atoi( argv[1] );
    if ( n <= 10 ) {
        printf( "%s: requires an argument > 10\n", argv[0] );
        return EXIT_FAILURE;
    }


    double ratio = gr( n );



    printf( "gr(%d) = %lf\n", n, ratio );



    return EXIT_SUCCESS;
}
```

4 Memory Consistency [3 marks]

Synchronization constructs like a spinlock or mutex frequently have `mfence` instructions in their implementations. Explain why this is necessary and give an example of a bad thing that could happen if it is not present.

Please initial:

5 Asynchronous I/O and Speedup [9 marks]

Consider the asynchronous I/O portion of assignment 1. Most requests take 50 ms. However, 1 out every 10 requests takes 500 ms. The provided input file contains 1000 puzzles. For the sake of simplicity, in this question, exactly 100 of the messages take 500 ms. You may ignore all overheads of configuring a request and checking a response (as it is negligible compared to the network communication time).

Part 1. Suppose you implemented your code so that a group of 10 requests is started and the next group begins only when all requests from the current group are complete.

1. Describe what the worst-case scenario is for this strategy, and calculate the total time to completion under that scenario. (2 marks)

2. Describe what the best-case scenario is for this strategy, and calculate the total time to completion under that scenario. (2 marks)

Part 2. Instead of using the whole-group strategy, you use the strategy where as soon as one of the 10 requests is done, the next request starts immediately.

3. Describe what the worst-case scenario is for this strategy, and calculate the total time to completion under that scenario. (2 marks)

4. Describe what the best-case scenario is for this strategy, and calculate the total time to completion under that scenario. (2 marks)

Part 3. [1 mark] If the average completion time for both scenarios is halfway between the worst-case and the best-case scenario, calculate the expected speedup of the average in part two over the average in part one. Report your answer to two decimal places.

Please initial:

Please initial: