ECE 459 W16 Midterm Solutions

J. Zarnett

February 20, 2016

(1)

```
#include <stdlib.h>
#include <pthread.h>

#define LENGTH (1024*1024)
int *vector;

void function( void* arg ) {
   int* startIndex = (int*) arg;
    for ( int i = 0; i < 1000; ++i) {
        setup (&vector[*startIndex], LENGTH/4);
    }
    free( arg );
    pthread_exit( 0 );
}


void setup(int *vector, int length) {
   for (int i = 0; i < length; i++) {
        vector[i] += 1;
    }
}

int main() {
    pthread_t threads[4];

    vector = (int*) calloc(LENGTH, sizeof( int ));

    for ( int i = 0; i < 4; ++i) {
        int * startIndex = malloc( sizeof ( int ) );
        *startIndex = i * (LENGTH/4);
        pthread_create(&threads[i], NULL, function, startIndex);
    }

    for ( int j = 0; j < 4; ++j) {
        pthread_join( threads[j], NULL );
    }
    pthread_exit( NULL );
}
```

Probably a lot of people forgot the `&` in `setup (&vector[*startIndex], LENGTH/4);`. Even I did when I wrote this solution. Whoops.

If you were a smart aleck and put `vector[i] = 1000;` instead of the for loop running a thousand times, I'll allow full marks for that. Because I did say you could modify the code however you wanted.

(2A)

The variables `pid` and `total` variables need to be scoped as PRIVATE. By default, these are scoped as SHARED. These variables need to be unique for each thread.

(2B) Subtraction is not associative. The additions $A + B$ and $B + A$ will always produce the same result, but $A - B$ and $B - A$ may produce different answers.

(3)

(A) Yes - this is okay if both threads are trying to write the same thing and the value won't change after being "set". In assignment 1, there was an array indicating whether a particular piece of the image had been received. If so, a thread changed the value from 0 to 1; if two threads are trying to write 1 at the same time that's okay, and because the value won't change once it's set to 1, the race condition does not cause a program error.

(B) Spinlocks are a good idea when attempting to acquire a locked we expect that the amount of time waiting for the lock is less than two process switches (getting blocked and then later unblocked). Perhaps we might also take into account the cache misses that also happen when we swap processes. But this must be on a multicore/multi-CPU system.

(C) The calculation is CPU intensive, so if there are 8 cores, each thread runs on one core. If there are more threads than cores, there will be swapping of threads in and out of the cores, so time is lost to swapping and cache misses when a different thread starts running on the core.

(D) The `volatile` qualifier is used to tell the compiler not to optimize a variable away in the event of something like an interrupt handler, and to indicate to the compiler not to store a variable in a register (if it can help it). But it provides no mechanisms for synchronization between variables (so race conditions still happen) and it does not prevent re-ordering.

(E) Spawning multiple threads and then collecting/combining results has some overhead cost. If the loop in question runs quickly enough or does not do "enough work", then the overhead cost of setup and cleanup of the additional threads outweighs the benefits of executing the loop in parallel. If the compiler believes that to be the case, it will refuse to parallelize the loop and say it is not profitable.

(4)

Step	CPU 0		CPU 1	
	x	y	x	y
0.	I	I	I	I
1.	E	I	I	I
2.	S	I	S	I
3.	S	M	S	I
4.	I	M	M	I
5.	I	S	M	S