ECE 459 Midterm Exam Solutions

J. Zarnett

January 31, 2017

(1)

(a) The important thing is that `a` and `b` should be pointing at the same memory location.

```
int x = 5;
swap( &x, &x );
```

is sufficient. If you want a more interesting solution:

```
int* x = malloc( sizeof( int ) );
*x = 5;
int* y = x;
swap( x, y );
```

(b) A loop carried dependency is one where the next iteration of the loop depends on the previous:

```
for( int i = 1; i < array_length; i++) {
    array[i] = array[i - 1] + 42;
}
```

(c) Set up the equation: $16 = h \times 9 + (1 - h) \times 300$

Solve for $h \rightarrow 0.97595$. Round to two decimal places: 0.98 (or 98%).

(d) Bank branch analogy is my favourite! Latency refers to the amount of time it takes for any individual customer to complete his/her desired transaction(s) at the branch location. Bandwidth is the total number of customers who can complete their transaction(s) during the day.

(e) Sometimes! If neither function modifies `x` or otherwise has side effects that would interfere with the other one, then they can run in parallel.

(2)

```
void search( int search_value );

int main( int argc, char** argv ) {
    int search_val;

    if ( argc < 2 ) {
        printf("A search value is required.\n");
        return -1;
    }
    search_val = atoi( argv[1] );

    search( search_val );
}

void search( int search_value ) {
     #pragma omp parallel for
     for ( int i = 0; i < ARRAY_SIZE; ++i) {
        if ( array[i] == search_value ) {
            printf("Found at %d by thread %d\n", i, omp_get_thread_num());
        }
    }
}
```

(3)

Memory Leaks

1. The variable `count` is allocated on line 34 and used as return value of the pthread. The value is read on line 70 under the name `thread_sum`. This is **definitely lost** and should be deallocated immediately after line 70 (inside the loop).

2. The variable `start` is allocated on line 61 and given in as a parameter to the pthread. There it is used under the name `start`. This memory is **definitely lost** and should be deallocated immediately before line 43 (after the for loop but before the call to `pthread_exit`).

3. The array `results` is is allocated on line 52 but is never deallocated. At the exit of the program it is **still reachable** and should be deallocated at line 80.

Concurrency Problems

1. `index` is a shared variable and it is modified by multiple threads on line 40. This could lead to an incorrect update of this variable where a write is lost (and then data in the results array might be overwritten!). To fix: either make this variable atomic or use a mutex to protect it.

(4)

```
#define NUM_ELVES 25
pthread_mutex_t cook_mutex;
pthread_mutex_t supplies_mutex;
sem_t done;
bool* elves;

void* elf( void* arg ) {
    int *id = (int*) arg;
    bool cooked = false;
    bool supplies = false;

    while (!cooked || !supplies) {
        if ( !cooked && 0 == pthread_mutex_trylock( &cook_mutex ) ) {
            cook_feast( *id );
            pthread_mutex_unlock( &cook_mutex );
            cooked = true;
        }
        if ( !supplies && 0 == pthread_mutex_trylock( &supplies_mutex ) ) {
            get_supplies( *id );
            pthread_mutex_unlock( &supplies_mutex );
            supplies = true;
        }
        clean_school( );
    }
    elves[*id] = true;
    sem_post( &done );
    free( id );
    pthread_exit( NULL );
}

int main( int argc, char** argv ) {
    int finished = 0;
    pthread_mutex_init( &cook_mutex, NULL );
    pthread_mutex_init( &supplies_mutex, NULL );
    sem_init ( &done, 0, 0 );
    elves = calloc( NUM_ELVES,  sizeof( bool ) );
    pthread_t threads[NUM_ELVES];

    for (int i = 0; i < NUM_ELVES; ++i) {
        int* id = malloc( sizeof (int) );
        *id = i;
        pthread_create(&threads[i], NULL, elf, id);
    }

    while( finished < NUM_ELVES ) {
        sem_wait( &done );
        int index;
        for (int j = 0; j < NUM_ELVES; j++) {
            if (elves[j] == 1) {
                elves[j] = 0;
                index = j;
                break;
            }
        }
        pthread_join(threads[index], NULL);
        printf("You are dismissed, elf %d\n", index);
        finished++;
    }

    pthread_mutex_destroy( &cook_mutex );
    pthread_mutex_destroy( &supplies_mutex );
    sem_destroy ( &done );
    free( elves );
    return 0;
}
```