UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFR11023 PARALLEL PROGRAMMING LANGUAGES AND SYSTEMS (LEVEL 11)

Wednesday 11 th May 2016

09:30 to 11:30

INSTRUCTIONS TO CANDIDATES

Answer any TWO questions.

All questions carry equal weight.

CALCULATORS MAY NOT BE USED IN THIS EXAMINATION

Year 4 Courses

Convener: I. Stark
External Examiners: A. Burns, A. Cohn, P. Healey, T. Field, T. Norman

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

1. (a) Describe the characteristics of the *bag of tasks* pattern, and the issues involved in deciding whether a bag of tasks algorithm has terminated. [*5 marks*]

   (b) The following code, taken from the course lecture notes, is a Pthreads monitor implementation of a re-usable counter barrier.

```
// Declare and initialize
pthread_mutex_t barrier;
pthread_cond_t go;
pthread_mutex_init(&barrier, NULL);
pthread_cond_init(&go, NULL);
int numArrived = 0;

void Barrier() {
  pthread_mutex_lock(&barrier);
  numArrived++;
  if (numArrived == numWorkers) {
    numArrived = 0;
    pthread_cond_broadcast(&go);
  } else {
    pthread_cond_wait(&go, &barrier);
  }
  pthread_mutex_unlock(&barrier);
}
```

   Explain in general terms what is meant by a *monitor* in parallel programming, and explain in specific terms how monitors are supported by the Pthreads library. You can use the code above for examples as appropriate. Both parts of your explanation (the general concept and the Pthreads specifics) should focus on concepts rather than syntax. [*10 marks*]

   (c) Explain how you would implement a monitor to support the bag of tasks pattern. Your design should focus on concurrency control issues and termination detection, using appropriate monitor mechanisms. You can assume the existence of a simple `Collection` data type, with unlimited capacity, which supports natural operations such as `add_item()`, `remove_item()`, `is_empty()`, and any others as you see fit, but which has no concept of concurrency control itself. Your answer can include discussion, simple diagrams and pseudocode as you choose. You are not expected to write complete, syntactically correct code. You may use Pthread style operations, or any other reasonable notation as long as its relationship to standard monitor concepts is clear. [*10 marks*]

2. (a) Explain why the following attempt to solve the critical section problem is broken.

```
boolean lock = false;

co [i = 1 to n] {
   while (something) {
     while (lock) ;
     lock = true;
     critical section;
     lock = false;
     non-critical section;
   }
}
```

[*4 marks*]

(b) Suppose a computer has an atomic decrement (DEC) instruction that also returns a value indicating whether the result is negative. DEC's effect is described by the following pseudocode which you should interpret as directly manipulating the contents of the variable `var`.

```
bool DEC (int var) =
< var = var-1;
  if (var>=0) return false; else return true; >
```

Using DEC, develop a pseudocode solution to the critical section problem for `n` processes. Don't worry about the "eventual entry" property. Try to make your solution as cache efficient as possible (in the sense of minimising coherence traffic). Describe clearly how your solution works and why it is correct and efficient. [*8 marks*]

(c) Discuss the similarties and differences beween the DEC operation described in part (b) and the semaphore operation P(). [*4 marks*]

QUESTION CONTINUES ON NEXT PAGE

(d) Consider the implementation of a synchronous buffer which allows the passing of data items from a group of producers to a group of consumers. The buffer should contain space for a single data item. The producer-consumer relationship must be *synchronous* in the sense that a producer, having deposited an item in the buffer, must wait until the item has been consumed before proceeding. Producers will attempt to produce, and consumers will attempt to consume, in unpredictable order. Each produced item must be consumed exactly once, but this can be by any consumer. An outline of the overall usage is as follows:

```
T buf;
### DECLARE SEMPAHORES HERE
co
  co [i = 1 to M] {
    while (whatever) {
      ...produce new data locally
      ### ADD CODE HERE
      buf = data;
      ### ADD CODE HERE
    }
  }
//
  co [j = 1 to N]  {
    while (whatever) {
      ### ADD CODE HERE
      result = buf;
      ### ADD CODE HERE
      ... process result locally
     }
  }
```

You must solve this synchronous buffering problem using only semaphores to enforce concurrency control.

Explain how you would add declarations of appropriate semaphores, and operations on these, in the positions indicated in the pseudocode. Include a brief explanation of your strategy. Keep your solution as simple as possible. [*9 marks*]

3. (a) The following code is extracted from the course notes on programming the Game of Life with Threading Building Blocks (TBB).

```
void NextGen(Grid oldMap, Grid newMap) {

   parallel_for (blocked_range<int>(1, maxrow+1),  // Range
                 CompNextGen(oldMap, newMap),      // Body
                 auto_partitioner());              // Partitioner

}
```

Explain the role and properties of each argument to this form of TBB's `parallel_for` template. Your explanation should discuss each concept in general terms, rather than the specifics of Game of Life, though you may also refer to the code above where this is helpful. You should explain clearly how issues of granularity are handled within a `parallel_for`. [*12 marks*]

(b) The naive sequential Fibonacci function from lectures was as follows.

```
int fib(int n) {
   if (n < 2) return n;
   else return fib(n-1)+fib(n-2);
}
```

Explain how you would use this algorithm to develop a TBB `parallel_reduce` solution to compute Fibonacci numbers, still using the naive approach. You should explain clearly how you would use TBB concepts to organize parallelism and to address issues of load balancing and granularity. You may find it helpful to write small code fragments for illustration, but a complete syntactically correct program is not required.

(Note: this question is about TBB concepts, so the fact that Fibonacci can be computed much more efficiently by other iterative means is not relevant, and should not be addressed). [*13 marks*]