Verónica Gaspes
School of IDE

Embedded Systems Programming

Written Exam

August 20, 2011, from 09.00 to 13.00

- **Allowed tools:** An English dictionary.
- **Grading criteria:** You can get at most 20 points.

  To pass you need at least 50% of the points.

  For the highest grade you need more than 90% of the points.
- **Responsible:** Verónica Gaspes. Telephone number 7380.

- **Read carefully!** Some exercises might include explanations, hints and/or some code. What you have to do in each exercise is marked with the points that you can get for solving it (as **(X pts.)**).
- **Write clearly!**
- **Motivate your answers!**

Good Luck!

1. **(2 pts.)** Write functions

   ```
   int areSet(unsigned int *port, unsigned int mask)
   int areClear(unsigned int *port, unsigned int mask)
   ```

   to test whether some bits in a port are set respectively clear . The bits that have to be tested are the bits in the second argument (`mask`) that are set.

2. One of the characteristics of most embedded programs is that they have to monitor and control devices in the environment of the processor. In the labs of the course, for example, the programs had to control an LCD while monitoring the joystick while controlling a piezoelement. Typically these devices evolve in parallel. In the labs the programs had to display prime numbers while a melody was playing in response to input from the joystick.

   (4 pts.) Discuss how such programs can be structured when you do **not** have access to a kernel that supports concurrency. Explain some of the problems that might arise and how to handle them.

3. **(3 pts.)** In the following fragment from laboration 2 there is a critical section. Identify it, explain what may happen, give an example of incorrect behaviour and solve the problem using a mutex.

   ```
   #include "tinythreads.h"

   int pp;

   void writeChar(char ch, int pos); // defined elsewhere
   int is_prime(long i);  // defined elsewhere

   void printAt(long num, int pos) {
     pp = pos;
     writeChar( (num % 100) / 10 + '0', pp);
     pp++;
     writeChar( num % 10 + '0', pp);
   }

   void computePrimes(int pos) {
      long n;
      for(n = 1; ; n++) {
          if (is_prime(n)) printAt(n, pos);
       }
   }
   ```

   see next page!

```
int main() {
  spawn(computePrimes, 0);
  computePrimes(3);
}
```

4. Using Tinytimber you can organize programs with *reactive objects* while programming in C. As a programmer you have to follow some conventions and Tinytimber guarantees that the methods of a reactive object are executed strictly sequentialy, thus protecting the local state of the object from critical section problems.

   (4 pts.) Program a class for reactive objects that can be used to *protect* (or encapsulate) an array of integers. The array to be encapsulated can be provided on object initialization. Let the type introduced for the class be `Array`. Then, the methods that have to be provided are

```
// record a position
int setPosition(Array *self, int x)

// set the value at the recorded position to x
int setElement(Array *self, int x)

// return the value at the recorded position
int getElement(Array *self, int x)
```

5. In laboration 4 you used a reactive object to implement a *blinker* that turns on and off a LED. The blinker can be started, stopped and the period can be set.

   (a) **(3 pts.)** Assuming that you have a class for reactive LEDs that can be turned on and off, implement a blinker. The blinker should not be hard-coupled to a LED, instead, it should be possible to instantiate blinkers for different LEDs.

   (b) **(1 pts.)** Explain the behaviour of your program if one of your blinkers is asked to start once it is already blinking. Here is a fragment of an application doing this:

   ```
   ASYNC(&blinker,start,NULL);
   AFTER(MSEC(100),&blinker,start,NULL);
   ```

6. (a) **(1 pts.)** What is a *utilization based schedulability test*?

   (b) **(1 pts.)** In the course we discussed utilization based schedulability tests for tasks sets with certain restrictions. What were these restrictions?

   (c) **(1 pts.)** For sets of tasks with the restrictions we considered, is there any reason to prefer Earliest Deadline First priority assignment to Rate Monotonic priority assignment?