CS 511: Homework Assignment 4*
Due: November 11, 11:55pm

1 Assignment Objectives

Get acquainted with some aspects of

- concurrent Erlang

2 Assignment Policies

Collaboration Policy. This homework may be done in individually or in pairs. Use of the Internet is allowed, but should not include searching for existing solutions. In case it is done in pairs, only one of the members should submit through Canvas. All submitted modules should include the attribute `-author("Names of authors").`

Under absolutely no circumstances code can be exchanged between students. Excerpts of code presented in class can be used.

Assignments from previous offerings of the course must not be reused. Violations will be penalized appropriately.

Late Policy. Late submissions are allowed with a penalty of 2 points per hour past the deadline.

3 Assignment

Write Erlang code to model a group of sensors. One Erlang module, in source file `sensor.erl`, should model a sensor process. Another Erlang module, in source file `watcher.erl`, should model "watcher" processes. Each watcher process should keep watch on as many as 10 sensors. Occasionally each sensor will report a measurement to its watcher. The measurement is a random integer in

*Based on Dan Duchamp's assignment 5, from prior editions of this course.

the range 1-10. Sensors should report measurements to their watcher every few seconds, the interval between successive measurements being a random number of milliseconds in the range 1-10000.

It should be possible to create and watch an arbitrary number of sensors. Every once in a while, a sensor will crash. When a sensor crashes, its watcher process should detect that fact and start a replacement sensor process.

Note that this is potentially a very large system, operation is completely asynchronous, and it includes automatic failure recovery, yet you should be able to program it around 100 lines of Erlang code.

(Once you have your logic working, try running with a very large number of processes. If you have written your "loops" with proper tail recursion, it should work. For instance, the example solution runs correctly with 300,000 simultaneous sensor processes on a standard modern desktop. To run with a large number of sensors, you might have to expand Erlang's process limit. To increase the cap, start Erlang like this: `erl +P NUMPROC` where `NUMPROC` is a power of two that specifies the process limit. Erlang creates many additional background processes so `NUMPROC` will have to be substantially larger than the number of sensors.)

3.1 Software Details

Sensor:

- Each sensor has an integer ID: 0, 1, etc.
- A measurement report should be a tuple that includes only the sensor's ID number and the measurement number.
- Generate a random measurement like this:

  ```
  Measurement = rand:uniform(11)
  ```

  If `Measurement` is in the range 1-10, report it to the watcher process. If `Measurement` is 11, crash with the report "`anomalous_reading`"
- Sleep for a random time between measurements like this:

  ```
  Sleep_time = rand:uniform(10000) ,
  timer:sleep(Sleep_time)
  ```

- To crash, call `exit/1`, as seen in class and described at `http://erlang.org/doc/man/erlang.html`

Watcher:

- Each sensor is watched by a single watcher process. For example, if there are 93 sensors then one watcher should watch sensors 0-9, a second should watch sensors 10-19, and so on; the tenth watcher should watch sensors 90-92.

- To start a new process:

  ```
  Pid = spawn(module, function, argument list)
  ```

  To start a new process and have the caller act as its monitor:

  ```
  { Pid, Ref } = spawn_monitor(module, function, argument list)
  ```

  Here, "`Ref`" is a unique "reference" that identifies the monitored process (but `Ref` is different from the Pid). We do not need references in this assignment; you can throw away the returned reference like this:

  ```
  { Pid, _ } = spawn_monitor(module, function, argument list)
  ```

- Each watcher should maintain a list of its sensors. Each item on the list should be a tuple that associates a sensor's ID number with its current Pid. You will have to update this list when a particular sensor is restarted because the new sensor process will have the same sensor number but a different Pid from the crashed sensor process. Read about the `lists:delete` function at `http://erlang.org/doc/man/lists.html`.

- Each watcher should print a line of output (using `io:fwrite`) whenever any of these events occurs:

  - Watcher starts: print initial list of sensors; each list item is a tuple that contains ID and Pid
  - Receives sensor reading: print sensor number and measurement number
  - Detects that sensor died: print sensor number (not Pid) and reason it died
  - Sensor is restarted: print updated list of sensors; each list item is a tuple that contains ID and Pid

4 Submission Instructions

Submit your Erlang source code via Canvas by the indicated date and time. Submit a zip file named `Assignment4_<Surnames>.zip` (where `<Surnames>` should be replaced by the surnames of the team members) containing all the files.