M.Sc. thesis project:
Practical Application of Systematic Concurrency Testing

Background Thorough verification and testing of concurrent program code is well-known to be difficult. Consider, for instance, a library for synchronization between threads, or for providing access to some shared data by many threads. When setting up a thorough testing procedure for such a library, one must consider all the different ways in which the threads in the library can interact. In previous research (much of it in UPMARC), we have developed techniques for addressing this problem by means of a technique called *systematic concurrency testing* (SCT). The idea is to systematically explore the set of possible thread schedulings (which determine how the threads interact). A special run-time scheduler drives the program execution, making decisions on scheduling whenever such decisions may affect the interaction between processes. A main problem is that the number of different thread schedulings grows very quickly (exponentially) with the length of the test executions. Therefore, clever techniques have been developed for reducing the number of executions that need to be explored, while still retaining coverage of all possible executions. At UPMARC, we have implemented these techniques in the tools Concuerror `http://concuerror.com/` and Nidhugg `https://github.com/nidhugg/nidhugg`, which are currently among the most efficient tools for performing systematic concurrency testing of programs written in Erlang and in C. (see also [1, 2, 3])

Goal The goal of this M.Sc. project is to use the Nidhugg tool to systematically test a nontrivial widely-used synchronization library, which is known to be nontrivial to test thoroughly. Our initial suggestion would be to use the *read-copy-update* library `http://liburcu.org/`, `https://en.wikipedia.org/wiki/Read-copy-update`. The RCU library provides efficient way to access shared data by many threads, in particular for data with many readers and few writers. The RCU library is heavily used in the Linux kernel, and a suitable subset of its code would be a suitable target for the project. In a previous M.Sc. thesis (available from us, and soon also on the Internet), Nidhugg has been used to test a subset of RCU for specific scenarios. The purpose of this M.Sc. project is to extend this work, applying Nidhugg to other functionalities in RCU and for different scenarios (one possibility would be to extend the previous work to the preemptible version).

Project The project consists in developing and executing a thorough test-suite for an interesting part and scenario in the RCU library. The task of the project could, as a suggestion, be broken down into:

- Understanding the RCU library and its function
- Developing a specification of the functionality to be tested (can be either informal or formal, but should be as precise and complete as possible)
- Designing a test suite that covers the envisaged uses of the library, and and argument why it is complete
- Using Nidhugg to execute the test suite, observing the results,
- If time permits, the test suite can be executed under different memory consistency models (SC, TSO).

Supervisors and Contact persons Bengt Jonsson (`bengt@it.uu.se`) and Kostis Sagonas (`kostis@it.uu.se`).

Bibliography

[1] Abdulla, P., Aronis, S., Faouzi Atig, M., Jonsson, B., Leonardsson, C., Sagonas, K.: Stateless model checking for TSO and PSO. In: TACAS, LNCS, vol. 9035, pp. 353–367. Springer (2015). Extended version available at `https://arxiv.org/abs/1501.02069`

[2] Abdulla, P., Aronis, S., Jonsson, B., Sagonas, K.: Optimal dynamic partial order reduction. In: POPL. pp. 373–384. ACM (2014)

[3] Christakis, M., Gotovos, A., Sagonas, K.: Systematic testing for detecting concurrency errors in Erlang programs. In: ICST. pp. 154–163. IEEE (2013)