L4 Project Progress Report

2079884F - W. David Fröhlingsdorf

Due by Friday the 16th of December 2016

1 Project Description

The project is based on a self-proposed topic and explores a novel idea for a locking protocol which shall be called *Inheritance Locking* hereinafter. Inheritance locks successfully prevent system deadlocks as well as starvation, which has been proven to be an issue in most mutual exclusive locking protocols.

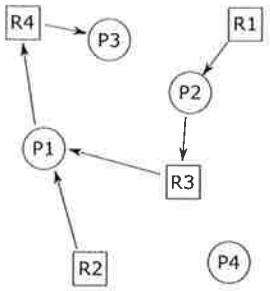
Here is a brief description of the protocol:

Lets consider the resource allocation graph (sometimes called wait-for graph) as introduced by Richard C. Holt in 1972. In particular, each process and resource is represented as a single node in a directed bipartite graph. Each node has at most one outgoing edge to one of its counterparts. That is, a resource node might have an outgoing edge to a process node which means that this resource is allocated to the according process. Similarly, a process might have an outgoing edge to a resource node which means that the process is waiting for the corresponding resource to become available. Note that in modern terms process would usually mean thread and resource would be the equivalent to a lock. However, for consistency we shall stick to the terms process and resource when considering the resource allocation graph. Furthermore, classically it is assumed that each resource might have more than one instance. However, this is rarely true in single mutex environments and for simplicity this point is therefore omitted at this stage. The novel idea of the Inheritance Locking protocol is to introduce a new mechanism for locking (that is allocating) and unlocking of a resource. In particular, if a process tries to lock a resource which is already allocated to another process, it will check the sink of the graph's component in which the resource resides in. If the sink happens to be the requesting process, the request will be successful as the allocated process is waiting for the requesting process. This means a requesting process might temporarily inherit a resource from a waiting process. In particular, while a process is waiting for a resource, resources already allocated to this process might be pre-empted.

Lets consider a specific example now. Figure 1 shows a resource allocation graph at an arbitrary point of time during execution. P1 and P2 are currently blocked as they are waiting for resource to become available while holding a number of resources themselves. P3 might successfully request R1, R2 or R3 as it is the sink of these graph nodes. On the other hand, P4 does not hold any

resource currently and would not be successful in requesting a resource as all resources are currently allocated and P4 is not the sink of any resource nodes. Therefore, if P4 tries to allocate a resource, it would become blocked until this resource becomes available.

Figure 1: Resource allocation graph



2 Progress

Up until now I have done an extensive literature review on the topic and have implemented an Inheritance Lock library for C/C++ using Pthreads. Moreover, I have written a micro-benchmark test suite for any generic locking protocol which checks

- The speed of locking and unlocking N locks with a single thread
- The speed of locking and unlocking a single lock with a single thread M times
- The capability of handling a deadlock situation
- The capability of handling a starvation situation

Moreover, I have successfully written a model for the SPIN-Model checker to verify that the Inheritance Locking protocol does not suffer from deadlocks nor from starvation, but ensures mutual exclusion.

3 Plan

The further plan of action is to improve the benchmark tests potentially using a genetic programming approach as well as refining the SPIN model. Furthermore, it would be very desirable to run the Inheritance Locking protocol on a number of standard micro and macro-benchmark tests. The next couple of months are therefore merely designated to benchmark testing and evaluation of the protocol.

4 Problems

The protocol itself contains some limitations and smaller issues which shall honestly be discussed in the dissertation.

However, the biggest problem so far has been that it turned out to be more difficult than expected to compare different protocols and concurrent programming approaches (transactional memory for example) to one another. This is partially due to the fundamentally different paradigms underlying the various protocols and approaches. Moreover, I found it often difficult to find usable open source code with good documentation on the internet. Last but not least, we found it problematic to decide whether work on Inheritance Locks (or similar) has been done before. During my literature review I could not find anything closely similar to it, however, there is still a considerable risk that the topic has been explored previously.