L4 Project Progress Report

2079884F - W. David Fröhlingsdorf

Due by Friday the 16$^{\text{th}}$ of December 2016

1 Project Description

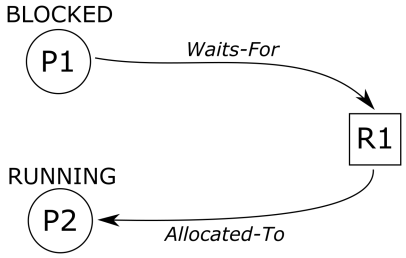
The project is based on a self-proposed topic and explores a novel idea for a locking protocol which shall be called *Inheritance Locking* hereinafter. Inheritance locks prevent system deadlocks as well as starvation, which has been proven to be an issue in most mutual exclusive locking protocols.

Here is a brief description of the protocol:

Let us consider the resource allocation graph (sometimes called wait-for graph) as introduced by Richard Holt [2]. In particular, each process and each resource is represented as a node in a directed bipartite graph. Each node has an out-degree (that is, the sum of outgoing edges, denoted by deg^+) of zero or one. In particular, 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 blocked while waiting for the corresponding resource to become available. Furthermore, a resource has to be allocated to a process if a requesting process is waiting for the resource to become available. An explanatory diagram of this graph type is shown in Figure 1. Formally:

$$
\begin{aligned}
G &= (N, E) \\
N &= (R \cup P) \\
E &\in (R \times P) \cup (P \times R) \\
deg^+(v) &= 0 \vee 1 \,|\, v \in N \\
deg^+(r) &= 1 \,|\, deg^-(r) > 0 \,, r \in R
\end{aligned}
$$

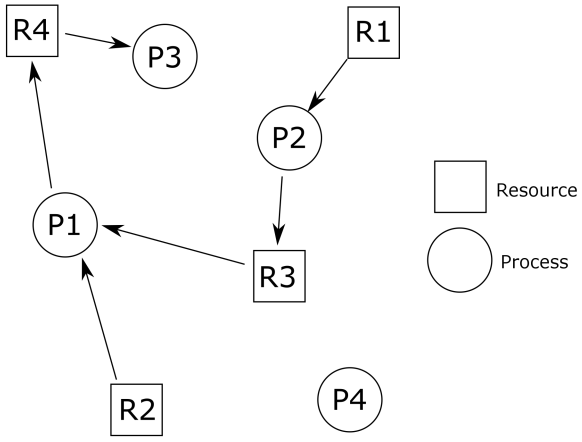
Figure 1: Resource allocation graph - Schema



Note that in modern terms process would usually mean thread and resource would be the equivalent to a lock. However, for consistency with Holt's paper 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 unit available. However, this assumption is not true in single-mutex locking protocols 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. A sink is a node with no outgoing edges (out-degree = 0). It is easy to see that each component of the graph has precisely one unique sink as each node has precisely one outgoing edge or is a sink. If the component's unique 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. Thus, the protocol prevents cycles in the graph which has been proven by Coffman to be one of the four necessary properties for a deadlock to occur [1].

Figure 2: Resource allocation graph - Example



Let's consider a specific example now. Figure 2 shows a resource allocation graph at an arbitrary point of time during execution. The processes P1 and P2 are currently blocked as they are waiting for a 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. Another way to view this circumstance is to consider reachability. There is no path from any of the resources to P4, hence P4 can not inherit any resource at this point of time. This means, if P4 tries to allocate a resource, it would become blocked until the resource becomes available.

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 using C preprocessor declaritives 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 generator program for the SPIN-Model checker which creates a model of the Inheritance Locking protocol with a parameterised number of threads and locks. Using LTL-properties, SPIN can then verify that the protocol does not suffer from deadlocks nor from starvation, but ensures mutual exclusion for the given number of threads and locks.

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 designated to thorough benchmark testing, evaluation and refinement of the protocol.

4 Problems

The protocol itself contains some limitations compared to other protocols which shall objectively discussed in the dissertation.

However, the biggest challenge so far has been to compare different protocols and concurrent programming approaches (transactional memory for example) to one another, which turned out to be more difficult than expected. This is partially due to the fundamentally different paradigms of 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, however, I came to the conclusion that to the best of my knowledge the topic has not been explored previously as recent overview papers do not mention the existence of such a protocol [3].

References

[1] Edward G Coffman, Melanie Elphick, and Arie Shoshani. System deadlocks. *ACM Computing Surveys (CSUR)*, 3(2):67–78, 1971.

[2] Richard C Holt. Some deadlock properties of computer systems. *ACM Computing Surveys (CSUR)*, 4(3):179–196, 1972.

[3] Gertrude Neuman Levine. The classification of deadlock prevention and avoidance is erroneous. *ACM SIGOPS Operating Systems Review*, 39(2):47–50, 2005.