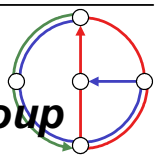
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing Group



HS 2009 Prof. Dr. Roger Wattenhofer, Thomas Locher, Remo Meier, Benjamin Sigg

Distributed Systems
Theory exercise 5

Assigned: December 4, 2009
Discussion: December 11, 2009

1 Spin Locks

A read-write lock is a lock that allows either multiple processes to read some resource, or one process to write some resource.

a) Write a simple read-write lock using only spinning, one shared integer and the CAS operation. Local variables are not allowed.

b) What is the problem with your lock?

Hint: what happens if a lot of processes access the lock repeatedly?

We now build a queue lock using only spinning, one local integer per process and the CAS operation.

c) To prepare for this task, answer the following questions:

i) Head and tail of the queue have to be stored in the shared integer. What is the "head" and the "tail", and how can they be stored in one integer?

Hint: could the head be a process id? Or is there a much easier solution?

ii) How could a process add itself to the queue?

Hint: you need the local integer of the process for this operation.

iii) When has a process acquired the lock?

iv) How does a process release the lock?

d) Write down the lock using pseudo-code. Do not forget to initialize all variables.

2 Bus and Caches

See slides 8/17, 8/20 and 8/29. We simulate some processes trying to acquire a lock. For this task use a dice or a coin to generate a sequence of random numbers $s_1 ... s_m$ in the range of 1 to 4. In the i'th round the process p_{s_i} can execute its next step. One step is long enough to access the state-variable exactly once.

a) Four processes perform Test&Set-locking. For a sequence of 10 to 20 rounds, write down what states the caches are in, and what data has to be moved around on the bus. In each round one randomly chosen process can execute one step. If a process acquires the lock, it releases the lock in its next step.

b) Repeat a) for Test&Test&Set-locking.

c) Explain why TTAS is faster than TAS using the results from a) and b).

3 DHT (Optional)

As we have discussed in the lecture, most DHTs are built on the same idea: a binary search tree. Each leaf of the tree is represented by a peer, nodes (including the root) do not really exist. A peer knows only a small subset of all the other existing peers. But if a peer knows the address of another peer it can always contact the other peer. The network is unreliable, messages can be lost, altered or arrive out of order.

Introduce new messages, protocols, restrictions or other ideas to protected a DHT from various Byzantine attacks. If you need to make assumptions, write them down. Each answer should be about 5-10 sentences.

a) Wrong lookup: A search for a key roughly requires $O(\log n)$ steps. In each step one new peer is included in the search. One way to search is to send a message through the DHT. The message contains the searched key and the address of the peer that started the search. The message is forwarded from one peer to the next such that it always gets a bit nearer to its (yet unknown) destination. Once the destination is reached, the final peer answers.

   A Byzantine peer sends the message either in the wrong direction, or to a non-existing peer.

b) Incorrect routing updates: Each peer maintains a routing table containing the addresses of about $\log n$ other peers. The peers send update messages to each other in order to keep their routing tables up to date.

   A Byzantine peer sends false updates, e.g, it tries to place dead links in a routing table.

c) Partitioning: If a peer wants to join a DHT it has to make contact with a peer that is already part of the DHT. The new peer can then ask the old peer about other nodes of the DHT and insert itself at an appropriate place.

   A Byzantine peer builds up his own private DHT by sending wrong messages to joining peers. It gives joining peers only the addresses of peers in his own isolated net.