ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

FS 2016 Prof. R. Wattenhofer

Computer Engineering II

Solution to Exercise Sheet 5

Quiz

1 Quiz

a) False. Unnamed pipes may be inherited to child processes of the process that created them, through file descriptor inheritance.

b) False. The file descriptors used to read from/write to a pipe may be shared by any number of processes. In case of concurrent writes/reads additional synchronization may be required.

c) False. Internally `counter++` is translated into at least 3 operations: a load operation from memory into a register, an addition on the register and a store from the register into memory. The scheduler may preempt the process at any point during the exection meaning that the 3 operations may be interleaved with operations from other processes, which may cause conflicts.

d) Race conditions are difficult to debug because...

- ...they are hard to reproduce since they depend on the specific schedule that the processor executed.
- ...they appear seemingly at random, again because of the dependency on the actual schedule. Two seemingly identical runs may not produce the same result.

e) Peterson's algorithm may not work on modern hardware because the memory accesses may be reordered to increase performance.

2 Atomic Tree

a) A simple example is the concurrent insertion of two leafs by two processes. Let's assume that we have a node a which is going to be the parent of the leafs b and b' which we want to insert. Both processes search for the position in which they'd like to insert their new leaf and the search ends at node a, both leafs would be added as a left child to a. Both processes read the node a and determine that there is no left child. The first process inserts b by directing a's left child pointer to b. The second process inserts b' by overwriting a's left child point to b', effectively undoing the first insert. We would have expected both b and b' to be inserted, but only one effectively is inserted due to the conflict.

b) By having a single lock that has to be acquired for all write operations we limit the concurrency of the list. Conceptually it would be acceptable to have multiple processes performing write operations at different, non-interfering, positions in the tree, e.g., one process could insert a node in one subtree, while another deletes a node in another subtree. With a single lock only one process can perform an operation at a time.

c) If we'd like to support transactions involving multiple write operations we will need to also lock for read operations in order to avoid reading an intermediate, inconsistent, state. A simple transaction could be moving a leaf node from one subtree to another subtree. This involves two write operations: removing the leaf node from the old subtree by overwriting its parent's child pointer and adding it to the other subtree by writing a pointer to the leaf in the new parent. If we were to read between the operations we would not be able to find the node neither in the old subtree nor in the new one.

d) Yes, the atomic swap ensures mutual exclusion. Since all read operations start from the root, they either traverse the old tree or the new tree. In both cases we do not read any intermediate state of a transaction. For writes we also work on our own copy, which guarantees a consistent view of the tree, and allows us to stage all changes before attempting to swap the root. The compare-and-swap on the root pointer allows us to atomically commit the staged changes to the datastructure, if no other transaction has changed it in the meantime. If the swap is unsuccessful, then the staged changes are discarded and we reattempt the transaction, hence there is no interference between writes.

e) The advantages are that we no longer limit the read-concurrency since all processes performing a read are guaranteed to work on a consistent snapshot of the datastructure. The disadvantages are that in high write contention we may be discarding a lot of transactions because another process committed while they were executing. This could be addressed by having an additional lock for write transactions such that only one transaction is executed concurrently. Another issue is that we create a partial copy of the datastructure for each transaction which needs to be garbage collected safely, when all processes have finished reading/writing to that copy.