HS 2016 Prof. R. Wattenhofer

Distributed Systems Part II

Exercise Sheet 10

Quiz

1 Quiz

a) When performing a concurrent update on a linked list using optimistic synchronization, how does one guarantee that no deadlocks occur?

b) What properties do (good) hash functions have? List as many as you can!

c) Under what circumstances is it acceptable to use the remainder function (mod x) as hash function for integer keys?

d) How could one improve the efficiency of finding an item within an already found bucket in a hash map? In what scenarios is this preferable to shrinking the bucket size by growing the hash map? Why do these scenarios only rarely occur?

e) How would you implement a hash map supporting inserting multiple values per key?

Advanced

2 Old Exam Question: Fine-Grained Locking

The goal of this exercise is to implement a heap with mutual exclusion. A heap is a binary tree, in which the value of the parent is smaller than the values of its children. The heap is stored in an array, with the root at index 1 and the children of a node i are $LEFT(i) = 2 \cdot i$ and $RIGHT(i) = 2 \cdot i + 1$. The basic functionality is implemented in Algorithm 1 and Algorithm 2.

Algorithm 1 Insert value

```
.....................
i = 1
.....................
while A[i] != null do
    ....................
    if A[i] > value then
        ...................
        exchange A[i] and value
        ...................
    end if
    ....................
    next = smallestChild(i)
    ....................
    i = next
    ....................
end while
.....................
A[i] = value
.....................
```

Algorithm 2 Remove smallest value

```
.....................
ret = A[1]
.....................
A[1] = ∞
.....................
i = 1
.....................
while A[i] != null do
    ....................
    next = smallestChild(i)
    ....................
    exchange A[i] and A[next]
    ....................
    i = next
    ....................
end while
.....................
A[i] = null // Mark as not used
.....................
return ret
```

a) (4 Points) How would you implement coarse-grained locking? What consequences does this have for concurrent access by multiple processes?

b) (8 Points) Complete the skeleton of the code in Algorithm 1 and Algorithm 2 to implement hand-over-hand locking. You may use *LOCK(j)* and *UNLOCK(j)*, which lock/unlock the jth element in the array. Not all lines are needed. You may use multiple statements per line.

c) (5 Points) Is your implementation deadlock free? Argue why deadlocks are not possible or provide an example of a deadlock.

d) (3 Points) When using hand-over-hand locking the root is always locked at the beginning of each operation. Could you use a different locking mechanism to avoid this contention of the root?