



Computer Engineering II

Exercise Sheet 8

Quiz

1 Quiz

- a) If requesting a lock is very expensive, how would you implement a linked list set?
- b) What properties do (good) hash functions have? List as many as you can!
- c) How would you implement a hash map supporting inserting multiple values per key?
- d) Which of the implementations for a list-based set is FIFO fair?

Basic

2 Livelock

In the lecture we discussed how to implement a Set using a linked list and the concept of optimistic synchronization. The main trick was to only lock affected parts of the list once a change should be applied. Are there bad situations in which the algorithm works badly?

- a) Is there a scenario in which two (or more) threads deadlock? If yes: give an example. If no: argue why.
- b) Is there a scenario in which one thread never succeeds in removing a node? If yes: give an example. If no: argue why.

3 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

```

1: i = 1
2: .....
3: while A[i] != null do
4:   .....
5:   next = smallestChild(i)
6:   .....
7:   if if(A[i] > value) then
8:     .....
9:     exchange A[i] and value
10:  .....
11:  end if
12:  .....
13:  i = next
14:  .....
15: end while
16: .....
17: A[i] = value
18: .....

```

Algorithm 2 Remove smallest value

```

1: .....
2: ret = A[1]
3: i=1
4: A[1] = ∞
5: .....
6: while A[i] != null do
7:   .....
8:   next = smallestChild(i)
9:   .....
10:  exchange A[i] and A[next]
11:  .....
12:  i = next
13: end while
14: A[i] = null // Mark as not used
15: .....
16: 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 j th 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?