



Computer Engineering II

Exercise Sheet Chapter 11

We categorize questions into four different categories:

Quiz Short questions which we will solve rather interactively at the start of the exercise sessions.

Basic Improve the basic understanding of the lecture material.

Advanced Test your ability to work with the lecture content. This is the typical style of questions which appear in the exam.

Mastery More interesting, but also more challenging. These questions are harder and we do not expect you to solve such exercises during the exam. Simpler questions about the same topics are however possible in the exam.

Questions marked with ^(g) may need some research on Google.

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 (A[i] > value) then
8:     exchange A[i] and value
9:   end if
10: .....
11:   i = next
12: .....
13: end while
14: .....
15: A[i] = value
16: .....
```

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:  if (A[next] != null) then
11:    exchange A[i] and A[next]
12:  else
13:    A[i] = null // Mark as not used
14:  end if
15: .....
16:   i = next
17: .....
18: end while
19: .....
20: 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?