Assignment 2: Synchronization, Satisfaction

ETH Zurich

1 Mutual Exclusion

1.1 Background

Consider the following algorithm for a process P_i in a set of processes $P_1, \ldots, P_n$:

```
turn := 0
∀ i ∈ {1, ..., n}: claimed[i] := false
```

P_i

```
    claimed[i] := true
2   while ∃ j ∈ {1, ..., n} \ {i} : claimed[j] = true loop
      claimed[i] := false
4     await (turn = 0 or turn = i)
      turn := i
6     claimed[i] := true
    end
8   critical section
    claimed[i] := false
10  turn := 0
    non-critical section
```

1.2 Task

Answer the following questions:

1. Does the algorithm enforce mutual exclusion? If so, justify your answer with an informal proof. If not, provide a sequence of actions to illustrate how mutual exclusion could be violated.

2. Does the algorithm guarantee the absence of deadlocks? If so, justify your answer with an informal proof. If not, provide a sequence of actions to illustrate how a deadlock could occur.

3. Does the algorithm guarantee the absence of starvation? If so, justify your answer with an informal proof. If not, provide a sequence of actions to illustrate how starvation could occur.

2 Yet Another Lock: Proofs

2.1 Background

This task is taken from *The Art of Multiprocessor Programming* [1]. Consider the following protocol to achieve n-thread mutual exclusion.

```
busy := false
P_i
    do {
2       do {
            turn := i
4       } while (busy)
        busy := true
6   } while (turn != i)
    critical section
8   busy := false
    non-critical section
```

2.2 Task

For each of the following questions either provide a proof, or display an execution where it fails.

1. Does the protocol satisfy mutual exclusion?
2. Is the protocol starvation-free?
3. Is the protocol deadlock-free?

3 Tree-based mutual exclusion

3.1 Background

This question assumes a "tree-based mutual exclusion" (TBME) algorithm which is based on the following idea: The algorithm can be represented by a binary tree where each internal (non-leaf) node represents a critical section shared by its descendants. The threads are at the leaves of the tree. The root of the tree is the main critical section shared by all the threads.

To enter the main critical section, a thread starts at its leaf in the tree. The thread is required to traverse the path from its leaf up to the root, entering all the critical sections on its path. Upon exiting the critical section, the thread traverses this path in reverse, this time leaving all the critical sections on its path. Figure 1 illustrates this process. If thread 1 wants to enter the main critical section, it must first enter critical section B. After having successfully entered critical section B, thread 1 must enter critical section A, and so on.

At each internal node, there is a maximum of two threads competing against each other to enter the node's critical section. Therefore, a mutual exclusion algorithm for two threads (e.g. Peterson's algorithm for 2 threads) can be used to implement the critical section of an internal node.

3.2 Task

1. What is the main advantage of the TBME algorithm over the Peterson algorithm for n threads?

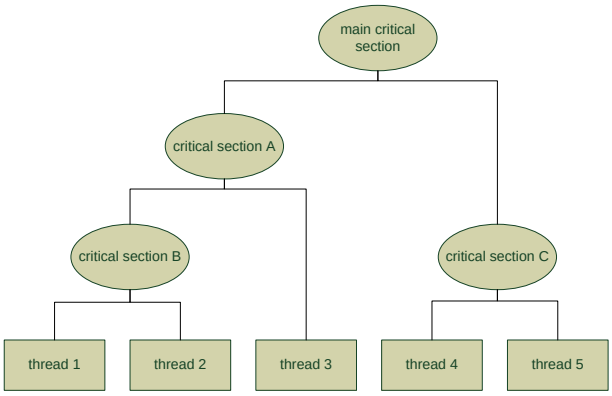

Figure 1: tree-based mutual exclusion algorithm example

2. Provide a Java implementation of the TBME algorithm using the Peterson algorithm for 2 threads.

References

[1] Maurice Herlihy und Nir Shavit: The Art of Multiprocessor Programming. Morgan Kaufmann, 2008