Class Assignment

-2018101116,Lakshmi Madhuri Y

Question1:- Explain why the fine-grained locking algorithm is not subject to deadlock.

Answer:-

- Usually,Deadlocks arrives when multiple locks are involved.
- Deadlock occurs when a process or thread enters a waiting state because a requested system resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process.
- In the case of **fine-grained locking,** this possibility is eliminated when the acquisition of the locks is always initiated from the sentinel to the final element of the list (tail).
- Another feature is the identity of the elements based on a single hash distributed in ascending order until the end of the list.
- In **Fine-grained locking,**Rather than locking the whole linked list at once, we add a lock to each node. Then, threads only lock the individual nodes on which they are operating.

```
public class FineSet extends SequentialSet {
// empty set
        public FineSet() {
                head = new LockableNode<>(Integer.MIN_VALUE); // smallest key
                tail = new LockableNode<>(Integer.MAX_VALUE); // largest key
                head.setNext(tail);
}
```

Each node includes a lock object, and lock and unlock methods that access the lock.

```
class LockableNode extends SequentialNode {
private Lock lock = new ReentrantLock();
void lock() { lock.lock(); } // lock node
 void unlock() { lock.unlock(); } // unlock node
}
```

Question2:-Explain why the optimistic and lazy locking algorithms are not subject to deadlock.

Answer:-

Optimistic Locking:-

- "Find without Locking".Problems may occur if the list is modified between when a threads finds a position and when it acquires locks on that position. Thus, we validate a position **after finding it and while the nodes are locked**, to verify that no interference took place
- This approach is optimistic because it works well when validation is often successful (so we don't have to repeat operations).
- Working-->
    - find the item's position inside the list without locking – as in SequentialSet
    - lock the position's nodes pred and curr
    - validate the position while the nodes are locked:
    - if the position is valid, perform the operation while the nodes are locked, then release locks
    - if the position is invalid, release locks and repeat the operation from scratch
    - Validate function:-

```
private boolean validate(Node pred, Node curr) {
    Node node = head;
    while (node.key <= pred.key) {
        if (node == pred)
            return pred.next == curr;
        node = node.next;
    }
    return false;
    }
```

- **Reason**:- threads operating on disjoint portions of the list can operate in parallel .When validation often succeeds, there is much less locking involved than in FineSet ,so less chances of Deadlock

Lazy Locking:-

Allowing for certain delays in the update process, i.e. replicas can be outdated for a certain time.It usually postpones Hardwork.For example , lets take removal of component-->

then it splits the function as 1)removing it logically by setting a tag bit (called as “valid“)

2)then later,removing it physically by unlinking it

To this end, each node includes a flag valid with setters and getters:

• valid() == true: the node is part of the set

• valid() == false: the node is being (or has been) removed

Reason for not having deadlock -->

“1.membership checking does not require any locking – it’s even wait free (it traverses the list once without locking

2.physical removal of logically removed nodes could be batched and performed when convenient – thus reducing the number of times the physical chain of nodes is changed, in turn reducing the expensive propagation of information between threads”