MODULE *cacheinvalidationv1*

EXTENDS *Naturals*

CONSTANTS
    $KEYS$

VARIABLES
    $database$,
    $cache$,
    $cacheFillStates$, $cacheFillStatus[key] =$ Fill state
    $invalidationQueue$

INSTANCE *cacherequirements*

$vars \triangleq \langle database, cache, cacheFillStates, invalidationQueue \rangle$

$InvalidationMessage \triangleq [key : KEYS]$

$CacheFillState \triangleq [state : \{\text{"inactive"}, \text{"started"}, \text{"respondedto"}\}, version : DataVersion]$

$CacheValue \triangleq CacheMiss \cup CacheHit$

$TypeOk \triangleq$
    $\land\ database \in [KEYS \to DataVersion]$
    $\land\ cache \in [KEYS \to CacheValue]$
    We track the cache fill state for each key. It is a multipart process
    $\land\ cacheFillStates \in [KEYS \to CacheFillState]$
    We model *invalidationQueue* as a set, because we cannot guarantee in-order delivery
    $\land\ invalidationQueue \in$ SUBSET $InvalidationMessage$

$Init \triangleq$
    $\land\ database = [k \in KEYS \mapsto 0]$
    $\land\ cache = [k \in KEYS \mapsto [type \mapsto \text{"miss"}]]$
    Cache fill states start inactive
    $\land\ cacheFillStates = [k \in KEYS \mapsto [$
                                        $state \mapsto \text{"inactive"}$,
                                         Version set to earliest possible version
                                        $version \mapsto 0]$
                                    $]$
    The invalidation queue starts empty
    $\land\ invalidationQueue = \{\}$

$DatabaseUpdate(k) \triangleq$
    The version of that key is incremented, representing a write
    $\land\ database' = [database$ EXCEPT

```
                    $![k] = database[k] + 1]$
        Adds invalidation message to queue.
        We don't need to model a delay in adding message as the cache can
        always delay handling message to similar effect.
        $\land$ $invalidationQueue' = invalidationQueue \cup \{[key \mapsto k]\}$
        $\land$ UNCHANGED $\langle cache, cacheFillStates \rangle$

  Cache Fill behavior
$CacheStartReadThroughFill(k) \triangleq$
        Read-through only occurs when the cache is unset for that value
        $\land$ $cache[k] \in CacheMiss$
        One cache fill request at a time
        $\land$ $cacheFillStates[k].state = $ "inactive"
        $\land$ $cacheFillStates' = [cacheFillStates$ EXCEPT $![k].state = $ "started"]
        $\land$ UNCHANGED $\langle database, cache, invalidationQueue \rangle$

  This is the moment the database provides a value for cache fill
$DatabaseRespondToCacheFill(k) \triangleq$
        $\land$ $cacheFillStates[k].state = $ "started"
        $\land$ $cacheFillStates' = [cacheFillStates$ EXCEPT
                          $![k].state = $ "respondedto",
                          $![k].version = database[k]$
                         ]
        $\land$ UNCHANGED $\langle database, cache, invalidationQueue \rangle$

   Cache incorporates the data
$CacheCompleteFill(k) \triangleq$
        $\land$ $cacheFillStates[k].state = $ "respondedto"
        $\land$ $cacheFillStates' = [cacheFillStates$ EXCEPT   Reset to 0
                          $![k].state = $ "inactive",
                          $![k].version = 0$
                         ]
        $\land$ $cache' = [cache$ EXCEPT
                       $![k] = [$
                          Cache value is now a hit
                         $type \mapsto$ "hit",
                          Set to whatever came back in response
                         $version \mapsto cacheFillStates[k].version$
                       ]
                  ]
        $\land$ UNCHANGED $\langle database, invalidationQueue \rangle$

  Cache fails to fill
$CacheFailFill(k) \triangleq$
        $\land$ $cacheFillStates[k].state = $ "respondedto"
          Cache fill state is reset, having not filled cache
```

```
    $\land$ $cacheFillStates' = [cacheFillStates$ EXCEPT
                                  $![k].state =$ "inactive",
                                  $![k].version = 0$
                                 ]
    $\land$ UNCHANGED $\langle database, cache, invalidationQueue \rangle$

 Handle invalidation message. Assume it is not taken off queue in case of
 failure. Therefore failure modeled as CacheHandleInvalidationMessage not
 occurring.
$CacheHandleInvalidationMessage \triangleq$
    $\land \exists message \in invalidationQueue :$  Dequeue invalidation queue in any order
         Remove message from queue
        $\land$ $invalidationQueue' = invalidationQueue \setminus \{message\}$
         Evict item from cache
        $\land$ $cache' = [cache$ EXCEPT $![message.key] = [type \mapsto$ "miss"$]]$
    $\land$ UNCHANGED $\langle cacheFillStates, database \rangle$

 Cache eviction model is unchanged
$CacheEvict(k) \triangleq$
    $\land$ $cache[k] \in CacheHit$
    $\land$ $cache' = [cache$ EXCEPT $![k] = [type \mapsto$ "miss"$]]$
    $\land$ UNCHANGED $\langle database, cacheFillStates, invalidationQueue \rangle$

 The cache will always be able to . . .
$CacheFairness \triangleq$
    $\exists k \in KEYS :$
         Complete the cache fill process
        $\lor$ $CacheStartReadThroughFill(k)$
        $\lor$ $DatabaseRespondToCacheFill(k)$  Write
        $\lor$ $CacheCompleteFill(k)$
         Process invalidation messages
        $\lor$ $CacheHandleInvalidationMessage$
```

Specification

```
$Next \triangleq$
    $\exists k \in KEYS :$
         Database states
        $\lor$ $DatabaseUpdate(k)$
         Cache states
        $\lor$ $CacheStartReadThroughFill(k)$
        $\lor$ $DatabaseRespondToCacheFill(k)$
        $\lor$ $CacheCompleteFill(k)$
        $\lor$ $CacheHandleInvalidationMessage$
        $\lor$ $CacheEvict(k)$
```

Cache fairness is included as part of the specification of system behavior.
This is just how the system works.

$Spec \triangleq Init \land \Box[Next]_{vars} \land \mathrm{WF}_{vars}(CacheFairness)$

\* Modification History
\* Last modified *Wed Jun* 15 12:45:25 *MST* 2022 by *elliotswart*
\* Created *Tue Jun* 14 20:36:02 *MST* 2022 by *elliotswart*