MODULE *GFS*

General *GFS* module. This module implements the general functioning of *GFS*, *w*/o focusing on write types. It specifies master functioning and adds some "time" functionality, but does not specify write/read-related "holes" in *PrimSecRep*.

```
EXTENDS MCPrimSecRep
```

```
CONSTANTS
        Chunk,        chunk IDs
        RepFactor     replication factor

gfsvars ≜ mcvars
```

Fill in some of the Holes in *PrimSecRep*:

```
GFS_ErrorTypes ≜ {"badver"}  error types at secondaries:
                             only one general type of error (or maybe none?)
```

```
GFS_TypeInvariant ≜
    ∧ TypeInvariant

GFS_Init ≜
    ∧ MCInit
```

Util functions

```
chunk_exists(c) ≜ master.objects[c].version > 0
```

```
 chunk_full(c) ≜ IF master.objects[c].prim = NoRep THEN FALSE
                 ELSE Len(data[master.objects[c].prim][c]) = ChunkSize
chunk_order ≜ ⟨FirstChunk, SecondChunk, ThirdChunk, FourthChunk, FifthChunk⟩
prev_chunk(c) ≜
  LET idx ≜ CHOOSE n ∈ 1 . . Len(chunk_order): chunk_order[n] = c IN  IF (idx - 1 = 0) THEN 0
     ELSE chunk_order[idx − 1]
```

MASTER functioning

Chunk creation. Masster creates a chunk if it doesn't already exist. The new chunk is replicated at a number of replicas. These replicas must be believed as alive by the master. No lease is given out yet. Normally, chunk creation should be only when needed, but now I do it anytime, for simplicity.

```
_CreateChunk(c, reps) ≜
   ∧ ∀ r ∈ reps :
        master.health[r] = "alive"  master thinks that all proposed reps are alive

    Set the reps as secondaries. No replica has lease for this chunk yet.
   ∧ master′ = [master EXCEPT !.objects[c].prim = NoRep,   no replica has lease yet
                              !.objects[c].sec = reps,
                              !.objects[c].version = 1]

   ∧ UNCHANGED ⟨data, cache, stat, channel, resps⟩
CreateChunk ≜
```

```
$\exists c \in Chunk :$
  $\land \neg chunk\_exists(c)$ new chunk does not exist

  $\land \lor prev\_chunk(c) = 0$
   $\lor \land prev\_chunk(c) \neq 0$
     $\land chunk\_exists(prev\_chunk(c))$ \* previous chunk already exists
     $\land chunk\_full(prev\_chunk(c))$ \* and is full
   $\land \exists reps \in$ SUBSET $(Rep) :$ Find a set of replica to store the new chunk
       $\land card(reps) = RepFactor$
       $\land \_CreateChunk(c, reps)$
```

The primary releases a new lease for a chunk that doesn't have a primary. This release only happens after an older lease expires (so this action is logically after *LeaseExpiry*.

```
$\_ReleaseLease(c, r) \triangleq$
    $\land master.objects[c].prim = NoRep$ No lease given for chunk $c$
    $\land r \in master.objects[c].sec$ replica $r$ is one of secondaries
    $\land master.health[r] =$ “alive” replica $r$ is believed to be alive by master (redundant)

     Set secondary $r$ to be the primary of chunk $c$ (so it will no longer be a *sec* for that chunk)
    $\land master' = [master$ EXCEPT $!.objects[c].sec = @ \setminus \{r\},$
                               $!.objects[c].prim = r,$
                               $!.objects[c].version = @ + 1]$ get to a new version

    $\land$ UNCHANGED $\langle data, stat, channel, cache, resps \rangle$
$ReleaseLease \triangleq$
   $\exists c \in Chunk :$
     $\land chunk\_exists(c)$ the chunk exists
     $\land master.objects[c].sec \neq \{\}$ there're replicas from where master can choose a new primary
     $\land \exists r \in Rep :$
         $\_ReleaseLease(c, r)$
```

Master detects a replica is dead. This might be the reality or not. Events such as network partitions can lead to the primary thinking a replcia is dead when it's alive.

```
$\_DetectDeath(r) \triangleq$
    $\land master.health[r] =$ “alive” master thinks replica $r$ used to be alive

     Update all the chunks that the replica stored.
     If $r$ was a primary in any of these, force lease expiry
     before the master can release the lease to someone else.
    $\land master' =$
       $[objects \mapsto [c \in Chunk \mapsto$
            IF $master.objects[c].prim = r$ THEN
               $master.objects[c]$ DO NOT revoke lease of primary, wait for it to expire
             ELSE  IF $r \in master.objects[c].sec$ THEN
               $[master.objects[c]$ EXCEPT $!.sec = @ \setminus \{r\}]$ invalidate a secondary
             ELSE  $master.objects[c]],$
        $health \mapsto [master.health$ EXCEPT $![r] =$ “dead”$]]$
    ∧ UNCHANGED ⟨data, cache, stat, channel, resps⟩
DetectDeath ≜
    ∧ ∃ r ∈ Rep : _DetectDeath(r)
```

Master actions: creates chunks, assigns replicas to them, releases leases after old leases have expired (or were revoked – not modeled); detects death of replicas and removes them.

```
GFS_MasterActions ≜
    ∨ CreateChunk ∨ ReleaseLease ∨ DetectDeath ∨ MasterActions
```

TIME AND CHANNELS: (sources of timeouts and losses) A lease expires suddenly, from both the master and the rep. Leases expire at the same time from everybody's cache. How do you ensure this in implementation?

```
_LeaseExpiration(c, r) ≜      lease of replica r for chunk c expires
    ∧ master.objects[c].prim = r     master knows r used to have the lease

    ∧ master' = [master EXCEPT !.objects[c].prim = NoRep,
                               !.objects[c].sec = IF master.health[r] = "alive"
                                                      THEN @ ∪ {r} ELSE @]
    ∧ cache' = [r1 ∈ Rep ↦ [cache[r1] EXCEPT
                    ![c] = IF r = @.prim THEN
                                [prim ↦ NoRep,
                                 sec ↦ IF stat[r].phase = "alive" THEN @.sec ∪ {r}
                                           ELSE @.sec,
                                 version ↦ @.version]
                           ELSE @]]
    ∧ UNCHANGED ⟨data, stat, channel, resps⟩
LeaseExpiration ≜
    ∧ ∃ c ∈ Chunk, r ∈ Rep : _LeaseExpiration(c, r)
```

Time actions: lease expiration and all the other actions from *PrimSecRep*.

```
GFS_TimeActions ≜
    ∨ LeaseExpiration ∨ TimeActions
```

REPLICA functioning

Replies to the client after the write is finished. If the write as committed at all replicas, then the write is considered successful. Otherwise failed. This is a hole from *PrimSecRep*.

```
GFS_ReplyFinishedWrite(r, c, w) ≜
    LET CommittedEverywhere ≜      the write has been committed to all replicas
                                   believed alive at the time the write was initiated
            ∀ r1 ∈ Rep : ∨ resps[r][c][r1] = "ok"     r1 responded ok
                         ∨ resps[r][c][r1] = "n/a"    OR r wasn't expecting a reply from r1
    IN
        IF CommittedEverywhere THEN
                Reply(r, "wrFinished", [ack ↦ "ok", object ↦ c, w ↦ w])
         ELSE
```

$$Reply(r, \text{“wrFinished”}, [ack \mapsto \text{“error”}, object \mapsto c, w \mapsto w])$$

SPECIFICATION

```
$GFS\_Next \triangleq$
        $\lor GFS\_MasterActions$
        $\lor GFS\_TimeActions$
        $\lor ReplicaActions$
        $\lor ClientActions$

$GFS\_Spec \triangleq \land GFS\_Init \land \Box[GFS\_Next]_{gfsvars}$
```

Invariants

DEBUG: Primary is unique. If a replica believes it's prim for a chunk, the master agrees with this.

```
$PrimIsFreshAndUniqueI \triangleq$
  $\land \forall c \in Chunk :$
      If any replica believes it's primary, then it's really primary
      $\land \forall r \in Rep : (cache[r][c].prim = r \Rightarrow master.objects[c].prim = r)$

      the primary knows it's primary
      $\land \lor master.objects[c].prim = NoRep$ \* either no primary
        $\lor stat[master.objects[c].prim].phase = \text{“dead”}$
        $\lor master.health[master.objects[c].prim] = \text{“dead”}$
        $\lor cache[master.objects[c].prim][c].prim = master.objects[c].prim$
            \* OR the prim knows it's prim
```

DEBUG: No replica that master believes as dead is a secondary. Note that the same does NOT hold for primaries!! There can be a primary that's known dead to master but that the master still has registered as prim until its lease expires.

```
$MasterCorrectnessI \triangleq$
  $\land \forall r \in Rep :$
    $(master.health[r] = \text{“dead”} \Rightarrow \neg(\exists c \in Chunk : r \in master.objects[c].sec))$
```

DEBUG: At no time is the primary a secondary.

```
$PrimIsNotSecI \triangleq$   a primary is not a secondary for the same chunk
  $\land \forall c \in Chunk :$
    $\lor \neg chunk\_exists(c)$
    $\lor master.objects[c].prim \notin master.objects[c].sec$

$GFS\_AllInvariants \triangleq$
  $\land GFS\_TypeInvariant$
  $\land PrimIsFreshAndUniqueI$
  $\land MasterCorrectnessI$
  $\land PrimIsNotSecI$
```

THEOREM $GFS\_Spec \Rightarrow \Box GFS\_AllInvariants$