Assignment 1

Reading - Chapter 1 of the textbook.
Project – Let us know who you'll partner with and whether you do the project in Java or C#. If you don't know Java or C#, start learning one of them now.

This problem is about how the ACID properties are affected by the internal structure and behavior of data management software. It also raises some design considerations that may affect your project. The code is simple, rather than useful or interesting, so as to make it easy to think and argue about the implementation of ACID properties.

BlockDB is a transactional block-oriented storage system that allows its users to issue transactions that access multiple blocks. A user of BlockDB identifies each block by its physical address on disk. BlockDB offers the following five operations. Underlined parameters are output.

- ```
  /* Start new transaction */
  int start()
  Returns: -1, if there is an active transaction
           tId < -1 (a new Transaction Identifier) otherwise
  ```

- ```
  /* If possible, read block at diskBlockAddr on behalf of transaction
  tId from disk into main memory and save a pointer to it in memBlock */
  int read(int diskBlockAddr, int tId, int *memBlock)
  Returns: 0 if successful, -1 otherwise
  ```

- ```
  /* Tell BlockDB that transaction tId has updated a block at
  diskBlockAddr */
  int write(int diskBlockAddr, int tId)
  Returns: 0 if there is no cached entry for diskBlockAddr
           1 otherwise
  ```

- ```
  /* Try to commit transaction tId */
  int commit(int tId)
  Returns: 0 if successful, -1 otherwise
  ```

- ```
  /* Abort transaction tId */
  int abort(int tId)
  Returns: 0
  ```

BlockDB is implemented as follows:

- BlockDB maintains a cache of blocks in main memory. It organizes this as a collection, called `Cache`, of cache entries. Each cache entry `e` is an object `Cache(e)` with three attributes:
  - `int Cache(e).tId`, which contains a transaction ID
  - `int Cache(e).diskBlockAddr`, which contains the identity (i.e. disk address) of the block stored in `Cache(e)`
  - `int Cache(e).oldBlock`, which contains the last committed content of the block
  - `int Cache(e).newBlock`, which contains the last written content of the block

  The cache is initially empty. This means that the cache has been allocated, and for all entries `e` in the cache, `Cache(e).tId = 0`.
  ```

- BlockDB maintains a variable `lastTransId`, which is the (sometimes negated) transaction identifier of the last transaction that was started. **It is initialized to 1**.
- There can be at most one instance of BlockDB running on a system at a given time. It has **exclusive access** to the disk drive that it uses.

BlockDB uses a raw disk drive, which supports two operations `diskRead` and `diskWrite`

- ```
  /* Read the value of block at diskBlockAddr from disk and return it in
  main memory at address memAddr */
  int diskRead(int diskBlockAddr, int memAddr)
  ```

- ```
  /* Store the content of the main memory beginning at address memAddr
  into block diskBlockAddr on disk */
  int diskWrite(int diskblockAddr, int memAddr)

  Returns:
  For each operation, read or write, the disk drive either performs the
  intended operation and returns 0 or it has no effect and returns -1.
  ```

BlockDB implements its five operations as follows. It executes operations one-at-a-time. That is, it executes each invoked operation in its entirety before executing the next invoked operation.

```
lastTrans = 1;

int start()
{
      if (lastTrans < 0) { return -1 }
      else
      {
            lastTrans = -(++lastTrans);
            return lastTrans
      }
}


int read(int diskBlockAddr, int tId, int *memBlock)
{
      /* find the cache element e containing the block whose disk address
      is diskBlockAddr */
      if (there is such a cache element e)
      {
            /* the disk block at diskBlockAddr is in cache */
            Cache(e).tId = tId;
            memBlock = &Cache(e).newBlock;
            /* &Cache(e).newBlock is the address of Cache(e).newBlock */
            return 0
      }
      else
      {
            /* pick a cache entry e, where Cache(e).tId = 0.
            If there is no such entry, then return -1 */
            status = diskRead(diskBlockAddr, &Cache(e).oldBlock);
            if (status != 0) { return -1 };
            Cache(e).newBlock = Cache(e).oldBlock;
```

```
            memBlock = &Cache(e).newBlock;
            Cache(e).diskBlockAddr = diskBlockAddr;
            Cache(e).tId = tId;
            return 0
      }
}


/* A transaction should call write(&Cache(e).newBlock, tId) after it
updates Cache(e).newBlock. */
int write(int diskBlockAddr, int tId)
{
      /* find the cache entry e for block diskBlockAddr */
      if (there is no such entry) { return 0 }
      else
      {
            Cache(e).tId = tId;
            return 1
      }
}


int commit(int tId)
{
      for (each cache entry e where Cache(e).tId == tId)
      {
           status = diskWrite(Cache(e).diskBlockAddr, &Cache(e).newBlock);
            Cache(e).tId = -tId;
            if (status == -1)
            {
                  Abort(tId);
                  return -1
            }
            Cache(e).oldBlock = Cache(e).newBlock
      }
      for (each cache entry e where Cache(e).tId == -tId) {
            Cache(e).tId = 0
      }
      lastTrans = -lastTrans;

      return 0
}


int abort(int tId)
{
      for (all cache entries e, where Cache(e).tId == -tId)
      {
            Repeat
            {
            status = diskWrite(Cache(e).diskBlockAddr, &Cache(e).oldBlock)
            } until (status == 0);
            /* Of course, this will not terminate if diskWrite keeps */
            /* failing, but ignore that issue */

            Cache(e).newBlock = Cache(e).oldBlock
      }
```

```
        for (all cache entries e)
        {
              Cache(e).tId = 0
        }
        lastTrans = -lastTrans;

        return 0
}
```

Questions

For each of the scenarios below, analyze whether transactions that use BlockDB satisfy the four ACID properties. In each case, if a property is violated, show a counterexample where it does not satisfy the property. In the latter case, you might want to suggest a way to fix the implementation, but don't lose sleep over it, since it might be pretty hard.

a. The operating system and BlockDB never crash. Transactions run serially. Please consider the case in which a transaction is terminated early (e.g., due to division by zero) and an agent detects this and aborts the transaction.

b. The operating system can crash, in which case main memory is lost. Transactions run serially.

c. The operating system and BlockDB never crash. Transactions run serially. DiskWrite(b, addr) always returns 0, but it might corrupt block b on disk.

d. A variation of (c): The operating system and BlockDB never crash. Transactions run serially. If DiskWrite(b, addr) returns **-1** then it might have corrupted block b on disk (but not if it returns 0).

e. The operating system and BlockDB never crash. The specification of Start is modified to allow up to two transactions to run concurrently. So its implementation is modified to keep track of the number of active transactions and return -1 if it is called when two transactions are concurrently active.