CS 485–Assignment 2: A Transactional Store

Prof. Jon Solworth

Due: Thursday, 5 Apr 2007

1 Introduction

You are to write a transactional store. Before each transaction you should write

```
begin(T);
```

and after each transaction

```
end(T);
```

The `begin` and `end` appear in the same scope and are implemented with macros which in turn use **catch** and **try**.

In between `begin` and the `end` any number of reads, writes, allocate, and release can be performed on blocks.

- void * T.read(int b); // read block b, b must be allocated
- T.write(int b,void *oldValue, void *value); // write block b if value not null, return old value, b must be allocated
- int T.allocate(); // allocate a previously unallocated block, return -1 on error
- T.release(int b); // release block b, b must be allocated
- T.abort(); // abort the current transaction and go back to beginning

Where b is a block address and each block is 4KBytes and is aligned.

You are to implement a library, `libtransaction.a` which contains the transaction code, and include file `transaction.h` so that user code (from your perspective, test code) includes `transaction.h` and links with `libtransaction.a`

Each process may have at most one transaction at a time which has begun but not ended. Between transaction `begin` and `end`, `read`, `write`, `release`, and `allocate` operations can occur in any order with the exception that if a read of b has been requested, than there cannot be a later write of b in that transaction. Hence, you must handle correct multiple reads of the same block and a read following a write of a block. (Extra credit if you allow write after read to the same block).

Note that if the transaction fails, there should be no visible changes to the store and the execution should rollback to the begin(T). This means not only that writes should not be visible, but also allocates and releases.

The system should allow concurrent updates to the store (multiple processes performing the transactions on the store concurrently) and should maximize the concurrency of the updates (one process should block another only if one write a block the other process is accessing).

It should not be possible using only transactions to cause deadlock, and you should describe why this is impossible.

Your system should be robust if any user process *using* this facility should terminate for any reason.

You should implement a durable store using logging and you must implement a recovery program which can to through the log and bring the store up to date.

You must reclaim disk storage including both store and log.

2 Instructions for handing in your program

Obviously, your program listing and program turnin must be complete by 12:00 on the due date. **No late programs will be accepted.** Its a real good idea to start on this program right away.

Please take particular care to show in your documentation:

- Why your program does not deadlock.
- Why your storage reclamation works.
- How much concurrency can be obtained.
- What you tested.
- What you did not test.
- What works.
- What does not work.

3 Grading Criteria

- Code quality (10 pts)
- Documentation (20 pts)
- Testing (20pts)
- Correctness (40 pts)
- Concurrency (10 pts)