**Goal**

Transactional support - concurrency control module (myPTM)

Supports serializable/atomic access as well as uncontrolled access

Doesn’t support durability (i.e. no system failures so no restart support)

**Description**

Operations: read, write, delete

Open - automatically opens in a+ (read, write, append) mode when op invoked

Close - when there are no active programs accessing the file

Programs:

Transaction - access files in atomic mode

Processes - access files in a normal mode

Supports concurrent access by both transactions and processes by:

**Strict 2PL** - ensure serializability for transactions

**WFG** - for deadlock detection and is free from livelocks

Transactions aborted by system due to deadlocks ignored (no restart)

**Undo recovery strategy** - ensures atomicity

Before images are kept in the buffer

Written to file on disk at commit time

**Design** (assume system failure free environment)

**Data structures** (i.e. lock tables)

**Methods**

To handle:

Initializations/configuration

Deadlocks and transaction failures (obliterate effects of aborted transaction)

**Implementation**

**File** - consists of records/tuples

Record - ID is unique primary key

Fields = (ID - integer | ClientName - 18-char string | Phone - 12-char string)

Stored in slotted pages whose size is 512 pages

Organize data files as Directed files

**Script file** - consists of a series of file operations (may be multiple sequences)

Operate on shared data file in modes (atomic/trans = 1 | normal/process = 0)

One operation per line

Each sequence/series of operations

Started by a Begin program primitive with execution mode

Series ends with either a Commit or Abort program primitive

\*For processes, Commit/Abort same terminating exec of process

filename will be a single letter such as X or Y

Operations:

B emode

Begin, with emode=1 (transaction) or emode=0 (process)

C

Commit

A

Abort

R filename val

Retrieve record with ID=val in filename

If record doesn’t exist, return -1

If file doesn’t exist, read aborted

M filename val

Retrieve record with phone area code=val in filename

If record doesn’t exist, return -1

If file doesn’t exist, read is aborted

W filename (t)

Write record t into filename

If file doesn’t exist, it is created

D filename

Delete filename

**Modules**

**Transaction Manager**

Reads commands from different program files concurrently

Passes commands to the scheduler

Have two modes of concurrent reading from program files:

round robin (one line from each file at a time in turns)

random (reads random order and random number lines)

**Scheduler**

Implements lock manager and deadlock detector

**Data Manager**

Responsible for maintaining data in data files

Executes reads and writes

Ensures atomicity of transactions

Two search methods -- scan and hash

Each set of application programs run using both methods

**Input**

multiple program files

buffer size

seed of random number generator

number of buffer pages

method of concurrent reading (round robin or random)

search method (scan or hash) -- possibly?

**Output**

Each manager associated with separate log file in which records all actions

Statistics

number of committed transactions

percentage of read and write operations

average response time