CS 2550 – PRINCIPLES OF DATABASE SYSTEMS (SPRING 2013)
DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF PITTSBURGH

Term Project: *myPTM*

Release: April 1, 2013 Due: 11:59 PM, April 11 (Design), April 22 (Protype), 2013.

Goal

The goal of this project is to provide limited *transactional* support for a small database system. By limited support we mean that it does not support durability. Specifically, the objective is to develop the concurrency control module (*myPTM*: my Pitt Transaction Manager) which, in addition to the standard uncontrolled access to files, it also provides both serializable and atomic access.

Description

myPTM handles *Read, Write* and *Delete* operations on shared data files. *myPTM* automatically opens a file in read, write and append mode (i.e.,"a+" in stdio library) when a program invokes an operation on the file and closes the file when there are no active programs accessing the file.

Programs are executed either as "transactions" or as "processes." Transactions access files in an *atomic mode* whereas processes access files in a *normal mode*. *myPTM* supports concurrent access to files by both transactions and processes by employing the *Strict Two-Phase Locking* protocol to ensure serializability for transactions. *myPTM* uses *wait-for graphs* for deadlock detection and is free from livelocks. It ensures transactions' atomicity by adopting an undo recovery strategy where all before images are kept in the buffer and they are written to the file on the disk at commit time. Transactions aborted by the system due to deadlocks should be ignored. You are not required to implement restarts.

Design

Your design should include the data structures (such as lock tables) and methods adopted, for example, to handle initializations/configuration, deadlocks and transaction failures, i.e., to obliterate the effects of aborted transactions. As mentioned above, *myPTM* does not support "durability", thereby you should assume system failure free environment.

Implementation

For this project, files consist of records (tuples) with the following fields:

(ID: integer, ClientName: 18-character long string, Phone: 12-character long string).
where ID is unique (Primary Key).

Within a file, records are stored in slotted pages whose size is 512 bytes. You can organize your data files as Directed files.

In order to illustrate the functionality of your prototype *myPTM*, you will be required to execute concurrently a number of application programs that operate on the shared data file in various modes (atomic or normal). All such programs have the same structure specified in script files. A script file consists of a series of file operations, one per line, introduced by a *Begin* program primitive associated with the execution mode that the file operations are supposed to execute. An operation series ends with either a *Commit* or Abort program primitive. In the case of processes (normal mode), Commit and Abort primitives behave alike, terminating the execution of the process. A script file may contain multiple such sequences of file operations.

Here is the list of command lines corresponding to program primitives and file operations: *filename* will be a single letter such as X or Y.

B EMode : Begin (Start) a new transaction (EMode=1) or new process (EMode=0).

C : Commit (End) current transaction (process).

A : Abort (End) current transaction (process).

R filename val : Retrieve the record with ID=val in filename. If the record does not exist, it returns -1. If file does not exist, the read is aborted.

M filename val : Retrieve the record(s) which have val as area code in Phone in filename. If the record does not exist, it returns -1. If file does not exist, the read is aborted.

W filename (t) : Write the record t into filename. If file does not exist, it is created.

D filename : Delete filename.

An example of a file command is

```
----------------------------------------------
     B 1
     R X 13
     W X (2, Thalia, 412-656-2212)
     R X 2
     M X 412
     C
----------------------------------------------
```

Each manager is associated with a separate log file in which it records all its actions. This is also true for the data manager which actually invokes the operations on the data files. At the end of the execution of a set of application programs, the data files accessed in an atomic mode contain all the operations of the committed transactions.

Furthermore, at the end of the execution of a set of applications programs, the statistics of the execution should be displayed. These statistics will be the *number of committed transactions*, the *percentage of read and write operations* and the *average response time*. In order to test the effectiveness of the use of Direct filed, you should assume two search methods: *scan (1)* and *hash (2)*. Each set of application programs should be run using both methods under limited buffer space. The number of buffer pages will be specified at the beginning of each execution.

You have the option of implementing your prototype *myPTM* in any language and on Windows or Unix-based operating system. You will be required to demonstrate your system and submit an electronic copy of your code, log and data files.

In summary, you will implement three modules: Transaction Manager (TM), Scheduler, and Data Manager (DM). The TM is responsible of reading commands from different program files concurrently and pass the commands to the Scheduler. You need to implement two methods of concurrent reading from program files: a Round Robin (which reads one line from each file at a time in turns) and a Random (which reads from files in random order, and reads a random number of lines from each file). You should allow the user to specify multiple program files at start time, the buffer size as well as the seed of the random number generator. The Scheduler implements the lock manager and deadlock detector. The DM is responsible of maintaining the data in the data files, i.e., executing the reads and the writes and ensuring atomicity of transactions.

Submission

Requirements (**Multiple deadlines**):

- All submissions are electronic via the homework submission page (link available in course web page).
- You must work in teams of 2. You will have to declare the members and team name (if you wish) of your team by **Thursday, April 4, 2013**. Send e-mail to cs2550-staff@cs.pitt.edu with your team name and the name and pitt username of each member, CCing the email to all members of the team.
- You must submit a 2-page design document in (PDF) as described above by **Thursday, April 11, 2013**. Name the submission file using the team name (e.g., `wildcats.pdf`) or your usernames (e.g., `pitt2-usr9.pdf`), if you don't have a team name
- As part of your final prototype submission (using the same naming as in the case of your design document), you will need to submit a README file, in which you briefly describe the features of your concurrency control, any shortcomings and how to run it.
- You must submit your assignment before the due date (**Monday, 11:59pm, April 22, 2013**).
- The demos will be scheduled on *Tuesday, Thursday and Friday, April 23, 25 and 26, 2013*.

Academic Honesty

The work in this project is to be done *independently* as a group. Discussions with other groups on the assignment should be limited to understanding the statement of the problem. Cheating in any way, including giving your work to someone else will result in an F for the course and a report to the appropriate University authority.