Cd ..CS 7330

Programming Homework 2 – (Strict) Two-phase locking

Due Date: Dec 8th 11:59pm (with late pass for 48 hours only)

For this program, you are to implement a version of (strict) two-phase locking. You can use either C++ or python to implement the project.

We first describe the various parts of the program

Database

To make thing simple, the database contains a list/array of integers. They will be referred to by its position in the list/array (e.g. integer 0, integer 15 etc.)

You will be provided with a database class with the following methods (we use db[k] to denote the k-th integer of the database).

Constructor:

- Database(int k, bool nonzero): Create a database that store k integers. If nonzero = true, then db[i] is initialized to i+1, else all db[i] = 0

Methods:

- int Read(int k): Return the db[k] (notice that the number is referenced from 0 to k-1)
- void Write(int k, int w): Set db[k] = w.
- Void Print(): Display the content in the database.

You are NOT allowed to modify the Database class.

Lock Manager

This is a object that is used to maintain all the locks that is held by various transactions. All lock requests have to be made through the lock manager.

Your lock manager should have the following methods:

Constructors:

- LockManager(…) : Create a new lock manager. (You are to determine what parameters need to be passed to)

Methods

- Int Request(int tid, int k, bool is_s_lock); where tid is the transaction id, k is the -k-th integer of the database where the lock is requested; is_s_lock is true if the request is for a S-lock, otherwise it is for an X-lock. Return 1 if lock is granted, 0 if not.
- Int ReleaseAll(int tid): release all the locks that is held by transaction tid. Return the number of locks released

- vector<pair<int, bool> > ShowLocks(Int tid): return all the locks that is being held by transaction tid. For C++ you should return a vector of pairs, each <int, bool> pair contains the item to be locked, and whether it is a S-lock (true) or a X-lock (false).
  For Python, you should return a list of tuples, where each tuple has the same format as the pairs specified above.

You are to implement this class and add new methods if you want.

Transaction

Each transaction is a list of commands that read from and write to the database. It also contains a set of local variables (integers) that the transactions can manipulate.

You should create a Transaction class, with the following method defined (we use local[m] to denote the m-th local variable):

Constructors:

- Transaction(int k): Create a transaction with k local variables (reference from 0 to k-1)

Method:

- void Read(Database& db, int source, int dest): Read the source-th number from the database and copy it local[dest]
- void Write(Database& db, int source, int dest): Write the value of the local[source] from the to the dest-th number in the database.
- void Add(int source, int v), void Mult(int source, int v): set local[source] = local[source] – v and local[source] = local[source] * v respectively
- void Copy(int s1, int s2): set local[s1] = local[s2]
- void Combine(int s1, int s2): set local[s1] = local[s1] + local[s2]
- void Display(): Display all local variables' value. You should list all the numbers on the same line, with one space between each of them.

You may need/want to store more info and provide more methods for the transaction class. You are welcomed to do that.

Main Task of the program

Your program should run using the following command (assume you compile your C++ program to a.out):

../a.out <number of items in database> < file 1> < file 2> … <transaction file n>

Where <number of items in database> is a positive number to denote the number of items in the database; and <file 1> ,, <file n> is the name of a set of files, each of them is a transaction.

DO NOT print a prompt and ask the user for input. Each file corresponds to a list of commands for a transaction. In terms of numbering, the first file should corresponds to T0, the second file T1 etc. Each file has the following format:

- The first line is two numbers. The first number is the number of instructions for the transaction. The second number is the number of local variables for that transaction.
- Each subsequent line is an instruction. Each instruction contains three words, the first is a letter, which denote the operator, and the next two are numbers are the operands. The list of instructions are as follows (x,y, d are all integers):
    - R x y – read db[x] and store it to local[y]
    - W x y – write local[x] to db[y]
    - A x d – local[x] = local[x] + d
    - M x d – local[x] = local[x] * d
    - C x y – local[x] = local[y]
    - O x y – local[x] = local[x] + local[y]
    - P x y – print the current elements in the database (x, y are ignored)

Your main program will have one database object, and then read in all the transaction files and create one transaction each. Then you should "run" all transaction using the following loop:

While (there are transactions still not finished)
Check if all transactions are blocked (see below),
if so,
print "Deadlock" and quit
otherwise
Randomly pick one of the transactions (it's ok to pick blocked transaction)
Try executing the next instruction of that transaction

A few points of notes:

- You are required to request the corresponding locks before executing a command
- If a transaction wants to read an item from the database and later on write that item, request an S-lock first, and then request the X-lock
- Whenever a transaction writes an item to the database, it needs to change the database immediately
- A transaction commits immediately after the last command is executed. All locks are released at that moment.
- Every time you try to execute an instruction, you should print the following like

    T<transaction id> execute <instruction> <k>

    Where <instruction> is the instruction to be executed (together with its parameter), and k is the k-th instruction of that transaction. You print this line every time you attempt to execute that instruction

- If a transaction requests a lock, after printing the previous line, you print the statement

    T<transaction id> request <S/X>-lock on item <item number> : G/D

    Where G (granted) / D (denied) correspond to whether the lock is granted or not.

- We do not have a wait queue for locks. Whenever a request is denied, the system will simply pick another transaction (maybe the same one) to execute. (It is ok if the same transaction is picked multiple times and the request is denied every time).

- We do not do any deadlock management. We check for deadlocks using a simplified algorithm:
    - For each transaction we keep a Boolean variable (name "blocked") to check if the current instruction is blocked (because of a denied lock request). We initially set all these variables to false. Whenever a transaction is blocked, they set "blocked" to true. However, whenever a transaction is allowed to proceed, the "blocked" variables in ALL transactions are set to false. Whenever all non-finished transactions have the "blocked" variable set to true, then a deadlock occurs.

  When a deadlock is detected. The system should print "Deadlock" and exit the program.

- If all transaction finishes, you should call the Print() method for the database to print the value, and then exit the program.

- Notice that the output specified should be the ONLY statements that is printed out. (Except for the P x y command where the database is printed) Each statement should be printed on a separate line. The amount of space between words does not matter. It is likely I will write a program to check whether your output is correct. The program should be able to handle variable number of spaces between characters but otherwise the output has to be as stated.

What to hand in

You need to hand in a copy of your source code ONLY. You should have comment in your source code to denote how to compile and run the program.

Bonus (25%)

For the bonus, you would implement deadlock avoidance using the wait-die rule specified in the slides. So when a transaction request a lock and cannot obtain it, there is a possibility that it will need to rollback. A few things of note:

- Transactions that are rolled back are NOT restarted.
- If a transaction has to abort, you should print the line

  T<transaction id> rolled back

- Since a write instruction require immediate update of the database, thus there is a possibility that updated values need to be rolled back. You are responsible to set up the process of rolling back transactions.
- You may want to implement new methods for the LockManager class to make your live easier.
- The main program logic should be the same as the main part, However, you are not required to reuse the same code of your main program (as you see fit).

You should write a separate program for the bonus code (but you are free to reuse any code from your base case).