King Saud University
College of Computer and Information Sciences

Department of Computer Science

CSC383 Project Report – 2st Semester 1441-1442

Project Specifications
Developing A Deadlock Handler

Name
Nada albrahim
Alanoud alsubaie
Noor almuqayied
Reem alasmari

1. Introduction

Deadlock is one of the biggest concerns these days, as databases are accessed from various sites and demand and allocation of resources may lead to a deadlock in the database. a deadlock is an unwanted situation in which two or more transactions are waiting indefinitely for one another to give up locks. Deadlock is said to be one of the most feared complications in DBMS as no task ever gets finished and is in waiting for state forever.

This project is aimed to implement a Timestamp deadlock prevention technique, these techniques are wait-die and wound-wait. Also, the project allows users to check the deadlock in a specific time they choose.

2. Specification

Class "DBProjectCode": This class reads from text file, detects and prevents a deadlock in a specific time and uses Graphical User Interface (GUI) tools to show the result on a friendly way.

3. Design

This program contains only one class to represent and apply the wait die and wound wait algorithms. First the user is requested to enter the file directory, which technique to use, and time to check the occurrence of a deadlock. After reading, the file's each line is entered in separate position in a string array, this array specifies the line number for each action(line), then an array is initiated for locks only, either shared or exclusive in the same order they appear in the file. Here where the check method called.

Check method compares Locks in locks array together each two in a row, if they both are shared nothing happens. Changes happen when either both exclusive or one of them. in this method we apply the comparison and report the positions of the two conflicting locks then they are sent to the method technique requested by the user, either wait-die or wound-wait.

Firstly, in Wait-die method the transactions numbers of the given positions are retrieved from the lock array to get their timestamp. Then timestamps are compared

together, and the algorithm is applied, if the requesting transaction has lower time stamp, means older one, it waits otherwise it dies.

Secondly, Wound-wait method does the same thing as wait-die except if the requesting transaction has lower timestamp its wounds (kills) the other transaction otherwise its waits.

Method's name	Description
`Void readFile (String filepath, int num)`	Reads the file, enters each line in separate position in a String array, counts number of transactions, generates array of locks, and generates array of timestamps.
`Void check (String [] fileLines, int [] timestamps, String [] locksArray, int userEnteredtechnique)`	Checks if a deadlock occurred or not by applying comparison between each pair in the `locksArray`, reports the positions of the two conflicting locks, and sends the positions depending on `userEnteredtechnique` variable to the requested method technique (`waitDie or woundWait`).
`Void waitDie (String [] fileLines, int [] timestamps, String [] locksArray, int pos1, int pos2)`	Extracts the transactions numbers of the given positions `pos1, and pos2` from the `locksArray`, gets their timestamp from `timestamps` array, and applies the algorithm (if the requesting transaction has lower time stamp, means older one, it waits otherwise it dies)
`Void woundWait(String[]fileLines, int[]timestamps, String[]locksArray, int pos1, int pos2)`	Extracts the transactions numbers of the given positions `pos1, and pos2` from the `locksArray`, gets their timestamp from `timestamps` array, and applies the algorithm (if the requesting transaction has lower timestamp its wounds (kills) the other transaction otherwise its waits.)

4. Implementation

The project was implemented using only one class with main, constructor and four different methods. The class extends `JFrame`, we use JFrame here as a container to our GUI application, in this class we implement three different arrays namely, `locks, timestamp, and FileLines` that needs to be mentioned.

The `locks` array contains in each position a lock line (e.g. lock -X(B)_2), `timestamp` integer array contains time stamp of each transaction, and `FileLines` contains every line in the file in separate position.

We implement four methods namely `readFile`, `check`, `waitDie`, and `woundWait` as detailed in the previous section. Our project mainly focuses on `check` method that checks for any deadlock possibility and sends the positions of the two actions (locks) that needed to prevent the deadlock to the user requested prevention techniques methods `waitDie` or `woundWait`, the following photo represents `check` method code.

```
public void check (String[] fileLines, int[] timestamps, String[] locksArray, int userEnteredtechnique){
    for (int i=0 ; i< locksArray.length-1; i++)
        for (int j=i+1; j< locksArray.length; j++)
            if (locksArray[j].charAt(6) == 'X' ||  locksArray[i].charAt(6) == 'X')
                    if (locksArray[j].charAt(8) == locksArray[i].charAt(8))
                        if (userEnteredtechnique == 1)
                            waitDie(fileLines, timestamps, locksArray, i,j);
                        else woundWait(fileLines, timestamps, locksArray, i,j);
}
```

We mentioned previously that we ask the user to enter a time to check the dead lock in, we implement it as the following

```
if (time > LinesNum)
       printtxt = "Time entered is out of the schedule interval";
else {
    if ( time < timeRollback )
        printtxt = "until time "+ time + " there is no deadlock";

    if (timeRollback == 0)
        printtxt = "there is no deadlock"; }
```

Such that in the first case we compare the user entered time with the whole schedule interval to notify the user that the time is out of the schedule interval if it was, otherwise the second case activation means that the entered time is lower than the rollback time and this indicates that no transaction has been rolled back from the beginning until the entered time, the third case activates when there is no deadlock at all. (Note that there is another `printtxt` that reports the number of rolled back transaction and on which time in both `waitDie` and `woundWait` methods, and these cases are write over the `printtxt` string when they are activated)

5. Conclusion

Deadlock is a big issue when designing DBMS, since it must deal with many users adjusting same information at the same time. Many algorithms are designed to overcome this problem. In this project the focus was in both Wait-die and wound-wait techniques where the user is free to choose any as he prefers from the GUI. Arrays are chosen to present the actions in general and locks in specific.