DEADLOCK DETECTION IN BANKING SYSTEMS

Priyanshu Sharma, Raghvan Pareek, Rahil Godha, Swamy Pulaparthi
IIT (ISM) Dhanbad

Description: - In this project We will be building a user-interactive Banking System with many functionalities, and we will be implementing deadlock detection using Banker's Algorithm when multiple transactions are occurring simultaneously.

Features: -

- **Accounts Creating / Deleting:** Admin can create a new account just by filling out few information like account number, name of account holder, opening balance etc. An already created account can also be deleted by admin.
- **Updating Account Details:** Admin can create a new account just by filling out few information like account number, name of account holder, opening balance etc. An already created account can also be deleted by admin. Details of an already created account can be updated by Admin.
- **Money Withdrawal / Deposit / Transfer:** Money can be withdrawn from or deposited to the bank account. Money Can also be transferred from one account to another. Moreover, Admin will be notified in case of any invalid transaction.
- **Simultaneous Transaction Scrips:** Bank Admin can run a *Transaction Script* having commands for multiple transactions between bank accounts. Bank Management System will check whether the given script will run successfully or not by using "*Deadlock Detection By Banker's Algorithm*". If no deadlock occurs, the script will run successfully and all possible sequences of transactions will be shown. In case of deadlock, user will be notified accordingly.

Technology stack: -

- **C/C++ Libraries:**
  - **iostream.h** (for standard input and output operations).
  - **stdio.h, fstream.h** (for file handling).
  - **stdlib.h** (for console operations like '*clear screen*').
  - **conio.h** (for function like '*getch()*').
- **DEVC++, VSCode** (text editor).

Brief Implementation Details: -

- **User Interface:**
  As the program starts, A Main Menu of possible actions will be displayed. User will be prompted to choose any option from the given menu and according the flow of program will proceed. Once the chosen action is completed user will be directed back to the Main Menu for further actions. User can exit the program once all the actions are performed.

- **Data Storage and Retrieval:**
  Data such as Back Account Details, Transaction History etc will be stored in the system using files. File records will be updated if any account id created, deleted or updated. Or when a transaction takes place.

- **Valid or Invalid Transactions:**
  Every transaction before executing will be checked if it is even valid or not. Some examples of invalid transactions are if sender doesn't have sufficient balance, if receiver have exceeded transfer limit, If multiple transactions occurring at same time and resulting balance of account is tend to go to negative etc.

- **Implementing *Banker's Algorithm for Deadlock Detection*:**
  When the Transaction Script is executed Banker's Algorithm will be used to detect whether the flow of money as described in the script is possible between bank accounts or not. If yes, All possible sequences of transactions will be shown, otherwise user will be notified accordingly.

Brief Theory and Explanation: -

- **Deadlock:**
  A deadlock is a situation in operating systems where two or more processes are blocked as they wait for each other to release resources. This leads to a state where neither process can move forward, resulting in a standstill. Deadlocks can occur in systems where multiple processes share resources and access to those resources is controlled through synchronization mechanisms like locks. Deadlocks are a major issue in concurrent programming and can cause significant problems such as reduced system performance, data corruption and even system failure. To prevent deadlocks, it's important to implement proper synchronization techniques and design algorithms that can detect and resolve deadlock situations.

- **Deadlock In Banking Systems:**
  Deadlocks can occur in banking systems when two or more transactions attempt to access the same account(s) at the same time, and each transaction is waiting for the other to release its lock on the account.
  For example, consider a scenario where two customers, A and B, attempt to transfer money from their respective accounts to other account at the same time. If A acquires a lock on account B and B acquires a lock on account A, and then both wait for the other to release their lock on the respective account to complete the transfer, a deadlock will occur. Neither transaction can proceed because each is waiting for the other to complete first.

- **Banker's Algorithm:**
  The Banker's algorithm is a resource allocation and deadlock avoidance algorithm used in operating systems to prevent deadlocks. It works by ensuring that the resources in the system are always allocated in a safe state. The algorithm maintains a table of available resources and a table of the maximum resources needed by each process.