SHORT ESSAY:

Consider a banking system maintain an account balance with two functions: deposit(amount) and withdraw(amount). These two functions are passed the amount that is to be deposited or withdrawn from the bank account balance.

Assume an account has account balance equals 500$. If the husband calls withdraw(100) and the wife calls deposit(100) concurrently, the expected result of the account balance must be 500$. But because of racing conditions, there will be more unexpected cases happen.

Case 1: The husband calls withdraw(100) and accesses the current balance of 500$. But before the new current balance is updated, the wife calls deposit($100) and accesses to the current balance of 500$. Then the husband's program changes the shared variable current balance to 400$ followed by the wife's program changing the current balance variable to 600$. So the final balance of account will be 600$.

Case 2: It happens the same but changing the order. The husband calls withdraw(100) before the new current balance of 600$ of the wife's program is updated, so the final result of account balance will be 400$.

The solution is to create a "Lock" on the current balance variable. The balance variable must be locked before accessing and changing it. If a method determines the variable is locked, it must wait until the other method finishes running.

The pseudocode:

```
Method withdraw(amount)
If Balance is locked, wait
else lock(balance)
Balance = balance – amount
Unlock(balance)
End withdraw()

Method deposit(amount)
If Balance is locked, wait
else lock(balance)
Balance = balance + amount
Unlock(balance)
End deposit()
```