Sistema de Gestión de Cuentas

Fernando Daniel Monroy Sánchez - A01750536

Eugenio Andrés Mejía Fanjón - A01412143

Descripción

Este programa en Go simula un sistema de gestión de cuentas que soporta depósitos, retiros y transferencias entre cuentas de manera concurrente. Utiliza `mutex` para manejar la sincronización y prevenir condiciones de carrera. Se realizan transacciones aleatorias para demostrar las capacidades del sistema.

Estructura Account

Representa una cuenta bancaria con las siguientes propiedades:

- `balance`: El saldo actual de la cuenta (entero).
- `mutex`: Un mutex para asegurar operaciones seguras en la cuenta desde múltiples hilos.
- `history`: Un slice de cadenas que almacena el historial de transacciones.

Los siguientes son los métodos de la estructura:

- `Account.Deposit(amount int)`
  - Deposita la cantidad especificada en la cuenta. Si la cantidad es negativa, se convierte en positiva. El método bloquea el mutex antes de modificar el saldo y agrega un mensaje al historial de transacciones.
- `Account.Withdraw(amount int)`
  - Retira la cantidad especificada de la cuenta si el saldo es suficiente. Si la cantidad es negativa, se convierte en positiva. El método bloquea el mutex antes de modificar el saldo y devuelve `true` si el retiro fue exitoso, de lo contrario `false`. La transacción se registra en el historial.
- `Account.TransferTo(to *Account, amount int)`
  - Transfiere la cantidad especificada desde la cuenta actual a la cuenta de destino si el saldo es suficiente. Asegura que los mutexes se bloqueen en un orden consistente para prevenir interbloqueos comparando las direcciones de memoria. La transacción se registra en el historial de ambas cuentas y el método devuelve `true` si es exitosa, de lo contrario `false`.

Programa

Se realizan 10 transacciones aleatorias (depósito, retiro o transferencia) entre cuentas aleatorias de manera concurrente utilizando `goroutines`. Se agrega un pequeño retraso entre transacciones para simular condiciones del mundo real.

La función principal inicializa el programa:

1. Inicializa el generador de números aleatorios con el tiempo actual.
2. Crea 10 cuentas con saldos iniciales aleatorios entre $1000 y $3000.
3. Lanza 5 `goroutines` para realizar transacciones concurrentemente.
4. Espera a que todas las transacciones finalicen usando un `WaitGroup`.
5. Imprime el saldo final y el historial de transacciones de cada cuenta.

Para ejecutar el programa, ejecutarlo con el siguiente comando en la terminal:

```
go run act5-3.go
```

La salida mostrará los saldos finales de todas las cuentas y sus historiales de transacciones.

Un ejemplo de salida es el siguiente:

```
Account 1 (0xc000090210) balance: $429
Account history:
- Withdrew $372
- Deposited $286
- Withdrew $895
- Transferred $205 to 0xc000090300
- Withdrew $347
- Transferred $112 to 0xc000090300

Account 2 (0xc000090240) balance: $762
Account history:
- Transferred $992 to 0xc0000902d0
- Received $335 from 0xc0000901b0
- Transferred $535 to 0xc0000901e0
- ...

...
```