**TRANSACTION** **PROCESSOR**

**PROBLEM STATEMENT:**

Read transaction details from JSON file and process it using circular buffer which has maximum capacity. Multiple readers should be able to read from circular buffer.

**SOLUTION:**

The solution is written using Golang

Circular buffer is represented using the following structure

type CircularBuffer struct {

    data    []string      // data holding json contents

    size    int           // size of circular buffer

    rpos    int           // Read position

    wpos    int           // Write position

    rwmutex \*sync.RWMutex // Read Write Mutex

}

Read write mutex is used for handling synchronization of circular buffer. It has 2 different lock methods: [RWMutex.Lock()](https://golang.org/pkg/sync/#RWMutex.Lock) for writers, and [RWMutex.RLock()](https://golang.org/pkg/sync/" \l "RWMutex.RLock) for readers. There are 2 different unlock methods, matching the different lock methods: [RWMutex.Unlock()](https://golang.org/pkg/sync/" \l "RWMutex.Unlock) and [RWMutex.RUnlock()](https://golang.org/pkg/sync/#RWMutex.RUnlock)

RWMutex allows multiple readers, or 1 writer. If a writer obtained the lock, no readers are allowed until the writer unlocks (also, no other writers allowed). If a reader obtains the read-lock, any other readers are allowed (but not a writer, not until all readers unlock).

TransactionProcessor represents main structure used to drive entire functionality

type TransactionProcessor struct {

    circularBuffer \*CircularBuffer // Circular buffer to store values

    readers        int             // Number of reader goroutines

    wg             sync.WaitGroup  // WaitGroup to coordinate goroutines

}

The execution starts from main function defined in main.go

Configurable values for circular buffer size and number of readers is present in main function.

1. Ring Buffer and Transaction processor structures are instantiated
2. JSON file is opened and contents are read
3. The contents are unmarshalled into record structure and written to circular buffer
4. Multiple readers will then consume the data from circular queue concurrently using goroutines.
5. To synchronize goroutine, waitgroup is used. It is used to wait till all readers are done reading from circular buffer.

TEST RESULTS:

Please run the below command to run the program



A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

UNIT TESTING AND COVERAGE:

Unit tests are written using go testing framework. Wrote tests to cover all functions and different scenarios and achieved 100% coverage except main function.

Cover package is used for reporting code coverage. Install it using below command.

A screen shot of a computer program

Description automatically generated

Please find below unit test results with coverageA screen shot of a computer

Description automatically generated

Please find below function coverage results

A screen shot of a computer

Description automatically generated

Please run the below command to view the coverage report in HTML in your default browser

Please find below coverage report as viewed in browser

A screenshot of a computer program

Description automatically generated