Go Concurrency

Large programs are divided into smaller sub-programs. Programs which run their smaller components at the same time is known as concurrency.

Goroutines

The parts of an application that run concurrently are called goroutines. Goroutines and channels are used for structuring concurrent programs.

A process is an independently executing entity running in a machine which runs in its own address space in memory. A process has threads which are simultaneously executing entities. Threads share the same address space of the process.

Goroutines are lightweight, much lighter than a thread. Goroutines run in the same address space, so access to shared memory must be synchronized; This can be done by sync package, but it is recommended to use channels to synchronize goroutines.

A goroutine is implemented as a function or method. It is called (invoked) with the 'go' keyword.

When the goroutine finishes, nothing is returned to the caller function.

# Go Race

A race condition occurs in Go when two or more goroutines try to access the same resource. It may happen when a variable attempts to read and write the resource without any regard to other routines.

# Go Mutex

Mutual Exclusion locks, or mutexes can be used to synchronize access to state and safely access data across many goroutines. It acts as a guard to the entrance of the critical section of code so that only one thread can enter the critical section at a time.

We set a lock around particular lines of code with it. While one Goroutine holds the lock, all other Goroutines are prevented from executing any lines of code protected by the same mutex, and are forced to wait until the lock is yielded before they can proceed

# Go Atomic Variable

Atomic variables are used to manage state, though sync/atomic package and avoid race conditions. Atomic counters can be accessed by multiple go routines.

# Go Channel

The channel acts as a pipe by which we send typed values from one Goroutine to another. It guarantees synchronization since only one Goroutine has access to a data item at any given time. The ownership of the data is passed between different Goroutine. Hence, By design it avoids the pitfalls of shared memory and prevent race condition.

# Go Worker Pools

Worker pools is a design in which a fixed number of m workers (Go goroutines) works on n tasks in a work queue (Go channel). The work resides in a queue until a worker finish its current task and pull a new one.