Concurrent Programming with Go - Part II

Concepts of Programming Languages
22 November 2018

Johannes Weigend (QAware GmbH)
University of Applied Sciences Rosenheim

Deadlock and Data race Detection

- The Go runtime detects a deadlock if all Go routines are asleep
- Data races are among the most common and hardest to debug types of bugs in concurrent systems.
- A data race occurs when two goroutines access the same variable concurrently and at least one of the accesses is a write.
- The Go Data race detector detects these problems at runtime

```
$ go test -race mypkg    // to test the package
$ go run -race mysrc.go  // to run the source file
$ go build -race mycmd   // to build the command
$ go install -race mypkg // to install the package ==> Performance impact
```

2

Data race examples

- Unprotected global variable (maps are not threadsafe!)

```go
var service map[string]net.Addr

func RegisterService(name string, addr net.Addr) {
    service[name] = addr
}

func LookupService(name string) net.Addr {
    return service[name]
}
```

Solution

```go
var (
    service   map[string]net.Addr
    serviceMu sync.Mutex
)

func RegisterService(name string, addr net.Addr) {
    serviceMu.Lock()
    defer serviceMu.Unlock()
    service[name] = addr
}

func LookupService(name string) net.Addr {
    serviceMu.Lock()
    defer serviceMu.Unlock()
    return service[name]
}
```

Necessary Conditions for Deadlock

- Deadlock can arise if the following 4 conditions hold simultaneously:
- Mutual exclusion: only one process at a time can use a resource
- Hold and wait: a process holding at least one resource is waiting to acquire additional resources held by other processes
- No preemption: a resource can be released only voluntarily by the process holding it, after that process has completed its task
- Circular wait (see slides)

www.cs.colostate.edu/~cs370/Fall15/slides/7deadlocks.pdf (http://www.cs.colostate.edu/~cs370/Fall15/slides/7deadlocks.pdf) 5

Building a Deadlock Avoidance Detector in Go

```go
// ResourceGraph is a simple RAG ResourceAllocationGraph.
// A graph is a collection of edges. Each edge is of the form
// edge := source -> [target1, target2, ... targetN]
type ResourceGraph struct {
    edges map[string][]string
}

// NewResourceGraph creates an empty graph.
func NewResourceGraph() *ResourceGraph {
    resourceGraph := new(ResourceGraph)
    resourceGraph.edges = make(map[string][]string)
    return resourceGraph
}

// AddLink adds a link to the graph.
func (r *ResourceGraph) AddLink(source, dest string) {
    destinations := r.edges[source]
    destinations = append(destinations, dest)
    r.edges[source] = destinations
}

// RemoveLink removes a link from the graph.
```

6

Building a Deadlock Avoidance Detector in Go

```
// ResourceManager detects deadlocks by finding cycles in a Ressource Allocation Graph.
// blocks, when a resource is in use. The user is responsible to implement a resource release strategy,
// Sample
// Acquire ("P1", "R1" ) : P1 <- R1 (ok) - Process 1 got Resource 1
// Acquire ("P1", "R3" ) : P1 <- R3 (ok) - Process 1 got Resource 3
// Acquire ("P2", "R3" ) : P2 <- R2 (ok) - Process 2 got Resource 2
// Acquire ("P2", "R1" ) : P2 -> R1 (wait) - Process 2 cant get Resource 1 (in use by Process 1) : wait
// Acquire ("P1", "R2" ) returns false : P1 -> R2 (deadlock) - acquire will recognize the deadlock and r
type ResourceManager struct {
    graph *ResourceGraph
    m     sync.Mutex
    c     sync.Cond
}

// NewResourceManager creates a Resource Manager.
func NewResourceManager() *ResourceManager {
    manager := new(ResourceManager)
    manager.c = sync.Cond{L: &manager.m}
    manager.graph = NewResourceGraph()
    return manager
}
```

Building a Deadlock Avoidance Detector in Go

```go
func (r *ResourceManager) Acquire(processName string, resourceName string) bool {
    r.m.Lock()
    defer r.m.Unlock()

    if resourceName == "" || processName == "" {
        panic(errors.New("processname or resourcename can not be empty"))
    }

    r.graph.AddLink(processName, resourceName) // add Px -> Ry

    for r.resourceIsInUse(resourceName, processName) {
        if r.graph.DetectCycle(processName, resourceName) {
            r.graph.RemoveLink(processName, resourceName)
            return false // Deadlock detected
        }
        // log.Printf("Blocking %v, %v, %v", resourceName, processName, r.graph)
        r.c.Wait()
    }

    r.graph.RemoveLink(processName, resourceName) // remove Px -> Ry
    r.graph.AddLink(resourceName, processName)    // add Ry -> Px

    return true // no deadlock
```

8

Building a Deadlock Avoidance Detector in Go

```go
func (r *ResourceManager) Release(processName string, resourceName string) {
    r.m.Lock()
    defer r.m.Unlock()

    r.graph.RemoveLink(resourceName, processName)
    r.c.Signal()
}

/**
 * A resource is in use when a process owns the resource :
 * R1 -> [Px]
 */
func (r *ResourceManager) resourceIsInUse(resourceName string, processName string) bool {
    process := r.graph.Get(resourceName)
    if process == nil {
        return false
    }
    return len(process) == 1
}
```

9

Dining Philosophers with ResourceManager

```
// take forks
gotForks := false
for !gotForks {
    gotForks = manager.Acquire("P" + p.id, "F" + p.id)
    if gotForks {
        gotForks = manager.Acquire(ph, f2)
        if !gotForks { // deadlock detected
            manager.Release("P" + p.id, "F" + ((p.id + 1) % COUNT)
        }
    } else {
        log.Println("Deadlock detected -> try again")
    }
}

// eat

// put forks
manager.Release("P" + id, "F" + ((id + 1) % COUNT))
manager.Release("P" + id, "F" + id)
```

github.com/jweigend/concepts-of-programming-languages/tree/master/cp/locks/resourcemanger (https://github.com/jweigend/concepts-of-programming-languages/tree/master/cp/locks/resourcemanger) github.com/jweigend/concepts-of-programming-languages/tree/master/cp/locks/philosophers (https://github.com/jweigend/concepts-of-programming-languages/tree/master/cp/locks/philosophers)

10

Thank you

Johannes Weigend (QAware GmbH)
University of Applied Sciences Rosenheim
johannes.weigend@qaware.de (mailto:johannes.weigend@qaware.de)
http://www.qaware.de (http://www.qaware.de)
@johannesweigend (http://twitter.com/johannesweigend)