Building CSP-style Concurrent Systems in Go

James Whitehead II

University of Oxford

jim.whitehead@cs.ox.ac.uk

March 20, 2012
About me

- Doctoral student
- World of Warcraft addon developer
  - Hacking World of Warcraft
  - World of Warcraft Programming: a Guide and Reference for Creating WoW Addons
- Open source contributor
  - Go (golang, chunkymonkey)
  - Lua (sputnik, WoW addons)
Introduction to Go

- Compiled (quickly)
- Statically typed
- Garbage collected
- Interface types
- Object-oriented, with no classes or hierarchy
- Concurrency primitives
Methods on types

```go
package main

import "fmt"

type IntMonoOp func(int) int
type IntBinOp func(int, int) int

func (fn IntBinOp) Curry(x int) IntMonoOp {
    return func(y int) int {
        return fn(x, y)
    }
}

var Add IntBinOp = func(x, y int) int { return x+y }

func main() {
    addtwo := Add.Curry(2)
    fmt.Printf("%d, %d, %d\n", addtwo(1), addtwo(2), addtwo(3))
}
```
Communicating Sequential Processes

A way to reason about and prove properties of concurrent systems.

- Construct simple sequential programs
- Run them in parallel
- Allow them to communicate
  - explicit channels
- Prove properties about the result
- Specification and implementation
Concurrency family.. network?

Story by Rob Pike: [http://goo.gl/KmYvA]
Concurrent in Go

Goroutines
The execution of a method or function call executing concurrently in the same address space as other goroutines.

go doSomething("http://golang.org")

- anonymous and decoupled
- no hierarchy
Concurrency in Go

“Do not communicate by sharing memory. Instead, share memory by communicating.”

Channels

- message passing pipe
- send waits for receive (and vice versa)
- statically typed
- first-class values
- any-to-any communication

```go
ch <- 5 // send a message on the channel
val := <-ch // receive a value from the channel
```
// A function that performs some computation
func doSomething(url string, done chan bool) {
    // perform some work
    done <- true
}

// The main entry point of the program
func main() {
    done := make(chan bool)
    go doSomething( "http://golang.org", done)
    <-done // without this, the program would exit immediately!
}
Uses for channels

Channels are the basis for communication between goroutines:

- Semaphores (buffered channel)
- Resource mutex
- Asynchronous results (futures/promises)
- Fork/join
- Client registration/callbacks
- Processing pipelines
Why choose Go?

- Message passing over explicit channels
- Considered:
  - CSP libraries for C, C++, Java, Python, etc.
  - Occam-$\pi$
  - Erlang, Scala
- C-level language (systems)
- Rich standard libraries (networking, http)
- Static types with a dynamic “feel”
- Simple, powerful concurrency

Go is worth considering for its unique collection of features.
Case Study: Webpipes

- Compositional web server toolkit
- Components are simple functions
- Sources, filters, and sinks
- Pipelines
- Process networks

Published in [Communicating Process Architectures 2011].
Case Study: MINIX file system

- Notoriously sequential
- Shared resources
- Caching and coherency
- Avoid explicit locking
- Model using CSP

Takes the microkernel principle and pushes it into the file server.
A “blocking” block device

// Interface for a block device
type BlockDevice interface {
    // Read a block into 'buf' from position 'pos'
    Read(buf interface{}, pos int64) error
    // Write the data from 'buf' to position 'pos'
    Write(buf interface{}, pos int64) error
    // Close the device
    Close() error
}
A “blocking” block device

// Turn a working BlockDevice into a faulty one
type FaultyDevice struct {
    Real    BlockDevice
    Blocked chan int64
    Unblock chan bool
}

func (dev *FaultyDevice) Read(buf interface{}, pos int64) error {
    // Signal that we are blocked
    dev.Blocked <- pos
    // Wait for someone to unblock us
    <-dev.Unblock
    return dev.Real.Read(buf, pos)
}
Conclusions

- Go is digestible; it has a small set of orthogonal, powerful features
- Concurrency is built into the language
- Different perspective
- Novel solutions to problems

Want to learn more?

- Communicating Sequential Processes
- Tour the language: [http://tour.golang.org/]
Questions? Comments? Beer?