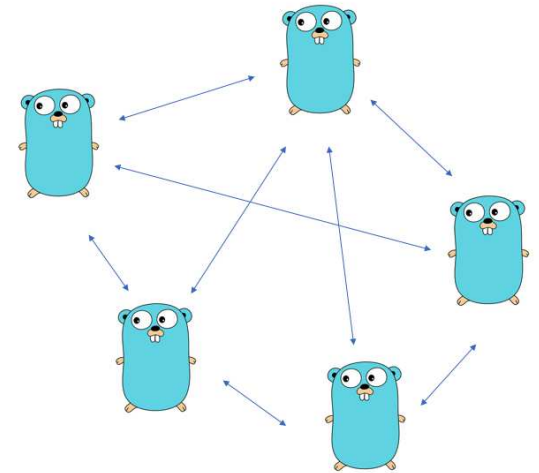


Go networking

Peter Borovanský, KAI, I-18,
borovan(a)ii.fmph.uniba.sk

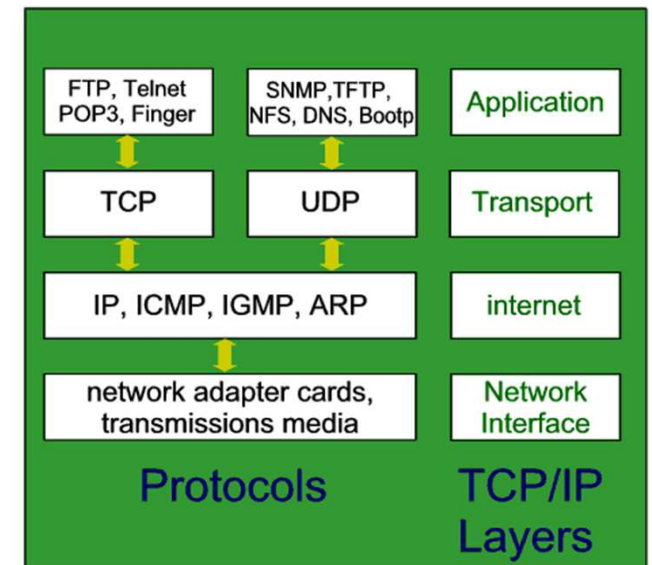


Prejdeme si v Go tri úrovne tzv. TCP Stacku, a naprogramujeme

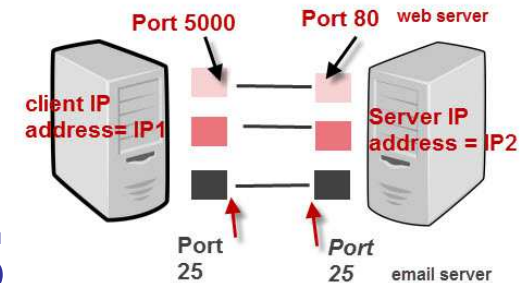
- klient/server aplikáciu cez TCP/IP sockety, príklad chat
- sntp udp klient (time server klient)
- malý Webcrawler

Zdroje:

- **Writing Web Applications**
 - <http://golang.org/doc/articles/wiki/>
- **Network programming with Go**
 - <http://jan.newmarch.name/go>



TCP/IP Ports&Sockets



IP Address + Port number = Socket

TCP/IP Ports And Sockets

- Socket = IP Address + Port Number
- IP Address 255^4
- Port 0..65536, ale
 - 0..1023 (0-0x3FF) - vyhradené pre rôzne serverovské servisy (IANA)
 - 1024..49151 (0x400-0xBFFF) - môžu byť vyhradené
 - 49152..65535 (0xC000-0xFFFF) – voľné pre klientské programy
- TCP/IP spojenie je spojenie dvoch socketov
 - klientského, vaše IP+dynamicky vygenerovaný port, napr. 61234
 - Serverovského, jeho IP+port serverovskej služby, napr. 80 pre web server
- TCP/IP protokol poskytuje
 - TCP Port – uverenie doručenia (acknowledge), opätovné vyslanie packetu
 - UDP Port – bez overenie, bez znovu vyslania nedoručeného packetu



stay at
127.0.0.1
wear a
255.255.255.0

"Hi, I'd like to hear a TCP joke."
"Hello, would you like to hear a TCP joke?"
"Yes, I'd like to hear a TCP joke."
"OK, I'll tell you a TCP joke."
"Ok, I will hear a TCP joke."
"Are you ready to hear a TCP joke?"
"Yes, I am ready to hear a TCP joke."
"Ok, I am about to send the TCP joke. It will last 10 seconds, it has two characters, it does not have a setting, it ends with a punchline."
"Ok, I am ready to get your TCP joke that will last 10 seconds, has two characters, does not have an explicit setting, and ends with a punchline."
"I'm sorry, your connection has timed out."
...Hello, would you like to hear a TCP joke?"



TCP Klient socket

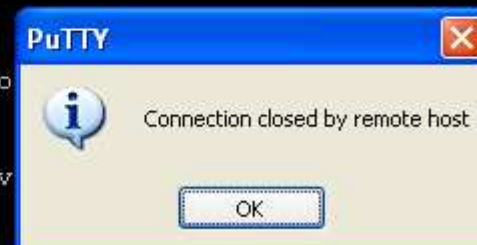
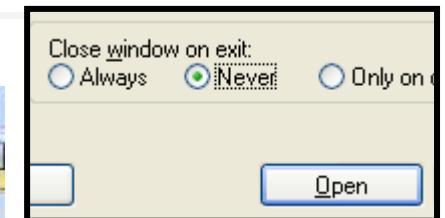
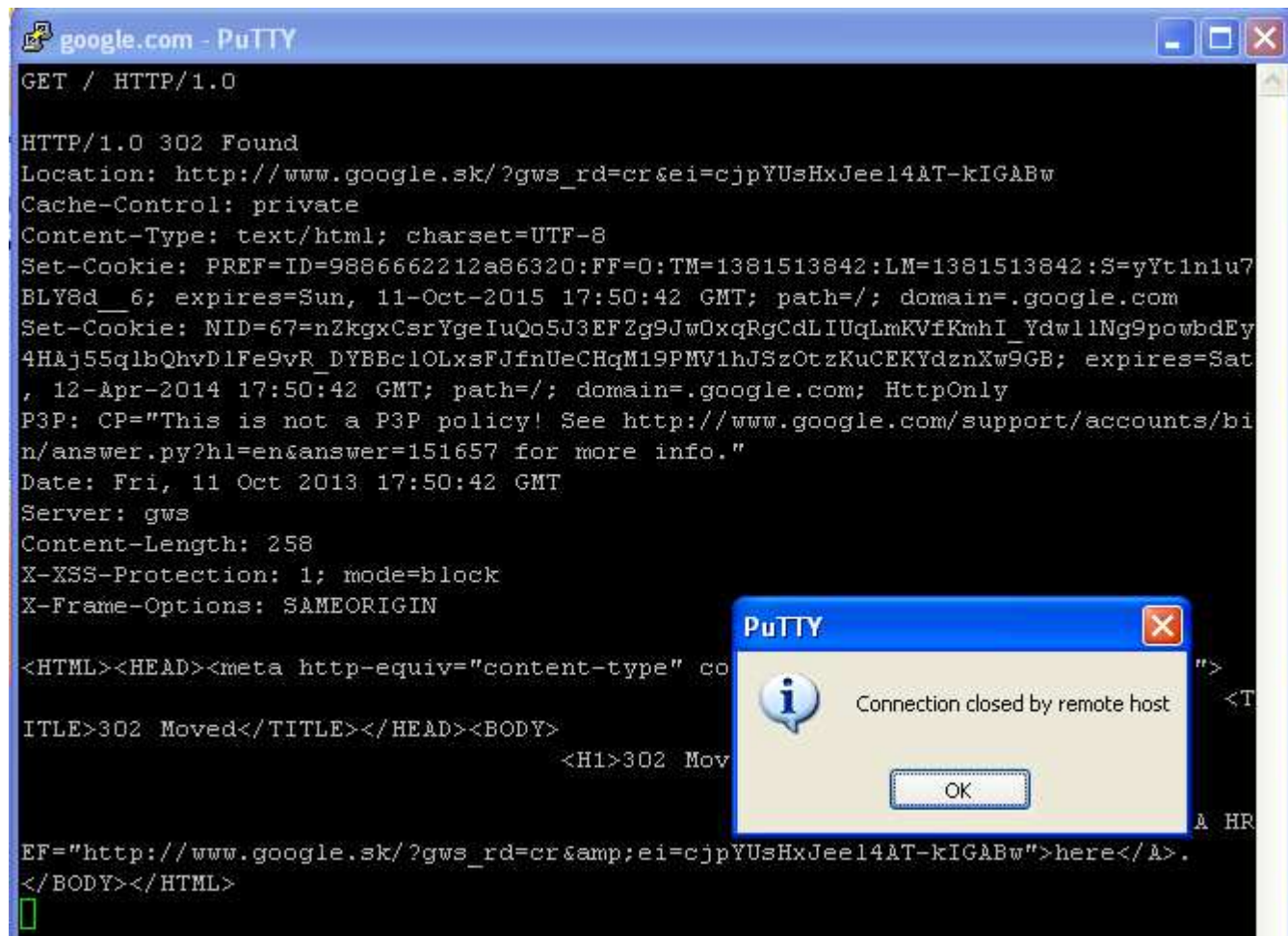
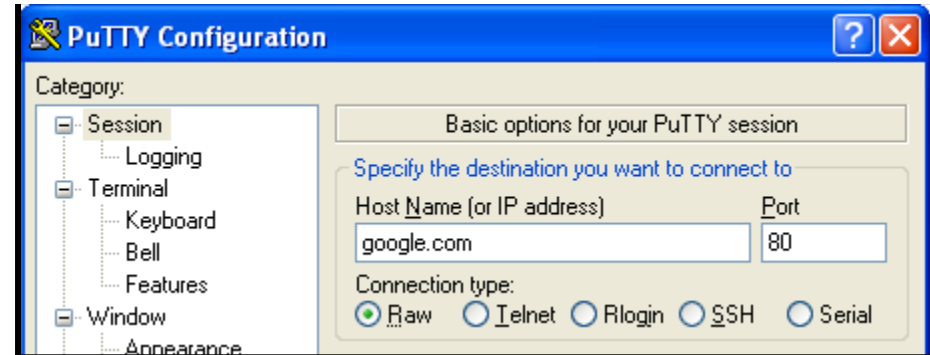
(pripája sa cez TCP klient socket na google.com:80)

```
import ( "bufio"      "fmt"      "io"      "net" )

func main() {           // vytvorenie spojenia client-socket
    conn, err := net.Dial("tcp", "google.com:80")
    if err != nil {
        fmt.Println("connection error: " + err.Error())
    } else {             // písanie do conn : net.Conn
        fmt.Fprintf(conn, "HEAD/ HTTP/1.0\r\n\r\n")
        r := bufio.NewReader(conn)           // bufio wrapper
        for {
            line, _, err := r.ReadLine()    // čítanie z conn
            if err == io.EOF {
                break
            }
            fmt.Printf("%s\n", line)
        }
    }
}
```

HTTP/1.1 status codes
1xx Informational responses
2xx Success
3xx Redirection
4xx Client errors
5xx Server errors

Putty/netcat





UDP Sntp klient

(pripája sa cez UDP na sntp server – Simple Network Time Protocol)

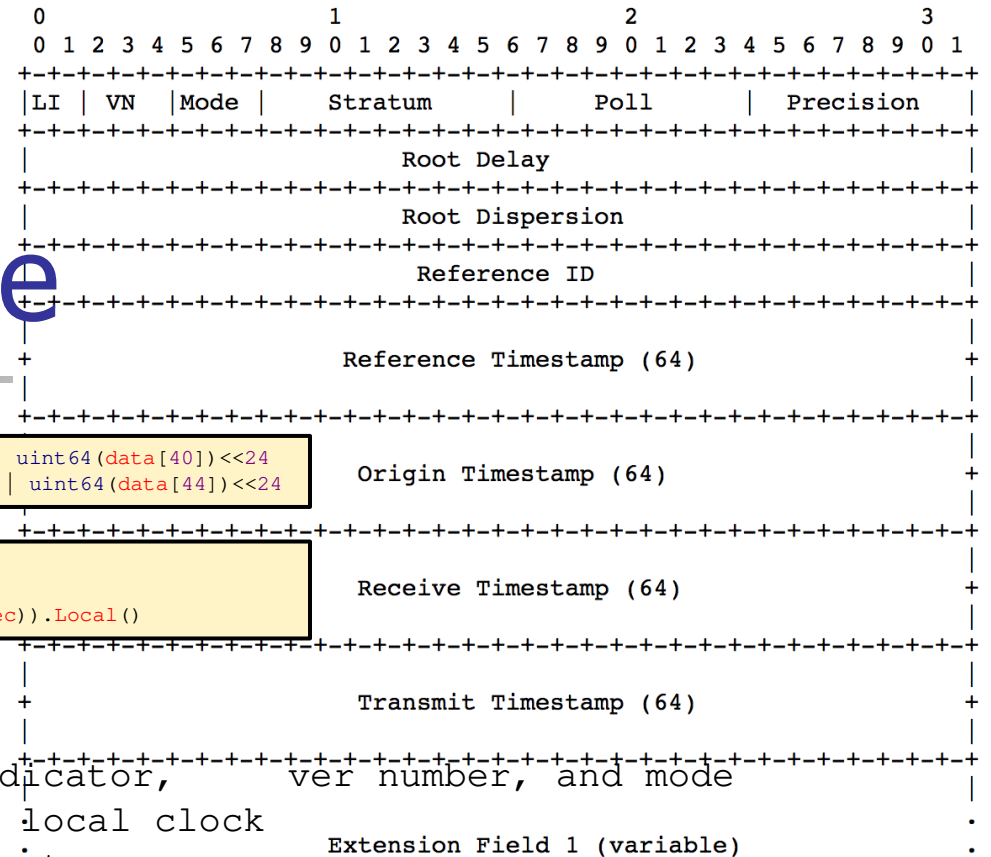
```

conn, err := net.Dial("udp", "0.sk.pool.ntp.org:123")
if err != nil { return }
r := bufio.NewReader(conn)
w := bufio.NewWriter(conn)
data := make([]byte, 48)
data[0] = 3<<3 | 3
conn.SetDeadline(time.Now().Add(5 * time.Second))
defer conn.Close()
w.Write(data) // send request
w.Flush()
data, _, err = r.ReadLine() // read response

var sec, frac uint64
sec = uint64(data[43]) | uint64(data[42])<<8 | uint64(data[41])<<16 | uint64(data[40])<<24
frac = uint64(data[47]) | uint64(data[46])<<8 | uint64(data[45])<<16 | uint64(data[44])<<24
nsec := sec * 1e9
nsec += (frac * 1e9) >> 32
t := time.Date(1900, 1, 1, 0, 0, 0, 0, time.UTC).Add(time.Duration(nsec)).Local()
fmt.Printf("Network time: %v\n", t)

```


Sntp response



```
sec = uint64(data[43]) | uint64(data[42])<<8 | uint64(data[41])<<16 | uint64(data[40])<<24
frac = uint64(data[47]) | uint64(data[46])<<8 | uint64(data[45])<<16 | uint64(data[44])<<24
```

```
nsec := sec * 1e9
nsec += (frac * 1e9) >> 32
t := time.Date(1900, 1, 1, 0, 0, 0, 0, time.UTC).Add(time.Duration(nsec)).Local()
```

type packet struct {

```
    Settings uint8 // 0: leap yr indicator, ver number, and mode
    Stratum   uint8 // 1: stratum of local clock
    Poll      int8  // 2: poll exponent
    Precision int8  // 3: precision exponent
    RootDelay uint32 // 4: root delay
    RootDispersion uint32 // 8: root dispersion
    ReferenceID  uint32 // 12: reference id
    RefTimeSec   uint32 // 16: reference timestamp sec
    RefTimeFrac  uint32 // 20: reference timestamp fractional
    OrigTimeSec  uint32 // 24: origin time secs
    OrigTimeFrac uint32 // 28: origin time fractional
    RxTimeSec    uint32 // 32: receive time secs
    RxTimeFrac   uint32 // 36: receive time frac
    TxTimeSec    uint32 // 40: transmit time secs
    TxTimeFrac   uint32 // 44: transmit time frac
```



Jednovláknový server

(počúva na porte localhost:8080)

```
import ( "bufio" "fmt" "io" "net" )

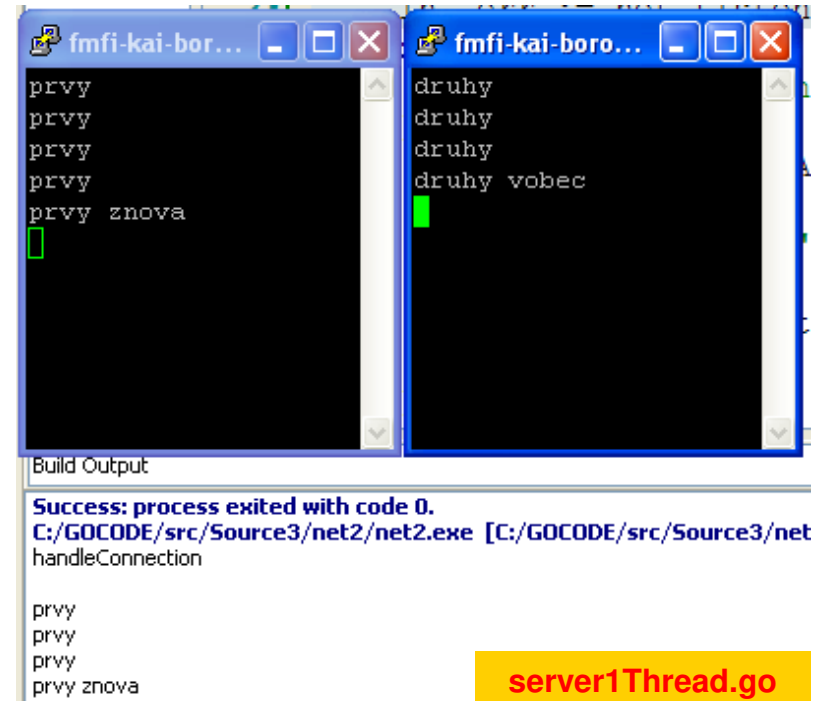
func main() {                                // porty <1024 majú špeciálne určenie
    ln, err := net.Listen("tcp", ":8080") // server socket
    if err != nil {
        fmt.Println("connection error: " + err.Error())
    } else {
        conn, err := ln.Accept() // blokuje kým ja niekto
        if err != nil {           // nepokúsi o spojenie
            fmt.Println("error: " + err.Error())
        } else {
            handleConnection(conn) // vieme vytvoriť len
        }                          // jedno spojenie na "server"
    }
}
```

server1thread.go

HandleConnection

(čítanie zo streamu nadviazanej konekcie)

```
func handleConnection(conn net.Conn) {  
    fmt.Println("handleConnection")  
    r := bufio.NewReader(conn) // wrapper na conn read stream  
    for {  
        line, _, err := r.ReadLine() // čítaj až do konca  
        if err == io.EOF {  
            break  
        } // a piš na konzolu  
        fmt.Printf("%s\n", line)  
    }  
}
```



fmfi-kai-bor... fmfi-kai-boro...

```
prvy  
prvy  
prvy  
prvy  
prvy znova  
[ ]
```

```
druhy  
druhy  
druhy  
druhy vobec  
[ ]
```

Build Output

Success: process exited with code 0.
C:/GOCODE/src/Source3/net2/net2.exe [C:/GOCODE/src/Source3/net
handleConnection

```
prvy  
prvy  
prvy  
prvy znova
```

server1Thread.go



Viacvláknový server

(čo nadviazaná konekcia, to jedno vlákno = go rutina)

```
func main() {  
    ln, err := net.Listen("tcp", ":8080")  
    if err != nil {  
        fmt.Println("connection error: " + err.Error())  
    } else {  
        for {  
            conn, err := ln.Accept()  
            if err != nil { // server by mal prežiť !!!  
                fmt.Println("error: " + err.Error())  
                continue  
            } // obslúženie konekcie spustíme v nezávislej  
            go handleConnection(conn) // goroutine  
        }  
    }  
}
```

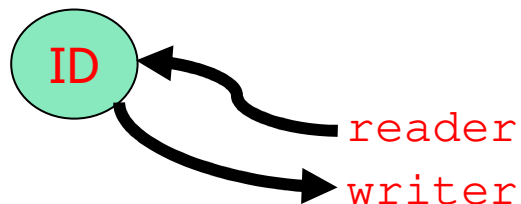
serverMultiThread.go

ChatClient

Keď poznáme základy, skúsime vytvoriť jednoduchý chat-server umožňujúci:

- viacero pripojení,
- broadcastuje komunikáciu všetkým

```
type ChatClient struct {  
    clientID int           // poradové číslo klienta  
    reader   *bufio.Reader // reader a writer z/do konekcie  
    writer   *bufio.Writer // klienta  
}  
  
func NewChatClient(clientID int, conn net.Conn) *ChatClient {  
    return &ChatClient{ // konštruktor vytvorí z ID a konekcie  
        clientID: clientID, // jednoznačné ID ChatClient  
        reader: bufio.NewReader(conn), // input pipe stream  
        writer: bufio.NewWriter(conn), // output pipe stream  
    }  
}
```



ChatRoom

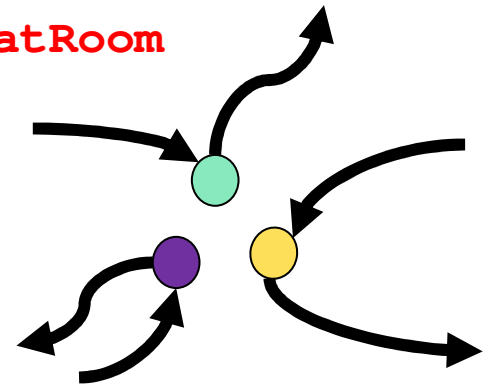
ChatRoom

- si pamätá všetkých ChatClientov
- vie pridať ďalšieho, keď sa pripojí

```
type ChatRoom struct {           // všetci ChatClienti
    clients []*ChatClient
}

func NewChatRoom() *ChatRoom { // prázdny ChatRoom
    chatRoom := &ChatRoom{
        clients: make([]*ChatClient, 0),
    }
    return chatRoom
}
```

```
func (chr *ChatRoom) AddChatClient(conn net.Conn) *ChatClient {
    chatclient := NewChatClient(len(chr.clients)+1, conn)
    fmt.Printf("new client: %d\n", chatclient.clientID)
    chr.clients = append(chr.clients, chatclient)
    return chatclient
}
```

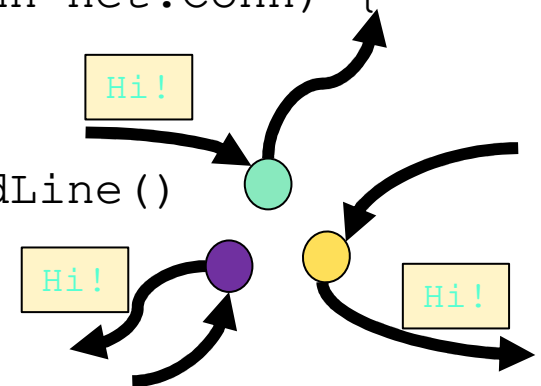


ChatRoom

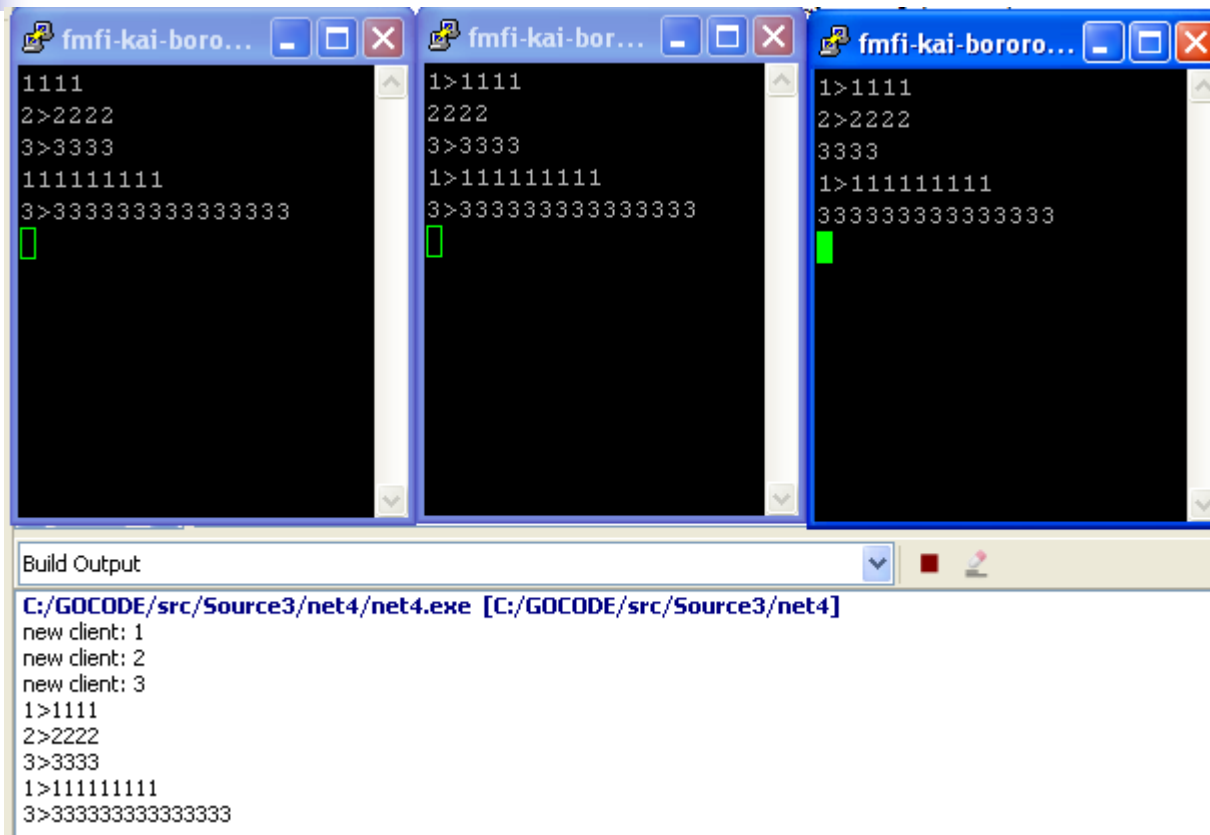
- echuje, čo jeden píše všetkým ostatným

ChatRoom

```
func (chr *ChatRoom) handleConnection (conn net.Conn) {  
    chclient := chr.AddChatClient(conn)  
    for {  
        line, _, err := chclient.reader.ReadLine()  
        if err == io.EOF {  
            break  
        }  
        msg := fmt.Sprintf("%d>%s\r\n", chclient.clientID, line)  
        fmt.Print(msg)      // výpis na konzolu chatroomu  
        for _, client := range chr.clients {  
            if client.clientID != chclient.clientID {  
                client.writer.WriteString(msg)  
                client.writer.Flush()  
            }  
        }  
    }  
}
```



ChatRoom v akcii



The image shows a screenshot of a chatroom application running in three terminal windows and a build output window. The terminal windows are titled "fmfi-kai-boro...", "fmfi-kai-bor...", and "fmfi-kai-bororo...". Each window displays a chat log with messages from three clients (1, 2, and 3). The messages are: "1>1111", "2>2222", "3>3333", "1>11111111", and "3>3333333333333333". The build output window at the bottom shows the command "C:/GOCODE/src/Source3/net4/net4.exe [C:/GOCODE/src/Source3/net4]" and the output "new client: 1", "new client: 2", "new client: 3", "1>1111", "2>2222", "3>3333", "1>11111111", and "3>3333333333333333".

```
fmfi-kai-boro... 1111
                  2>2222
                  3>3333
                  11111111
                  3>3333333333333333
                  [ ]

fmfi-kai-bor... 1>1111
                 2222
                 3>3333
                 1>11111111
                 3>3333333333333333
                 [ ]

fmfi-kai-bororo... 1>1111
                    2>2222
                    3333
                    1>11111111
                    3333333333333333
                    [ ]

Build Output
C:/GOCODE/src/Source3/net4/net4.exe [C:/GOCODE/src/Source3/net4]
new client: 1
new client: 2
new client: 3
1>1111
2>2222
3>3333
1>11111111
3>3333333333333333
```


http klient je nadstavba TCP
zaobal'uje nízko-úrovňovú komunikáciu

HTTP client

- GET Request-URI CRLF
- [GET | HEAD | POST] Request-URI HTTP-Version CRLF
- GET http://google:80 HTTP/1.0

```
url := "http://google.com"
response, err := http.Head(url)
fmt.Println(response.Status)           // 200 OK
for k, v := range response.Header { // Content-Type:
    fmt.Println(k+":", v)             } // [text/html; charset=ISO-8859-2]
response, err = http.Get(url)
fmt.Println("\nbody:")
reader := bufio.NewReader(response.Body) // čítame telo
for {
    line, _, err := reader.ReadLine()
    if err == io.EOF {
        break
    }
}
```



Čo s telom ?

```
for {
    line, _, err := reader.ReadLine()
    if err == io.EOF { break }
    strline := string(line)
    var httpRef = regexp.MustCompile(
        `(?(i)href\s*=\s*(\"([^\"]*\")|'([^']*'|\"([^\"]*\")>\s]+)))`
    )
    matches := httpRef.FindAllString(strline, -1)
    for _, match := range matches {
        fmt.Println(match)
    }
}

body:
href="/search?"
href="http://www.google.sk/imghp?hl=sk&tab=wi"
href="http://maps.google.sk/maps?hl=sk&tab=w1"
href="http://www.youtube.com/?gl=SK&tab=w1"
href="https://mail.google.com/mail/?tab=wm"
href="https://drive.google.com/?tab=wo"
href="https://www.google.com/calendar?tab=wc"
...
```

Crawl uses fetcher to recursively crawl pages starting with url, to a maximum of depth



WebCrawler

TODO:

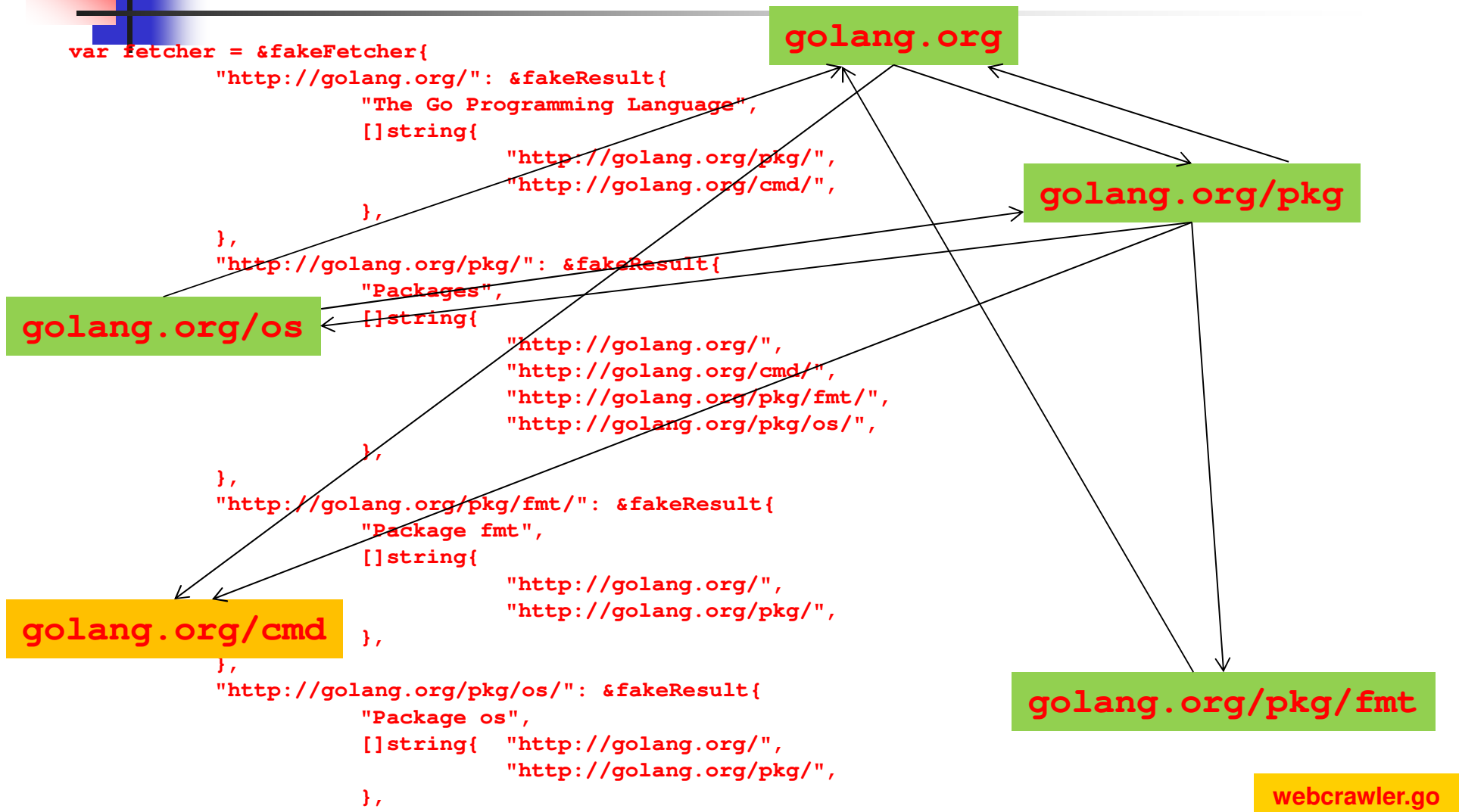
Fetch URLs in parallel.

Don't fetch the same URL twice.

```
func Crawl71(url string, depth int, fetcher Fetcher) {  
    if depth <= 0 {  
        return  
    }  
    body, urls, err := fetcher.Fetch(url)  
    if err != nil {  
        fmt.Println(err)  
        return  
    } // naivné prehľadávanie do hĺbky, bez kontroly  
    fmt.Printf("found: %s %q\n", url, body)  
    for _, u := range urls {  
        Crawl71(u, depth-1, fetcher)  
    }  
    return  
}
```

<https://tour.golang.org/concurrency/10>

WebCrawling



WebCrawlerR



```
// HashMap navštívených linkov
var visited = make(map[string]bool)

func CrawlR(url string, depth int, maxDepth int) {
    if depth <= maxDepth { // ak nie som príliš hlboko
        visited[url] = true // ok, bol som tu ...
        suburls := crawlPageR(url, depth) // získaj urls
        for _, url := range suburls.suburls { // prejdí ich
            if _, seen := visited[url]; seen { // ak si tam
                continue // bol, preskoč
            }
            CrawlR(url, depth+1, maxDepth) // inak rekurzia
        }
    }
}
```

[0:http://golang.org/] "The Go Programming Language"
[1:http://golang.org/pkg/] "Packages"
not found: http://golang.org/cmd/
[2:http://golang.org/pkg/fmt/] "Package fmt"
[2:http://golang.org/pkg/os/] "Package os"

WebCrawlerR



```
type Urls struct {
    depth int                // hĺbka podstránky od koreňa
    suburls []string         // zoznam linkov na nej
}

func crawlPageR(url string, depth int) *Urls {
    body, urls, err := fetcher.Fetch(url) // toto nemáme ☹️
    if err != nil {
        fmt.Println(err)
    } else {
        fmt.Printf("found[%d:%s] %q\n", depth, url, body)
    }
    return &Urls{depth + 1, urls}
}
```




WebCrawler 2

```
var (  
    // akonáhle prejdeme stránku s adresou url, všetky  
    // jej vnorené Urls zapíšeme ho do kanálu  
    globalQueueOfUrls = make(chan Urls)  
    totalRuns = 0      // počet spustení crawlPage  
                        // t.j. veľkosť fronty nespracovaných Urls  
    visited = make(map[string]bool) // navštívené urls  
)  
  
func crawlPage(url string, depth int) {  
    body, urls, err := fetcher.Fetch(url)  
    if err ...  
    fmt.Printf("[%d:%s] %q\n", depth, url, body)  
    globalQueueOfUrls <- Urls{depth + 1, urls}  
}
```



WebCrawler 2

size: 287
7.6544378s

```
func Crawl(url string, depth int) {  
    totalRuns++           // spracuj hlavnú stránku  
    visited[url] = true    // navštívili sme ju  
    go crawlPage(url, 0)   // pridaj jej Urls do fronty  
    for totalRuns > 0 {    // kým je niečo vo fronte  
        totalRuns--       // dekrementuj veľkosť fronty  
        next := <-globalQueueOfUrls // vyber z fronty  
        if next.depth > depth { continue } // prihlboko  
        for _, url := range next.suburls { // do hĺbky  
            if _, seen := visited[url]; seen { continue }  
            visited[url] = true  
            totalRuns++           // nerekurzívne spracuj  
            go crawlPage(url, next.depth) // podstránky  
        }  
    }  
}
```