김진욱 (rein@ncsoft.com)
개발 7실, SP 팀

GO PROGRAMMING LANGUAGE #4

Channel 관련 추가 사항

- msg <- ch // blocking read

- Non-block 처리?
  - msg, ready <- ch // polling
  - if ready { /* process msg */ }
    else { /* process other-things */ }

Network Programming Using Go

Basic Concepts

Issues

- Socket API
- I/O Multiplexing
- Synchronization
- …

Socket API: Package net

- Address Wrapprer
  - type Addr : network name + address string
  - IPAddr for IP
  - UnixAddr for Unix domain socket
  - TCPAddr for TCP connection
  - UDPAddr for (pseudo) UDP connection

Socket API: Package net

- Listener Wrapper (1/2)
  - Performs connection-oriented socket-listen
  - Listen(net, local-address-string)
  - Supports TCP or Unix domain socket
  - ex) tcp, tcp4, tcp6, unix

Socket API: Package net

- Listener Wrapper (2/2)
  - Performs datagram communication
  - ListenPacket(net, local-address-string)
  - Supports UDP or Unix domain socket(datagram)
  - ex) udp, udp4, udp6, unixgram

Socket API: Package net

- Connection Wrapper: interface Conn
  - IPConn, TCPConn, UDPConn
  - Performs common socket operations
  - Conn.Read, Write, Close
  - Conn.SetTimeout, SetRead(Write)Timeout
  - DialTCP or DialUDP to make connection(?)
  - ListenTCP or ListenUDP to accept connection(?)

I/O Multiplexing

- Multiple go-routines as worker threads
- Multiple Process with internal go-routines
- Select using range() construct over channels
- Distribute the work-load with remotely connected channels (package netchan)

Synchronization

- Channel
- Locks (mutex)
- 3rd party concurrent data-structure packages
- …

Network Programming Using Go

Re-implementing the troll

What Is Troll?

- Network Simulation(?) Tool for UDP
- Can make UDP datagram to be
  - Delayed,
  - Dropped,
  - Duplicated,
  - Or garbled
- See http://courses.engr.illinois.edu/ece435/Labs/Troll/troll.htm

Troll

- Simulates UDP datagrams between two nodes (ip, port pair)
- Delay, duplicate, modify, drop the datagram using predefined parameters
- Cannot use it as our test tools
  - Need a license for commercial use

Network Programming Using Go

Re-implementing the troll: Implementation

Idea

- Work as an UDP Proxy
- Drop, garble, duplicate the datagrams, based on pre-defined probability
- Each go-routine processes <src, dst> pair
- Each go-routine has internal datagram scheduler to delay the datagrams

Idea

- Accept an UDP Datagram from the source
- Scheduler chooses the datagram to send
- Delay is calculated using some probability function
- Based on some probability distribution, drop, duplicate or garble the packet

Create UDP Proxy (1/2)

```
srcAddr, err := net.ResolveUDPAddr(os.Args[1])
if err != nil {
    fmt.Println("Address resolution failed")
    return
}

src, err := net.ListenUDP("udp4", srcAddr)
if err != nil {
    fmt.Println("Listen error")
    return
}
```

Create UDP Proxy (2/2)

```
dstAddr, err := net.ResolveUDPAddr(os.Args[2])
if err != nil {
    fmt.Println("Address resolution failed")
    return
}

dst, err := net.DialUDP("udp4", nil, dstAddr)
if err != nil {
    fmt.Println("Connect error")
    return
}
```

Create UDP Proxy (2/2)

```
dstAddr, err := net.ResolveUDPAddr(os.Args[2])
if err != nil {
    fmt.Println("Address resolution failed")
    return
}

dst, err := net.DialUDP("udp4", nil, dstAddr)
if err != nil {
    fmt.Println("Connect error")
    return
}
```

Create Workers and Wait

```
for i := 0; i < 4; i++ {
    go worker(src, dst, mean)
}

ch := make(chan int)
<- ch
```

Heap for Scheduler (1/2)

- Package container/heap, interface heap
- Operations
  - Push(h heap, x interface{})
  - Pop(h heap) x interface{ }
- container/vector : need to implement Less() to be used as a heap interface

Heap for Scheduler (2/2)

```
type WorkItem struct {
    buffer [] byte
    deadline int64
}

type WorkItemHeap struct { vector.Vector }

func (h* WorkItemHeap) Less(i, j int) bool {
    return h.At(i).(WorkItem).deadline
< h.At(j).(WorkItem).deadline
}
```

Worker go-routine (1/3)

```
func worker(src, dst *net.UDPConn, delay float32) {
    var buffer [1024] byte
    h := new(WorkItemHeap)
```

Worker go-routine (1/3)

```
func worker(src, dst *net.UDPConn, delay float32) {
    var buffer [1024] byte
    h := new(WorkItemHeap)
```

Worker go-routine (2/3)

```
for {
       var timeout int64 = 1e9 // 1 sec
       var now int64 = time.Nanoseconds()
       for h.Len() > 0 { // timer expiration
           deadline := h.At(0).(WorkItem).deadline
           if deadline > now {
               timeout = deadline - now
               break
           } else { /* process the datagram */
       }
       src.SetReadTimeOut(timeout);
```

Worker go-routine (3/3)

```
        n, _, err := src.ReadFromUDP(buffer[0:1024])
        if err != nil {
            /* continue if timeout occurred */
        }
        heap.Push(h,*NewWorkItem(buffer[0:n],
                                    calcDelay(delay)))
    } // end of for
}
```

Process The Datagram

```
if checkDrop() { // 버리거나
    heap.Pop(h)
    continue
} else if checkModify() {
    modifyPacket(h.At(0).(WorkItem).buffer)
} else if checkDuplicate() {
    heap.Push(h,
      *NewWorkItem(h.At(0).(WorkItem).buffer,
       calcDelay(delay))
dst.Write(h.At(0).(WorkItem).buffer)
Heap.Pop(h)
```

Q & A