Distributed Communication 5st practice

Li Jianhao
lijianhao288@hotmail.com

1 Basics

1.1 CPU setting and Duration

CPU setting
package: "runtime"
Syntax: runtime.NumCPU()
Syntax: _ = runtime.GOMAXPROCS(< *Argument* >)

Time duration
package: "time"
Syntax:
start := time.Now()
...
duration := time.Since(start)
fmt.Println("Time: ", duration)

```
package main

import (
    "fmt"
    "sync"
    "time"
    "runtime"
)

func main(){
    fmt.Println(runtime.NumCPU())
    _ = runtime.GOMAXPROCS(8)

    start := time.Now()
    var mu sync.Mutex
    var wg sync.WaitGroup

    var result = 0

    for i := 0; i < 10000; i++ {
        wg.Add(1)
        go func (a int) {
            defer wg.Done()
```

```
            mu.Lock()
            result += a
            mu.Unlock()
        }(i)
    }

    wg.Wait()

    fmt.Println(result)
    duration := time.Since(start)
    fmt.Println("Time:␣", duration)
}
```

Listing 1: cpu

_ = runtime.GOMAXPROCS(8)

```
8
49995000
Time:  10.6265ms
```

_ = runtime.GOMAXPROCS(1)

```
8
49995000
Time:  41.1052ms
```

1.2 Channel

Syntax: $< ChannelName >$:= make(chan $< Type >$)
Syntax(Send): $< ChannelName > <- < Message >$
Syntax(Receive): $< Message > := <- < ChannelName >$

```
package main

import (
    "fmt"
    "sync"
)

func main() {

    var wg sync.WaitGroup

    c := make(chan string)

    wg.Add(1)
    go func() {
        defer wg.Done()
        s := "Hello"
        c <- s
        fmt.Println("g1␣sent␣",s)
    }()

    wg.Add(1)
```

```
    go func() {
        defer wg.Done()
        r := <- c
        fmt.Println("g2 received ", r)
    }()

    wg.Wait()
}
```

Listing 2: Channel without buffer

```
g1 sent  Hello
g2 received  Hello
```

1.3 Buffered Channel, Close, For range

Syntax: $< ChannelName >$:= make(chan $< Type >$, <BufferSize>)
Syntax: close($< ChannelName >$)
Syntax:
for $< iterator >$:= range $< ChannelName >$ {
    }

```
package main

import (
    "fmt"
    "sync"
    "time"
)

func main() {

    var wg sync.WaitGroup

    c := make(chan int,5)

    wg.Add(1)
    go func() {
        defer wg.Done()
        for i:=0;i<10;i++{
            c<-i
            fmt.Println("g1 sent ",i)
        }
        close(c)
    }()

    wg.Add(1)
    go func() {
        defer wg.Done()
        time.Sleep(5*time.Second)
        for r := range c {
            fmt.Println("g2 received ", r)
        }
```

```
    }()

    wg.Wait()
}
```

Listing 3: Channel with buffer

```
g1 sent  0
g1 sent  1
g1 sent  2
g1 sent  3
g1 sent  4
(After a few seconds)
g2 received  0
g2 received  1
g2 received  2
g2 received  3
g2 received  4
g2 received  5
g1 sent  5
g1 sent  6
g1 sent  7
g1 sent  8
g1 sent  9
g2 received  6
g2 received  7
g2 received  8
g2 received  9
```

2 Practice

2.1 p1

Create a channel of string named c. This channel does not have a buffer.

The main function starts a new goroutine, let's call it g1. g1 send a "Hello" to channel c.

Then the main function tries to receive a message from channel c and print it out.

2.2 p2

Create a channel of int named c. This channel's buffer size is 5.

Start two new goroutines try to send integers 0-9 to channel c and print out "g1(or g2) sent n". Let's call them g1 and g2. The g2 will

wait until the g1 finishes its sending. The g2 will close the channel after it finishes sending.

After creating those two goroutines, the main goroutine first sleeps 2 seconds. Then start to use a for range loop to receive and print out the messages from channel c. The duration of the message receiving is measured and printout.