

Cloud Computing and Cloud Native Systems

ECE 573

Homework 3

Abhilash Kashyap Balasubramanyam

abalasubramanyam@hawk.iit.edu

A20566944

Fall 2024

Illinois Institute of Technology

Date: October 3, 2024

Table of Contents

Solutions.....	1
1 Ping Pong Question	1
1.1 Original Code.....	1
1.2 Output.....	2
2 The desired output through Step 1	2
2.1 Problem Statement - Step 1	2
2.2 Edited code for Step 1.....	3
2.3 Output.....	4
3 The desired output through Step 2	4
3.1 Problem Statement - Step 2	4
3.2 Edited Code for Step 2.....	4
3.3 Output.....	5
4 Explanation	6

List of Code Snippets

Code 1: Given Code from the question.	1
Code 2: Edit to the code made Problem Statement - Step 1 instructions.	3
Code 3: Edited code after Problem Statement - Step 2 instruction.	5

List of Figures

Figure 1: Raw un-edited program output.	2
Figure 2: First edit to the program based on Problem Statement - Step 1 instructions.	4
Figure 3: Final edit to the program based on Problem Statement - Step 2 instructions.	6

Solutions

1 Ping Pong Question

Consider the following Go code where two goroutines ping and pong are supposed to output “ping” and “pong” alternatively by collaborating via channels.

1.1 Original Code

```
package main

import (
    "fmt"
)

func ping(in <-chan bool, out chan<- int, n int) {
    for i := 0; i < n; i++ {
        <-in // wait for signal from pong or start
        fmt.Printf("ping %d\n", i)
        out <- i // let pong do its job
    }
    close(out) // notify pong of done
}

func pong(in <-chan int, out chan<- bool, done chan<- struct{}) {
    for i := range in { // get i from ping
        fmt.Printf("pong %d\n", i)
        out <- false // let ping do its job
    }
    close(done) // notify main of done
}

func main() {
    pi := make(chan bool) // line B
    po := make(chan int)
    done := make(chan struct{})
    defer close(pi)

    go ping(pi, po, 10)

    go pong(po, pi, done)

    fmt.Println("Start!")
    // line A

    <-done
}
```

Code 1: Given Code from the question.

The above Code 1 represents the original code given in the problem statement.

1.2 Output

The immediate output derived from running the raw program given in the problem statement is as given below in the Figure 1. It is observed that the program doesn't behave as intended and doesn't produce the necessary expected output.

A screenshot of a debugger's output window. The window has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The DEBUG CONSOLE tab is active, showing the following text: "Detaching and terminating target process", "dlv dap (1968) exited with code: 0", "Starting: C:\Users\Abhilash\go\bin\dlv.exe dap --listen=127.0.0.1:52236 from c:\Users\Abhilash\Documents\ECE 573\HW_3", "DAP server listening at: 127.0.0.1:52236", "Type 'dlv help' for list of commands.", and "Start!".

```
PROBLEMS 333 OUTPUT DEBUG CONSOLE TERMINAL PORTS
Detaching and terminating target process
dlv dap (1968) exited with code: 0
Starting: C:\Users\Abhilash\go\bin\dlv.exe dap --listen=127.0.0.1:52236 from c:\Users\Abhilash\Documents\ECE 573\HW_3
DAP server listening at: 127.0.0.1:52236
Type 'dlv help' for list of commands.
Start!
```

Figure 1: Raw un-edited program output.

2 The desired output through Step 1

2.1 Problem Statement - Step 1

The desired output should be

```
Start!
ping 0
pong 0
ping 1
pong 1
ping 2
pong 2
ping 3
pong 3
ping 4
pong 4
ping 5
pong 5
ping 6
pong 6
ping 7
pong 7
ping 8
pong 8
ping 9
pong 9
```

However, go reports a deadlock after displaying "Start!". Add a line at line A

so that ping and pong will start to output messages as desired. (Hint: send something to ping!)

2.2 Edited code for Step 1

```
package main

import (
    "fmt"
)

func ping(in <-chan bool, out chan<- int, n int) {
    for i := 0; i < n; i++ {
        <-in // wait for signal from pong or start
        fmt.Printf("ping %d\n", i)
        out <- i // let pong do its job
    }
    close(out) // notify pong of done
}

func pong(in <-chan int, out chan<- bool, done chan<- struct{}) {
    for i := range in { // get i from ping
        fmt.Printf("pong %d\n", i)
        out <- false // let ping do its job
    }
    close(done) // notify main of done
}

func main() {
    pi := make(chan bool) // line B
    po := make(chan int)
    done := make(chan struct{})
    defer close(pi)

    go ping(pi, po, 10)

    go pong(po, pi, done)

    fmt.Println("Start!")
    pi <- true // line A

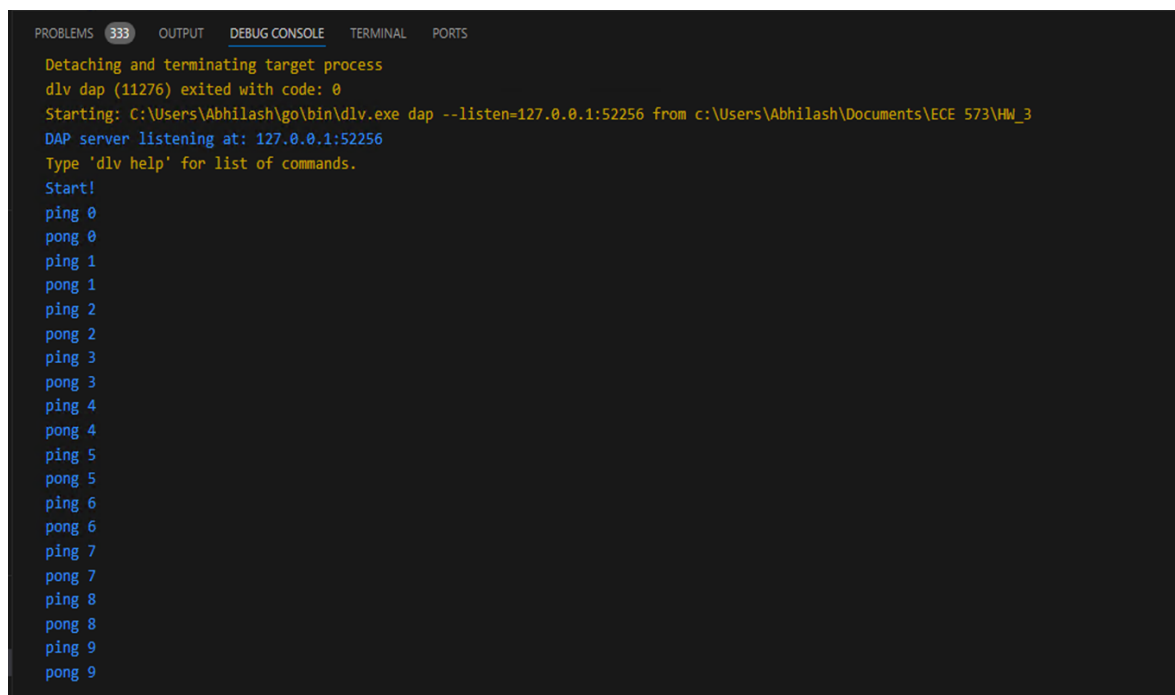
    <-done
}
```

Code 2: Edit to the code made Problem Statement - Step 1 instructions.

The code snippet present above in Code 2 represents the edits made to the original code which fixed part of the problem as suggested by the problem statement.

2.3 Output

The edited output from the first step of the process in editing the original program given in the problem statement is as given below in the Figure 2. It is observed that the program behaves as per the expectations set by The desired output through Step 1.



```
PROBLEMS 333 OUTPUT DEBUG CONSOLE TERMINAL PORTS
Detaching and terminating target process
dlv dap (11276) exited with code: 0
Starting: C:\Users\Abhilash\go\bin\dlv.exe dap --listen=127.0.0.1:52256 from c:\Users\Abhilash\Documents\ECE 573\HW_3
DAP server listening at: 127.0.0.1:52256
Type 'dlv help' for list of commands.
Start!
ping 0
pong 0
ping 1
pong 1
ping 2
pong 2
ping 3
pong 3
ping 4
pong 4
ping 5
pong 5
ping 6
pong 6
ping 7
pong 7
ping 8
pong 8
ping 9
pong 9
```

Figure 2: First edit to the program based on Problem Statement - Step 1 instructions.

3 The desired output through Step 2

3.1 Problem Statement - Step 2

Still, go reports a deadlock after displaying all desired messages. Modify line B to resolve the issue. Explain why. (Hint: what if you send something to a channel but no one is receiving?)

3.2 Edited Code for Step 2

The code snippet present above in Code 3 represents the secondary edits made to the Edited code for Step 1 which fixed the entire problem as suggested by the problem statement.

```

package main

import (
    "fmt"
)

func ping(in <-chan bool, out chan<- int, n int) {
    for i := 0; i < n; i++ {
        <-in // wait for signal from pong or start
        fmt.Printf("ping %d\n", i)
        out <- i // let pong do its job
    }
    close(out) // notify pong of done
}

func pong(in <-chan int, out chan<- bool, done chan<- struct{}) {
    for i := range in { // get i from ping
        fmt.Printf("pong %d\n", i)
        out <- false // let ping do its job
    }
    close(done) // notify main of done
}

func main() {
    pi := make(chan bool, 1) // line B
    po := make(chan int)
    done := make(chan struct{})
    defer close(pi)

    go ping(pi, po, 10)

    go pong(po, pi, done)

    fmt.Println("Start!")
    pi <- true // line A

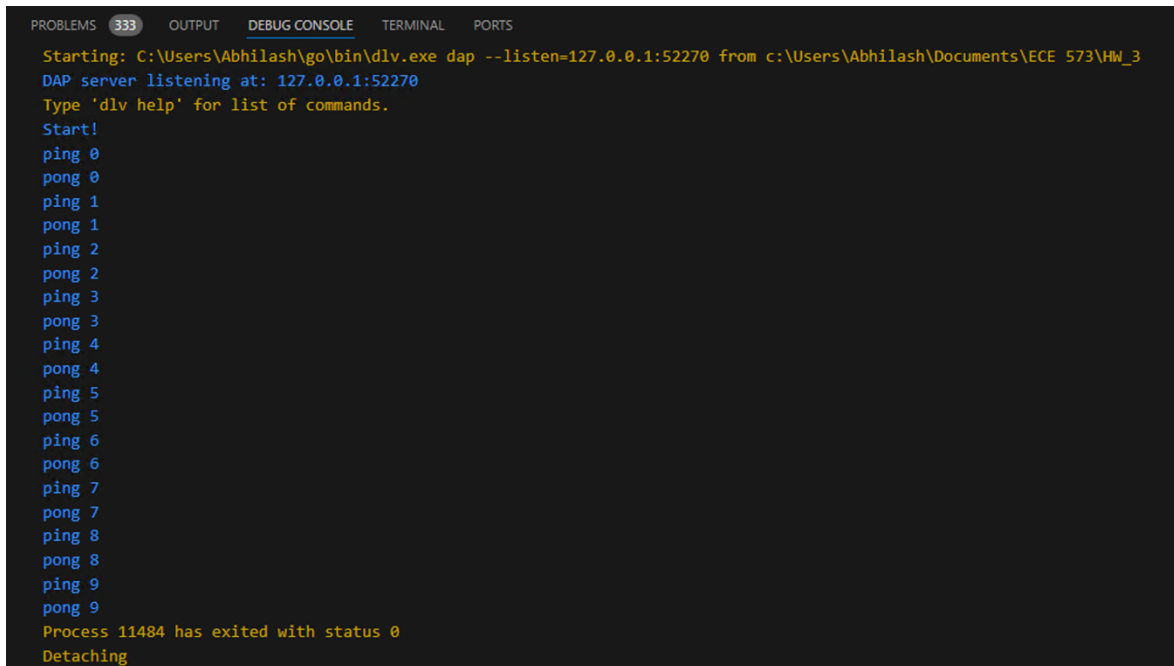
    <-done
}

```

Code 3: Edited code after Problem Statement - Step 2 instruction.

3.3 Output

The edited output from the final step of the process in editing the original program given in the problem statement is as given below in the Figure 3. It is observed that the program behaves thoroughly as per the expectations set by The desired output through Step 2, thus solving the entire problem.



```
PROBLEMS 333 OUTPUT DEBUG CONSOLE TERMINAL PORTS
Starting: C:\Users\Abhilash\go\bin\dlv.exe dap --listen=127.0.0.1:52270 from c:\Users\Abhilash\Documents\ECE 573\HW_3
DAP server listening at: 127.0.0.1:52270
Type 'dlv help' for list of commands.
Start!
ping 0
pong 0
ping 1
pong 1
ping 2
pong 2
ping 3
pong 3
ping 4
pong 4
ping 5
pong 5
ping 6
pong 6
ping 7
pong 7
ping 8
pong 8
ping 9
pong 9
Process 11484 has exited with status 0
Detaching
```

Figure 3: Final edit to the program based on Problem Statement - Step 2 instructions.

4 Explanation

The pong go-routine tries to broadcast a value on the pi channel (`out <- false`) after the final pong message is displayed, but there is no recipient for this value, which results in a deadlock. This is because the out channel (`po`) has already been closed by the ping go-routine after completing its iterations.

The pong go-routine may communicate its final value without blocking, even in the absence of a recipient, by creating a buffered channel with a capacity of 1, which we call `pi`. This breaks the impasse.

A predetermined number of values can be stored on a buffered channel before the transmit process stops. In this instance, a buffer size of 1 is plenty to keep the intended behavior in place and avoid the deadlock.