Exam

Li Jianhao
lijianhao288@hotmail.com

1 Exam

Each solution should be a file or a few separate files. Send the zip file of all the program files to my email in 90min after the exam begins. (The zip file's name should be your NEPTUN code)

1.1 p1

Create a struct **request**. Its fields: first (type int), second (type int).

Create a function **product** which takes a **request** and return the product of its **first** and **second**.

In the main function, create a typed value of the request **r1** with arguments 2 and 3. Call the function **product** and pass **r1** as the argument. Print out the result.

1.2 p2

Declare an uint64 **receivedCount**. Declare a channel of int **c**.

The main function creates two goroutines (We call it g1 and g2).

g1 send 0 till 9 to the channel **c**. For each sending, g1 will print out "g1 sent $< n >$". After sending the numbers, g1 do an operation to indicate that no more values will be sent to **c**.

g2 use a for range to receive messages from the channel **c**. For each received message, g2 print "g2 received $< n >$" and add 1 to the **receivedCount atomically** with the function in package "sync/atomic".

The main function **wait** for g1 and g2 and print out "Received count:" and **receivedCount**.

1.3 p3

Create 1 publisher and 2 consumers (**solutionIncrease** and **solutionDouble**). They use a fanout exchange with the name **p3fanoutExchange**.

1. The publisher sends numbers 0-10 (0 and 10 are included) to **p3fanoutExchange**. It will send an "END" after all.

2. Each consumer binds their private queue to the **p3fanoutExchange**. The consumer name (or ConsumerTag) of **solutionIncrease** is "Inc". The consumer name (or ConsumerTag) of **solutionDouble** is "Double". Each consumer uses 3 goroutines to receive messages from the Golang channel parallelly. If the received message is "END", the consumer is canceled. The main goroutine will wait until all those 3 goroutines finish and print "Finish" before terminate. The **solutionIncrease** increase the received number by 1 and prints out "Received: $< i >$ Result: $< i+1 >$". The **solutionDouble** double the received number and print out "Received: $< i >$ Result: $< i * 2 >$".

1.4 p4

Create 2 servers (**IntDoubler** and **StringDuplicator**) and 2 clients. The only difference of the clients is that they send different jobs. The client1 send strings: "123", "abc", "abb", "ccd", "456". The client2 send strings: "aaa", "bbb", "777", "ccc", "999". Create a direct exchange with the name **p4Exchange**.

1. The servers receive the jobs in a balanced round-robin way. For each received message, the server process it, get a result, send back the result to the related client, and print out "Received job: $< job >$ Published response: $< response >$". Different server process the messages in different ways. The server **IntDoubler** receive integer strings from a shared queue **intQueue** and double it("123"$- >$"246"). The server **StringDuplicator** receive strings from a shared queue **stringQueue** and duplicate it("ab"$- >$"abab").

2. The clients publish messages to **p4Exchange** and print out "Published job: $< msg >$". Before the clients publish messages, it will check if the message is a number by try to convert it to integer. If it is integer, the client send it to **intQueue** (With publishing routing key "int"). If it is not integer, the client send it to **stringQueue** (With publishing routing key "string"). The jobs will be processed by the servers. Each client will receive their own responses. For each received response message, if the response is related to the job sent by this client, this client will print out "Job: $< job >$ Got response: $< response >$". If not, print out "Got a not related msg"