# End-term Exam

Li Jianhao
lijianhao288@hotmail.com

# 1 Exam

Each problem in a few separate files. Send the zip of all the program files to lijianhao@inf.elte.hu before 10 am.

## 1.1 p1

Create 1 publisher and 1 consumer. They use a direct exchange with the name **p1Exchange**. The publisher sends a string "job". The consumer bind queue **jobQueue** to **p1Exchange** with the key "p1". The consumer convert the received string to upper case and print out "Received: "job" Result:< *result* >".

## 1.2 p2

Create 1 publisher and 2 consumers. They use a direct exchange with the name **p2Dexchange** and a fanout exchange with the name **p2Fexchange**.

1. The publisher sends two private messages to **p2Dexchange**(one for consumer1, another for consumer2). Each consumer will receive their own private message from their own private queue. The bounding routing key of consumer1's private queue is p2key1. The bounding routing key of consumer1's private queue is p2key2. For each received message, the consumer print out "Received private message: " and the message.

2. The publisher sends two broadcast messages to **p2Dexchange** with publishing routing key "bc". All the consumers will receive

all the broadcast messages. For each received message, the consumer print out "Received direct broadcast message: " and the message.

3. The publisher sends two more broadcast messages to **p2Fexchange**. All the consumers will receive all the broadcast messages. For each received message, the consumer print out "Received fanout broadcast message: " and the message.

4. The publisher sends 5 tasks messages to **p2Dexchange** with publishing routing key "p2task". The consumers receive the tasks in a balanced round-robin way from a shared queue **p2SharedQueue**. For each received message, the consumer print out "Received task message: " and the message.

## 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"