НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра прикладної математики

Звіт

із лабораторної роботи № 4

із дисципліни «Інформаційні системи»

на тему:

«Знайомство із принципом роботи брокера повідомлень»

|  |  |
| --- | --- |
| Виконала: | Керівник: |
| студент групи КМ-01  Боженко А. О. | *Саяпіна . І. О.* |

Київ – 2023

Зміст

[МЕТА 2](#_Toc151578033)

[Основна частина 3](#_Toc151578034)

[Завдання 1 3](#_Toc151578035)

[Завдання 2: 4](#_Toc151578036)

[Завдання 3: 6](#_Toc151578037)

[Висновки 8](#_Toc151578038)

[Додаток A. Скрипти програм 9](#_Toc151578039)

[Завдання 1.Publisher 9](#_Toc151578040)

[Завдання 1.Consumer 9](#_Toc151578041)

[Завдання 2.Publisher 10](#_Toc151578042)

[Завдання 2.Consumer 11](#_Toc151578043)

[Завдання 3.Publisher 11](#_Toc151578044)

[Завдання 3. Consumer 12](#_Toc151578045)

# МЕТА

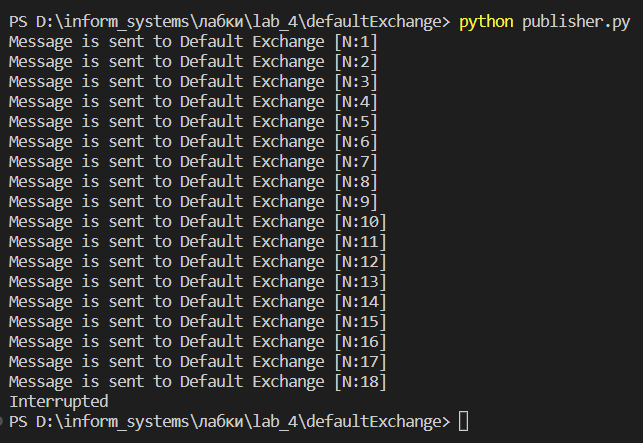
Дослідити організацію асинхронного режиму обміну повідомлень на основі роботи з брокером повідомлень RabbitMQ.

# Основна частина

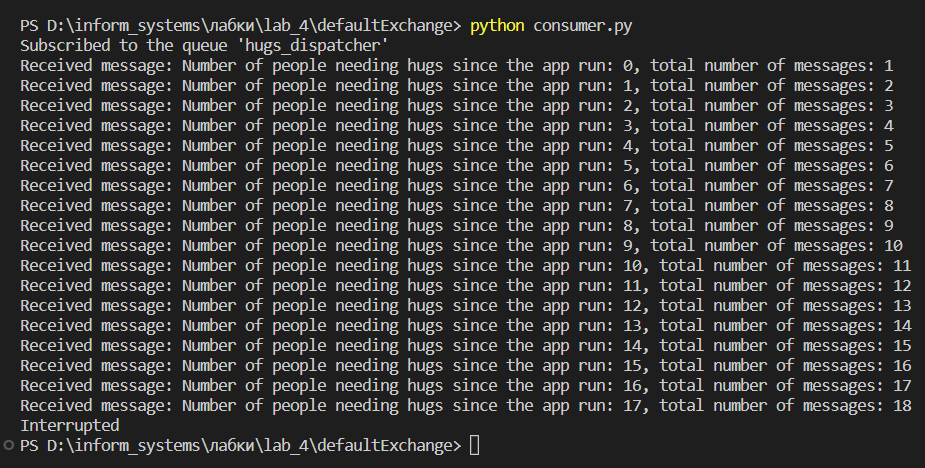
Завдання 1

1. Змініть код Producer`а, щоб він генерував інший текст повідомлень за Вашим   
   вибором.   
   2. Змініть назву черги повідомлень як у коді Producer`а , так і в коді Cоnsumer`а та   
   переконайтеся, що вони використовують однакову назву черги.   
   3. Змініть код Producer`а, щоб генерувати повідомлення з іншим інтервалом.   
   4. Змініть код Cоnsumer`а, щоб роздрукувати кількість повідомлень, які він   
   отримав, на додаток до вмісту повідомлення.   
      
   6. За результатами роботи додайте до звіту код Producer`а, Cоnsumer`а та скріни 10-  
   15 повідомлень, що вони виводять.

Виконання:



Мал 1.1 – скриншот тестування роботи Publisher’a



Мал 1.2 – скриншот тестування роботи Consumer’a

Зміни:

1. Повідомлення Publisher’a змінено з

“Message from publisher N: [counter]”

На

“Number of people needing hugs since the app run: [counter]”

1. Назва черги перйменована з «dev-queue» на «hugs\_dispatcher»
2. Інтервал затримки генерування повідомлення змінено з [1, 3] до [2, 5] секунд
3. Додано до Consumer’а змінну counter, що у функції callback збільшується й демонструємо в стандартному потоці (command line), при отриманні повідомлення від Publisher’a

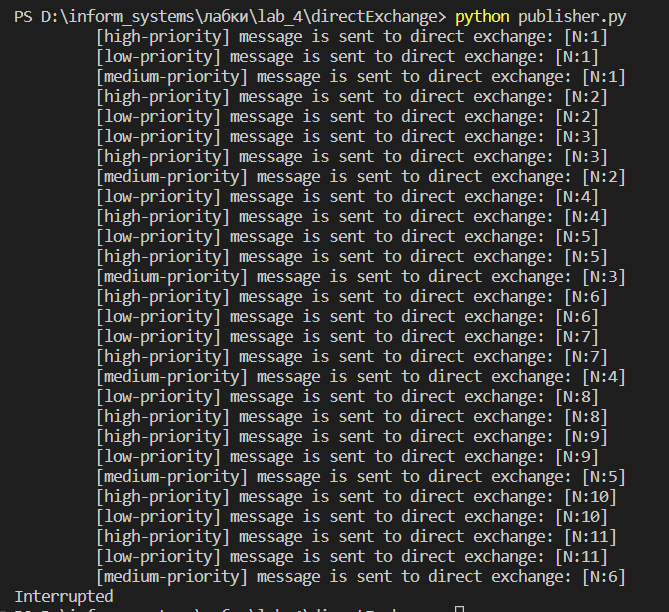
Завдання 2:

1. Відповідно до варіанту створити необхідні обмінники (exchange), черги (queue) та   
   зв'язки (binding).   
   2. Навести код Сonsumer'ів та Publisher'а, який дозволяє перевірити правильність   
   налаштувань.   
   3. Навести скрін з 15-20 повідомленнями, відправленими Publisher`ом та відповідних   
   отриманих повідомлень Сonsumer'ів.

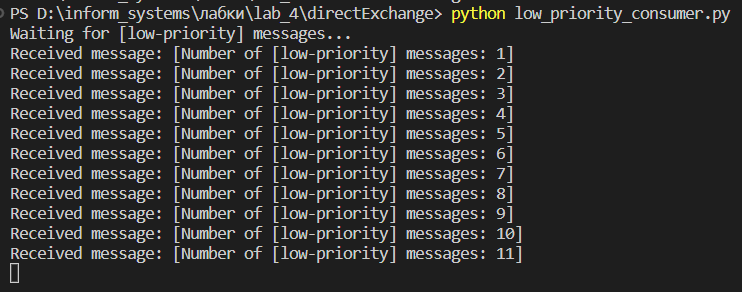
Варіант 3:

Створіть прямий обмін із трьома прив’язаними до нього чергами.   
Перша черга повинна отримувати повідомлення з ключем маршрутизації "high-  
priority", друга черга повинна отримувати повідомлення з ключем   
маршрутизації "medium-priority", третя черга повинна отримувати   
повідомлення з ключем маршрутизації "low-priority".

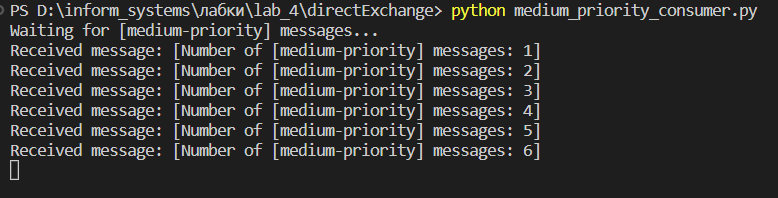
Виконання:



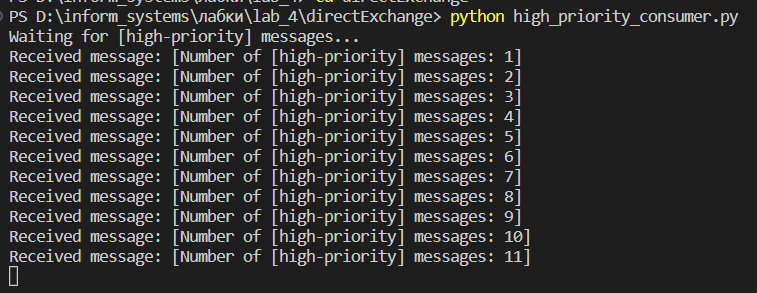
Мал. 2.1 – скриншот тестування Publisher’a



Мал. 2.2 – скриншот тестування Consumer’a черги low-priority



Мал. 2.3 – скриншот тестування Consumer’a черги medium-priority



Мал. 2.4 – скриншот тестування Consumer’a черги high-priority

Завдання 3:

1. Відповідно до варіанту створити необхідні обмінники (exchange), черги (queue) та   
   зв'язки (binding).   
   2. Навести код Сonsumer'ів та Publisher'а, який дозволяє перевірити правильність   
   налаштувань.   
   3. Навести скрін з 15-20 повідомленнями, відправленими Publisher`ом та відповідних   
   отриманих повідомлень Сonsumer'ів.

Варіант 3:

Створіть обмінник для програми обміну повідомленнями, де ключі   
маршрутизації мають формат «message.<recipient\_id>.<message\_type>», де   
recipient\_id — це унікальний ідентифікатор користувача, який є одержувачем   
повідомлення, а message\_type може бути текстом , зображення чи відео.   
Створіть окремі черги, які підписуються на обмінник для кожного одержувача,   
щоб отримувати відповідні повідомлення.

Виконання:

Нехай є 3 Consumer’и:

user\_1

user\_2

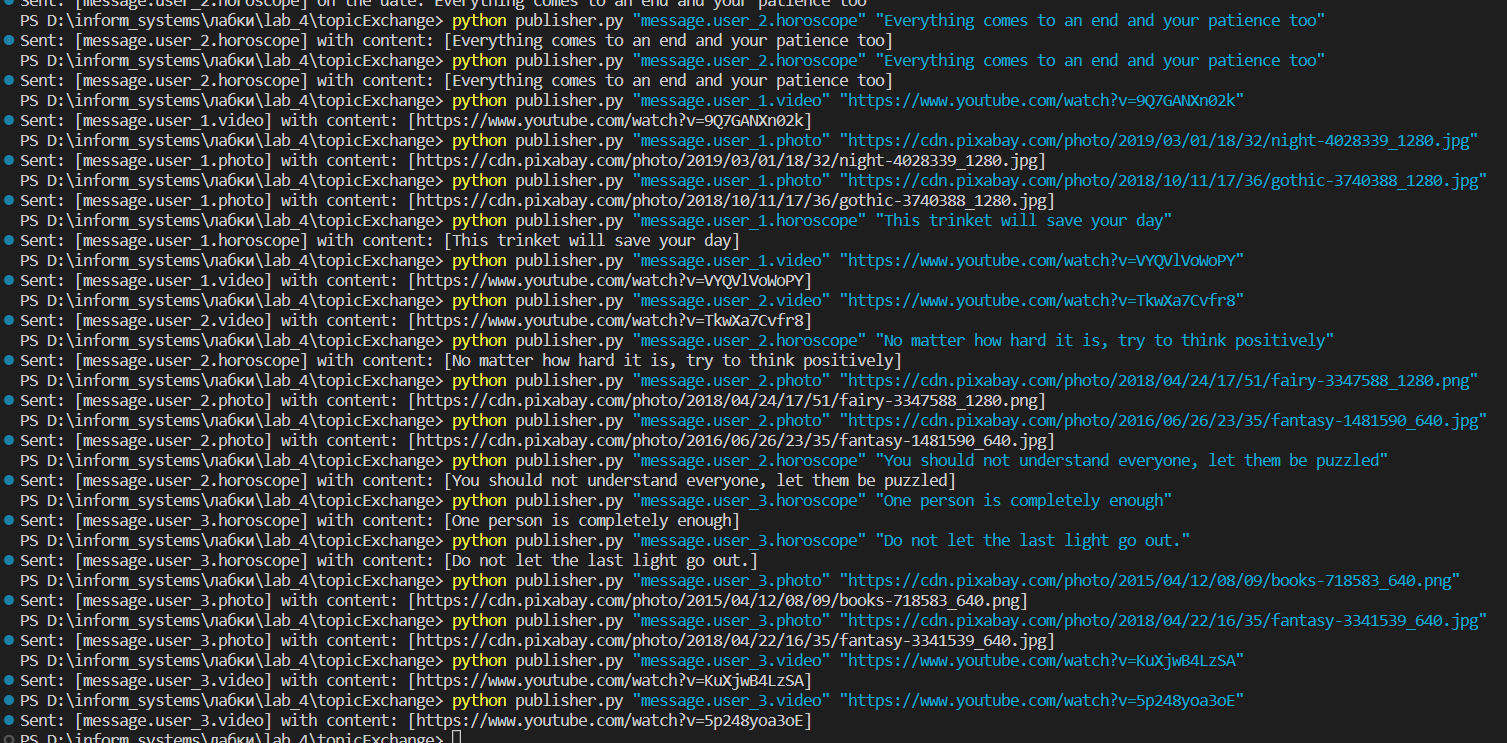
user\_3

Є 3 типи повідомлень:

horoscope – текстове

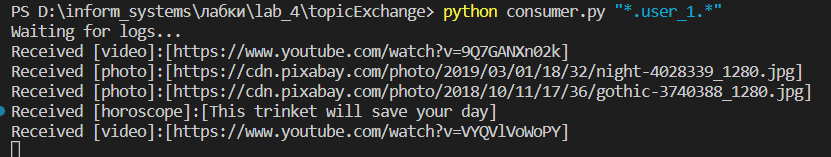
photo – гіперпосилання на фото

Video - uіперпосилання на відео

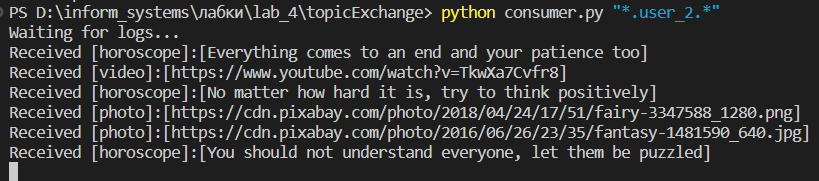


Мал. 3.1 – скриншот тестування Publisher’а,

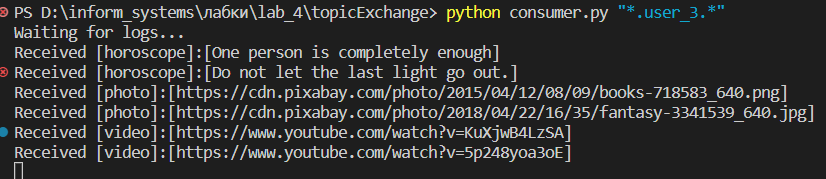
надсилання різним cоnsumer’ам повідомлень



Мал. 3.2 – скриншот тестування Consmer’а, що підписаний на чергу user\_1



Мал. 3.2 – скриншот тестування Consmer’а, що підписаний на чергу user\_2



Мал. 3.2 – скриншот тестування Consmer’а, що підписаний на чергу user\_3

# Висновки

Реалізовано 3 типи exchange подій у брокері повідомлень RabbitMQ: default (1 завдання), direct (2 завдання), topic (3 завдання). Скрипти publisher’ів та Consumer’ів реалізовано мовою python, із застосуванням бібліотеки-клієнта pika. Програми елементарні, у формі консольних застосунків.

# Додаток A. Скрипти програм

Завдання 1.Publisher

import pika

from random import randint

from time import sleep

import os

import sys

IP\_ADDRESS = "localhost"

QUEUE = "hugs\_dispatcher"

MIN\_TIME\_TO\_SLEEP = 2

MAX\_TIME\_TO\_SLEEP = 5

def main():

    counter = 0

    while (True):

        time\_to\_sleep = randint(MIN\_TIME\_TO\_SLEEP, MAX\_TIME\_TO\_SLEEP)

        sleep(time\_to\_sleep)

        # set up connection to broker

        connection = pika.BlockingConnection(pika.ConnectionParameters(IP\_ADDRESS))

        # set up channel

        channel = connection.channel()

        # set up queue

        channel.queue\_declare(queue=QUEUE,

                              durable=False,

                              exclusive=False,

                              auto\_delete=False)

        message = f"Number of people needing hugs since the app run: {counter}"

        counter += 1

        body = bytes(message, encoding="utf-8")

        channel.basic\_publish(exchange="",

                              routing\_key=QUEUE,

                              body=body)

        print(f"Message is sent to Default Exchange [N:{counter}]")

if \_\_name\_\_ == "\_\_main\_\_":

    try:

        main()

    except KeyboardInterrupt:

        print('Interrupted')

        try:

            sys.exit(0)

        except SystemError:

            os.\_exit(0)

Завдання 1.Consumer

import pika

import sys

import os

IP\_ADDRESS = "localhost"

QUEUE = "hugs\_dispatcher"

MIN\_TIME\_TO\_SLEEP = 2

MAX\_TIME\_TO\_SLEEP = 5

def main():

    global counter

    # set up connection to broker

    connection = pika.BlockingConnection(pika.ConnectionParameters(IP\_ADDRESS))

    # set up channel

    channel = connection.channel()

    # set up queue

    channel.queue\_declare(queue=QUEUE,

                          durable=False,

                          exclusive=False,

                          auto\_delete=False)

    def callback(ch, method, properties, body):

        global counter

        message = body.decode("utf-8")

        counter += 1

        print(f"Received message: {message}, total number of messages: {counter}")

    channel.basic\_consume(queue=QUEUE,

                          on\_message\_callback=callback,

                          auto\_ack=True)

    print(f"Subscribed to the queue '{QUEUE}'")

    channel.start\_consuming()

if \_\_name\_\_ == "\_\_main\_\_":

    counter = 0

    try:

        main()

    except KeyboardInterrupt:

        print('Interrupted')

        try:

            sys.exit(0)

        except SystemError:

            os.\_exit(0)

Завдання 2.Publisher

import pika

import sys

import os

from time import sleep

from random import randint

import threading

# priorities:

#

# high-priority

# medium-priority

# low-priority

IP\_ADDRESS = "localhost"

EXCHANGE = "direct\_logs"

EXCHANGE\_TYPE = "direct"

MIN\_TIME\_TO\_WAIT = 1

priorities = {"high-priority": 2,

              "medium-priority": 4,

              "low-priority": 6}

stop\_flag = False

def main(priority\_delay, priority\_name):

    counter = 0

    while not stop\_flag:

        sleep(priority\_delay)

        connection = pika.BlockingConnection(pika.ConnectionParameters(host=IP\_ADDRESS))

        channel = connection.channel()

        channel.exchange\_declare(exchange=EXCHANGE,

                                 exchange\_type=EXCHANGE\_TYPE)

        counter += 1

        message = f"Number of [{priority\_name}] messages: {counter}"

        body = bytes(message, encoding="utf-8")

        channel.basic\_publish(exchange=EXCHANGE,

                              routing\_key=priority\_name,

                              body=body)

        print(f"\t [{priority\_name}] message is sent to direct exchange: [N:{counter}]")

if \_\_name\_\_ == "\_\_main\_\_":

    threads = []

    try:

        for priority, max\_time\_to\_wait in priorities.items():

            time\_to\_wait = randint(MIN\_TIME\_TO\_WAIT, max\_time\_to\_wait)

            threads.append(threading.Thread(target=main, args=(time\_to\_wait, priority)))

        [t.start() for t in threads]

        # Wait for the user to press Ctrl+C

        while True:

            sleep(1)

    except KeyboardInterrupt:

        stop\_flag = True

        [t.join() for t in threads]

        print('Interrupted')

        try:

            sys.exit(0)

        except SystemError:

            os.\_exit(0)

Завдання 2.Consumer

Наведено приклад consumer’а, що підписаний на чергу з ключем-маршрутизатором “high-pririty”, але скрипти для інших черг аналогічні, з відмінністю значення змінної priority\_name.

import pika

import sys

import os

IP\_ADDRESS = "localhost"

EXCHANGE = "direct\_logs"

EXCHANGE\_TYPE = "direct"

priority\_name = "high-priority"

def main():

    connection = pika.BlockingConnection(pika.ConnectionParameters(host=IP\_ADDRESS))

    channel = connection.channel()

    channel.exchange\_declare(exchange=EXCHANGE,

                             exchange\_type=EXCHANGE\_TYPE)

    result = channel.queue\_declare(queue="", exclusive=True)

    queue\_name = result.method.queue

    channel.queue\_bind(queue=queue\_name,

                       exchange=EXCHANGE,

                       routing\_key=priority\_name)

    def callback(ch, method, properties, body):

        message = body.decode("utf-8")

        print(f"Received message: [{message}]")

    channel.basic\_consume(queue=queue\_name,

                          on\_message\_callback=callback,

                          auto\_ack=True)

    print(f"Waiting for [{priority\_name}] messages...")

    channel.start\_consuming()

if \_\_name\_\_ == "\_\_main\_\_":

    try:

        main()

    except KeyboardInterrupt:

        print("Interrupted")

        try:

            sys.exit(0)

        except SystemError:

            os.\_exit(0)

Завдання 3.Publisher

import pika

import sys

from datetime import date

IP\_ADDRESS = "localhost"

EXCHANGE = "topic\_logs"

EXCHANGE\_TYPE = "topic"

def main():

    connection = pika.BlockingConnection(

        pika.ConnectionParameters(host=IP\_ADDRESS))

    channel = connection.channel()

    channel.exchange\_declare(exchange=EXCHANGE, exchange\_type=EXCHANGE\_TYPE)

    routing\_key = sys.argv[1] if len(sys.argv) > 1 else "anonimous.info"

    message = " ".join(sys.argv[2:]) if len(sys.argv) > 2 else "No messages yet, take a hug instead!"

    body = bytes(message, encoding="utf-8")

    channel.basic\_publish(exchange=EXCHANGE,

                          routing\_key=routing\_key,

                          body=body)

    print(f"Sent: [{routing\_key}] with content: [{message}]")

    channel.close()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Завдання 3. Consumer

import pika

import sys

import os

IP\_ADDRESS = "localhost"

EXCHANGE = "topic\_logs"

EXCHANGE\_TYPE = "topic"

def main():

    connection = pika.BlockingConnection(

        pika.ConnectionParameters(host='localhost'))

    channel = connection.channel()

    channel.exchange\_declare(exchange=EXCHANGE, exchange\_type=EXCHANGE\_TYPE)

    result = channel.queue\_declare(queue="", exclusive=True)

    queue\_name = result.method.queue

    binding\_keys = sys.argv[1:]

    for b\_k in binding\_keys:

        channel.queue\_bind(queue=queue\_name,

                           exchange=EXCHANGE,

                           routing\_key=b\_k)

    print("Waiting for logs...")

    def callback(ch, method, properties, body):

        message\_type = method.routing\_key.split(".")[-1]

        body = body.decode("utf-8")

        print(f"Received [{message\_type}]:[{body}]")

    channel.basic\_consume(queue=queue\_name,

                          on\_message\_callback=callback,

                          auto\_ack=True)

    channel.start\_consuming()

if \_\_name\_\_ == "\_\_main\_\_":

    try:

        main()

    except KeyboardInterrupt:

        print('Interrupted')

        try:

            sys.exit(0)

        except SystemError:

            os.\_exit(0)