МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический
университет Петра Великого»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 Математика и компьютерные науки

Отчет по дисциплине «Сети ЭВМ и телекоммуникации компьютерных сетей» Лабораторная работа №3 «Организация межпроцессорного взаимодействия с помощью очереди сообщений RabbitMQ»

Студент,
группа 5130201/20102 _____ Шклярова К. А.

Преподаватель _____ Мулюха В.А.

«_____»_____ 2025 г.

Санкт-Петербург, 2025

1 Постановка задачи

Используя Docker создать контейнеры, необходимые для реализации следующего функционала с использованием RabbitMQ, а также показать, как именно осуществляется передача в этих условиях.

12 вариант: Организовать барьерную синхронизацию работы трех процессов, объяснить ее работу.

Барьерная синхронизация — механизм, при котором несколько процессов/потоков должны достичь определенной точки выполнения (барьера), прежде чем продолжить выполнение дальше. Пока все процессы не достигли барьера, ни один из них не продолжает выполнение.

1.1 Описание RabbitMQ

Брокер сообщений — программное обеспечение, которое позволяет выстроить действия в информационных системах таким образом, чтобы обеспечить асинхронный обмен сообщениями между сервисами.

Асинхронный обмен предполагает отправку запроса или сообщения от одного сервиса к другому, при этом деятельность сервиса-отправителя не приостанавливается в ожидании ответа от получателя.

Для обеспечения асинхронной доставки сообщений используется специальное программное обеспечение — брокер сообщений.

RabbitMQ - брокер сообщений с открытым исходным кодом, который реализует протокол AMQP.

RabbitMQ состоит из нескольких основных компонентов:

• Queue - структура данных, где хранятся сообщения до того, как получатель их обработает.

• Message - единица данных, которая передается от отправителя (Producer) к получателю (Consumer) через очередь.

• Exchange - обменник. Компонент, который осуществляет маршрутизациию, т.е. распределение данных по очередям на основе binding.

• Binding - связь между обменником и очередью, которая определяет, какие сообщения в какую очередь будут направлены.

• Producer - сервис, отправляющий данные.

• Consumer - сервис, который получает и обрабатывает данные.

2 Предварительная настройка

2.1 Установка Docker

1. Перейдите на https://www.docker.com/products/docker-desktop
2. Скачайте Docker Desktop для вашей ОС.
3. Установите и запустите Docker Desktop.

2.2 Настройка Docker Compose

Docker Compose — это конфигурационный файл, в котором описываются все контейнеры, нужные для запуска приложения с помощью Docker Compose. Он позволяет одной командой запускать несколько контейнеров, указывая, как они должны работать вместе.

Листинг 1. docker-compose.yml

```
services:

  rabbitmq: #Центральный брокер сообщений для обмена данными между сервисами
    image: rabbitmq:3-management
    ports:
      - "5672:5672" # Порт для AMQP (основной протокол RabbitMQ), порты для подключения клиентов
      - "15672:15672" # Порт для веб-интерфейса управления
    environment:
      RABBITMQ_DEFAULT_USER: user # Логин по умолчанию
      RABBITMQ_DEFAULT_PASS: password
    healthcheck:
      test: ["CMD", "rabbitmq-diagnostics", "ping"]
      interval: 10s # Проверка каждые 10 секунд
      timeout: 5s # Таймаут 5 секунд
      retries: 5

  barrier: # Координатор синхронизации (барьер) для процессов
    build: ./barrier # Собирается из Dockerfile в ./barrier
    depends_on:
      rabbitmq:
        condition: service_healthy # Запустится только после готовности RabbitMQ
    environment:
      - TOTAL_WORKERS=3 # Общее количество процессов
      - TOTAL_BARRIERS=3 # Количество барьеров

  worker1:
    build: ./worker # Собирается из Dockerfile в ./worker
```

```
28      environment:
29        - WORKER_ID=1 # Уникальный идентификатор
30        - TOTAL_WORKERS=3
31        - TOTAL_BARRIERS=3
32      depends_on:
33        rabbitmq:
34          condition: service_healthy
35
36    worker2:
37      build: ./worker
38      environment:
39        - WORKER_ID=2
40        - TOTAL_WORKERS=3
41        - TOTAL_BARRIERS=3
42      depends_on:
43        rabbitmq:
44          condition: service_healthy
45
46    worker3:
47      build: ./worker
48      environment:
49        - WORKER_ID=3
50        - TOTAL_WORKERS=3
51        - TOTAL_BARRIERS=3
52      depends_on:
53        rabbitmq:
54          condition: service_healthy
```

2.3 Реализация процессов

2.3.1 Настройка Dockerfile

Инструкция для создания Docker-образа процессов.

Листинг 2. Dockerfile

```
1 FROM python:3.10-slim # Базовый образ на основе Python 3.10
2 WORKDIR /app # Создаёт директорию /app внутри контейнера и делает её рабочей.
3
4 COPY worker.py . # Копирует файл worker.py с хоста (компьютера) в контейнер (в /app, так как указан WORKDIR)
5 RUN pip install pika # Устанавливает Python-библиотеку pika через pip
6 CMD ["python", "worker.py"] # Задаёт команду по умолчанию при запуске контейнера.
```

2.3.2 Реализация worker

Общее описание:

1. Каждый процесс (worker1, worker2, worker3) выполняет свою часть работы.

2. В определённых точках (барьерах) воркеры синхронизируются:

– Сообщают о достижении барьера.

– Ждут разрешения продолжить работу.

3. RabbitMQ выступает координатором через очереди.

Листинг 3. worker.py

```
import pika
import os
import time
import random
from datetime import datetime

worker_id = int(os.environ.get("WORKER_ID", "0"))
total_barriers = int(os.environ.get("TOTAL_BARRIERS", "3"))

def connect_to_rabbitmq(retries=10, delay=5): # Пытается подключиться к RabbitMQ с повторными попытками
    credentials = pika.PlainCredentials('user', 'password') # Конфигурация подключения к RabbitMQ
    parameters = pika.ConnectionParameters('rabbitmq', credentials=credentials)
    for attempt in range(retries):
        try:
            return pika.BlockingConnection(parameters)
        except pika.exceptions.AMQPConnectionError: # Обработка ошибки подключения
            print(f"[Worker {worker_id}] RabbitMQ not ready, retrying in {delay} seconds...")
            time.sleep(delay) # Ждёт перед следующей попыткой.
    raise Exception("Could not connect to RabbitMQ")

# Стартовая задержка для случайного старта
initial_delay = random.uniform(1, 5)
print(f"[Worker {worker_id}] Starting after initial delay of {initial_delay:.2f} seconds...")
time.sleep(initial_delay)

for barrier_num in range(1, total_barriers + 1): # Цикл по барьерам
    connection = connect_to_rabbitmq()
    channel = connection.channel() # Создаёт AMQP-канал для операций

```

```
30      barrier_queue = f"barrier_queue_{barrier_num}" # Общая очере
            дь для уведомлений о достижении барьера N всеми процессам
            и
31      release_queue = f"release_queue_{barrier_num}_{worker_id}" #
            Уникальная очередь для процесса W на барьере N
32
33      channel.queue_declare(queue=barrier_queue, durable=True)
34      channel.queue_declare(queue=release_queue, durable=True)
35
36      #Имитация работы
37      print(f"[Worker {worker_id}] Doing work before barrier {
            barrier_num}...")
38      work_time = random.uniform(1, 10)
39      time.sleep(work_time)
40
41      # Уведомление о достижении барьера
42      reached_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
43      print(f"[Worker {worker_id}] Reached barrier {barrier_num}
            at {reached_time}")
44      message = f"Worker {worker_id} reached barrier {barrier_num}
            at {reached_time}"
45      channel.basic_publish(exchange='', routing_key=barrier_queue
            , body=message) # Отправка сообщения в barrier_queue
46      print(f"[Worker {worker_id}] Sent signal for barrier {
            barrier_num}")
47
48      # Ожидание разрешения продолжить
49      def callback(ch, method, properties, body):
50          release_msg = body.decode()
51          print(f"[Worker {worker_id}] Received: '{release_msg}'
                at {datetime.now().strftime('%H:%M:%S')}")
52          ch.stop_consuming() # Прекращает ожидание новых сообщени
                й
53
54      print(f"[Worker {worker_id}] Waiting at barrier {barrier_num
            }...")
55      channel.basic_consume(queue=release_queue,
            on_message_callback=callback, auto_ack=True) # Подписывае
            тся на release_queue
56      channel.start_consuming() # Блокирует выполнение, пока не пр
            идёт сообщение
57      connection.close()
58
59  print(f"[Worker {worker_id}] All barriers passed. Work complete.
        ")
```

2.4 Реализация барьера

2.4.1 Настройка Dockerfile

Инструкция для создания Docker-образа барьера.

Листинг 4. Dockerfile

```
FROM python:3.10-slim

WORKDIR /app
COPY barrier.py .

RUN pip install pika

CMD ["python", "barrier.py"]
```

2.4.2 Реализация barrier

Общее описание:

1. Процессы (workers) выполняют задачи и сообщают о достижении барьеров.

2. Координатор (этот код) отслеживает состояние барьеров и разрешает продолжение.

3. RabbitMQ выступает в роли:

- Транспорта для сообщений
- Временного хранилища состояний
- Механизма синхронизации

Листинг 5. barrier.py

```
import os
import pika
import time
from datetime import datetime
from collections import defaultdict

TOTAL_WORKERS = int(os.environ.get("TOTAL_WORKERS", "3"))
TOTAL_BARRIERS = int(os.environ.get("TOTAL_BARRIERS", "3"))

def connect_to_rabbitmq(retries=10, delay=5):
    credentials = pika.PlainCredentials('user', 'password')
    parameters = pika.ConnectionParameters('rabbitmq',
        credentials=credentials)
    for attempt in range(retries):
        try:
```

```
15                 return pika.BlockingConnection(parameters)
16             except pika.exceptions.AMQPConnectionError:
17                 time.sleep(delay)
18         raise Exception("Could not connect to RabbitMQ")
19
20 worker_status = defaultdict(set)  # хранит для каждого барьера м
       ножество ID процессов, которые его достигли
21
22 def make_callback(barrier_num, channel):
23     def callback(ch, method, properties, body): # Это обработчик
           сообщений, который будет вызываться при получении сообще
          ния из очереди.
24         message = body.decode()
25         worker_id = int(message.split()[1]) # Извлекаем ID проце
               сса
26         timestamp = message.split("at")[-1].strip() # Извлекаем
               время
27
28         if worker_id not in worker_status[barrier_num]: # Добавл
               яем процесс в множество только если его там еще нет
29             worker_status[barrier_num].add(worker_id)
30
31         if len(worker_status[barrier_num]) == TOTAL_WORKERS: # д
               остигли ли барьера ВСЕ процессы
32             release_time = datetime.now().strftime('%Y-%m-%d %H
                   :%M:%S')
33             for wid in range(1, TOTAL_WORKERS + 1):
34                 queue = f"release_queue_{barrier_num}_{wid}" # Ф
                       ормируем имя персональной очереди
35                 channel.queue_declare(queue=queue, durable=True)
                       # Убеждаемся, что очередь существует
36                 # Отправляем сообщение-разрешение
37                 channel.basic_publish(exchange='', routing_key=
                       queue, body=f"release barrier {barrier_num}")
38     return callback
39
40 connection = connect_to_rabbitmq()
41 channel = connection.channel()
42
43 for barrier_num in range(1, TOTAL_BARRIERS + 1):
44     queue = f"barrier_queue_{barrier_num}"
45     channel.queue_declare(queue=queue, durable=True) # создает о
           чередь если её нет
46     channel.basic_consume(queue=queue, # имя очереди для подписк
           и
47     on_message_callback=make_callback(barrier_num, channel), # с
           оздает уникальный обработчик для каждого барьера
48     auto_ack=True) # автоматическое подтверждение получения сооб
```

```
        щений
49
50 channel.start_consuming()
```

2.5 Запуск

Все контейнеры можно запустить слудеющей командой:
docker−compose up −−build

3 Результаты работы

3.1 Логи worker1

[Worker 1] Starting after initial delay of 2.97 seconds...
[Worker 1] Doing work before barrier 1...
[Worker 1] Reached barrier 1 at 2025-05-06 21:26:00
[Worker 1] Sent signal for barrier 1
[Worker 1] Waiting at barrier 1...
[Worker 1] Received: 'release barrier 1' at 21:26:02
[Worker 1] Doing work before barrier 2...
[Worker 1] Reached barrier 2 at 2025-05-06 21:26:06
[Worker 1] Sent signal for barrier 2
[Worker 1] Waiting at barrier 2...
[Worker 1] Received: 'release barrier 2' at 21:26:06
[Worker 1] Doing work before barrier 3...
[Worker 1] Reached barrier 3 at 2025-05-06 21:26:15
[Worker 1] Sent signal for barrier 3
[Worker 1] Waiting at barrier 3...
[Worker 1] Received: 'release barrier 3' at 21:26:15
[Worker 1] All barriers passed. Work complete.

3.2 Логи worker2

[Worker 2] Starting after initial delay of 1.43 seconds...
[Worker 2] Doing work before barrier 1...
[Worker 2] Reached barrier 1 at 2025-05-06 21:25:59

[Worker 2] Sent signal for barrier 1

[Worker 2] Waiting at barrier 1...

[Worker 2] Received: 'release barrier 1' at 21:26:02

[Worker 2] Doing work before barrier 2...

[Worker 2] Reached barrier 2 at 2025-05-06 21:26:06

[Worker 2] Sent signal for barrier 2

[Worker 2] Waiting at barrier 2...

[Worker 2] Received: 'release barrier 2' at 21:26:06

[Worker 2] Doing work before barrier 3...

[Worker 2] Reached barrier 3 at 2025-05-06 21:26:14

[Worker 2] Sent signal for barrier 3

[Worker 2] Waiting at barrier 3...

[Worker 2] Received: 'release barrier 3' at 21:26:15

[Worker 2] All barriers passed. Work complete.

3.3 Логи worker3

[Worker 3] Starting after initial delay of 3.59 seconds...

[Worker 3] Doing work before barrier 1...

[Worker 3] Reached barrier 1 at 2025-05-06 21:26:02

[Worker 3] Sent signal for barrier 1

[Worker 3] Waiting at barrier 1...

[Worker 3] Received: 'release barrier 1' at 21:26:02

[Worker 3] Doing work before barrier 2...

[Worker 3] Reached barrier 2 at 2025-05-06 21:26:04

[Worker 3] Sent signal for barrier 2

[Worker 3] Waiting at barrier 2...

[Worker 3] Received: 'release barrier 2' at 21:26:06

[Worker 3] Doing work before barrier 3...

[Worker 3] Reached barrier 3 at 2025-05-06 21:26:12

[Worker 3] Sent signal for barrier 3

[Worker 3] Waiting at barrier 3...

[Worker 3] Received: 'release barrier 3' at 21:26:15

[Worker 3] All barriers passed. Work complete.

4 Заключение

В результате лабораторной работы реализована барьерная синхронизация между тремя процессами с использованием брокера сообщений RabbitMQ и показано, как она работает. Контейнеры были связаны между собой через docker-compose.