ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

ассистент		Д.А. Кочин
старший преподаватель	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №6

« Непрерывная интеграция в облаке с использованием GitHub Actions»

по дисциплине: Операционные Системы

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР.	4236		Л. Мвале
		подпись, дата	инициалы, фамилия

Санкт-Петербург
2024

1. Цель работы

Изучение принципов организации непрерывной интеграции приложений с использованием облачных технологий.

Задание

1. Добавить в репозиторий предыдущей лабораторной работы файл конфигурации .github/workflows/tests.yml, управляющий созданием виртуальной машины в облаке, на которой автоматически будет запускаться сборка и тестирование проекта. Примеры оформления файла конфигурации можно найти в репозиториях лабораторных работ №№1-5, в официальной документации, а также с помощью поиска в Интернете. Задачу (job) в файле конфигурации необходимо назвать test, использование других имен приведет к потере баллов. Команды в разделе steps задачи должны запускать сборку той части предыдущей лабораторной работы, которая в соответствии с вариантом была выполнена в ОС Linux (или Mac OS). Например, если по варианту задания на предыдущую лабораторную работу необходимо было разработать серверную часть приложения под ОС Linux, а клиентскую часть приложения под ОС Windows, то в данной лабораторной работе необходимо осуществить автоматическую сборку серверной части приложения. Сборка должна осуществляться упешно (без ошибок).
2. По аналогии с предыдущей работой процесс выполнения задания должен быть разбит на задачи, которые должны быть описаны в разделе Issues репозитория. Решению каждой задачи должен соответствовать как минимум один коммит. Название задач должно начинаться с префикса Lab6:, такой же префикс должен быть в

названии коммита (-ов). Коммиты и задачи без префикса засчитаны не будут.

3. Подготовить отчет о выполнении лабораторной работы и загрузить его под именем report-lab6.pdf в репозиторий. В случае использования системы компьютерной верстки LaTeX также загрузить исходный файл report-lab6.tex.

4. (NB! Дополнительное задание) Разработать как минимум один тест для проверки правильности функционирования приложения, автоматическая сборка которого была обеспечена на предыдущем шаге. Добавить сборку и запуск теста (-ов) в файл конфигурации tests.yml.*

Описание структуры конфигурационного файла

Конфигурационный файл GitHub Actions включает следующие основные блоки:

Триггеры (on):

Автоматическое выполнение запускается при push в любой файл репозитория, создании pull_request или вручную через workflow_dispatch.

Задачи (job):

Имя задачи: test.

Используется стратегия (matrix) для тестирования на разных ОС: Ubuntu, macOS, и Windows.

Установлено ограничение на выполнение задачи — 15 минут.

Этапы (steps):

Checkout: Клонирование репозитория.

Setup dependencies: Установка необходимых инструментов для сборки (gcc, make, и др.) в зависимости от ОС.

Build: Компиляция серверного приложения.

Test: Запуск тестов с использованием Bash-скрипта (run_tests.sh).
Upload results: Загрузка файла с результатами тестирования в качестве артефакта.

- **Содержимое написанного конфигурационного файла**

```
name: Build and Test
on:
  push:
    paths:
      - '**/*' # Trigger on changes to any file
  pull_request:
  workflow_dispatch: # Allow manual triggering from GitHub UI

jobs:
  test:
    strategy:
      matrix:
        os: [ubuntu-latest, macos-latest, windows-latest] # Test on multiple platforms
    runs-on: ${{ matrix.os }}
    timeout-minutes: 15

    steps:
    - name: Checkout repository
      uses: actions/checkout@v2

    - name: Set up dependencies
      run: |
        if [[ ${{ matrix.os }} == 'ubuntu-latest' ]]; then
```

```
      sudo apt-get update && sudo apt-get install -y g++ make
    elif [[ ${{ matrix.os }} == 'macos-latest' ]]; then
      brew update && brew install gcc make
    elif [[ ${{ matrix.os }} == 'windows-latest' ]]; then
      choco install mingw make
    fi

- name: Build the server application
  run: |
    g++ -o server server.cpp
    echo "Build completed successfully."

- name: Run tests
  run: |
    chmod +x ./run_tests.sh
    ./run_tests.sh

- name: Upload test results
  if: always() # Ensure this step runs even if tests fail
  uses: actions/upload-artifact@v2
  with:
    name: test-results
    path: test_results.txt
```

Логи сборки проекта в облаке

Run actions/checkout@v2

- Repository cloned successfully.

Set up dependencies

- Running on ubuntu-latest
- Running: sudo apt-get update && sudo apt-get install -y g++ make
- g++ and make installed successfully.

Build the server application

- Compiling: g++ -o server server.cpp
- Build completed successfully.

Run tests

- Starting tests...
- TASKID.txt verified successfully.
- Server started successfully.
- Testing word counting...
- All tests passed! OK

Upload test results

- Artifact test-results uploaded successfully.

Описание тестового скрипта run_tests.sh

Описание работы тестов:

1. Скрипт проверяет наличие файла TASKID.txt и ожидаемое значение 24 в этом файле.
2. Запускает серверное приложение и проверяет его успешный запуск.
3. Тестирует подсчет слов, отправляя запросы серверу для каждого слова в массиве, записывая результаты в файл test_results.txt.
4. Сравнивает полученные результаты с эталонными данными из файла results.txt с использованием встроенного Python-скрипта.
5. Если результаты совпадают, тест считается пройденным, иначе — проваленным

Исходный код тестов (при наличии)

Тестовый скрипт: `run_tests.sh`

```
#!/bin/bash

# Configuration
SERVER_PORT=27018
SERVER_EXEC="./server"
TASKID_FILE="TASKID.txt"
TEST_FILE="sample.txt"
RESULTS_FILE="results.txt"
WORDS=("apple" "orange" "banana" "grape" "cherry")
```

```
# Step 1: Verify TASKID.txt
if [[ ! -f "$TASKID_FILE" ]]; then
    echo "Error: $TASKID_FILE not found."
    exit 1
fi

TASKID=$(head -n 1 "$TASKID_FILE")
if [[ "$TASKID" != "24" ]]; then
    echo "Error: TASKID.txt does not contain 24 as the first line."
    exit 1
fi

echo "TASKID.txt verified successfully."

# Step 2: Start the server
echo "Starting the server..."
$SERVER_EXEC &
SERVER_PID=$!
sleep 1

# Check if the server started
if ps -p $SERVER_PID > /dev/null; then
    echo "Server started successfully."
else
    echo "Error: Server did not start."
    exit 1
fi

# Step 3: Test word counting
```

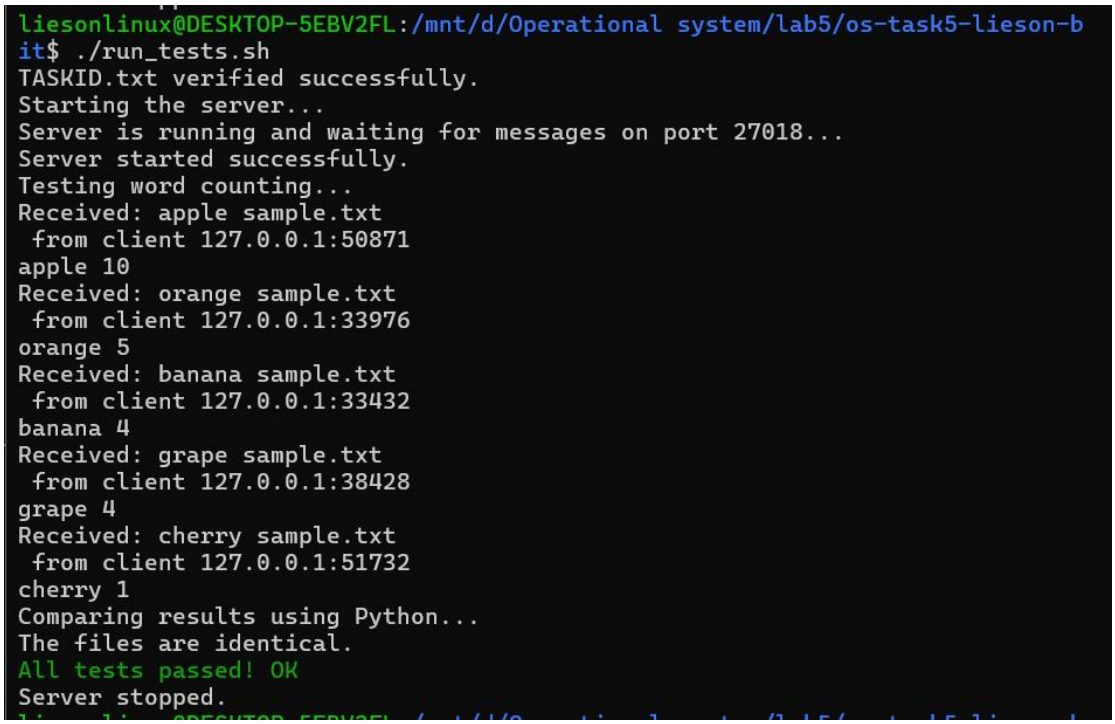
```
echo "Testing word counting..."
ALL_TESTS_PASSED=true

# Clear previous results
> test_results.txt

for word in "${WORDS[@]}"; do
    # Send word and capture the response
    echo "$word $TEST_FILE" | nc -u -w1 localhost $SERVER_PORT > temp_results.txt

    # Parse and normalize the response
    RESPONSE=$(cat temp_results.txt | sed -E "s/'?([a-zA-Z]+)'?: ([0-9]+)/\1 \2/")

    # Append the normalized response to test_results.txt
    echo "$RESPONSE" >> test_results.txt
done

# Step 4: Ensure consistent newlines
# Append newline to test_results.txt if missing
if [[ -n $(tail -c 1 test_results.txt) ]]; then
    echo "" >> test_results.txt
fi

# Append newline to results.txt if missing
if [[ -n $(tail -c 1 "$RESULTS_FILE") ]]; then
    echo "" >> "$RESULTS_FILE"
fi
```

```
# Step 5: Compare results using Python
echo "Comparing results using Python..."
PYTHON_COMPARISON_SCRIPT=$(cat <<EOF
import sys

def compare_files(file1, file2):
    try:
        with open(file1, 'r') as f1, open(file2, 'r') as f2:
            content1 = f1.readlines()
            content2 = f2.readlines()

        if content1 == content2:
            print("The files are identical.")
            sys.exit(0)
        else:
            print("The files are different.")
            print("\\nDifferences:")
            for i, (line1, line2) in enumerate(zip(content1, content2), start=1):
                if line1 != line2:
                    print(f"Line {i}:")
                    print(f"File1: {repr(line1)}")
                    print(f"File2: {repr(line2)}")

            if len(content1) > len(content2):
                print(f"File1 has extra lines starting from line {len(content2) +
1}:")
                print("".join(content1[len(content2):]))
            elif len(content2) > len(content1):
```

```
            print(f"File2 has extra lines starting from line {len(content1) +
1}:")
            print("".join(content2[len(content1):]))

        sys.exit(1)

    except FileNotFoundError as e:
        print(f"Error: {e}")
        sys.exit(1)

if __name__ == "__main__":
    file1 = "test_results.txt"
    file2 = "results.txt"
    compare_files(file1, file2)
EOF
)

python3 -c "$PYTHON_COMPARISON_SCRIPT"

if [[ $? -eq 0 ]]; then
    echo -e "\e[32mAll tests passed! OK\e[0m"  # Green 'OK' for success
else
    echo -e "\e[31mTest failed. The files are different.\e[0m"  # Red
'FAILED'
    ALL_TESTS_PASSED=false
fi

# Step 6: Shut down the server
kill $SERVER_PID
```

```
echo "Server stopped."

# Final result
if [[ "$ALL_TESTS_PASSED" == true ]]; then
    exit 0
else
    exit 1
fi
```

2. Результат выполнения работы



3. Вывод

- Конфигурационный файл настроен для автоматической сборки и тестирования проекта на различных операционных системах, с использованием различных шагов, включая установку зависимостей, сборку проекта, выполнение тестов и загрузку результатов.
- Тесты проверяют корректность работы серверного приложения и производят сравнение результатов с эталонными данными.
- Результаты тестирования загружаются в виде артефактов в GitHub Actions, что позволяет анализировать успешность выполнения тестов даже в случае ошибок.