

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин

Лабораторная работа №1
по курсу
«Технологии распределенных вычислений и анализа данных»

Выполнила:

магистрант группы 355841
А.В. Деркач

Проверил:

к.т.н., доцент каф. ЭВМ
Д.Ю. Перцев

Минск 2024

1 ЗАДАНИЕ

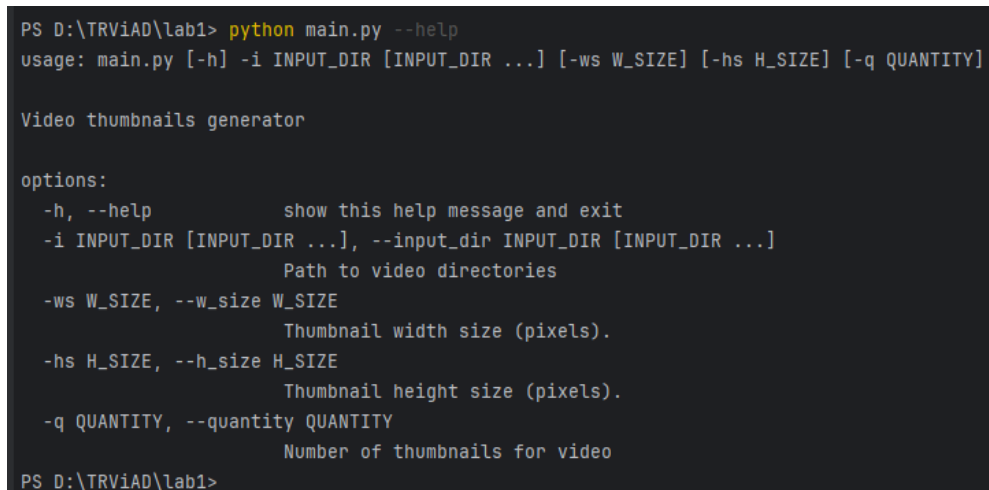
На вход подаются папки с множеством видеофайлов. На выходе, в каждой проанализированной папке, для каждого видеофайла, создается некоторый набор миниатюр.

Бонусные задачи: видеофайлы расположены на разных устройствах, присутствует распределенная система генерации.

Допускается использовать: любую операционную систему, любой язык программирования, любые технологии и библиотеки алгоритмов.

2 ВЫПОЛНЕНИЕ РАБОТЫ

Реализована пользовательская утилита, которая запускается с помощью командной строки и поддерживает параметры, представленные на рисунке 2.1.



```
PS D:\TRViAD\lab1> python main.py --help
usage: main.py [-h] -i INPUT_DIR [INPUT_DIR ...] [-ws W_SIZE] [-hs H_SIZE] [-q QUANTITY]

Video thumbnails generator

options:
  -h, --help            show this help message and exit
  -i INPUT_DIR [INPUT_DIR ...], --input_dir INPUT_DIR [INPUT_DIR ...]
                        Path to video directories
  -ws W_SIZE, --w_size W_SIZE
                        Thumbnail width size (pixels).
  -hs H_SIZE, --h_size H_SIZE
                        Thumbnail height size (pixels).
  -q QUANTITY, --quantity QUANTITY
                        Number of thumbnails for video
PS D:\TRViAD\lab1>
```

Рисунок 2.1 – Параметры для настройки генерации миниатюр

Для запуска утилиты с указанием трех директорий видеофайлов (суммарно 15 видео), десяти миниатюр для генерации размером 854x480 (суммарно 150 миниатюр) выполняется команда:

```
python main.py -i '../videos_1' '../videos_2' '../videos_3'
-ws 854 -hs 480 -q 10
```

Утилита анализирует все переданные папки и для каждого найденного видеофайла генерирует заданное количество миниатюр с заданным размером. Каждое видео разбивается на фреймы и случайным образом отбираются фреймы для сохранения в качестве миниатюры.

Результат генерации видеофайлов в трех директориях представлен на рисунке 2.2. На рисунке 2.3 представлен результат генерации миниатюр для одного видео.

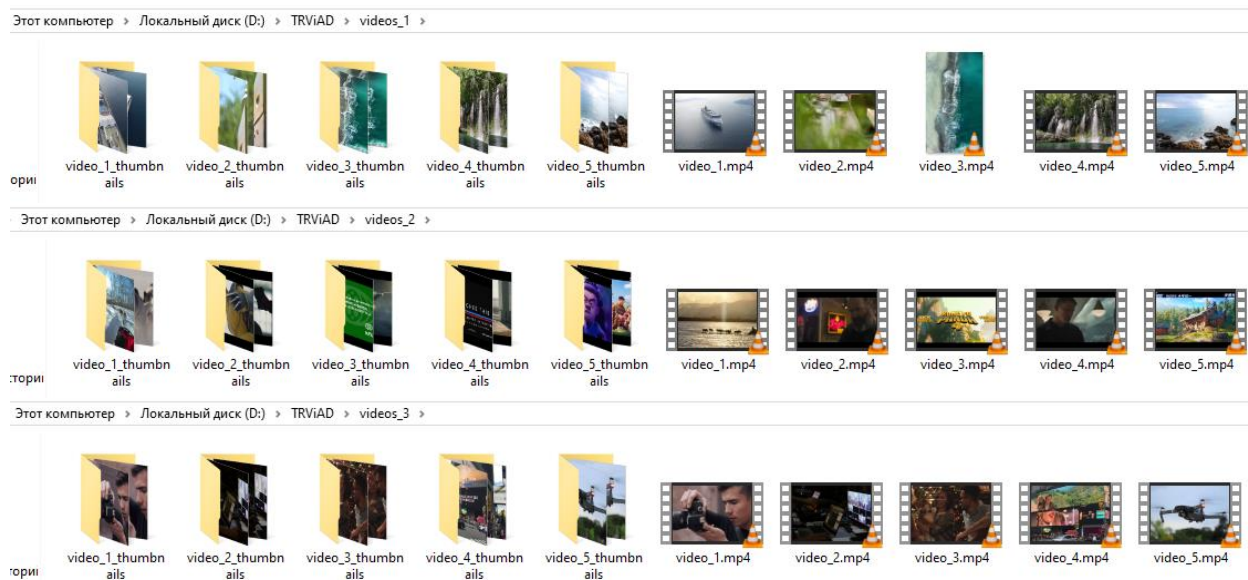


Рисунок 2.2 – Результат генерации миниатюр для видеофайлов в трех директориях

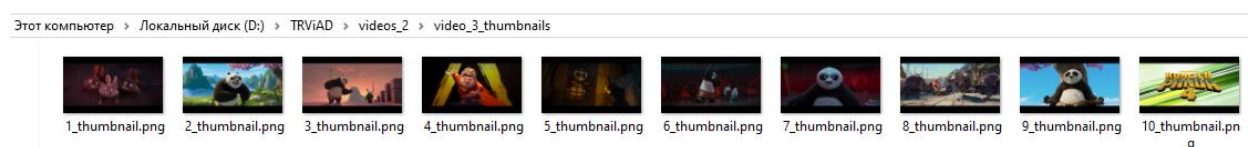


Рисунок 2.3 – Результат генерации миниатюр для одного видеофайла

В утилите предусмотрена распределенная система генерации с балансировщиком нагрузки. Для этого указываются адреса узлов (серверов), которые через REST API принимают видео и возвращают список миниатюр для этого видео. Все узлы работают параллельно, поэтому чем больше узлов – тем быстрее обработаются видеофайлы. Узлы могут располагаться на разных серверах, доступных утилите.

Для тестирования балансировщика нагрузки было добавлено дополнительной логирование, демонстрирующее сервис, на который осуществлялась отправка задачи по обработке одного видео (см. рисунок 2.4).

```

PS D:\TRViAD\lab1> python main.py -i '../videos_1' '../videos_2' '../videos_3' -ws 854 -hs 480 -q 10
http://localhost:8080
http://localhost:8081
http://localhost:8082
http://localhost:8080
http://localhost:8081
http://localhost:8080
http://localhost:8081
http://localhost:8082
http://localhost:8080
http://localhost:8081
http://localhost:8080
http://localhost:8081
http://localhost:8082
http://localhost:8080
http://localhost:8081
Thumbnails successfully generated.
PS D:\TRViAD\lab1>

```

Рисунок 2.4 – Логирование балансировщика нагрузки узлов генерации

Код реализации пользовательского приложения на языке Python:

```

001. import argparse
002. import os
003. import shutil
004. from concurrent.futures.thread import ThreadPoolExecutor
005.
006. import cv2
007. import numpy as np
008. import requests
009.
010. VIDEO_FORMATS = ['.WEBM', '.MPG', '.MP2', '.MPEG', '.MPE', '.MPV', '.OGG',
011.                  '.MP4', '.M4P', '.M4V',
012.                  '.AVI', '.WMV', '.MOV', '.QT', '.FLV', '.SWF', '.AVCHD']
013.
014. WORKER_NODES = [
015.     "http://localhost:8080",
016.     "http://localhost:8081",
017.     "http://localhost:8082"
018. ]
019.
020. def get_videos_from_directory(video_directory_path: str):
021.     directory_video_files = []
022.     directories_to_process = [video_directory_path]
023.
024.     while directories_to_process:
025.         current_directory = directories_to_process.pop(0)
026.         directory_entries = os.listdir(current_directory)
027.
028.         for entry in directory_entries:
029.             full_entry_path = os.path.join(current_directory, entry)
030.
031.             if os.path.isdir(full_entry_path):
032.                 directories_to_process.append(full_entry_path)
033.             else:
034.                 _, file_extension = os.path.splitext(full_entry_path)

```

```

035.             if file_extension.upper() in VIDEO_FORMATS:
036.                 directory_video_files.append(full_entry_path)
037.
038.     return directory_video_files
039.
040.
041. def save_video_thumbnails(video_path: str, thumbnails: list[np.ndarray]):
042.     video_thumbnails_directory = os.path.join(
043.         os.path.dirname(video_path), "%s_thumbnails" %
044.         (os.path.splitext(os.path.basename(video_path))[0])
045.     )
046.     if os.path.exists(video_thumbnails_directory):
047.         shutil.rmtree(video_thumbnails_directory)
048.
049.     os.mkdir(video_thumbnails_directory)
050.
051.     for i, thumbnail in enumerate(thumbnails):
052.         cv2.imwrite('%s/%d_thumbnail.png' % (video_thumbnails_directory,
053. i + 1), thumbnail)
054.
055. def process_videos_parallel(videos: list[str], thumbnail_width: int,
056. thumbnail_height: int, num_thumbnails: int):
057.     with ThreadPoolExecutor(max_workers=len(WORKER_NODES)) as executor:
058.         futures = []
059.         for video_path in videos:
060.             worker_url = WORKER_NODES[len(futures) % len(WORKER_NODES)]
061.             futures.append(executor.submit(process_video, video_path,
062. thumbnail_width, thumbnail_height, num_thumbnails,
063. worker_url))
064.
065.         for future in futures:
066.             future.result()
067.
068. def process_video(video_path: str, thumbnail_width: int, thumbnail_height:
069. int, num_thumbnails: int, worker_url: str):
070.     with open(video_path, 'rb') as file:
071.         files = {'video': file}
072.         params = {
073.             'thumbnail_width': thumbnail_width,
074.             'thumbnail_height': thumbnail_height,
075.             'num_thumbnails': num_thumbnails
076.         }
077.         response = requests.post(worker_url + '/process_video', files=files,
078. data=params)
079.         response.raise_for_status()
080.         response_data = response.json()
081.
082.         thumbnails_json = response_data['thumbnails']
083.         thumbnails = [np.array(thumbnail) for thumbnail in thumbnails_json]
084.
085.         save_video_thumbnails(video_path, thumbnails)
086.
087.
088. def main():
089.     parser = argparse.ArgumentParser(description='Video thumbnails
090. generator')

```

```

090.         parser.add_argument('-i', '--input_dir', type=str, nargs='+',
required=True, help='Path to video directories')
091.         parser.add_argument('-ws', '--w_size', type=int, default=640,
help='Thumbnail width size (pixels).')
092.         parser.add_argument('-hs', '--h_size', type=int, default=640,
help='Thumbnail height size (pixels).')
093.         parser.add_argument('-q', '--quantity', type=int, default=5,
help='Number of thumbnails for video')
094.         args = parser.parse_args()
095.
096.         try:
097.             input_directories = args.input_dir
098.             thumbnail_width = args.w_size
099.             thumbnail_height = args.h_size
100.             num_thumbnails = args.quantity
101.
102.             for video_directory_path in input_directories:
103.                 directory_videos =
get_videos_from_directory(video_directory_path)
104.
105.                 for i in range(0, len(directory_videos), len(WORKER_NODES)):
106.                     group = directory_videos[i:i + len(WORKER_NODES)]
107.                     process_videos_parallel(group, thumbnail_width,
thumbnail_height, num_thumbnails)
108.
109.                     print("Thumbnails successfully generated.")
110.         except Exception as e:
111.             print("Oops! Something went wrong... :( \n{}".format(e))
112.
113.
114. if __name__ == '__main__':
115.     main()

```

Код реализации узла генерации изображений на языке Python (каждый узел дублируется и задается свой порт):

```

01. import os
02. import tempfile
03.
04. from flask import Flask, request, jsonify
05.
06. from video_thumbnails_generator import VideoThumbnailsGenerator
07.
08. app = Flask(__name__)
09.
10.
11. @app.route('/process_video', methods=['POST'])
12. def process_video():
13.     video_file = request.files['video']
14.     thumbnail_width = int(request.form['thumbnail_width'])
15.     thumbnail_height = int(request.form['thumbnail_height'])
16.     num_thumbnails = int(request.form['num_thumbnails'])
17.
18.     with tempfile.TemporaryDirectory() as temp_dir:
19.         try:
20.             video_path = os.path.join(temp_dir, video_file.filename)
21.             video_file.save(video_path)
22.
23.             generator = VideoThumbnailsGenerator(video_path, thumbnail_width,
thumbnail_height, num_thumbnails)

```

```

24.         generated_thumbnails = generator.generate()
25.
26.         thumbnails_data = [thumbnail.tolist() for thumbnail in
generated_thumbnails]
27.         return jsonify({'thumbnails': thumbnails_data})
28.     except Exception as e:
29.         return 'Something went wrong... {}'.format(e), 500
30.
31.
32. if __name__ == '__main__':
33.     app.run(debug=True, port=8080)

```

Код реализации алгоритма генерации на языке Python:

```

01. import random
02.
03. import cv2
04. import numpy as np
05.
06.
07. class VideoThumbnailsGenerator:
08.     VIDEO_FORMATS = ['.WEBM', '.MPG', '.MP2', '.MPEG', '.MPE', '.MPV',
'.OGG', '.MP4', '.M4P', '.M4V',
09.                     '.AVI', '.WMV', '.MOV', '.QT', '.FLV', '.SWF', '.AVCHD']
10.
11.     def __init__(self, video_path: str, thumbnail_width=640,
thumbnail_height=640, num_thumbnails=5):
12.         self.video_path = video_path
13.         self.thumbnail_width = thumbnail_width
14.         self.thumbnail_height = thumbnail_height
15.         self.num_thumbnails = num_thumbnails
16.
17.     def generate(self):
18.         video_frames_number = self.get_video_frames_number(self.video_path)
19.         video_thumbnails_frames_indexes =
self.select_random_video_frames(video_frames_number)
20.         video_thumbnails_frames =
self.get_frames_from_video(self.video_path, video_thumbnails_frames_indexes)
21.
22.         return self.convert_frames_to_thumbnails(video_thumbnails_frames)
23.
24.     def select_random_video_frames(self, video_frames_number: int):
25.         frames_number_to_select = self.num_thumbnails
26.
27.         if frames_number_to_select >= video_frames_number:
28.             frames_number_to_select = video_frames_number
29.
30.         return random.sample(range(video_frames_number),
frames_number_to_select)
31.
32.     def convert_frames_to_thumbnails(self, frames: list[np.ndarray]):
33.         thumbnails = []
34.         for frame in frames:
35.             frame_hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
36.             thumbnail = cv2.resize(frame_hsv, (self.thumbnail_width,
self.thumbnail_height))
37.             result_thumbnail = cv2.cvtColor(thumbnail, cv2.COLOR_HSV2BGR)
38.
39.             thumbnails.append(np.array(result_thumbnail))
40.
41.         return thumbnails

```

```

42.
43.     @staticmethod
44.     def get_frames_from_video(video_path: str, frame_indexes: list[int]):
45.         cap = cv2.VideoCapture(video_path)
46.         if not cap.isOpened():
47.             raise ValueError("Unable to open video file
48. '{}'.format(video_path))
49.         frames = []
50.         for index in frame_indexes:
51.             cap.set(cv2.CAP_PROP_POS_FRAMES, index)
52.
53.             ret, frame = cap.read()
54.             if not ret:
55.                 raise ValueError("Failed to read frame from video file
56. '{}'.format(video_path))
57.             frames.append(frame)
58.
59.             cap.release()
60.
61.             return frames
62.
63.     @staticmethod
64.     def get_video_frames_number(video_path: str):
65.         cap = cv2.VideoCapture(video_path)
66.         if not cap.isOpened():
67.             raise ValueError("Unable to open video file
68. '{}'.format(video_path))
69.         video_frames_number = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
70.
71.         cap.release()
72.         return video_frames_number

```

3 ВЫВОДЫ

В ходе выполнения лабораторной работы была подготовлена пользовательская утилита для генерации миниатюр видеофайлов в заданных директориях операционной системы (ОС). Утилита реализована с применением распределительной системы генерации и балансировщиком нагрузки между несколькими REST API «узлов» генерации. Тестирование производилось с использованием трех «узлов» на ОС Microsoft Windows 10 и 11.