



Instituto Politécnico  
de Viana do Castelo

Licenciatura em ENGENHARIA INFORMÁTICA / Degree INFORMATICS ENGINEERING

# INTEGRAÇÃO DE SISTEMAS

## Trabalho Prático N.º 1A

### Desafio RPC Python


N.º do Aluno 1 – Nome do Aluno 1;

N.º do Aluno 2 – Nome do Aluno 2

Orientador(es):

• Professor Doutor XXX, Professor Doutor YYYY

## ■ Índice

- 
1. Introdução e Objetivos
  2. O que é RPC (Remote Procedure Call)?
  3. Para que é usado o RPC?
  4. Evidências dos desafios de RPC Python
  5. Conclusão
  6. Bibliografia e Referências Web

## ■ Introdução e Objetivos



Neste trabalho aborda uma metodologia de comunicação que é chamada de RPC – Remote Procedure Call. No decorrer desta apresentação aborda-se a definição do RPC bem como alguns exemplos de onde é usada este protocolo de comunicação. Posteriormente, apresenta-se as evidências dos desafios propostos pelo docente com uma breve explicação das funcionalidades utilizadas para o desenvolvimento do código.

## ■ Introdução e Objetivos



Os objetivos para este trabalho são:

- Estabelecer a comunicação entre o cliente e o servidor utilizando RPC
- Utilizar a biblioteca Pillow para manipular imagens no servidor;
- Criar um Dockerfile que instala as dependências necessárias, incluindo a biblioteca Pillow;
- Implementar funcionalidades para criar uma tarefa de processamento de imagem e consultar o status da tarefa;

## ■ 2. O que é o RPC (Remote Procedure Call)?



RPC (Remote Procedure Call) é um protocolo de comunicação que permite a execução de procedimentos num programa num sistema remoto, como se fossem *local calls*, ou seja, permite a solicitação de um procedimento num sistema remoto, abstraindo assim a complexidade da comunicação entre sistemas distribuídos.

### ■ 3. Para que é usado o RPC?

- Comunicação em Sistemas Distribuídos;
- Aplicações Web;
- Processamento de Imagens e Vídeos;
- Sistemas de Gestão de Servidores: Apache Kafka, RabbitMQ, ...;

### ■ 3. Evidências – Exercício 1 (Servidor)

```
from xmlrpc.server import SimpleXMLRPCServer, SimpleXMLRPCRequestHandler
from PIL import Image, ImageFilter
import io
import base64
```

1. SimpleXMLRPCServer e SimpleXMLRPCRequestHandler são classes que fornecem funcionalidades para criar um servidor
2. A biblioteca PIL faz o processamento de imagens e permite a manipulá-las.
3. O *io* fornece funcionalidades para trabalhar com a entrada e saída de dados
4. O *base64* permite a codificação e decodificação dos dados.

### ■ 3. Evidências – Exercício 1 (Servidor)

```
# função que converte a imagem para cinza
def convert_to_grayscale(encoded_image):
    image_data = base64.b64decode(encoded_image.encode())
    image = Image.open(io.BytesIO(image_data))
    grayscale_image = image.convert("L")
    output_buffer = io.BytesIO()
    grayscale_image.save(output_buffer, format="JPEG")
    return base64.b64encode(output_buffer.getvalue()).decode()
```

Função que converte  
imagem para cinza

```
#função que redimensiona a imagem -- recebe como parametros a imagem codificada e as dimensoes para redimensionar
def resize_image(encoded_image, width, height):
    image_data = base64.b64decode(encoded_image.encode())
    image = Image.open(io.BytesIO(image_data))
    image.thumbnail((width, height))
    output_buffer = io.BytesIO()
    image.save(output_buffer, format="JPEG")
    return base64.b64encode(output_buffer.getvalue()).decode()
```

Função que redimensiona  
a imagem com o tamanho  
passado nos parametros

```
#função que roda a imagem num certo angulo -- recebe como parametros a imagem codificada e o angulo
def rotate_image(encoded_image, angle):
    image_data = base64.b64decode(encoded_image.encode())
    image = Image.open(io.BytesIO(image_data))
    rotated_image = image.rotate(angle)
    output_buffer = io.BytesIO()
    rotated_image.save(output_buffer, format="JPEG")
    return base64.b64encode(output_buffer.getvalue()).decode()
```

Função que roda a  
imagem consoante o  
angulo passado no  
parametro

```
#função que desfoca a imagem -- recebe como parametro a imagem codificada
def apply_blur(encoded_image):
    decoded_image = base64.b64decode(encoded_image)
    image = Image.open(io.BytesIO(decoded_image))
    blurred_image = image.filter(ImageFilter.GaussianBlur(5))
    buffered = io.BytesIO()
    blurred_image.save(buffered, format="JPEG")
    encoded_blurred_image = base64.b64encode(buffered.getvalue())
    return encoded_blurred_image.decode()
```

Função que aplica o  
desfoque na imagem



### ■ 3. Evidências – Exercício 1 (Servidor)

```
#registar servidor
server = SimpleXMLRPCServer(("0.0.0.0", 6000), requestHandler=CustomRequestHandler)
#permite ao cliente saber quais as funções que fazem parte do servidor
server.register_introspection_functions()

#registar funções
server.register_function(convert_to_grayscale, 'convert_to_grayscale')
server.register_function(resize_image, 'resize_image')
server.register_function(rotate_image, 'rotate_image')
server.register_function(apply_blur, 'apply_blur')

print("RPC Server is ready to accept requests...")
server.serve_forever()
```

1. Registar servidor
2. Registar as funções do servidor
3. Esperar pelos *requests* por parte do cliente

### ■ 3. Evidências – Exercício 1 (Cliente)

```
import xmlrpc.client
import os
import base64
```

1. Importar funcionalidades para cliente;
2. Permite manipulação de pastas/ficheiros dentro do sistema (vou criar uma pasta para imagens processadas);
3. Permite a codificação e decodificação de dados.

### ■ 3. Evidências – Exercício 1 (Cliente)

```
def connection_to_server():
    return xmlrpc.client.ServerProxy("http://localhost:6000/RPC2")
```

- Faz a conexão com o servidor

```
def rotate_image(encoded_image, proxy):
    try:
        rotated_image = proxy.rotate_image(encoded_image, 127)
        return rotated_image
    except Exception as e:
        print(f"Error: {e}")
        return None
```

- Roda a imagem

```
def resize_image(encoded_image, proxy):
    try:
        resized_image = proxy.resize_image(encoded_image, 100, 100)
        return resized_image
    except Exception as e:
        print(f"Error: {e}")
        return None
```

- Redimensiona a imagem

```
def convert_to_grayscale(encoded_image, proxy):
    try:
        grayscale_image = proxy.convert_to_grayscale(encoded_image)
        return grayscale_image
    except Exception as e:
        print(f"Error: {e}")
        return None
```

- Converte imagem para cinza

```
def apply_blur(encoded_image, proxy):
    try:
        blurred_image = proxy.apply_blur(encoded_image)
        return blurred_image
    except Exception as e:
        print(f"Error: {e}")
        return None
```

- Aplica um desfoque

### ■ 3. Evidências – Exercício 1 (Cliente)

```
def main():
    proxy = connection_to_server()
    while True:
        print("\nMenu:")
        print("1. Rotate Image")
        print("2. Resize Image")
        print("3. Convert to Grayscale")
        print("4. Blur image")
        print("0. Exit")

        choice = input("Enter your choice: ")

        if choice == "1":
            encoded_image = get_encoded_image()
            if encoded_image is not None:
                rotated_image = rotate_image(encoded_image, proxy)
                if rotated_image is not None:
                    save_image(rotated_image, "rotated_image.jpg")
                    print("Image rotated and saved as 'rotated_image.jpg'.")

        elif choice == "2":
            encoded_image = get_encoded_image()
            if encoded_image is not None:
                resized_image = resize_image(encoded_image, proxy)
                if resized_image is not None:
                    save_image(resized_image, "resized_image.jpg")
```

```
        elif choice == "3":
            encoded_image = get_encoded_image()
            if encoded_image is not None:
                grayscale_image = convert_to_grayscale(encoded_image, proxy)
                if grayscale_image is not None:
                    save_image(grayscale_image, "grayscale_image.jpg")
                    print("Image converted to grayscale and saved as 'grayscale_image.jpg'.")

        elif choice == "4":
            encoded_image = get_encoded_image()
            if encoded_image is not None:
                blurred_image = apply_blur(encoded_image, proxy)
                if blurred_image is not None:
                    save_image(blurred_image, "blurred_image.jpg")
                    print("Image blurred and saved as 'blurred_image.jpg'.")

        elif choice == "0":
            print("Thank for using the app! See you :)")
            break

    else:
        print("Invalid choice. Please try again.")
```

### ■ 3. Evidências – Exercício 1 (Cliente)

```
def get_encoded_image():
    image_filename = input("Enter image filename: ")

    if not os.path.isfile(image_filename):
        print(f"Error: File '{image_filename}' not found.")
        return None

    with open(image_filename, "rb") as image_file:
        encoded_image = base64.b64encode(image_file.read()).decode()

    return encoded_image

def save_image(encoded_image, filename):
    with open(filename, "wb") as f:
        f.write(base64.b64decode(encoded_image.encode()))

if __name__ == "__main__":
    main()
```

Retorna imagem codificada

Envia imagem para uma  
pasta de imagens  
processadas

### ■ 3. Resultado – Exercício 1

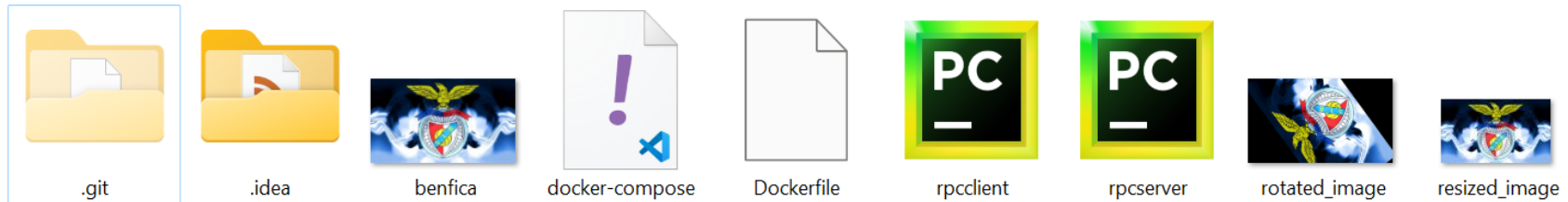
```
C:\Users\rascb\OneDrive\Ambiente de Trabalho\LEI\LEI\3 ano\1ºsem  
estre\IS\desafioRPC>docker-compose up  
[+] Building 0.0s (0/0)  
[+] Running 1/0  
✓ Container desafiorpc-rpc-server-1 Running 0.0s  
Attaching to desafiorpc-rpc-server-1  
|
```

- Servidor a correr

```
Menu:  
1. Rotate Image  
2. Resize Image  
3. Convert to Grayscale  
4. Blur image  
0. Exit  
Enter your choice: 1  
Enter image filename: benfica.jpg  
Image rotated and saved as 'rotated_image.jpg'.
```

- Pedido para rodar imagem

## ■ 3. Resultado – Exercício 1



- Diretório com as imagens modificadas guardadas

### ■ 3. Evidencias – Exercício 2 (Servidor)

```
def process_image(task_id):  
    if task_id in task_status and task_status[task_id] == "In Progress":  
        image_path = os.path.join(os.getcwd(), f"{task_id}.jpg")  
  
        image = Image.open(image_path)  
  
        image = image.filter(ImageFilter.BLUR)  
        image = image.convert("L")  
  
        processed_images_dir = os.path.join(os.getcwd(), "processed_images")  
  
        processed_image_path = os.path.join(processed_images_dir, f"{task_id}.jpg")  
        image.save(processed_image_path)  
  
        task_status[task_id] = "Completed"  
  
        os.remove(image_path)  
  
    return True  
else:  
    return False
```

Assim que no menu o cliente escolha completar um processo a imagem passa a ser processada, isto é, é modificada para cor cinza e fica desfocada e por último é enviada para a pasta, *processed\_images*.



### ■ 3. Evidencias – Exercício 2 (Servidor)

```
def create_task(image_data):  
    task_id = str(uuid.uuid4()) #string aleatoria  
  
    image_path = os.path.join(os.getcwd(), f"{task_id}.jpg") # o nome da imagem vai ser o id + .jpg  
    with open(image_path, "wb") as f:  
        f.write(image_data.data)  
  
    task_status[task_id] = "Created"  
  
    return task_id  
  
def get_task_status(task_id):  
    status = task_status.get(task_id, "Invalid Task ID")  
    return status
```

- Adiciona uma nova tarefa à lista de tarefas e coloca num diretório
- Retorna o estado da tarefa

### ■ 3. Evidencias – Exercício 2 (Servidor)

```
def get_processed_image(task_id):
    image_path = os.path.join(os.getcwd(), "processed_images", f"{task_id}.jpg")
    if os.path.exists(image_path):
        with open(image_path, "rb") as f:
            image_data = Binary(f.read())
        return image_data
    else:
        return None

Rui Cerqueira
def change_task_status(task_id, new_status):
    if new_status == "Completed":
        process_result = process_image(task_id)
        if not process_result:
            return False

    current_status = get_task_status(task_id)

    if current_status == "Created" and new_status == "In Progress":
        task_status[task_id] = "In Progress"
        return True
    elif current_status == "In Progress" and new_status == "Completed":
        task_status[task_id] = "Completed"
        return True
    else:
        return False
```

- Retorna a imagem processada
- Muda o estado da tarefa consoante a vontade do cliente (não deixa a imagem no estado *created* passar para o estado *completed*)

### ■ 3. Evidencias – Exercício 2 (Servidor)

```
with SimpleXMLRPCServer(('0.0.0.0', 8000), requestHandler=RequestHandler) as server:
    server.register_introspection_functions()

    server.register_function(create_task, 'create_task')
    server.register_function(get_task_status, 'get_task_status')
    server.register_function(get_processed_image, 'get_processed_image')
    server.register_function(process_image, 'process_image')
    server.register_function(change_task_status, 'change_task_status')

    print("Server is running on port 8000...")
    server.serve_forever()
```

- Regista as funções
- Espera por *requests* por parte do cliente

### ■ 3. Evidencias – Exercício 2 (Cliente)

```
def create_task(image_path):
    with open(image_path, "rb") as f:
        image_data = xmlrpc.client.Binary(f.read())

    task_id = proxy.create_task(image_data)
    return task_id

Rui Cerqueira
def change_task_status(task_id, new_status):
    result = proxy.change_task_status(task_id, new_status)
    if result:
        print(f"Task {task_id} status changed to {new_status}.")
    else:
        print(f"Failed to change status for Task {task_id}.")

Rui Cerqueira
def get_task_status(task_id):
    status = proxy.get_task_status(task_id)
    return status
```

- Cria uma nova task
- Muda o estado da task (verifica se o resultado da mudança está correta)
- Retorna o estado da tarefa

### ■ 3. Evidencias – Exercício 2 (Cliente)

```
def menu():
    while True:
        print("\nMenu:")
        print("1. Create Task")
        print("2. Change Task Status to 'In Progress'")
        print("3. Change Task Status to 'Completed'")
        print("4. Check Task Status")
        print("5. Quit")

        choice = input("Enter your choice (1/2/3/4/5): ")

        if choice == "1":
            image_path = input("Enter the path of the image: ")
            task_id = create_task(image_path)
            print(f"Task created. ID: {task_id}")
        elif choice == "2":
            task_id = input("Enter the task ID: ")
            change_task_status(task_id, "In Progress")
        elif choice == "3":
            task_id = input("Enter the task ID: ")
            status = get_task_status(task_id)
            if status == "In Progress":
                change_task_status(task_id, "Completed")
            else:
                print("You can only mark a task as 'Completed' if it's in progress.")
        elif choice == "4":
            task_id = input("Enter the task ID: ")
            status = get_task_status(task_id)
```

- Menu do cliente para poder realizar as suas operações

### ■ 3. Resultados – Exercício 2

```
C:\Users\rascb\OneDrive\Ambiente de Trabalho\LEI\LEI\3 ano\1ºsem  
estre\IS\desafioAsyncRPC>docker-compose up  
[+] Building 0.0s (0/0)  
[+] Running 1/0  
✓ Container desafioasynrpc-rpc-server-1 Created 0.0s  
Attaching to desafioasynrpc-rpc-server-1
```

- Servidor a correr

```
Menu:  
1. Create Task  
2. Change Task Status to 'In Progress'  
3. Change Task Status to 'Completed'  
4. Check Task Status  
5. Quit  
Enter your choice (1/2/3/4/5): 1  
Enter the path of the image: onda.jpg  
Task created. ID: 02e4e8de-2d30-4551-943b-07635ebc33a6
```

- Criar tarefa

### ■ 3. Resultados – Exercício 2

```
Menu:
1. Create Task
2. Change Task Status to 'In Progress'
3. Change Task Status to 'Completed'
4. Check Task Status
5. Quit
Enter your choice (1/2/3/4/5): 2
Enter the task ID: 02e4e8de-2d30-4551-943b-07635ebc33a6
Task 02e4e8de-2d30-4551-943b-07635ebc33a6 status changed to In P
rogress.
```

- Mudar estado da tarefa para *In Progress*

### ■ 3. Resultados – Exercício 2

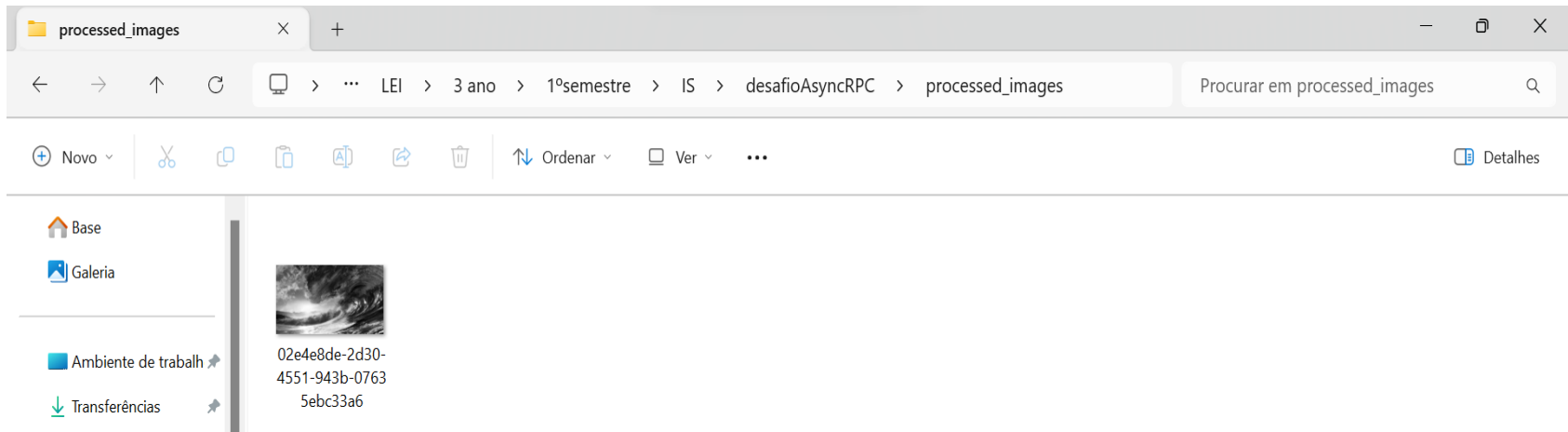
```
Menu:
1. Create Task
2. Change Task Status to 'In Progress'
3. Change Task Status to 'Completed'
4. Check Task Status
5. Quit
Enter your choice (1/2/3/4/5): 3
Enter the task ID: 02e4e8de-2d30-4551-943b-07635ebc33a6
Failed to change status for Task 02e4e8de-2d30-4551-943b-07635ebc33a6.

Menu:
1. Create Task
2. Change Task Status to 'In Progress'
3. Change Task Status to 'Completed'
4. Check Task Status
5. Quit
Enter your choice (1/2/3/4/5): 4
Enter the task ID: 02e4e8de-2d30-4551-943b-07635ebc33a6
Task Status: Completed
```

- Mudar estado da tarefa para *Completed*



## ■ 3. Resultados – Exercício 2



- Dentro da pasta `processed_images` aparece a imagem já processada

### ■ 3. Conclusão



Ao enfrentar estes desafios, encontrei algumas dificuldades ao colocar o servidor em contentores Docker, mas com a orientação dos docentes, consegui superar essa etapa com sucesso. Relativamente aos exercícios propostos, alcancei os objetivos estabelecidos. Em resumo, ao enfrentar estes desafios e devido a alguma pesquisa, constatei que o protocolo de comunicação RPC é amplamente utilizado no processamento de imagens, como é evidenciado por estes desafios.

## ■ 6. Referencias Web e Bibliografias



Código Desafio RPC – Exercício 1: <https://github.com/ruicerqueira231/desafioRPC1>

Código DesafioRPC – Exercício 2: <https://github.com/ruicerqueira231/desafioRPC2>

RPC: <https://tech-lib.wiki/cpp/>

o teu • de partida



Instituto Politécnico  
de Viana do Castelo

[www.ipvc.pt](http://www.ipvc.pt)