Universidade Federal do Sul e Sudeste do Pará
Faculdade de Computação e Engenharia Elétrica
Curso de Engenharia da Computação
Iago Costa das Flores - 201840601017
Juliana Batista da Silva - 201740601024

Sistemas Distribuídos
Atividade Avaliativa 05

Marabá
2021

Iago Costa das Flores - 201840601017
Juliana Batista da Silva - 201740601024

Sistemas Distribuídos
Atividade Avaliativa 05

Relatório apresentado no curso de Engenharia da Computação, turma de 2018 como obtenção de nota parcial na disciplina de Sistemas Distribuídos, ministrada pelo Professor Dr. Warley Muricy Valente Junior.

Marabá
2021

UNIFESSPA

Sumário

1 - Introdução

Este relatório tem como objetivo, demonstrar os resultados da implementação de uma solução desenvolvida para solucionar o problema de exclusão mútua envolvendo o conhecido caso do barbeiro, onde vários clientes necessitam utilizar o mesmo recurso mas não podem consumir o recurso ao mesmo tempo.

O algoritmo desenvolvido, faz o gerenciamento das requisições dos novos clientes, para decidir se o cliente que fez a requisição poderá ou não entrar na região crítica para consumir o recurso, no caso o barbeiro. Para o desenvolvimento da atividade, foi utilizado a linguagem python e a biblioteca pyro, desenvolvida para possibilitar a utilização de chamadas remotas em python.

2 - Desenvolvimento do projeto

Para a elaboração do código foi utilizada a linguagem de programação Python em conjunto com as bibliotecas threading para criar threads, time para dar pausas na execução do código e Pyro.api para simular o RMI com o python.

2.1 Clientes

Na figura 01 temos o script da parte de configuração para um cliente, para fazer novos clientes basta mudarmos o nome do cliente nas linhas 8, 25, 27, 29 e executar o script no terminal, e terá mais um cliente replicado para testes.

```python
import random
import threading
import time
import Pyro5.api
@Pyro5.api.expose
class Cliente:
    def __init__(self):
        self.id = 'Cliente 1'
    def concorrer(self, id, rc, cont, uri):
        message = id+' '+rc+' '+cont+' '+uri
        print("Chamada de função por: "+str(uri))
        return message
daemon = Pyro5.api.Daemon()        # make a Pyro daemon
uri_cliente = daemon.register(Cliente)    # register the greeting maker as a Pyro object
print("Ready. Object uri Cliente =", uri_cliente)       # print the uri so we can use it in the client later
def requisicoes():
    uri_exclusao = input("What is the Pyro uri = ").strip()
    exclusao = Pyro5.api.Proxy(uri_exclusao)     # get a Pyro proxy to the greeting object
    time.sleep(10)
    lista1 = [random.randint(0,50) for x in range(20)]
    lista2 = [random.randint(0,50) for x in range(20)]
    lista3 = [random.randint(0,50) for x in range(20)]
    for item in range(0, 20):
        time.sleep(8)
        print(exclusao.concorrer("Cliente1", 'Barba', str(lista1[item]), str(uri_cliente)))
        time.sleep(1)
        print(exclusao.concorrer("Cliente1", 'Bigode', str(lista2[item]), str(uri_cliente)))   # call method normally
        time.sleep(3)
        print(exclusao.concorrer("Cliente1", 'Cabelo', str(lista3[item]), str(uri_cliente)))   # call method normally

thread1 = threading.Thread(target=daemon.requestLoop, args=())
thread2 = threading.Thread(target=requisicoes, args=())
thread1.start()
thread2.start()
thread1.join()
thread2.join()
```

Figura 01: Script do cliente.

Na figura 02 temos um exemplo de execução do cliente1.py. É válido lembrar que o client1.py precisa se conectar ao objeto do algoritmo de exclusao.py que fará o tratamento e enfileiramento das requisições vindas dos clientes antes de serem redirecionados para a região crítica representada pelo barbeiro.py

```
server@Ubuntu-PC:~/Documents/Estudos/Per-odo-6-facul/SISTEMAS-DISTRIBUIDOS/atividades/atividade07/pyro$ python3 client1.py
Ready. Object uri Cliente = PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577
What is the Pyro uri = PYRO:obj_b4ec778fde0b49efaf566eb29e7d0007@localhost:34003
Mensagem enviada: Cliente1 Barba 43 PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577
Mensagem enviada: Cliente1 Bigode 16 PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577
Mensagem enviada: Cliente1 Cabelo 28 PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577
Mensagem enviada: Cliente1 Barba 6 PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577
Mensagem enviada: Cliente1 Bigode 18 PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577
Mensagem enviada: Cliente1 Cabelo 19 PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577
Mensagem enviada: Cliente1 Barba 30 PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577
Mensagem enviada: Cliente1 Bigode 47 PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577
Mensagem enviada: Cliente1 Cabelo 24 PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577
```

barbeiro
exclusao
cliente1
cliente2

Figura 02: Execução e print do terminal de um cliente

2.2 Barbeiro

Na figura 03 temos o Script do barbeiro, é possível notar da linha 05 até 30 a classe do barbeiro com os métodos cortarCabelo(), cortarBigode(), cortarBarba() e a função de execução desses métodos que seria a trabalhar(). Da linha 31 até 37 temos a configuração do objeto RMI do barbeiro e criação e execução da thread com um loop para ficar escutando as mensagens recebidas pelo algoritmo de exclusão.

```python
import threading
import Pyro5.api
import time
You, 17 hours ago | 1 author (You)
@Pyro5.server.expose
class Barbeiro(object):
    def __init__(self):
        self.nome = 'Barbeiro'
    def cortarCabelo(self, uri):
        print("{} cortando cabelo".format(uri))
        time.sleep(5)
        print("{} acabou de cortar o cabelo".format(uri))
    def cortarBarba(self, uri):
        print("{} cortando barba".format(uri))
        time.sleep(4)
        print("{} acabou de cortar a barba".format(uri))
    def cortarBigode(self, uri):
        print("{} cortando bigode".format(uri))
        time.sleep(3)
        print("{} acabou de cortar o bigode".format(uri))
    def trabalhar(self, id, rc, cont, uri):
        message = id + " " + rc + " " + cont + " " + uri
        metodo = rc
        print("Cliente da vez:" + message)
        if metodo == 'Cabelo':
            self.cortarCabelo(uri)
        elif metodo == 'Barba':
            self.cortarBarba(uri)
        elif metodo == 'Bigode':
            self.cortarBigode(uri)      You, 17 hours ago • ===
        return "Cliente atendido: "+message+' --- Barbeiro livre novamente!!!'
daemon = Pyro5.api.Daemon()  # make a Pyro daemon
uri = daemon.register(Barbeiro)  # register as a Pyro object
print("Ready. Object uri Barbeiro = ", uri)  # print the uri so we can use it in the client later
thread1 = threading.Thread(target=daemon.requestLoop, args=())
# ligar thread
thread1.start()
thread1.join()
```

Figura 03: *Script do barbeiro.py*

Na figura 04 temos os logs de execução do barbeiro, é possível verificar o recebimento de mensagens do algoritmo de exclusão e a execução da função trabalhar.

```
server@Ubuntu-PC:~/Documents/Estudos/Per-odo-6-facul/SISTEMAS-DISTRIBUIDOS/atividades/atividade07/pyro$ python3 barbeiro.py
Ready. Object uri Barbeiro =  PYRO:obj_9c59843945a0470f8c481b6705fa0919@localhost:45597
Cliente da vez:Cliente2 Barba 7 PYRO:obj_fad0415b58a941be9030079da3e73ac9@localhost:39377
PYRO:obj_fad0415b58a941be9030079da3e73ac9@localhost:39377 cortando barba
PYRO:obj_fad0415b58a941be9030079da3e73ac9@localhost:39377 acabou de cortar a barba
Cliente da vez:Cliente1 Bigode 16 PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577
PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577 cortando bigode
PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577 acabou de cortar o bigode
Cliente da vez:Cliente1 Barba 6 PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577
PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577 cortando barba
PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577 acabou de cortar a barba
Cliente da vez:Cliente1 Bigode 18 PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577
PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577 cortando bigode
PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577 acabou de cortar o bigode
Cliente da vez:Cliente1 Cabelo 19 PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577
PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577 cortando cabelo
PYRO:obj_e297f3aeb9e649bfaea63b9fd7d4d9dd@localhost:44577 acabou de cortar o cabelo
Cliente da vez:Cliente2 Bigode 7 PYRO:obj_fad0415b58a941be9030079da3e73ac9@localhost:39377
PYRO:obj_fad0415b58a941be9030079da3e73ac9@localhost:39377 cortando bigode
PYRO:obj_fad0415b58a941be9030079da3e73ac9@localhost:39377 acabou de cortar o bigode
```

barbeiro
exclusao
cliente1
cliente2

Figura 04: *Execução do barbeiro.py.*

2.3 Exclusão

Na figura 05 temos o algoritmo de exclusão usado para receber e enfileirar as requisições vindas dos clientes de acordo com o valor de seu contador recebido junto com o restante da mensagem. A sua estrutura é dividida em duas threads que ficam rodando um loop para recebimento de requisições por parte dos clientes através da biblioteca Pyro (rmi), que seria a thread 1. Na thread 2 a lista de requisições já está ordenada aguardando apenas o envio realizado a cada um segundo para evitar problemas de execução.

```python
import threading
import Pyro5.api
import time
lista = []
lista_ordenada = []
You, 19 hours ago | 1 author (You)
@Pyro5.server.expose      You, 21 hours ago • ===
class Exclusao(object):
    def __init__(self):
        self.nome = 'Exclusao'
    def concorrer(self, id, rc, cont, uri):
        message = id + " " + rc + " " + cont + " " + uri
        lista.append(message.split())
        return "Mensagem enviada: "+message
def lista_ordenada(lista):
    return sorted(lista, key = lambda x: int(x[2]))
def enviar_mensagem():
    time.sleep(15)
    barbeiro = Pyro5.api.Proxy(uri_barbeiro)
    while True:
        try:
            time.sleep(1)
            if lista:
                lista_ordered = lista_ordenada(lista)
                resposta = barbeiro.trabalhar(lista_ordered[0][0], lista_ordered[0][1], lista_ordered[0][2], lista_ordered[0]
                lista.remove(lista_ordered[0])
                print("Tamanho da lista "+str(len(lista)))
                print(resposta)
            else:
                print("Lista vazia \n Barbeiro aguardando clientes!!!")
                time.sleep(5)
        except Exception as e:
            print(e)
daemon = Pyro5.api.Daemon()  # make a Pyro daemon
uri = daemon.register(Exclusao)  # register as a Pyro object
print("Ready. Object uri Exclusao = ", uri)  # print the uri so we can use it in the client later
uri_barbeiro = input("What is the Pyro uri = ").strip()
barbeiro = Pyro5.api.Proxy(uri_barbeiro)     # get a Pyro proxy to the greeting object
thread1 = threading.Thread(target=daemon.requestLoop, args=())
thread2 = threading.Thread(target=enviar_mensagem, args=())
thread1.start()
thread2.start()
thread1.join()
thread2.join()
```

Figura 05: *Script de exclusão.py*

Na figura 06 temos os prints de execução. É possível notar a conexão feita entre o algoritmo de exclusão e o algoritmo do barbeiro no início. Após feita a conexão a exclusão irá ordenar as requisições dos clientes e enviar uma a uma para o barbeiro “trabalhar”.

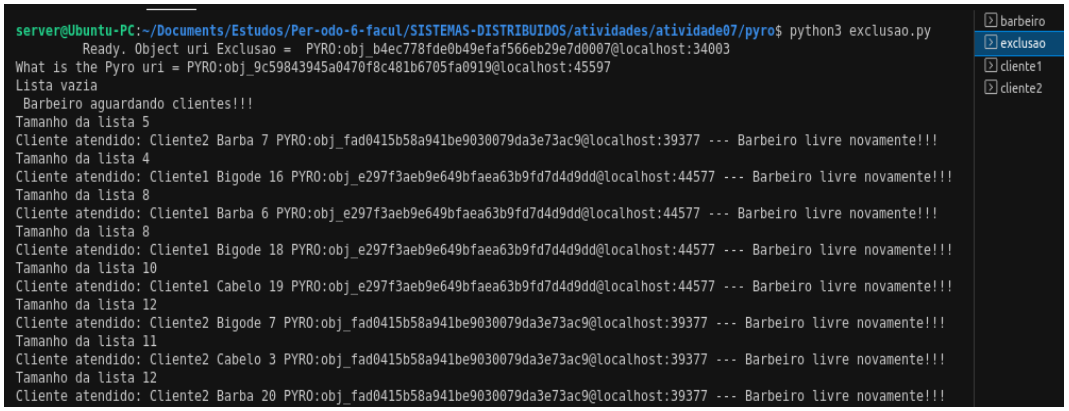

Figura 06: *Execução do algoritmo de exclusão.py*

3 - Conclusão

Ao longo do desenvolvimento da atividade surgiram alguns obstáculos relacionados ao entendimento e à complexidade da questão. Os assuntos relacionados à chamadas remotas foram passados em aulas mas utilizando o RMI em java, que além de se tratar de uma linguagem fortemente tipada, ainda é muito massante, deixando o script final com muitas linhas de código. Pela simplicidade da escrita, optou-se por utilizar python, mas com essa decisão ainda sim não tornou o desenvolvimento mais fácil, tendo em vista de que apesar dos contras em se utilizar java, o RMI é uma opção nativa da própria linguagem, o que diminuiria os problemas de bugs, inconsistências e incompatibilidade utilizando uma solução não nativa como a do python.

Apesar dos problemas que surgiram na implementação do código, foi possível solucionar o problema proposto no roteiro da atividade, onde o recurso em questão, no caso o barbeiro, só poderá ser consumido apenas por um cliente de cada vez.

4 - Referências

Tanenbaum, A.S.; Steen, M.V. 2002. **Distributed Systems - Principles and Paradigms**. Prentice Hall.

RELÓGIOS de Lamport: Algoritmos Distribuídos - Exclusão Mútua. 1. [*S. l.*], 2021. Disponível em: https://pt.wikipedia.org/wiki/Rel%C3%B3gios_de_Lamport. Acesso em: 2 ago. 202