

Para esta atividade, foi desenvolvido uma comunicação do tipo cliente-servidor em python pelo método RMI, com a biblioteca Pyro5.

Os códigos do cliente e servidor estão apresentados abaixo, mas eles também podem ser visualizados neste link:

<https://github.com/chsantos234/lab-sistemas-distribuidos/tree/main/atv4>

1. Códigos do Servidor

```
import Pyro5.api
from random import randint

# classe da atividade
@Pyro5.api.expose
class GreetingMaker():
    def get_fortune(self,name):
        value = randint(0,10000)
        return f"Hello, {name}. Here is your fortune message:\nTomorrow's lucky number is {value}."

# classe nova adicionada (duas funções)
@Pyro5.api.expose
class Math():
    def sum(self,valores):
        return sum(valores)

    def mult(self,valores):
        multi = 1
        for i in valores:
            multi *= i
        return multi
```

Figura 1 instâncias do servidor.

Na Figura 1 acima, é possível observar as duas classes de instâncias presentes no código do servidor, a classe de exemplo GreetingMaker e a nova classe Math.

A classe GreetingMaker apresenta a função `get_fortune(self,name)`, que recebe um nome e retorna uma mensagem com um “número da sorte” aleatório entre 0 e 10000.

A classe Math apresenta a função `sum(self,valores)` que soma todos os números presentes no parâmetro valores, e a função `mult(self,valores)` que irá multiplicar os valores em ordem.

```
24
25 daemon = Pyro5.server.Daemon() # Pyro daemon
26 ns = Pyro5.api.locate_ns() # Pesquisa do servidor
27
28 # registro das classes como objetos Pyro
29 uri = daemon.register(GreetingMaker)
30 uri2 = daemon.register(Math)
31
32 # registro dos objetos com um nome no servidor
33 ns.register("example.greeting", uri)
34 ns.register("example.math", uri2)
35
36 print("Ready.")
37
38 # inicializa o loop de eventos do servidor e espera uma conexão
39 daemon.requestLoop()
```

Figura 2 inicialização do servidor.

A Figura 2 apresenta a inicialização do servidor RMI com o registro das classes Math e GreetingMaker em URIs, e o registro dessas URIs no servidor para serem disponibilizadas ao cliente.

2. Códigos do Cliente

```
1 import Pyro5.api
2
3 math_obj = Pyro5.api.Proxy("PYRONAME:example.math")
4 greeting_obj = Pyro5.api.Proxy("PYRONAME:example.greeting")
5 print('funções do servidor:\n- greetings\n- sum\n- multip')
```

Figura 3 inicialização do cliente.

A Figura 3 mostra a inicialização do cliente pela leitura das URIs no servidor RMI, retornando objetos proxy que serão utilizados pelo cliente para a execução das funções presentes na Figura 1.

```

7   while True:
8       try:
9           function_call = input(':')
10          if function_call == 'exit': raise KeyboardInterrupt
11
12          elif function_call == 'greetings':
13              name = input('What is your name? ')
14              print(greeting_obj.get_fortune(name))
15
16          elif function_call == 'sum':
17              valores = input('digite os valores a serem somados\n:').split(' ')
18              valores = list(map(lambda x: int(x),valores))
19              print(math_obj.sum(valores))
20
21          elif function_call == 'multip':
22              valores = input('digite os valores a serem multiplicados\n:').split(' ')
23              valores = list(map(lambda x: int(x),valores))
24              print(math_obj.mult(valores))
25
26          else:
27              raise ModuleNotFoundError
28
29      except KeyboardInterrupt:
30          print("finalizando programa:")
31          break
32      except ModuleNotFoundError:
33          print(f'nenhum módulo com nome de {function_call}')
34      except Exception as e:
35          print(e)

```

Figura 4 execução e tratamento de erro do cliente.

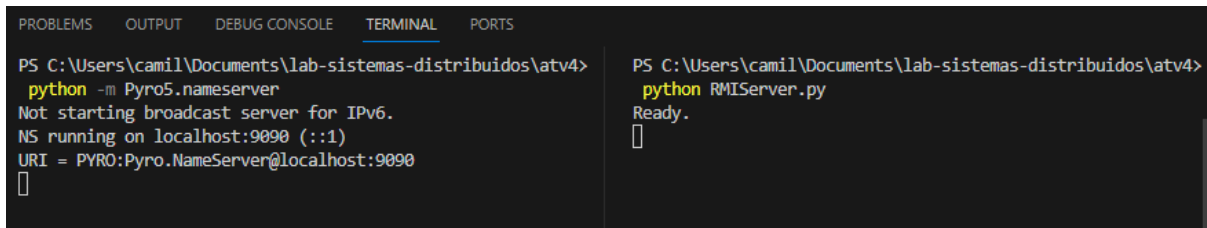
A Figura 4 acima apresenta o loop while presente no cliente para a chamada de input do usuário, e a execução da função correspondente ao input.

Ademais, o código apresenta tratamento de erro para eventuais exceções, com mensagens personalizadas para as exceções do tipo KeyboardInterrupt, para finalização do cliente pelo input 'exit' ou pelo ctrl + C, e do tipo ModuleNotFound quando o input não corresponder a alguma função cadastrada.

3. Execução no terminal

Para executar o servidor e cliente RMI, são necessários três terminais executando os seguintes comandos:

1. python -m Pyro5.nameserver
2. python RMIServer.py
3. python RMIClient.py

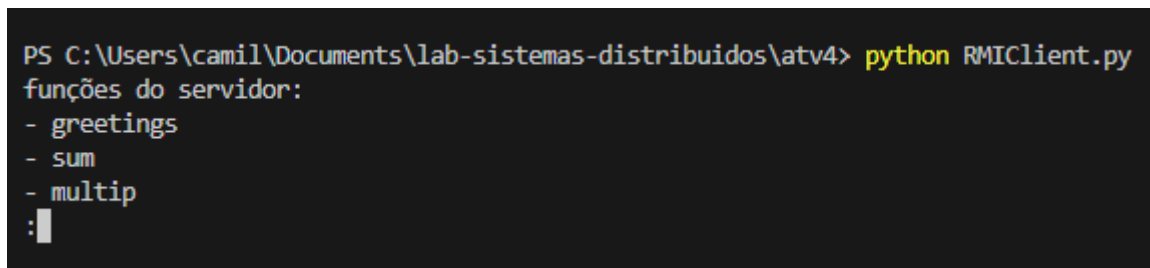


```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\camil\Documents\lab-sistemas-distribuidos\atv4> python -m Pyro5.nameserver
Not starting broadcast server for IPv6.
NS running on localhost:9090 (::1)
URI = PYRO:Pyro.NameServer@localhost:9090
[]

PS C:\Users\camil\Documents\lab-sistemas-distribuidos\atv4> python RMIServer.py
Ready.
[]
```

Figura 5 inicialização do Pyro5 e servidor.

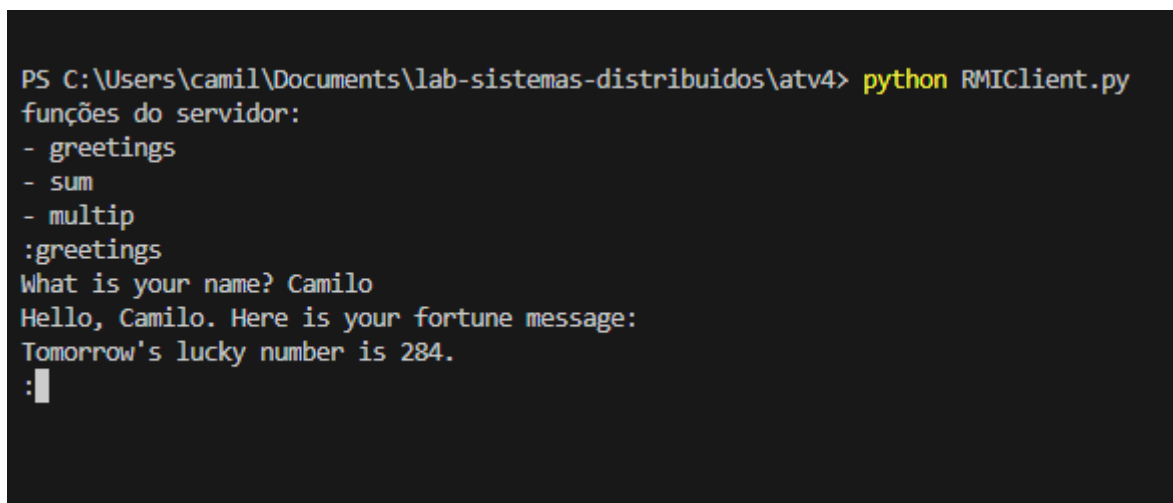
A Figura 5 mostra a utilização dos comandos `python -m Pyro5.nameserver` e `python RMIServer.py` para a inicialização do Pyro name server e o servidor RMI respectivamente.



```
PS C:\Users\camil\Documents\lab-sistemas-distribuidos\atv4> python RMIClient.py
funções do servidor:
- greetings
- sum
- multiplic
:
[]
```

Figura 6 print das funções disponíveis.

A Figura 6 mostra a inicialização do cliente RMI pelo comando `python RMIClient.py`, e como ele apresenta as funções disponíveis para o usuário por um print.



```
PS C:\Users\camil\Documents\lab-sistemas-distribuidos\atv4> python RMIClient.py
funções do servidor:
- greetings
- sum
- multiplic
:greetings
What is your name? Camilo
Hello, Camilo. Here is your fortune message:
Tomorrow's lucky number is 284.
:
[]
```

Figura 7 função `get_fortune`.

A Figura 7 mostra a utilização da função `get_fortune`, utilizando “Camilo” como nome de entrada.

```
:sum
digite os valores a serem somados
:2 5 6 4 3
20
:
```

Figura 8 função sum.

A Figura 8 mostra a execução da função sum com as entradas 2 5 6 4 3.

```
:multip
digite os valores a serem multiplicados
:3 6 9
162
:
```

Figura 9 função mult.

A Figura 9 mostra a execução da função mult com as entradas 3 6 9.

```
:test
nenhum módulo com nome de test
:multip
digite os valores a serem multiplicados
:a
invalid literal for int() with base 10: 'a'
:exit
finalizando programa:
PS C:\Users\camil\Documents\lab-sistemas-distribuidos\atv4> |
```

Figura 10 exemplificação das exceções.

Por fim, a Figura 10 mostra a exemplificação de algumas exceções como a ModuleNotFound, ValueError e KeyboardInterrupt quando o input é igual a "exit".