

Alunos:

Victor Augusto Souza de Oliveira DRE: 113044501

Lucas Clemente Miranda Braga DRE: 115159700

Funcionamento Geral

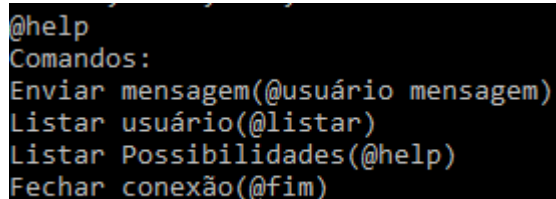
O cliente se conecta ao servidor.

Solicita o nome do usuário como entrada e envia para o servidor uma requisição de login com o nome como parâmetro. Além disso, passa também como parâmetro a função print que será executada no cliente sempre que chegar um evento. Após esse passo o cliente já aparece na lista de usuários ativos do servidor.

Após o login, o cliente solicita os comandos possíveis de serem enviados ao servidor.

Em seguida entra em modo ocioso, onde já é possível enviar comandos e mensagens utilizando “@” na frente indicando que se trata de um usuário/comando e também receber mensagens.

Segue os comandos abaixo, eles podem ser obtidos digitando *@help*:



```
@help
Comandos:
Enviar mensagem(@usuário mensagem)
Listar usuário(@listar)
Listar Possibilidades(@help)
Fechar conexão(@fim)
```

Testes Realizados

- 1) o usuário torna-se ativo e visualiza a lista de outros usuários ativos (experimentar com a lista vazia e não-vazia)

Usuário torna-se ativo ao entrar com seu nome. Em seguida e sem necessidade de entrada por parte do cliente, já é printado os comandos que podem ser utilizados.

Optamos pelo comando *@listar*, que lista os usuários ativos. O resultado foi o próprio usuário, ou seja, não há outro usuário ativo.

```

victor@DESKTOP-NU99129:~/sistdist/distributed_systems/tarefa_4$ python3 cli.py
<netref class 'rpyc.core.netref.__main__.Echo'>
ECHO
Entre com o seu nome de usuário, sem espaços: Victor

Cadastro Realizado com Sucesso

Comandos:
Enviar mensagem(@usuário mensagem)
Listar usuário(@listar)
Listar Possibilidades(@help)
Fechar conexão(@fim)

@listar
Victor,

```

Com outros usuários ativos:

```

victor@DESKTOP-NU99129:~/sistdist/distributed_systems/tarefa_4$ python3 cli.py
<netref class 'rpyc.core.netref.__main__.Echo'>
ECHO
Entre com o seu nome de usuário, sem espaços: Lucas

Cadastro Realizado com Sucesso

Comandos:
Enviar mensagem(@usuário mensagem)
Listar usuário(@listar)
Listar Possibilidades(@help)
Fechar conexão(@fim)

@listar
Victor,Lucas,

```

2. o usuário inicia uma conversa com um; e depois com vários usuários ativos ao mesmo tempo (comunicação entre pares):

Usuário inicia conversa com clientes Lucas e Victor utilizando @nome_de_usuario.

```

Entre com o seu nome de usuário, sem espaços: Caio

Cadastro Realizado com Sucesso

Comandos:
Enviar mensagem(@usuário mensagem)
Listar usuário(@listar)
Listar Possibilidades(@help)
Fechar conexão(@fim)

@Lucas boa tarde!
@Victor boa noite!

```

3. o usuário recebe mensagens de um; e depois de vários usuários ativos ao mesmo tempo;

Usuário Tales recebendo mensagem de dois outros usuários, Vanessa e Victor. O nome de quem enviou aparece no início da mensagem.

```
Entre com o seu nome de usuário, sem espaços: Tale  
Cadastro Realizado com Sucesso  
Comandos:  
Enviar mensagem(@usuário mensagem)  
Listar usuário(@listar)  
Listar Possibilidades(@help)  
Fechar conexão(@fim)  
  
Vanessa : bom dia  
Victor : bom descanso  
Victor : você merece
```

4. o usuário sai da aplicação e deixa de aparecer na lista de usuários ativos;

Após usuária Vanessa sair do chat , deixa de aparecer na lista de usuários ativos.

```
@listar  
Victor,Tales,Vanessa,  
  
Usuário Vanessa saiu do chat.  
@listar  
Victor,Tales,
```

5. um usuário sai da aplicação antes de finalizar a conversa com outros usuários

Após um usuário sair do chat nenhuma funcionalidade muda para os que continuaram ativos, só deixa de ser possível enviar mensagem para o usuário offline, ao tentar é exibida a mensagem *Usuário : Fulano não está no chat*

```
Usuário Vanessa saiu do chat.  
@Tales ainda é possível enviar mensagem  
@Vanessa menos para a Vanessa  
  
Usuário :  
Vanessa  
Não está no chat
```

6. usuário manda mensagem para outros dois usuários uma só vez

É possível enviar mensagens para mais de um usuário de uma só vez, basta colocar os nomes do clientes com @ na frente como a figura abaixo demonstra.

```
@listar
Victor,Tales,José,
@Victor @Tales essa mensagem será entregue aos dois
```

7. usuário tenta se cadastrar porém já tem alguém com aquele nome

A aplicação não permite cadastrar caso já haja alguém com o mesmo nome ativo e solicita um novo nome.

```
victor@DESKTOP-NU99129:~/sistdist/distributed_systems/tarefa 4$ python3 cli.py
<netref class 'rpyc.core.netref.__main__.Echo'>
ECHO
Entre com o seu nome de usuário, sem espaços: Victor
Nome já cadastrado
Entre com o seu nome de usuário, sem espaços: _
```

Alterações e Adaptações Realizadas

Mudanças na arquitetura de sistema

O planejado era ter a camada de processamento e dados totalmente contida no servidor e a camada de usuário no cliente, na prática a camada de processamento ficou dividida entre o servidor e o cliente. No cliente ficaram processamentos simples(exemplo: determinar qual comando o usuário efetuou) que permitem e que liberam o servidor para focar em coisas mais complexas e importantes (manter lista de mensagens e clientes atualizadas).

O planejado era ter um dicionário com nome e endereço do cliente, porém utilizando RPC com função de callback assíncrona não foi necessário salvar diretamente o endereço de cada usuário. A informação de identificação foi o próprio nome do cliente, que foi salvo na classe instanciada ao fazer uma conexão, a função de callback também foi salva dessa forma.

Outras alterações relevantes:

Implementação da função de callback assíncrona

Para implementação da função de callback assíncrona foi utilizado como consulta a documentação da biblioteca rpyc, em especial a parte que trata de Eventos e Operações assíncronas. A implementação dessa função de callback foi feita da seguinte forma:

Quando a aplicação do cliente é iniciada é criada uma background thread(“thread de fundo”) que ficará “escutando” o retorno do servidor.

Ao solicitar login é passada a função de callback. O servidor então cria uma nova thread cuja função é ficar constantemente checando se há mensagens no dicionário endereçadas ao cliente. Quando há uma mensagem a thread chama a função de callback e passa como argumento a mensagem. A thread de fundo então recebe esse evento e executa no cliente a função de callback, que então printa a mensagem.

