IPS Instituto Politécnico de Setúbal Escola Superior de Tecnologia de Setúbal	COMPUTAÇÃO PARALELA E DISTRIBUÍDA 2022/2023
	Laboratório 07: Serviço de Chat

Objetivos do laboratório

Neste laboratório iremos implementar um serviço de chat com suporte a multi-utilizadores. O trabalho consiste na implementação de um servidor em **Python** e dos clientes do chat. Exemplo de utilização na aplicação cliente:

```
Username?
> Bob
You are now connected, Bob!
> Hello everyone
(Bob) Hello everyone
(John) Hi
(Rock) Hey
> exit
Goodbye Bob!
```

Este laboratório tem por base o ficheiro **chat_base.zip** que contém uma implementação inicial do servidor de chat (*server.py*) e do cliente (*client.py*).

Crie um projeto no seu IDE com estes dois ficheiros, e execute cada um deles em separado (primeiro o servidor e depois o cliente). Leia o código de ambos os ficheiros, tente compreender como funcionam e verifique que o servidor faz o eco das mensagens enviadas pelo cliente.

Nível 1 – Nome de utilizador

1. Altere o código da aplicação cliente de modo a pedir inicialmente o nome de utilizador. Deverá enviar o nome de utilizador para o servidor antes de qualquer outra comunicação.

2. Modifique o código no servidor de modo a receber o nome do cliente e retornar a confirmação "*You are now connected, X!*". O cliente deverá imprimir esta confirmação na consola.

3. Modifique o necessário de modo que o cliente, após enviar um "*exit*", aguarde pela resposta "Goodbye X!" do servidor antes de desligar o *socket*.

Nível 2 – Múltiplos clientes

Execute o servidor e dois clientes ao mesmo tempo, e verifique que o segundo cliente bloqueia após o envio do nome de utilizador. O servidor está no ciclo *while* interior bloqueado com a conexão do primeiro cliente.

1. Verifique a documentação de Python sobre *threads* e respetivos métodos e classes em https://docs.python.org/3/library/threading.html.

2. Considere o excerto seguinte, e modifique o servidor de modo a iniciar uma *thread* para lidar com a ligação de um cliente.

```
def handle_client(client_connection):
  # Obtém nome de utilizador
  ...
  print("New client:", username)
  while True:
       # Recebe mensagem do cliente
       if msg == "exit":
            ...
            break
      # Imprime a mensagem do cliente
   # Fecha a ligação do cliente
   print("Client disconnected:", username)



while True:
  # Espera pela ligação do cliente
  # Cria thread
  thread = threading.Thread(target=handle_client,args=[client_connection])
  thread.start()
```

Execute o servidor e as duas instâncias do cliente e teste o funcionamento correto da aplicação.

Nível 3 – Broadcast

Até agora os clientes enviam mensagens, mas apenas recebem as suas próprias mensagens.

1. Altere o código do servidor de modo a criar uma lista de conexões vazia. Após receber a conexão do utilizador, e antes de criar a *thread*, deverá adicionar a conexão à lista de conexões.

2. Da mesma forma, na função que gere a ligação com os clientes no servidor, remova a conexão da lista de conexões após fechar o socket do cliente.

3. Por fim, modifique a mesma função de modo que o servidor envie a mensagem para todos os clientes na lista de conexões. A mensagem deverá conter o nome do utilizador que a enviou entre parênteses (ex: *(Bob) Hello!*).

Teste com dois clientes e verifique que as mensagens de um cliente só são recebidas pelo outro quando o segundo cliente envia ele próprio uma mensagem (ou carrega na tecla *enter*). Porque razão isto acontece?

Nível 4 – Receção de várias mensagens nos clientes

Neste nível pretendemos alterar os clientes de modo a que estes possam estar ativamente à escuta de mensagens sem que tenham obrigatoriamente de as enviar primeiro.

```
def listening_thread(client_socket):
    while True:
        # Recebe mensagem do servidor
        # Imprime a mensagem do servidor


# Pede nome do utilizador
# Envia para o servidor
# Espera pela mensagem "Connected"
# Cria thread para receber mensagens do servidor
…
```

1. Modifique o código do cliente de modo a criar uma *thread* para ficar em ciclo à espera das mensagens enviadas pelo servidor e coloque aí todo o código relacionado com a receção de mensagens. Desta forma é possível isolar as mensagens provenientes do servidor do mecanismo de envio de mensagens.

2. Caso obtenha erros no socket do cliente após enviar "exit", isto surge porque termina o *socket* do cliente antes de terminar a thread que está à escuta de mensagens via *socket*. Para resolver este problema, verifique se a resposta do servidor contém "Goodbye" e nesse caso faça *break* para terminar a *thread.* Mais à frente teste se a *thread* terminou (*thread.join()*) antes de prosseguir com o *break* que termina o *socket* do cliente.

Nível 5 – Enviar uma mensagem a todos os clientes

Modifique o código de modo a que, sempre que um cliente (utilizador) se liga ao servidor, seja enviada uma mensagem a todos os outros clientes.