Computação Distribuída
2019 / 2020

Licenciatura em Engenharia Informática

Lab. 04 – Introdução aos sockets de rede

Introdução

Nesta ficha iremos implementar um pequeno serviço de rede cujo objectivo é fornecer o cálculo de operações matemáticas. O trabalho consiste na implementação de um pequeno servidor em Python que será responsável por fornecer um serviço e um programa cliente que irá consumir o serviço. Exemplo de utilização na aplicação cliente:

```
> 1+1
2
```

A comunicação entre o servidor e os clientes é feita recorrendo a *sockets* TCP/IP. Um *socket* de rede é um ponto de comunicação entre dois programas a correr sobre uma rede de computadores, e pode ser considerado como o acesso mais *low-level* que um programador tem ao subsistema de rede. Para construirmos o servidor e o cliente do serviço, teremos de definir como iremos utilizar os sockets.

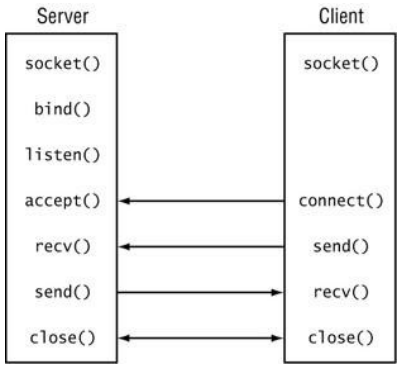
Servidor: O servidor terá inicialmente de definir um **endereço IP** e uma **porta** onde irá estar à escuta de comunicações. Em termos gerais, o servidor tratará de:

1. Criar um *socket* do tipo TCP/IP que lidará com endereços IPv4 (do tipo 127.0.0.1).
2. Associar o *socket* a um par (endereço, porto) usando o comando *bind()*.
3. Ficar à escuta por ligações usando o comando *listen()*.

Cliente: O cliente apenas terá de se ligar ao servidor (sabendo qual o **endereço IP** e a **porta**) e enviar mensagens:

1. Criar um socket do tipo TCP/IP semelhante ao do servidor.
2. Ligar-se ao servidor usando o comando *connect()*.

Após o cliente se ligar ao servidor usando o comando *connect()* o servidor terá de aceitar a ligação usando o comando *accept()*. As comunicações propriamente ditas serão feitas usando os comandos *send()* (ou *sendall*) e *recv()* até terminar as ligações usando o comando *close()*.



Este laboratório tem por base o ficheiro **4-sockets-base.zip** que contém uma implementação inicial do servidor (*server.py*) e do cliente (*client.py*).

Crie um projecto 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 *print* da mensagem enviada pelo cliente.

Nível 1 – Input

Considere o código da aplicação cliente:

1. Altere o código de modo a enviar a mensagem “Hello from the client application!”. Verifique que o servidor escreve a mensagem no seu *output*.

2. Altere o código de modo a pedir a mensagem a enviar ao utilizador. Como apenas pode enviar *bytes* por um socket, utilize o comando *encode()*:

```
msg = input(“> “)
client_socket.send(msg.encode())
```

Nível 2 – Conexões sucessivas

Altere o código do servidor de modo a que o servidor não termine após a primeira conexão de um cliente. Dito de outra forma, espera-se que o servidor possa receber várias ligações sucessivas usando o ciclo *while*.

```
while True:
    # Espera pela ligação do cliente
    # Imprime a mensagem do cliente
    # Fecha a ligação do cliente
```

Coloque o servidor a funcionar e depois execute a aplicação cliente várias vezes. Se tudo estiver a funcionar como se pretende, deverá ver as várias mensagens no *output* do servidor.

Nível 3 – Terminação

Considere o seguinte excerto:

```
while True:
    # do something
    if msg == “exit”:
        break
```

1. Modifique o código do cliente de modo a permitir enviar várias mensagens até o utilizador inserir a palavra “exit”. Experimente enviar várias mensagens até escrever “exit” e repare que o servidor apenas escreve a primeira mensagem de cada vez que executa o cliente.

2. Resolva o problema modificando o código do servidor de modo a permitir receber várias mensagens do mesmo cliente até este enviar a mensagem “exit”:

```
while True:
    # Espera pela ligação do cliente

    while True:
        # Recebe mensagem do cliente
        # Imprime a mensagem do cliente
        if msg == “exit”:
            break

    # Fecha a ligação do cliente
```

Execute o servidor e o cliente, envie várias mensagens e verifique que o servidor as imprime.

Nível 4 – Resposta do servidor

Até agora o cliente envia as mensagens mas não recebe respostas do servidor.

1. Altere o código do servidor de modo a reenviar a mensagem que recebeu do cliente (se este não enviou o “exit”).

2. Altere o código do cliente de modo a ler e imprimir a resposta do servidor de cada vez que envia uma mensagem para o servidor (se não enviou o “exit”, é claro). Deverá obter a resposta do servidor usando o comando *client_socket.recv(1024).decode()*.

Nível 5 – Eval

O serviço implementado até agora é basicamente um serviço de “eco”, onde as mensagens recebidas pelo servidor são novamente reenviadas. Neste último passo pretendemos alterar o servidor para implementar um serviço de resolução de operações matemáticas.

1. Modifique o código do servidor para imprimir a avaliação das mensagens enviadas pelo cliente. Isto é, se o cliente enviar a mensagem “1+1”, deverá imprimir 2 na consola do servidor. Utilize a função *eval().*

2. Modifique o servidor de modo a enviar o resultado do *eval()* como resposta para o cliente. Tome em atenção que a avaliação de operações matemáticas resultará num inteiro, pelo que poderá ter que converter a resposta em string utilizando o comando *str()*.

3. Modifique o servidor de modo a que não falhe no *eval* se o cliente enviar uma mensagem inválida. Utilize *try-except Exception* para enviar uma mensagem em caso de erro.

(fim de enunciado)