

Sistemas Distribuidos

Práctica 1

Nicolás Corsini

UDP-1:

UDP-1.1

Código del cliente:

```
In [1]: 1 import socket
2
3 msg = input("Input message: ")
4 bytes_tx = str.encode(msg)
5
6 server_address = ("127.0.0.1", 6780)
7 socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
8 socket.sendto(bytes_tx, server_address)
9
10 bytes_rx = socket.recvfrom(1024)
11 print("RX: ", bytes_rx)
12 socket.close()
```

Input message: Hello, I'm Client 1
RX: (b"Hello, I'm the server", ('127.0.0.1', 6780))

Código del servidor:

UDP 1.1

```
In [*]: 1 import socket
2
3 msg = "Hello, I'm the server"
4 bytes_tx = str.encode(msg)
5
6 socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
7 socket.bind(("127.0.0.1", 6780))
8
9 while True:
10     bytes_rx = socket.recvfrom(1024)
11     message = bytes_rx[0]
12     address = bytes_rx[1]
13     print(str(message))
14     socket.sendto(bytes_tx, address)
15     if str(message) == "b'close'":
16         socket.close()
17         break
18
```

b"Hello, I'm Client 2"
b"Hello, I'm Client 1"

Como se puede observar, el servidor arranca y se queda esperando un número ilimitado de respuestas. Al ejecutar 2 clientes diferentes, el servidor recibe el envío de los dos y les responde.

UDP-1.2

Código del servidor:

```
In [1]: 1 import socket
2
3 msg = "Hello, I'm the server"
4 bytes_tx = str.encode(msg)
5
6 port = input("Connection port: ")
7
8 # Check if port is a number
9 while port.isnumeric() == False:
10     port = input("Port must be numeric: ")
11
12 port = int(port)
13
14 # If port is not a valid number: print("Invalid port")
15 try:
16     socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
17     socket.bind(("127.0.0.1", port))
18     print("Server up and listening!")
19
20     while True:
21         bytes_rx = socket.recvfrom(1024)
22         message = bytes_rx[0]
23         address = bytes_rx[1]
24         print("\nConnection received.")
25         print("Message received: ", str(message))
26         print("Client address: ", str(address))
27         socket.sendto(bytes_tx, address)
28         if str(message) == "b'close'":
29             print("Received close instruction.")
30             print("Server shutting down")
31             socket.close()
32             break
33 except:
34     print("Invalid Port")
```

```
Connection port: aaa
Port must be numeric: 3748743
Invalid Port
```

El servidor comprueba que el puerto especificado es numérico y que es válido. Si el puerto es válido muestra:

```
Connection port: 6780
Server up and listening!
```

Además, si el cliente le mandase el mensaje “close”, el servidor se pararía automáticamente.

Código del cliente:

UDP 1.2

```
In [1]: 1 import socket
2
3 msg = "Hello I'm client 3"
4 bytes_tx = str.encode(msg)
5
6 ip = input("Enter server IP: ")
7 port = input("Enter server port: ")
8 # Check if port is a number
9 while port.isnumeric() == False:
10     port = input("Port must be numeric: ")
11
12 port = int(port)
13
14 server_address = (ip, port)
15
16 try:
17
18     socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
19     socket.sendto(bytes_tx, server_address)
20
21     bytes_rx = socket.recvfrom(1024)
22     print("RX: ", bytes_rx)
23     socket.close()
24
25 except:
26
27     print("Server not running on that address")
```

```
Enter server IP: 127.1.1.1
Enter server port: 6780
Server not running on that address
```

Si no encuentra al servidor, muestra el mensaje de “Server not running on that address”.

Si la address es correcta, el cliente manda el mensaje y el servidor lo recibe:

```
Connection received.
Message received: b"Hello I'm client 3"
Client address: ('127.0.0.1', 55172)
```

En el cliente se muestra la respuesta del servidor:

```
Enter server IP: 127.0.0.1
Enter server port: 6780
RX: (b"Hello, I'm the server", ('127.0.0.1', 6780))
```

UDP-1.3

Código del servidor:

UDP 1.3

```
In [1]: 1 import socket
2 import pickle
3
4 msg = "Hello, I'm the server"
5 bytes_tx = str.encode(msg)
6
7 port = input("Connection port: ")
8
9 # Check if port is a number
10 while port.isnumeric() == False:
11     port = input("Port must be numeric: ")
12
13 port = int(port)
14
15 # If port is not a valid number: print("Invalid port")
16 try:
17     socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
18     socket.bind(("127.0.0.1", port))
19     print("Server up and listening!")
20
21     while True:
22         bytes_rx = socket.recvfrom(1024)
23         message = pickle.loads(bytes_rx[0])[0]
24         client_id = pickle.loads(bytes_rx[0])[1]
25         address = bytes_rx[1]
26         print("\nConnection received.")
27         print("Message received: ", str(message))
28         print("Number of characters of message", len(message))
29         print("Client Id: ", str(client_id))
30         print("Client address: ", str(address))
31         bytes_tx = str.encode(str(len(message)) + " characters received")
32         socket.sendto(bytes_tx, address)
33         if str(message) == "close":
34             print("Received close instruction.")
35             print("Server shutting down")
36             socket.close()
37             break
38     except:
39         print("Invalid Port")
```

El servidor recibe un array empaquetado con pickle y lo desempaqueta guardando el id del cliente y el texto del mensaje. Le manda al cliente el número de caracteres de su mensaje.

Código del cliente:

UDP 1.3

```
In [3]: 1 import socket
2 import pickle
3
4 client_id = 3
5
6 msg = input("Input message: ")
7 msg_encoded = pickle.dumps(msg, 0)
8
9 id_encoded = pickle.dumps(str(client_id), 0)
10
11 array = [msg, str(client_id)]
12 array_encoded = pickle.dumps(array, 0)
13
14 # bytes_tx = pickle.dumps(msg_encoded, 0)
15
16 ip = input("Enter server IP: ")
17 port = input("Enter server port: ")
18 # Check if port is a number
19 while port.isnumeric() == False:
20     port = input("Port must be numeric: ")
21
22 port = int(port)
23
24 server_address = (ip, port)
25
26 try:
27
28     socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
29     socket.sendto(array_encoded, server_address)
30
31     bytes_rx = socket.recvfrom(1024)
32     print("RX: ", bytes_rx)
33     socket.close()
34
35 except:
36
37     print("Server not running on that address")

Input message: Hello, I'm the client 3
Enter server IP: 127.0.0.1
Enter server port: 6780
RX: (b'23 characters received', ('127.0.0.1', 6780))
```

El cliente empaqueta un array que contiene el mensaje y su Id y lo manda al servidor.

UDP-1.4

Código del servidor:

```
5 msg = "Hello, I'm the server"
6 bytes_tx = str.encode(msg)
7
8 server_ip = str(argv[1])
9 port = str(argv[2])
10
11 # Check if port is a number
12 while port.isnumeric() == False:
13     port = input("Port must be numeric: ")
14
15 port = int(port)
16
17 # If port is not a valid number: print("Invalid port")
18 try:
19     socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
20     socket.bind((server_ip, port))
21     print("Server up and listening!")
22
23     while True:
24         bytes_rx = socket.recvfrom(1024)
25         message = pickle.loads(bytes_rx[0])[0]
26         client_id = pickle.loads(bytes_rx[0])[1]
27         address = bytes_rx[1]
28         print("\nConnection received.")
29         print("Message received: ", str(message))
30         print("Number of characters of message", len(message))
31         print("Client Id: ", str(client_id))
32         print("Client address: ", str(address), "\n")
33         if str(message) == "exit":
34             print("Received exit instruction.")
35             print("Server shutting down")
36             bytes_tx = pickle.dumps("Received exit instruction, shutting down server", 0)
37             socket.sendto(bytes_tx, address)
38             break
39         else:
40             bytes_tx = pickle.dumps(str(len(message)) + " characters received", 0)
41             socket.sendto(bytes_tx, address)
42 except:
43     print("Invalid Port")
```

El código del servidor es el mismo pero pasado a VS para poder llamarlo más cómodamente por consola. En este caso, la IP y el puerto se obtienen de la línea de comandos.

Código del cliente:

```
import socket
import pickle
from sys import argv
import time

server_ip = argv[1]
port = str(argv[2])
client_id = argv[3]

print(server_ip)
print(port, type(port), len(port))
print(client_id)

id_encoded = pickle.dumps(str(client_id), 0)

# bytes_tx = pickle.dumps(msg_encoded, 0)

# Check if port is a number
while port.isnumeric() == False:
    port = input("Port must be numeric: ")

port = int(port)

server_address = (server_ip, port)

seconds = 10
# time.clock()

connection = True
elapsed = 0
socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)

while connection==True:
    connection = False
    msg = input("Input message: ")
    print("Waiting for server...")
    start = time.time()
    msg_encoded = pickle.dumps(msg, 0)
    array = [msg, str(client_id)]
    array_encoded = pickle.dumps(array, 0)
    while not connection:
        try:
            socket.sendto(array_encoded, server_address)
            bytes_rx = socket.recvfrom(1024)
            msg_received = pickle.loads(bytes_rx[0])
```

```

        connection = True
    except:
        connection = False
        time.sleep(1)
        elapsed = time.time() - start
        if elapsed > seconds:
            print("Server time out")
            break

if connection == True:
    try:
        print("Response: ", msg_received, "\n")
        # socket.close()
    except:
        # socket.close()
        print("Error")
if msg == "exit":
    socket.close()
    break

```

Si el cliente no recibe conexión con el servidor durante 10 segundos muestra un mensaje de “Time out”:

```

(env) C:\Users\corsi\OneDrive\Escritorio\ICAI\Sistemas Distribuidos\Pract1\UDP-1>python Client.py 127.0.0.1 6780 3
127.0.0.1
6780 <class 'str'> 4
3
Input message: Hello
Waiting for server...
Server time out

```

Esto permite iniciar el cliente antes que el servidor, pues se tiene una ventana de 10 segundos para hacerlo.

Si el servidor está corriendo correctamente:

```

Input message: Hello
Waiting for server...
Response:  5 characters received

Input message: Hello again
Waiting for server...
Response:  11 characters received

Input message: exit
Waiting for server...
Response:  Received exit instruction, shutting down server

```

El cliente puede mandar tantos mensajes como quiera. Si manda ‘exit’, hará que se cierre el servidor.

UDP-2:

UDP-2.1

Código del servidor:

```
In [*]: 1 import socket
2 import pickle
3 from sys import argv
4 import time
5 from datetime import date
6 import datetime
7
8
9 server_ip = "127.0.0.1"
10 port = "6780"
11
12 # Check if port is a number
13 while port.isnumeric() == False:
14     port = input("Port must be numeric: ")
15
16 port = int(port)
17
18 # If port is not a valid number: print("Invalid port")
19 try:
20     socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
21     socket.bind((server_ip, port))
22     print("Server up and listening!")
23
24     while True:
25         bytes_rx = socket.recvfrom(1024)
26         message = pickle.loads(bytes_rx[0])[0]
27         client_id = pickle.loads(bytes_rx[0])[1]
28         address = bytes_rx[1]
29         print("\nConnection received.")
30         print("Message received: ", str(message))
31         print("Number of characters of message", len(message))
32         print("Client Id: ", str(client_id))
33         print("Client address: ", str(address), "\n")
34         if str(message) == "exit":
35             print("Received exit instruction.")
36             print("Server shutting down")
37             bytes_tx = pickle.dumps("Received exit instruction, shutting down server", 0)
38             socket.sendto(bytes_tx, address)
39             break
40         else:
41             day = date.today()
42             now = datetime.datetime.now()
43             bytes_tx = pickle.dumps([day, now], 0)
44             socket.sendto(bytes_tx, address)
45     except Exception as e:
46         # print("Invalid Port")
47         print(e)
```

Server up and listening!

Si recibe cualquier instrucción que no sea 'exit', el servidor su hora exacta en el momento de la recepción.

Código del cliente:

UDP 2.1

```
In [3]: 1 import socket
2 import pickle
3 from sys import argv
4 import time
5
6 server_ip = "127.0.0.1"
7 port = "6780"
8 client_id = 3
9
10 print(server_ip)
11 print(port, type(port), len(port))
12 print(client_id)
13
14
15 id_encoded = pickle.dumps(str(client_id), 0)
16
17
18 # bytes_tx = pickle.dumps(msg_encoded, 0)
19
20 # Check if port is a number
21 while port.isnumeric() == False:
22     port = input("Port must be numeric: ")
23
24 port = int(port)
25
26 server_address = (server_ip, port)
27
28
29 seconds = 5
30
31 connection = True
32 elapsed = 0
33 socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
34
35 while connection == True:
36     connection = False
37     # msg = input("Input message: ")
38     msg = "Time request"
39     print("Waiting for server...")
40     start = time.time()
41     msg_encoded = pickle.dumps(msg, 0)
42     array = [msg, str(client_id)]
43     array_encoded = pickle.dumps(array, 0)
```

```

44     while not connection:
45         try:
46             socket.sendto(array_encoded, server_address)
47             bytes_rx = socket.recvfrom(1024)
48             day_received = pickle.loads(bytes_rx[0])[0]
49             time_received = pickle.loads(bytes_rx[0])[1]
50
51             connection = True
52         except:
53             connection = False
54             time.sleep(1)
55             elapsed = time.time() - start
56             print(elapsed)
57             if elapsed > seconds:
58                 print("Server time out")
59                 break
60
61     if connection == True:
62         if msg != "exit":
63
64             formatted_response = str(time_received)
65             try:
66                 print("Response: ", formatted_response, "\n")
67                 # socket.close()
68             except:
69                 # socket.close()
70                 print("Error")
71         else:
72             try:
73                 msg_received = pickle.loads(bytes_rx[0])
74                 print("Response: ", msg_received, "\n")
75             except:
76                 print("Error")
77         break
78
79     if msg == "exit":
80         socket.close()
81         break
82
83

```

El cliente manda una petición de hora al servidor y espera 5 segundos para recibirla.

```

Server Ip: 127.0.0.1
Port: 6780
Client ID: 3
Waiting for server...
Response: 2023-02-02 19:24:00.537258

```

UDP-2.2

Código del cliente:

UDP 2.2

```
In [1]: 1 import socket
2 import pickle
3 from sys import argv
4 import time
5 from datetime import date
6 import datetime
7
8 server_ip = "127.0.0.1"
9 port = "6780"
10 client_id = 3
11
12 print(server_ip)
13 print(port, type(port), len(port))
14 print(client_id)
15
16
17 id_encoded = pickle.dumps(str(client_id), 0)
18
19
20 # bytes_tx = pickle.dumps(msg_encoded, 0)
21
22 # Check if port is a number
23 while port.isnumeric() == False:
24     port = input("Port must be numeric: ")
25
26 port = int(port)
27
28 server_address = (server_ip, port)
29
30
31 seconds = 5
32
33 connection = True
34 elapsed = 0
35 socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
36
37 while connection == True:
38     client_day = date.today()
39     client_now = datetime.datetime.now()
40     connection = False
41     # msg = input("Input message: ")
42     msg = "Time request"
43     print("Waiting for server...")
44     start = time.time()
45     msg_encoded = pickle.dumps(msg, 0)
```

```

46 array = [msg, str(client_id)]
47 array_encoded = pickle.dumps(array, 0)
48 while not connection:
49     try:
50         socket.sendto(array_encoded, server_address)
51         bytes_rx = socket.recvfrom(1024)
52         day_received = pickle.loads(bytes_rx[0])[0]
53         time_received = pickle.loads(bytes_rx[0])[1]
54
55         connection = True
56     except:
57         connection = False
58         time.sleep(1)
59         elapsed = time.time() - start
60         print(elapsed)
61         if elapsed > seconds:
62             print("Server time out")
63             break
64
65 if connection == True:
66     if msg != "exit":
67
68         formatted_response = str(time_received)
69         print("Current client time: ", str(client_now))
70         diff = time_received - client_now
71         try:
72             print("Response: ", formatted_response)
73             print("Time difference: ", str(diff), "\n")
74             # socket.close()
75         except:
76             # socket.close()
77             print("Error")
78     else:
79         try:
80             msg_received = pickle.loads(bytes_rx[0])
81             print("Response: ", msg_received, "\n")
82         except:
83             print("Error")
84         break
85
86 if msg == "exit":
87     socket.close()
88     break
89

```

El cliente guarda su hora local antes de hacer la llamada y recibe la hora del servidor. Con esos dos tiempos, calcula la diferencia entre ellos para ver cuánto ha tardado la conexión.

```

Server Ip: 127.0.0.1
Port: 6780
Client ID: 3
Waiting for server...
Current client time: 2023-02-02 19:31:16.088921
Response: 2023-02-02 19:31:16.088921
Time difference: 0:00:00

```

Como se están ejecutando ambos en la misma máquina, el tiempo es demasiado pequeño como para medirlo.

UDP-2.3

Código del servidor:

```
In [ ]: 1 import socket
2 import pickle
3 from sys import argv
4 import time
5 from datetime import date
6 import datetime
7
8
9 server_ip = "127.0.0.1"
10 port = "6780"
11
12 # Check if port is a number
13 while port.isnumeric() == False:
14     port = input("Port must be numeric: ")
15
16 port = int(port)
17
18 # If port is not a valid number: print("Invalid port")
19 try:
20     socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
21     socket.bind((server_ip, port))
22     print("Server up and listening!")
23
24     while True:
25         bytes_rx = socket.recvfrom(1024)
26         message = pickle.loads(bytes_rx[0])[0]
27         client_id = pickle.loads(bytes_rx[0])[1]
28         address = bytes_rx[1]
29         # print("\nConnection received.")
30         # print("Message received: ", str(message))
31         # print("Number of characters of message", len(message))
32         # print("Client Id: ", str(client_id))
33         # print("Client address: ", str(address), "\n")
34         if str(message) == "exit":
35             print("Received exit instruction.")
36             print("Server shutting down")
37             bytes_tx = pickle.dumps("Received exit instruction, shutting down server", 0)
38             socket.sendto(bytes_tx, address)
39             break
40         else:
41             day = date.today()
42             now = datetime.datetime.now()
43             bytes_tx = pickle.dumps([day, now], 0)
44             socket.sendto(bytes_tx, address)
45     except Exception as e:
46         # print("Invalid Port")
47         print(e)
```

Server up and listening!

Se han comentado todos los prints para que no se pierdan recursos en mostrar las 100.000 recepciones.

Código del cliente:

UDP 2.3

```
In [24]: 1 import socket
2 import pickle
3 from sys import argv
4 import time
5 from datetime import date
6 import datetime
7
8 server_ip = "127.0.0.1"
9 port = "6780"
10 client_id = 3
11
12 counter = 0
13 iterations = int(input("Enter number of iterations: "))
14
15 id_encoded = pickle.dumps(str(client_id), 0)
16
17 # bytes_tx = pickle.dumps(msg_encoded, 0)
18
19 # Check if port is a number
20 while port.isnumeric() == False:
21     port = input("Port must be numeric: ")
22
23
24 port = int(port)
25
26 server_address = (server_ip, port)
27
28
29 seconds = 5
30
31 connection = True
32 elapsed = 0
33 socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
34 chrono_start = datetime.datetime.now()
35
36 acc_diff = datetime.timedelta(days = 0)
37
38 while connection == True and counter < iterations:
39     client_day = date.today()
40     client_now = datetime.datetime.now()
41     connection = False
42     # msg = input("Input message: ")
43     msg = "Time request"
44     # print("Waiting for server...")
45     start = time.time()
```

```

46 msg_encoded = pickle.dumps(msg, 0)
47 array = [msg, str(client_id)]
48 array_encoded = pickle.dumps(array, 0)
49 while not connection:
50     try:
51         socket.sendto(array_encoded, server_address)
52         bytes_rx = socket.recvfrom(1024)
53         day_received = pickle.loads(bytes_rx[0])[0]
54         time_received = pickle.loads(bytes_rx[0])[1]
55
56         connection = True
57     except:
58         connection = False
59         time.sleep(1)
60         elapsed = time.time() - start
61         print(elapsed)
62         if elapsed > seconds:
63             print("Server time out")
64             break
65
66 if connection == True:
67     if msg != "exit":
68
69         formatted_response = str(time_received)
70         # print("Current client time: ", str(client_now))
71         diff = time_received - client_now
72         acc_diff = acc_diff + diff
73     else:
74         try:
75             msg_received = pickle.loads(bytes_rx[0])
76             # print("Response: ", msg_received, "\n")
77         except:
78             print("Error")
79
80         counter = counter + 1
81
82 if msg == "exit":
83     socket.close()
84     break
85
86 chrono_finish = datetime.datetime.now()
87
88 time_spent = chrono_finish - chrono_start
89 print("Total difference ", acc_diff)
90 avg_diff = acc_diff/counter
91 print("Number of iterations: ", str(counter))
92 print("Total time spent: ", time_spent)
93 print("Average time per iteration: ", str(avg_diff))
94

```

El cliente recibe como entrada un número de iteraciones. Entonces, hará tantas llamadas al servidor como iteraciones. En cada llamada, calcula el tiempo que tarda el mensaje en enviarse al servidor y en devolverse al cliente. Al final, calcula el tiempo medio de todas las iteraciones.

```

Enter number of iterations: 100000
Total difference  0:00:03.677367
Number of iterations:  100000
Total time spent:  0:00:09.738644
Average time per iteration:  0:00:00.000037

```