Tema 4
Sockets en Python

Sistemas Distribuidos
Grado en Ingeniería Informática
Universidad Carlos III de Madrid

Contenido

- Menejo de direcciones
- Sockets TCP
- Sockets UDP
- Ejemplos

Direcciones IP

- El módulo `ipaddress` incluye clases para trabajar con direcciones de red IPv4 y IPv6
- Ejemplo:

`ip_address.py`

```
import binascii
import ipaddress

addr = ipaddress.ip_address('176.58.10.138')

print(addr)
print('  IP version:', addr.version)
print('  is private:', addr.is_private)
print('  packed form:', binascii.hexlify(addr.packed))
print('  integer:', int(addr))
print()
```

Sockets en Python

- El módulo socket incluye funciones para trabajar con direcciones y sockets:
  - Manejo de direcciones
  - Sockets Stream
  - Sockets Datagrama

Obtener el nombre de un host

gethostname.py

```
import socket


name = socket.gethostname();
print(name + ':   ' + socket.gethostbyname(name))
```

Conversión dominio-punto a decimal-punto

dns.py

```
import socket
import sys

arguments = len(sys.argv)
if arguments < 2:
    print('Uso: dns  <host>')
    exit()

try:
    hostname, aliases, addresses =
                        socket.gethostbyaddr(sys.argv[1]);
    print(sys.argv[1] + ':  ',  hostname)
    print(sys.argv[1] + ':  ',  aliases)
    print(sys.argv[1] + ':  ',  addresses)

except socket.error as msg:
    print('ERROR: ', msg)
```

Conversión decimal-punto a decimal-punto

dns.py

```
import socket
import sys

arguments = len(sys.argv)
if arguments < 2:
    print('Uso: dns  <host>')
    exit()

try:
    hostname, aliases, addresses =
                        socket.gethostbyaddr(sys.argv[1]);
    print(sys.argv[1] + ':  ',  hostname)
    print(sys.argv[1] + ':  ',  aliases)
    print(sys.argv[1] + ':  ',  addresses)
except socket.error as msg:
    print('ERROR: ', msg)
```

Funciones para trabajar con sockets TCP

- `socket.socket:` crea un socket
- `socket.connect`: establece una conexión con un servidor
- `socket.accept`: acepta una conexión
- `socket.bind`: asocia una dirección a un socket
- `socket.listen`: prepara para aceptar conexiones
- `socket.close`: cierra la conexión
- `socket.recv(len):` lee como mucho `len` bytes de un socket. Devuelve el número de bytes leídos
- `socket.send`(len): envía datos por el socket. Devuelve el número de bytes enviados.
- `socket.sendall(len)`: envía todos los datos (`len`) por el socket, haciendo todos los reintentos posibles. Asegura que se envían todos los bytes siempre que no hay error

Funciones para trabajar con sockets UDP

- `socket.recvfrom:` recibe datos de un socket UDP
- `socket.sendtpo:` envía datos por un socket UDP

Sockets TCP. Servidor Eco

server_echo.py

```
import socket
import sys

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server_address = ('localhost', 10000)
sock.bind(server_address)

sock.listen(1)
```

Sockets TCP. Servidor Eco

server_echo.py

```
while True:
    print('waiting for a connection')
    connection, client_address = sock.accept()
    try:
        print('connection from', client_address)

        while True:
            data = connection.recv(16)
            print('received {!r}'.format(data))
            if data:
                print('sending data back to the client')
                connection.sendall(data)
            else:
                print('no data from', client_address)
                break

    finally:
       connection.close()
```

Sockets TCP. Cliente Eco

client_echo.py

```
import socket
import sys

arguments = len(sys.argv)
if arguments < 3:
    print('Uso: client_echo  <host> <port>')
    exit()

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server_address = (sys.argv[1], int(sys.argv[2]))
print('connecting to {} port {}'.format(*server_address))
sock.connect(server_address)
```

Sockets TCP. Cliente Eco

client_echo.py

```
try:

    message = b'Esto es una cadena \0'
    sock.sendall(message)

    message = ''
    while True:
        msg = sock.recv(1)
        if (msg == b'\0'):
            break;
        message += msg.decode()

    print('Recibido= ' + message)

finally:
    print('closing socket')
    sock.close()
```

Sockets TCP. Cliente Eco

client_echo.py

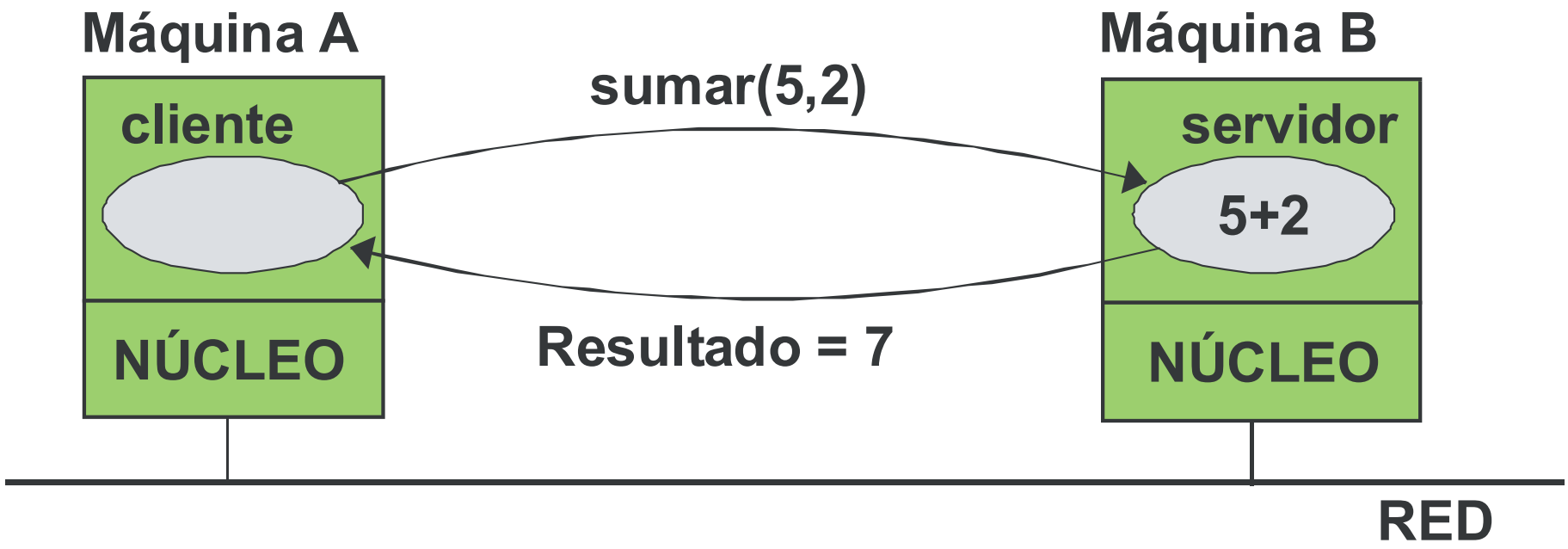
```
try:

    message = b'Esto es una cadena \0'
    sock.sendall(message)

    message = ''
    while True:
        msg = sock.recv(1)
        if (msg == b'\0'):
            break;
        message += msg.decode()

    print('Recibido= ' + message)

finally:
    print('closing socket')
    sock.close()
```

Fin de cadena

secuencia de bytes

decodifica la secuencia de bytes a un string

Ejemplo: servidor y cliente usando sockets *stream* en Python



Servidor TCP

server_calc.py

```
import socket
import sys

def readNumber(sock):
    a = ''
    while True:
        msg = sock.recv(1)
        if (msg == b'\0'):
            break;
        a += msg.decode()

    return(int(a,10))

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

server_address = ('localhost', 10009)
sock.bind(server_address)

sock.listen(5)
```

convierte a la secuencia de bytes a un string

Servidor TCP

server_calc.py

```
while True:
    print('waiting for a connection')
    connection, client_address = sock.accept()
    try:
        print('connection from', client_address)

        a = readNumber(connection)
        b = readNumber(connection)
        op = readNumber(connection)

        if (op == 0):
            res = a + b
        else:
            res = a - b

        message = str(res) + "\0"
        message = message + "\0"
        connection.sendall(message.encode())

    finally:
        # Clean up the connection
        connection.close()
```

convierte a una secuencia de bytes

Cliente TCP

client_calc.py

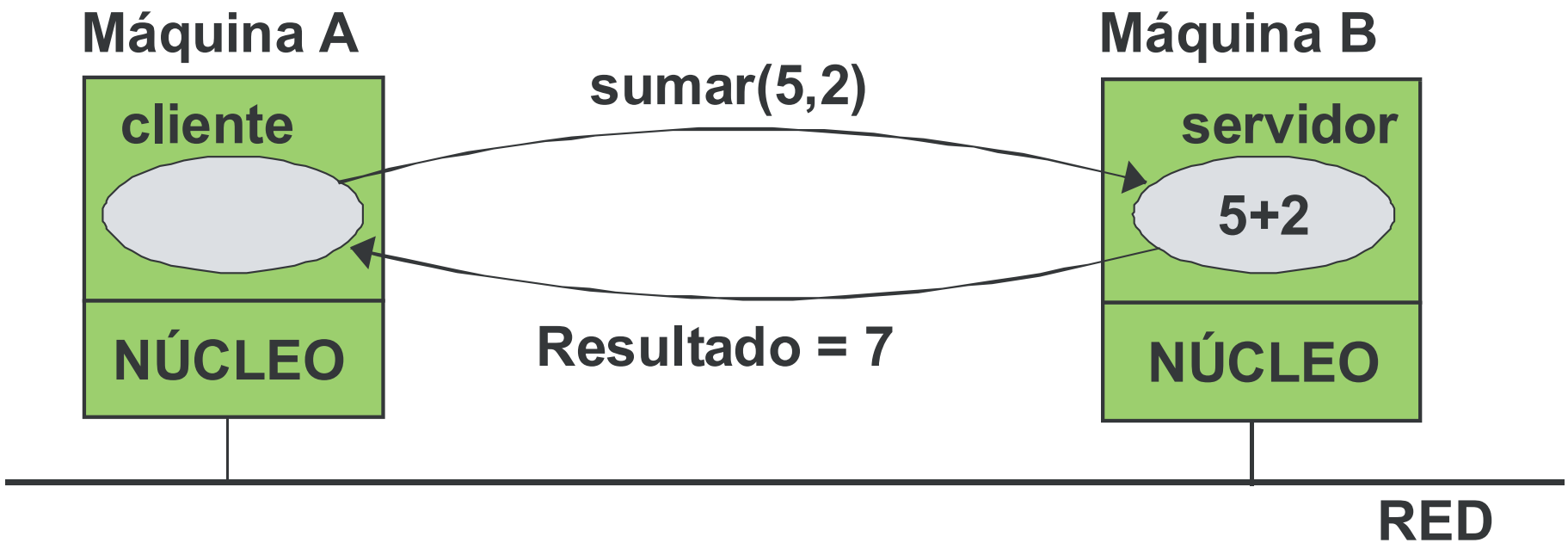
```
import socket
import sys

def readNumber(sock):
    a = ''
    while True:
        msg = sock.recv(1)
        if (msg == b'\0'):
            break;
        a += msg.decode()

    return(int(a,10))

arguments = len(sys.argv)
if arguments < 3:
    print('Uso: client_calc  <host> <port>')
    exit()

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Cliente TCP

client_calc.py

```
server_address = (sys.argv[1], int(sys.argv[2]))
print('connecting to {} port {}'.format(*server_address))
sock.connect(server_address)

try:
    a = 5
    b = 8
    op = 1
    sock.sendall(str(a).encode())
    sock.sendall(b'\0')
    sock.sendall(str(b).encode())
    sock.sendall(b'\0')
    sock.sendall(str(op).encode())
    sock.sendall(b'\0')

    res = readNumber(sock)
    print(res)
finally:
    print('closing socket')
    sock.close()
```

Ejemplo: servidor y cliente usando sockets *UDP* en Python



Servidor UDP

server_calc.py

```
import socket
import sys
import struct

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

server_address = ('localhost', 10009)
sock.bind(server_address)
```

Servidor UDP

server_calc.py

```
while True:
        data, addr = sock.recvfrom(1024)
        print(data, addr)

        a,b,op = struct.unpack("III", data)

        print(a)
        print(b)
        print(op)

        if op==0:
            res = a + b
        else:
            res = a - b

        data = struct.pack("I", res)
        sock.sendto(data, addr)
```

Cliente UDP

client_calc.py

```
import socket
import struct
import sys

arguments = len(sys.argv)
if arguments < 3:
    print('Uso: client_calc  <host> <port>')
    exit()

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

try:
    serverAddress = (sys.argv[1], int(sys.argv[2]))
```

Cliente UDP

client_calc.py

```
    a = 10
    b = 500
    op = 0

    data = struct.pack('III', a, b, op)

    sock.sendto(data, serverAddress)
    message, addr = sock.recvfrom(1024)

    res = struct.unpack("I", message)
    print(res)


finally:
    print('closing socket')
    sock.close()
```

Servidor TCP concurrente

server_calc_conc.py

```
import threading
import socket

def readNumber(sock):
    a = ''
    while True:
        msg = sock.recv(1)
        if (msg == b'\0'):
            break;
        a += msg.decode()

    return(int(a,10))
```

Servidor TCP concurrente

server_calc_conc.py

```
def worker(sock):
    try:
        a = readNumber(sock)
        b = readNumber(sock)
        op = readNumber(sock)


        if (op == 0):
            res = a + b
        else:
            res = a - b

        message = str(res) + "\0"
        message = message + "\0"
        sock.sendall(message.encode())
    finally:
        # Clean up the connection
        sock.close()
```

Servidor TCP concurrente

server_calc_conc.py

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

server_address = ('localhost', 10009)
sock.bind(server_address)

sock.listen(5)

while True:
    print('waiting for a connection')
    connection, client_address = sock.accept()
    print('connection from', client_address)

    t = threading.Thread(target=worker, name='Daemon',
                                    args=(connection,))
    t.start()
```

Más funciones para trabajar con sockets

- `socket.setsockopt`: actualiza las opciones sobre un socket
- `socket. getsockopt`: recupera el valor asociado a las opciones de un socket
- socket.ntohl(*x*)
- socket.ntohs(*x*)
- socket.htonl(*x*)
- socket.htons(*x*)
  - Equivalente a las funciones vistas para C