5008CEM Programming For Developers

Concurrent and Asynchronous Applications 1

Beate Grawemeyer | John Halloran

Intended Learning

- Understand the multithreading concept
- Be able to explain applications of multithreading
- Understand and manage data sharing in multithreaded environments
- Understand implementation of threading in Python
  - with reference to the key libraries

What is multithreading?

- Many of the applications you've written so far do one thing after another
- Multithreading allows us to run an application as separate processes which run 'asynchronously'
  - i.e., the timing is arbitrary
- This results in processes which run at the same or similar / overlapping times

Applications of multithreading

- We'll now look at two applications of multithreading:
  1. Chat server
  2. Newspaper size getter

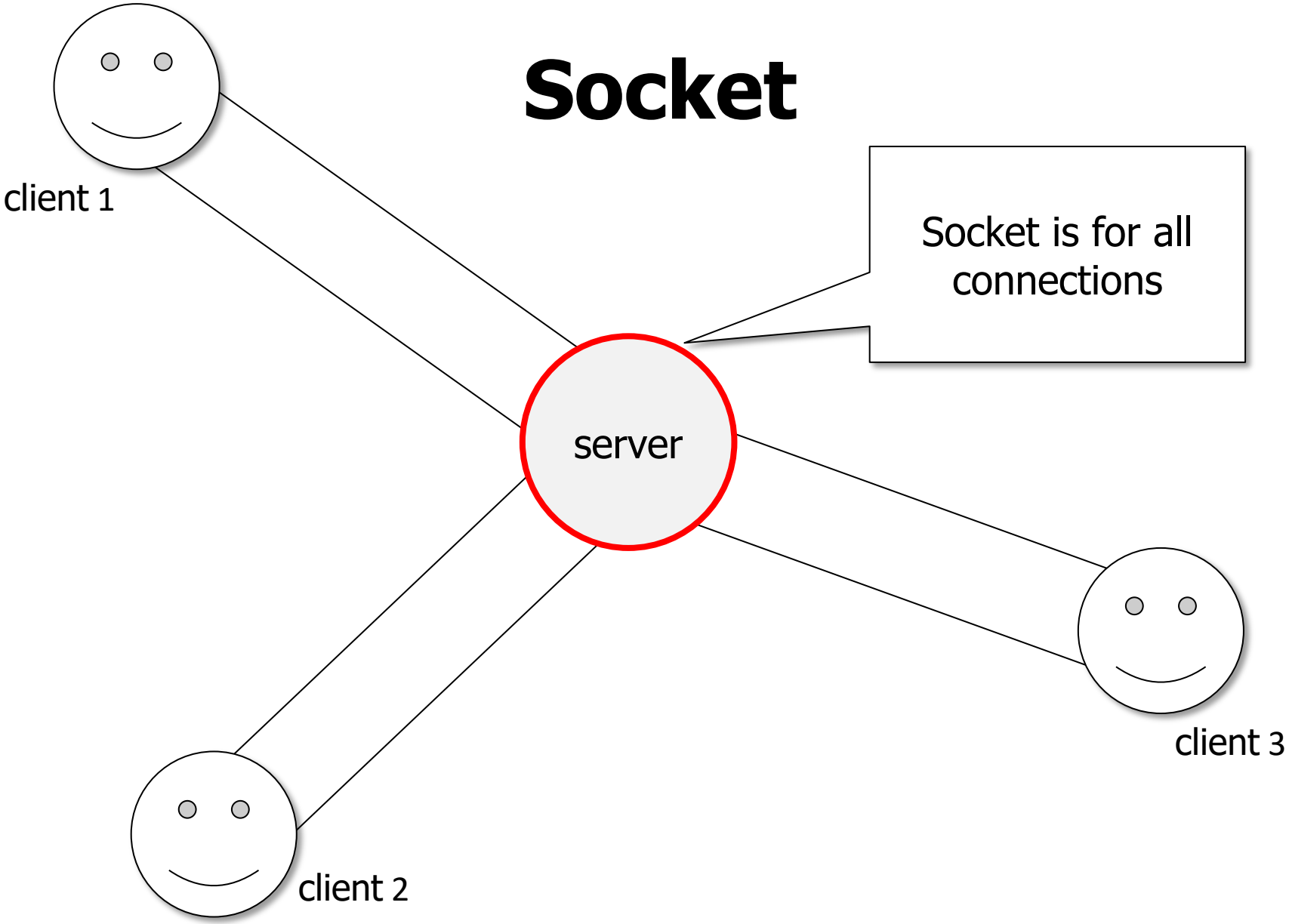
Code

- Check out the instructions in the document 5008CEM_Concurrent_1_Lecture-Instructions.doc (on Aula)
- Get the code from Aula – CONCURRENT-1-CODE, and add it to your Codio project

Concurrent Example 1: CHAT SERVER

Chat Server

- The first example of a threaded environment is a chat server
- Open server.py and follow the instructions at the top
- See what happens


Socket
client 1
Socket is for all connections
server
client 3
client 2

server.py

```
import socket
import threading

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

sock.bind(('127.0.0.1', 12345))

sock.listen(1)

connections = []

def handler (c, a):
    while True:
        data = c.recv(4098)
        for connection in connections:
            if connection != c:
                  connection.send(data)
        if not data:
              break

while True:
    c, a = sock.accept()
    cThread = threading.Thread(target=handler, args=(c, a))
    cThread.daemon = True
    cThread.start()
    connections.append(c)
    print(connections)
```

import libraries

create socket

bind socket to port (we're using localhost)

Listen for connections made to the socket

Create and initialise a list for connections

Receive data from connections and broadcast data to all connections other than the sender.

create and start a thread for the connection; handle traffic for the connection; append the new connection; print connections

Note that the thread calls the handler function above

client.py

```
import socket
import threading

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

sock.connect(('127.0.0.1', 12345))

def sendMsg():
    while True:
        msg = input().encode()
        sock.send(msg)

iThread = threading.Thread(target=sendMsg)
iThread.daemon = True
iThread.start()

while True:
    data = sock.recv(4098).decode()
    if not data:
        break
    print(data)
```

import libraries

create socket

connect (we're using localhost)

function to send messages

create and start a thread for the connection; send the message

receive messages

Threaded Connections