TP3 : Programmation Réseau Année : 2022/2023

1. Tester les programmes donnés, et comprendre ce qu’ils effectuent
2. Modifier les deux premiers programmes pour exécuter trois threads simultanément, et se synchroniser sur leurs terminaisons.
3. Reprendre les programmes Serveurs des TP1 ou TP2, et écrire des programmes de serveurs multithreads.

from concurrent.futures import ThreadPoolExecutor

from concurrent.futures import as\_completed

values = [2,3,4,5]

def square(n):

 return n \* n

def main():

 with ThreadPoolExecutor(max\_workers = 3) as executor:

 results = executor.map(square, values)

for result in results:

 print(result)

if \_\_name\_\_ == '\_\_main\_\_':

 main()

from concurrent.futures import ThreadPoolExecutor

from time import sleep

def task(message):

 sleep(2)

 return message

def main():

 executor = ThreadPoolExecutor(5)

 future = executor.submit(task, ("Completed"))

 print(future.done())

 sleep(2)

 print(future.done())

 print(future.result())

if \_\_name\_\_ == '\_\_main\_\_':

 main()

from time import sleep

from threading import Thread

# a custom function that blocks for a moment

def task(sleep\_time, message):

    # block for a moment

    sleep(sleep\_time)

    # display a message

    print(message)

 # create a thread

thread = Thread(target=task, args=(1.5, 'New message from another thread'))

# run the thread

thread.start()

# wait for the thread to finish

print('Waiting for the thread...')

thread.join()

from time import sleep

from threading import Thread

# custom thread class

class CustomThread(Thread):

    # override the run function

    def run(self):

        # block for a moment

        sleep(1)

        # display a message

        print('This is coming from another thread')

 # store return value

        self.value = 99

# create the thread

thread = CustomThread()

# start the thread

thread.start()

# wait for the thread to finish

print('Waiting for the thread to finish')

thread.join()

# get the value returned from run

value = thread.value

print(f'Got: {value}')

TP4 : Programmation Réseau Année : 2022/2023

1. Compléter les deux bouts de programmes ci-dessous.
2. Tester ces deux programmes
3. Modifier le programme qui convient en utilisant l’extension de la classe thread.

import socket

IP = socket.gethostbyname(socket.gethostname())

PORT = 5656

ADDR = (IP, PORT)

FORMAT = "utf-8"

SIZE = 1024

DISCONNECT\_MSG = "Bye"

def main():

 client\_soc = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

 client\_soc.connect(ADDR)

 print(f"[CONNECTED] Client connected to server at {IP} {PORT}")

 connected = True

 while connected:

 msg = input("> ")

 client\_soc.send(msg.encode(FORMAT))

 if msg == DISCONNECT\_MSG:

 connected = False

 else :

 msg = client\_soc.recv(SIZE).decode(FORMAT)

 print(f"[SERVER] {msg}")

 client\_soc.close()

if \_\_name\_\_ == "\_\_main\_\_":

 main()

import socket

import threading

IP = socket.gethostbyname(socket.gethostname())

PORT = 5656

ADDR = (IP, PORT)

FORMAT = "utf-8"

SIZE = 1024

DISCONNECT\_MSG = "Bye"

def handle\_client(conn, addr):

 print(f"[NEW CONNECTION] {addr} Connected ...")

 connected = True

 while connected:

 msg = conn.recv(SIZE).decode(FORMAT)

 if msg == DISCONNECT\_MSG:

 connected = False

 print(f"{[addr]} {msg}")

 msg = f"Msg Received : {msg}"

 conn.send(msg.encode(FORMAT))

 conn.close()

def main():

 print("[STARTING] Server is starting...")

 server\_soc = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

 server\_soc.bind(ADDR)

 server\_soc.listen()

 print(f"[LISTENING] Server is listening on {IP} : {PORT}")

 while True:

 conn, addr = server\_soc.accept()

 cl\_thread = threading.Thread(target=handle\_client, args=(conn, addr))

 cl\_thread.start()

 print(f"[ACTIVE CONNECTIONS] {threading.activeCount() - 1}")

if \_\_name\_\_ == "\_\_main\_\_":

 main()