Escribir una aplicación cliente servidor que muestre las direcciones y puertos de todos los clientes conectados del lado del servidor y devuelva a cada cliente el día y hora de conexión, y el tiempo que estuvo (o está conectado).
Servidor base

```python
import socket
import threading
import time

# Variable para rastrear el número de clientes activos
active_clients = 0
active_clients_lock = threading.Lock()

# Función para manejar conexiones de clientes
def funCliente(client_socket, client_address):
    global active_clients

    print(f"[INFO] Conexión establecida desde {client_address}")

    # Incrementar el contador de clientes activos
    with active_clients_lock:
        active_clients += 1

    # Obtener la fecha y hora de conexión
    connection_time = time.time()

    # Enviar fecha y hora de conexión al cliente
    client_socket.send(f"Bienvenido! Conexión establecida el {time.ctime(connection_time)}\n".encode())

    while True:
        # Recibir datos del cliente
        request = client_socket.recv(1024)
        if not request:
            break

    # Calcular tiempo de conexión
    connected_time = time.time() - connection_time
    client_socket.send(f"Te has desconectado. Tiempo conectado: {connected_time} segundos\n".encode())

    # Cerrar conexión con el cliente
    client_socket.close()
    print(f"[INFO] Conexión cerrada con {client_address}")
```

```python
    # Decrementar el contador de clientes activos
    with active_clients_lock:
        active_clients -= 1

# Función para manejar las conexiones entrantes
def servidorOn():
    # Configurar el servidor
    server_host = "127.0.0.1"
    server_port = 6667

    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((server_host, server_port))
    server.listen(5)

    print(f"[INFO] Servidor escuchando en {server_host}:{server_port}")

    while True:
        # Aceptar conexiones entrantes
        client_socket, client_address = server.accept()

        # Iniciar un hilo para manejar la conexión con el cliente
        client_thread = threading.Thread(target=funCliente, args=(client_socket, client_address))
        client_thread.start()

# Función para verificar si todos los clientes están cerrados
def ver_clientes():
    global active_clients

    while True:
        time.sleep(1)  # Esperar un segundo antes de verificar
        with active_clients_lock:
            if active_clients == 0:
                print("[INFO] Todos los clientes están cerrados. Cerrando el servidor.")
                break
```

```python
# Iniciar el servidor en un hilo
server_thread = threading.Thread(target=servidorOn)
server_thread.start()

# Iniciar un hilo para verificar si todos los clientes están cerrados
check_clients_thread = threading.Thread(target=ver_clientes)
check_clients_thread.start()

# Esperar a que el servidor y el hilo de verificación terminen
server_thread.join()
check_clients_thread.join()
```

Cliente base

```python
import socket
import sys

def main():
    # Configurar el cliente
    server_host = 'localhost'
    server_port = 6667

    # Conectar al servidor
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((server_host, server_port))

    # Recibir mensaje de bienvenida del servidor
    print(client.recv(1024).decode())

    # Cerrar conexión con el servidor
    client.close()

main()
```

Terminal

```
juanp@K21:~/proyectos/protocolos$ python3 servidor.py
[INFO] Servidor escuchando en 127.0.0.1:6667
[INFO] Todos los clientes están cerrados. Cerrando el servidor.
[INFO] Conexión establecida desde ('127.0.0.1', 47916)
[INFO] Conexión cerrada con ('127.0.0.1', 47916)
```

```
juanp@K21:~/proyectos/protocolos$ python3 cliente.py
Bienvenido! Conexión establecida el Mon May 13 16:16:37 2024

juanp@K21:~/proyectos/protocolos$
```

En el cliente se realiza la configuración y se establece la conexión con el servidor. Primero definimos la dirección del servidor y el puerto al cual se conecta, luego creamos el socket el cual después se conecta con el servidor. Una vez que se conecta exitosamente el cliente recibe el mensaje de confirmación de parte del servidor en el cual se encuentra los datos del tiempo y día de conexión, una vez que printea el mensaje cierra la conexión.
Por el lado del servidor, una vez definido la dirección IP y su puerto para poder escuchar las conexiones entrantes, creamos un socket el cual se enlaza a la dirección y al puerto identificado, escucha las conexiones y una vez que un cliente se conecta, realizamos el manejo del mensaje mediante **funCliente()**. cuya función es preparar el mensaje que enviará al cliente, en donde contiene el tiempo de conexión y su fecha, para despues cerrar la conexion.La función **ver_clientes()** sirve para poder verificar si el servidor tiene algún cliente conectado en el caso de que no sea el caso se detiene la ejecución.

Realizar la misma aplicación tanto para C-S con Concurrencia Aparente (Select) como C-S Concurrente.
De ser necesario agregar tiempo de espera, loop, sleep, con contadores para demorar los procesos.
Implementar una limpieza de recursos al salir de los programas (agregar opción de pregunta al usuario para cerrar los clientes).

C-S Concurrencia Aparente (Select)
Servidor

```python
import socket
import select
import time

# Lista de sockets de clientes y sus direcciones
client_sockets = {}
client_addresses = {}

# Diccionario para almacenar los datos de los clientes
client_data = {}

# Función para manejar conexiones de clientes
def funCliente(client_socket, client_address):
    print(f"[INFO] Conexión establecida desde {client_address}")

    # Obtener la fecha y hora de conexión
    connection_time = time.time()

    # Enviar fecha y hora de conexión al cliente
    client_socket.send(f"Bienvenido! Conexión establecida el {time.ctime(connection_time)}\n".encode())

    # Inicializar los datos del cliente
    client_data[client_socket] = {
        "connection_time": connection_time,
        "messages_received": 0
    }

    while True:
        # Recibir datos del cliente
        request = client_socket.recv(1024)
        if not request:
            break
        elif request.decode().lower() == 'exit':
            break
```

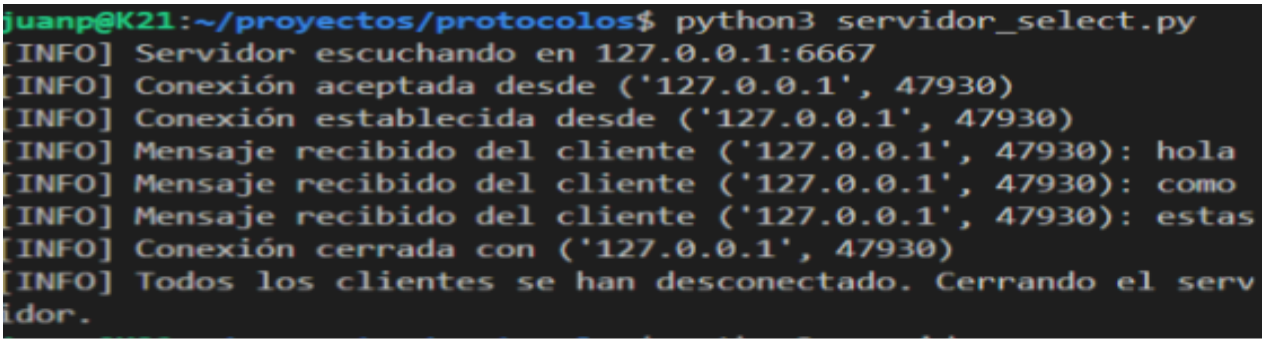
```python
            # Contar los mensajes recibidos
            client_data[client_socket]["messages_received"] += 1
            print(f"[INFO] Mensaje recibido del cliente {client_address}: {request.decode()}")

            # Responder al cliente
            response = f"Mensaje recibido ({client_data[client_socket]['messages_received']}).\n"
            client_socket.send(response.encode())

    # Calcular tiempo de conexión
    connected_time = time.time() - client_data[client_socket]["connection_time"]
    client_socket.send(f"Te has desconectado. Tiempo conectado: {connected_time} segundos\n".encode())

    # Cerrar conexión con el cliente
    print(f"[INFO] Conexión cerrada con {client_address}")
    client_socket.close()
    del client_sockets[client_socket]
    del client_addresses[client_socket]
    del client_data[client_socket]
```

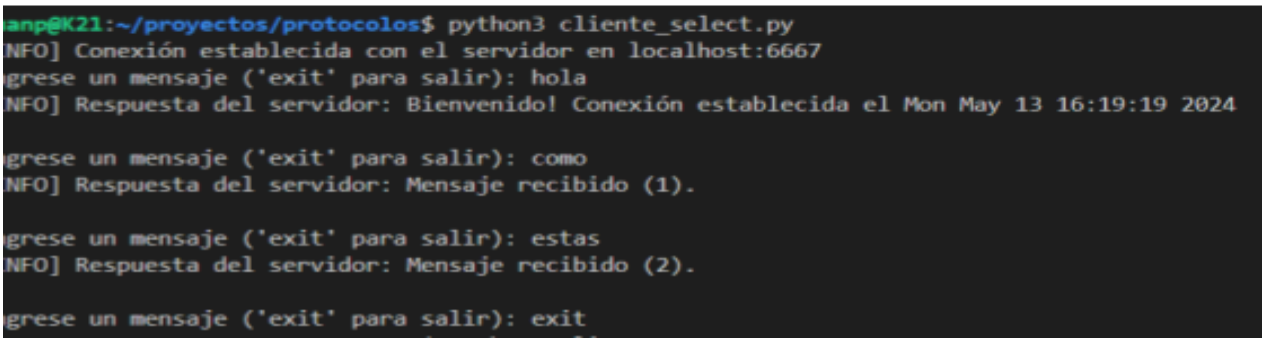
```python
# Función principal del servidor
def serverOn():
    # Configurar el servidor
    server_host = "127.0.0.1"
    server_port = 6667

    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server.bind((server_host, server_port))
    server.listen(5)

    print(f"[INFO] Servidor escuchando en {server_host}:{server_port}")

    # Agregar el socket del servidor a la lista de sockets a monitorear
    client_sockets[server] = None

    while True:
        # Utilizar select para manejar múltiples conexiones de clientes
        readable, _, _ = select.select(list(client_sockets.keys()), [], [])

        for sock in readable:
            if sock == server:
                # Acepta la conexión entrante
                client_socket, client_address = server.accept()
                client_sockets[client_socket] = client_address
                client_addresses[client_socket] = client_address
                print(f"[INFO] Conexión aceptada desde {client_address}")
            else:
                # Maneja la conexión del cliente
                funCliente(sock, client_sockets[sock])

        # Cerrar el servidor si no hay clientes conectados
        if len(client_sockets) == 1:  # Solo queda el socket del servidor
            print("[INFO] Todos los clientes se han desconectado. Cerrando el servidor.")
            server.close()
            break
# Función principal
serverOn()
```

Cliente

```python
import socket
import select
import time

# Función principal del cliente
def funServer():
    # Configurar el cliente
    server_host = 'localhost'
    server_port = 6667

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((server_host, server_port))
    print(f"[INFO] Conexión establecida con el servidor en {server_host}:{server_port}")

    try:
        while True:
            # Esperar la entrada del usuario
            message = input("Ingrese un mensaje ('exit' para salir): ")

            # Enviar el mensaje al servidor
            client_socket.sendall(message.encode())

            # Salir si el usuario escribe "exit"
            if message.lower() == "exit":
                break

            # Esperar la respuesta del servidor solo si estamos listos para recibir
            readable, _, _ = select.select([client_socket], [], [], 1)
            if client_socket in readable:
                response = client_socket.recv(1024)
                print("[INFO] Respuesta del servidor:", response.decode())
            else:
                print("[INFO] Esperando respuesta del servidor...")

    except KeyboardInterrupt:
        print("[INFO] Cliente cerrado.")
        client_socket.close()
# Función principal
funServer()
```

Terminal





En este caso el servidor nuevamente crea un socket TCP y lo enlaza a una dirección IP y puerto,para escuchar las conexiones entrantes. No obstante, a diferencia del servidor básico, este utiliza **select** para manejar múltiples conexiones de clientes de manera más eficiente. Si hay una conexión entrante la acepta y agrega el nuevo socket del cliente a una lista para ser monitoreados.En la función **fun Cliente()** ejecuta y maneja cada conexión de cliente en un hilo separado y envía un mensaje al cliente, conteniendo los datos pedidos, en esta caso el cliente es capaz de enviar más mensajes pero no envía de manera constante los datos de tiempo, sino que es capaz de recibir los mensajes y una vez que recibe un mensaje exit el servidor busca el cliente que realiza la petición y cierra la conexión.
El cliente a diferencia de su versión base utiliza select para esperar la respuesta del servidor de manera más eficiente. Cuando recibe la respuesta la imprime y vuelve a enviar un nuevo mensaje.

C-S concurrente

Servidor

```python
import socket
import threading
import time

# Lista de threads de clientes
client_threads = []

# Función para manejar conexiones de clientes
def funCliente(client_socket, client_address):
    print(f"[INFO] Conexión establecida desde {client_address}")

    # Obtener la fecha y hora de conexión
    connection_time = time.time()

    # Enviar fecha y hora de conexión al cliente
    client_socket.send(f"Bienvenido! Conexión establecida el {time.ctime(connection_time)}\n".encode())

    while True:
        # Recibir datos del cliente
        request = client_socket.recv(1024)
        if not request:
            break

    # Calcular tiempo de conexión
    connected_time = time.time() - connection_time
    client_socket.send(f"Te has desconectado. Tiempo conectado: {connected_time} segundos\n".encode())

    # Cerrar conexión con el cliente
    client_socket.close()
    print(f"[INFO] Conexión cerrada con {client_address}")
```

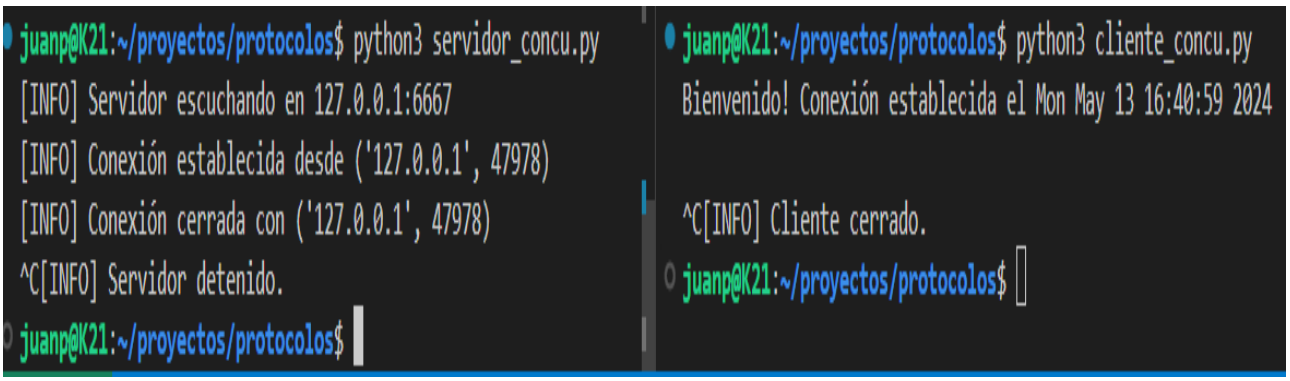
```python
# Función principal del servidor
def servidorOn():
    # Configurar el servidor
    server_host = "127.0.0.1"
    server_port = 6667

    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((server_host, server_port))
    server.listen(5)

    print(f"[INFO] Servidor escuchando en {server_host}:{server_port}")

    while True:
        # Aceptar conexiones entrantes
        client_socket, client_address = server.accept()

        # Iniciar un hilo para manejar la conexión con el cliente
        client_thread = threading.Thread(target=funCliente, args=(client_socket, client_address))
        client_thread.start()
        client_threads.append(client_thread)

# Función para limpiar recursos
def cleanup():
    for thread in client_threads:
        thread.join()


try:
    servidorOn()
except KeyboardInterrupt:
    print("[INFO] Servidor detenido.")
    cleanup()
```

Cliente

```python
import socket
import threading
import time

# Función para enviar mensajes al servidor
def send_message(client_socket, message):
    client_socket.send(message.encode())

# Función para recibir mensajes del servidor
def receive_message(client_socket):
    return client_socket.recv(1024).decode()

# Función para manejar la interacción con el servidor
def funServer(client_socket):
    try:
        while True:
            # Enviar solicitud al servidor
            send_message(client_socket, "Datos del cliente")
            # Recibir respuesta del servidor
            response = receive_message(client_socket)
            print(response)

            # Esperar un tiempo antes de enviar la siguiente solicitud
            time.sleep(3)
    except Exception as e:
        print(f"Error: {e}")
    finally:
        client_socket.close()
```

```python
# Función principal del cliente
def main():
    # Configurar el cliente
    server_host = 'localhost'
    server_port = 6667

    try:
        # Conectar al servidor
        client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client_socket.connect((server_host, server_port))

        # Recibir mensaje de bienvenida del servidor
        print(receive_message(client_socket))

        # Iniciar un hilo para interactuar con el servidor
        interaction_thread = threading.Thread(target=funServer, args=(client_socket,))
        interaction_thread.start()

        # Esperar a que el hilo de interacción termine
        interaction_thread.join()

    except KeyboardInterrupt:
        print("[INFO] Cliente cerrado.")
    except Exception as e:
        print(f"Error: {e}")
    finally:
        client_socket.close()

main()
```

Terminal



El cliente envía solicitudes al servidor periódicamente, mientras que el servidor maneja múltiples conexiones de clientes de forma concurrente, enviando mensajes de bienvenida y calculando el tiempo de conexión para cada cliente. Ahora a diferencia del anterior este código no utiliza select si no las múltiples conexiones se manejan usando hilos separados, cada hilo representa una conexión con un cliente. También se le agregó una función **cleanup** que sirve para limpiar los recursos al final de la ejecución del servidor.