



Seminario 4

Creación de aplicaciones Cliente/Servidor

1. Objetivo

El objetivo de este seminario es introducir al alumno en los conceptos básicos para el desarrollo de aplicaciones cliente-servidor que utilizan sockets UDP y sockets TCP.

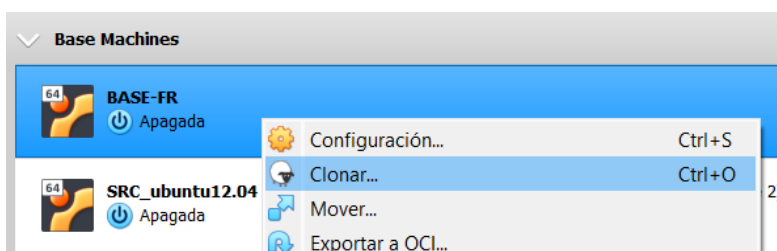
1.1. Información básica para la realización del seminario

En esta sección se ofrece la información básica y las referencias necesarias para llevar a cabo las tareas que se proponen en la práctica.

1.2. Acceso al sistema y elección del sistema operativo

Para la realización de este seminario no es necesario el uso de ningún sistema operativo en particular, ya que se realizará el lenguaje de programación *Python*, que es un lenguaje multiplataforma. No obstante, recomendamos la instalación del editor de código fuente *Visual Studio Code* [1] y la extensión de *Python*.

También es posible la utilización de la máquina virtual de seminarios de la asignatura de Fundamentos de Redes disponible en PRADO. Para ello, es necesario realizar una clonación de dicha máquina virtual. Para esto, se hace clic derecho sobre la máquina ya importada en Virtualbox y se escoge la opción Clonar.



Se puede cambiar el nuevo nombre de la máquina por FR-cliente/servidor, por ejemplo. Es muy importante que en este punto se seleccione la opción generar nuevas direcciones MAC para todos los adaptadores de red. Elegido esto se clicca sobre Siguiente.

Nuevo nombre de máquina y ruta

Seleccione un nombre y opcionalmente una carpeta para la nueva máquina virtual. La nueva máquina será un clon de la máquina **BASE-FR**.

Nombre:

Ruta:

Política de dirección MAC:

Opciones adicionales: ☐ Mantener nombres de disco

☐ Mantener UUIDs hardware



Por último, se escoge la opción clonación enlazada y se clicla sobre Clonar.

Tipo de clonación

Seleccione el tipo de clonación que desea crear.

Si selecciona **Clonación completa**, una copia exacta (incluyendo todos los archivos de disco duro virtual) de la máquina original serán creados.

Si selecciona **Clonación enlazada**, una nueva máquina será creada, pero los archivos de las unidades de disco duro virtuales serán vinculados a los archivos de disco duro virtual de la máquina original y no podrá mover la nueva máquina virtual a una computadora diferente sin mover los originales también.

Si crea una **Clonación enlazada** entonces una nueva instantánea será creada en la máquina virtual original como parte del proceso de clonación.

☐ Clonación completa

☒ Clonación enlazada

Clonar

Cancelar

Nota importante: Realizad la clonación clicando sobre el botón derecho desde la máquina original importada. Esa máquina será ahora la máquina base de esta clonación y no debe encenderse para minimizar los conflictos que esto pudiera generar. Sólo se utilizará la máquina clonada.

Los puertos sirven para identificar aplicaciones (procesos)

2. Códigos del cliente y servidor en UDP

Prepare los códigos tanto de este apartado como del siguiente para poder seguir la clase de seminarios con mayor fluidez. Los conceptos teóricos asociados serán repasados en la clase de seminarios.

Cread el archivo clienteUDP.py que contenga el siguiente código:

```
import socket
s_client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s_client.sendto(b'Hola clase', ('localhost', 12345))
s_client.close()
```

Handwritten notes:
 - *Address family* (pointing to AF_INET)
 - *Envío de datagrama* (pointing to sendto)
 - *convierte string a array de bytes* (pointing to b'Hola clase')
 - *Nombre* (pointing to localhost)
 - *Puerto* (pointing to 12345)

Cread el archivo servidorUDP.py que contenga el siguiente código:

```
import socket
s_server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s_server.bind(('', 12345))
data, clientaddr = s_server.recvfrom(4096)
s_server.close()
print(data.decode('utf-8'))
```

Handwritten notes:
 - *Puerto* (pointing to 12345)
 - *cantidad leída del datagrama* (pointing to 4096)
 - *print(data.decode('utf-8'))* (pointing to the print statement)
 - *IP si hay varias tarjetas de red.* (pointing to the empty string in bind)
 - *mensaje remitente* (pointing to clientaddr)

Estudie cada una de las líneas de código que componen ambos programas. Puede consultar para esto la siguiente documentación: <https://docs.python.org/3/library/socket.html>

Desde el servidor ejecute el programa con `py servidorUDP.py`

Desde el cliente ejecute el programa con `py clienteUDP.py`

Reto. ¿Qué habría que incluir en el código para que el servidor pudiera ver el contenido enviado por el cliente?

Reto. ¿Qué habría que incluir en el código para que el servidor pudiera responder al cliente "Bienvenido a clase"?



3. Códigos del cliente y servidor en TCP

Cread el archivo `servidorTCP.py` que contenga el siguiente código:

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("", 9999)) → en qué tarjeta está escuchando
s.listen(1) → poner en modo escucha
Socket ← sc, addr = s.accept() → remitente
while True:
    recibido = s.recv(1024) → recibir 1024 bytes
    if recibido.decode() == "close":
        break
    print(str(addr[0]) + " dice: ", recibido.decode())
    s.send(recibido)
print("Adios.")
s.close() → socket de datos
s.close() → socket de escucha
```

¿Qué indica el entero de la función `listen` del socket?

Cread el archivo `clienteTCP.py` que contenga el siguiente código:

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("localhost", 9999))
while True:
    mensaje = input("Mensaje a enviar >> ")
    s.send(mensaje.encode())
    if mensaje == "close":
        break
print("Adios.")
s.close()
```

Estudie cada una de las líneas de código que componen ambos programas. Puede consultar para esto la siguiente documentación: <https://docs.python.org/3/library/socket.html>

Desde el servidor ejecute el programa con `py servidorTCP.py`

Desde el cliente ejecute el programa con `py clienteTCP.py`

Debe escribir el mensaje a enviar al servidor.

¿Cuál es el mensaje que ha de escribir para que el programa del cliente finalice?

Reto. Rellene los huecos del fichero `servidorWeb.py` para que funcionen como un servidor web y se pueda acceder desde cualquier navegador con la url: <http://localhost:8080/>.

Reto. Montar un servidor que convierta en minúscula todos los mensajes que recibe del cliente y se reenvíe la conversión al cliente.

Reto. Montar un cliente/servidor resistente a fallos en la transmisión a nivel de aplicación. Con las siguientes características:

↑
TCP ya lo hace. Lo suyo es implementarlo en UDP.



- El cliente envía un mensaje
- El servidor al recibirlo lo elimina con un 33% de probabilidad y si es así pide al cliente que lo reenvíe, si no, le dice que se ha recibido correctamente.
- El cliente reenvía el mensaje hasta que el servidor confirma que se ha recibido correctamente.

Para profundizar

Para estudiar en mayor profundidad los conceptos de cliente/servidor utilizando TCP o UDP se recomienda estudiar el capítulo 2 del libro *Computer Networking* [2].

Para ver cómo programar socket concurrentes con *Python* se recomienda el tutorial [3].

Bibliografía

- [1] Visual Studio Code: <https://code.visualstudio.com/>
- [2] James, Kurose, y Ross Keith. *Computer Networking: A Top-Down Approach*. Boston Munich, 2016. Capítulo 2.
- [3] GeeksforGeeks. «Socket Programming with Multi-Threading in Python», 30 de septiembre de 2017. <https://www.geeksforgeeks.org/socket-programming-multi-threading-python/>.