PRÁCTICA 2: DIRECCIONAMIENTO DE APLICACIONES, INTRODUCCIÓN A LOS SOCKETS

El objetivo de esta práctica es doble: (a) repasar cómo los sockets son creados por las aplicaciones e identificados por el sistema operativo de un equipo (proceso de desmultiplexión visto en clase); (b) inspeccionar los sockets activos en un equipo, asociándolos con las aplicaciones que los han creado. Por ello, antes de continuar, quizás quieras revisar con atención los documentos de teoría donde se explican estos temas. Si después de esta revisión todavía te quedan dudas conceptuales puedes consultar otras fuentes o preguntar al profesorado.

Si haces la práctica en el CdC, arranca con Linux ya que en la última parte debes realizar un programa en python. Si lo haces con Windows siempre puedes ejecutar la máquina virtual de la asignatura Fundamentos de Programación**.**

Enlaces de utilidad:

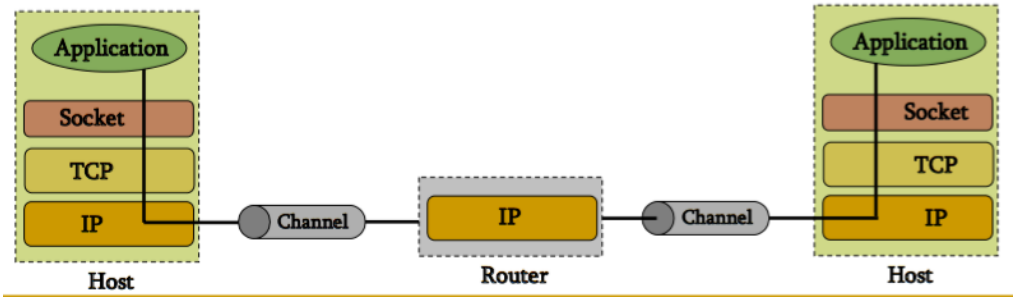
- [1] Sobre python: (tutorial básico): https://www.tutorialspoint.com/python/index.htm
- [2] Sobre Python y sockets: https://www.ibm.com/developerworks/linux/tutorials/l-pysocks/l-pysocks-pdf.pdf
- [3] Python y networking:: https://www.tutorialspoint.com/python/python_networking.htm
- [4] Sockets en Python: https://docs.python.org/3.0/library/socket.html
- [5] tutorial Python http://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/ifstatements.html

Requerimientos para realizar la práctica:

- Ordenador con Linux con un intérprete de Python >=2.X instalado. Opcionalmente, se puede utilizar Windows con un intérprete de Python instalado (ver https://www.python.org/downloads/windows/).
- Leer, entender y practica con algún tutorial básico sobre Python. (2h)

PARTE I: INTRODUCCIÓN A LOS SOCKETS

Los sockets son puntos de comunicación que utilizan los procesos para intercambiar datos entre sí. A través de ellos se pueden enviar o recibir mensajes con/desde otros procesos que estén otra una máquina remota o local (p.ej. cualquier equipo conectado a Internet) de una forma parecida a leer o escribir bytes en un fichero local. Existe una API para utilizar sockets disponible en todos los sistemas operativos para casi todos los lenguajes de programación.



Existen básicamente dos tipos de sockets (en realidad hay más tipos) :

- **Stream sockets** (tipo flujo de bytes). Ofrecen un servicio de transporte de mensajes orientado a conexión y fiable. Usan el protocolo de transporte TCP.
- **Datagram sockets** (tipo datagrama o carta). Ofrecen un servicio best-effort sin conexión previa y con mensajes de un tamaño máximo de 65.500bytes. Usan el protocolo de transporte UDP.

Cada proceso de aplicación que desee usar la red debe crear al menos un socket para poder enviar y recibir mensajes. Cada vez que creamos un nuevo socket, el sistema operativo nos devuelve un nuevo descriptor de fichero asociado a nuestra aplicación. En la estructura interna del descriptor, el sistema operativo guardará información como el tipo de socket creado, direcciones IP y números de puerto locales asociados al socket, etc. Por lo tanto, podemos quedarnos con la idea de que usar los sockets es algo parecido a usar las funciones para acceder a ficheros en un disco (p.ej. en C, la función `fopen` también crea un nuevo descriptor del fichero y `fprintf /fscanf` lo utilizan).

Python nos ofrece un módulo llamado socket a través del cual accedemos a la API de los sockets. Es necesario importar este módulo en nuestro código antes de realizar cualquier llamada los objetos de su librería.

Ejemplo 1: Abra un terminal y ejecute Python en modo línea de comandos.

```
%>touch borrame.txt
%>python
>>> import socket
>>> miSocket = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
>>> miSocket2 = socket.socket( socket.AF_INET, socket.SOCK_DGRAM )
>>> print [miSocket.fileno(),miSocket2.fileno()]
[3, 4]
>>> a = open('./borrame.txt')
>>> quit()
```

En Python, el método de clase socket devuelve un objeto al cual se le pueden aplicar los métodos correspondientes (ver [2][3][4]). En el ejemplo anterior, a través de este método creamos una instancia de un socket de tipo flujo (constante SOCK_STREAM) que utilizará la familia de direcciones de Internet (constante AF_INET). Esta instancia a la que hemos llamado miSocket tiene un manejador asociado (descriptor de fichero) cuyo valor es 3. También creamos otro socket de tipo datagrama cuyo descriptor de fichero es 4. Al usar Python la orientación a objetos, no necesitaremos usar el descriptor de fichero para usar el socket correspondiente (ya que el descriptor de fichero del socket es un atributo del

objeto socket), sino que podemos usar los propios métodos de la instancia en sí. Sin embargo, en otros lenguajes (p.ej. en C) si sería necesario usar este manejador en cada llamada a las funciones de la librería de socket para que el sistema operativo pueda identificar el socket a utilizar.

Del ejemplo anterior se deduce que **un socket siempre estará asociado al proceso de aplicación que lo haya creado**. Así pues, es común que en un host (equipo informático) coexistan numerosos sockets activos tanto de tipo udp como de tipo tcp. En el código anterior, pruebe a abrir un segundo terminal y ejecutar **netstat –tuna** o alternativamente **ss –tuna** (en modo superusuario en Linux,) en Linux para ver los sockets abiertos en su equipo y las aplicaciones que los han creado. Puedes aprender más sobre las opciones de **ss** en https://www.linux.com/learn/intro-to-linux/2017/7/introduction-ss-command . En Windows, puedes abrir un terminal de powershell y escribir **netstat –ano** (en modo administrador de Windows) para ver los sockets que maneja el S.O. y los procesos que han creado cada socket. También puedes usar tasklist para ver el mapeo de pid a nombre de un programa.

Direccionamiento local de sockets: dirección IP y número de puerto.

Cada socket tiene una dirección que lo identifica localmente. Esta dirección es una tupla que consiste en una dirección del interfaz de red (dirección IP si estamos usando sockets de la familia de direcciones de Internet AF_INET) y un número de puerto. En algunos lenguajes como C, esta dirección se encuentra encapsulada en una estructura de datos tediosa de manipular (`struct sockaddr_in`). Sin embargo, en Python, esta se puede escribir directamente como una tupla ('192.168.1.1', 80).

No obstante, también es posible utilizar nombres (alias) en lugar de las direcciones o números puertos correspondientes. Por ejemplo,

- Si usamos el nombre de una máquina en lugar de su dirección IP, se usa de forma transparente el servicio DNS y se acepta la primera respuesta válida como dirección IP. No obstante, antes de preguntar al DNS, se examina el fichero `/etc/hosts` de la máquina. (examina este fichero en tu ordenador para ver cómo a ciertas direcciones IP se les asigna un nombre de forma local).

- Si usamos el nombre de un servicio, estamos *bautizando* con un nombre al número de puerto asociado a su proceso servidor. Por ejemplo, en lugar del puerto 80 asociado al proceso servidor web se puede usar directamente 'http'). Para mapear nombres de servicios a números de puerto se utiliza el fichero **`/etc/services`** de tu ordenador. (examina este fichero en tu ordenador). Recuerda que muchos servicios tienen un número de puerto reservado por la IANA (a nivel mundial). En http://www.iana.org/assignments/service-names-port-numbers/service-

names-port-numbers.xhtml puedes encontrar la lista de números de puerto reservados para nombres de servicio.

Ejemplo 2: Abra un terminal ejecutando python en modo comando (las respuestas obtenidas dependerán de su ordenador y no tienen por qué coincidir con las mostradas.

```
>>> import socket
# nombre y direccion ip de mi maquina.
>>>print socket.gethostname(),":",socket.gethostbyname(socket.gethostname())
argos.us.es : 193.147.162.146

#resolución de nombre y servicio
>>> socket.getaddrinfo('www.marca.es','http')
[(2, 2, 17, '', ('193.110.128.199', 80)),
(2, 1, 6, '', ('193.110.128.199', 80))]

#creamos un socket udp y preguntamos por su direccion.
>>> nuevoSocket = socket.socket( socket.AF_INET, socket.SOCK_DGRAM )
>>> nuevoSocket.sendto( "hola",('trajano.us.es',80))
5
>>> nuevoSocket.getsockname()
('0.0.0.0', 60656)
```

El primer comando utiliza los métodos de clase gethostname() y gethostbyname() para devolver el nombre de la máquina local (recuerda, /etc/hosts) así como su dirección IP respectivamente. En segundo comando, el método `getaddrinfo` realiza una resolución de nombre de máquina usando la aplicación dns de forma transparente. Además, como le damos un nombre de servicio, utiliza el fichero local /etc/services para averiguar el número de puerto asociado al servidor de dicho servicio. Observamos que obtenemos dos respuestas. Esto es debido a que en el fichero aparecen dos entradas asociadas al servicio http (una el puerto 80/tcp y otra el puerto 80/udp).

En la última parte del código creamos un nuevo socket de tipo udp y preguntamos por su dirección antes de utilizarlo. Tras utilizar dicho socket para enviar un mensaje (p.ej. "hola", 5 bytes a la máquina trajano.us.es y el puerto 80/udp), si preguntamos por la dirección asociada a dicho socket, podremos ver el número de puerto que el sistema operativo le ha asignado (en nuestro caso el 60656). **Por lo tanto, cuando creamos un socket, el sistema operativo le asigna automáticamente una dirección al usarlo por primera vez**. Prueba a repetir el envío anterior ("hola" a trajano.us.es) y capturar el paquete con wireshark, examinando tanto la capa de aplicación como la de transporte.

Consultar la lista de sockets activos en un equipo: netstat (o también ss en linux)

El sistema operativo mantiene una lista con los sockets activos en un equipo, identificando a cada socket de forma única y ofreciendo estadísicas sobre el uso de cada uno.

Existe un comando para consultar esta lista de sockets junto con sus direcciones asociadas: **netstat** (en Linux o Windows) o bien **ss** (sólo en Linux). Las opciones de este comando varían con el sistema operativo empleado. Por ejemplo, para consultar la lista de socket activos en Linux:

```
%> netstat –atun
```

Mira en http://www.binarytides.com/linux-netstat-command-examples/ (Linux) o http://www.hsc.fr/ressources/articles/win_net_srv/netstat.html (Windows) para más ejemplos de uso de netstat y sus diferentes opciones. Si quieres ver el pid de cada programa que ha creado cada socket deberás ejecutar el comando netstat en modo administrador o superusuario y poner la opción correspondiente.

Ahora es un buen momento para consultar la lista de sockets activos de tu ordenador y comprobar que nuestro nuevo socket udp del Ejemplo 2 se ha creado y, efectivamente, está vinculado al puerto que hemos averiguado.

Ejemplo 3: Abre un nuevo terminal. Ejecuta netstat y comprueba si existe el socket udp que creaste en el ejemplo anterior. Esta es la salida del comando netstat en el ordenador del profesor (mac os x).

```
argos:~ aestepa$ netstat -an -f inet
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4       0      0  127.0.0.1.6234         *.*                    LISTEN
tcp4      31      0  193.147.162.146.53817  108.160.172.193.443    CLOSE_WAIT
tcp4       0      0  193.147.162.146.53267  162.125.32.129.443     ESTABLISHED
tcp4       0      0  127.0.0.1.631          *.*                    LISTEN
tcp4       0      0  *.17500                *.*                    LISTEN
udp4       0      0  *.60656                *.*
udp4       0      0  *.10000                *.*
udp4     181      0  *.17500                *.*
```

En la tabla anterior podemos encontrar nuestro socket udp asociado al puerto 60656. Además de otros muchos sockets activos que, quizás, no esperábamos. Es importante que sepas que:

- **Un socket udp se utiliza para enviar o recibir mensajes independientemente del extremo remoto con el que nos comuniquemos.** (i.e. con un solo socket nos podemos comunicar una vez con un destino y otra vez con otro destino diferente, y por dicho socket recibiremos mensajes de cualquier origen). **El sistema operativo lo identifica exclusivamente por su dirección local y el puerto local**. En la tabla anterior podemos observar varios sockets de tipo udp asociados a diferentes puertos (la * significa cualquier dirección IP local del equipo).
- **Un socket tcp de conexión se utiliza para enviar y recibir mensajes sólo entre los dos extremos de una conexión (**proceso en ordenador local – proceso en el ordenador remoto). Los **sockets de una conexión TCP se identifican tanto por la IP y puerto local como por la IP y puerto remoto de la conexión**. Estos sockets se crean cuando un cliente pide una conexión que es aceptada por un servidor a través del método correspondiente.
- **Un socket tcp de escucha o bienvenida, sólo se usa para recibir peticiones de conexión remotas**. El socket inicialmente creado por un servidor siempre será de este tipo y sólo los procesos de tipo servidor usan este tipo de sockets. El sistema operativo los identifica sólo por la IP local y puerto local.

Cada vez que llega un nuevo paquete al ordenador, el sistema operativo examina las cabeceras de red y transporte para identificar a qué socket debe repartírselo en función de la tabla anterior.

Averigüe los sockets activos en su equipo. Si omite –n en el comando netstat, en lugar de números de puerto, podrá ver los nombres de las máquinas y servicios asociados. Si usted tiene como puerto local algún nombre de servicio significa que tiene ejecutando un servidor. Truco, **si quiere filtrar los resultados de netstat pruebe a hacer un pipe con grep** (p.ej. netstat –an | grep udp).

Ejercicio 1: Averiguar los sockets tcp y udp activos en el equipo. Mirar si se está ejecutando algún servidor (consejo, mira los números de puerto en /etc/services o elimina la opción n).

Ejercicio 2: Abrir dos terminales. Grabar en un fichero la salida de `netstat` filtrando sólo los sockets de tipo udp. Crear un programa en Python que cree dos sockets nuevos de tipo udp. En otra ventana, volver a repetir el paso 1. Ejecute el comando diff para ver las diferencias entre los dos ficheros. ¿Qué dirección tienen los dos sockets que has creado?. Pruebe desde el terminal de Python cerrar los dos sockets creados (use el método de instancia `close()`). Compruebe con netstat que dichos sockets han desaparecido de la lista de sockets activos

Ejercicio 3: mira el ejemplo de Listing 10 del documento tutorial tercero de enseñanza virtual (sockets in Python). Cree un socket tcp en y conéctese con el servidor web de trajano.us.es en el puerto 80. Pruebe a identificar su nuevo socket de conexión a través del comando netstat.

Ejercicio 4(opcional) si eres root o administrador, responde: ¿Qué proceso ha creado cada socket en tu ordenador? (existe una opción de netstat para ello pero debe ser ejecutado en modo administrador). En Windows, puedes usar el comando tasklist para ver el nombre de un programa dado su pid.

Selección de la dirección de un socket

Los procesos pueden ser clasificados como tipo cliente o tipo servidor. Los procesos de tipo servidor reciben peticiones de procesos de tipo cliente. Por ello, los clientes deben saber la dirección del socket al que dirigir sus peticiones. **Ello implica que tanto la dirección IP (o nombre de máquina) como el número de puerto de los servidores deben ser públicos, o, al menos, conocidos por los clientes.** Por ello, al escribir el código de un servidor nos interesará asignar una dirección concreta a su socket. Para ello se utiliza el método **`bind`**.

Ejemplo 4: Abra un nuevo terminal y ejecute python en línea de comandos

```
>>> import socket
>>> dgramSock = socket.socket( socket.AF_INET, socket.SOCK_DGRAM )
```

```
>>> dgramSock.bind( ('127.0.0.1', 53000) )
>>> dgramSock.getsockname()
('127.0.0.1', 53000)
```

Ahora pruebe a mirar con netstat en otro terminal si este socket aparece en la lista del sistema operativo. ¿Aparece?. Ya puede cerrar los dos terminales.

Ejemplo 5: Abra un nuevo terminal y hagamos un servidor cuyo socket udp esté asociado al puerto 23000 de cualquier interfaz de red nuestra máquina (dirección comodín 0.0.0.0). A continuación, reciba un mensaje a través de dicho socket

```
>>> import socket
>>> dgramSock = socket.socket( socket.AF_INET, socket.SOCK_DGRAM )
>>> dgramSock.bind( ('', 23000) )
>>> dgramSock.getsockname()
('0.0.0.0', 23000)
>>> msg, (addr, port) = dgramSock.recvfrom( 100 )
```

Como puede comprobar, el terminal se queda bloqueado al ejecutar la orden de recibir 100 bytes. Este método es bloqueante y no podremos seguir escribiendo hasta que no haya recibido algo por la red.

Ahora abra un segundo terminal y ejecute en modo comando.

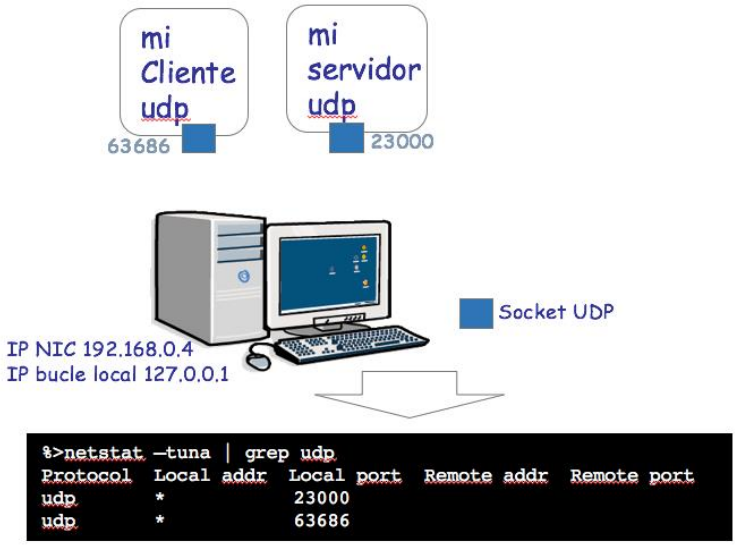
```
>>> import socket
>>> dgramSock = socket.socket( socket.AF_INET, socket.SOCK_DGRAM )
>>> dgramSock.sendto("hola", ('127.0.0.1', 23000) )
4
```

Como puede comprobar, el terminal del servidor se desbloquea al recibir el mensaje. Ahora podemos escribir en el terminal del servidor la siguiente línea:

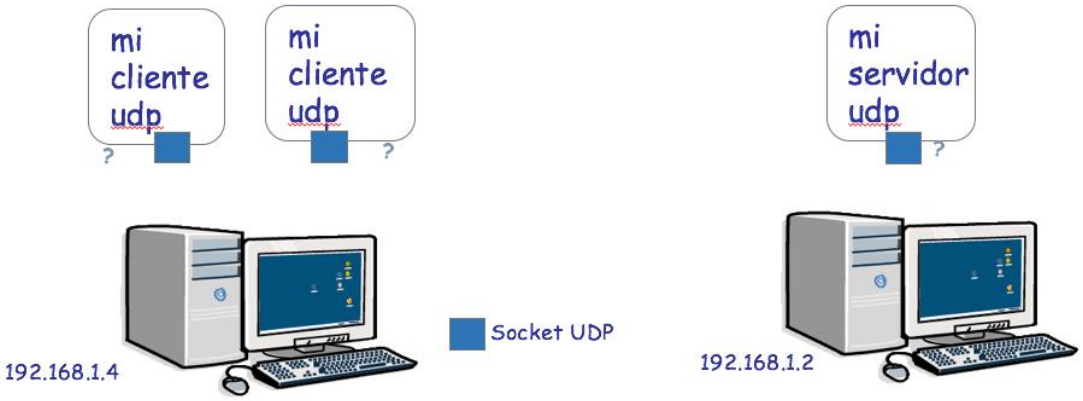
```
>>> print msg, "recibido de", addr,port
hola recibido de 127.0.0.1 63686
```

Como podemos comprobar, el proceso cliente ha creado un socket udp cuya dirección (asignada por el S.O.) es ('0.0.0.0',63686), mientras que el proceso servidor ha creado un socket udp cuya dirección (asignada por nosotros) es ('0.0.0.0',23000). Dado que la dirección 0.0.0.0 representa cualquier dirección IP de la máquina local (p.ej. 127.0.0.1, o 192.168.0.4), siempre que enviemos un mensaje a una dirección IP donde se ejecuta el servidor y al puerto 23000, llegará al proceso servidor. También podemos observar que el método de recepción nos devuelve no sólo el mensaje, sino la dirección del socket que nos lo envió. De esta forma el servidor podría responder enviando un nuevo mensaje al cliente.

Gráficamente, podríamos representar la comunicación entre sockets de ambos procesos como:



Ejercicio 5. Dibuje un esquema similar al anterior pero con dos hosts, donde el proceso servidor usa un socket udp vinculado al puerto 22222 y se ejecuta en una máquina con la dirección IP 192.168.1.2. Contra este proceso servidor se comunicarán dos procesos clientes similares que se ejecutarán en una máquina con la dirección IP 192.168.1.4 (invéntese el número de puerto que el sistema operativo asignaría a cada cliente). ¿Podría escribir la lista de sockets activos en cada máquina?.

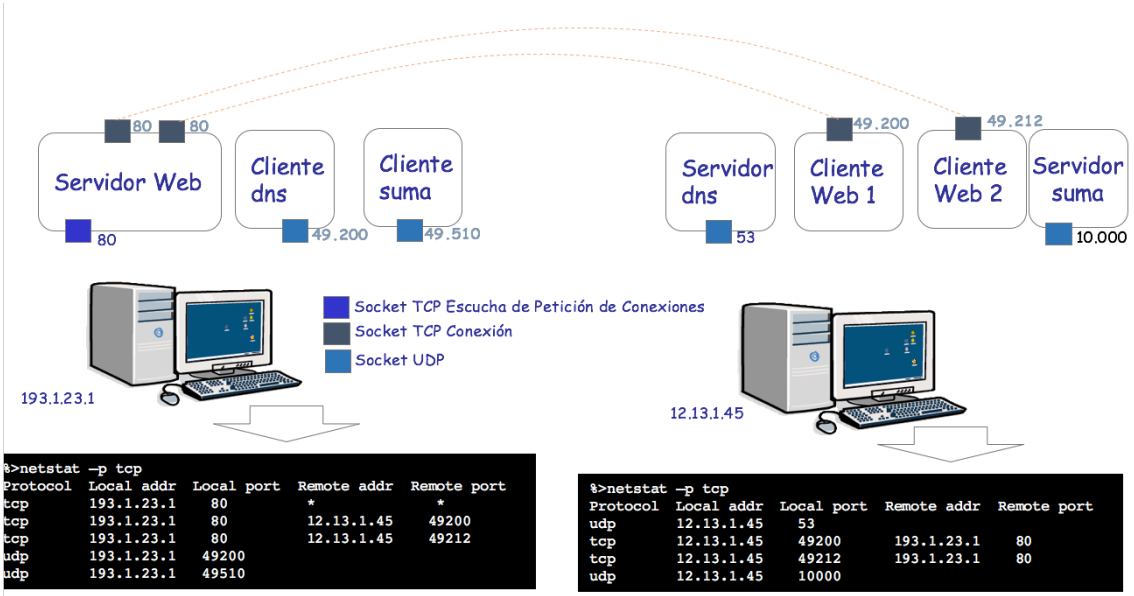


Ejercicio 6. Pruebe a realizar el ejercicio 5 en la realidad con un compañero cuyo ordenador se ejecute en la misma subred que la suya o simplemente usted sólo en el CdC usando dos equipos

.

Finalmente, recuerde que los sockets de tipo TCP son diferentes. El servidor TCP debe crear un socket (de escucha) y vincularlo a una dirección. Pero dicho socket sólo es utilizado para recibir peticiones de conexión. Cada vez que el servidor acepta un nueva petición de conexión, se crea un nuevo socket TCP de tipo conexión que el sistema operativo identifica por los parámetros de la conexión (IPlocal:PuertoLocal – IPremota:PuertoRemoto). Por lo tanto, si un servidor

aceptase simultáneamente a varios clientes remotos (p.ej. usando distintos hilos), estaría creando un socket por cada conexión. Esto se ilustra en la siguiente figura, donde las conexiones se han dibujado como líneas punteadas.



Ejercicio 7. Examine la figura anterior y conteste: (a) a cuantos clientes está conectado el servidor web? (b) a qué dirección IP y puerto tendrá que enviar una operación el cliente suma? (c) a cuántos clientes puede responder como máximo el servidor dns?

Ejercicio 8. Imagínese el siguiente escenario. El equipo A (IPa) ejecuta: 1 cliente DNS, 2 clientes webs, 1 cliente SMTP. El el equipo B (IPb) ejecuta: un servidor de cada una de las aplicaciones anteriores y un cliente web. Suponga que todos los clientes se comunican con su servidor correspondiente. (nota: el servidor SMTP usa el puerto 25/tcp). Dibuje en un esquema como los anteriores los procesos de aplicación y sockets que se ejecutan en los equipos A y B. Una en su diagrama con una flecha los sockets de tipo TCP que estén conectados. Escriba la tabla de sockets activos de cada equipo (A y B) de tipo UDP y de tipo TCP (tanto de escucha como de conexión)

Ejercicio 9: (opcional). ¿Serías capaz de crear todos los sockets del ejercicio 8 con python usando un nuevo terminal para cada proceso? ¿Podrías verificar lo con netstat? (puedes hacerlo en dos equipos o hacerlo todo en un solo equipo)

¿QUÉ TENDRÍAS QUE HABER APRENDIDO?

- Cómo las aplicaciones crean sockets para el uso de la red y los diferentes tipos de sockets que puedes usar en tus aplicaciones.
- **Consultar la lista de sockets activos** en un equipo
- Cómo el sistema operativo hace llegar a cada socket local los mensajes que llegan por la red en función de las cabeceras IP y TCP/UDP recibidas.
- Cómo crear sockets de tipo udp o tcp y consultar su dirección asociada.
- Cómo establecer la dirección de un socket usando el método bind.

- Dibujar esquemáticamente los procesos de aplicación y los sockets creados por éstos, relacionando los sockets de contexión tcp entre sí.