	UNIVERSIDAD MAYOR DE SAN ANDRÉS		LAB-273	
			VERSIÓN: 01	PÁGINA: 1 de 3
	CARRERA DE INFORMÁTICA		FECHA: 04/2021	
	GUÍA PARA PRÁCTICAS DE LABORATORIO DE TELEMÁTICA (LAB273)		VIGENCIA: 2021	

LABORATORIO N° 9

APLICACIONES DE RED CON PYTHON

1. OBJETIVOS DE APRENDIZAJE

- Aplicar el concepto de socket en el desarrollo aplicaciones distribuidas usando el modelo Cliente-Servidor.
- Construir simples aplicaciones cliente-servidor usando el API de sockets provistas por Python

2. HERRAMIENTAS Y MATERIALES REQUERIDOS

- Equipo requerido. Un PC Linux/Windows con conexión a internet
- Software. Python y un IDE de su preferencia

3. INTRODUCCION

Python es un lenguaje de programación interpretado multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es administrado por la Python Software Foundation y posee una licencia de código abierto.

Las implementaciones estándar de Python compilan declaraciones de código fuente a un formato intermedio conocido como código de bytes lo que proporciona portabilidad, ya que es un formato independiente de la plataforma. También como Python es de código abierto, se incluye automáticamente con distribuciones de Linux, computadoras Macintosh y algunos productos y hardware.

Python tiene una gran biblioteca estándar, usada para una diversidad de tareas, y en lo que se refiere a la programación de aplicaciones de red, viene con módulos estándar de Internet que permiten que los programas realicen una amplia variedad de tareas de red, en modo cliente y servidor. Los programas en Python pueden comunicarse usando sockets, transferir archivos por FTP; analizar archivos XML, enviar, recibir correos electrónicos, buscar páginas web por URL y más. Las bibliotecas de Python hacen que estas tareas sean extraordinariamente sencillas.

Los servicios de red pueden verse a diferente escala, a un nivel básico, los sockets representan la interfaz entre la capa de aplicación y la de red y pueden ser implementados de diferentes modos: sockets de dominio UNIX, TCP, UDP, etc. La biblioteca socket de Python provee clases específicas para manejar el transporte común, así como también una interfaz genérica para controlar todo lo demás.

Módulo socket


Todos los sockets son creados mediante la función *socket*, la cual tiene la siguiente sintaxis:

```
misocket = socket(socket_family, socket_type)
```

- *socket_family*: es la familia de protocolos que es usada como mecanismo de transporte. Estos valores son constantes tales como AF_INET, PF_INET, PF_UNIX, PF_X25, entre otras.
- *socket_type*: el tipo de comunicación entre los dos extremos de la conexión, usualmente se usa SOCK_STREAM para protocolos orientados a conexión y SOCK_DGRAM para protocolos sin conexión.

Entre los principales métodos de los objetos socket se tienen:

- *socket.bind()*: este método vincula una dirección (hostname, nro de puerto) a un socket.

	UNIVERSIDAD MAYOR DE SAN ANDRÉS		LAB-273	
	CARRERA DE INFORMÁTICA		VERSIÓN: 01	PÁGINA: 2 de 3
	GUÍA PARA PRÁCTICAS DE LABORATORIO DE TELEMÁTICA (LAB273)		FECHA: 04/2021	
			VIGENCIA: 2021	

- `socket.listen()`: configura e inicia un oyente TCP.
- `socket.accept()`: acepta una conexión de cliente TCP y espera hasta que se realice
- `socket.connect()`: inicia una conexión con un servidor TCP
- `socket.recv()`: recibe un mensaje TCP
- `socket.send()`: transmite un mensaje TCP
- `socket.recvfrom()`: recibe un mensaje UDP
- `socket.sendto()`: transmite un mensaje UDP
- `socket.close()`: cierra un socket

Referencias Bibliográficas. Disponibles en la plataforma virtual del curso

- Documentación oficial del sitio de Python - <https://docs.python.org/3.4/tutorial/>
- Guía de Programación en Redes con Python. Elaborado por los auxiliares de la signatura (Alberth Michael Apaza Apaza y Vladimir Bony Chacolla Condori)

4. PROCEDIMIENTO

El módulo socket de Python provee una interfaz para la API para los sockets de Internet. Por lo que a continuación presentaremos el código básico para servidores y clientes que se comunicaran mediante sockets usando TCP y UDP.

Comunicación usando TCP

Programa Servidor	Programa Cliente
<pre> from socket import * addr = ("localhost", 21567) srvSock = socket(AF_INET, SOCK_STREAM) srvSock.bind(addr) srvSock.listen() while True: cliSock, addr = srvSock.accept() print ('...conexion recibida') data = cliSock.recv(200) cliSock.send(b"Resp: " + data) cliSock.close() srvSock.close() </pre>	<pre> from socket import * dest = ('localhost', 21567) cliSock = socket(AF_INET, SOCK_STREAM) cliSock.connect(dest) data = input('> ') cliSock.send(data.encode("utf-8")) data = cliSock.recv(200) print(data.decode("utf-8")) cliSock.close() </pre>


El programa servidor atenderá peticiones en el puerto especificado, por cada petición recibida devolverá el mensaje enviado por el cliente y permanecerá escuchando por nuevas peticiones. El programa cliente se conectará con el servidor e ingresará por el teclado el mensaje a enviar, cuando reciba la respuesta terminará su ejecución.

Ejecute ambos programas y verifique la comunicación. Nótese el código compacto y el grado de abstracción que proporciona Python permitiendo una fácil lectura y comprensión.

Comunicación usando UDP

Los servidores UDP no requieren mucha configuración como sus pares TCP ya que no son orientados a la conexión, solo deberán esperar conexiones.

A continuación, se muestra un ejemplo de una comunicación básica ente un cliente y un servidor usando UDP.

	UNIVERSIDAD MAYOR DE SAN ANDRÉS		LAB-273	
	CARRERA DE INFORMÁTICA		VERSIÓN: 01	PÁGINA: 3 de 3
	GUÍA PARA PRÁCTICAS DE LABORATORIO DE TELEMÁTICA (LAB273)		FECHA: 04/2021	
			VIGENCIA: 2021	

Programa Servidor <pre> from socket import * from time import ctime ADDR = ('', 21567) srvsock = socket(AF_INET, SOCK_DGRAM) srvsock.bind(ADDR) while True: print('esperando mensaje...') data, addr = srvsock.recvfrom(512) srvsock.sendto(b'Res: ' + data, addr) print('se recibió:', data.decode()) srvsock.close() </pre>	Programa Cliente <pre> from socket import * ADDR = ('localhost', 21567) cliSock = socket(AF_INET, SOCK_DGRAM) data = input('> ') cliSock.sendto(data.encode("utf-8"), ADDR) data, ADDR = cliSock.recvfrom(512) print(data.decode("utf-8")) cliSock.close() </pre>
---	--

Ejecute los programas y verifique la operación

5. EJERCICIOS

1. Desarrolle un servidor UDP que retorne la fecha y hora. Asimismo, el servidor aleatoriamente responderá solo a la mitad de las peticiones recibidas simulando una situación de pérdida de paquetes. Cuando no deba enviar la respuesta deberá desplegar un mensaje indicando que ha descartado el paquete y cerrar la conexión.
2. Desarrolle un programa cliente con sockets TCP para conectarse a un servidor web. Su programa deberá enviar peticiones HEAD y mostrar la respuesta del servidor.