2017

TALLER DE PROYECTO II

Trabajo Práctico N°1

Preisegger, Juan Santiago	407/7
Libutti, Leandro Ariel	759/0
Astudillo, Matías Nahuel	309/5

Contenido

Ejercicio 1

Luego de declarar las dependencias en el archivo de texto requierements.txt (Figura 1), procedemos a instalarlas ejecutando el comando sudo pip install -r requierements.txt (Figura 2). La dependencia que instalamos es el servidor Flask.

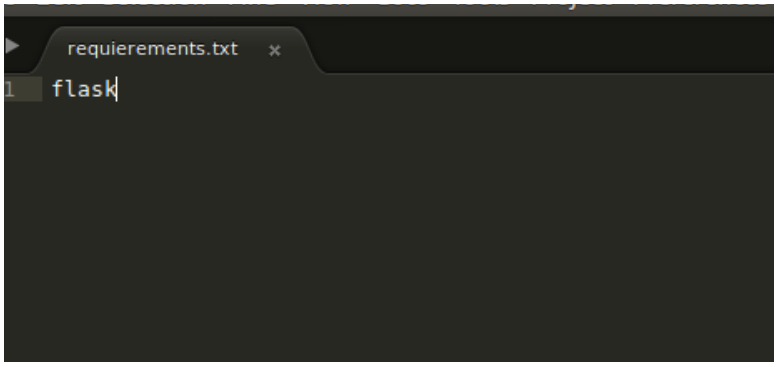

Figura 1. Contenido del archivo requierements.txt

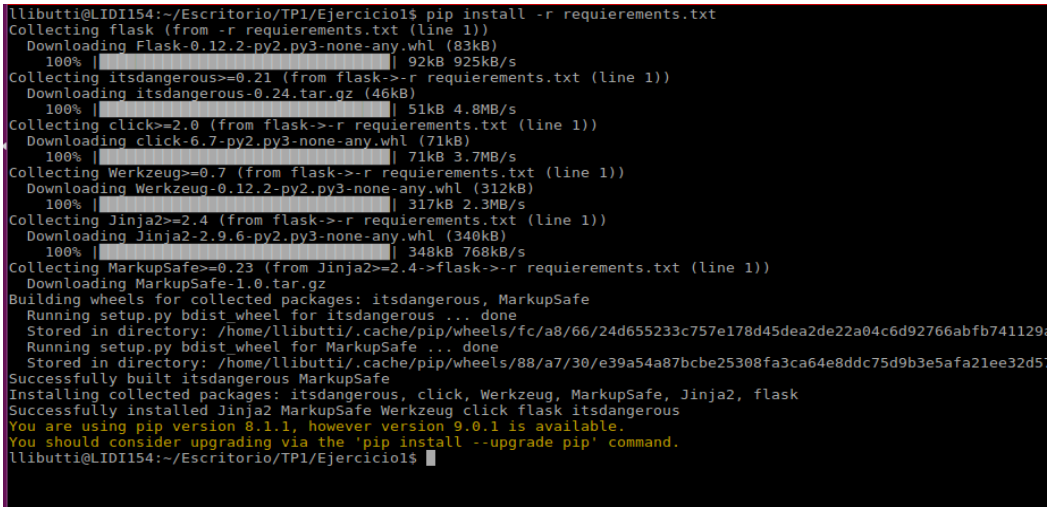

Figura 2. Instalación del archivo requierements.txt

Un ejemplo de uso del servidor Flask es el siguiente:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
      return 'Hola amigos de Geeky Theory!'

if __name__=='__main__':
      app.run(host='0.0.0.0')
```

Se puede observar que luego que el cliente ingresa a través de un explorador de web la ip 0.0.0.0 de localhost se envía una petición de acceso al

servidor el cual crea una instancia de *app* con el argumento *__main__* necesario para que Flask busque files, plantillas y otros archivos. Luego el cliente recibirá la respuesta generada por el servidor definida en la función de la ruta “/” la cual retorna un string en pantalla del navegador.

NOTA: El archivo requierements.txt se encuentra en la carpeta “Trabajo Practico 1/Ejercicio 2” del repositorio de GitHub.

Ejercicio 2

Para obtener la demostración de cómo llega la información del lado del cliente en forma de HTML se define una función utilizando el lenguaje Python. Esta función permite obtener los datos enviados desde el cliente al servidor. A continuación, se detalla la función implementada:

```
@app.route('/form', methods = ['GET'])
def action_form (frecuencia= None):
      frecuencia = int(request.args["frecuencia"])
      return render_template('response.html', frec= frecuencia)
```

Luego de que el servidor obtiene el dato de la frecuencia, la misma es enviada al cliente para que pueda visualizarla a través de la función render_template.

El código utilizado en el archivo HTML es el siguiente:

```
<html>
          <head>
              <h1>Datos Recibido</h1>
          </head>

            <body>
                  <p> La variable recibida de python se informa de
                  la siguiente manera {{ frec }} </p>
            </body>
      </html>
```

La ejecución del programa se realiza desde consola ingresando el comando “Python ejemplo.py” (Figura 3).

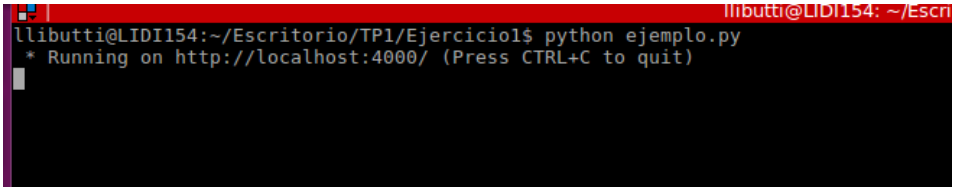

Figura 3. Ejecución del ejemplo desarrollado.

Por último, utilizamos la herramienta wireshark para mostrar la información que contiene una consulta y respuesta en HTTP con python. Esta herramienta provee una función denominada “Follow HTTP Stream” que permite visualizar la información de la consulta en color rojo y la respuesta en color azul (Figura 4).

La consulta puede observarse que se realiza desde el puerto 4000 de localhost y es una petición GET. Luego de realizada la petición, HTTP genera la respuesta la cual posee el texto html y la cabecera HTTP donde informa si la respuesta es correcta (valor 200 OK) y los servidores utilizados (Werkzeug y Python) (Figura 5).

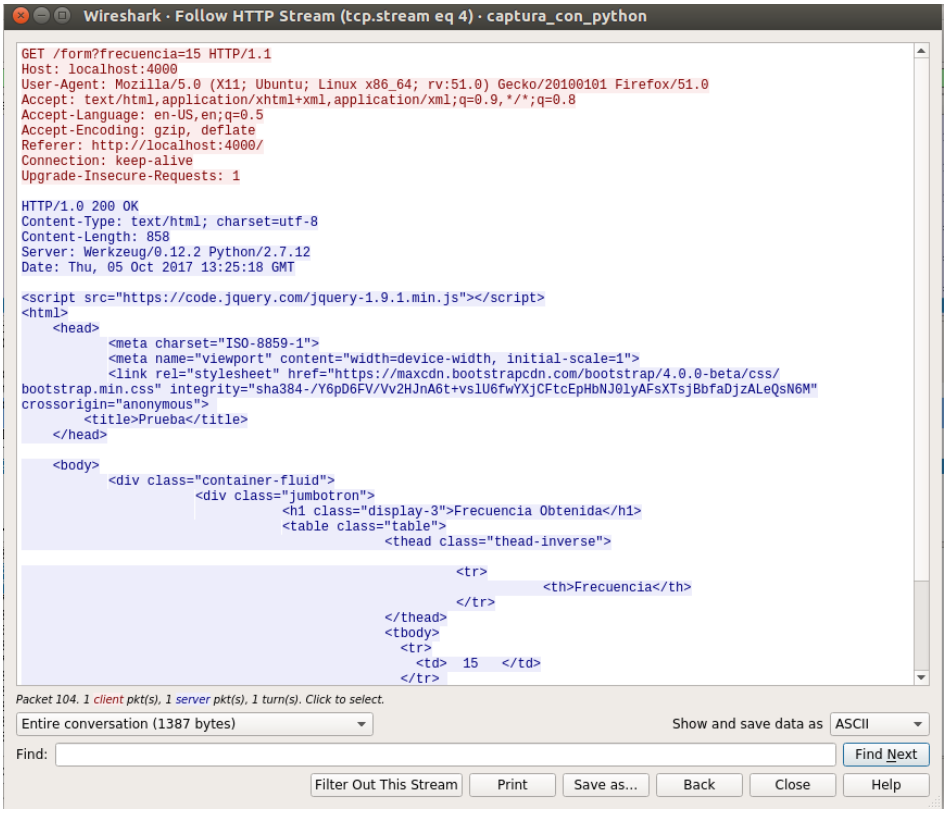

Figura 4. Consulta y respuesta visualizada con Follow HTTP Stream.

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 858
Server: Werkzeug/0.12.2 Python/2.7.12
Date: Thu, 05 Oct 2017 13:25:18 GMT
```

Figura 5. Cabecera de la respuesta HTTP con python

NOTA: El archivo de la prueba se encuentra en la carpeta “Trabajo Practico 1/Ejercicio 2” del repositorio de GitHub.

Ejercicio 3

Para realizar esta solución se crearon dos archivos en python los cuales uno actúa como productor de datos (productor.py). Las acciones que realiza este proceso son:

1. El cliente debe ingresar el tiempo que debe transcurrir entre la generación de las muestras.
2. Conectar la base de datos MySQL.
3. Generar dato aleatorio.
4. Realizar la consulta en MySQL para ingresar el dato generado.
5. Realizar “commit” de la consulta realizada para que el dato generado quede permanente en la base de datos.
6. Vuelve al punto 3.

El otro proceso es el consume los datos de la base de datos MySQL (consumidor.py). Las acciones que realiza este proceso son:

1. El cliente debe ingresar el periodo de muestreo.
2. Conectar la base de datos MySQL.
3. Realizar la consulta en MySQL para obtener una tupla de la base de datos.
4. Enviar los datos al cliente en formato HTML, junto con los promedios solicitados.
5. El script del lado de HTML realiza la acción de refrescar la pantalla (a partir del periodo de muestreo ingresado).
6. Por último, en el HTML se muestra la información de la tupla actual que se está analizando y los datos de los promedios solicitados.
7. Se realiza el punto 2 nuevamente hasta analizar todos los datos presentes en la base de datos.

Antes de realizar la ejecución se debe instalar el archivo con las dependencias necesarias (Figura 6 y Figura 7).

Luego ejecutamos desde la consola del sistema operativo los archivos consumidor.py y productor.py simultáneamente (Figura 8 y Figura 9).

Figura 6. Dependencias del archivo requierements.txt

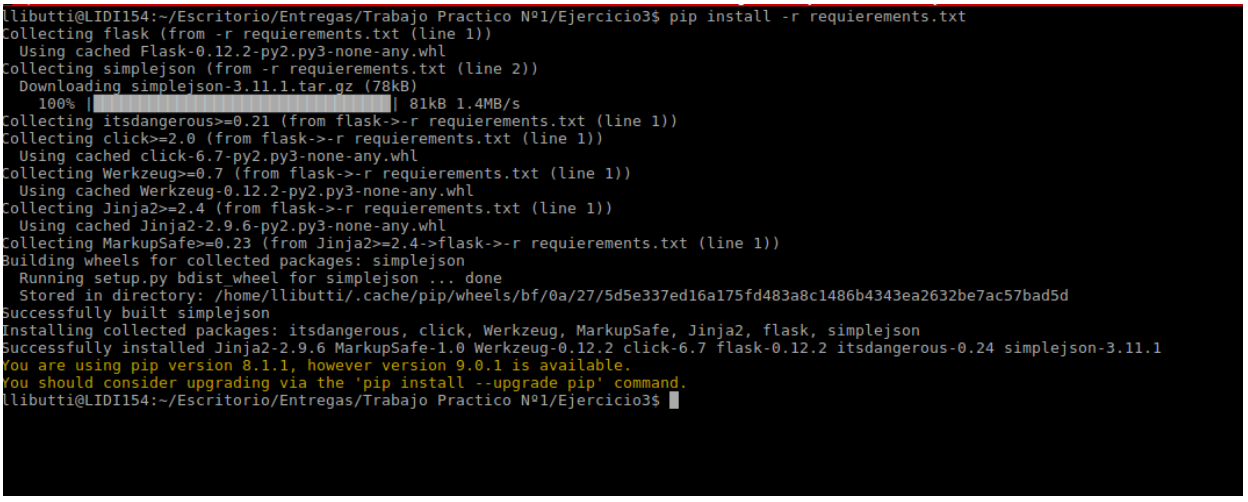

Figura 7. Instalación del archivo requierements.txt

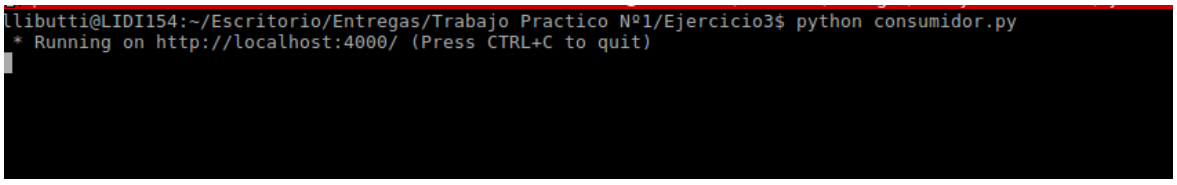

Figura 8. Ejecución del archivo consumidor.py

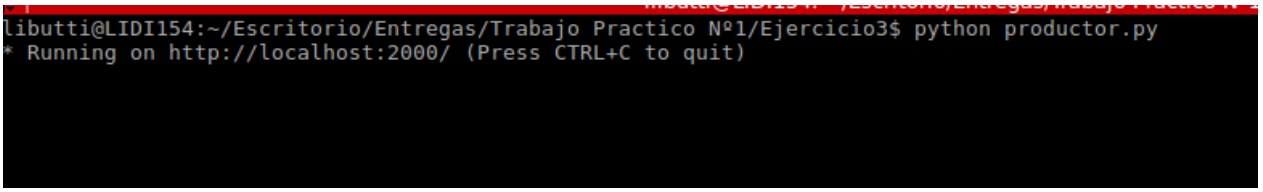

Figura 9. Ejecución del archivo productor.py

ACLARACIÓN: antes de ejecutar los archivos consumidor.py y productor.py, linux debe tener instalada la librería python-mysqldb la cual es necesaria instalar desde consola. El comando utilizado es “sudo apt-get install python-mysqldb” (Figura 10).

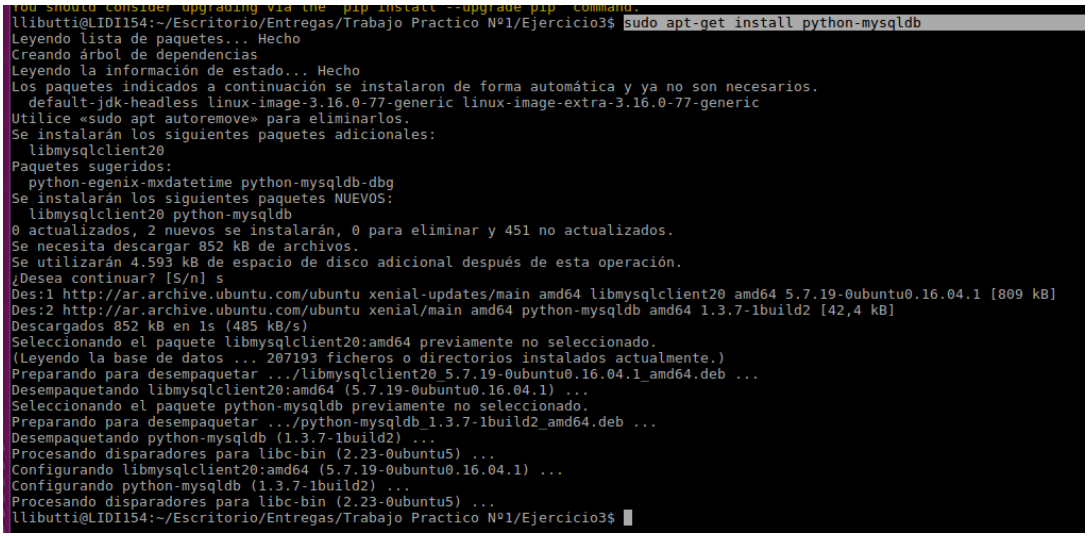

Figura 10. Instalación de la librería MySQL-python

NOTA: El archivo de la prueba se encuentra en la carpeta “Trabajo Practico 1/Ejercicio 3” del repositorio de GitHub.

Ejercicio 4

Para este ejercicio se utilizó la misma solución que el ejercicio anterior. La única modificación realizada es la plantilla inicial (form_c.html y form_p.html). En la misma se insertó el tag select para que el usuario elija entre las opciones predefinidas para el periodo de muestreo.

Para ejecutar los archivos de Python se utiliza el mismo procedimiento explicado en el ejercicio 3.

NOTA: El archivo de la prueba se encuentra en la carpeta “Trabajo Práctico 1/Ejercicio 4” del repositorio de GitHub.

Ejercicio 5

La problemática de la concurrencia existente en la simulación se halla en que ambos procesos intentan acceder al mismo tiempo a la base de datos (utilizar el mismo recurso), lo que produciría un bloqueo en el acceso de uno de los procesos. Al mismo tiempo, al agregar la posibilidad de cambiar el período de muestreo, lo que ocurriría es que puede que uno de los dos procesos este a una frecuencia mayor que el otro. Lo que esto conlleva a que uno intente leer información que el otro no escribió todavía.

En el caso de que sea en un ambiente real en el cual estamos queriendo medir diversas variables a una frecuencia indicada, deberíamos analizar la frecuencia a la que el sensor nos envía los datos para así configurar el microcontrolador a la misma frecuencia para no perder datos, como así también, no leer datos erróneos o señales sin sentido. Esto ocurriría en el caso de que uno intenta escribir información que el otro no alcanza a leer y también se repite el caso de error mencionado en la simulación.

Los problemas de concurrencia existentes en el ambiente real podrían darse en el caso de que existan varios sensores conectados a la placa intentando enviarle información por el mismo puerto debería coordinarse los tiempos de envío porque si no existirían colisiones entre los valores enviados, produciendo así datos incorrectos. Otro posible problema se da cuando todos los sensores están conectados a distintos puertos y el microcontrolador los va sensando de a uno pero cuando este intenta enviarlos al servidor (y este está siendo accedido por lectores) no puede acceder, por lo tanto, almacena los valores hasta tener acceso. Esto dependerá de la memoria que se disponga, si esta se agota por el tiempo de espera se perderán valores.
Posibles problemas de tiempo real que podrían producirse en general:

- No tener responsividad. Los sensores envían sus datos mediante interrupciones, generando así anidamientos para atenderlas e informando así datos críticos cuando ya el suceso pasó.
- No tener determinismo. Llegó un valor de un sensor por el cual había que activar un actuador de una manera determinada y la señal hacia este no llegó, produciendo así una falla en la funcionalidad final del sistema.
- No tener confiabilidad. El sistema en ambientes electromagnéticos o a determinada temperatura no anda o los sensores envían datos erróneos.

Ejercicio 6

En la simulación planteada por nosotros tenemos una aplicación que genera datos corriendo en un puerto, la cual escribe en una base de datos y, a su vez, tenemos una segunda aplicación que lee la base de datos y muestrea a una determinada frecuencia los datos. En este caso no existirían errores como los analizados ya que la base de datos no es leída por más de un proceso en simultáneo, de la misma manera que no se pierden datos nunca y no existen datos erróneos.
En el caso real podrían producirse datos erróneos por algún problema en sensores o por colisiones en las redes de comunicación utilizadas y, a su vez, se podría perder información porque los sensores no mantienen los datos por tiempo infinito y si el microcontrolador no los sensa en el momento apropiado estos se perderían.