Taller de Proyecto 2

Práctica 1

Integrantes:

- Arcuri, Antonio 538/8
- Blanco Regojo, Mauricio 492/0
- Santarcangelo Zazzetta, Marco 652/8

Indice

Página

Enunciado 1)

Generar el archivo ‘requirements.txt’ con las dependencias necesarias para poder levantar un servidor con Flask. Explicar un ejemplo de uso con la secuencia de acciones y procesos involucrados desde el inicio de la interacción con un usuario hasta que el usuario recibe la respuesta.

Resolución ejercicio 1:

Para la resolución de este ejercicio- y toda la presente práctica -se hizo uso de una computadora con sistema operativo Ubuntu 16.04.

En primera medida se deberá instalar tanto Python como Pip, siendo Python un lenguaje de programación y Pip un sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en lenguaje Python. Para esto, utilizando el sistema operativo Ubuntu, en la terminal se ingresarán los siguientes comandos:

```
sudo apt-get install Python

sudo apt-get install Python-pip
```

Pip permite la instalación de un conjunto de paquetes, en este caso imprescindibles para levantar un servidor con Flask, a partir de la creación de un documento “requirements.txt” en el cual se encuentran detalladas las dependencias necesarias. Dentro del repositorio de GitHub del proyecto se encontrará este documento, el cual solo tiene como ítem:

```
Flask
```

Una vez generado el archivo “requirements.txt”, este tiene que ser ejecutado ejecutando el comando:

```
pip install –r requirements.txt
```

El proceso se compone de dos pasos:

1. El usuario solicita acceder a la página escribiendo la dirección en el navegador, lo cual genera un método GET de HTTP.
2. El servidor responde enviando el documento HTML mediante HTTP (método GET) que es visto por el usuario en el navegador.

La Fig. 1. demuestra la metodología de conexión entre usuarios y el servidor, siendo la red de comunicaciones el navegador web.

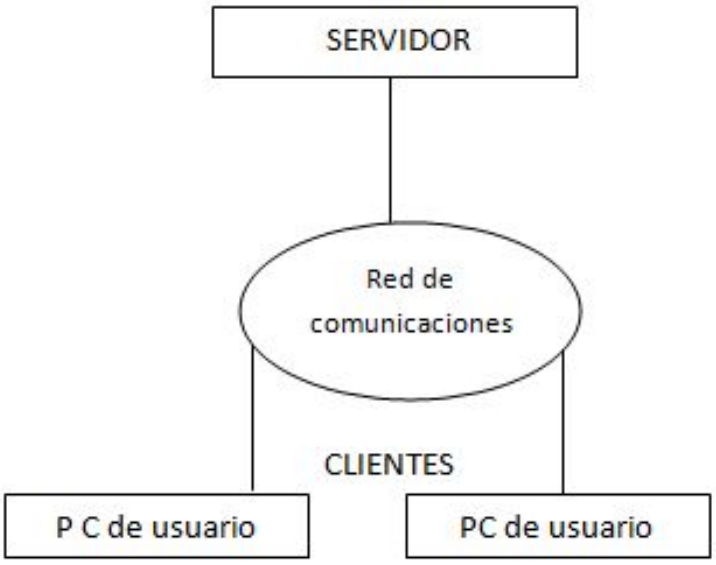

Fig. 1. Gráfico de interacción entre los usuarios y el servidor.

Enunciado 2)

Desarrollar un experimento que muestre si el servidor HTTP agrega o quita información a la genera un programa Python. Nota: debería programar o utilizar un programa Python para conocer exactamente lo que genera y debería mostrar la información que llega del lado del cliente, a nivel de HTTP o, al menos, a nivel de HTML (preferentemente HTTP).

Resolución ejercicio 2:

El servidor agrega información con el objetivo de poder enviar los datos generados originalmente desde el cliente al servidor y viceversa.

El programa utilizado es el ejemplo otorgado por la cátedra, python.ej, el cual genera información sobre el nombre, el email y el sexo del usuario.

A continuación se muestra el tráfico HTTP correspondiente a la aplicación python capturado mediante Wireshark. Las mismas muestran datos de los diferentes protocolos de las capas del modelo OSI. Con el fin de facilitar la lectura se omitió la información de los protocolos TCP e IP, pero permanecen en la captura. Los datos propios de la aplicación python están marcados en negrita.

Primer GET, realizado por el usuario

```
Frame 74: 360 bytes on wire (2880 bits), 360 bytes captured (2880 bits) on
interface 0
Linux cooked capture
```

```
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 42910, Dst Port: 8080, Seq: 1, Ack: 1,
Len: 292
Hypertext Transfer Protocol
       GET / HTTP/1.1\r\n
       [Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]
              [GET / HTTP/1.1\r\n]
              [Severity level: Chat]
              [Group: Sequence]
       Request Method: GET
       Request URI: /
       Request Version: HTTP/1.1
       Host: localhost:8080\r\n
       User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:45.0)
Gecko/20100101
Firefox/45.0\r\n
```

Respuesta del servidor, con el código HTML del formulario

```
Frame 88: 5402 bytes on wire (43216 bits), 5402 bytes captured (43216 bits) on
interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 8080, Dst Port: 42910, Seq: 158, Ack:
293, Len: 5334
Hypertext Transfer Protocol
       HTTP/1.0 200 OK\r\n
       [Expert Info (Chat/Sequence): HTTP/1.0 200 OK\r\n]
              [HTTP/1.0 200 OK\r\n]
              [Severity level: Chat]
              [Group: Sequence]
       Request Version: HTTP/1.0
       Status Code: 200
       Response Phrase: OK
       Content-Type: text/html; charset=utf-8\r\n
       Content-Length: 5334\r\n
       [Content length: 5334]
       Server: Werkzeug/0.12.2 Python/2.7.12\r\n
       Date: Tue, 05 Sep 2017 22:23:01 GMT\r\n
       \r\n
       [HTTP response 1/1]
       [Time since request: 0.001701658 seconds]
       [Request in frame: 74]
       File Data: 5334 bytes
Line-based text data: text/html
       <html>\n
       [...código HTML del formulario...]
       </html>
```

POST realizado por el usuario, pueden verse los datos introducidos en el formulario

```
Frame 265: 528 bytes on wire (4224 bits), 528 bytes captured (4224 bits) on
interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 42916, Dst Port: 8080, Seq: 1, Ack: 1,
Len: 460
Hypertext Transfer Protocol
       POST /form HTTP/1.1\r\n
       [Expert Info (Chat/Sequence): POST /form HTTP/1.1\r\n]
              [POST /form HTTP/1.1\r\n]
              [Severity level: Chat]
              [Group: Sequence]
       Request Method: POST
       Request URI: /form
       Request Version: HTTP/1.1
       Host: localhost:8080\r\n
       User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:45.0)
Gecko/20100101 Firefox/45.0\r\n
HTML Form URL Encoded: application/x-www-form-urlencoded
       Form item: "nombre" = "marco"
       Key: nombre
       Value: marco
       Form item: "apellido" = "marco"
       Key: apellido
       Value: marco
       Form item: "email" = "marco@marco.com"
       Key: email
       Value: marco@marco.com
       Form item: "sexo" = "masc"
       Key: sexo
       Value: masc
```

Respuesta del servidor, puede verse el archivo HTML “response” con los datos enviados anteriormente

```
Frame 279: 3254 bytes on wire (26032 bits), 3254 bytes captured (26032 bits) on
interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 8080, Dst Port: 42916, Seq: 158, Ack:
461, Len: 3186
Hypertext Transfer Protocol
       HTTP/1.0 200 OK\r\n
       [Expert Info (Chat/Sequence): HTTP/1.0 200 OK\r\n]
```

```
            [HTTP/1.0 200 OK\r\n]
            [Severity level: Chat]
            [Group: Sequence]
        Request Version: HTTP/1.0
        Status Code: 200
        Response Phrase: OK
        Content-Type: text/html; charset=utf-8\r\n
        Content-Length: 3186\r\n
        [Content length: 3186]
        Server: Werkzeug/0.12.2 Python/2.7.12\r\n
        Date: Tue, 05 Sep 2017 22:23:33 GMT\r\n
        \r\n
        [HTTP response 1/1]
        [Time since request: 0.002056807 seconds]
        [Request in frame: 265]
        File Data: 3186 bytes
Line-based text data: text/html
        <html>\n
            [... código de response.html ...]
        \t\t\t <h1>\302\241Bienvenido a TP2!</h1>     \n
        \t\t\t</div>        \t\n
        \n
        \t\t\t<ul class="list-group">\n
        \t\t\t <li class="list-group-item"><b>Nombre:</b> marco </li>\n
        \t\t\t <li class="list-group-item"><b>Apellido:</b> marco </li>\n
        \t\t\t <li class="list-group-item"><b>Email:</b>  marco@marco.com </li>\n
        \t\t\t <li class="list-group-item"><b>Sexo:</b>  Masculino </li>\n
        \t\t\t</ul>         \t\n
        \n
            </div>        \n
        \n
        </body>\n
        </html>
```

Enunciado 3)

Generar un proyecto de simulación de acceso a valores de temperatura, humedad, presión atmosférica y velocidad del viento.

a) Un proceso simulará una placa con microcontrolador y sus correspondientes sensor/es o directamente una estación meteorológica proveyendo los valores almacenados en un archivo o en una base de datos. Los valores se generan periódicamente (frecuencia de muestreo).

b) Un proceso generará un documento HTML conteniendo:

   i) Frecuencia de muestreo
   ii) Promedio de las últimas 10 muestras
   iii) La última muestra

c) El documento HTML generado debe ser accesible y responsivo.

Aclaración: Se deberá detallar todo el proceso de adquisición de datos, cómo se ejecutan ambos procesos (ya sea threads o procesos separados), el esquema general, las decisiones tomadas en el desarrollo de cada proceso y la interacción del usuario.

Resolución ejercicio 3:

Los archivos de la resolución de este ejercicio se encuentran en el repositorio GitHub del proyecto.

Se crearon dos aplicaciones **sim.py** y **app.py**, de las cuales la primera actúa como un microcontrolador que obtiene los valores correspondientes a la temperatura, humedad, presión atmosférica y velocidad del viento, y la segunda aplicación tomará las muestras y las exhibirá en una página web. Ambas se comunican almacenando y retirando la información de un archivo de texto.

La Fig. 2. muestra como el microcontrolador añade las nuevas mediciones a un archivo, el cual posteriormente será leído por la aplicación que presentará en la página web la información.

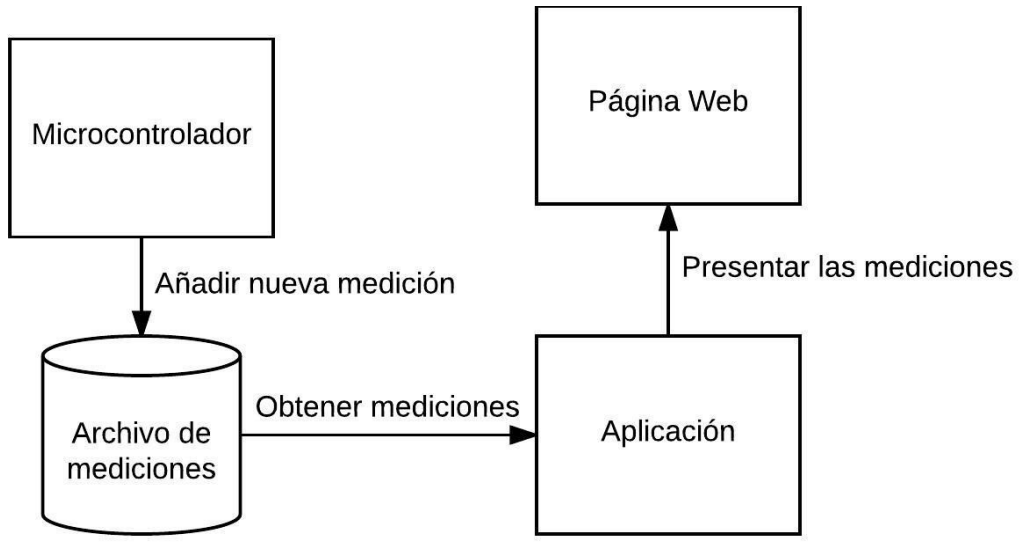

Fig. 2. Diagrama de arquitectura de software del programa realizado en el ejercicio 3.

En el simulador, los datos se generan mediante un while infinito que introduce números aleatorios en el archivo de texto y espera una cantidad de segundos predeterminada mediante la secuencia *sleep()*.

Enunciado 4)

Agregar a la simulación anterior la posibilidad de que el usuario elija entre un conjunto predefinido de períodos de muestreo (ej: 1, 2, 5, 10, 30, 60 segundos). Identifique los cambios a nivel de HTML, de HTTP y de la simulación misma.

Resolución ejercicio 4:

Al inciso anterior se le añadió un formulario en el cual se deberán ingresar valores entre 1 y 60. Este campo modifica la frecuencia de obtención de datos por parte de los sensores y de actualización de la interfaz web.

La Fig. 3. permite observar que, luego de los cambios realizados al inciso anterior, la página web puede indicarle a la aplicación cual es la nueva frecuencia, y esta la almacenará en un archivo para que el microcontrolador pueda obtener la nueva frecuencia de muestreo.

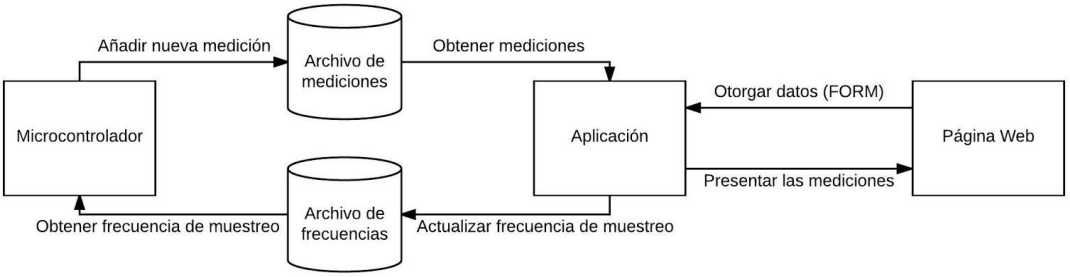

Fig. 3. Diagrama de arquitectura de software del programa realizado en el ejercicio 4.

Enunciado 5)

Comente la problemática de la concurrencia de la simulación y específicamente al agregar la posibilidad de cambiar el período de muestreo. Comente lo que estima que podría suceder en el ambiente real ¿Podrían producirse problemas de concurrencia muy difíciles o imposibles de simular? Comente brevemente los posibles problemas de tiempo real que podrían producirse en general.

Resolución ejercicio 5:

En esencia, el problema del productor-consumidor es un ejemplo clásico de problema de sincronización de multiprocesos. El programa describe dos procesos, productor y consumidor, ambos comparten un buffer de tamaño finito. La tarea del productor es generar un producto, almacenarlo y comenzar nuevamente; mientras que el consumidor toma (simultáneamente) productos uno a uno. El problema consiste en que el productor no añada más productos que la capacidad del buffer y que el consumidor no intente tomar un producto si el buffer está vacío.

En este caso, los problemas de concurrencia son similares. Uno de los problemas es que el consumidor intente obtener un dato antes de que se haya generado uno nuevo, y el otro, que el productor produzca un dato luego de que el consumidor haya tomado uno.

Para solucionar estos problemas se utilizan mecanismos para sincronizar los procesos.

Enunciado 6)

¿Qué diferencias supone que habrá entre la simulación planteada y el sistema real? Es importante para planificar un conjunto de experimentos que sean significativos a la hora de incluir los elementos reales del sistema completo.

Resolución ejercicio 6:

Existen demasiadas variables que tienen que ser contempladas en un sistema real, tales como alimentación de los dispositivos, medios de comunicación, problemas de ruido, interferencia y distorsión, delay de la comunicación, latencia, etc. La simulación planteada es una aproximación al sistema real pero considerando que esté fuera perfecto. A la hora de crear un sistema completo real se deben considerar soluciones a los distintos tipos de problemas que puedan afectar su ejecución. Deben planificarse experimentos para comprobar que el sistema responderá correctamente antes cualquier problema, ya que si se plantea, por ejemplo, un delay en la comunicación, se deberá esperar que los datos estén listos antes de que sean leídos.