TRABAJO PRÁCTICO N°1
Taller de Proyecto II

Integrantes

Monti, Agustín 123/9

Lanciotti Julieta 1107/1

Antes de la realización del trabajo se llevaron a cabo los siguientes pasos para la instalación de Python y el microframework a utilizar, Flask.

Primer paso: instalar Python 2.7 en Windows 8.1

- Descargar Python 2.7.13 https://www.python.org/ftp/python/2.7.13/python-2.7.13.msi
    - Seleccionar opción PIP durante la instalación
- Configurar Python para que pip funcione:
    - Dirigirse a: C://Python27
    - Copiar la URL del archivo python27.exe
    - Dirigirse a Mi Pc – Propiedades – Configuración avanzada del sistema – Variables de Entorno – editar variable del sistema PATH – Nueva – Agregar: C:\Python27 y C:\Python27\Scripts

Segundo Paso: Instalar Flask. Existen dos maneras, ejecutando el archivo requirements.txt o con pip. (pip install Flask).

EJERCICIO 1

Se genera un archivo de texto llamado requirements.txt:

- Contenido del documento: Flask==0.12.2 (la versión no es necesaria).
- Para ejecutar estas dependencias se debe abrir la consola CMD (caso Windows).
- Ejecutar: `pip install -r /path/to/requirements.txt`

De esta manera, a la hora de ejecutar el comando, comenzarán a descargarse todas las dependencias escritas en el archivo .txt. Una gran ventaja de esto es que se evita el instalar dependencia por dependencia. En el caso de un gran programa, se ahorrará tiempo, sino se debería proceder a instalarlas con pip.

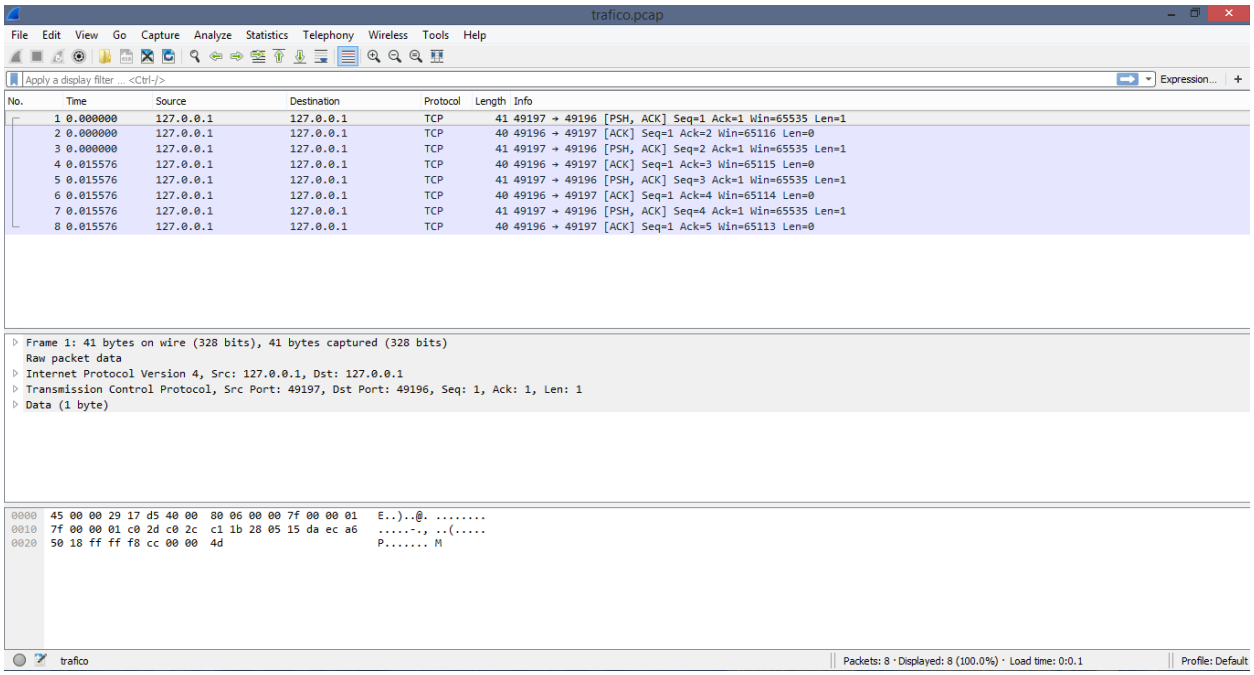
EJERCICIO 2

Para este ejercicio se utilizaron los siguientes programas:

- RawCap: permite capturar el tráfico de datos de una red y guardarlos en una captura en disco.
- WireShark: es un analizador de protocolos utilizado para realizar análisis y solucionar problemas en redes de comunicaciones, para desarrollo de software y protocolos, y como una herramienta didáctica. Permite examinar datos de una red o de un archivo de captura salvado en disco.

Como primera medida se generó un programa .py que interactúe con la pagina web para enviar datos. En este caso, se utilizó lo planteado en el ejercicio 3 como ejemplo.

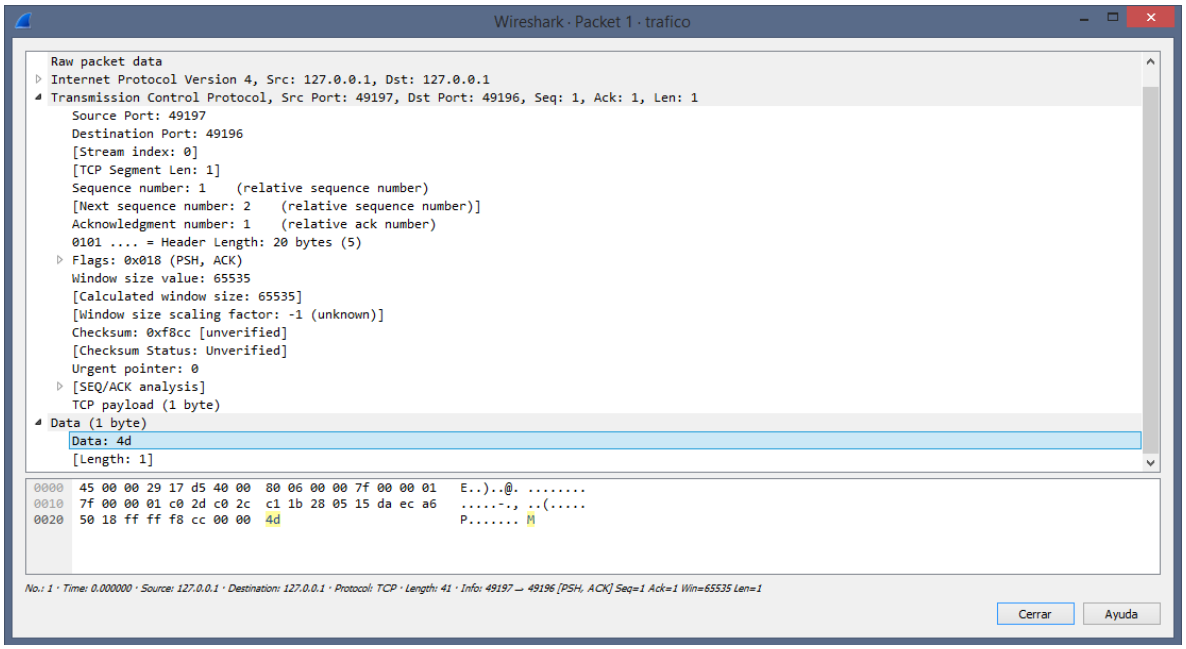
- Abrir CMD (en Windows).
- Dirigirse al directorio donde se encuentra RawCap.exe
- `RawCap.exe ip imagen.pcap`
    - IP: la dirección (sin el puerto) donde está la página web que se quiere analizar
    - imagen.pcap: nombre de la imagen que será analizada con wireshark que tendrá el tráfico de paquetes.
- Abrir el archivo .pcap generado con Wireshark.



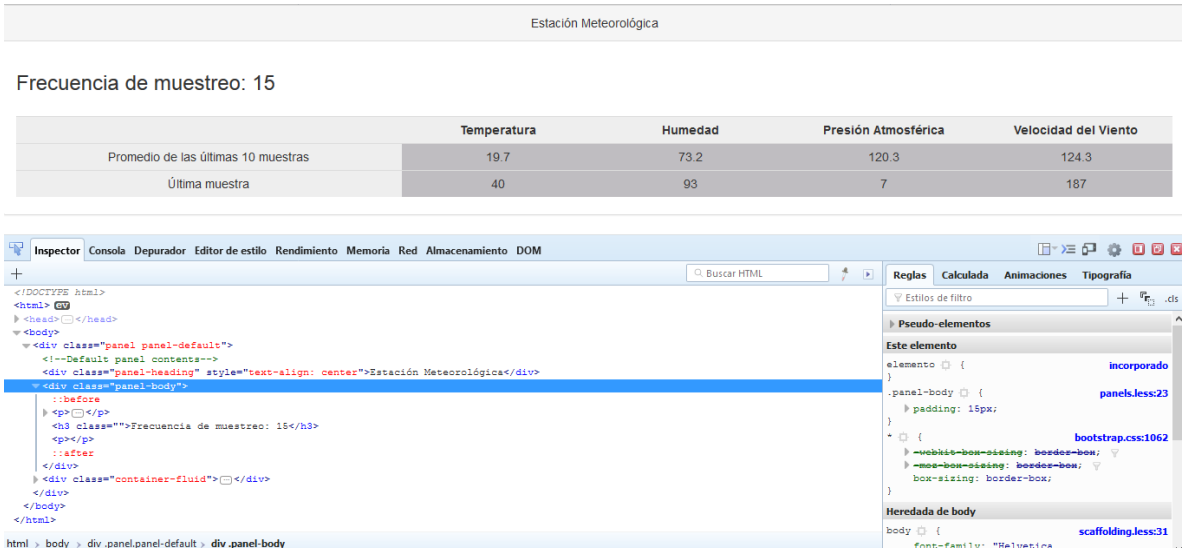
La imagen muestra el tráfico de paquetes, capturada por RawCap, que se produjo cuando se envió información a la página y hasta que llegó a esta para que se muestre al usuario.

La siguiente imagen muestra a detalle lo que realmente se envia en los paquetes, entre ellos:

- Protocolo
- Tiempo requerido
- Tamaño del paquete
- Datos enviados
- Etc.



Es decir, se puede notar como no solamente llegan datos HTML, o código con JavaScript a la página. También llegan mucha más cantidad de información que no se puede ver Inspeccionando la página, como por ejemplos los POST enviados.



Datos mostrados inspeccionando la pagina.

EJERCICIO 3

Para la simulación de una Estación Meteorológica, se realizaron dos procesos.

- *Generador.py*: este proceso se encarga de la simulación de la estación y provee valores random de temperatura, humedad, presión atmosférica y velocidad del viento.
    - El proceso guardará esos valores en un archivo llamado datos.txt, los cuales se generarán cada 1 segundo en formato temperatura/humedad/presión/velocidad (Ejemplo: 50/30/1100/20).
- *ej3.py*: este proceso se encargará de levantar el servidor utilizando Flask y de leer los datos generados en el archivo datos.txt. En el index de la página web se mostrarán en una tabla el promedio de las últimas 10 muestras y la última muestra. Además de la frecuencia de muestreo.
    - Respecto a la frecuencia de muestreo, se la consideró a esta como “el tiempo que debe pasar hasta que se lean los datos de la estación”, con un valor ya preestablecido.

Explicación sobre el proceso ej3.py

Para lograr que se muestren los datos recolectados por la Estación Meteorológica, se realizó un proceso el cual lee los datos del archivo. Dicho proceso lee todo el archivo en un cierto instante y guarda en listas los valores de los respectivos datos. De esta manera, llamando a otro proceso, se calculó el promedio de las últimas 10 muestras leídas.
En este caso, cada cierto tiempo (frecuencia de muestreo) se realizará lo indicado anteriormente y se actualizará automáticamente la página con los datos leídos.

Los dos procesos, generador.py y ej3.py, se ejecutan concurrentemente; es decir, mientras se estén generando nuevos datos en el archivo de texto, el programa ej3.py leerá cada cierto tiempo ese archivo. Esto provoca ciertos inconvenientes que serán explicados en el ejercicio 5.

Cambios realizados:

- En un principio se pensaron los procesos de forma secuencial, lo que no iba a permitir simular el funcionamiento correcto de la Estación Meteorológica. Por lo tanto, se creó en un script aparte (Generador.py) y ej3.py que se encarga de leer los datos y levantar el servidor.
- Se le agrego Bootstrap para mejorar la visualización de la pagina. Para ello se opto por la versión precompilada de la misma. Su instalación se lleva a cabo descargandolo desde la pagina oficial, descomprimiendo el archivo y ubicandolo en el directorio del proyecto. Finalmente se los debe incluir dentro de las vistas entre los tags <script></script> y <link></link>

EJERCICIO 4

A diferencia con el ejercicio 3, en este caso el usuario debe ingresar la frecuencia de muestreo. Para esto se realizó un Form.

Se modificó el archivo ej3.py y se consideró por defecto una frecuencia de muestreo de 15 segundos para que comience a actualizar los datos hasta que el usuario ingrese otro tiempo.
Cuando se envía la nueva frecuencia, la página actualizará el valor a mostrar y leerá los datos del sensor cuando pase el tiempo ingresado.
A comparación con el ejercicio 3, en este hubo cambios a nivel HTML y HTTP. Cuando se realiza un formulario (Form - forma de interactuar el usuario con la página), se debe especificar la manera en que se van a enviar y recibir esos datos, por lo que se realizó a nivel HTTP con GET. Con la ayuda de la implementación de Bootstrap, se hizo que la página sea responsive y accesible al usuario. Se utilizo en particular el tipo container-fluid para que la vista se adapte al tamaño de pantalla en donde se visualice.

Cosas a destacar en general

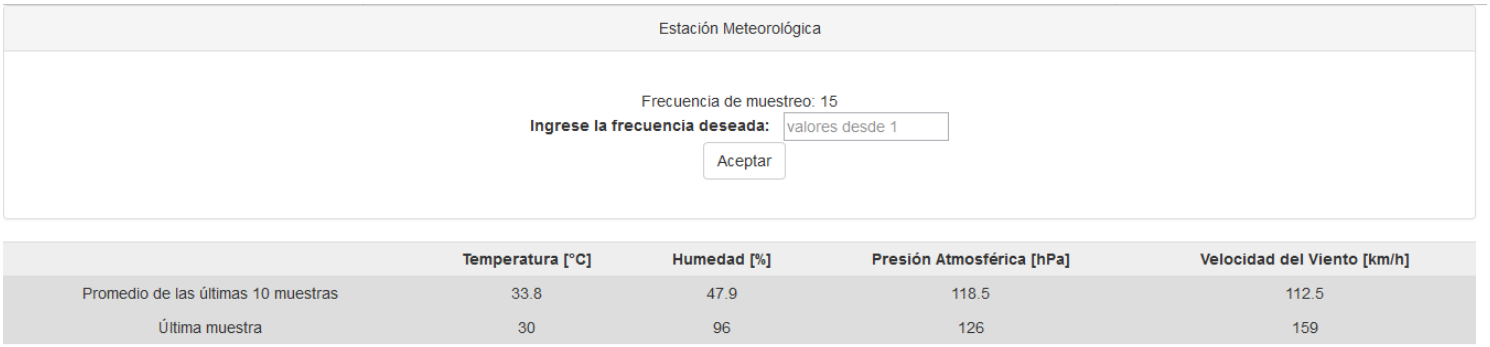
- Para que la página se actualice sola cada cierta frecuencia de muestreo, se utilizó JavaScript.

```
<script type="text/javascript"> setTimeout(function
() {window.location.reload(1);}, {{frec
}}*1000);</script>
```

  - El siguiente codigo, ubicado en la vista, recibe como parámetro la frecuencia de muestreo y la multiplica por 1000 para pasarlo a segundos.

  - *Inconvenientes:* en primera instancia, a nivel HTTP, se realizó el form con POST. El problema fue que al actualizar la página, esta solicitaba reenviar los datos (la frecuencia ingresada por el usuario), por lo que se recurrió a cambiar a GET, ya que además, no guardaba la frecuencia de forma correcta.

- En el ejercicio 3 la frecuencia de muestreo ya está preestablecida, y es mayor a la frecuencia de generación de datos (5 segundos).
- En caso de que ocurra que el archivo datos.txt no tenga ningun dato aún y se requiera la lectura, la página mostrará un mensaje de advertencia.



EJERCICIO 5

Los principales problemas de concurrencia surgen cuando se realizan operaciones de escritura y lectura en el archivo datos.txt. Debido a que no se implementó una solución al problema del tipo productor-consumidor, puede darse el caso que nuestro generador de datos (sensor-productor) no haya producido datos y que el consumidor (estación meteorológica) intente leer dicho archivo.

Otros problemas de concurrencia que pueden darse es la interacción con el usuario con el sistema de sensado. Entre que el usuario ingresa la frecuencia y esta se actualiza, hay muestras que se perderán. Depende de la situación real si esta situación es válida o no. Para nuestro caso, el proceso Generador.py genera muestras cada 1 segundo, si la estación meteorológica consume antes, se le dara aviso al usuario produciéndose el error.

Otra problemática que no se tuvo en cuenta es el no uso de un sistema operativo de tiempo real (RTOS), suponiendo que los periodos en que toman las muestras los sensores no son críticas. Caso contrario habría que optar por alguna de las opciones que encontramos en RTOS.

EJERCICIO 6

- En sistemas de tiempo real los sensores trabajan bajo algún protocolo (si son muy economicos puede ser uno no estándar). En nuestro caso genera las muestras y automáticamente guarda en un archivo compartido.
- En el sistema real existirá mayor precisión pero con cierto rango de error en la toma de valores.
- Los sensores utilizados en una Estación Meteorológica presentan cierto tiempo de vida y su funcionamiento puede ser cada vez peor, generando errores a largo plazo.
- En un sistema real de este tipo, cuando se toman valores con sensores, es necesario que exista cierto delay entre la toma de estos ya que los sensores no funcionan de forma inmediata.