Trabajo Práctico 1

Taller de Proyecto 2 (I118)

García, Agustín (824/0)
Sanchez, Agustín (939/2)
Ternouski, Nahuel (756/6)

La Plata, Septiembre de 2017

Índice

El repositorio se encontrará alojado en github (véase link). Y se puede ver en ejecución en: `https://tpii-tp1.herokuapp.com`.

1. Ejercicio 1

El archivo se puede ver a través del link: requirements.txt. Luego de instalar todas las dependencias y definir la ruta «root» en `app.py`, se ejecuta en consola el siguiente comando:

```
FLASK_APP=app.py flask run
```

Hecho esto aplicación se estará ejecutando en la URL: `http://127.0.0.1:5000`.

Cada vez que un cliente realiza una petición a un servidor, se ejecutan los siguientes pasos (se tiene como ejemplo la aplicación de esta práctica):

- Un usuario accede a una URL, en ese caso localhost:5000.
- El navegador descodifica la URL, separando sus diferentes partes. Así identifica el protocolo de acceso, la dirección DNS o IP del servidor, el posible puerto opcional (en ese caso 5000) y el objeto requerido del servidor.
- Se abre una conexión TCP/IP con el servidor, llamando al puerto TCP correspondiente.
- Se realiza la petición. Para ello, se envía el comando necesario (en ese caso GET), la dirección del objeto requerido (el contenido de la URL que sigue a la dirección del servidor), la versión del protocolo HTTP empleada (casi siempre HTTP/1.0) y un conjunto variable de información, que incluye datos sobre las capacidades del navegador, datos opcionales para el servidor, etc.
- El servidor devuelve la respuesta al cliente. Consiste en un código de estado y el tipo de dato MIME de la información de retorno, seguido de la propia información (en este caso el código HTML de las tablas con los datos meteorológicos).
- Se cierra la conexión TCP.

2. Ejercicio 2

Para observar la comunicación entre el servidor y el cliente se utiliza la herramienta llamada «Wireshark» que facilita la lectura de mensajes HTTP.

HTTP es un protocolo orientado a transacciones que sigue el esquema petición-respuesta entre un cliente y un servidor. El cliente realiza una petición enviando un mensaje, con cierto formato al servidor. El servidor le envía un mensaje de respuesta.

Se utilizó el programa del ejercicio 3 para analizar la información que se intercambia entre el usuario y el sistema.

En primer lugar, se analizó el «REQUEST» que se genera al ingresar a la raíz del sitio desde un navegador. Desde el programa mencionado se capturó el mensaje de la figura 1.

Como se puede observar en la figura, el método de la petición, es de tipo «GET», lo que implica la solicitud de un recurso. El mismo se especifica en la URI, que en este caso es la raíz de la página. También se define la versión del protocolo HTTP que soporta el navegador, en este caso 1.1 y por último el tipo de contenido que espera recibir, que es HTML.

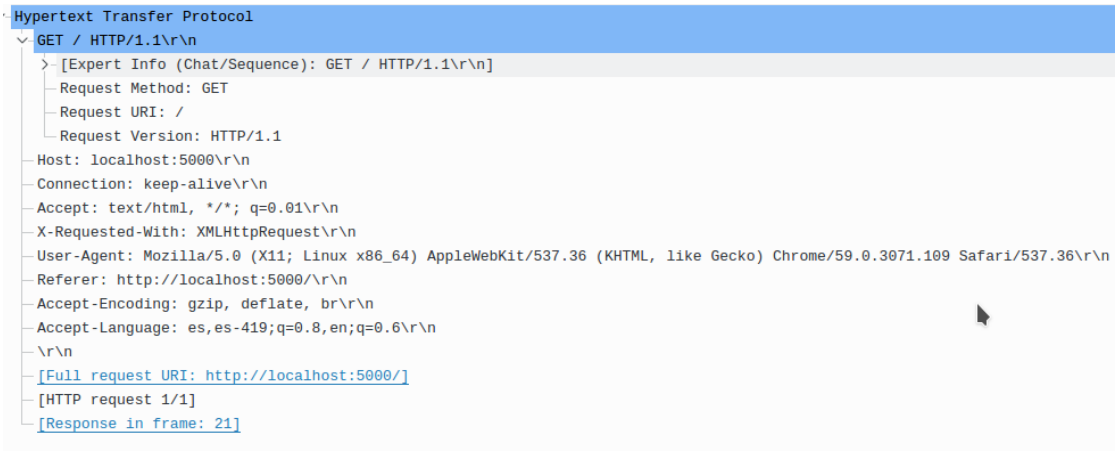

Figura 1: Mensaje capturado de tipo REQUEST.

Una vez recibido el mensaje de «REQUEST» por el servidor web, se atiende la petición generando una respuesta con el recurso solicitado. Esto se puede observar en el mensaje capturado de la figura 2.

En este tipo de mensajes se observa un código de respuesta que indica el resultado de la solicitud, en este caso la misma esta representada por el código 200 que significa que la solicitud fue resulta exitosamente. Además, se especifica la longitud en bytes de la información transmitida. Y por último, en el campo de datos se encuentra el HTML completo del sitio web.

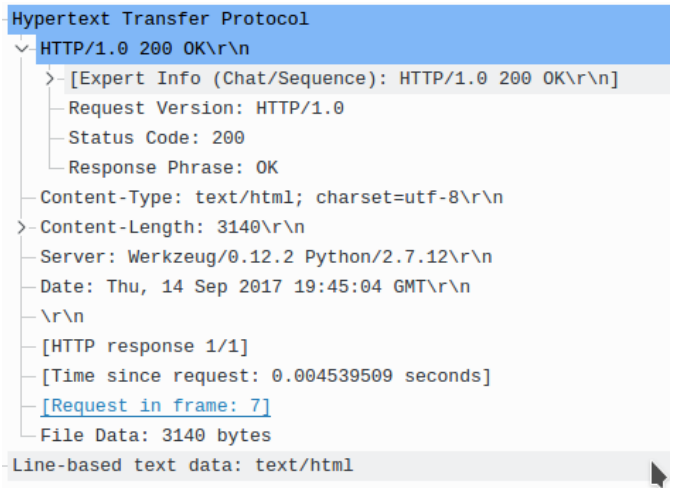

Figura 2: Mensaje capturado de tipo RESPONSE.

3. Ejercicio 3

La aplicación tiene treienta s procesos corriendo en paralelo, una es la que contiene a el servicio web (que se encuentra en el archivo `app.py`) y otra es la simulación del microcontrolador con sus sensores (que se encuentra en el archivo `sensors.py`) que se divide en dos: el generador de datos y el encargado de limpiar los datos para que no se acumule datos innecesariamente.

El proceso que atiene los servicios web, administra dos tipos de solicitudes (GET y POST) dentro de la ruta raíz de la aplicación. En primer lugar el método GET es iniciado cuando el cliente ingresa por primera vez al sitio, lo cual el servidor responde con la información solicitada, junto con el parámetro por defecto: tiempo de muestreo un segundo. El método POST se genera cuando el cliente quiere cambiar el tiempo de muestreo de los datos, esto activa la rutina para modificar el tiempo de muestreo del sensor y refresca la página web con los datos nuevos.

Por otro lado se tiene un modulo que simula el sensado de los datos, el cual define las funciones: `start`, `change_sampling`, `cleaner_run`, `sensors_run` y `stop`.

La función `start` realiza las primeras tareas de la simulación del sensor, para que el servicio de los datos inicie en un estado conocido, esto es: eliminar los archivos que estén creados previamente, inicialización de las variables a utilizar y por ultimo inicia los dos procesos que van a ejecutarse (`sensors_thread` y `cleaner`).

La función `sensors_run` ejecuta uno de los dos procesos del sensor que se encarga de generar los datos aleatoriamente de forma periódica, teniendo en cuenta los problemas de concurrencia que pueden surgir con la memoria compartida por otros procesos. Para ello, se realizo un mutex controlando el acceso al archivo y una variable `muestreolock` para el acceso a la variable que almacena el muestreo. Cuando se supera un cierto limite de datos almacenados el mismo proceso despierta a el `cleaner` para que reduzca el tamaño del archivo. Los datos ingresados al sistema son: temperatura, humedad, presión y velocidad del viento. Estos son guardados en un archivo con un formato de separación con el carácter pipe «|».

La función `cleaner_run` es la rutina del proceso limpiador, que como se mencionó anteriormente es el encargado de minimizar los datos almacenados en el sistema. En esta función se leen los últimos datos sensados y se sobre escriben con los viejos, de esta forma solo se dejan los datos útiles para el sistema.

La función `change_sampling` detiene momentáneamente la ejecución de ambos procesos, para luego reactivarlos con la nueva frecuencia de muestreo.

Por último, la función `stop` detiene la ejecución normal de los procesos involucrados, es decir, no los para completamente.

Todos los datos obtenidos del sensor son mostrados al cliente de forma resumida como tabla, de tal manera de que sepa la muestra actual y un promedio de las últimas dos para cada una de las variables.

En este ejercicio no se contempla ninguna interacción del usuario con el sistema, es decir, únicamente se muestran los datos.

Inciso C Se utiliza un framework llamado bootstrap con la versión 4 beta, que dentro de las funcionalidades que presenta, se incluye la adaptabilidad entre distintos tamaños de pantalla, es decir, «responsive». De esa forma utilizando clases (como: `row`,`col-md-X`, `col-sm-X`, etc) ya establecidas dentro de CSS y JavaScript por el framework se personaliza las etiquetas dentro de HTML.

4. Ejercicio 4

A nivel HTML se agrego un formulario (ver fragmento 1), en el cual se pueden elegir las destinas opciones disponibles en segundos (1, 2, 5, 10, 30, 60). El código python embebido en el formulario es para que se mentare en la etiqueta *selec* el valor que el usuario había seleccionado previamente.

```
1  <form action="/" method="POST">
2    <div class="form-group">
3      <div class="row justify-content-center">
4        ...
5        <div class="col-md-6 col-lg-4" style="padding-bottom: 10px">
6          <select class="form-control" name="muestreo">
7            {\% for j in ["1", "2", "5", "10", "30", "60"] \%}
8              {\% if muestreo == j \%}
9                <option selected="selected" value="{{j}}">{{j}} seg</option
                     ↪ >
10             {\% else \%}
11               <option value="{{j}}">{{j}} seg</option>
12             {\% endif \%}
13           {\% endfor \%}
14         </select>
15       </div>
16           <div class="col-md-auto align-middle text-center">
17             <button type="submit" class="btn btn-primary">
18               Actualizar tiempo
19             </button>
20           </div>
21     </div>
22   </div>
23 </form>
```

Fragmento 1: Ejemplo de un formulario HTML5.

Por el lado de HTTP se debió incorporar el método POST para administrar el pedido del cliente y de esta forma actualizar con el simulador del microcontrolador.

A nivel simulación se tiene una variable en el servidor web que almacena la frecuencia de muestreo actual y que se actualiza según la solicitud de «POST» realizado por el usuario. Además, se le avisa al `sensors.py` que actualice la frecuencia que sensa los datos.

5. Ejercicio 5

El principal problema de concurrencia que se advierte, es el del Lector-Escritor y se relaciona con el hecho de que dos tareas comparten un mismo recurso. Por un lado se tiene un proceso, «deamon», que periódicamente escribe sobre un archivo las muestras sensadas. Mientras que, el proceso que administra el web server, lee asincrónicamente la información almacenada para mostrarla cuando el usuario lo requiera. Puede ocurrir que el cliente solicite la información en el mismo instante en que el proceso deamon se encuentre guardando los datos sensados. Esto será más probable a medida que se aumente la frecuencia de muestreo. Los problemas mencionados podrían resolverse utilizando una base de datos que ya tiene provisto el acceso concurrente a la

misma. Sin embargo, se optó por utilizar un lock por software de la siguiente manera. Cada vez que se va a utilizar el recurso compartido se crear un archivo denominado "lockz cuando se deja de utilizar, se elimina. De esa forma, cada tarea, antes de usar el recurso, debe chequear que el archivo lock no exista.

Por otra parte, la frecuencia de muestreo podría estar almacenada en una variable compartida entre el proceso sensor y el server. Como toda variable compartida, debe tener su precaución a la hora del acceso y escritura de la misma, por ejemplo, utilizando locks. En el ambiente real, es probable que esto no suceda, por lo que el microcontrolador debería prever la recepción asincrónica de mensajes que indiquen que debe cambiar su frecuencia de muestreo.

6. Ejercicio 6

Es probable que el sistema real tenga tiempos de respuesta distintos a los de la simulación, y que la precisión de los datos provistos sea distinta en función de los tipos de sensores que se utilicen. Adicionalmente dependiendo del contexto en el que vaya a utilizarse el sistema real, este podría correr riesgos ambientales como los de un ambiente industrial o de clima extremo. Dichos riesgos no pueden ponerse a prueba en la simulación.

Por otra parte cabe destacar que los diferentes datos que están involucrados en el sistema real son transmitidos desde distintos lugares por lo tanto se incorpora una problemática adicional que es la transmisión y recepción de los datos a través de una red con acceso a internet. Hecho que en la simulación no hacía falta el proveedor ISP.

Además, las recursos compartidas presentes en la simulación, en la realidad no existen. Por lo que se debe reemplazar el archivo de los datos sensados por una base de datos hosteada en algún lugar y las variables de configuración que permitían establecer y mostrar la frecuencia de muestreo desaparece. Por lo que es necesario un mecanismo que permita leer y modificar esas variables asincrónicamente.