CÁTEDRA DE SISTEMAS OPERATIVOS II

Departamento de Computación

FCEFyN - UNC

Trabajo Práctico 3 – Sistemas Embebidos

Maero Facundo
Mat: 38479441

Córdoba, Mayo de 2017

Índice

1. Introducción

En el presente Informe se mostrará el proceso de diseño, desarrollo, testing y documentación de un software que mediante el uso de la interfaz CGI y un webserver liviano instalado en una placa embebida, permita conectarse a la misma, acceder a información del sistema, controlar y descargar archivos relacionados al monitoreo de estaciones hidrometeorológicas, y gestionar los módulos del Kernel en la placa.

1.1 Propósito.

El propósito de este Trabajo Práctico es aprender a utilizar la interfaz CGI, de la mano de scripts escritos en Perl, o programas compilados en C, que permiten la comunicación entre el cliente y el servidor web, de manera que este último ejecute los scripts que permiten procesar archivos, para luego mostrar los resultados nuevamente en la página web.

1.2 Ámbito del Sistema.

El sistema se conforma de dos partes: el webserver, que se ejecuta en una placa Intel Galileo o Raspberry PI, el cliente, que puede tomar lugar en la computadora del usuario, un celular, tablet, o cualquier dispositivo con acceso a la red.
El cliente se comunica con el server mediante la página web, e interactúa con él mediante la interfaz gráfica brindada.

1.3 Definiciones, Acrónimos y Abreviaturas

- CGI: Common Gateway Interface
- Webserver: servidor web.

1.4 Descripción General del Documento

El presente documento muestra una visión del proceso desarrollo del trabajo práctico, su diseño, implementación, esquemas a seguir y conclusiones sacadas del mismo.

2. Descripción General

2.1 Perspectiva del Producto

El producto solicitado es un sistema que gestione la conexión con una plataforma servidor, para acceder a los datos telemétricos de diferentes estaciones sensoras.
Las estaciones proporcionan sus datos al servidor, que los recopila en un archivo separado por comas (.CSV), el cual debe ser accedido e interpretado.

2.2. Funciones del Producto

Las funciones que proporciona el producto son:

- Conexión entre cliente y servidor mediante un navegador web.
- Acceso a los datos de las estaciones mediante una interfaz web que brinda información sobre los sensores y las estaciones.
- Acceso a información del sistema que ejecuta el webserver.
- Instalación y eliminación de módulos del Kernel en el webserver.

2.3. Características de los Usuarios

El usuario del programa solo necesita conocer la dirección IP del servidor web para poder utilizarlo. Luego, todas las opciones disponibles se presentan en pantalla con una interfaz simple y fácil de utilizar.

2.4. Suposiciones y Dependencias

- Se supone que el servidor posee un archivo con la base de datos, llamado “datos_meteorologicos.CSV”, de donde se leerán los resultados de las mediciones.
- El proyecto se ejecutará en un embebido con entorno Linux.
- Existe conectividad entre el servidor y el cliente, para poder realizar la comunicación.

3. Requisitos Específicos

3.1. Interfaces Externas

Para interactuar con el programa, se utiliza el navegador web, y la página que muestra el mismo. Por este medio se muestran resultados al cliente, y se introducen comandos dirigidos al servidor, quien muestra nuevamente por la página los resultados de la instrucción recibida.

3.2. Funciones

El servidor permite realizar las siguientes funciones:

- **Mostrar información del sistema:** tal como modelo de CPU, estado de la memoria RAM, versión de Kernel instalada, nombre de usuario, hora y fecha actual, uptime, entre otros.
- **Mostrar información sobre estaciones:** permite ver el promedio de las mediciones para los sensores instalados, ver las precipitaciones diarias y mensuales de una estación, y descargar la información recaudada de cada una.
- **Drivers y módulos:** permite ver una lista con los módulos del Kernel instalados actualmente, y apartados como su tamaño y estado. Además permite subir un módulo propio, instalarlo y eliminarlo.
- **Documentación:** permite ver la documentación del código fuente del proyecto.

3.3. Requisitos de Rendimiento

El proyecto debe funcionar en una placa embebida, por lo que es necesario el uso de un webserver liviano, y no uno con características que no van a usarse, como Apache. Por este motivo se eligió Lighttpd. Además, el uso de la interfaz CGI implica que el procesamiento se realiza en el servidor, lo que reafirma la necesidad de utilizar una aplicación lo más liviana posible.

3.4. Restricciones de Diseño

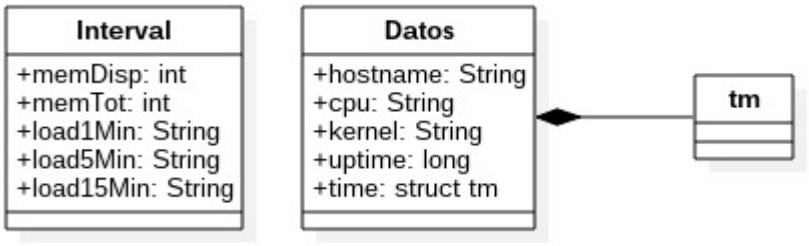
- No debe implementarse **connect** y **disconnect** como en el Práctico 1, ya que no se están utilizando Sockets.
- Al subir el archivo de tipo módulo, debe chequearse que efectivamente sea de este tipo, e informar de error en caso de uso incorrecto.
- Todos los procesos deben ser mono-thread.
- No deben utilizarse IDEs para el desarrollo, prefiriendo el uso de editores de texto y archivos Makefile para la compilación y linkeo de archivos.
- Para la compilación es recomendable utilizar flags como -Werror, -Wall y -pedantic.

4. Diseño de solución

Para diseñar el software que cumpla estos requisitos, se utilizaron como base los códigos desarrollados por el alumno en el Trabajo Práctico 1, para mostrar información sobre las estaciones meteorológicas, y también código desarrollado en la materia Sistemas Operativos I, en el práctico sobre la estructura /proc de Linux, para obtener información del sistema.
A partir de esta base se llegó al siguiente diseño:

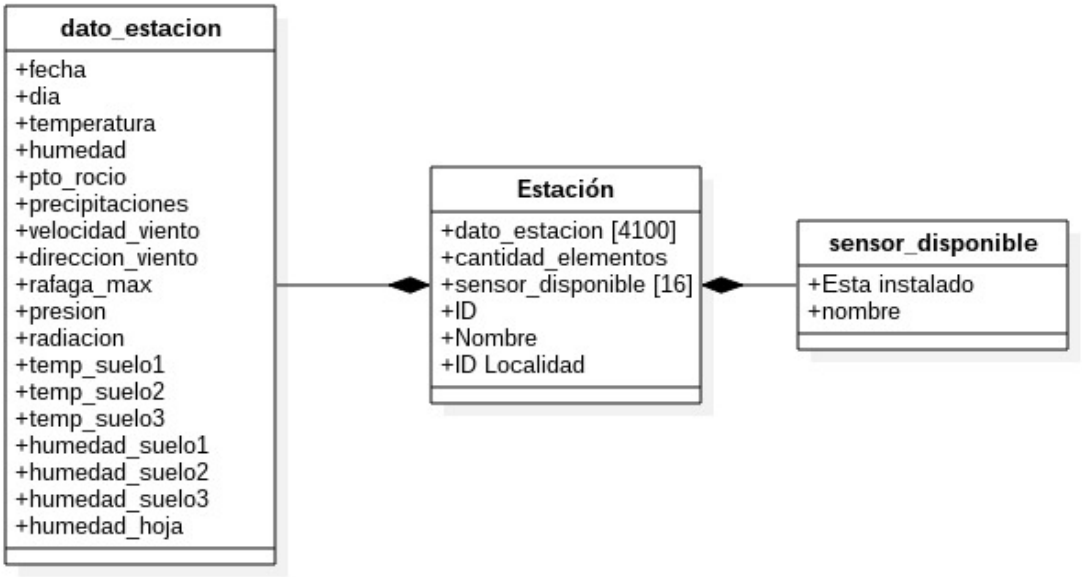
1. Para el apartado de información del sistema, se modificó la estructura general del programa **ksamp**, que obtiene información de archivos en /proc. Estos datos se almacenan en dos estructuras: **Datos** almacena información general sobre el sistema, como modelo de CPU y uptime. **Interval** almacena información más dinámica, como la memoria disponible, o el uso de la CPU.

   Al solicitar el cliente ver esta información, se abren los archivos necesarios en /proc y se parsea la información necesaria. Luego se muestra por el navegador una página web con el contenido de las estructuras descritas.



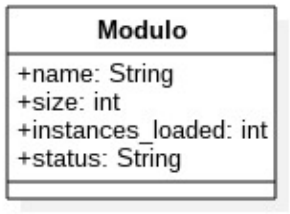
2. Para el apartado de información sobre las estaciones, también se modificó el código fuente del primer Trabajo Práctico de la materia. Se conservaron las estructuras de datos utilizadas, y los métodos de acceso al archivo .CSV.

   Nuevamente, si el cliente solicita acceder a alguna información de este apartado, se lee el archivo con los datos, se guardan y se envía de nuevo al navegador del cliente la página web con lo que solicitó.

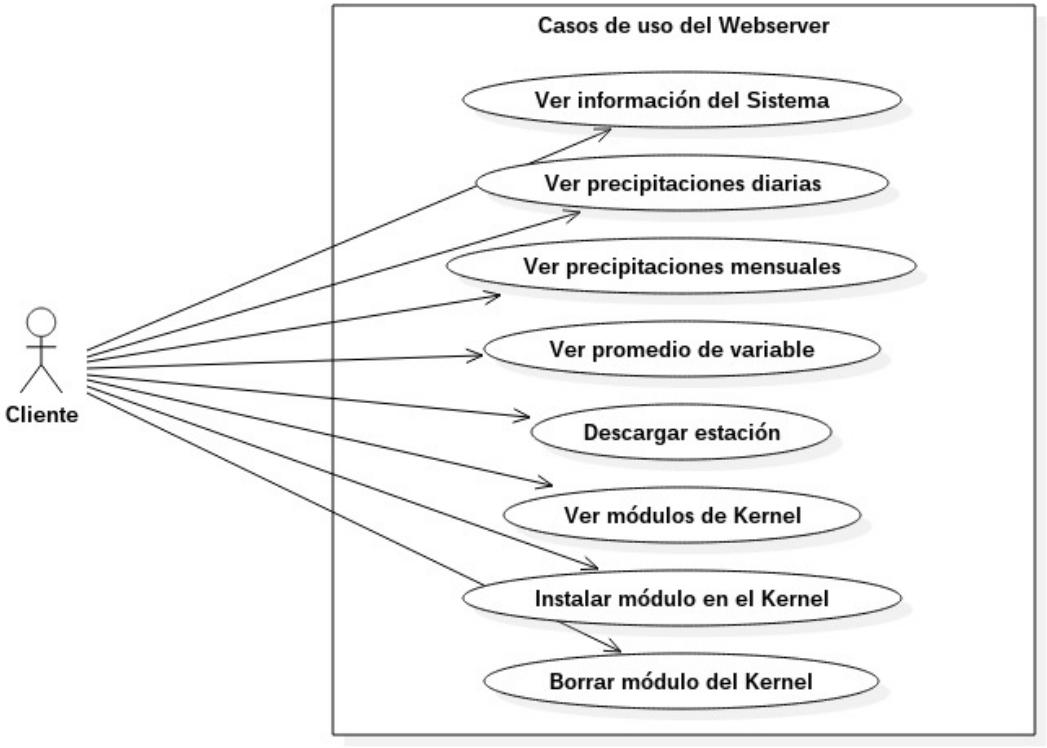


3. Para el apartado que involucra los módulos de Kernel instalados, se hizo uso del archivo /proc/modules, que muestra información sobre los mismos, tal como su tamaño, veces cargado y estado actual. [1]

Si se solicita información sobre los módulos, se explorará el archivo con la información necesaria, se llenará la estructura de tipo Módulo, y se mostrará al cliente el resultado mediante una página web.

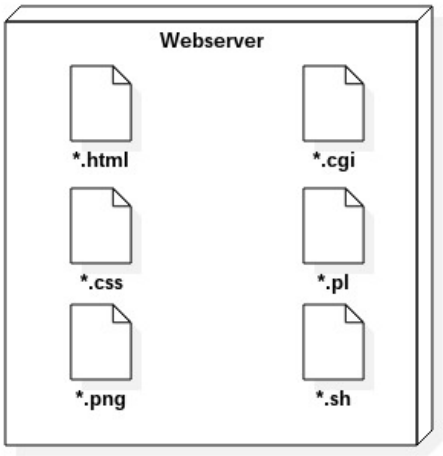


Desde el punto de vista del cliente, el diagrama de casos de uso que refleja los requerimientos solicitados es:



El deployment de los recursos en el servidor tiene la siguiente estructura:

- Archivos html, que conforman las páginas web estáticas.
- Archivos cgi y pl, que aportan el contenido dinámico.
- Archivos css, que aportan estilos a los html y cgi.
- Imágenes e íconos a mostrar en la página (png).
- Scripts auxiliares (shell scripts).



5. Implementación y Resultados

Para la implementación y desarrollo se utilizó el software Sublime Text. Para las pruebas se utilizó una máquina virtual en VirtualBox, que actúa como Servidor, y el Sistema Operativo Linux nativo actúa como cliente.

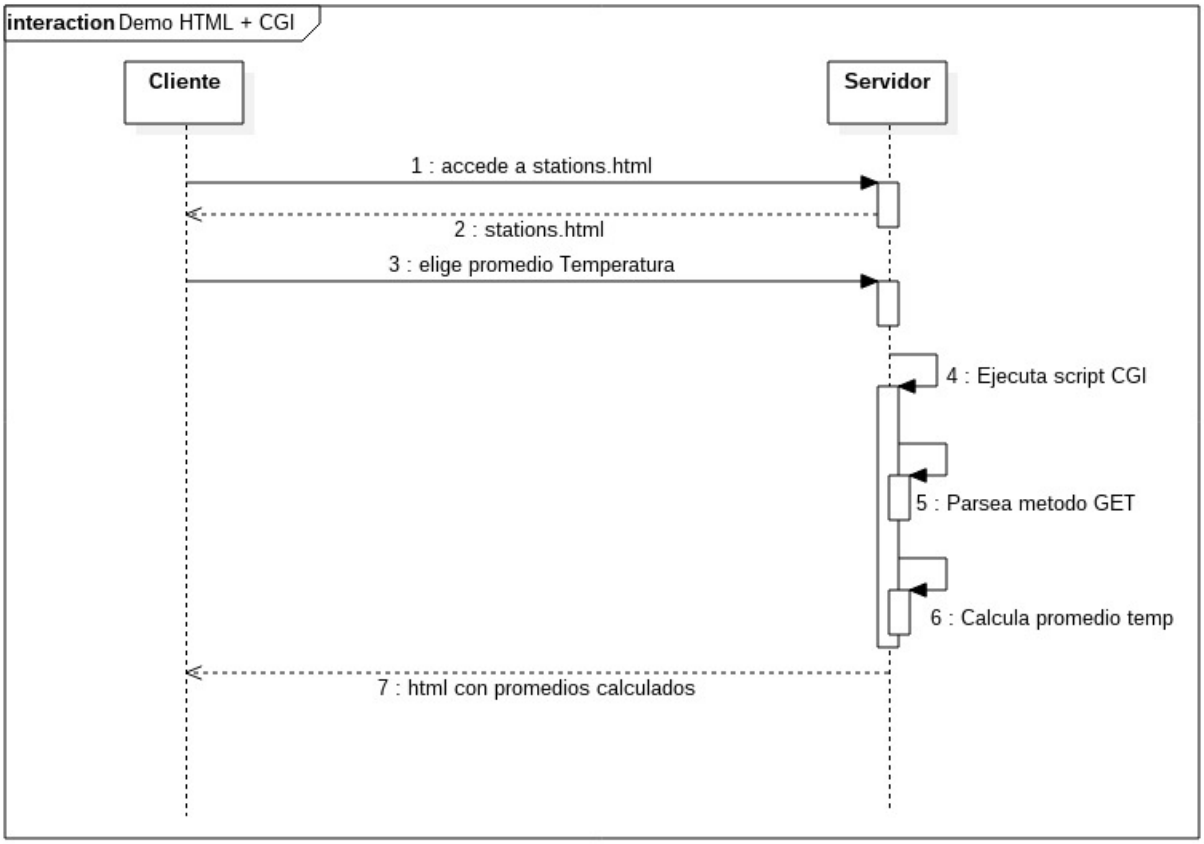
Con respecto a la selección del webserver a utilizar, se realizó una investigación en Internet sobre las alternativas, que se redujeron a Apache, Nginx, Monkey Server y Lighttpd. Apache no es una alternativa viable ya que su consumo de memoria es elevado, a pesar de ser ampliamente compatible y fácil de configurar. Nginx también se descartó por no ofrecer soporte nativo para CGI.
Finalmente se seleccionó Lighttpd por su bajo consumo de memoria, y su más fácil configuración comparado con Monkey Server. [2]

El diseño de la interfaz web fue realizado en base a un template de w3Schools, y utilizando su librería de estilos CSS. [3]
A partir de esto, se utilizaron diversos recursos de la página para construir la interfaz presentada en el trabajo.

La programación en el servidor se realizó mediante la interfaz CGI, usando scripts en Perl y binarios compilados en C.
Perl fue utilizado puntualmente a la hora de descargar y subir un archivo al servidor.

Si se ingresa a la página índice se verá una pantalla de bienvenida, que presenta el título del trabajo. Las pestañas superiores permiten acceder a las funciones antes descritas.
La información del sistema es mostrada mediante un script CGI. Al presionar en la pestaña **System Info**, el servidor comienza a ejecutar el script ksamp.cgi, el cual busca información en /proc, lo guarda en una estructura de datos, y al finalizar imprime código HTML para que el navegador le muestre al cliente el resultado. En medio de este código HTML se encuentran los datos anteriores. Para obtener la fecha y hora actual se utilizó la estructura tm que brinda Linux, y la función localtime. [4]

Para el caso de la pestaña **Stations** el procedimiento es similar, con la salvedad que se presenta primero una página web con los menús desplegables, y solo al seleccionar una opción se envía al servidor la petición que desencadena la ejecución de un binario cgi.
El siguiente diagrama de secuencia pretende explicar un poco más el orden de los pasos de la ejecución.



El programa cgi no puede ser llamado con argumentos como cualquier binario en C, por lo que debe usarse otra alternativa. HTTP brinda dos métodos para permitir la comunicación entre servidor y cliente: GET y POST. [5]
Se optó por utilizar GET ya que su implementación es más simple, y el alcance del proyecto no amerita implementar el nivel de seguridad que brinda POST.

Siguiendo la estructura del método GET, se al hacer click en una petición de promedio, por ejemplo, se envía al servidor la solicitud de ejecución del script CGI, con dos argumentos:

estaciones.cgi?**q=promedio**&**a=temp**

El query, simbolizado con 'q' es el nombre de la función solicitada, y el argumento 'a' es el parámetro que recibiría la función. En el ejemplo se está solicitando el promedio de la variable temperatura.
El script realiza el parseo de la cadena recibida, y si el resultado cumple con la estructura descrita, y los argumentos son válidos, ejecuta la orden. Si se ingresa un comando incorrecto, se notifica del error al cliente.

Para realizar la descarga se utiliza un script en Perl, que gestiona el proceso.
Se establece el directorio desde donde se podrán descargar archivos: **/var/www/html/files**. Los archivos de texto a descargar que genera el programa cgi en C son guardados allí.
Luego deben realizarse chequeos de seguridad antes de comenzar la descarga. Primero se verifica si el tamaño del archivo no supera un límite preestablecido por el script. Luego, como el script se encarga solo de las descargas, si se detecta un intento de subida de archivos, se informa del error.

Luego se obtiene el nombre del archivo deseado, parseando la cadena que brinda el método GET, y se pasa por un filtro formado por una expresión regular que verifica la estructura general del archivo solicitado.
Esto se hace para controlar que no se introduzcan comandos maliciosos, como una secuencia de barras y puntos, para cambiar la carpeta actual y descargar cualquier archivo del sistema (../../).

La Regex en cuestión es la siguiente: **^(\w+[\w.-]+\.\w+)$**
Se buscan palabras seguidas de puntos o guiones, seguidos de un punto y una nueva palabra (la extensión del archivo). En otras palabras, se filtra según los nombres corrientes que recibe un archivo, como perro.jpg o informe.txt.

Si estos controles se pasan con éxito, el archivo es descargado correctamente en el navegador del cliente. [6]

Finalmente, para el apartado referido a módulos del Kernel el procedimiento de desarrollo seguido es similar a lo antes comentado. La pestaña **Drivers and Modules** redirige a un programa CGI que explora el archivo /proc/modules, y muestra en pantalla una lista con información importante sobre cada uno. Además se presenta al usuario un **form** para seleccionar un archivo de su disco duro y subirlo al servidor.

Para gestionar la subida de archivos se utilizó nuevamente un script en Perl.
Este script setea la carpeta donde se guardarán los archivos que recibe el servidor: **/var/www/html/uploads.** Se obtiene el nombre del archivo que se quiere subir, y se comienzan a realizar controles de seguridad, para evitar uso indebido del servidor.

Se separa la cadena recibida en nombre y extensión. Luego se utiliza una Regex similar a la anterior, pero más restrictiva: **^(\w+[\w.-]+\.ko)$**
Aquí se busca una combinación de letras, puntos y guiones para el nombre del archivo, pero se restringe la extensión del mismo a ".ko", que denota módulos del Kernel. [7]

Si estos controles son pasados con éxito, se sube el archivo al servidor, y finalmente se llama al script que realiza la instalación propiamente dicha: **install.sh**

Este script automatiza el proceso de instalación de los módulos subidos. Explora la carpeta uploads/, e intenta ejecutar el comando insmod con cada archivo que se encuentre en la misma. Luego redirige al script **modules.cgi**, el mismo que se accedió anteriormente, que muestra la página web con la lista de módulos instalados.
Si todo salió bien, debería verse una nueva entrada en la lista.

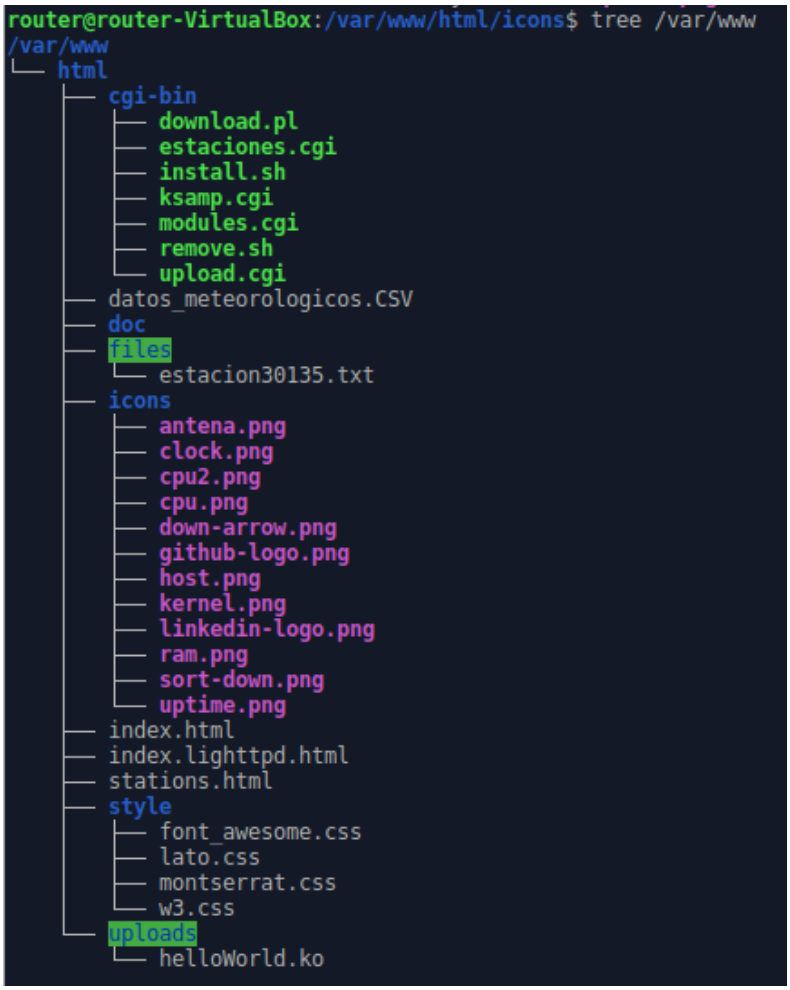
A modo de prueba se incluye en el proyecto el código fuente de un módulo simple que imprime Hola Mundo en los logs del Kernel, para probar el funcionamiento. [8]
Este módulo funcionará si el servidor es instalado en un sistema con arquitectura x86.
Si el servidor operará en una placa Raspberry PI, se provee el binario compilado del mismo módulo del Kernel: **helloworld_arm.ko** en la carpeta **ksrc/**

Para el borrado del módulo recién instalado, se provee el shell script **remove.sh**, que busca en la carpeta uploads/ todos los archivos, e intenta eliminar el módulo del kernel con el mismo nombre que el archivo que encuentre. Luego elimina el archivo.

Cabe mencionar que una optimización aplicada al proyecto ahorra mucho procesamiento innecesario. Si al solicitar la descarga de una estación dada, el servidor encuentra que el

archivo que quiere generar ya existe en la carpeta files/, no lo genera nuevamente, ni lee el archivo CSV, sino que directamente lo provee para que el cliente lo descargue.

La estructura de directorios del proyecto es la siguiente:



Con respecto a la estructura general del proyecto, se hizo un Makefile que compile y realice el linkeo de los archivos, guardando los .o en la carpeta obj, y los binarios finales en la carpeta cgi-bin.
Se incluyó un target para ejecutar CppCheck sobre el proyecto, guardando los resultados en el archivo **err.txt.**

El archivo Makefile tiene además un target que luego de compilar todo el proyecto, prepara una carpeta con todos los archivos necesarios para ser copiada directamente en la ruta de donde el webserver lee los archivos .

La estructura resultante es una carpeta **html/** que contiene archivos html, css, iconos, scripts cgi y archivos auxiliares para el funcionamiento del servidor.

Finalmente se incluye un shell script que realiza todas las configuraciones necesarias y mueve la carpeta anterior en la ruta correcta.

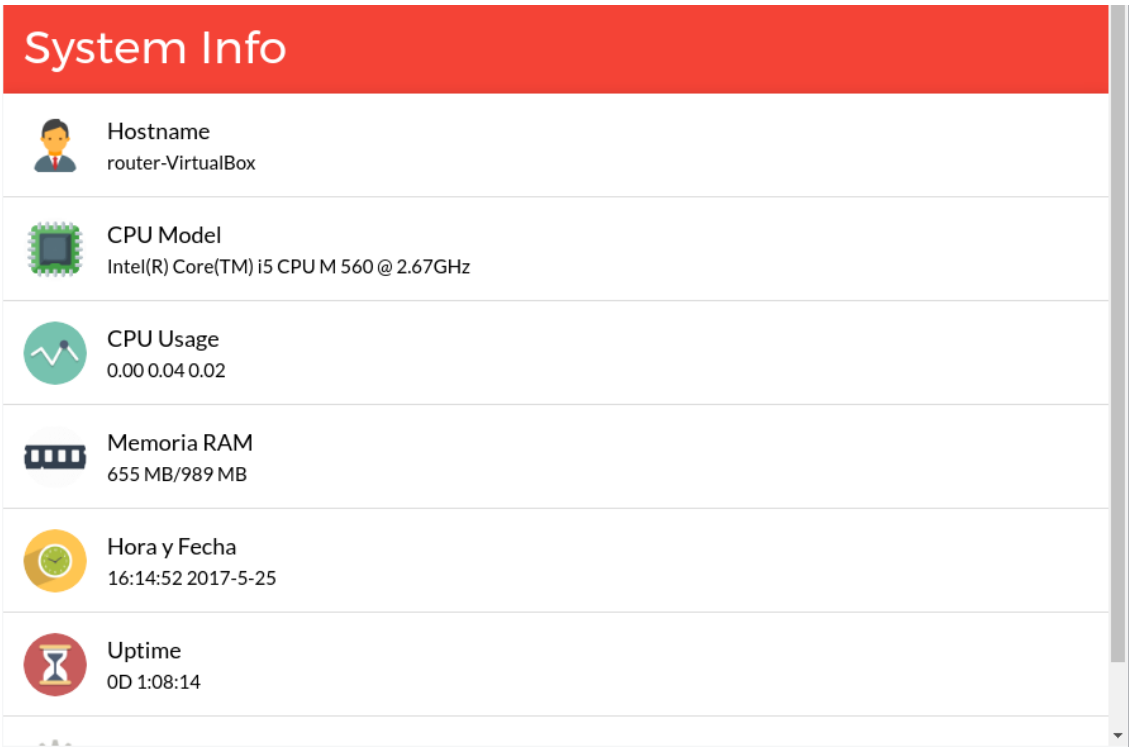
El archivo, **conf.sh**, realiza lo siguiente:

- Copia el archivo de configuración **lighttpd.conf** en **/etc/lighttpd/**
- Luego reinicia el servicio para aplicar los cambios.
- Copia la carpeta **html/** generada en la compilación a **/var/www/**
- Cambia los permisos de las carpetas **files/** y **uploads/**, que contienen archivos a descargar y subidos, respectivamente, para que el usuario **www-data** pueda modificarlas.
- Finalmente, crea un archivo llamado **lighttpdConf** en la ruta **/etc/sudoers.d/**, directorio que guarda configuraciones referidas a los permisos de cada usuario. En ese archivo se escribe la regla que indica que el usuario www-data tiene permiso de administrador, sin necesidad de contraseña, para ejecutar los programas **insmod** y **rmmod**.

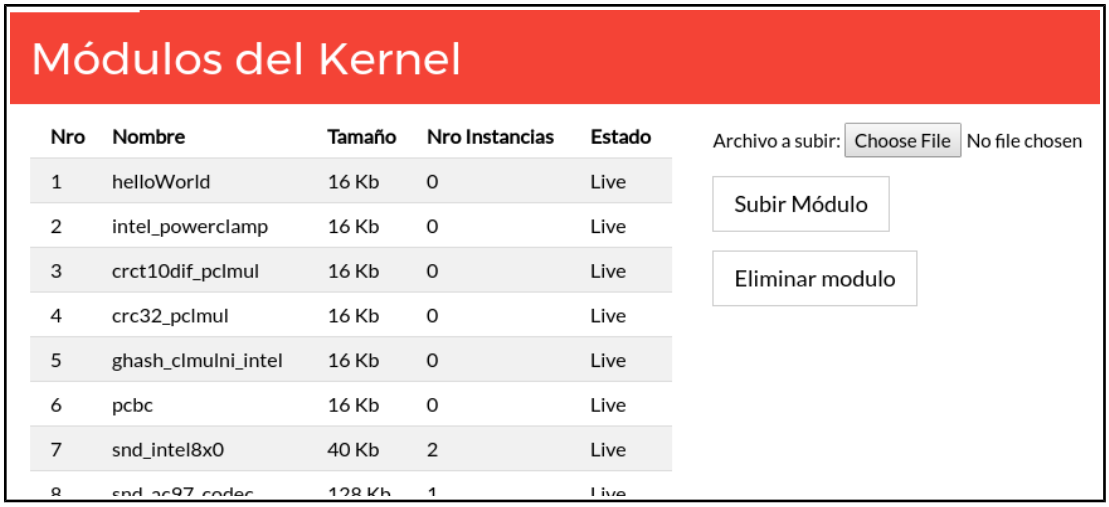
A continuación se adjuntan capturas de pantalla del programa en ejecución:

- Pantalla de bienvenida, con una barra superior para seleccionar las opciones disponibles.

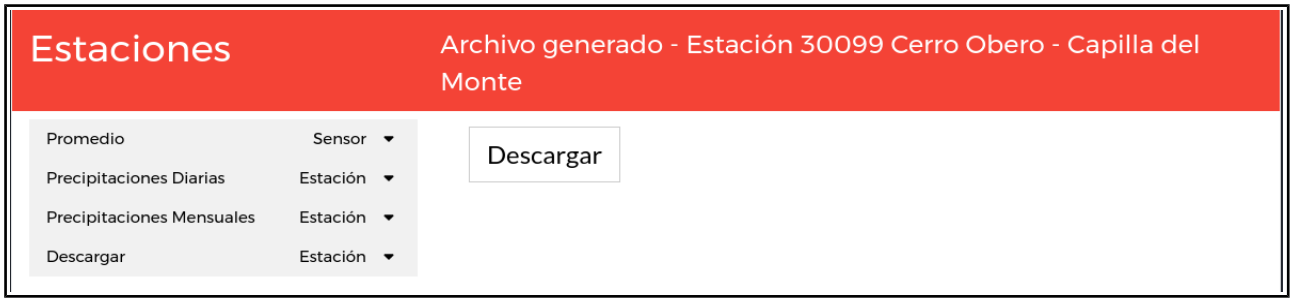
- Página con información sobre el sistema.

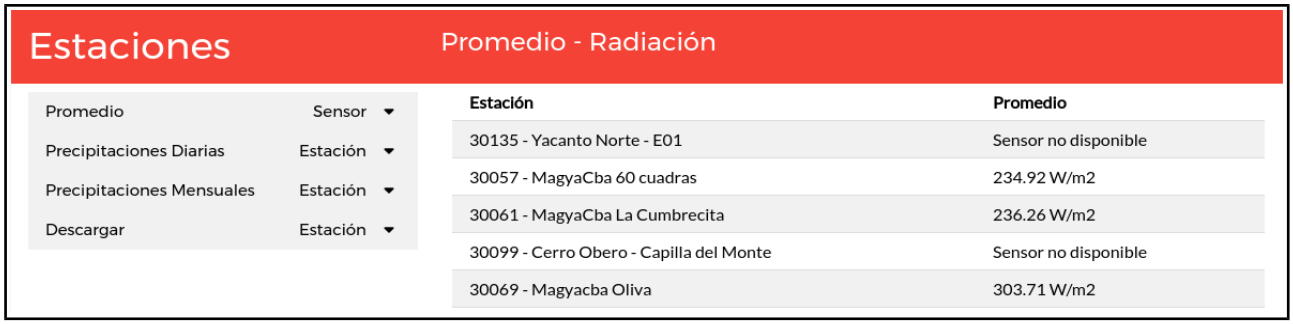


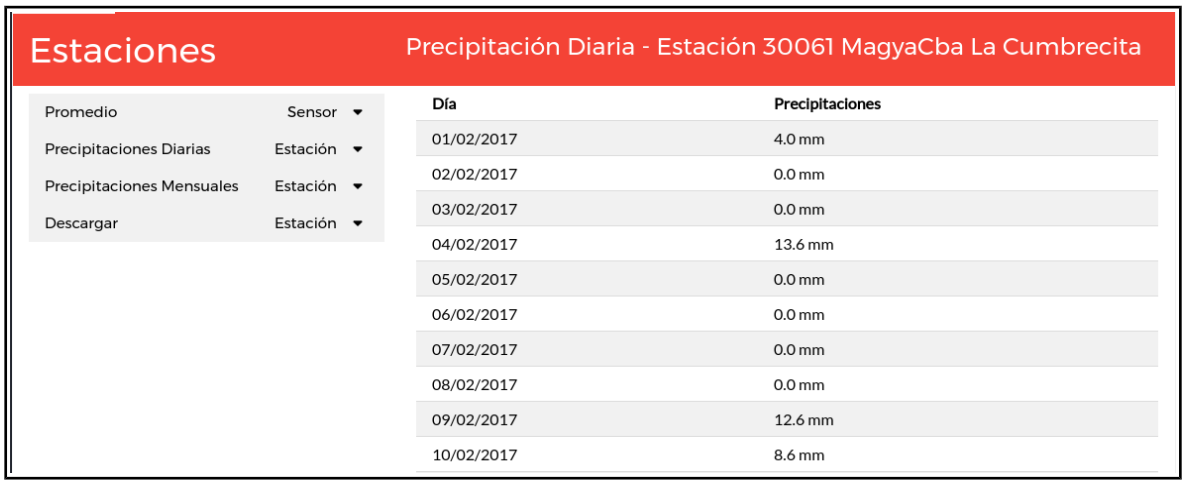
- Página que lista los módulos instalados en el Kernel. Puede verse que el primere elemento de la lista es el módulo de prueba helloWorld.



- Página con información sobre las estaciones. Se muestra la interfaz cuando se seleccionan las opciones de descarga, promedio y precipitaciones diarias.







6. Conclusiones

Como conclusión del proyecto, se aprendió a interactuar con un servidor web, teniendo que investigar las opciones disponibles, y configurar una alternativa liviana.
Luego se tuvo que aprender a utilizar el protocolo CGI, para brindar interactividad a la página web diseñada. Este práctico fue una buena oportunidad para realizar un diseño simple pero atractivo de una página web, además de toda la programación que lleva por detrás para su correcto funcionamiento.
Fue muy positivo y favorable tener código que puede ser reusado, y en mayor medida beneficioso, si lo programó el mismo alumno, ya que ayuda a una mejor comprensión de lo que se está haciendo. Se halló un poco dificultoso recortar los proyectos anteriores, pero con paciencia se hizo correctamente y el servidor, como se ve en las capturas de pantalla, funciona como es esperado.

7. Apéndices

A la hora de realizar este trabajo se utilizó contenido de las siguientes URL:

- [1] https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/5/html/Deployment_Guide/s2-proc-modules.html
- [2] https://help.dreamhost.com/hc/en-us/articles/215945987-Web-server-performance-comparison
- [3] https://www.w3schools.com/w3css/tryw3css_templates_start_page.htm#
- [4] https://stackoverflow.com/questions/1442116/how-to-get-date-and-time-value-in-c-program
- [5] https://www.w3schools.com/tags/ref_httpmethods.asp
- [6] https://bytes.com/topic/perl/insights/857373-how-make-file-download-script-perl
- [7] https://www.sitepoint.com/uploading-files-cgi-perl/
- [8] http://www.tldp.org/LDP/lkmpg/2.6/html/lkmpg.html#AEN121