Universidad Nacional
de Córdoba

Cátedra de Sistemas Operativos II

Trabajo Práctico Nº III

ZIMMEL CECCÓN, Ezequiel José

https://gitlab.com/sistiop2/tp3_embebido

2 de febrero de 2020

Índice

Introducción

En la intersección de Software, Hardware y Comunicaciones nos podemos encontrar a los Sistemas Embebidos. Los mismos son sistemas que pueden definirse como computadoras destinadas a un fin determinado, es decir, computadoras con requerimientos de hardware, software y comunicaciones bien específicos.

Propósito

El propósito del presente Trabajo Práctico es el de diseñar e implementar un software que sea capaz de correr en un *webserver* y ejecutar código CGI en su backend. Debe permitir entender las capacidades y limitaciones del hardware al momento de desarrollar una aplicación sobre un sistema embebido, qué se debe tener en cuenta al trabajar sobre los mismos, y cómo adaptar el software para poder cumplir los objetivos planteados a pesar de las posibles limitaciones presentes.

El software debe poder realizar tres tareas:

1. Una página que reporte información sobre recursos varios del sistema embebido.
2. Una página con un formulario que permita definir una fecha (año y día DOI) y que, al enviarlo, devuelva la lista de archivos disponibles de GOES 16 en AWS (producto ABI-L2-CMIPF, canal 13).
3. Una página que liste los módulos instalados en el sistema y que posea un formulario que permita subir un archivo al servidor, controlar que éste sea un archivo válido e instalarlo en el kernel del sistema operativo.

Ámbito del Sistema

Para el desarrollo del trabajo se usará una computadora Raspberry pi modelo 3B+ con sistema operativo Raspbian como servidor. Las razones por las que se han elegido el sistema operativo se desarrollarán más adelante.

Como cliente se puede utilizar cualquier dispositivo dotado de un navegador web. Por defecto, es necesario disponer de un router o switch de capa 3 para servir las direcciones IP mediante DHCP.

Definiciones, Acrónimos y Abreviaturas

- **CGI**: *Common Gateway Interface*. Interfaz orientada a ejecutar programas externos en un servidor de información de forma independiente a la plataforma.
- **Meta-variable**: Parámetro que envía información desde el servidor al script. No necesariamente es una variable en el ambiente del sistema operativo, aunque es la aplicación más común.

- **URI**: *Uniform Resource Identifier*. Cadena de caracteres que identifica los recursos de una red de forma unívoca. No cambian en el tiempo, como puede ser el caso de los URL.
- ***Webserver***: Software, típicamente instalado en una computadora dedicada, capaz de realizar conexiones con otras computadoras (llamadas clientes),las que realizan peticiones sobre el servidor y el mismo las procesa y entrega una respuesta.
- **HTTP**: *Hyper Text Transfer Protocol*. Protocolo de comunicación para la transferencia de información en la *World Wide Web*.
- **AWS**: (Amazon Web Services) es una colección de servicios de computación en la nube pública que en conjunto forman una plataforma de computación en la nube.

Referencias

- *https://www.geeksforgeeks.org/perl-cgi-programming/*
- *https://registry.opendata.aws/noaa-goes/*
- *https://www.perl.com/article/perl-and-cgi/*
- *https://stackoverflow.com/questions/9763348/upload-file-using-perl-cgi*
- *https://metacpan.org/pod/CGI::Easy::SendFile*
- *https://stackshare.io/stackups/apache-httpd-vs-lighttpd-vs-nginx*
- *https://www.geeksforgeeks.org/perl-get-vs-post-in-cgi/*
- *https://wofford-ecs.org/DataAndVisualization/CGIProgramsAndWebForms/material.htm*
- *http://www.parkansky.com/tutorials/bdlogcgi.htm*
- *https://es.wikipedia.org/wiki/Setuid*

Descripción General del Documento

El propósito de este documento es proporcionar, a quien lo lea, un entendimiento básico de los requisitos y alcance de este proyecto, así como los pasos que siguieron para su diseño e implementación.

Descripción General

En la presente sección se explicarán los distintos aspectos del trabajo. Las funciones que ofrece el producto, la perspectiva del mismo, las características de los usuarios, requisitos para futuras iteraciones, entre otras cosas.

Perspectiva del Producto

El producto solicitado es un sistema que gestione la conexión con una plataforma servidor, para acceder a los datos específicos del sistema, así como también poder realizar consultas a Amazon y permitir la carga/descarga de módulos ya compilados a dicho sistema.

La solución implementada requiere la conexión de los equipos involucrados a un router o switch de capa 3 con DHCP activado para brindar los servicios a los clientes. La forma de interactuar con los clientes es mediante páginas estáticas.

Funciones del Producto

Las funcionalidades del producto son provistas por el servidor en el sistema embebido exclusivamente. Los clientes sólo acceden a las mismas mediante un navegador web.

Las funciones a proveer son tres:

1. Página que muestre información del sistema (Procesador, consumo, memoria, *uptime*, fecha y hora actual)
2. Página con un formulario que permita ingresar un año y día juliano, mediante el cual se obtiene una lista de los archivos disponibles en un AWS sobre Goes 16 (producto ABI-L2- CMIPF, canal 13).
3. Página que liste los módulos instalados en el sistema y que permita subir un módulo de kernel para su instalación. También se debe dar la opción de eliminar dicho módulo.

Características de los Usuarios

El usuario del programa solo necesita tener un conocimiento básico de la navegación por internet y conocer la dirección IP del *webserver* para poder utilizarlo. Luego, todas las opciones disponibles se presentan en pantalla con una interfaz simple y fácil de utilizar.

Restricciones

Para poder utilizar los servicios es necesario estar conectado a la misma red que el servidor.

Suposiciones y Dependencias

- El sistema embebido debe ser tipo Raspberry Pi o similar, y poseer MMU.
- Para probar el *webserver*, se necesita tener instalado un navegador web.
- Existe conectividad entre el servidor y el cliente, para poder realizar la comunicación.
- El servidor debe tener instalados todos los paquetes necesarios para su funcionamiento. Es condición necesaria haber instalado en el servidor:
    - perl libcgi-pm-perl
    - apache
    - awscli

Requisitos Específicos

Interfaces Externas

La interfaz externa a utilizar por el embebido debe permitir conectarse a un router con DHCP habilitado. Dicha conexión puede darse mediante cable ethernet o mediante Wi-Fi. Para la implementación del trabajo se ha optado por cable de red. Los clientes se conectan a la red a la que pertenece el servidor por los medios que el router permite.

Funciones

Las funciones se proveen mediante una interfaz web sencilla. La misma provee botones para elegir la acción a realizar.

- Al consultar los recursos del embebido, la página debe mostrar información, actualizada cada 5 segundo, sobre:
  - procesador,
  - memoria,
  - *uptime*,
  - fecha/hora actual.
- Para hacer la consulta GOES, se debe permitir al usuario ingresar, en un campo de texto, el año y día DOI a consultar. Luego, se debe mostrar una página que liste los resultados.
- Cuando se ingresa a administrar drivers, se debe mostrar una página con tres elementos:
  - Campo para subir archivo
  - Cuadro de texto para escribir el nombre del módulo a eliminar
  - Listado de los módulos instalados.

Requisitos de Rendimiento

El servidor debe ser capaz de brindar las funciones requeridas en un promedio de tiempo que no resulte incómodo para el usuario. A los fines de definir criterios de aceptación, se establece un tiempo de respuesta menor o igual a 5 segundos como aceptable.

Restricciones de Diseño

- El sistema operativo a instalar debe ser tipo GNU/Linux.
- El sistema embebido debe ser tipo Raspberry Pi o similar y debe poseer MMU.
- El *webserver* se debe ejecutar automáticamente en el inicio del sistema embebido.
- La interfaz web debe implementarse con HTML.
- Las aplicaciones deben programarse en CGI (Perl o C).

- Se debe ofrecer una forma de insertar/remover un nuevo módulo al sistema embebido.
- Se debe controlar que el archivo subido sea válido (.ko).
- Se debe utilizar Cppcheck y compilar con las flags -Werror -Wall y -pedantic.

Diseño de solución

Entorno

Como prerrequisito, antes de iniciar con el trabajo, es necesario la elección de la imagen de linux que utilizaremos en la Raspberry PI 3 Model B+ (de ahora en adelante rpi), para trabajar con ella.

Algunas características importantes del hardware son:

- CPU + GPU: Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz,
- 1Gb RAM, compartido con la GPU,
- Wi-Fi: 2.4GHz IEEE 802.11.b/g/n/ac,
- Ethernet: Gigabit Ethernet sobre USB 2.0 (300 Mbps)

Elección del Sistema Operativo

El fabricante de Raspberry pone a disposición en su sitio web una serie de sistemas operativos, no obstante es posible instalar cualquier distribución basada en arquitectura ARM. Sistemas operativos candidatos:

- **Raspbian**: Sistema operativo basado en Debian, S.O. oficial para la Raspberry Pi. Compatible con todos sus modelos y especialmente diseñada para ejecutarse en la misma. Presenta como ventajas una gran comunidad de usuarios, soporte oficial del fabricante, compatibilidad con todos los recursos de hardware, y haber sido diseñada para correr específicamente en la arquitectura del embebido.
- **Windows IoT core**: Sistema operativo de Microsoft orientado a instalarse en sistemas embebidos. Optimizado para dispositivos pequeños y de bajo costo.
- **Ubuntu**: A partir del modelo 2 de la Raspberry Pi, se puede ejecutar una versión de Ubuntu para procesadores ARM.

El SO que se eligió para llevar a cabo dicho proyecto fue **Raspbian**, distribución basada en Debian, debido a que utiliza programas más livianos y se encuentra más optimizado para su uso en la plataforma. Las distribuciones basadas en Ubuntu experimentan latencias prolongadas en tiempo de arranque y durante la ejecución de ciertos programas.

Elección del *webserver*

Los *webserver* analizados tiene como principal característica el soporte para CGI:

- **Apache**: es el *webserver* más utilizado. Para los sitios web pequeños y medianos, Apache tiene varias ventajas sobre Nginx, como su fácil configuración, muchos módulos y un entorno amigable para principiantes. Sin embargo, es un *webserver* muy pesado. Tiene soporte para CGI.
- **Nginx**: usa menos memoria que Apache y puede manejar aproximadamente cuatro veces más solicitudes por segundo. Esto último no es muy relevante en el marco del problema dado, ya que es una página web pequeña. Es más difícil de instalar y configurar que Apache. No tiene soporte para CGI.
- **Lighttpd**: su objetivo (característica) primordial es el de ser rápido, seguro, flexible y fiel a los estándares. Es un *webserver* recomendable, especialmente en servidores con excesiva carga, ya que requiere menos capacidad de proceso y memoria RAM. Tiene soporte para CGI.

- **Busybox httpd**: Busybox es un programa que combina varias utilidades en un solo ejecutable pequeño. Entre esas utilidades, se encuentra un demonio que procesa y responde peticiones HTTP. Para aplicaciones de muy pequeña escala, y con poca necesidad de prestaciones, es una buena alternativa cuando se requiere hacer uso de recursos de hardware limitados. Compatible con CGI.

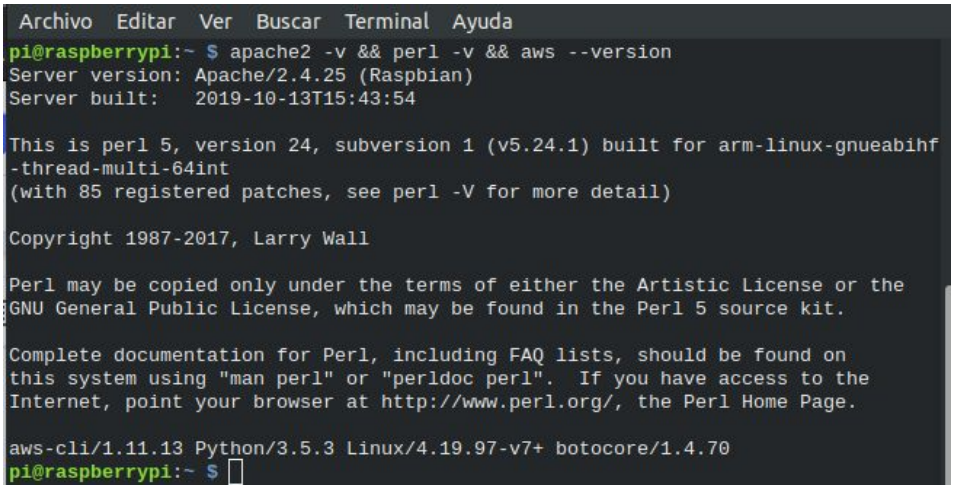
Si bien tanto Nginx como Lighttpd presentan mejor desempeño en el consumo de memoria y tiempo de respuesta durante instancias de mucho tráfico, se eligió al *webserver* Apache por contar con extensa documentación y disponer de módulos que simplifican y dotan de flexibilidad la puesta a punto del servicio.

Instalación y configuración del entorno

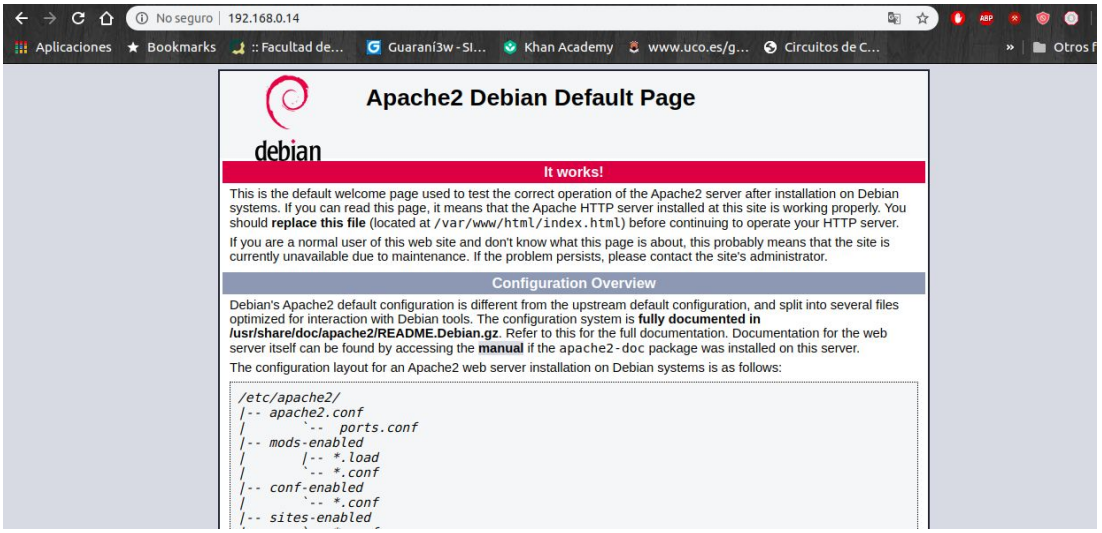
Realizada la elección del sistema operativo y del *webserver*, se procede a la instalación en una unidad SD, que será montada en la Raspberry Pi 3 B+:

1. Preparación de la tarjeta SD con la imagen de Raspbian, descargada desde: *https://www.raspberrypi.org/downloads/raspbian/*
   Puede emplearse el software *Rufus* para cargar los archivos en la memoria SD.
2. Conectar la Raspberry a su fuente de alimentación y periféricos, para realizar las configuraciones iniciales.
3. Instalar y configurar el SO. Para la configuración, se hace uso del script raspi-config. Es necesario habilitar SSH y expandir el *filesystem*. Terminado este paso, ya se puede prescindir del uso de los periféricos conectados a la Raspberry Pi.
4. Instalar los paquetes necesarios:
   a. *sudo apt update*
   b. *sudo apt install apache2*
   c. *sudo apt install perl libcgi-pm-perl*
   d. *sudo apt install awscli*
   e. Reiniciar el servicio apache2 mediante *sudo /etc/init.d/apache2 restart*, o reiniciar el equipo mediante *reboot*. Apache automáticamente levantará el servicio cuando inicie el equipo.

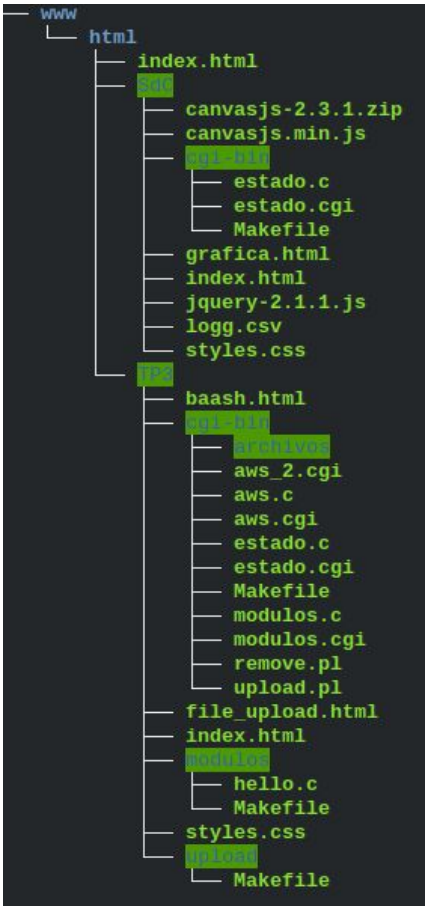
Verificamos que los complementos se encuentre instalados:



Reiniciado el servicio de Apache, verificamos que éste se encuentre levantando.



Los proyectos deben montarse en el directorio */var/www/*, como se muestra en la siguiente imagen. El proyecto de nombre **TP3** es el que aplica al presente documento.



Verificamos si el módulo CGI se encuentra cargado; en caso de no estar, lo habilitamos mediante *sudo a2enmod cgid_module*.

```
pi@raspberrypi:~ $ apache2ctl -t -D DUMP_MODULES | grep cgid
AH00558: apache2: Could not reliably determine the server's f
ully qualified domain name, using 127.0.1.1. Set the 'ServerN
ame' directive globally to suppress this message
 cgid_module (shared)
pi@raspberrypi:~ $
```

Por último, suele ser necesario definir el directorio donde ejecutaremos archivos CGI. Esto último se realiza en el archivo *apache2.conf*.

```
<Directory /var/www/>
        Options Indexes FollowSymLinks
        AllowOverride None
        Require all granted
</Directory>

<Directory /var/www/html/TP3/cgi-bin>
        Options +ExecCGI
        AddHandler cgi-script .cgi .pl
</Directory>

<Directory /var/www/html/SdC/cgi-bin>
        Options +ExecCGI
        AddHandler cgi-script .cgi .pl
</Directory>
```

Implementación y Resultados

Antes de entrar en detalle con la implementación detallaremos los “métodos” que se tiene para interactuar con la aplicación CGI. De esta manera, al momento de realizarse una consulta al servidor, se define como el programa recibe dichos datos:

- **GET**: Para el caso de CGI, este método hace que el programa reciba los datos desde la cadena de la consulta (query string). El programa debe procesar e interpretar dicha cadena para luego ejecutar la acción deseada. La consulta realizada pasa a formar parte de la URL, junto con todas las variables que hayan en el medio. Se inicia el campo de variables con “*?*”, y se separan las mismas con el símbolo “*&*”. No es recomendable para grandes volúmenes de información (más de 1024 bytes, límite de una URL). Este método se usa principalmente para consultar datos almacenados en el servidor.
- **POST**: Cuando se implementa este método en una aplicación con CGI, el servidor le transmite los datos desde la entrada estándar. Esto implica que no se pueda consultar por el carácter de fin de archivo (EOF). En su lugar, es necesario consultar el tamaño del contenido (content length). Este método es el recomendado para realizar consultas que puedan modificar los datos en el servidor, o enviar grandes volúmenes de información.

Detallado los métodos de interacción con la aplicación CGI, el sistema fue diseñado de forma tal que lo primero que ve el usuario es una página índice con tres botones. Con los mismos, se accede a cada una de las funcionalidades que provee el embebido. En ningún momento la información solicitada a devuelta queda registrada en la URL, haciendo uso en todo momento del método POST.

Cada página corresponde a un código CGI. El acceso a AWS es un script escrito en Perl, mientras que el acceso a los módulos y el estado del sistema se encuentra escrito en C.

Se detallan las características y los detalles de implementación:

- Estado del sistema. La obtención de los datos se realiza mediante funciones implementadas en C. Por otro lado, para realizar su visualización en la página web, se agrega la estructura HTML de la página en el código C. Cuando se compila se obtiene un archivo CGI que interpreta el código HTML.

```
   printf("%s%c%c\n", "Content-Type:text/html;charset=iso-8859-1", 13, 10);
   printf("<META HTTP-EQUIV=REFRESH CONTENT=5>");
   printf("<TITLE>Estado del sistema</TITLE>\n");
   printf("<body style=background-color:rgba(10,50,0,0.5);>");
   printf("<font color=white face=Verdana>");
   printf("<h1 align=center>SISTEMA EMBEBIDO - TP3</h1>");
   printf("<H2 align=center><tt>ESTADO DEL SISTEMA \n</tt></H2></font>");
   printf("<hr>");
   CPU();
   printf("<div></div>");
   cputimes();
   printf("<div></div>");
   memmory();
   printf("<div></div>");
   fecha();
   printf("<div></div>");
   bootTime();
   printf("<div></div>");
   uptime();
   printf("<div></div>");
   load();
   printf("<div> </div>");
   printf("<hr>");
        printf("<button   type=button   style=font-size:12pt;height:35px;width:100px
onclick=javascript:window.close()>Salir</button></p>");
   printf("</body>");
   return 0;
```

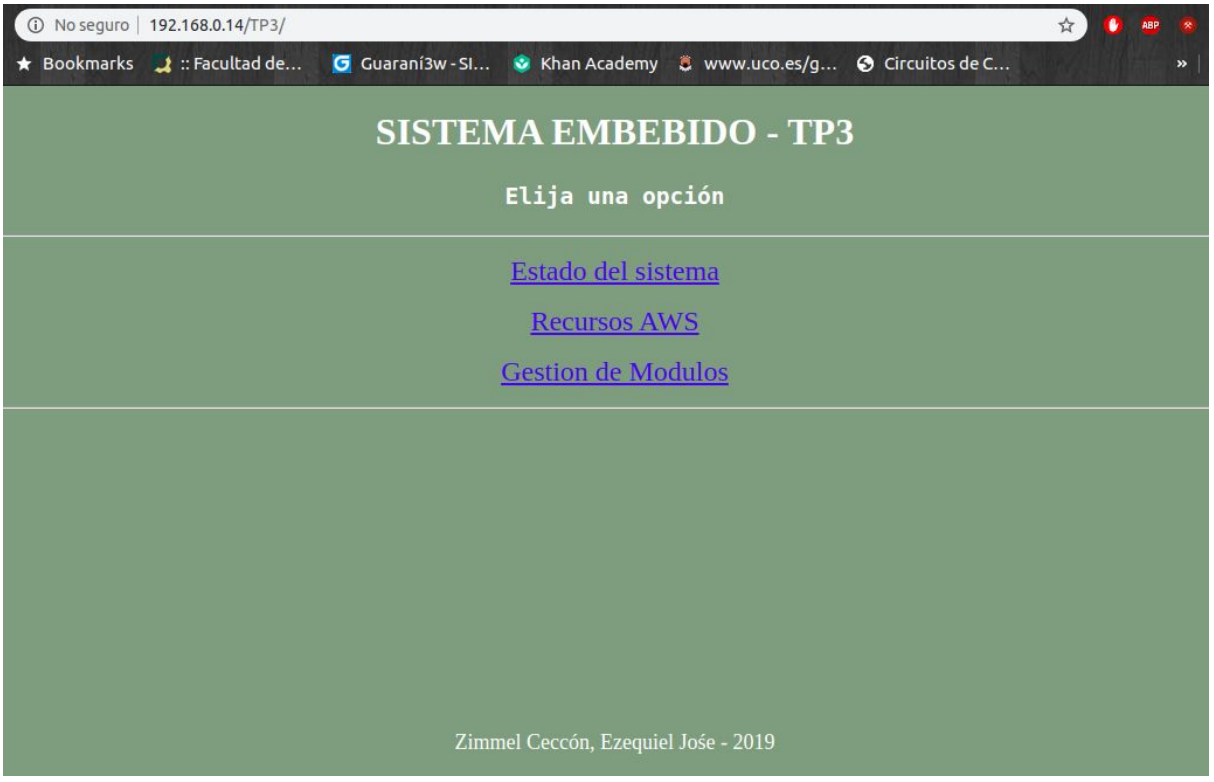
- Recursos AWS. Al ingresar, se le solicitará al usuario que introduzca, en un campo de texto, la fecha y día que desea consultar, en el formato <año>/<dia>/. A su vez, se permite seleccionar el producto y el canal a solicitar. El sistema procesará la consulta y, cuando obtenga los datos, los listará.
  El core de la implementación se encuentra en la siguiente línea de código, en donde se busca por nombre de producto y fecha, filtrando luego el canal a mostrar:

```
              foreach(`aws     s3     --human-readable     --recursive     ls
s3://noaa-goes16/ABI-$Prod/$Date/ --no-sign-request | grep -E '$Ch'`)
   {
```

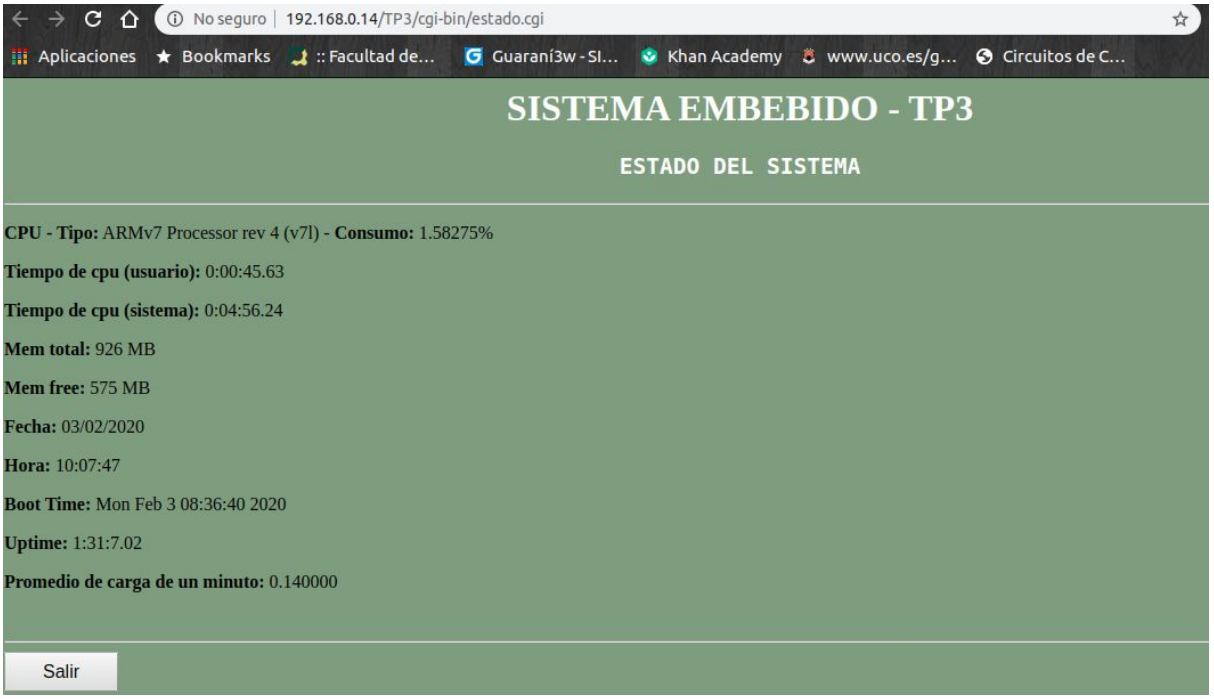
```
        print "<br>";
        print "$_ ";
        print "<br>";
    }
```

- Gestión de Módulos. Con esta opción, el usuario recibe un listado de todos los módulos instalados en el sistema y se brinda la opción de agregar o quitar módulos. Cuando se suba un archivo para ser instalado, el sistema primero debe corroborar que sea de la extensión ".ko". Si cumple con dicha condición, intentará instalarlo. Para eliminar un módulo, se presenta una caja de texto en donde el usuario escribe el nombre del módulo a eliminar. En ambos casos, para cargar/descargar el módulo se solicitará el ingreso de contraseña para validar la operación.

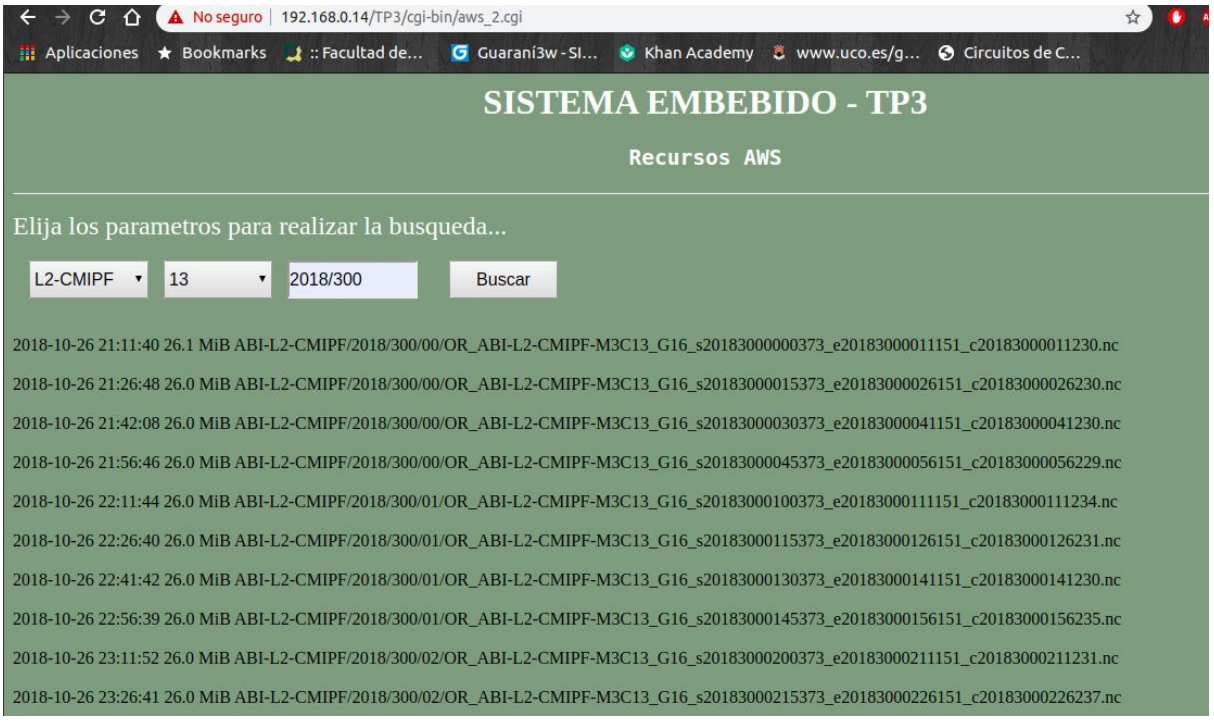
La manera de acceder al sitio es ingresando una URL con el siguiente delimitador: *http:\\<IP sistema embebido>/TP3*



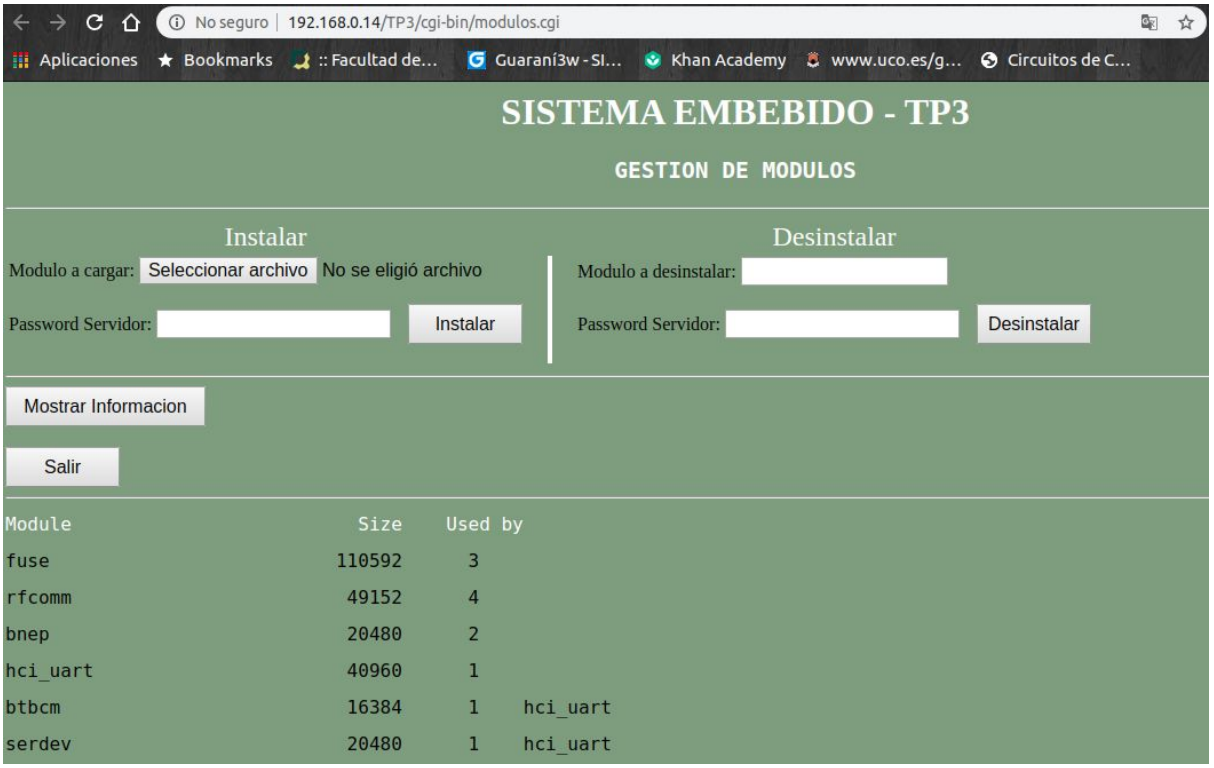
Estado del sistema. Muestra el estado de servidor.



Recursos AWS. Permite realizar la búsqueda por producto, canal y fecha. Obtenidos los resultado, son desplegados en orden ascendente de fecha.



Gestión de Módulos. Por defecto, la página muestra los módulo cargados actualmente. A su vez, brinda la posibilidad de agregar un nuevo módulo o desinstalar uno ya cargado.



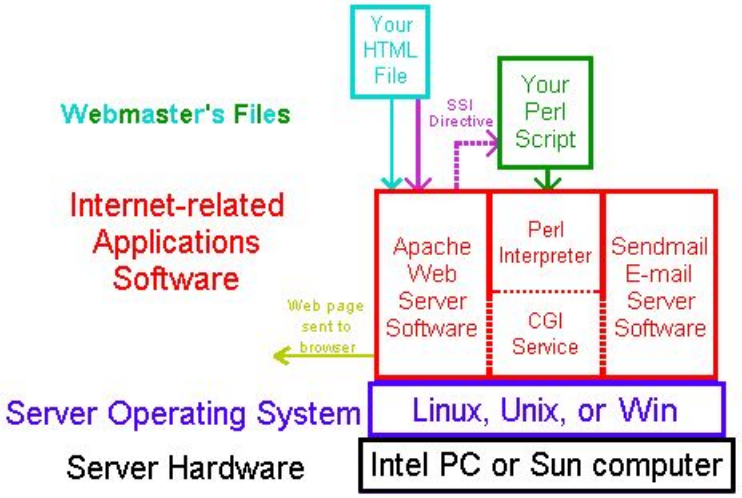
Cuando seleccionamos la opción “Seleccionar archivo” sólo se nos permite seleccionar archivos con extensión .ko.

La comunicación mediante CGI se realiza con cuatro métodos posibles:

1. **Variables de entorno** (meta-variables). Variables específicas establecidas por el servidor cuando se ejecuta una aplicación CGI. Para la acción de transferir archivo, nos interesa CONTENT_LENGTH y CONTENT_TYPE.
2. **Línea de comandos**. Usado para casos especiales.
3. **Entrada estándar**. Es invocado cuando el método HTTP utilizado es POST.
4. **Salida estándar**. Envía los resultados del programa CGI a la salida estándar. Puede ser enviado directamente al navegador del usuario, o al servidor web para que realice alguna acción.

Pasos:

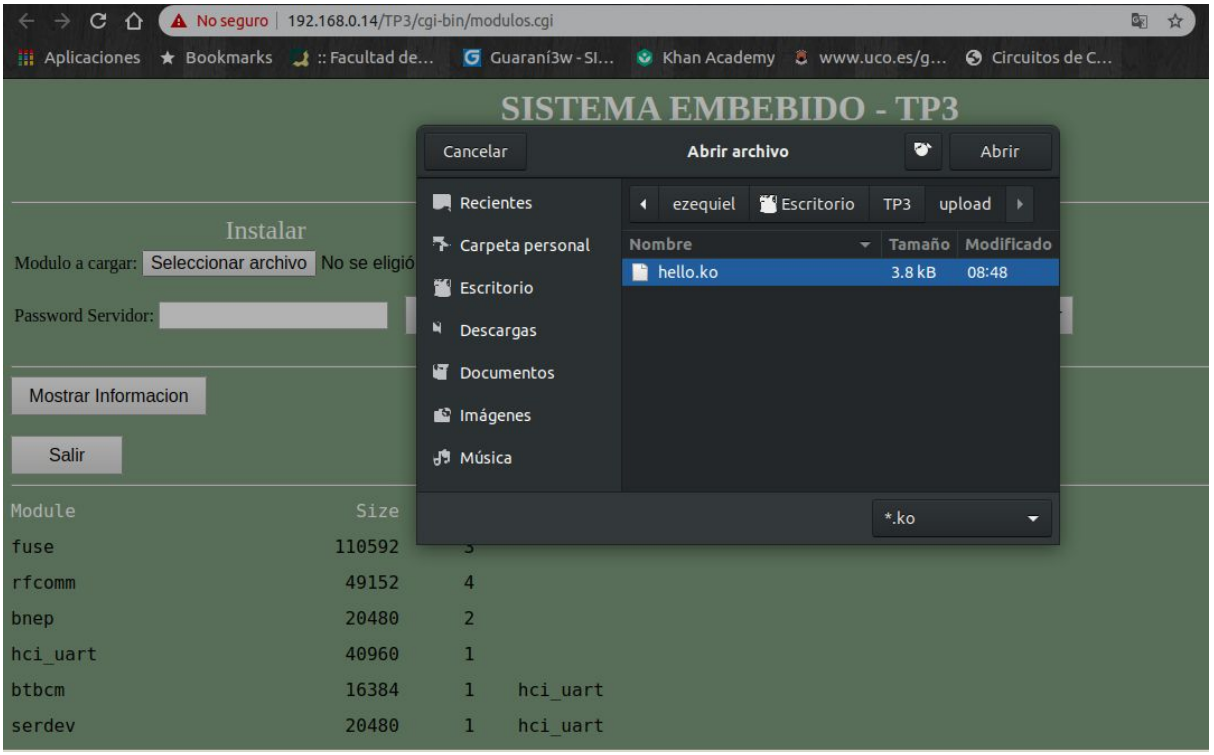
1. El cliente realiza una consulta en forma de URI.
2. El servidor actúa como puerta de entrada a la aplicación:
    a. Recibe la consulta
    b. Selecciona script CGI para servir a la consulta, interpretándose como un ejecutable.
    c. Convierte la consulta a consulta CGI. Se le pasan los datos mediante meta-variables o el cuerpo del mensaje.
    d. Ejecuta el script.
    e. Convierte la respuesta CGI a una consulta para el cliente (HTTP). Un script siempre debe dar una respuesta no vacía ante una consulta.
    f. Al servir la consulta, es el encargado de implementar los protocolos de seguridad y autenticación necesarios. También debe realizar la traducción entre protocolos de ser necesario.



El archivo se transmite en bloques de tamaño fijo mediante protocolo HTTP. Desde el lado receptor (embebido) debe haber una porción de código capaz de interpretar dichos bloques, ordenarlos y unirlos. Dicho programa se lo conoce como “parser”. El mismo abre un archivo nuevo y vacío para escritura, a medida que recibe los bloques, los va escribiendo sobre el archivo y, al finalizar la transferencia, lo cierra.
Antes de comenzar la transferencia, el intérprete CGI debe obtener el nombre de archivo.

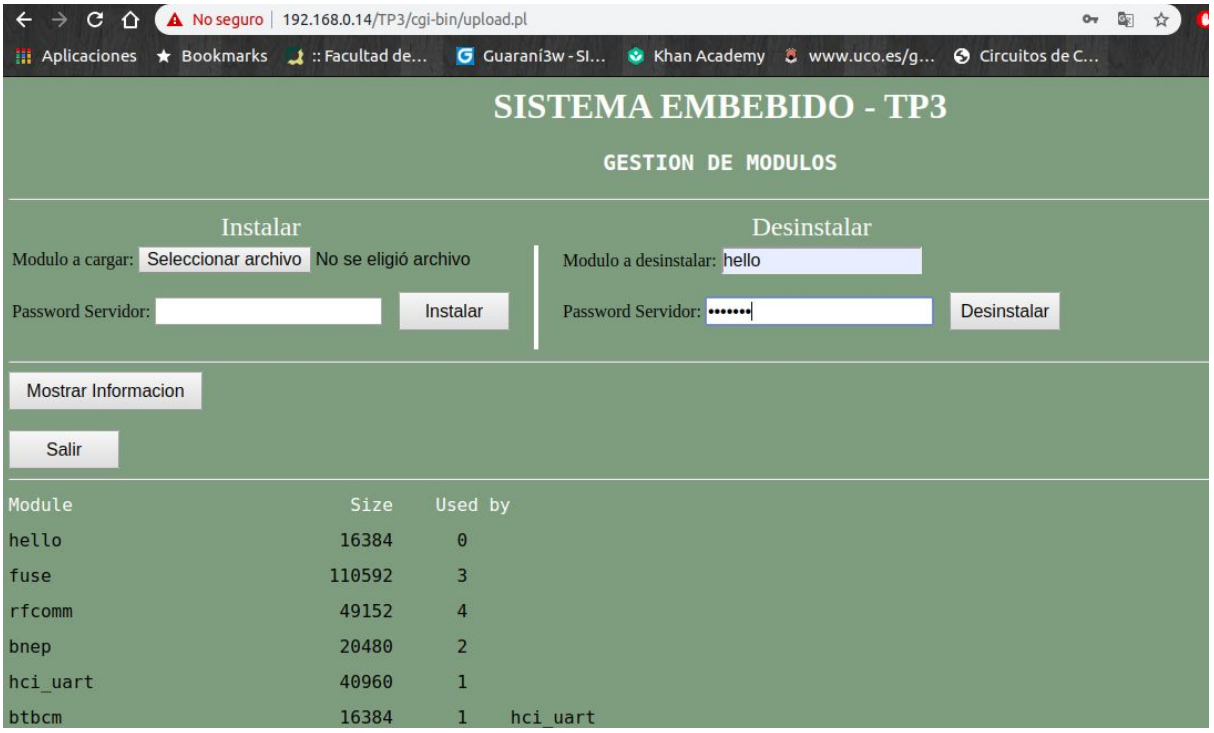
Al momento de enviar los bloques, también se envían otros datos en forma de cabeceras. Por ejemplo, el tipo de contenido MIME.
Cuando se procesa un archivo subido, CGI crea un archivo temporal en el disco y pasa un file handler a ese archivo. Luego de completada la transferencia, dicho archivo se elimina.
Algunos puntos para el caso de sistemas operativos UNIX-compatibles:

- Las meta-variables son pasadas como variables de entorno, con el mismo nombre. Estas variables son accedidas por la función C getenv().
- Se accede a la línea de comandos mediante los argumentos de la función main().
- El directorio actual de trabajo es, típicamente, el directorio donde actualmente se encuentra el script.



Validadas las credenciales que solicita el servidor, el módulo es cargado y desplegado junto a los otros.

En caso de querer desinstalar un módulo, basta con ingresar el nombre de éste y la credencial para autenticar la operación.



Para insertar un módulo de kernel se hace uso de uno de los cuatro métodos de interacción con CGI: la línea de comandos. El script CGI invoca a los comandos “insmod” y “rmmod” tal y como si hubiese sido escrito a mano por un usuario. Previamente, se le debe haber dado los permisos necesarios al script, y se debe haber modificado los permisos a los

comandos en cuestión para ser ejecutados por grupos con sudo chmod 4755 (4: setuid/7:rwx para dueño/5: rw para grupo/5: rw para otros). Setuid Se utiliza para elevar temporalmente los permisos a los usuarios para poder ejecutar los comandos que requieren privilegios elevados. Con setuid, se le brinda a los usuarios los privilegios del dueño de ejecutable (root en nuestro caso). Por motivos de seguridad, este bit impide que el usuario que recibió estos nuevos permisos temporales pueda alterar el nuevo proceso.

Conclusiones

Con este trabajo práctico he obtenido un entendimiento más profundo sobre los sistemas embebidos, principalmente a tener en cuenta sus restricciones de hardware. Típicamente, al momento de realizar un desarrollo, se suelen instalar muchos complementos, o no considerar cuestiones como el tamaño de las variables empleadas. Al momento de instalar software, se tiende a ir por el que más prestaciones tiene, independientemente de su consumo de recursos. Todas estas facilidades tiene su contrapunto y deben ser dejadas de lado al momento de trabajar con un sistema embebido, y se debe hacer lo posible para que el mismo pueda cumplir sus objetivos con los recursos que dispone.

Por otro lado se aprendió a trabajar con herramientas para generar web dinámicas, y el proceso de interacción entre el usuario y el servicio que brinda el servidor.