UNIVERSIDAD NACIONAL DE CÓRDOBA

Facultad de Ciencias Exactas, Físicas y Naturales

Cátedra de Sistemas Operativos II

Sistemas Embebidos

Tarazi Pedro Esequiel

8 de Mayo de 2016

Índice

INTRODUCCION

Un sistema embebido es un sistema de computación diseñado para realizar una o algunas pocas funciones dedicadas. Este tipo de sistema, se ejecuta en una electrónica programable con unos recursos de hardware muy limitados, que por lo general debe reaccionar en tiempo real a sucesos externos al sistema. Algunos ejemplos de sistemas embebidos podrían ser dispositivos como un taxímetro, un sistema de control de acceso, sistema de control de una fotocopiadora, etc. Por lo general, estos sistemas se programan en lenguaje ensamblador del microprocesador incorporado sobre el mismo, o también utilizan lenguajes como C. En este trabajo, usamos la placa desarrollada por Intel, denominada Intel Galileo Gen. 1. Esta placa tiene una arquitectura x86, y combina la tecnología de Intel con el soporte de Arduino.

Otro de los puntos clave de este proyecto, es el uso de un Web Server, que es instalado en la Intel Galileo. Un Web Server es un software que procesa una aplicación del lado del servidor, realizando conexiones con el cliente y generando una respuesta en cualquier lenguaje. El código recibido por el cliente suele ser compilado y ejecutado por un navegador web. La transmisión de datos se realiza, generalmente, bajo el protocolo HTTP.

CGI es una tecnología de la WWW que permite a un cliente solicitar datos de un programa ejecutado en un servidor web. Especifica un estándar para transferir datos entre el cliente y el programa. En una aplicación CGI, el servidor web pasa las solicitudes del cliente a un programa externo, el cual puede estar escrito en cualquier lenguaje que soporte el servidor. Por razones de portabilidad, se suelen usar lenguajes de script. La salida es enviada al cliente en lugar del archivo estático tradicional. Los lenguajes más usados son C, C++, Perl, Java, Visual Basic.

En este documento se detallará lo realizado en el proyecto, junto con los pasos que se llevaron a cabo, para finalmente mostrar los resultados obtenidos.

DESCRIPCION DEL PROBLEMA

El Servicio Meteorológico Nacional le solicita a la Escuela de Ingeniería en Computación de la FCEFyN-UNC, que realice el diseño, desarrollo y testing del sistema de adquisición de datos de Estaciones Meteorológicas Automáticas (AWS). Se solicita que:

1) Se sincronice el registro intermedio entre el Simulador de generación de datos el proceso d la AWS
2) Que se realice un estudio de los distintos Web Servers disponibles para sistemas embebidos y justifique la elección de uno. Instarlo en el SO y ejecución automática al inicio
3) Sobre el servidor, desarrollar una interfaz web simple, con múltiples pestañas, donde cada pestaña debe mostrar, utilizando CGI:
    a. Información sobre recursos del Sistema Embebido.
    b. Get_telemetry: abre en una ventana la última telemetría
    c. Get_datta: abre el total de telemetrías obtenidas
    d. Erase_datta: vacía el buffer de datos obtenidos
    e. Listado de módulos instalados
    f. Formulario que permita subir un archivo al servidor, controlar que sea válido, e instalarlo. Agregar botón de desinstalación.
4) Desarrollar un módulo simple y vacío, que imprima Hello Word al instalarse y Goodbye World al desinstalarse del kernel.

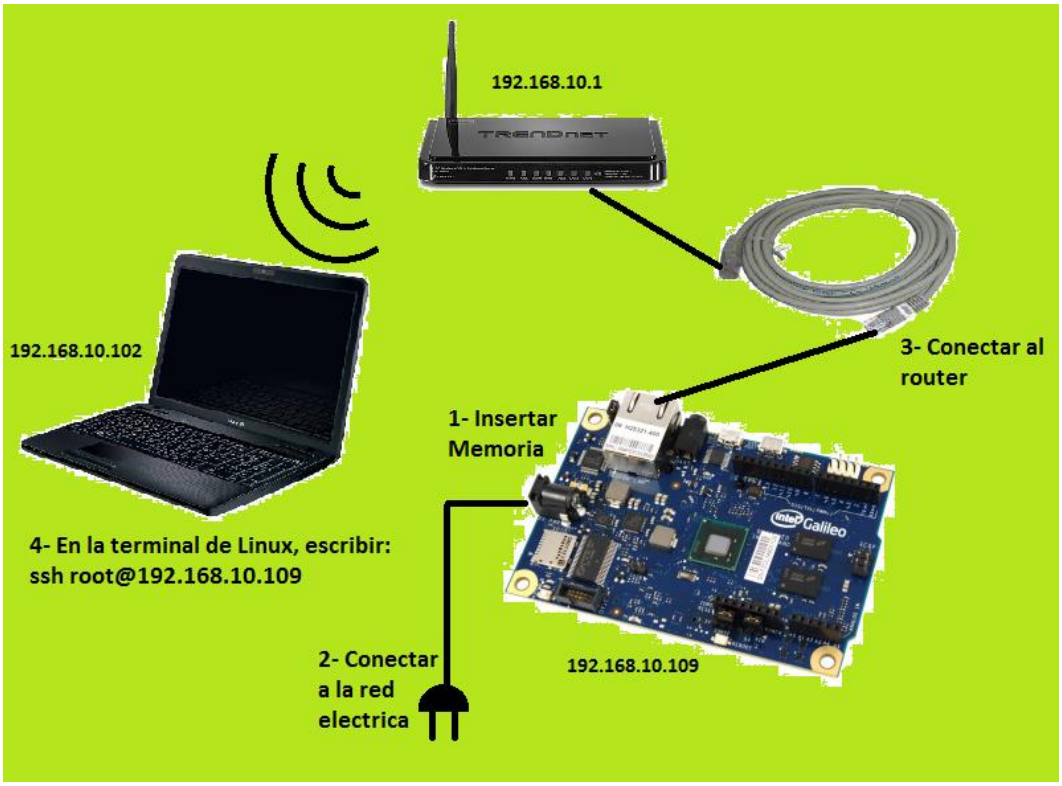
REQUERIMIENTOS Y TAREAS

- Se debe sincronizar el registro intermedio entre el Simulador y la AWS
- WEB SERVER:
    - Se debe investigar, comparar y elegir uno. Justificar la elección.
- Interfaz Web, usando CGI:
    - Debe tener un botón para mostrar información de recursos del sistema embebido.
    - Debe tener un botón que muestre el último registro de telemetría obtenido.
    - Debe tener un botón que muestre todos los registros de telemetría.
    - Debe tener un botón que borre los datos registrados.
    - Debe tener un botón que muestre el listado de módulos instalados.
    - Debe tener un botón que permita subir un archivo al servidor.
        - Controlar que el archivo sea válido.
        - Debe tener un botón para instalar el modulo.
        - Debe tener un botón para remover el modulo.
- Desarrollar un módulo simple que imprima “Hello, World” al instalarse en el kernel, y “Goodbye, World” al desinstalarse del mismo.

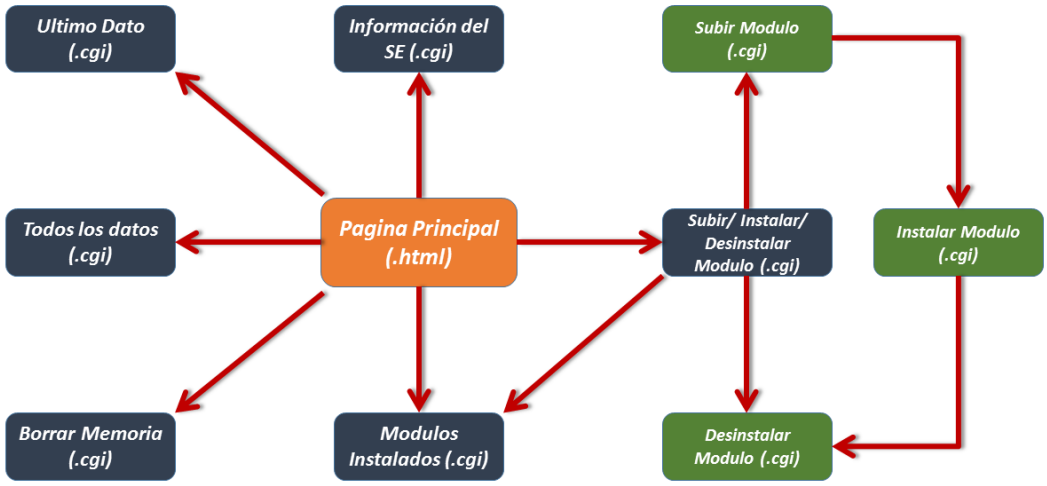
DISEÑO Y DOCUMENTACION

Diseño

A continuación, se muestre la forma de conexión del hardware realizada durante las pruebas del proyecto. Se cuenta con una PC, una placa Intel Galileo Gen.1, con un Router y con un cable de red cruzado:



Al ser un trabajo basado en HTML, el software se fue creando y testeando mediante visualizaciones. La estructura de la página es la siguiente:



Documentación

En este apartado se detallarán paso a paso, dos puntos clave del proyecto: la instalación del Web Server, y la compilación del módulo HelloWorld que se va a instalar en el kernel de Linux.

Existen infinidad de Web Server para Sistemas Embebidos, algunos libres y de código abierto, y otros mediante pago. En este proyecto se priorizaron los de código abierto u Open Source.

Uno de los Web Server más conocido es el Apache HTTP Server, más conocido como Apache. Es el más usado en el mundo. Desarrollado por Apache Software Foundation. Es comúnmente usado en sistemas basados en Unix. Este Web Server fue utilizado para realizar las pruebas en la PC, obteniendo resultados sin inconvenientes. Uno de los problemas de este servidor web es la engorrosa configuración para la utilización del estándar CGI, por lo que se derrochó una gran cantidad de tiempo en la puesta a punto del mismo. Este no puede instalarse en la Intel Galileo ya que la misma no cuenta con los requisitos de hardware que el servidor precisa.

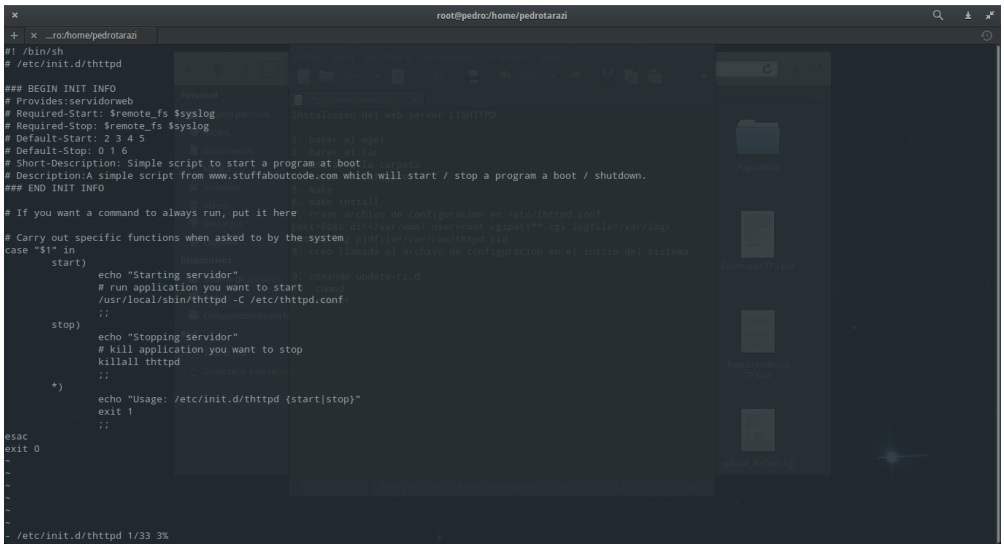
Lighttpd es otro servidor web de código abierto optimizado para sistemas críticos de alta velocidad, siendo seguro y flexible. Con el paso del tiempo, fue ganando popularidad a tal punto de ser uno de los más usados. Suporta interface CGI, FastCGI y SCGI. Acepta lenguajes Python, Perl, Ruby y Lua. Una de las desventajas es que no soporta el envio de grandes archivos CGI y FastCGI.

El Web Server elegido para ejecutarse en la Intel Galileo es el llamado “thttpd”. Es de código abierto, desarrollado por ACME Laboratories. Es pequeño, rápido, simple y portable. Opera en sistemas Unix como FreeBS, SunOS, Solaris, Linux y OSF.

Existen muchísimos más Web Server, como Monkey, BOA, Caddy, Jetty, etc.

A continuación, se detalla la instalación de thttpd en la Intel Galileo Gen.1:

1. Conectar PC, mediante ssh, a la Galileo:
   ssh root@192.168.10.109
   password: root
2. cd /tmp
3. wget http://acme.com/software/thttpd/thttpd-2.27.tar.gz
4. tar -zxvf thttpd-2.27.tar.gz
5. cd thttpd-2.27.tar.gz
6. ./configure
7. make
8. make install
   - En caso de error sobre la no existencia del grupo “www”, escribir:
     groupadd www
   - En caso de error sobre la no existencia de la carpeta “man1”, escribir:
     mkdir /usr/local/man/man1
   - make install
9. Crear archivo de configuración en /etc
   vi /etc/thttpd.conf
10. Agregar la siguiente línea:
    port=8080 dir=/var/www/ user=root cgipat=**.cgi
    logfile=/var/log/thttpd.log pidfile=/var/run/thtpd.pid
    - dir: Se coloca la ruta donde se desea tener el index.html
11. Guardar archivo presionando “ESC”, luego “:x” y posteriormente “ENTER”
12. Crear llamada al archivo de configuración en el inicio del sistema.
    vi /etc/init.d/thttpd
13. Escribir el texto de la imagen:



14. Guardar archivo y salir.
15. Cambiar permisos de ejecución del script de inicio
    chmod ugo+x /etc/init.d/thttpd

16. Actualizar inicio del sistema
    update-rc.d thttpd defaults
17. Reiniciar Intel Galileo
    reboot
18. A continuación, se muestra imagen del proceso iniciado al iniciar el sistema:

```
root pedrotarazi # ssh root@192.168.10.109
The authenticity of host '192.168.10.109 (192.168.10.109)' can't be est
ablished.
ECDSA key fingerprint is 69:8f:cb:87:dc:fd:96:8f:a6:b6:c4:64:de:b4:53:9
0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.10.109' (ECDSA) to the list of know
n hosts.
root@galileo:~# ps | grep thttpd
  187 root      2872 S    /usr/local/sbin/thttpd -C /etc/thttpd.conf
  232 root      2508 S    grep thttpd
root@galileo:~#
```

Como se puede ver, el proceso 187 corresponde al Web Sever thttpd y se ejecuta apenas iniciado el sistema.

Por otro lado, tenemos la creación del binario “HelloWorld.ko”, mediante la compilación de un código fuente en lenguaje C. La imagen de Linux que provee Intel está basada en el proyecto Yocto, que, en pocas palabras, es un proyecto Open Source que provee templates, herramientas y métodos que ayudan que permiten generar imágenes de sistemas basados en Linux para sistemas embebidos con las características que se necesiten. Esta imagen no posee los headers del kernel, por lo que no se puede compilar el modulo dentro de la placa, por lo que este proceso se debe realizar de otra manera. La creación del binario para instalar en el kernel de la Intel Galileo, es un proceso largo, complicado, y que requiere mucho tiempo. Por eso, esto se realizó en una sola PC, junto a otros compañeros, por lo que el binario es el mismo al de varios compañeros. A continuación, se detalla el paso a paso para la creación del mismo, y en el anexo se coloca el código fuente del driver creado por mí, junto con su makefile, a pesar de no ser el driver que se instala en esta placa:

1. Se deben instalar todas las builds tolos en el sistema en que se llevará a cabo la compilación. En Linux, esto es:
   sudo apt-get install build-essentials
2. Descargar el código fuente del proyecto yocto-galileo desde el enlace “Fuentes” en la sección “Archivos de cumplimiento GPL” en la página de descargas de Intel. http://software.intel.com/es-es/iot/hardware/galileo/downloads
3. Descomprimir los fuentes en el home del entorno en que se llevara a cabo la compilación. Esto no debe realizarse en particiones diferentes a la que contiene la raíz del sistema, ya que generará errores en los enlaces simbólicos que contiene el proyecto. Ingresar a la carpeta que se descomprime usando el

comando:
cd nombre_carpeta_descomprimida

4. Dentro de la carpeta “poky”, en la carpeta “meta-intel-galileo/récipes-kernel/”, crear un directorio con el nombre del módulo y, dentro del mismo, copiar el código fuente del módulo junto al archivo de configuración de extensión “.bb” que utilizar bitbake para llevar a cabo la compilación. Bajo el directorio “meta-skeleton/récipes-kernel/”, dentro de la carpeta “poky”, se encuentra desarrollado un módulo “hello-world” junto al archivo de configuración de bitbake correspondiente que puede ser usado como guía.
5. Al directorio que se creó con el nombre del módulo y que contiene el código fuente del mismo y archivo de configuración bitbake, copiarlo dentro del directorio “build_galileo” que se encuentra en la carpeta que se genera al descomprimir el proyecto junto a la carpeta “poky”.
6. Dentro de la carpeta “poky”, ejecutar:
   “source oe-init-build-env../build-galileo/”
   Este comando nos cambia de posición al directorio “build_galileo” y crea variables de entorno que serán usadas durante el proceso de compilación.
7. Ejecutar:
   “bitbake nombre_de_modulo”
   El comando anterior comienza con el proceso, el cual consisto en la descarga de todos los fuentes y dependencias necesarios para generar un build del proyecto completo, para luego automáticamente generar un build del módulo en cuestión. Cable aclara que el proceso, dependiendo de la velocidad de conexión a la red y del hardware del sistema en que se compila, puede tomar más de una o dos horas, y que se requieren, como mínimo, 10Gb de espacio libre en disco.
8. Una vez finalizado el proceso, dentro de build_galileo y bajo la ruta “tmp/work/quark-poky-linux/hello-mod/0.1-r0/”, se encuentra el archivo con extensión “.ko” del módulo compilado.

IMPLEMENTACION Y RESULTADOS

Debido a la imposibilidad de crear la imagen del sistema, a continuación, se detalla mediante imágenes, cada una de las pestañas pedidas en el enunciado, cumpliendo así con los requerimientos funcionales del sistema:

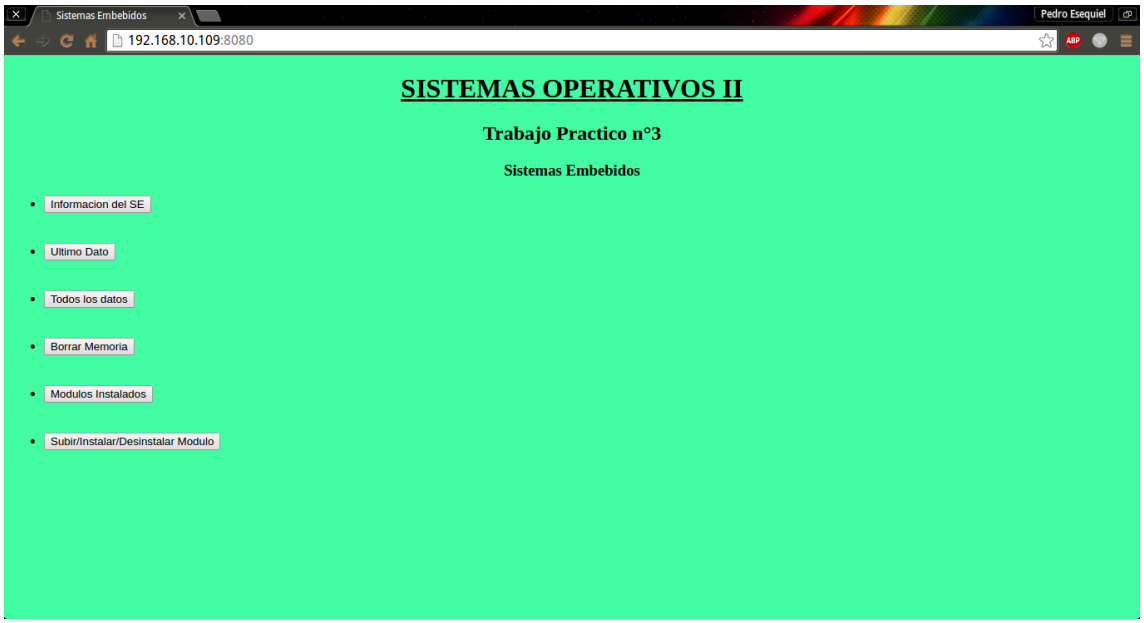


Ilustración 1 - Principal

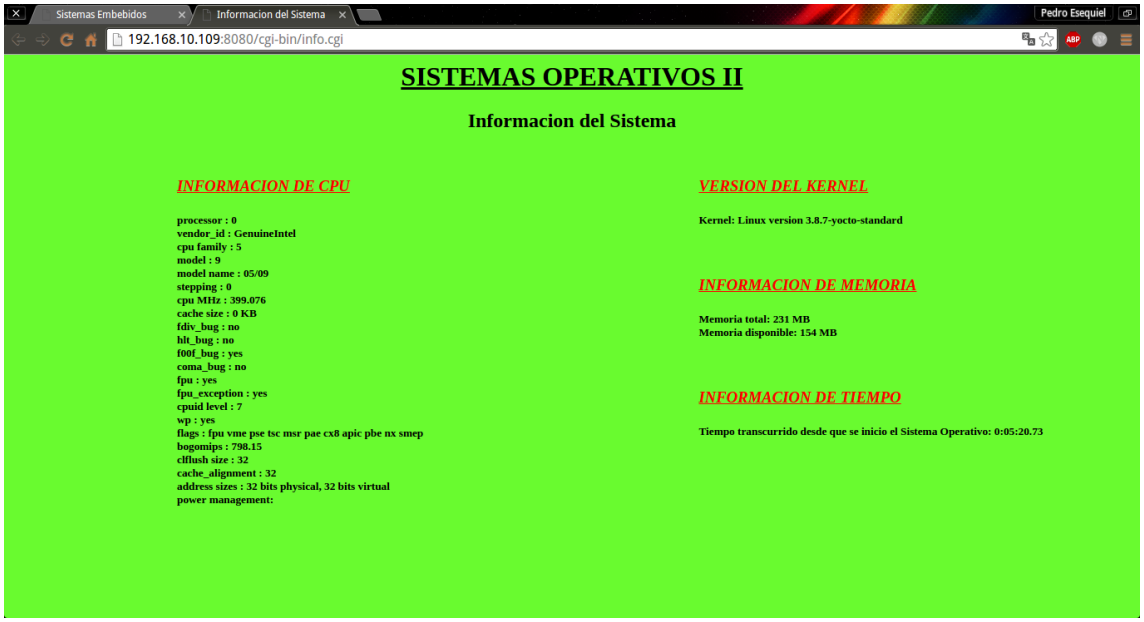


Ilustración 2 – Información del Sistema

Ilustración 3 – Get_Telemetry

Ilustración 4 – Get_Datta

Ilustración 5 – Erase_datta

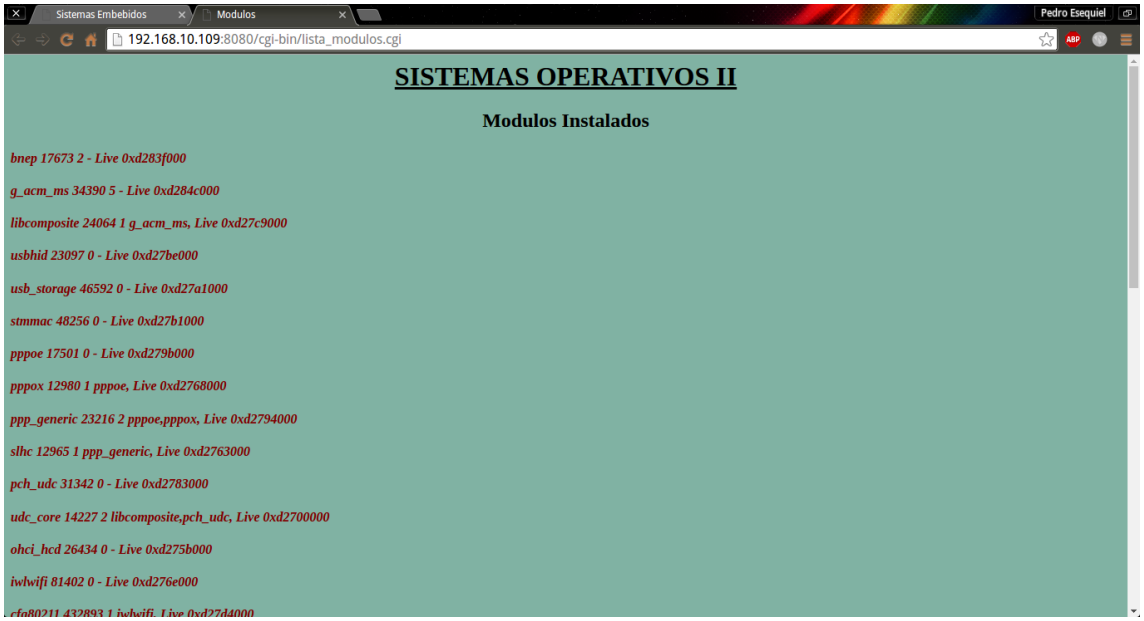


Ilustración 6 – Lista de Modulos instalados

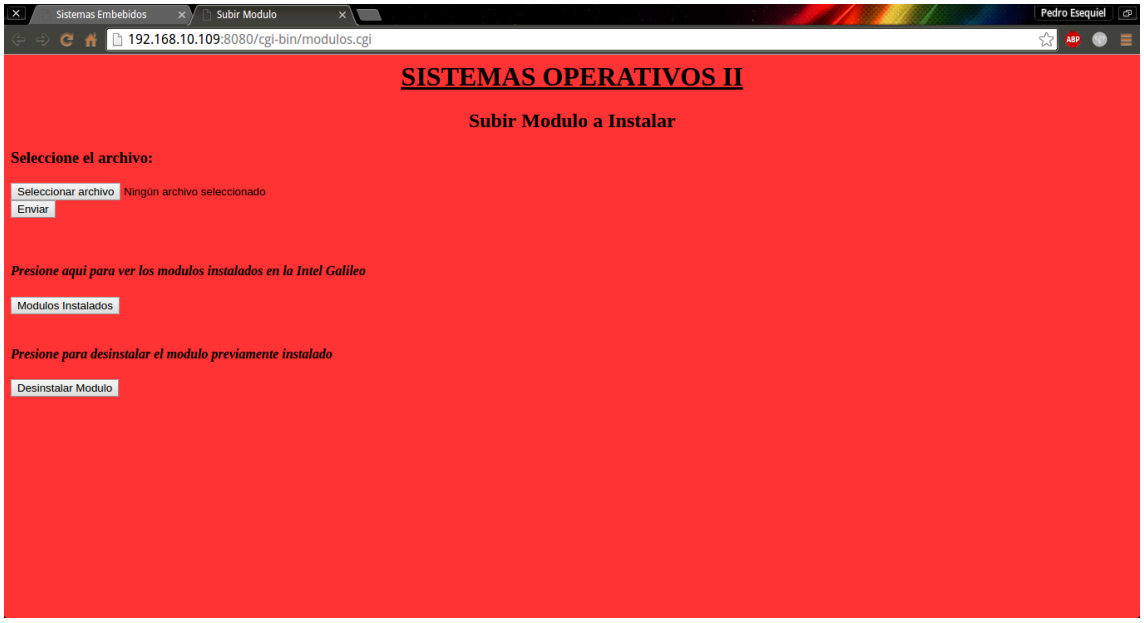


Ilustración 7 – Pestaña para subir un modulo. Tiene opciones para mostrar los modulos ya instalados, y para desinstalar el modulo si ya fue instalado previamente.

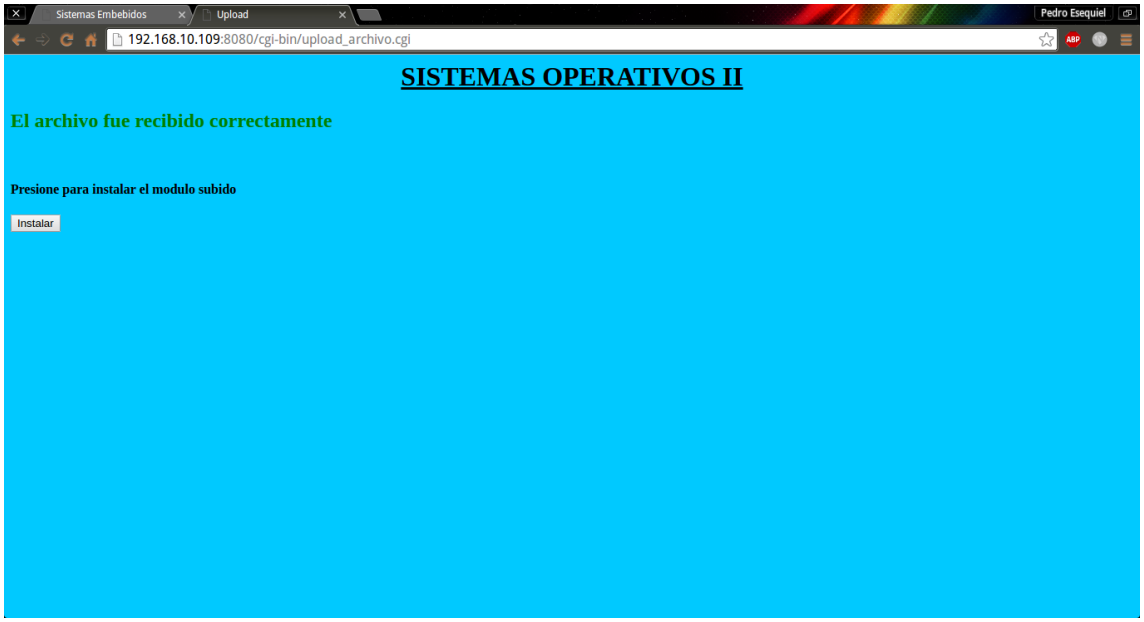


Ilustración 8 – Archivo subido y recibido correctamente. Verificada la extensión.

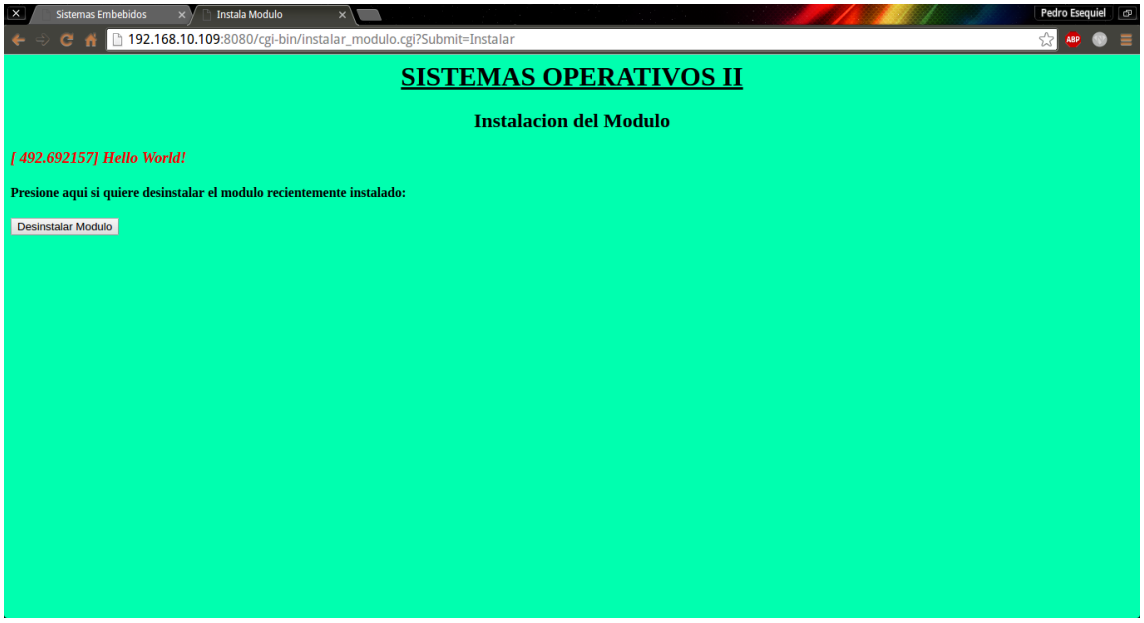


Ilustración 9 – Modulo Instalado. Muestra el mensaje "Hello, World". Tiene la opción de desinstalarlo.

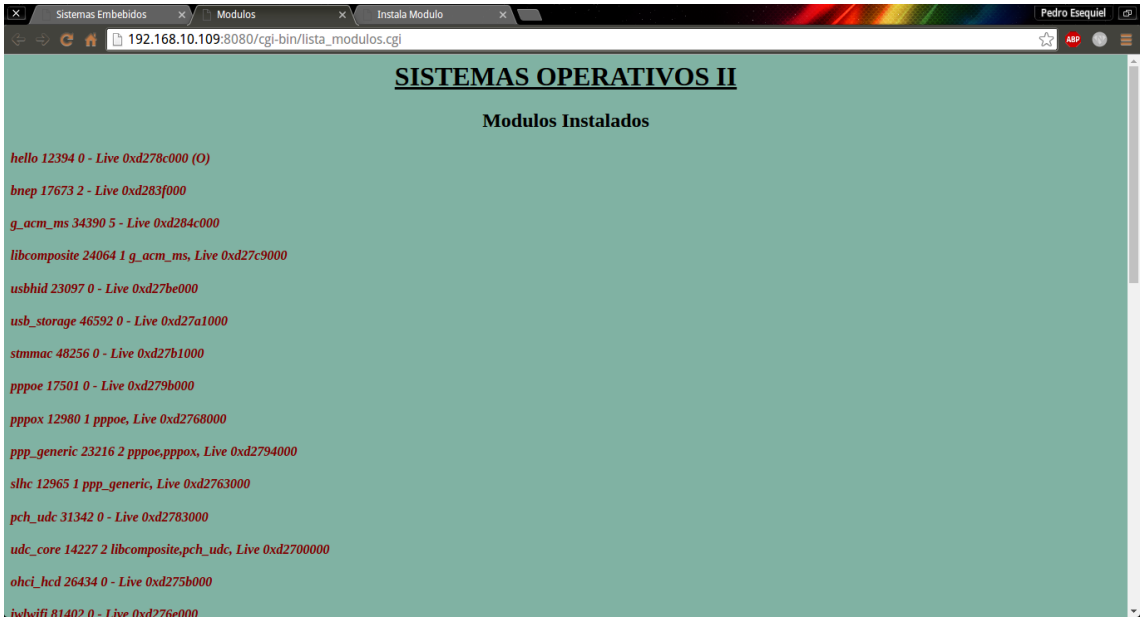


Ilustración 10 – Lista de Módulos que verifica que se instaló correctamente

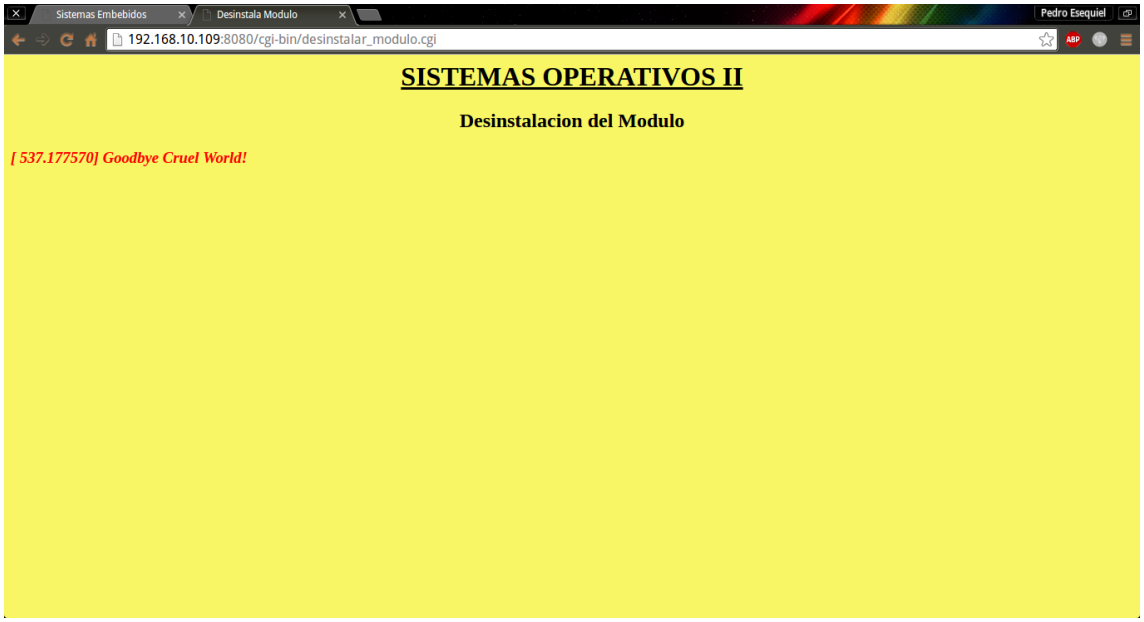


Ilustración 11 – Modulo desinstalado. Muestra el mensaje "Goodbye World".

Ilustración 12 – Verificación de que el modulo fue desinstalado del kernel.

Como se puede ver, se cumplió con todos los requerimientos. Además, se muestra que el acceso se realiza mediante el navegador web, en el cual, en su barra de direcciones, se escribe la direccion IP del Web Server para acceder al mismo.

CONCLUSIONES

Durante la realización de este proyecto, se pudo conocer más a fondo el funcionamiento de un servidor web, junto con la configuración del mismo para el manejo de aplicaciones CGI. Entendí como funciona una aplicación CGI y su comunicación con el Web Server.

Además, se comprendió el manejo de semáforos y su necesidad en este proyecto, y en cualquier otro proyecto que contenga variables o recursos compartidos por dos o más procesos. Son una gran herramienta para la sincronización de los mismos. Una desventaja de este mecanismo de sincronización es la gran cantidad de instrucciones distribuida por todo el código, lo que hace complicado mantenimiento y corrección.

ANEXO

Codigo del Driver HelloWorld:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

static int __init hello_init(void)
{
	printk(KERN_INFO "Hello, World!\n");
	return 0;
}

static void __exit hello_exit(void)
{
	printk(KERN_INFO "Goodbye, World!\n");
}

module_init(hello_init);

module_exit(hello_exit);
```

Makefile:

```
OBJECTS := driver_TP3.o

obj-m := driver_TP3.o
fifo-objs := $(OBJECTS)

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD)
	modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD)
	clean
```