

**Universidad Nacional de Córdoba.**  
**Facultad de Ciencias Exactas, Físicas y**  
**Naturales.**



- SISTEMAS OPERATIVOS II -

**Trabajo Práctico N° 3:**  
**- *Sistemas Embebidos (FreeRTOS)* -**

Alumno:

- GOLDMAN, Nicolás.

Fecha de entrega:

06 / 06 / 2019

# 1. INDICE

<b>INDICE</b>	<b>1</b>
<b>Introducción</b>	<b>2</b>
<b>Objetivo</b>	<b>2</b>
<b>Desarrollo</b>	<b>2</b>
<b>Implementación</b>	<b>3</b>
<b>Resultados de las pantallas</b>	<b>9</b>
<b>Conclusión</b>	<b>10</b>

# 1. Introducción

En la intersección de Software, Hardware y Comunicaciones nos podemos encontrar a los Sistemas Embebidos. Los mismos son sistemas que, si bien su definición varía con la literatura, se pueden definir como computadoras de uso específico, es decir, computadoras con requerimientos de hardware, software y comunicaciones bien definidos. Es por esto la importancia que poseen este tipo de sistemas para los Ingenieros en Computación, y que da origen al presente práctico.

## 2. Objetivo

El objetivo del presente trabajo práctico es que el estudiante sea capaz de diseñar una aplicación para un sistema embebido.

## 3. Desarrollo

Se pide que sobre un sistema embebido tipo Raspberry Pi o similar, que posea MMU, se desarrolle una aplicación para un sistema embebido, que debe cumplir con:

1. Instalar un sistema operativo GNU/Linux en el sistema embebido, justificar la selección del SO.
2. Realizar un estudio de las distintas implementaciones web servers disponibles para sistemas embebidos, realice una comparación y justifique la selección de uno de ellos, que deberá instalar en el sistema operativo, y que deberá ejecutarse automáticamente cada vez que este se reinicia el sistema embebido.
3. Sobre el servidor web, debe desarrollarse una interfaz web simple, con múltiples páginas, donde cada una deben mostrar diferentes vista; utilizando HTML, CGI y Pearl o C.
4. Desarrollar un módulo (driver) simple y vacío, que solo imprime “Hello World” al instalarse y “Goodbye World” al ser removido del kernel. Este será el módulo que se debe instalar en el punto anterior.

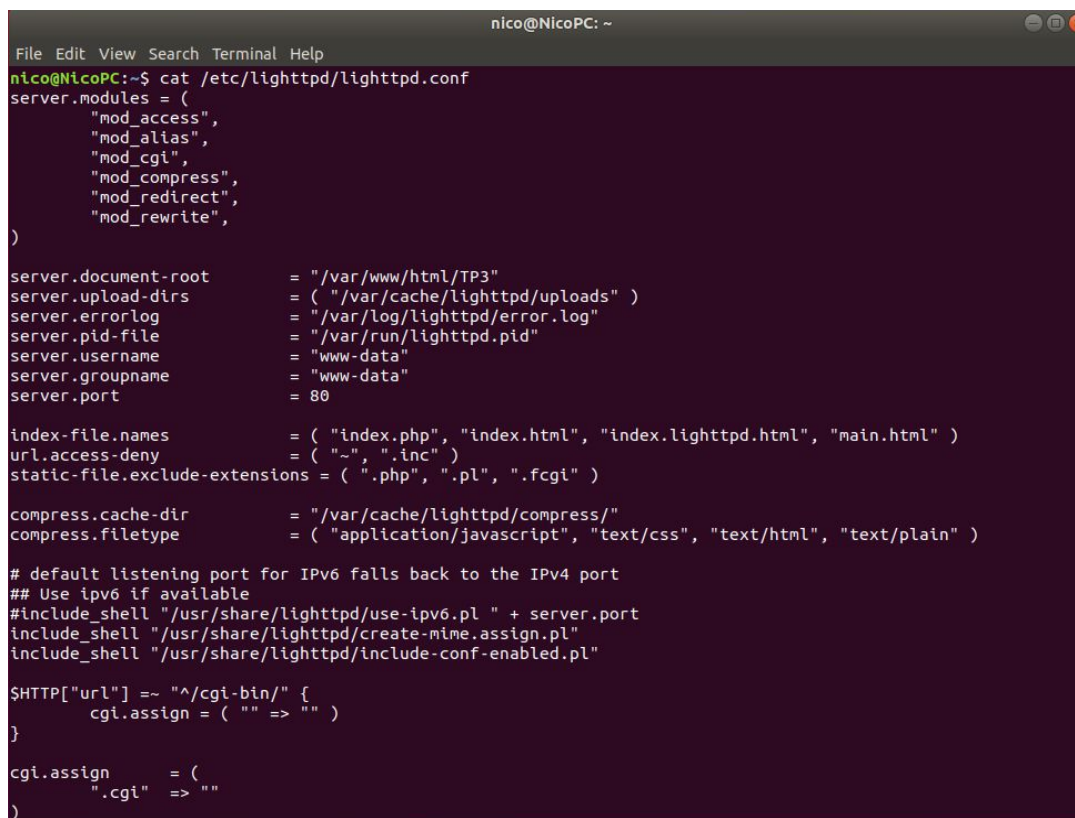
## 4. Implementación

Para el desarrollo del trabajo práctico 3 de la materia, se implementaron diferentes recursos y conocimientos aprendidos durante lo largo de la carrera y algunos otros investigados a medida que iban surgiendo dudas y necesidades.

Para comenzar se implementaron las vistas simples que requiere el trabajo haciendo uso de HTML y CSS. Las vistas creadas utilizando ambos, fueron la pantalla principal y aquellas que no requerían hacer uso de CGI.

Una vez creadas las primeras páginas, se decidió implementar el servidor web. Investigando entre algunos de ellos, Lighttpd fue el elegido debido a que proporciona una forma relativamente simple de configurar un servidor web sin cargar demasiado la capacidad procesamiento que es limitada. Es adecuada para una Raspberry Pi como servidor web ligero para un sitio web personal que no sea muy pesado ni requiera tantos recursos.

Se instaló el servidor web y se configuró de la siguiente manera:



```
nico@NicoPC: ~
File Edit View Search Terminal Help
nico@NicoPC:~$ cat /etc/lighttpd/lighttpd.conf
server.modules = (
    "mod_access",
    "mod_alias",
    "mod_cgi",
    "mod_compress",
    "mod_redirect",
    "mod_rewrite",
)

server.document-root    = "/var/www/html/TP3"
server.upload-dirs       = ( "/var/cache/lighttpd/uploads" )
server.errorlog          = "/var/log/lighttpd/error.log"
server.pid-file          = "/var/run/lighttpd.pid"
server.username          = "www-data"
server.groupname         = "www-data"
server.port              = 80

index-file.names         = ( "index.php", "index.html", "index.lighttpd.html", "main.html" )
url.access-deny          = ( "~", ".inc" )
static-file.exclude-extensions = ( ".php", ".pl", ".fcgi" )

compress.cache-dir       = "/var/cache/lighttpd/compress/"
compress.filetype        = ( "application/javascript", "text/css", "text/html", "text/plain" )

# default listening port for IPv6 falls back to the IPv4 port
## Use ipv6 if available
#include_shell "/usr/share/lighttpd/use-ipv6.pl " + server.port
include_shell "/usr/share/lighttpd/create-mime.assign.pl"
include_shell "/usr/share/lighttpd/include-conf-enabled.pl"

$HTTP["url"] =~ "^/cgi-bin/" {
    cgi.assign = ( "" => "" )
}

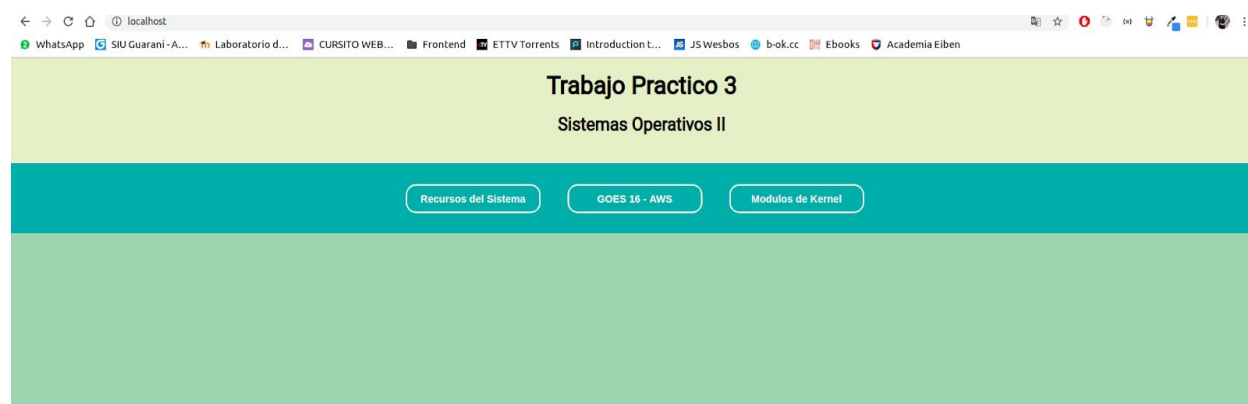
cgi.assign = (
    ".cgi" => ""
)
```

En la imagen de arriba, podemos ver el archivo de configuraciones del servidor elegido. El mismo se modificó para poder tener soporte a archivos .cgi que era una de las consignas más importantes del trabajo. Para ello se agregó la línea de **“mod\_cgi”** en donde se especifican todos los módulos que aceptará nuestro servidor.

También se modificó el **document-root**, dejándolo en el directorio `/var/www/html/TP3` que contiene todos los archivos html, css, cgi y ko nombrados en la consigna.

Por último se agregaron las últimas líneas que hacen de Handler ante las consultas que contengan un archivo cgi para ser ejecutado.

Una vez realizada esta configuración y guardado los archivos en el directorio correspondiente, en un navegador se ingresó al localhost de la pc y se comprobó el correcto funcionamiento del servidor.



Vista principal de la interfaz web

Cuando se comprobó el funcionamiento del servidor, se comenzaron a desarrollar los primeros archivos en el lenguaje C cuyos requerimientos se detallaron en la consigna; para luego, pasar todos ellos a la raspberry.

Se explican en diferentes secciones los códigos y comportamientos de cada uno de ellos.

- **Recursos del sistema:**

El código con la solución se adjunta en la entrega del trabajo. Los puntos principales para explicar son las formas de obtener cada uno de los recursos.

```
/* Get the uptime from /proc/uptime */
FILE *fp;
fp = fopen("/proc/uptime", "r");
char buff[3500];
fread(buff, 3500, 1, fp);
fclose(fp);

long uptime;
char *ptr;
uptime = strtol(buff, &ptr, 10);
```

```
/* Get actual date and time */
time_t t = time(NULL);
struct tm *tm = localtime(&t);
char today[64];
assert(strftime(today, sizeof(today), "%c", tm));
```

```
//Memory stats:
printf("<p><b>Memory Stats: </b></p>\n");

char *cmd = "grep Mem /proc/meminfo";
FILE *cmdfile = popen(cmd, "r");
char str1[20], str2[20], str3[20], str4[20], str5[20], str6[20];
long memTot, memFree, memAv;

fscanf(cmdfile, "%s %ld %s %s %ld %s", str1, &memTot, str2, str3, &memFree, str4, str5, &memAv, str6);
printf("<ul><li> %s %ld %s</li> <li> %s %ld %s</li> <li> %s %ld %s</li></ul>", str1, &memTot, str2, str3, &memFree, str4, str5, &memAv, str6);
pclose(cmdfile);

//CPU stats:
printf("<p><b>CPU Stats: </b></p>\n");
char *cmd2 = "cat /proc/cpuinfo | grep \"model name\"";
FILE *cmdfile2 = popen(cmd2, "r");

char *cmd3 = "cat /proc/top";
FILE *cmdfile3 = popen(cmd3, "r");

char str7[20], str8[20], str9[20], str10[20], str11[20], str12[20], str13[20], str14[20], str15[20];
fscanf(cmdfile2, "%s %s %s %s %s %s %s", str7, str8, str9, str10, str11, str12, str13, str14, str15);

char oneMin[20];
fscanf(cmdfile3, "%s id.", oneMin);
printf("<ul><li>Model Name: %s %s %s %s %s %s %s</li><li>Load Average: %d</li></ul>", str9, str10, str11, str12, str13, str14, str15, oneMin);
```

Como se puede ver en las imágenes de arriba se utilizaron diferentes mecanismos para obtener los datos sobre recursos que se solicitan sobre el sistema embebido.

Para el caso del uptime se lo hizo mediante la función *fopen* que guarda el output del comando que especificamos en un archivo indicado, el cual se lee y almacena en un buffer para luego pasarlo como valor de una variable de entero o long.

En cuanto a la fecha y hora actual se utilizó la función strftime la cual formatea el tiempo representado en la estructura timeptr de la forma en que querramos (%c en este caso → ej: *Sun Aug 19 02:56:02 2012*) y lo almacena en una variable de tipo “string” (char[]). La función *assert* se encarga de comprobar que la anterior explicada se ejecute sin errores.

Por último se utilizó el mismo método para las funciones de **memoria** y **cpu stats**. Ambas hacen uso de **popen** la cual abre un proceso para input o output y almacena el

valor del mismo en un FILE que pasamos como argumento. Luego se encarga de leer dicho archivo y *parsear* su contenido en variables que luego imprimimos mediante html. Los comandos son **cat** en diferentes archivos del /proc.

- **Modulos de kernel:**

En esta sección hay 2 paginas diferentes. La primera es la que nos muestra todos los módulos que tenemos cargados en nuestro kernel; y la segunda es la que tiene el formulario para cargar (o quitar) un nuevo módulo.

La página que muestra los módulos instanciados en el kernel de nuestro sistema operativo es simplemente el comando **lsmod** que ejecutamos mediante una llamada a *system()* y guardamos en un archivo. De la misma forma que se explicó en la sección anterior se realiza el manejo de dicho archivo para guardar su contenido en una variable *buffer* y luego parsearlo para ubicarlo en tablas de HTML.

Para el parseo del contenido del buffer se utilizaron principalmente las funciones **strtok\_r** y **strcat**. La primera la usamos para reconocer el espacio entre una palabra y otra que en el caso del output de lsmod significaría la diferencia entre el nombre, tamaño y las instancias. La función recibe como parámetro un “token” que representa cada línea de nuestro archivo (fue separado anteriormente mediante strtok\_r y delimitador “\n” en cambio del espacio “ ”) .

Una vez que la función devuelve NULL quiere decir que llegamos al final de la línea y la próxima vez entraremos con la próxima línea para poder realizar la separación entre las palabras.

La segunda página es la que nos permite cargar el módulo de kernel como se pedía en la consigna. La misma es un simple formulario de HTML que nos permite elegir un módulo que tengamos en algún directorio de nuestro ubuntu para cargarlo en el kernel. Cuando elegimos dicho archivo, el navegador llama a un cgi encargado de comprobarlo y cargarlo mediante el comando **insmod**.

Una vez cargado podemos volver a la página de los módulos instanciados, y comprobar que se cargó correctamente el módulo; al mismo tiempo tenemos un botón que nos permite eliminarlo del kernel haciendo uso del **rmmod**.

- **Archivos del GOES16 (en aws s3):**

En este último caso y requerimiento de la consigna, se tiene otro llamado a sistema mediante un comando que nos permite conectarnos al servicio de storage de AWS y buscar los archivos ubicados en dicho s3 correspondientes al producto **ABI-L2-CMIPF** en el **canal 13**.

```
/* Guardo las respuestas del formulario en variables */
char * formResponse;
int year, day;
bool valid = true;

formResponse = getenv("QUERY_STRING");

if(formResponse == NULL){
    printf("<p>Error! Error in passing data from form to script.</p>");
    valid = false;
} else sscanf(formResponse, "doi=%d&anio=%d", &day, &year);

/* Busco los archivos de la consulta */
char * command;
command = calloc(256, sizeof(char));
strcpy(command, "echo nein");
snprintf(command, 250, "aws s3 --no-sign-request ls --recursive noaa-goes16/ABI-L2-CMIPF/%d/%03d/ | grep M3C13 > list.txt", year, day);
//printf("<p>%s</p>", command);

system(command);
free(command);

char buffer[50000];
FILE *fp;
fp = fopen("list.txt", "r");
fread(buffer, 50000, 1, fp);
fclose(fp);
system("rm list.txt");
```

```
/* Print statics */
printf("<div class = \"other-section\">\n");
printf("<p>El <b>año</b> seleccionado: %d</p>", year);
printf("<p>El <b>DOI</b> seleccionado: %d</p>", day);
printf("</div>\n");

/* Print table with scans */
char * token;
token = strtok(buffer, "\n");

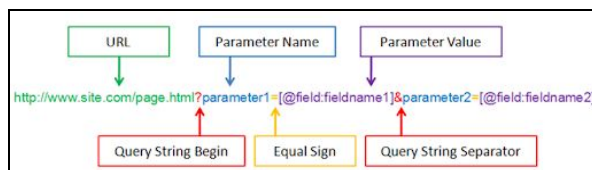
int odd = 1;
printf("<div class = \"table-goes\">\n");
printf("<p style='font-size:140%; background-color:#e7f0c3; margin:0; padding:12; color:black; width: 1100px; text-align: center; border-left: 2px solid #03d'> Channel 13 Scans </b></p>", day, year);

while(token != NULL){
    if(odd%2 == 0){
        printf("<p style='margin:0; padding:8; background-color:#ccedd2; min-width: 1108px; border-left: 2px solid; border-right: 2px solid;'>\n");
    }
    else{
        printf("<p style='margin:0; padding:8; background-color:#a7e9af; min-width: 1108px; border-left: 2px solid; border-right: 2px solid;'>\n");
    }
    printf("%s", token);
    printf("</p>\n");
    token = strtok(NULL, "\n");
    odd++;
}

printf("<p style = 'margin:0px; height: 0px; border-left: 2px solid;min-width: 1124px; border-right: 2px solid; border-bottom: 2px solid;'> </p>");
printf("</div>\n");
```



En el código C de esta página tenemos algunas partes importantes como la línea del **getenv("QUERY\_STRING")**.



Como se puede ver en la imagen de arriba, el Query String es la parte de una dirección URL que nos trae la respuesta a algún formulario con el método GET.

Usamos el método *getenv()* para acceder a cualquiera de las variables de entorno CGI, en este caso el "QUERY\_STRING" explicado anteriormente.

192.168.43.58/pi/cgi-bin/goes16.cgi?doi=123&anio=2018

URL después de submit al formulario

Una vez que obtenemos *doi* y *anio* (como muestra la imagen de arriba), lo parseamos mediante la función **sscanf** la cual busca los strings que especificamos y almacena en variables lo que especifiquemos en símbolos "%".

En este punto se cuenta con los datos pasados por formulario almacenados en variables de enteros, y se continúa con el guardado del output de la consulta de AWS en un archivo llamado *list.txt*.

Para poder realizar la impresión en HTML, tenemos que analizar dicho archivo de texto haciendo uso de un buffer para guardar todos sus caracteres e imprimirlos de la misma forma que se explicó para los módulos mediante el método **strtok()**.

## 5. Resultados de las pantallas

The screenshot shows a web browser window with the URL `192.168.43.58/pi/cgi-bin/recursos.cgi?7`. The page title is 'Recursos'. It displays system status information:

- Uptime del sistema: 0 days, 01:04:51
- Fecha y hora actual: Tue Feb 18 12:45:17 2020
- Memoria disponible:
  - MemTotal: 948304 kB
  - MemFree: 688420 kB
  - MemAvailable: 811580 kB
- Procesador:
  - Modelo: ARMv7 Processor rev 4 (v7r)
  - Promedio de carga (promedio de tareas en cola de ejecución en el ultimo minuto): 0.00

At the bottom, there is a 'Back to main' button.

Pantalla de Recursos del Sistema (sobre la raspi)

The screenshot shows a web browser window with the URL `192.168.43.58/pi/cgi-bin/goes16.cgi?do=123&anio=2018`. The page title is 'Consultas GOES:'. It displays the selected year '2018' and DOI '123'. Below this, there is a table titled '[123/2018] - Channel 13 Scans' showing a list of scan records with columns for date, time, and scan ID.

[123/2018] - Channel 13 Scans	
2018-05-02 21:11:45	27544662 ABH2-CMIPF/2018/123/00/OR_ABH2-CMIPF-M3C13_G16_s20181230000409_e20181230011187_c20181230011259.nc
2018-05-02 21:26:45	27531564 ABH2-CMIPF/2018/123/00/OR_ABH2-CMIPF-M3C13_G16_s20181230015409_e20181230026187_c20181230026261.nc
2018-05-02 21:41:45	27513359 ABH2-CMIPF/2018/123/00/OR_ABH2-CMIPF-M3C13_G16_s20181230030409_e20181230041187_c20181230041260.nc
2018-05-02 21:56:44	27496054 ABH2-CMIPF/2018/123/00/OR_ABH2-CMIPF-M3C13_G16_s20181230045409_e20181230056188_c20181230056263.nc
2018-05-02 22:11:42	27477304 ABH2-CMIPF/2018/123/01/OR_ABH2-CMIPF-M3C13_G16_s20181230100409_e20181230111187_c20181230111259.nc
2018-05-02 22:26:43	27462221 ABH2-CMIPF/2018/123/01/OR_ABH2-CMIPF-M3C13_G16_s20181230115409_e20181230126187_c20181230126260.nc
2018-05-02 22:41:43	27449047 ABH2-CMIPF/2018/123/01/OR_ABH2-CMIPF-M3C13_G16_s20181230130409_e20181230141187_c20181230141262.nc
2018-05-02 22:56:45	27445254 ABH2-CMIPF/2018/123/01/OR_ABH2-CMIPF-M3C13_G16_s20181230145409_e20181230156188_c20181230156264.nc
2018-05-02 23:11:46	27433332 ABH2-CMIPF/2018/123/02/OR_ABH2-CMIPF-M3C13_G16_s20181230200409_e20181230211188_c20181230211256.nc
2018-05-02 23:26:59	27423833 ABH2-CMIPF/2018/123/02/OR_ABH2-CMIPF-M3C13_G16_s20181230215409_e20181230226188_c20181230226262.nc
2018-05-02 23:41:45	27418264 ABH2-CMIPF/2018/123/02/OR_ABH2-CMIPF-M3C13_G16_s20181230230409_e20181230241188_c20181230241286.nc
2018-05-02 23:56:47	27413164 ABH2-CMIPF/2018/123/02/OR_ABH2-CMIPF-M3C13_G16_s20181230245409_e20181230256188_c20181230256262.nc
2018-05-03 00:11:49	27400172 ABH2-CMIPF/2018/123/03/OR_ABH2-CMIPF-M3C13_G16_s20181230300409_e20181230311188_c20181230311260.nc

Pantalla de consulta al S3 de Amazon

The screenshot shows a web browser window with the URL `192.168.43.58/pi/cgi-bin/modules.cgi?`. The page title is 'Modulos cargados en el sistema:'. It displays a table with the following data:

Module	Size	Instances	Used by information
hello_module	16384	0	
fuse	110592	3	
rcomm	49152	4	
bnep	20480	2	
hci_uart	40960	1	
btbcm	16384	1	hci_uart
serdev	20480	1	hci_uart
bluetooth	385024	29	hci_uart, bnep, btbcm, rcomm
ecdh_generic	28672	1	bluetooth
brcmfmac	311296	0	
brcmutil	16384	1	brcmfmac
sha256_generic	20480	0	

Pantalla de consulta de módulos con **lsmod**

## 6. Conclusión

Para concluir con el trabajo práctico 3 de la materia, se puede decir que es una tarea muy importante ya que se pueden aprender conceptos muy importantes que no han sido estudiados antes en ninguna otra materia.

El poder instalar un servidor web en la raspberry nos permite trabajar con un embebido lo cual es importante conocer cómo usarlo, configurarlo, transferir archivos y muchas otras cosas que se realizaron para este trabajo.

Además de la posibilidad de trabajar con un sistema tan potente como la RasPi, nos permitió aprender tanto la teoría como la práctica de un concepto tan utilizado como el CGI.

Se afirmaron algunos conceptos mínimos de desarrollo web visto levemente en las materias de nuestra carrera, pero importantes de conocer con las posibilidades de trabajo en dicha área de la especialidad.