Proyecto 2

“Micro Servidor Web”

Alumno: Santiago Rubén Barboza
Lu:106007
Materia: Redes de Computadoras 2016

Consigna

Empleando el lenguaje de programación C, se desea implementar un servidor web bajo sistema operativo GNU/Linux. El mismo se ejecutará desde la consola y deberá quedar ejecutándose en segundo plano (background) hasta recibir una señal de finalización.

Formato

Para llamar al programa se debe respetar el siguiente Formato:

servidorHTTP [IP][:puerto] [-h]

Donde los parámetros -h, ip y :puerto son opcionales. El significado de las diversas opciones del programa es el siguiente:

- [IP]: número de IP al cuál se debe asociar el programa para escuchar los requerimientos. Por Defecto es localhost.
- [:Puerto]: número de puerto en el cuál se deben decepcionar las peticiones. Por defecto es el puerto 80.
- [-h]: lista las opciones de invocación del programa (sintaxis) y su semántica (con una aclaración sintética en base a la que se brinda en este enunciado).

Asumi que el orden de los operandos no varia, significa que si esta el parametro ip va a ser el primero, que de estar el parametro -h siempre sera el ultimo,etc. La variante de asmir cualquier orden no agregaba complejidad al programa pero si le quitaba mucha facilidad de lectura.

Resolución

En caso de que el programa detecte que se ingreso la opcion -h, muestra por pantalla una ayuda con la manera de especificar los parametros y su significados. Si no lo se intenta setear el ip y puerto donde recibira los requerimientos el servidor. Por defecto estos seran 127.0.0.1 y 80. en caso de que el primer parametro corresponda al puerto se asume que no se ingreso el ip y solo se seta el puerto, sino se setea el ip y en caso de tener otro parametro setea el numero de puerto.

Una vez establecidos el servidor y puerto donde escuchar los equerimientos se crea el socket de Bienvenida. En caso de no poder crearlo se sale reportando un error al crear el socket.

En caso de poder crearlo, se lo liga al ip y puerto especificados, en casao de no poder se cierra reportando al usuario el error. Estando ya creado se pone a escuchar en ese socket a la espera de nuevos requerimientos con una cola de hasta 5 mensajes.

Para cada requerimiento entrante, se lo acepta creando un socket tcp para establecer la conexión con el cliente (En caso de no poder, cierra reportando error). Para poder manejar multiples requerimientos se hace uso de la funcion fork() que le permite crear un proceso hijo para que se encarge de atender el ultimo requerimiento.

El proceso hijo lee el mensaje http enviado por el cliente e intenta corroborar su formato. El formato de un mensaje HTTP es:

“METODO /archivo HTTP/version
{CAMPOi=Valori}*

Contenido”

Si el metodo fuese un POST o un HEAD, que son valores validos de metos, pero que no fueron implementados, se le responde al cliente con un mensaje HTTP con codigo 501 METHOD NOT IMPLEMENTED. En otro caso si el metodo no es GET o el archivo no comienza con “/” o el potocolo NO es un HTTP/1.x responde al cliente con un mensaje 400 BAD REQUEST.

Por ultmo si el formato es el correcto, se saca del nombre del archivo la extension, y dependiendo de ella se llama al metodo que resuelve ese tipo de consulta en particular. En caso de no se especificara el nombre intenta con los nombres por deffecto (index.html, index.htm, index.php) .

En caso de poder procesar la consulta se devuelve la respuesta dentro del contenido de un mensaje http 200 OK. Sino se logra, es por no haber podido halla el archivo, por lo que se contesta con un mensaje http 404 NOT FOUND.

Procedimientos

Para poder implentar el programa de una forma ordenada se armo una bateria importante de metodos. Dichos metodos son:

- **void atender(int socket)**
  Es el metodo a cargo de leer el mensaje e identificar las distintas partes del mismo (metodo, archivo, protocolo,etc). En caso de ser un mensaje no valido, responde al cliente con un mensaje 400 BAD REQUEST. Ademas, luego de consultar por la extension del archivo, determina que metodo de consulta se implementa. Para funcionar requiere que le envien por pararametro un socket TCP previamente conectado con el cliente.
  METODOS DE CONSULTA,
- **void consultarHTML(char* filename,int socket)**
  Intenta abrir el archivo .html **filename** pedido, de poder hacerlo, lee todo su contenido y lo envia dentro de un mensaje http 200 OK.En caso de que no pudiese abrirlo correctamente envia un mensaje http 404 NOT FOUND. Como parametros recibe el nombre del archivo html y el socket TCP previamente conectado con el cliente.
- **void consultarMIME(char* filename,char* ext,int socket)**
  Intenta abrir los archivos tipo .jpg, .gif o .png como archios binarias de lectura, y envia el contenido encapsulado en un mensaje http 200 OK con MIME-Version:1.0 y Content-Type: image/**extension.** En el caso de jpg, la extension combia de jpg a jpeg como esta especificado en la lista de tipos MIME. En caso de no poder abrir correctamente el archivo se le responde al cliente con unmensaje http 404 NOT FOUND. Como parametros recibe el nombre del archivo html, la extension de ese archivo y el socket TCP previamente conectado con el cliente.
- **void ejecutarPHP(char* filename,int socket)**
  Del nombre del archivo, separa el nombre propiamente deicho del pasaje de parametros en el el encabezaso GET. Luego interpreta el .php ejecutando al interprete de php php-cgi. Para poder realizarlo el metodo crea un hijo (a traves

de un fork) que es el responsable de ejecutar el interprete, y se redirecciona la salida del interprete hacia un buffer del padre utilizando al funcion pipe(que conecta un descriptor de escritura con uno de lectura) y la funcion dup2 (que redirecciona la salida hacia un descriptor). Para poder implementarlo me base en una respuesta de un usuario de un post y lo adapte a la situcion en la cual estaba trabajando agregando el pasaje de parametro y con la interrupcion de esperar a la terminacion del proceso hijo mediante un wait. El post nombrado es: http://stackoverflow.com/questions/2605130/redirecting-exec-output-to-a-buffer-or-file .En caso de que el interprete devolviese el estado 404 NOT FOUND, envia al cliente un mensaje http 404 NOT FOUND. Si el .php es intepretado correctamente se encapsula la salida del interprete en un mensaje http 200 OK. Para funcionar requiere que le envien por pararametros el nombre del archivo y un socket TCP previamente conectado con el cliente.

- **void consultarDefaults(int socket)**
  Cuando no se especifica el nombre del archivo consulta si puede abrir algun html por defecto, y en caso de poder hacerlo encapsula su contenido dentro de un mensaje HTPP 200 OK, en caso de no poder abrir ningun html por defecto intenta ejecutar “index.php”. Para funcionar requiere que le envien por pararametro un socket TCP previamente conectado con el cliente

METODOS AUXILIARES

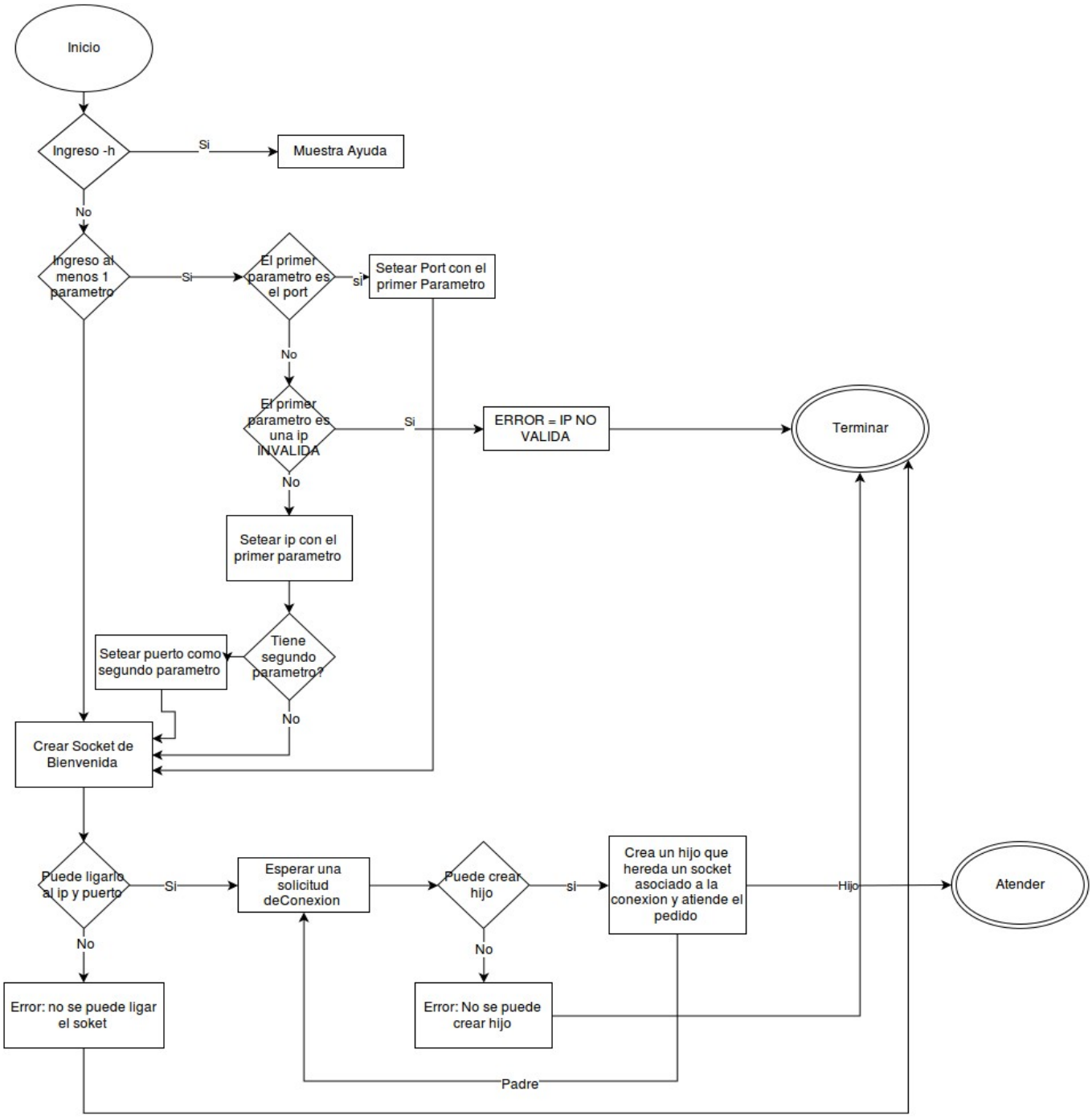
- **FILE* defaultsHTML(FILE * archivo)**
  Intenta archivos por defecto “index.html” y “index.htm”, cuando estos no estan especificados en el mensaje . Devuelve el descriptor del archivo html que pudo abrir. En caso de no poder abrir ninguno de los 2 devuelve NULL. Para funcionar requiere que se le pase por parametro la variable donde se guardara el descriptor de archivo.
- **void salirError(char* error)**
  Realiza un EXIT_FAILURE registrando el error pasado por parameetro
- **char* getExt(char* filename)**
  Recibe un filename y retorna su extension. Si este no tubiera, retorna NULL
- **void setearSignals()**
  Determina el comportamiento del programa ante señales que se pueden atrapar. El programa solo debe finalizar con una señal SIGUSR1 para eso todas las señáles que finalizan el programa son ignoradas (excepto kill) y en caso de recibir una señal SIGUSR1 ejecuta un manejador terminar que solo termina la ejecucion delprograma
- **void imprimirAyuda()**
  Imprime por pantalla la ayuda del programa y lo cierra a continuacion. La razon principal para modularizar esta funcionlaida era hacer ma legible el codigo principal pero como segunda razon esta poder modificar los mensajes de ayuda de una manera sencilla de necesitar hacerlo.

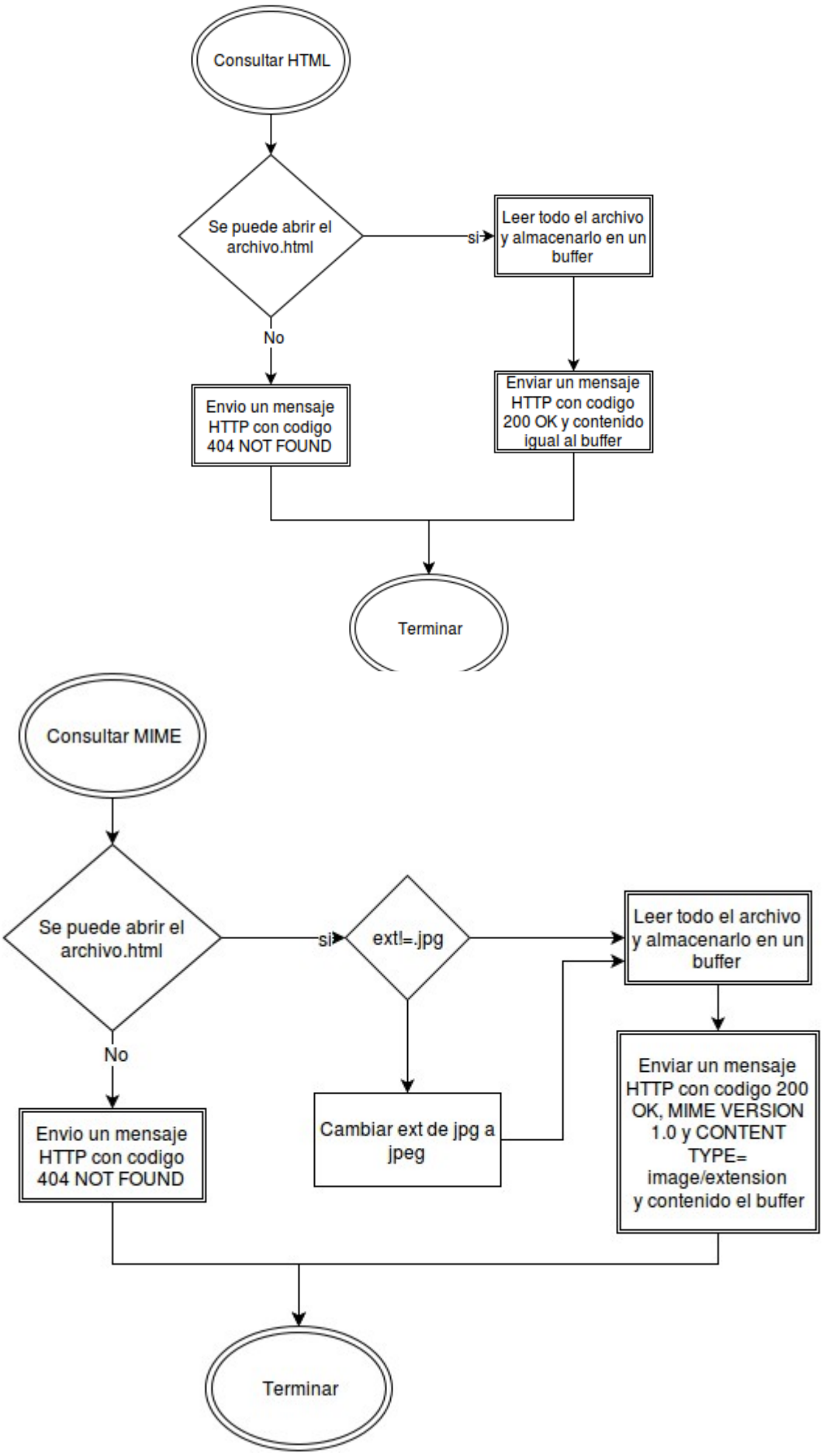
Implementación

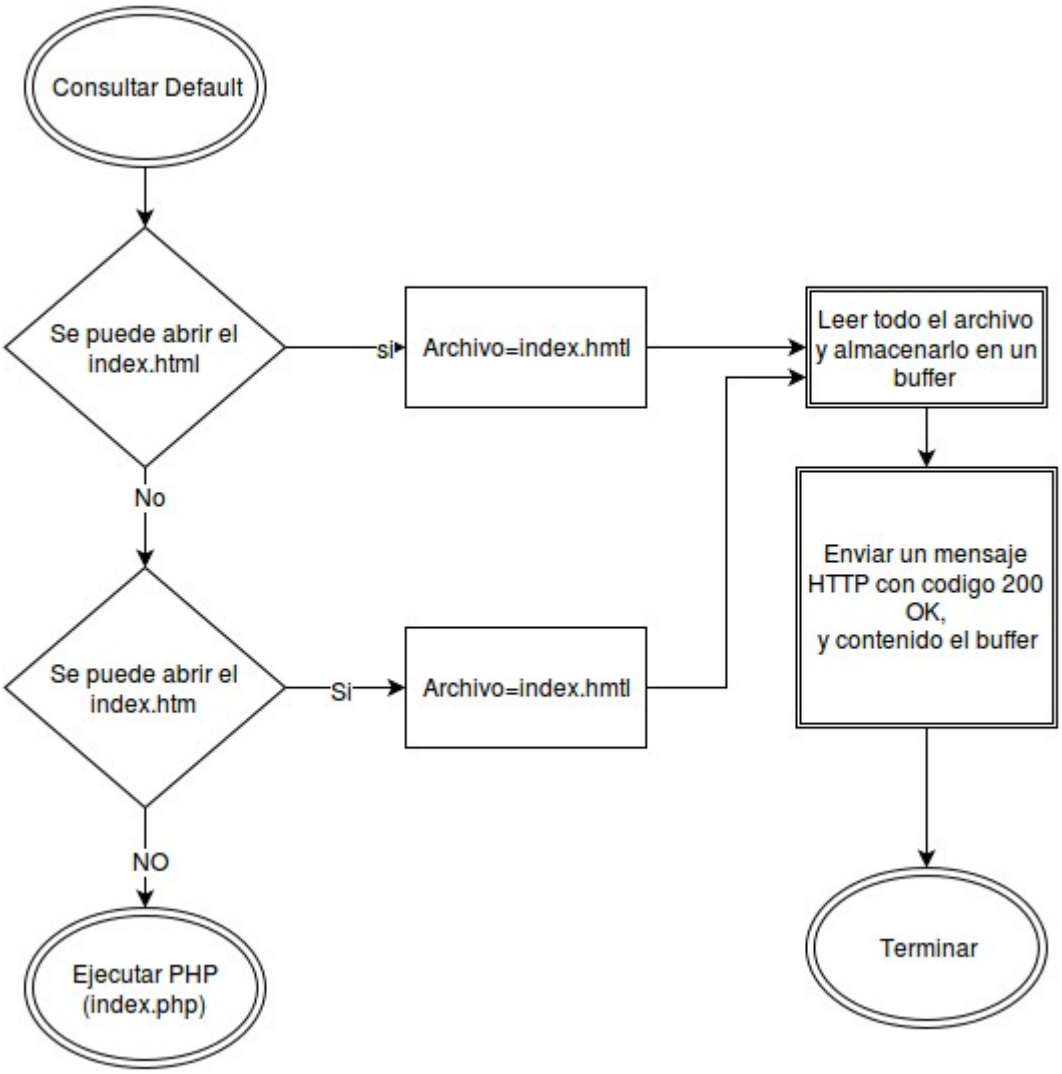
El servidor solo utiliza IPv4, esta compilado con un gcc version 4.6.3 20120306 (Red Hat 4.6.3-2) (GCC)

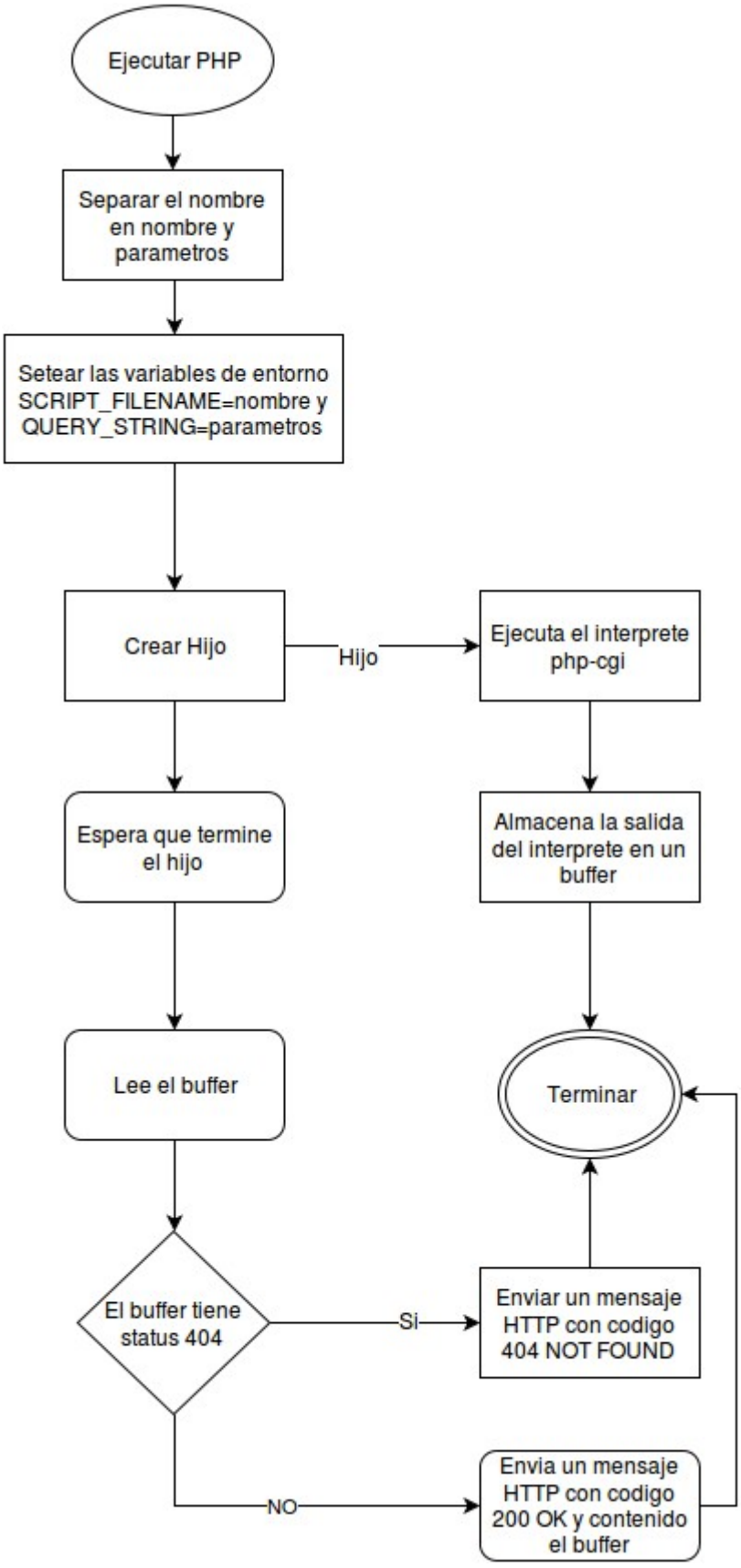
El algoritmo principal es el siguiente:
Si el parámetro -h esta presente se muestra la ayuda del programa.
Sino
Se procesa el ip y el puerto si los hubiera
Si se puede ligar un socket bienvenida a ese ip y puerto
El socket se queda escuchando en ese ip y puerto
Por cada nuevo requerimiento de conexion al ip:puerto
Si se puede Se crea un proceso hijo
El hijo hereda el socket tcp de conexión y atienda el pedido
El padre si tiene algun proceso hijo zombie lo ignora.
Sino cierra con error al no poder forkear
Sino cierra con error al no poder ligar el socket Bienvenida
atender(socket)
Lee completo el mensaje http enviado por el cliente
Separa el encabezado en 3 partes: pedido, archivo y protocolo
Si el pedido es un POST o un HEAD,no estan implentados
Se responde con un 501 NOT IMPLEMENTED
Sino
Si el encabezado no es valido
Se responde 400 BAD REQUEST
Sino
Determino la extension
Si la extension es .html o htm
Ejecuto consutarHMTL
Sino Si la extension es jpg, gif o png
Ejecuto consultar MIME
Sino Si la extension es php
Ejecuto ejecutarPHP
Sino si es archivo vacio
Ejecuto consultarDefault
Sino
pide otra extension como .mp3
Envio un 404 NOT FOUND

Diagrama de Flujo




Consultar HTML
Se puede abrir el archivo.html
si
Leer todo el archivo y almacenarlo en un buffer
No
Envio un mensaje HTTP con codigo 404 NOT FOUND
Enviar un mensaje HTTP con codigo 200 OK y contenido igual al buffer
Terminar
Consultar MIME
Se puede abrir el archivo.html
si
ext!=.jpg
Leer todo el archivo y almacenarlo en un buffer
No
Envio un mensaje HTTP con codigo 404 NOT FOUND
Cambiar ext de jpg a jpeg
Enviar un mensaje HTTP con codigo 200 OK, MIME VERSION 1.0 y CONTENT TYPE= image/extension y contenido el buffer
Terminar


Consultar Default
Se puede abrir el index.html
si
Archivo=index.hmtl
Leer todo el archivo y almacenarlo en un buffer
No
Se puede abrir el index.htm
Si
Archivo=index.hmtl
Enviar un mensaje HTTP con codigo 200 OK,
y contenido el buffer
NO
Ejecutar PHP (index.php)
Terminar


Ejecutar PHP
Separar el nombre en nombre y parametros
Setear las variables de entorno SCRIPT_FILENAME=nombre y QUERY_STRING=parametros
Crear Hijo
Hijo
Ejecuta el interprete php-cgi
Espera que termine el hijo
Almacena la salida del interprete en un buffer
Lee el buffer
Terminar
El buffer tiene status 404
Si
Enviar un mensaje HTTP con codigo 404 NOT FOUND
NO
Envia un mensaje HTTP con codigo 200 OK y contenido el buffer

Código del programa

```
#include <arpa/inet.h>
#include <signal.h>
#include <stdio.h>
#include <wait.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/stat.h>
#include <syslog.h>

#define MAX_TAM 1024*1024          //Maximo tamaño transferido
#define MAX_REQ 2048               //Maximo largo del request
#define NOT_FOUND "HTTP/1.0 404 Not Found\r\nContent-type: text/html\r\n\r\n<HTML><TITLE>Not Found</TITLE><BODY><P>El servidor NO encontro la pagina Solicitada.</BODY></HTML>\r\n"
#define NOT_IMP "HTTP/1.0 501 Method Not Implemented\r\n Content-type: text/html\r\n\r\n<HTML><TITLE>Method Not Implemented</TITLE><BODY><P>El metodo solicitado es NO ha sido implementado por el server.</BODY></HTML>\r\n"
#define BAD_REQ "HTTP/1.0 400 Bad Request\r\n Content-type: text/html\r\n\r\n<HTML><TITLE>Bad Request</TITLE><BODY><P>El metodo solicitado es Incorrecto.</BODY></HTML>\r\n"
#define Largo_NF 160
#define Largo_NI 200
#define Largo_BR 150

typedef struct sockaddr_in tSocket;

//Metodos Usados:
FILE* defaultsHTML(FILE * archivo,char* filename);        //Intenta abrir archivos por defectos
void consultarHTML(char* filename,int socket);       //envia el contenido de un arch .html
void consultarMIME(char* filename,char* ext,int socket);   //envia ..jpg, .png y .gif
void ejecutarPHP(char* filename,int socket);//Ejecuta el .php y envia su resultado
void consultarDefaults(int socket);      //Intenta consultar los archivos por defecto cuando no se especifican
void salirError(char* error);  //Realiza un EXIT_FAILURE pero primero muestra a error por Pantalla
char* getExt(char* filename);//Recibe un filename y retorna su extension, si este no tubiera, retorna NULL
void setearSignals();                    //
void imprimirAyuda();                            //
int getSize(char* filename);
void atender(int socket);                //Atiendo el request de un Cliente en el socket TCP socket
int ipValida(char *ipAddress);

int main(int argc, char *argv[]) {
        char *server="127.0.0.1";                //Ip del que escucha el server
        int port=80;                             //Puerto ligado al server

        //Asumo orden en el pasaje de parametro
        //De no  tener orden solo encapsularia el if en un for con i del 1 argc-1
        // e indexaria a argv con i en vez de con argc-1
        // servidorHTTP [IP] [:Puerto] [-h]
        if(argc>1 &&strcmp("-h", argv[argc-1]) == 0)//Si Necesita ayuda, imprimo la misma y salgo
         imprimirAyuda();
```

```
setearSignals();            //Seteo los handlers de las distintas Signals
openlog("servidorHTTP",LOG_PID,LOG_LOCAL0);         //abro el syslog

int i=fork();
if(i==0){                                           //Realizamos un fork para trabajar en Background
 //Si tiene Puerto
 if(argc>1)
 {
        if(strncmp(":",argv[1],1)==0)    // tiene puerto pero no IP
                port=atoi((argv[1]+1));             //guardo el puerto
        else    //tiene ip
        {
                server=argv[1];                                         //guardo el ip
                if(argc>2 && strncmp(":",argv[1],1)==0)         //reviso si tiene puerto
                        port=atoi((argv[1]+1));                         //guardo puerto
        }
 }

 //Ya tengo los valores del ip y puerto asignados
 syslog(LOG_INFO,"El ip es: %s y el puerto %d\n",server,port);

 if(!ipValida(server))
        salirError("ERROR: Direccion IP NO VALIDA\nh");
 int clilen, pid;                       //
 tSocket server_addr, client_addr;
 int socket_B,socketTcp;//Socket_B es el socket de Bienvenida, SocketTcp el de comunicacion

 socket_B=socket(AF_INET, SOCK_STREAM, 0);      //Socket TCP
 if(socket_B==-1)
        salirError("No se pudo crear el socket TCP de Bienvenida\n");

 bzero((char *) &server_addr, sizeof(server_addr));     //like memset a 0  en todo ese espacio
 server_addr.sin_family = AF_INET;                                      //Tipo net
 server_addr.sin_addr.s_addr = inet_addr(server);       //direccion del ip
 server_addr.sin_port = htons(port);                                    //Puerto para el socket

 if (bind(socket_B, (struct sockaddr *) &server_addr, sizeof(server_addr)) < 0)  //realizamos el bind
{
        salirError("Error al ejecutar el bind\n");

 listen(socket_B,5);                    //cola de 5 pedidos
 clilen = sizeof(client_addr);

 while(1){
        //Creamos un socket TCP y se lo asignamos a la conexion
        socketTcp= accept(socket_B, (struct sockaddr *) &client_addr, &clilen);
        if(socket_B==-1)
                salirError("No se pudo crear el socket TCP\n");
        pid=fork();
        if(pid<0)
                salirError("No se pudo crear fork\n");

        if (pid == 0)  {                //Hijo
                close(socket_B);                //cerramos la conexion con el socket padre
                atender(socketTcp);     //Atencion del pedido
                exit(0);
        }
```

```
                else close(socketTcp);              //cerramos conexion con socket hijo
                waitpid(-1,NULL, WNOHANG);          //un wait a cualquier hijo de manera no bloqueante
         }
        }
        return 0;
}


//Atiendo el request de un Cliente en el socket TCP socket
void atender(int socket)
{
        char linea[MAX_REQ],msj[MAX_REQ],filename[100]=".";
        char respNF[MAX_TAM]=NOT_FOUND,respNI[MAX_TAM]=NOT_IMP,
        respBR[MAX_TAM]=BAD_REQ;

        read(socket,msj,MAX_REQ);                           //Leo un mensaje de  a lo sumo 2048 bytes
        while(strstr(msj,"\r\n\r\n")==NULL){    //Mientras el mensaje no esta completo
                read(socket,linea,MAX_REQ);
                strcat(msj,linea);
        }

        char* ext;
        char* pedido= strtok(msj," ");              //Consumo metodo
        char* file=strtok(NULL," ");                //Guardo el nombre del archivo(sin espacios)
        char* protocolo=strtok(NULL,"\n");      //Almaceno el Protocolo utilizado


        if((strcmp(pedido,"POST")==0)||(strcmp(pedido,"HEAD")==0)) //Reviso consultas no impl
                send(socket,respNI,strlen(respNI),0);    //Envio NOT_IMPLEMENTED
        else if(pedido==NULL ||file==NULL ||protocolo==NULL || (strcmp(pedido,"GET")!=0)||
strncmp(protocolo,"HTTP/1.",5)!=0|| strncmp(file,"/",1)!=0)
         send(socket,respBR,strlen(respBR),0);  //Envio BAD_REQUEST
        else
        {
         //ya tengo guardado el filename

         strcat(filename,file);                         //Agrego el '.' adelante de /archivo
         ext=getExt(filename);

         if(strcmp(ext,".html")==0 ||strcmp(ext,".htm")==0)
                consultarHTML(filename,socket);
         else if (strcmp(ext,".png")==0 ||strcmp(ext,".gif")==0 ||strcmp(ext,".jpg")==0)
                        consultarMIME(filename,ext,socket);
                else if(strncmp(ext,".php",4)==0)
                                ejecutarPHP(filename,socket);
                        else if(strcmp(ext,"./")==0)
                                        consultarDefaults(socket);
                                else
                                        send(socket,respNF,strlen(respNF),0);   //Envio NOT_IMP
        }
}

//Recibe el nombre del archivo html y devuelve el mensaje a enviar
void consultarHTML(char* filename,int socket)
{
        int filesize=getSize(filename);
        char *txt=malloc(filesize+Largo_NF);
```

```
	char linea[filesize];
	FILE * archivo   = fopen(filename,"r");

	strcpy(txt,NOT_FOUND);
	if(archivo!=NULL)                       //Si no ecuentra archivo, 404 NOT_FOUND
	{
	 //Ya tenemos el archivo abierto
	 strcpy(txt, "HTTP/1.0 200 OK\r\nContent-Type: text/html\r\n\r\n");

	 while(!feof(archivo))
	 {
		fgets(linea,filesize,archivo);
		strcat(txt,linea);
	 }
	 fclose(archivo);
	}
	send(socket,txt,strlen(txt),0);
	free(txt);
}

//Recibe el nombre del archivo MIME
void consultarMIME(char* filename,char* ext, int socket)
{

	char txt[MAX_TAM]=NOT_FOUND,linea[4096];
	FILE * archivo   =fopen(filename,"rb");
	if(archivo==NULL)
	 send(socket,txt,strlen(txt),0);
	else
	{
	 if(strcmp(ext,".jpg")==0)strcpy(ext,".jpeg");
	 //Ya tenemos el archivo abierto
	 strcpy(txt,  "HTTP/1.0 200 OK\r\nMIME-Version: 1.0\r\nContent-Type:  image/");
	 strcat(txt,ext+1);           //copio extension
	 strcat(txt,"\r\n\r\n");
	 send(socket,txt,strlen(txt),0); //Envio Encabezado

	 int num;
	 char LINEA[100];

	 while(!feof(archivo))
	 {
		num=fread(linea,1,100,archivo);
		send(socket,linea,num,0);
	 }
	}
}

//Ejecuta el .php
void ejecutarPHP(char* filename,int socket)
{
	char txt[MAX_TAM]="HTTP/1.0 200 OK\r\n";
	char* name=strtok(filename,"?");
	char* param=strtok(NULL,"\n");
	char* parametro= malloc(strlen(param)+13);
	char* archivo= malloc(strlen(param)+13);
	char buffer[MAX_TAM];
```

```
	int *status,num,pipefd[2];

	strcpy(parametro,"QUERY_STRING=");
	if(param!=NULL)
	 strcat(parametro,param);
	strcpy(archivo,"SCRIPT_FILENAME=");
	strcat(archivo,name);


	pipe(pipefd);
	if (fork() == 0){
	   close(pipefd[0]);    // close reading end in the child
	   dup2(pipefd[1], 1);  // send stdout to the pipe
	   dup2(pipefd[1], 2);  // send stderr to the pipe
	   close(pipefd[1]);    // this descriptor is no longer needed

	 putenv(parametro);
	 putenv(archivo);
	 execlp("php-cgi","php-cgi",NULL);

	}else{    // parent
	   close(pipefd[1]);  // close the write end of the pipe in the parent
	 wait(NULL);
	 read(pipefd[0], buffer, sizeof(buffer));
	}
	if(strncmp(buffer,"Status: 404",11)==0)
		strcpy(txt,NOT_FOUND);
	else
	 strcat(txt,buffer);
	send(socket,txt,strlen(txt),0); //Envio Mensaje
}

//Intenta consultar los archivos por defecto cuando no se especifican
void consultarDefaults(int socket)
{
	char filename[100];
	FILE* archivo=defaultsHTML(archivo,filename);
	if(archivo!=NULL)
	{

	 int filesize=getSize(filename);
	 char *txt=malloc(filesize+Largo_NF);
	 char *linea=malloc(filesize);

	 strcpy(txt,  "HTTP/1.0 200 OK\r\nContent-Type: text/html\r\n\r\n");
	 while(!feof(archivo))
	 {
		fgets(linea,filesize,archivo);
		strcat(txt,linea);
	 }

	 send(socket,txt,strlen(txt),0); //Envio Mensaje
	 fclose(archivo);
	 free(txt);
	}
	else
	 ejecutarPHP("./index.php",socket);
```

```
}

//Intenta abrir archivos por defectos cuando estos no estan especificados en el mensaje
FILE* defaultsHTML(FILE * archivo,char* filename)
{
	strcpy(filename,"./index.html");
	archivo  = fopen(filename,"r");
	if(archivo==NULL){
	 strcpy(filename,"./index.htm");
	 archivo = fopen(filename,"r");
	}
	return archivo;
}
//Recibe un filename y retorna su extension, si este no tubiera, retorna NULL
char* getExt(char* filename)
{
	char *aux,*ext=strstr(filename,".");
	aux=ext;
	while(ext!=NULL )
	{
	 aux=ext;
		ext = strstr(ext+1, ".");
	}
	return aux;
}

void * terminar(int myint)
{
	syslog(LOG_INFO,"\nSe recepciono la señal, el programa finalizará.\n\n");
	closelog();
	exit(EXIT_SUCCESS);}
void setearSignals()
{
	signal(2,SIG_IGN);
	signal(3,SIG_IGN);
	signal(15,SIG_IGN);
	signal(SIGUSR2,SIG_IGN);
	signal(SIGUSR1, (void *)  terminar );

}
void imprimirAyuda()
{
	//codigo de ayuda
	printf("\n\n    Usage: servidorHTTP [IP] [:Puerto] [-h]\n");
	printf("* IP        Nro de IP donde el programa escucha los requerimientos, por defecto  es
localhost\n");
	printf("* Puerto              Nro de puerto en el cual se deben decepcionar las peticiones. Por defecto
debera ser el puerto 80\n");
	printf("* -h        Mensaje de Ayuda\n\n");
	exit(EXIT_SUCCESS);
}
void salirError(char* error)
{
	syslog(LOG_ERR,"Error: %s",error);
	exit(EXIT_FAILURE);
}
```

```
int getSize(char* filename)
{
	struct stat st;
	stat(filename, &st);
	return st.st_size;
}
int ipValida(char *ipAddress)
{
   struct sockaddr_in sa;
   int result = inet_pton(AF_INET, ipAddress, &(sa.sin_addr));
   return result != 0;
}
```