```c
#include "servidorHTTP.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#include <ctype.h>
#include <signal.h>
#include <syslog.h>


/* Muestra el mensaje de error y termina el programa con EXIT_FAILURE */
void error(char *msg) {
    log_error(msg);
    perror(msg);
    exit(EXIT_FAILURE);
}

/* Para loggear errores en el system log */
void log_error(char *msg) {
    openlog ("servidorHTTP", LOG_CONS | LOG_PID | LOG_NDELAY, LOG_LOCAL0);
    syslog (LOG_ERR, msg);
    closelog ();
}

/* Para loggear informacion en el system log */
void log_info(char *msg) {
    openlog ("servidorHTTP", LOG_CONS | LOG_PID | LOG_NDELAY, LOG_LOCAL0);
    syslog (LOG_INFO, msg);
    closelog ();
}

/* Muestra mensaje de ayuda */
void ayuda() {
   printf("Modo de uso: ./servidorHTTP [servidor][:puerto] [-h]\n \n");
   printf("\t[servidor]: \tDireccion IP del servidor. (Default: localhost) \n"
);
   printf("\t[:puerto]: \tPuerto del servidor. (Default: 80)\n");
   printf("\t[-h]: \t\tAyuda por pantalla (este mensaje). \n");
   printf("\n");
   printf("Si no se especifica servidor o puerto, la direccion \n");
   printf("por defecto sera 127.0.0.1 y el puerto por defecto sera 80.\n\n");
   exit(EXIT_SUCCESS);
}

/* Manejador de señales */
void signalHandler(int sig) {
    if (sig == SIGUSR1) {
        exit(EXIT_SUCCESS);
    }
}

/* Verifica si una IP es valida */
```

```
58 int verificarIP(char * servidor){
59     int i=inet_addr(servidor);
60     if(i==INADDR_NONE) {
61         return 0;
62     } else {
63         return 1;
64     }
65 }
66 
67 /* Verifica si un puerto es valido */
68 int verificarPuerto(char * puerto){
69     int p = atoi(puerto);
70     if (p>=1 && p<=65535) {
71         return 1;
72     } else {
73         return 0;
74     }
75 }
76 
77 int main(int argc, char *argv[]) {
78     // Reviso si me ingresaron un -h para mostrar ayuda
79     // Si asi fuera, se muestra la ayuda y se termina el programa
80     int i;
81     for (i = 1; i <= (argc - 1); i++) {
82         if (strcmp("-h", argv[i]) == 0) {
83             ayuda();
84         }
85     }
86 
87     // Configuración para el manejo de señales
88     signal(SIGUSR1, (void *)signalHandler);
89     signal(SIGUSR2, (void *)signalHandler);
90     signal(SIGCHLD, SIG_IGN);
91     signal(SIGABRT, (void *)signalHandler);
92     signal(SIGHUP, (void *)signalHandler);
93     signal(SIGINT, (void *)signalHandler);
94     signal(SIGQUIT, (void *)signalHandler);
95     signal(SIGTERM, (void *)signalHandler);
96 
97     // Correr el programa en background. Parametros (1,1) para no cambiar path
    ni stdin/out/err!
98     if (daemon(1,1) < 0) { error(ERROR_DAEMON);}
99 
100    // En este punto del programa si esta corriendo es porque no preguntaron p
    or ayuda
101    char * servidor = NULL;
102    char * puerto = NULL;
103    // Asigno Servidor:Puerto, si existiera
104    // En caso de que no ingreso argumentos, uso el default
105    if (argc == 1) {
106        // No pasaron argumentos (uso default!)
107        servidor = strdup(LOCALHOST);     // 127.0.0.1
108        puerto = strdup(WEBPORT);        // 80
109    } else {
110        char * str = argv[1];
111 
112        // Split Servidor:Puerto
113        servidor = strtok(str, ":");
```

```
114         puerto = strtok(NULL, ":");
115 
116         if (strstr(str,":"))  {
117             // Ingreso :puerto pero no ingreso servidor
118             puerto = servidor;
119             servidor = strdup(LOCALHOST);
120         } else {
121             // Ingreso servidor:puerto o solo servidor
122 
123             // Si solo me pasaron servidor (y no puerto)
124             // entonces asigno el puerto por defecto (80)
125             if (puerto == NULL) { puerto= strdup(WEBPORT); }
126         }
127     }
128 
129     log_info(servidor);
130     log_info(puerto);
131 
132     if (servidor == NULL || puerto == NULL) {
133         error(ERROR_INPUT_DATOS);
134     } else     if (!verificarIP(servidor) || !verificarPuerto(puerto)) {
135         error(ERROR_IP_PORT);
136     }
137 
138     int sockfd, newsockfd, portno, clilen, pid;
139     struct sockaddr_in serv_addr, cli_addr;
140 
141     // creo un socket TCP y obtengo el File Descriptor
142     sockfd = socket(AF_INET, SOCK_STREAM, 0);
143     if (sockfd < 0)
144         error(ERROR_ABRIR_SOCKET);
145 
146     bzero((char *) &serv_addr, sizeof(serv_addr));
147     portno = atoi(puerto);
148     serv_addr.sin_family = AF_INET;
149     serv_addr.sin_addr.s_addr = inet_addr(servidor);
150     serv_addr.sin_port = htons(portno);
151 
152     if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
153         error(ERROR_BIND_SOCKET);
154 
155     listen(sockfd,5);
156     clilen = sizeof(cli_addr);
157     while (1) {
158         newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
159         if (newsockfd < 0)
160             error(ERROR_ACCEPT_SOCKET);
161 
162         pid = fork();
163 
164         if (pid < 0) {
165             error(ERROR_FORK);
166         } else if (pid == 0)  {
167             // Soy hijo, me desligo del socket paterno y atiendo el pedido
168             close(sockfd);
169             atenderPedido(newsockfd);
170             exit(EXIT_SUCCESS);
171         } else  {
```

```
172             // Soy padre, me desligo del socket de mi hijo
173             // y sigo atendiendo pedidos.
174             close(newsockfd);
175         }
176     }
177     // No deberia llegar aca
178     return 0;
179 }
180 
181 /* Parsea el mensaje dado en buffer y retorna el tipo de mensaje,
182  * la ruta del archivo y el protocolo usado.
183  * No se encarga de hacer chequeos sobre la correctitud del mensaje.
184  *  */
185 void parseMsg (char * buf, char ** tipoMsg, char ** ruta, char ** protocolo) {

186        char * token = strtok(buf, "\n\r");
187     *tipoMsg = strtok(token," ");
188     *ruta = strtok(NULL," ");
189     *protocolo = strtok(NULL," ");
190 }
191 
192 int archivoExiste(char * archivo) {
193     int rval = access (archivo, F_OK);
194     if (rval == 0)
195             return 1;
196     else return 0;
197 }
198 
199 int archivoAbrible(char * archivo) {
200     int rval = access (archivo, R_OK);
201     if (rval == 0)
202             return 1;
203     else return 0;
204 }
205 
206 long archivoSize(char * archivo) {
207     FILE *file = fopen(archivo, "rb");
208     if (!file) error(ERROR_ABRIR_ARCHIVO);
209     fseek(file, 0, SEEK_END);
210     long f_size = ftell(file);
211     fseek(file, 0, SEEK_SET);
212     fclose(file);
213     return f_size;
214 }
215 
216 /* Revisa si una cadena dada es GET */
217 int esGet(char * msg){
218     if (strcmp("GET", msg) == 0)
219         return 1;
220     else return 0;
221 }
222 
223 /* Revisa si una cadena dada es barra (/) */
224 int esBarra(char * msg){
225     if (strcmp(msg,"/")==0)
226         return 1;
227     else return 0;
228 }
```

```
229
230 /* Manda headers a traves del socket,
231  * segun el tipo de respuesta y de contenido */
232 void mandarHeader(int sock, char * resp){
233     if (resp!=NULL) {
234         if (send(sock,resp,strlen(resp),0) < 0) {
235             error(ERROR_SEND_SOCKET);
236         }
237     }
238 }
239
240 /* Manda headers a traves del socket,
241  * segun el tipo de respuesta y de contenido */
242 void mandarHeaders(int sock, char * tipoResp, char * tipoCont){
243     mandarHeader(sock,tipoResp);
244     mandarHeader(sock,tipoCont);
245 }
246
247 /* Manda un header 4xx con un rechazo,
248  * ademas le manda contenido en HTML
249  * para mostrar una pagina de Error. */
250 void mandarRechazo(int sock, char * tipoResp, char * titulo, char * mensaje){
251     mandarHeaders(sock,tipoResp,CT_HTML);
252     // Preparo un mensaje HTML con el error 4xx o 5xx
253     char * rta = strdup("<html><body><title>");
254     strcat(rta,titulo);
255     strcat(rta,"</title><h1>");
256     strcat(rta,titulo);
257     strcat(rta,"</h1><p>");
258     strcat(rta,mensaje);
259     strcat(rta,"</p></body></html>");
260     mandarHeader(sock,rta);
261 }
262
263 /* Dado un socket y un archivo, se manda dicho archivo
264  * en modo binario a traves del socket */
265 void mandarArchivo(int sock, char * archivo){
266     size_t lfile = archivoSize(archivo);
267     char * bufferSalida = malloc(lfile);
268
269     FILE *file = fopen(archivo, "rb");
270     if (!file) error(ERROR_ABRIR_ARCHIVO);
271     int c; size_t t = 0;
272
273     int nro,n;
274     while(!feof(file))
275     {
276         nro = fread(bufferSalida,1,128,file);
277         n = send(sock,bufferSalida,nro,0);
278         if (n < 0) error(ERROR_SEND_SOCKET);
279     }
280     free(bufferSalida);
281 }
282
283 /* Revisa si una cadena es .html o .htm */
284 int esHTML(char * archivo){
285     char * arch = getExtension(minusculas(archivo));
286     if ((strcmp(arch,".html")==0) || (strcmp(arch,".htm")==0))
```

```
287         return 1;
288     else return 0;
289 }
290 
291 /* Revisa si una cadena es .jpg */
292 int esJPG(char * archivo){
293     char * arch = getExtension(minusculas(archivo));
294     if (strcmp(arch,".jpg")==0)
295         return 1;
296     else return 0;
297 }
298 
299 /* Revisa si una cadena es .gif */
300 int esGIF(char * archivo){
301     char * arch = getExtension(minusculas(archivo));
302     if (strcmp(arch,".gif")==0)
303         return 1;
304     else return 0;
305 }
306 
307 /* Revisa si una cadena es .png */
308 int esPNG(char * archivo){
309     char * arch = getExtension(minusculas(archivo));
310     if (strcmp(arch,".png")==0)
311         return 1;
312     else return 0;
313 }
314 
315 /* Revisa si una cadena es .php */
316 int esPHP(char * archivo){
317     char * arch = getExtension(minusculas(archivo));
318     if (strcmp(arch,".php")==0)
319         return 1;
320     else return 0;
321 }
322 
323 /* Dado una cadena, obtiene su extension (si la tuviese) */
324 char * getExtension(char* archivo) {
325     char *aux, *ext=strstr(archivo,".");
326     aux=ext;
327     while (ext !=NULL) {
328         aux= ext;
329         ext = strstr(ext+1, ".");
330     }
331     return aux;
332 }
333 
334 /* Funcion para agregar un caracter al final de una cadena */
335 char * appchr(char * str, const char chr) {
336     size_t len = strlen(str);;
337     char * ptr = malloc(len+2);
338     strcpy(ptr,str);
339     ptr[len] = chr;
340     ptr[len+1] = '\0';
341     return ptr;
342 }
343 
344 /* Recibe el mensaje de un socket hasta que se lean dos "enters" seguidos.
```

```
345  * Devuelve un puntero al principio de lo leido (buffer) */
346 char * recibirMensaje(int sock) {
347     char * buffer = malloc(sizeof(char));;
348     char * letra  = malloc(sizeof(char));
349     char a,b,c,d;
350     int n;
351     a = 1 ; b = 1; c = 1; d = 1; n = 1;
352     int termine = 0;
353 
354     while (!termine && n>0){
355        n = recv(sock,letra,1,0);
356        if (n < 0) error(ERROR_RECV_SOCKET);
357        d = letra[0];
358        buffer = appchr(buffer,d);
359        // Si veo dos enters seguidos activo el flag de terminacion
360        if ((a == 10 && b == 13 && c == 10 && d == 13) || (a == 13 && b == 10 &
    & c == 13 && d == 10)){
361            termine = 1;
362        } else {
363            a = b;
364            b = c;
365            c = d;
366        }
367     }
368     free(letra);
369     return buffer;
370 }
371 
372 /* Dada una cadena de caracteres, devuelve la misma cadena pero en minusculas
    */
373 char * minusculas(char * str){
374     int i;
375     char * rta = malloc(strlen(str));
376     for(i = 0; str[i]; i++){
377         rta[i] = tolower(str[i]);
378     }
379     return rta;
380 }
381 
382 /* Dada una ruta mediante archivo, obtiene el nombre del archivo
383  * y los argumentos (si los hubiera), los separa y retorna
384  * la ruta al indice (mediante el primer parametro)
385  * y sus argumentos mediante el segundo parametro */
386 void verificarPHP(char ** archivo, char ** argumentos){
387     char * arch = strdup(*archivo);
388     char * ruta = strtok(arch, " ?\n\r");
389     * argumentos = strstr(*archivo, "?");
390     * archivo = ruta;
391 }
392 
393 /* Se encarga de la parte PHP. Hace el fork, el hijo el exec(php-chi)
394  * y el padre envia el mensaje generado por el hijo */
395 void procesarPHP(int sock, char * archivo, char * parametros){
396     int pipefd[2];
397     if (pipe(pipefd) < 0) { error(ERROR_PIPE); }
398 
399     pid_t pid = fork();
400 
```

```
401     if (pid == 0) {
402         close(pipefd[0]);    // close reading end in the child
403         dup2(pipefd[1], 1);  // mando standart output al pipe
404         dup2(pipefd[1], 2);  // mando standart error al pipe
405         close(pipefd[1]);    // this descriptor is no longer needed
406 
407         char * query;
408         if (parametros!=0) {
409             // Cargo en la variable de entorno QUERY_STRING
410             // los parametros que me hayan pasado
411             query = malloc(13+strlen(parametros));
412             strcpy(query,"QUERY_STRING=");
413             strcat(query,parametros+1);
414             putenv(query);
415         } else {
416             // No me pasaron parametros, "reseteo el query string"
417             putenv("QUERY_STRING=");
418         }
419 
420         // Cargo en la variable de entorno SCRIPT_FILENAME
421         // el archivo que este solicitando el usuario
422         char script [16+strlen(archivo)];
423         strcpy(script,"SCRIPT_FILENAME=");
424         strcat(script,archivo);
425         putenv(script);
426 
427         execlp("php-cgi","php-cgi",NULL);
428     } else if (pid > 0) {
429         // Soy el padre
430         char buf[1];
431         close(pipefd[1]);  // close the write end of the pipe in the parent
432         int n;
433 
434         // Espero a que termine mi hijo
435         int returnStatus;
436         waitpid(pid,&returnStatus,0);
437 
438         char * buffer = malloc(sizeof(char));;
439 
440         while (read(pipefd[0], buf, sizeof(buf)) != 0)
441         {
442             // Cargo lo que me respondio php-cgi en un buffer
443             buffer = appchr(buffer,buf[0]);
444         }
445         // Lo mando
446         mandarHeader(sock,RTA_200);
447         mandarHeader(sock,buffer);
448         free(buffer);
449     }
450     else if (pid < 0) { error(ERROR_FORK); }
451 }
452 
453 /* Metodo principal que se encarga de atender un pedido mediante un socket dad
    o */
454 void atenderPedido(int sock) {
455     int n;
456 
457     char * tipoMsg = NULL;
```

```
458     char * ruta = NULL;
459     char * protocolo = NULL;
460 
461     char * buffer = recibirMensaje(sock);
462 
463     // Analizo el mensaje (la primera linea es la que importa en realidad)
464     // Obtengo el tipo de metodo, la ruta y el protocolo utilizado.
465     parseMsg(buffer, &tipoMsg, &ruta, &protocolo);
466 
467     char * archivo = NULL;
468     char * parametros = NULL;
469     if (esBarra(ruta)) {
470         // Si me pasaron / tengo que buscar si tengo index.html, index.htm, o
    index.php
471         // Sobreescribir sobre el valor de la variable archivo
472         if (archivoExiste("index.html")) {
473             archivo = strdup("index.html");
474         } else if (archivoExiste("index.htm")) {
475                 archivo = strdup("index.htm");
476             } else if (archivoExiste("index.php")) {
477                     archivo = strdup("index.php");
478                 }
479     } else {
480         archivo = ruta+1;
481     }
482 
483     // Verifico si el archivo es PHP y separo sus parametros ("desgloso")
484     // Esto lo hago en este punto porque luego se hara el chequeo
485     // de la existencia del archivo (y necesitamos unicamente el nombre del ar
    chivo)
486     verificarPHP(&archivo,&parametros);
487 
488     if (!ruta || !tipoMsg || !protocolo) {
489         // Me mandaron mal la request (alguno de los elementos del primer reng
    lon es vacio (NULL);
490         mandarRechazo(sock,RTA_400,"400 Bad Request", "The request sent didn't
     have the correct syntax.");
491     } else {
492         // Me mandaron un request que "puedo entender"
493         // Trato de interpretarlo y trabajarlo
494         if (esGet(tipoMsg)) {
495             if(archivoExiste(archivo)) {
496                 // El archivo existe
497                 if (archivoAbrible(archivo)) {
498                     // El archivo se puede abrir
499                     if (esHTML(archivo)) {
500                         // Es HTML o HTM
501                         mandarHeaders(sock,RTA_200,CT_HTML);
502                         mandarArchivo(sock,archivo);
503                     } else if (esJPG(archivo) || esPNG(archivo) || esGIF(archi
    vo)) {
504                             // Es JPG, GIF o PNG
505                             if (esJPG(archivo)) {
506                                 mandarHeaders(sock,RTA_200,CT_JPG);
507                             }
508                             else if (esGIF(archivo)) {
509                                     mandarHeaders(sock,RTA_200,CT_GIF);
510                                 }
```

```
511                                     else {
512                                         mandarHeaders(sock,RTA_200,CT_PNG);
513                                     }
514                                 // Ya mande los headers, ahora mando el archivo
515                                 mandarArchivo(sock,archivo);
516                             } else if (esPHP(archivo)) {
517                                 // es PHP
518                                 procesarPHP(sock,archivo,parametros);
519                             } else {
520                                 // No es un tipo valido (extension desconocida) pe
    ro existe el archivo
521                                 // Tomar una decision de diseño. Por ejemplo, mand
    ar un 200 OK y el contenido del archivo
522                                 // Ojo con esto, podria influir en la "seguridad"
    del servidor
523                                 mandarRechazo(sock,RTA_403,"403 Forbidden", "Not a
    llowed to access the resource and authorization will not help.");
524                             }
525                         } else {
526                             // Archivo no se puede abrir. Mando Error 403
527                             mandarRechazo(sock,RTA_403,"403 Forbidden", "Not allowed t
    o access the resource and authorization will not help.");
528                         }
529                     } else {
530                         // Archivo no existe, Mando Error 404
531                         mandarRechazo(sock,RTA_404,"404 Not Found", "The requested fil
    e was not found.");
532                     }
533                 } else {
534                     // Metodo no permitido, mando Error 501
535                     mandarRechazo(sock,RTA_501,"501 Not Implemented", "The requested m
    ethod is not implemented.");
536                 }
537             }
538             free(buffer);
539 
540 }
541 
```