MONGOOSE

Elaborado por: Ukranio Coronilla

Muchas veces es necesario tener un servidor web incrustado en nuestra aplicación distribuida, para poder brindar una interfaz amigable a los usuarios. Para ello utilizaremos el código Mongoose disponible en:

https://cesanta.com/docs/overview/intro.html

Además de un servidor web, MONGOOSE incorpora una API no bloqueante de un solo hilo de ejecución, y orientada a eventos de los protocolos TCP, UDP, HTTP, WebSocket, CoAP y MQTT para clientes o servidores, el cual se puede compilar en C o en C++.

Todo el código fuente se encuentra en los archivos mongoose.c y mongoose.h cuyas ligas se encuentran en la liga anterior y se muestan a continuación:

https://raw.githubusercontent.com/cesanta/mongoose/master/mongoose.c

https://raw.githubusercontent.com/cesanta/mongoose/master/mongoose.h

En esta práctica vamos a ocupar para la comunicación entre cliente y servidor el protocolo HTTP(véase https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto), y vamos a transferir entre ellos texto en el formato HTML (véase https://es.wikipedia.org/wiki/HTML).

Iniciamos compilando un servidor HTML simple, para lo cual debe descargar los archivos moongose.c y mongoose.h del sitio web de cesanta. Posteriormente compile con g++ la implementación moongose.c con el siguiente código ejemplo con nombre server_HTML.cpp :

```
g++ server_HTML.cpp mongoose.c -o server_HTM
```

Ejemplo tomado de https://cesanta.com/docs/http/server-example.html , donde se encuentra mucha información adicional.

```
// Copyright (c) 2015 Cesanta Software Limited
// All rights reserved

#include "mongoose.h"

static const char *s_http_port = "8000";
static struct mg_serve_http_opts s_http_server_opts;

static void ev_handler(struct mg_connection *nc, int ev, void *p) {
  if (ev == MG_EV_HTTP_REQUEST) {
    mg_serve_http(nc, (struct http_message *) p, s_http_server_opts);
  }
}
```

```
int main(void) {
  struct mg_mgr mgr;
  struct mg_connection *nc;

  mg_mgr_init(&mgr, NULL);
  printf("Starting web server on port %s\n", s_http_port);
  nc = mg_bind(&mgr, s_http_port, ev_handler);
  if (nc == NULL) {
    printf("Failed to create listener\n");
    return 1;
  }

  // Set up HTTP server parameters
  mg_set_protocol_http_websocket(nc);
  s_http_server_opts.document_root = ".";  // Serve current directory
  s_http_server_opts.enable_directory_listing = "yes";

  for (;;) {
    mg_mgr_poll(&mgr, 1000);
  }
  mg_mgr_free(&mgr);

  return 0;
}
```

Antes de ejecutar el código, observe que en la línea

```
static const char *s_http_port = "8000";
```

se está especificando el puerto del servidor HTTP como el 8000. Al ejecutarlo tendrá un servidor HTML en el puerto 8000; lo cual puede comprobar ejecutando un cliente (navegador web) y poniendo la dirección local de la computadora y el puerto 8000 como sigue:

http://127.0.0.1:8000/

La línea `mg_mgr_init(&mgr, NULL);` Inicializa el manejador de los eventos posibles y almacena en la estructura `mgr` las conexiones que se encuentren activas.
La línea `nc = mg_bind(&mgr, s_http_port, ev_handler);` Crea una conexión de escucha especificando en el tercer parámetro la función manejadora con los posibles eventos que va a manejar. En este caso dentro de la función manejadora se valida si existe el evento MG_EV_HTTP_REQUEST, el cual indica que ha llegado una solicitud HTTP. La solicitud HTTP recibida, se pasa como `struct http_message` a través del apuntador p del manejador. Una lista de todos los eventos HTTP que se pueden manejar está en:
https://cesanta.com/docs/http/events.html

En el manejador la función mg_serve_http básicamente implementa un servidor de archivos estáticos HTTP. Su comportamiento se basa en la lista de opciones que se encuentran almacenadas en su último parámetro, la variable `s_http_server_opts`, la cual es del tipo `struct mg_serve_http_opts`. Esta variable se inicializa con las líneas:

```
s_http_server_opts.document_root = ".";  // Serve current directory
s_http_server_opts.enable_directory_listing = "yes";
```

La función precedente a estas líneas mg_set_protocol_http_websocket, adjunta un manejador de eventos HTTP a la conexión, el cual en este caso será `nc`.

Finalmente, en un ciclo infinito, la función que hace ocurrir la magia, mg_mgr_poll la cual se encarga de iterar sobre todos los sockets, aceptar conexiones nuevas, enviar y recibir datos, cerrar conexiones y mandar llamar a las funciones manejadoras de eventos para cada evento dado.

Ejercicio 1

Reutilizando el código de un servicio REST sencillo tomado del código elaborado por el alumno Rolando Romero Téllez en un proyecto final, y que se encuentra disponible en este servidor con nombre mongoose_REST.tar.gz ; cree una página HTML que reciba del usuario una dirección IP de broadcast y que al darle click en el botón de búsqueda, imprima las direcciones de las IPs que tienen los servidores activos (servidores del capítulo 9 del manual) así como los tiempos de respuesta de cada servidor.

En la página web vienen más ejemplos interesantes como el que está disponible en mi servidor y cuyo nombre es:

mongoose_ejemplo.tar

Pruébelo ¡¡¡¡¡