**linux下libwebsockets编译及实例**

    最近想自己搭建一个webscoket协议的服务器，打算用libwebsockts这个库。

**下载代码编译。**

编写一个shell脚本

#!/bin/sh
*# wget http://git.warmcat.com/cgi-bin/cgit/libwebsockets/snapshot/libwebsockets-1.4-chrome43-firefox-36.tar.gz*
*# tar xvzf libwebsockets-1.4-chrome43-firefox-36.tar.gz*
*#cd libwebsockets-1.4-chrome43-firefox-36*
*# build*
git clone https://github.com/warmcat/libwebsockets.git
cd libwebsockets
documentRoot=`pwd`
mkdir build
cmake ..
make

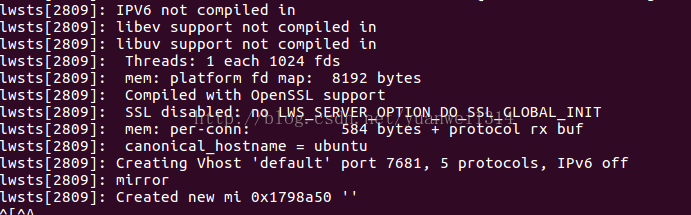
执行shell脚本就下载编译成功了。

**运行一个例子**

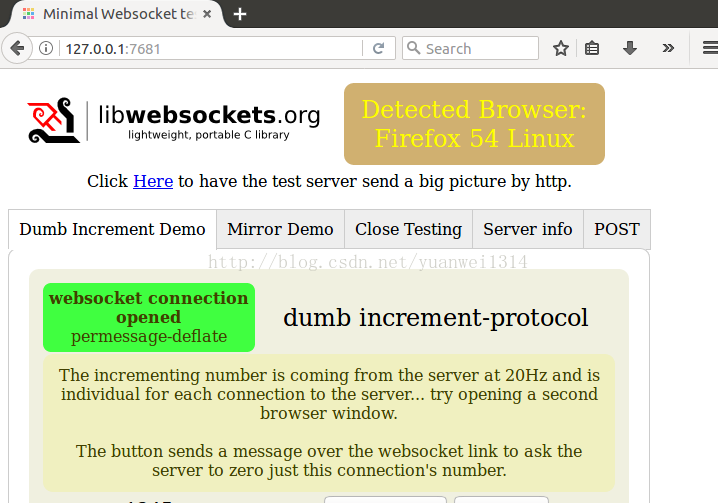
其实在库里面有一些实例的.

cd build
#!/bin/sh
documentRoot=`pwd`
libdir=libwebsockets
cd $libdir/build/bin
./libwebsockets-test-server --resource\_path=$documentRoot/test-server

 这样就运行自带的服务器。运行结果.



浏览器输入127.0.0.1:7681结果:



这样就是完整的结果了。

**搭建自己的websocket server**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <signal.h>

#include <libwebsockets.h>

#define KGRN "\033[0;32;32m"

#define KCYN "\033[0;36m"

#define KRED "\033[0;32;31m"

#define KYEL "\033[1;33m"

#define KMAG "\033[0;35m"

#define KBLU "\033[0;32;34m"

#define KCYN\_L "\033[1;36m"

#define RESET "\033[0m"

static int destroy\_flag = 0;

static void INT\_HANDLER(int signo) {

destroy\_flag = 0;

}

*/\* \**

*\* websocket\_write\_back: write the string data to the destination wsi.*

*\*/*

int websocket\_write\_back(struct lws \*wsi\_in, char \*str, int str\_size\_in)

{

if (str == NULL || wsi\_in == NULL)

return -1;

int n;

int len;

unsigned char \*out = NULL;

if (str\_size\_in < 1)

len = strlen(str);

else

len = str\_size\_in;

out = (unsigned char \*)malloc(sizeof(unsigned char)\*(LWS\_SEND\_BUFFER\_PRE\_PADDING + len + LWS\_SEND\_BUFFER\_POST\_PADDING));

*//\* setup the buffer\*/*

memcpy (out + LWS\_SEND\_BUFFER\_PRE\_PADDING, str, len );

*//\* write out\*/*

n = lws\_write(wsi\_in, out + LWS\_SEND\_BUFFER\_PRE\_PADDING, len, LWS\_WRITE\_TEXT);

printf(KBLU"[websocket\_write\_back] %s\n"RESET, str);

*//\* free the buffer\*/*

free(out);

return n;

}

static int ws\_service\_callback(

struct lws \*wsi,

enum lws\_callback\_reasons reason, void \*user,

void \*in, size\_t len)

{

switch (reason) {

case LWS\_CALLBACK\_ESTABLISHED:

printf(KYEL"[Main Service] Connection established\n"RESET);

break;

*//\* If receive a data from client\*/*

case LWS\_CALLBACK\_RECEIVE:

printf(KCYN\_L"[Main Service] Server recvived:%s\n"RESET,(char \*)in);

*//\* echo back to client\*/*

websocket\_write\_back(wsi ,(char \*)in, -1);

break;

case LWS\_CALLBACK\_CLOSED:

printf(KYEL"[Main Service] Client close.\n"RESET);

break;

default:

break;

}

return 0;

}

struct per\_session\_data {

int fd;

};

int main(void) {

*// server url will usd port 5000*

int port = 5000;

const char \*interface = NULL;

struct lws\_context\_creation\_info info;

struct lws\_protocols protocol;

struct lws\_context \*context;

*// Not using ssl*

const char \*cert\_path = NULL;

const char \*key\_path = NULL;

*// no special options*

int opts = 0;

*//\* register the signal SIGINT handler \*/*

struct sigaction act;

act.sa\_handler = INT\_HANDLER;

act.sa\_flags = 0;

sigemptyset(&act.sa\_mask);

sigaction( SIGINT, &act, 0);

*//\* setup websocket protocol \*/*

protocol.name = "my-echo-protocol";

protocol.callback = ws\_service\_callback;

protocol.per\_session\_data\_size=sizeof(struct per\_session\_data);

protocol.rx\_buffer\_size = 0;

*//\* setup websocket context info\*/*

memset(&info, 0, sizeof info);

info.port = port;

info.iface = interface;

info.protocols = &protocol;

info.extensions = lws\_get\_internal\_extensions();

info.ssl\_cert\_filepath = cert\_path;

info.ssl\_private\_key\_filepath = key\_path;

info.gid = -1;

info.uid = -1;

info.options = opts;

*//\* create libwebsocket context. \*/*

context = lws\_create\_context(&info);

if (context == NULL) {

printf(KRED"[Main] Websocket context create error.\n"RESET);

return -1;

}

printf(KGRN"[Main] Websocket context create success.\n"RESET);

*//\* websocket service \*/*

while ( !destroy\_flag ) {

lws\_service(context, 50);

}

usleep(10);

lws\_context\_destroy(context);

return 0;

}

执行脚本:

#!/bin/sh
libdir=libwebsockets
g++ -g -o <span style="font-family:Arial, Helvetica, sans-serif;">ws\_server </span><span style="font-family:Arial, Helvetica, sans-serif;">test\_ws\_server.cpp -I$libdir/lib -I$libdir/build -L$libdir/build/lib -lwebsockets</span>

**编写自己的websocket client**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <signal.h>

#include <pthread.h>

#include <libwebsockets.h>

#define KGRN "\033[0;32;32m"

#define KCYN "\033[0;36m"

#define KRED "\033[0;32;31m"

#define KYEL "\033[1;33m"

#define KBLU "\033[0;32;34m"

#define KCYN\_L "\033[1;36m"

#define KBRN "\033[0;33m"

#define RESET "\033[0m"

static int destroy\_flag = 0;

static int connection\_flag = 0;

static int writeable\_flag = 0;

static void INT\_HANDLER(int signo) {

destroy\_flag = 0;

}

struct session\_data {

int fd;

};

struct pthread\_routine\_tool {

struct lws\_context \*context;

struct lws \*wsi;

};

static int websocket\_write\_back(struct lws \*wsi\_in, char \*str, int str\_size\_in)

{

if (str == NULL || wsi\_in == NULL)

return -1;

int n;

int len;

unsigned char \*out = NULL;

if (str\_size\_in < 1)

len = strlen(str);

else

len = str\_size\_in;

out = (unsigned char \*)malloc(sizeof(unsigned char)\*(LWS\_SEND\_BUFFER\_PRE\_PADDING + len + LWS\_SEND\_BUFFER\_POST\_PADDING));

*//\* setup the buffer\*/*

memcpy (out + LWS\_SEND\_BUFFER\_PRE\_PADDING, str, len );

*//\* write out\*/*

n = lws\_write(wsi\_in, out + LWS\_SEND\_BUFFER\_PRE\_PADDING, len, LWS\_WRITE\_TEXT);

printf(KBLU"[websocket\_write\_back] %s\n"RESET, str);

*//\* free the buffer\*/*

free(out);

return n;

}

static int ws\_service\_callback(

struct lws \*wsi,

enum lws\_callback\_reasons reason, void \*user,

void \*in, size\_t len)

{

switch (reason) {

case LWS\_CALLBACK\_CLIENT\_ESTABLISHED:

printf(KYEL"[Main Service] Connect with server success.\n"RESET);

connection\_flag = 1;

break;

case LWS\_CALLBACK\_CLIENT\_CONNECTION\_ERROR:

printf(KRED"[Main Service] Connect with server error.\n"RESET);

destroy\_flag = 1;

connection\_flag = 0;

break;

case LWS\_CALLBACK\_CLOSED:

printf(KYEL"[Main Service] LWS\_CALLBACK\_CLOSED\n"RESET);

destroy\_flag = 1;

connection\_flag = 0;

break;

case LWS\_CALLBACK\_CLIENT\_RECEIVE:

printf(KCYN\_L"[Main Service] Client recvived:%s\n"RESET, (char \*)in);

if (writeable\_flag)

destroy\_flag = 1;

break;

case LWS\_CALLBACK\_CLIENT\_WRITEABLE :

printf(KYEL"[Main Service] On writeable is called. send byebye message\n"RESET);

websocket\_write\_back(wsi, "Byebye! See you later", -1);

writeable\_flag = 1;

break;

default:

break;

}

return 0;

}

static void \*pthread\_routine(void \*tool\_in)

{

struct pthread\_routine\_tool \*tool = (struct pthread\_routine\_tool\*)tool\_in;

printf(KBRN"[pthread\_routine] Good day. This is pthread\_routine.\n"RESET);

*//\* waiting for connection with server done.\*/*

while(!connection\_flag)

usleep(1000\*20);

*//\*Send greeting to server\*/*

printf(KBRN"[pthread\_routine] Server is ready. send a greeting message to server.\n"RESET);

websocket\_write\_back(tool->wsi, "Good day", -1);

printf(KBRN"[pthread\_routine] sleep 2 seconds then call onWritable\n"RESET);

sleep(1);

printf(KBRN"------------------------------------------------------\n"RESET);

sleep(1);

*//printf(KBRN"[pthread\_routine] sleep 2 seconds then call onWritable\n"RESET);*

*//\*involked wriable\*/*

printf(KBRN"[pthread\_routine] call on writable.\n"RESET);

lws\_callback\_on\_writable(tool->wsi);

}

int main(void)

{

*//\* register the signal SIGINT handler \*/*

struct sigaction act;

act.sa\_handler = INT\_HANDLER;

act.sa\_flags = 0;

sigemptyset(&act.sa\_mask);

sigaction( SIGINT, &act, 0);

struct lws\_context \*context = NULL;

struct lws\_context\_creation\_info info;

struct lws \*wsi = NULL;

struct lws\_protocols protocol;

memset(&info, 0, sizeof info);

info.port = CONTEXT\_PORT\_NO\_LISTEN;

info.iface = NULL;

info.protocols = &protocol;

info.ssl\_cert\_filepath = NULL;

info.ssl\_private\_key\_filepath = NULL;

info.extensions = lws\_get\_internal\_extensions();

info.gid = -1;

info.uid = -1;

info.options = 0;

protocol.name = "my-echo-protocol";

protocol.callback = &ws\_service\_callback;

protocol.per\_session\_data\_size = sizeof(struct session\_data);

protocol.rx\_buffer\_size = 0;

protocol.id = 0;

protocol.user = NULL;

context = lws\_create\_context(&info);

printf(KRED"[Main] context created.\n"RESET);

if (context == NULL) {

printf(KRED"[Main] context is NULL.\n"RESET);

return -1;

}

wsi = lws\_client\_connect(context, "localhost", 5000, 0,

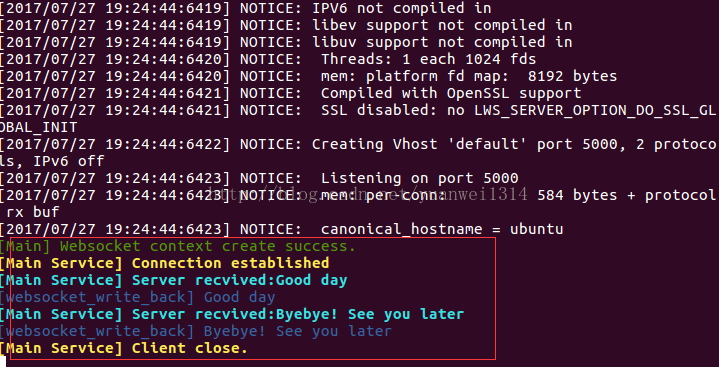
"/", "localhost:5000", NULL, protocol.name, -1); if (wsi == NULL) { printf(KRED"[Main] wsi create error.\n"RESET); return -1; } printf(KGRN"[Main] wsi create success.\n"RESET); struct pthread\_routine\_tool tool; tool.wsi = wsi; tool.context = context; pthread\_t pid; pthread\_create(&pid, NULL, pthread\_routine, &tool); pthread\_detach(pid); while(!destroy\_flag) { lws\_service(context, 50); } lws\_context\_destroy(context); return 0; } 编写脚本:

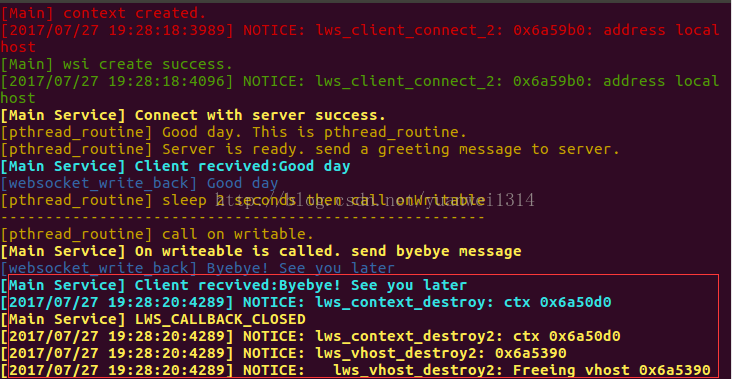
#!/bin/sh

libdir=libwebsockets

g++ -g -o ws\_client test\_ws\_client.cpp -I$libdir/lib -I$libdir/build -L$libdir/build/lib -lwebsockets -lpthread

**运行结果**





红线部分是交互的结果.这样就是完整的一套逻辑了.