# Lthread C++11 Bindings Documentation

***Release 1.0***

**Hasan Alayli**

November 07, 2014

Contents

CHAPTER 1

# Introduction

lthread_cpp is a C++11 binding to C's lthread version.

Currently, lthread is supported on FreeBSD, OS X, and Linux (x86 & 64bit arch).

## 1.1 Installation & Linking

```
https://github.com/halayli/lthread_cpp.git
cd lthread_cpp
cmake .
sudo make install
```

### 1.1.1 Linking

Pass *-llthread_cpp* to your compiler to use lthread_cpp in your program.

## 1.2 License

CHAPTER 2

# Lthread

```
#include <lthread_cpp/lthread.h>
using namespace lthread;
```

**class Lthread**

Launches a single lthread in the background.

## 2.1 Member Functions

**Lthread**()

**Lthread(&Method, params,...)**

**Lthread(&Class::Method, params,...)**

Creates new *lthread* object and associates it with an lthread. The constructor copies/moves all arguments args... to an lthread-accessible storage.

**Join**(uint64_t *timeout_ms*)
Joins on a single lthread and blocks until the lthread returns.

**Parameters timeout_ms(optional, default=0)** Milliseconds to wait joining on another lthread.

**Throws** `LthreadTimeout()` on timeout.

void **Detach**()
Marks the lthread launched as detachable to be freed upon return. This is a direct binding to *lthread_detach*

lthread_t* **Id**()

**Returns** an lthread_t* ptr pointing to the original lthread created by *lthread_create*

bool **Joinable**()

**Returns** *true* if the lthread can be joined on(i.e., launched)

**Note:** Lthread objects are movable but not copyable.

## 2.2 Exceptions

### 2.2.1 LthreadTimeout

**class LthreadTimeout**
Empty class raised on Lthread *join* timeout call.

## 2.3 Example

```
#include <vector>
#include <lthread_cpp/lthread.h>

using namespace lthread_cpp;

void MyMethod(std::vector<int> my_vec) {}

void Run()
{
  std::vector<int> v{1,2,3,4};
  Lthread t1{&MyMethod, v};
  t1.Detach();
}

int main()
{
  Lthread t{&Run};
  t.Detach();
  Lthread::Run();
}
```

```
cc -std=c++11 test.cc -o test -llthread_cpp -llthread -lpthread -lstdc++ && ./test
```

CHAPTER 3

# Socket

```
#include <lthread_cpp/socket.h>
using namespace lthread::net;
```

**class Socket**

A wrapper around lthread's socket calls. *Socket* instance is returned by `TcpConnect()` and `TcpListener` and cannot be constructed on its own.

## 3.1 Member Functions

These functions reflect their lthread equivalent and must be called inside lthreads.

size_t **Send**(const char* *buf*)
Sends a C-style string over a socket.

**Parameters const char* buf** NULL-terminated buffer.

**Returns** Number of bytes sent.

**Throws** `SocketException` on socket failure.

size_t **Send**(const char* *buf*, size_t *length*)
Sends *length* bytes over a socket.

**Parameters**

- **const char* buf** – Ptr to buffer containing data to send.
- **size_t length** – Number of bytes to send from *buf*.

**Returns** Number of bytes sent.

**Throws** `SocketException` on socket failure.

size_t **Recv**(char* *buf*, size_t *length*, int *timeout_ms=1000*)
Receives up to *length* bytes over a socket.

**Parameters**

- **char* buf** – Buffer to read data into.
- **size_t length** – Buffer size to fill.
- **timeout_ms(optional, default=1000)** – Milliseconds to wait before timing out.

**Throws** `SocketException` on socket failure.

**Throws** SocketTimeout if a timeout occured. *timeout_ms=0* waits indefinitely.

void **Close**()
Closes the network socket.

size_t **Writev**(struct iovec* *v*, int *iovcnt*)
Sends an iovec over a socket.

**Parameters**

- **struct iovec* v** – iovec pointing to one or more ptr/size entries.
- **int iovcnt** – Number of entries in the iovec.

**Throws** SocketException on socket failure.

size_t **RecvExact**(char* *buf*, size_t *length*, int *timeout_ms=1000*)
Receives exactly *length* bytes into buf.

**Parameters**

- **char* buf** – Buffer to read data into.
- **size_t length** – Buffer size to fill.
- **timeout_ms(optional, default=1000)** – Milliseconds to wait before timing out.

**Throws** SocketException on socket failure.

**Throws** SocketTimeout if it timed out before receiving the full number of bytes.

void **WaitWrite**(int *timeout_ms=1000*) const
Waits until the socket is writable.

**Parameters timeout_ms(optional, default=1000)** Milliseconds to wait before timing out.

**Throws** SocketException on socket failure.

**Throws** SocketTimeout if timeout occured.

void **WaitRead**(int *timeout_ms=1000*) const
Waits until the socket is readable.

**Parameters timeout_ms(optional, default=1000)** Milliseconds to wait before timing out.

**Throws** SocketException on socket failure.

**Throws** SocketTimeout if timeout occured.

bool **IsConnected**() const
Returns true if socket is connected.

int **fd**() const
Returns the fd wrapped in the Socket instance.

std::string **Ip**() const
Returns the remote IP Address as a string.

**Returns** string containing IP address.

std::string **Desc**() const
Returns remote_ip:ephemeral_port as a string

Socket& **operator=**(Socket&& *rr_c*)
Moves a socket from one instance to another.

---

**Note:** Socket objects are movable but not copyable.

---

## 3.2 Exceptions

### 3.2.1 SocketTimeout

**class** `SocketTimeout`
Empty class raised on socket timeout operations.

### 3.2.2 SocketException

**class** `SocketException`
Inherits std::exception(), raised on socket errors.

CHAPTER 4

# TcpListener

```
#include <lthread_cpp/listener.h>

using namespace lthread_cpp::net;
```

**class `TcpListener`**

## 4.1 Member Functions

**`TcpListener`**(const std::string& *ip*, short int *port*)
Initializes *TcpListener* instance with the ip and port specified.

void **`Listen`**()
Binds IP and port to socket.

**Throws** `SocketException` if it fails to bind or listen.

Socket **`Accept`**()
Blocks until a new connection is accepted.

**Returns** A new `Socket` object for the new connection.

**Throws** `SocketException` if *lthread_accept()* failed.

void **`Close`**()
Closes listening port.

## 4.2 Examples

```
#include <lthread_cpp/lthread.h>
#include <lthread_cpp/socket.h>
#include <lthread_cpp/listener.h>

using namespace lthread_cpp;
using namespace lthread_cpp::net;

void HandleConnection(Socket& s)
{
  s.Send("Hi");
}
```

```
void Run()
{
  TcpListener listener("127.0.0.1", 8090);
  listener.Listen();
  while (1) {
    Socket s = listener.Accept();
    Lthread t1 {&HandleConnection, std::move(s)};
    t1.Detach();
  }
}

int main()
{
    Lthread t{&Run};
    t.Detach();
    Lthread::Run();
}

cc -std=c++11 test.cc -o test -llthread_cpp -llthread -lpthread -lstdc++ && ./test
```

CHAPTER 5

# TcpConnect

```
#include <lthread_cpp/socket.h>
```

Socket **TcpConnect**(const std::string& *host_or_ip*, short int *port*, int *timeout_ms=1000*)

Connects to a remote host.

**Parameters**

- **const std::string& host_or_ip** – Host to connect to.
- **short port** – Tcp port.
- **int timeout_ms(1000)** – Milliseconds to wait connecting.

**Returns** `Socket` for the new connection

**Throws** `SocketException` on socket failure.

## 5.1 Examples

```
#include <lthread_cpp/lthread.h>
#include <lthread_cpp/socket.h>

using namespace lthread_cpp;
using namespace lthread_cpp::net;

void Run()
{
 Socket s = TcpConnect("127.0.0.1", 80);
 s.Send("GET / HTTP/1.1\r\n\r\n");

 char response[1024];
 s.Recv(response, 1024, 0);
 s.Send("Cool!");
 // s closes as it goes out of scope
}

int main()
{
  Lthread t{&Run};
  t.Detach();
  Lthread::Run();
}
```

```
cc -std=c++11 test.cc -o test -llthread_cpp -llthread -lpthread -lstdc++ && ./test
```

CHAPTER 6

# SSLSocket

```
#include <lthread_cpp/ssl.h>
using namespace lthread_cpp::net;
```

**class SSLSocket**

Turns a `Socket()` to `SSLSocket()`

**Attention:** You must call lthread_cpp::net::SSLSocket::init() once before any SSLSocket connection is received or established.

## 6.1 Member Functions

**static** void **Init**(const std::string& *server_pem_filename*, const std::string& *server_key_filename*, const std::string& *ca_cert_filename*, const std::string& *ca_path*)

Initializes SSL settings.

**Throws** `SSLException` if it failed to initialize SSL context with any of the values provided.

**SSLSocket**(Socket&& *s*)

Initializes/wraps a new SSLSocket from an existing established `Socket`. Requires calling either `SSLSocket::Accept()` or `SSLSocket::Connect()` afterwards depending on whether the underlying TCP connection was accepted by the listener using `Accept()` or established via `TcpConnect()`.

**SSLSocket**()

Initializes n new SSLSocket ready to connect to peer using `SSLSocket::Connect()`.

void **Accept**(int *timeout_ms=5000*)

Initiates an SSL Accept with the assumption that the TCP connection was accept(2)-ed and not established via connect(2).

**Throws** `SSLException` if ssl accept failed.

void **Connect**(const std::string& *host_or_ip*, short int *port*, int *timeout_ms*)

Establishes a TCP connection to host/ip:port and initiates an SSL Connect afterwards.

**Throws** `SSLException` if SSL connect failed

**Throws** `SocketException` on socket failure.

void **RequirePeerVerification**()

Will set SSL peer verification flag on.

std::string **GetCertCommonName**()
Returns common name in certificate received.

size_t **Send**(const char* *buf*, int *timeout_ms=5000*)
Sends a C style string over SSL socket.

**Parameters const char* buf** NULL-terminated buffer.

**Throws** SSLException on socket failure.

size_t **Send**(const char* *buf*, size_t *length*, int *timeout_ms=5000*)
Sends length bytes of buf over SSL socket.

**Parameters**

- **const char* buf** – Ptr to buffer containing data to send.
- **size_t length** – Number of bytes to send from *buf*.
- **timeout_ms(optional, default=5000)** – Milliseconds to wait before timing out.

**Throws** SSLException on socket failure.

size_t **Recv**(char* *buf*, size_t *length*, int *timeout_ms=5000*)
Receives up to length bytes and place them into buf.

**Parameters**

- **char* buf** – Buffer to read data into.
- **size_t length** – Buffer size to fill.
- **timeout_ms(optional, default=5000)** – Milliseconds to wait before timing out.

**Throws** SSLException on socket failure.

void **Close**()
Cleanly closes SSL socket and its underlying TCP connection.

---

**Note:** SSL objects are movable but not copyable.

---

## 6.2 Exceptions

### 6.2.1 SSLException

**class SSLException**
Inherits SocketException, raised on SSL errors.

## 6.3 Examples

```
using namespace lthread;
using namespace lthread::net;

void Proxy::HandleConnection(Socket& tcp_conn)
{

  SSLSocket client;
  std::string common_name;
```

```
  // do an SSL handshake over the new tcp connection we just received and grab
  // the required customer certificate after it has been verified against
  // CA certificates provided to SSLSocket::Init
  try {
    SSLSocket ssl_socket(std::move(tcp_conn));
    ssl_socket.RequirePeerVerification();
    ssl_socket.Accept();
    common_name = ssl_socket.GetCertCommonName();
    client = std::move(ssl_socket);
  } catch (SocketException& e) {
    LOG(ERROR) << "SSL handshake failed from "
        << tcp_conn.Desc() << ". (" << e.what() << ")";
    return;
  }

  // At this point, client can send/recv bytes over established SSL
      client.Send("hello world!\n");
}
```