hURL

A file transfer library

Jennifer Bi, Maria Javier, Aryeh Zapinsky

jb3495@columbia.edu, mj2729@columbia.edu, aryeh.zapinsky@columbia.edu

Spring 2018

- Inspired by curl/libcurl command line tool + library
    - curl supports:

      DICT, FILE, FTP, FTPS, Gopher, **HTTP, HTTPS,** IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMB, SMBS, SMTP, SMTPS, Telnet and TFTP
    - ~122,700 lines of code
- Implemented with ASIO 1.12.1 (non-Boost)

Motivation

We’re migrating over to C++

Most of the developers use ASIO

cURL like interface, features

Motivation

We’re migrating over to C++

Most of the developers use ASIO

cURL like interface, features

Great opportunity to use Modern C++ !

Performing a transfer

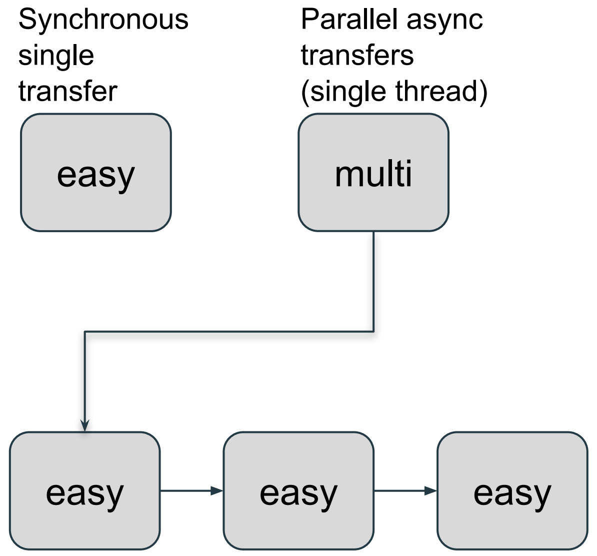
1. create handle
2. set options (configurability is important)
3. perform transfer
4. extract info upon completion (via callbacks)
5. repeat (handles are designed to be reused)

Modern C++ Design principles

- RAII
  - Global init
  - Global cleanup
  - Only call once at beginning and end of program
- Type safety
  - va_arg macro
- Exception handling
  - CURLcode

Design of cURL

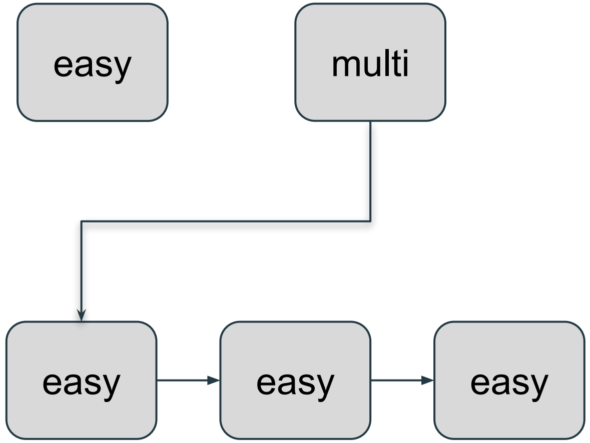
```
CURL *curl = curl_easy_init();
if(curl) {
  CURLcode res;
  curl_easy_setopt(curl, CURLOPT_URL,
"http://example.com");
  res = curl_easy_perform(curl);
  curl_easy_cleanup(curl);
}
```


Synchronous single transfer
Parallel async transfers (single thread)
easy
multi
easy
easy
easy

Code from:
https://curl.haxx.se/libcurl/c/curl_easy_perform.html

Design of cURL

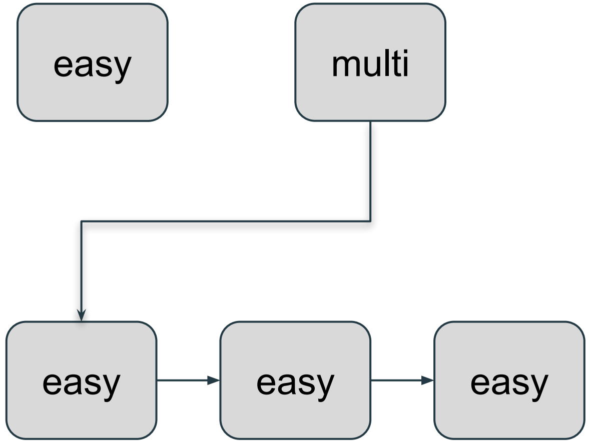
```
CURL *curl = curl_easy_init();
if(curl) {
  CURLcode res;
  curl_easy_setopt(curl, CURLOPT_URL,
"http://example.com");
  res = curl_easy_perform(curl);
  curl_easy_cleanup(curl);
}
```



Code from:
https://curl.haxx.se/libcurl/c/curl_easy_perform.html

Design of cURL

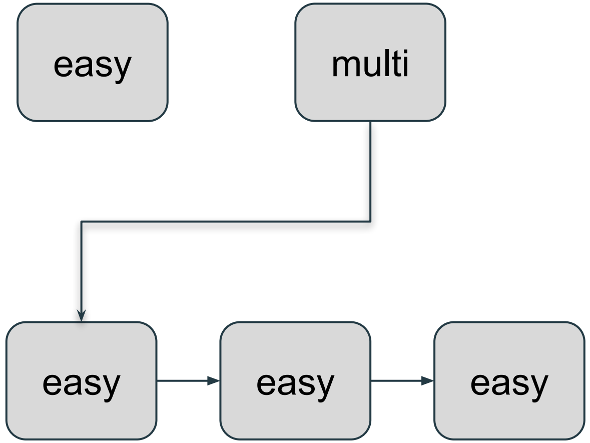
```
CURL *curl = curl_easy_init();
if(curl) {
  CURLcode res;
  curl_easy_setopt(curl, CURLOPT_URL,
"http://example.com");
  res = curl_easy_perform(curl);
  curl_easy_cleanup(curl);
}
```



Code from:
https://curl.haxx.se/libcurl/c/curl_easy_perform.html

Design of cURL

```
CURL *curl = curl_easy_init();
if(curl) {
  CURLcode res;
  curl_easy_setopt(curl, CURLOPT_URL,
"http://example.com");
  res = curl_easy_perform(curl);
  curl_easy_cleanup(curl);
}
```



Code from:
https://curl.haxx.se/libcurl/c/curl_easy_perform.html

Design of hURL (using RAII)

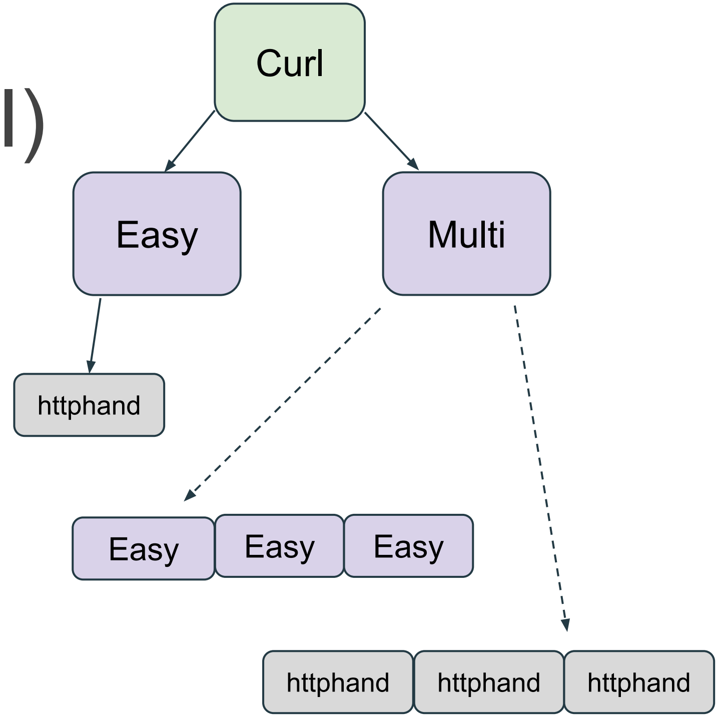
`Easy::perform, Multi::perform`

→ `httphand` goes in and out of scope in perform()

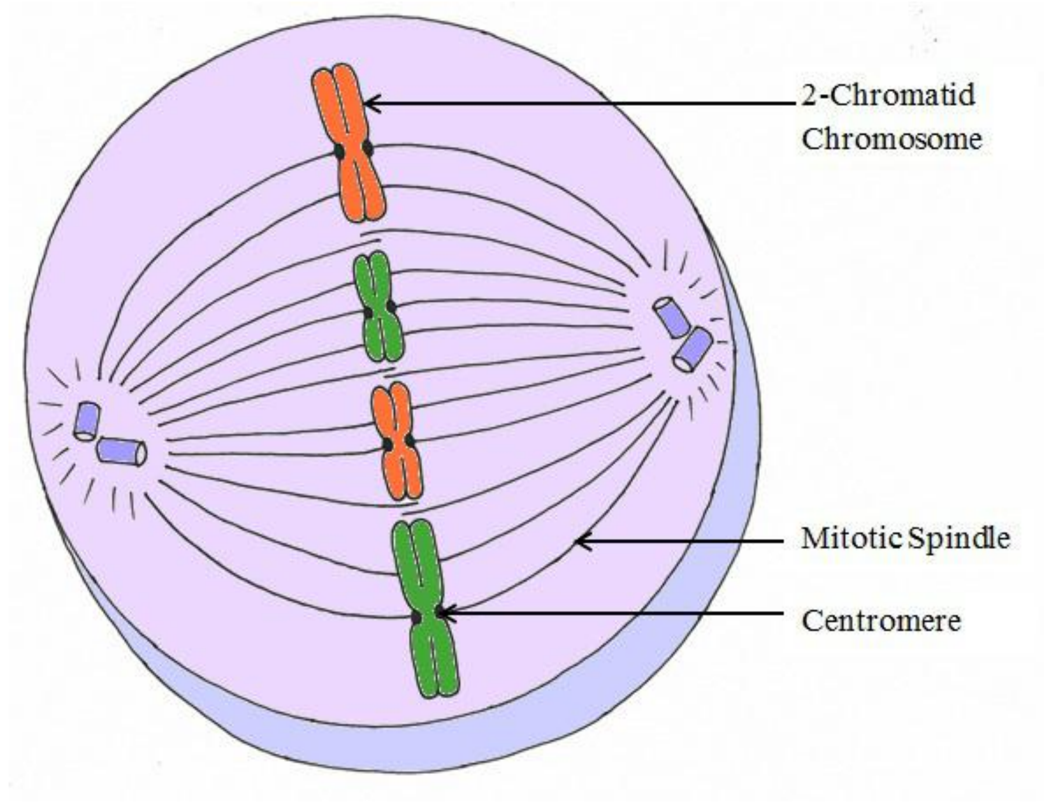
→ `struct UserDefined` belongs to Easy and Multi handles via shared_ptr

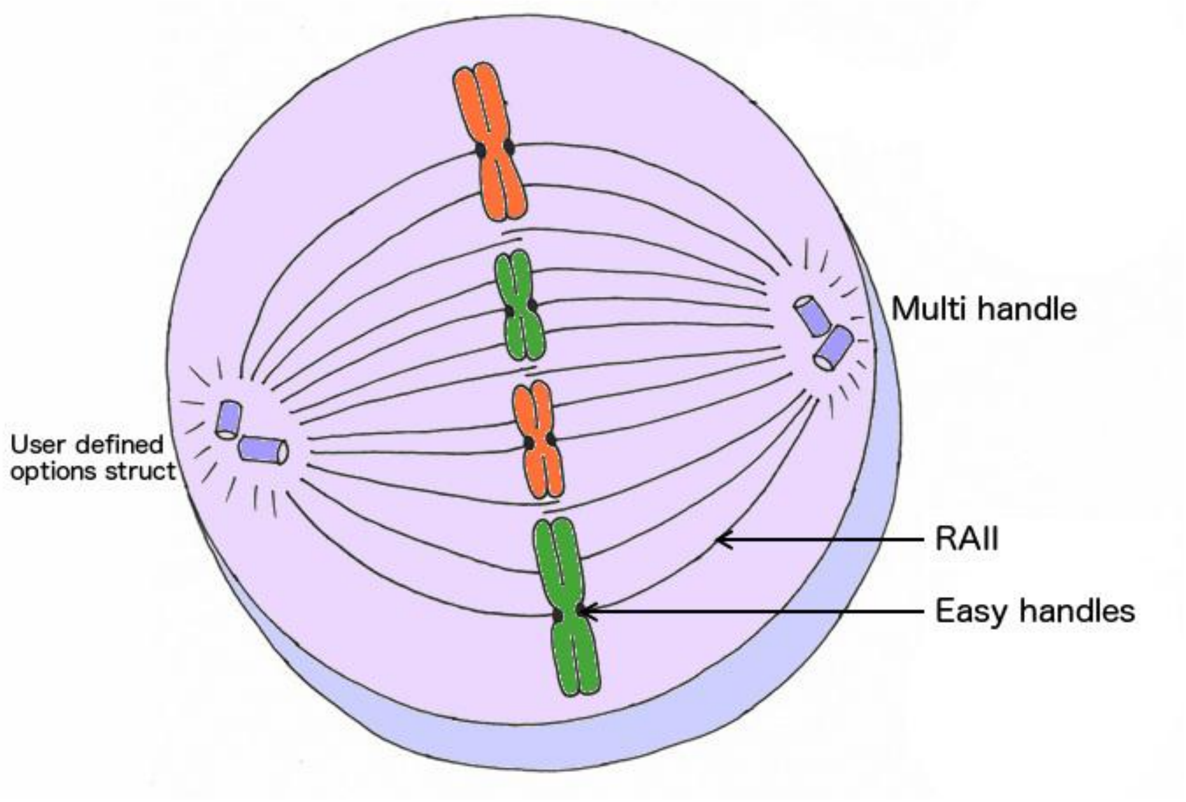
The upshot: httphand is disposable Asio service,
Easy handles have user defined attributes, persistent!

→ libcurl is thread safe but has no internal thread synchronization

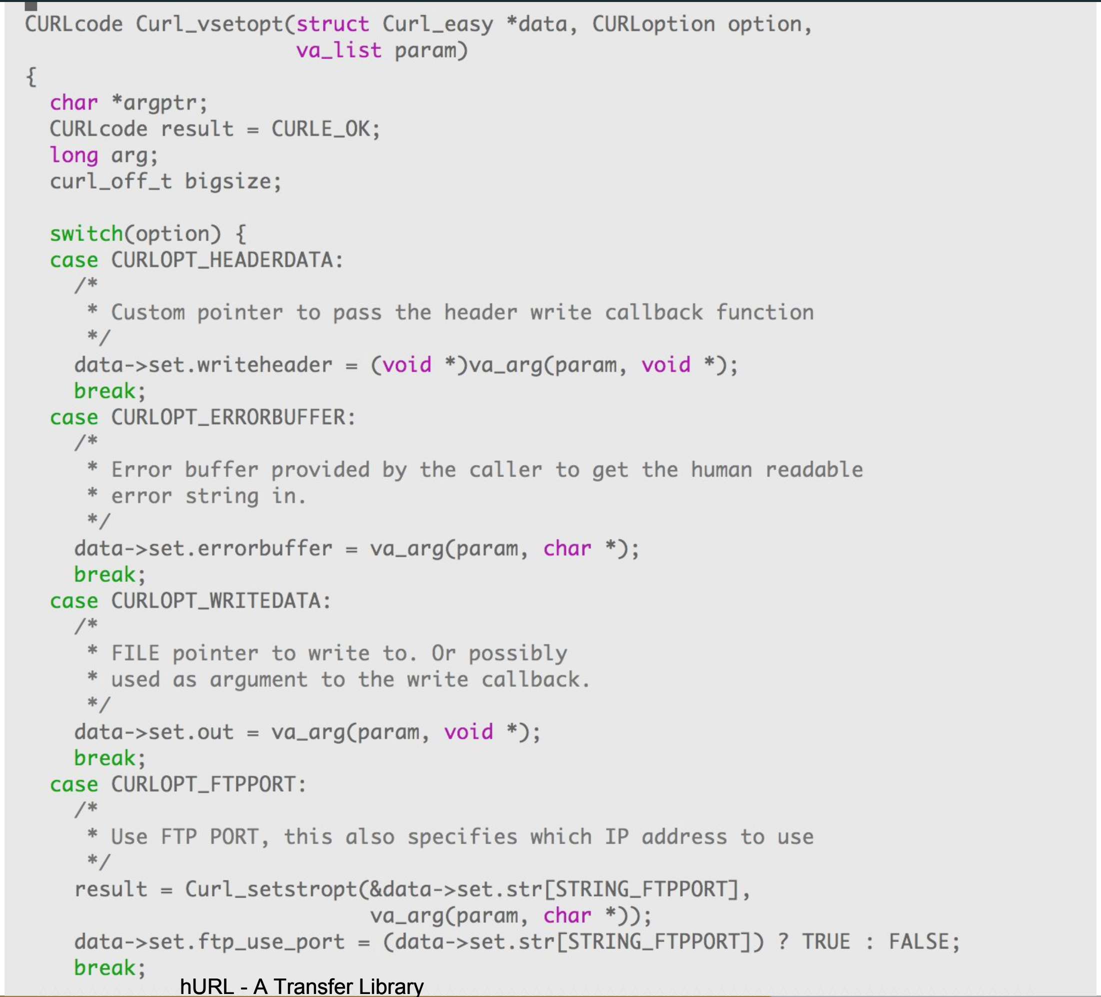


Code from: https://curl.haxx.se/libcurl/c/curl_easy_perform.html


2-Chromatid Chromosome
Mitotic Spindle
Centromere


Multi handle
User defined options struct
RAII
Easy handles

Typesafe ?



```
CURLcode Curl_vsetopt(struct Curl_easy *data, CURLoption option,
                      va_list param)
{
  char *argptr;
  CURLcode result = CURLE_OK;
  long arg;
  curl_off_t bigsize;

  switch(option) {
  case CURLOPT_HEADERDATA:
    /*
     * Custom pointer to pass the header write callback function
     */
    data->set.writeheader = (void *)va_arg(param, void *);
    break;
  case CURLOPT_ERRORBUFFER:
    /*
     * Error buffer provided by the caller to get the human readable
     * error string in.
     */
    data->set.errorbuffer = va_arg(param, char *);
    break;
  case CURLOPT_WRITEDATA:
    /*
     * FILE pointer to write to. Or possibly
     * used as argument to the write callback.
     */
    data->set.out = va_arg(param, void *);
    break;
  case CURLOPT_FTPPORT:
    /*
     * Use FTP PORT, this also specifies which IP address to use
     */
    result = Curl_setstropt(&data->set.str[STRING_FTPPORT],
                            va_arg(param, char *));
    data->set.ftp_use_port = (data->set.str[STRING_FTPPORT]) ? TRUE : FALSE;
    break;
```

Variadic Template Matching

```
template<typename T>
void Curl::setOpt(int a, T b) {
    Curl::_setopt(a, b);
}

template<typename T, typename... Args>
void Curl::setOpt(int a, T b, Args... args) {
    Curl::_setopt(a, b);
    Curl::setOpt(args...);
}
```

A great article: https://eli.thegreenplace.net/2014/variadic-templates-in-c/

Variadic Template Matching

```
void Curl::_setopt(int a, string b) {
    switch (a) {
    case CURLPP_OPT_URL:
        parse_url(static_cast<string>(b));
        break;
    case CURLPP_OPT_HOST:
        defs->host = static_cast<string>(b);
        break;
    case CURLPP_OPT_PATH:
        defs->path = static_cast<string>(b);
        break;
    case CURLPP_OPT_SSLCERT:
        defs->clientcert = static_cast<string>(b);
        break;
    default: std::cerr << "Error: CURLPP_OPT not yet supported" << "\n";
    }
}
```

Variadic Template Matching

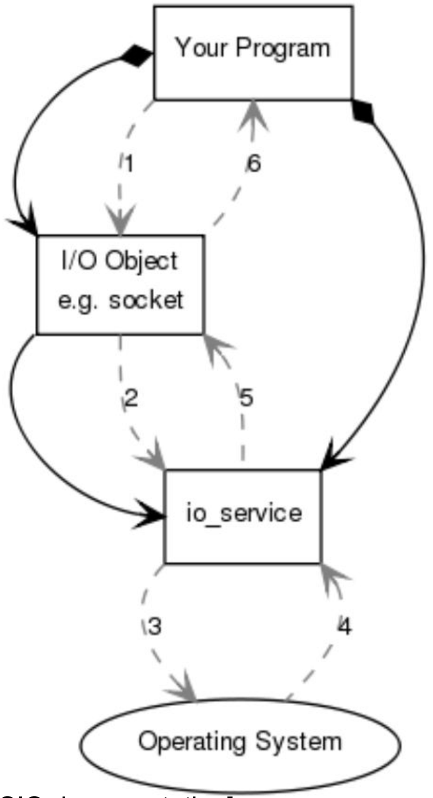
```
void Curl::_setopt(int a, std::function<int(const unsigned char *, std::size_t)>
 f){
    switch (a) {
    case CURLPP_OPT_WRITEFN:
        defs->writeback = f;
        break;
    case CURLPP_OPT_READFN:
        defs->readback = f;
        break;
    default: std::cerr << "Error: CURLPP_OPT not yet supported" << "\n";
    }
}
```

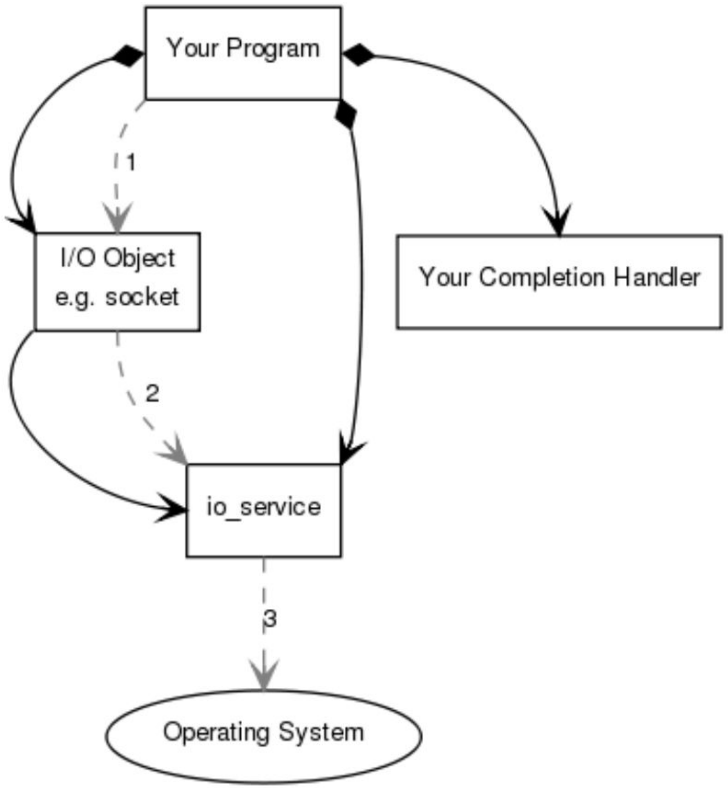
Asio overview, and why we chose to use it

Rationale given by Asio reference manual:

- **Portability*, scalability, ease of use***
- **Efficiency.** “The library should support techniques such as scatter-gather I/O, and allow programs to minimise data copying.”
- **Basis for further abstraction.** “The library should permit the development of other libraries that provide higher levels of abstraction. For example, implementations of commonly used protocols such as HTTP.”

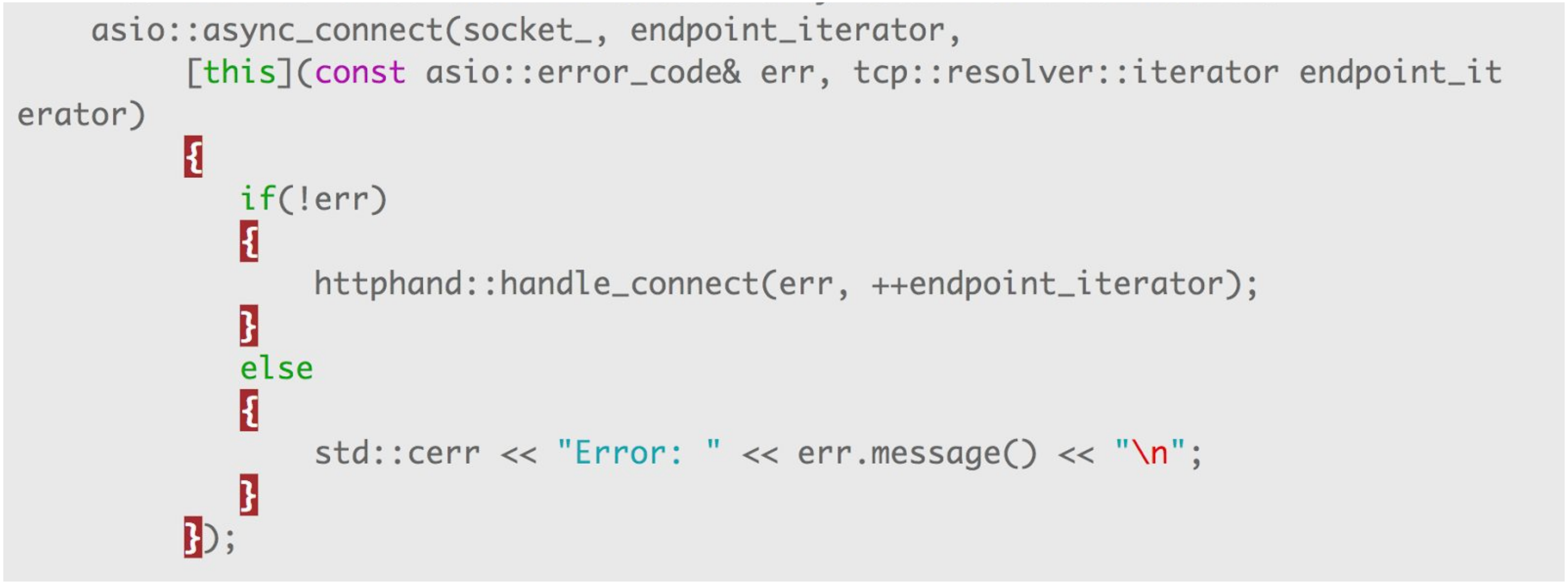
Synchronous vs Asynchronous operations





[Images from ASIO documentation]

Using completion handlers



```
    asio::async_connect(socket_, endpoint_iterator,
        [this](const asio::error_code& err, tcp::resolver::iterator endpoint_it
erator)
        {
            if(!err)
            {
                httphand::handle_connect(err, ++endpoint_iterator);
            }
            else
            {
                std::cerr << "Error: " << err.message() << "\n";
            }
        });
```

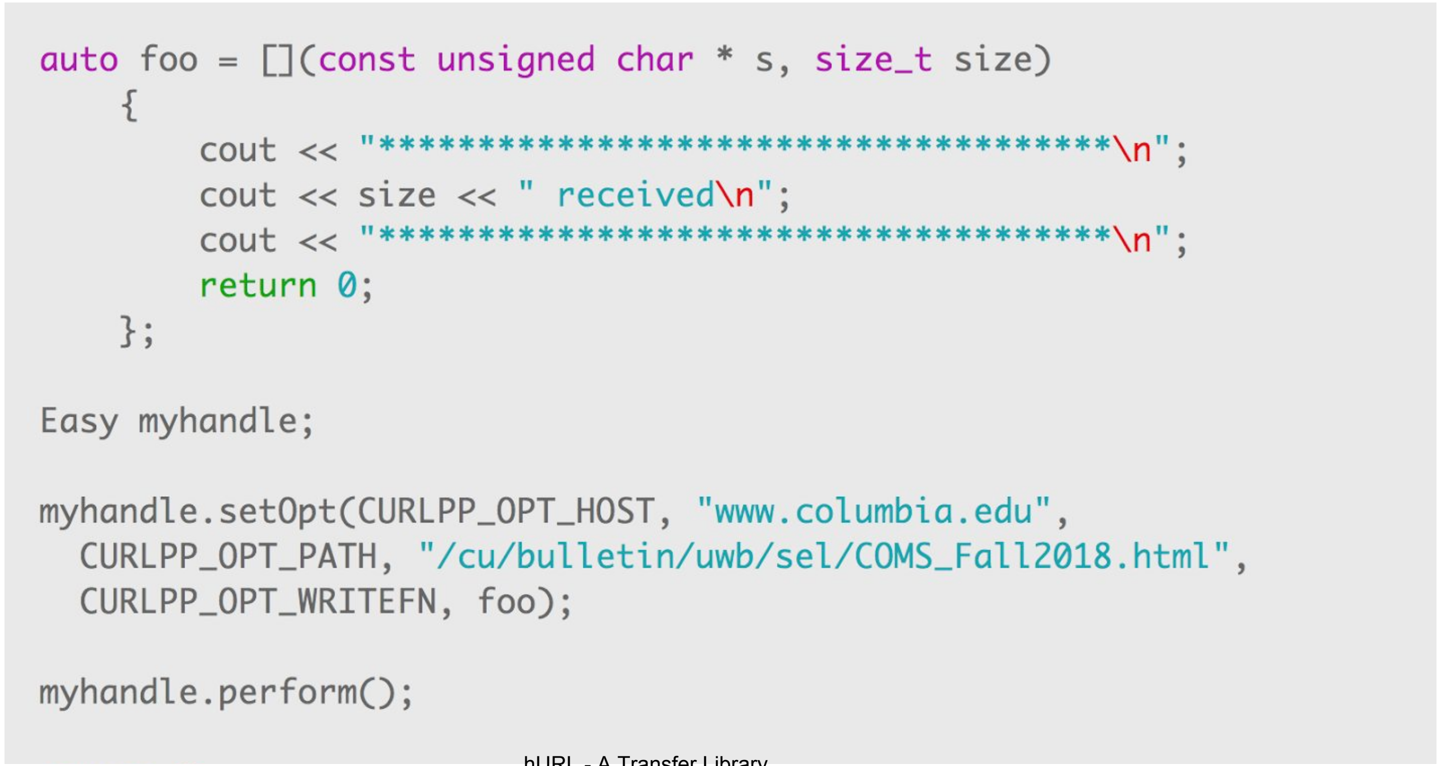
httphand, sslhand

- inherit from abstract class `protocol` which defines interface between Easy/Multi handles and Asio functionality
- Allows for future additions to support as many protocols as curl
- Implemented with chains of lambda completion handlers

  → memory allocation optimization!

- Multi threading

Using Lambdas



```
auto foo = [](const unsigned char * s, size_t size)
    {
        cout << "*************************************\n";
        cout << size << " received\n";
        cout << "*************************************\n";
        return 0;
    };

Easy myhandle;

myhandle.setOpt(CURLPP_OPT_HOST, "www.columbia.edu",
  CURLPP_OPT_PATH, "/cu/bulletin/uwb/sel/COMS_Fall2018.html",
  CURLPP_OPT_WRITEFN, foo);

myhandle.perform();
```

Results

	HTTP		HTTPS	
	KB	sec	KB	sec
	23	0.168	69	0.095
	746	0.120	219	0.219
	778000	95.416	222	0.124
hURL			14000	1.646
	HTTP		HTTPS	
	KB	sec	KB	sec
	23	0.173	69	0.102
	746	0.109	219	0.227
	778000	87.984	222	0.119
cURL			14000	1.249

<u>Environment:</u>
macOS 10.13.2
3.1 GHz Intel Core i7
Compiler: Apple LLVM version 9.1.0 (clang-902.0.39.1)
-O2 flag

Stuff we’ve hURLed

- Large data-sets (COCO) ~ 6GB

…

www.goZeroG.com
zero G
THE SHARPER IMAGE
SPACE FLORIDA

Challenges

Is Asio portable?

- Non-boost version requires boost in examples
- OpenSSL, deprecated X509 hash functions

→ causes SSL handshake with server to fail :(

HTTP connection, not HTTPS

Version 1.2

- caches for connections and DNS names
- Support for other protocols
- single-threaded parallel transfers using coroutines
- offloading long read operations to a separate thread using asio::post and asio::dispatch message passing.

Acknowledgements

This work would not have been possible without Professor Stroustrup.

No, literally.

We’d like to thank Kai-Zhan and Abhishek for reading our proposal, giving us feedback, and being there for us.