snode.c

Volker Christian

October 21, 2020

snode.c

- What is snode.c
- Feature List (not complete)
- Copyright
  - Initiator and Main developer
  - Contributors (Students)
    - Json Middleware
    - Regular-Expression Route-Mapping
    - Runtime Module Loading
- Requirements
- Components
- TODOs
- Example Applications
  - Very Simple HTTP/S Web-Server for static HTML-Pages
  - Receive Data by HTTP-Post Request
  - Extract Server and Client Information (host name, IP, port, SSL/TLS information)
  - Using Regular Expressions in Routes

What is snode.c

snode.c (Simple NODE in C++) is a lightweight, highly extendable, high-performance layer-based framework for network applications (servers and clients) in the spirit of node.js written entirely in C++. The development of the framework started during the summer semester 2020 in the context of the course **Network and Distributed Systems** of the masters program **Interactive Media** at the departement **Informatics, Communications and Media** at the **University of Applied Sciences Upper Austria, Campus Hagenberg** to give students an insight into the fundamental techniques of network and web frameworks. Main focus (but not only) of the framework is “Machine to Machine” (M2M) communication and here especially the field of “Internet of Things” (IoT). Keep in mind, that the framework is still in heavy development, APIs can break from commit to commit. But the highest level API (express) is considered stable.

Feature List (not complete)

- Non-blocking, event-driven (asynchronous), single-threaded, single-tasking, layer-based design that supports timer (single-shot, interval) and network communication (TCP/IPv4).
- Application protocol independent TCP server and TCP client functionality. Application protocols (HTTP, ...) can be easily and modularly connected.
- TLS/SSL encryption for server and client.
- Support of X.509 certificates. TLS/SSL connections can be protected and authenticated using certificates.
- Fully implemented HTTP(S) application protocol layer (server and client) to which the application logic can be easily connected.
- High-Level Web API layer with JSON support similar to the API of node.js/express. (Speedup compared to node.js/express approx. 40)

Copyright

GNU Lesser General Public License

Initiator and Main developer

- Volker Christian (me_AT_vchrist.at, volker.christian_AT_fh-hagenberg.at)

Contributors (Students)

Json Middleware

- Marlene Mayr
- Anna Moser
- Matteo Prock
- Eric Thalhammer

Regular-Expression Route-Mapping

- Joelle Helgert
- Julia Gruber
- Patrick Brandstätter
- Fabian Mohr

Requirements

- GCC Version 10: As snode.c uses most recent C++-20 language features
- libeasyloggingpp: For logging
- openssl: For SSL/TLS support
- doxygen: For creating the API-documentation
- cmake:
- iwyu: To check for correct and complete included headers
- libmagic: To recognize the type of data in a file using “magic” numbers
- clang-format: To format the sourcecode consistently
- nlohmann-json3-dev: For JSON support

Components

- libnet (directory net) low level multiplexed network-layer (server/client) with event-loop supporting legacy- and tls-connections
- libhttp (directory http) low level http server and client
- libexpress (directory express) high level web-api similar to node.js's express module
- example applications (directory apps)

TODOs

- Add better error processing in HTTPResponseParser
- Extend regex-match in Router (path-to-regex in JS)
- Add some more complex example apps (Game, Skill for Alexa, ...)
- Add WebSocket support for server and client, legacy and tls
- Add support for Template-Engines (similar to express)
- Add additional transport protocols (udp, bluetooth, local, ipv6, ...)
- Implement other protocols (contexts) than http (e.g. mqtt, ftp would be interresting, telnet, smtp)
- Finalize cmake-rules to support install
- Add cmake description files
- Add “logical” events
- Support “promisses”
- Add OAUTH2 server/client authentification
- Write unit tests
- Porting snode.c to MS-Windows
- Porting snode.c to macOS

Receive Data via HTTP-Post Request

```
#include <easylogging++.h>

#include "legacy/WebApp.h"
#include "tls/WebApp.h"

#define CERTF <PATH_TO_SERVER_CERTIFICATE_CHAIN_FILE>
#define KEYF <PATH_TO_SERVER_CERTIFICATE_KEY_FILE>
#define KEYFPASS <PASSWORD_OF_THE_SERVER_CERTIFICATE_KEY_FI

int main(int argc, char* argv[]) {
    express::WebApp::init(argc, argv);

    express::legacy::WebApp legacyApp;
    legacyApp.get("/", [] APPLICATION(req, res) {
        res.send("<html>"
                 "    <head>"
                 "        <style>"
                 "            main {"
```

Extract Server and Client Information (host name, IP, port, SSL/TLS information)

```
#include <easylogging++.h>
#include <openssl/x509v3.h>

#include "legacy/WebApp.h"
#include "tls/WebApp.h"
#include "middleware/StaticMiddleware.h"

#define CERTF <PATH_TO_SERVER_CERTIFICATE_CHAIN_FILE>
#define KEYF <PATH_TO_SERVER_CERTIFICATE_KEY_FILE>
#define KEYFPASS <PASSWORD_OF_THE_SERVER_CERTIFICATE_KEY_F

#define SERVERROOT <PATH_TO_ROOT_OF_WEBSITE>

using namespace express;

Router getRouter() {
    Router router;
```

Using Regular Expressions in Routes

```
#include "legacy/WebApp.h"
#include "tls/WebApp.h"

#include <easylogging++.h>

using namespace express;

Router router() {
    Router router;

    // http://localhost:8080/account/123/username
    router.get("/account/:userId(\\d*)/:username", [] APPLI
        VLOG(0) << "Show account of";
        VLOG(0) << "UserId: " << req.params["userId"];
        VLOG(0) << "UserName: " << req.params["userName"];

        std::string response = "<html>"
                               "  <head>"
```