llhttp

CI passing

Port of http_parser to llparse.

Why?

Let's face it, http_parser is practically unmaintainable. Even introduction of a single new method results in a significant code churn.

This project aims to:

- Make it maintainable
- Verifiable
- Improving benchmarks where possible

More details in Fedor Indutny's talk at JSConf EU 2019

How?

Over time, different approaches for improving http_parser's code base were tried. However, all of them failed due to resulting significant performance degradation.

This project is a port of http_parser to TypeScript. llparse is used to generate the output C source file, which could be compiled and linked with the embedder's program (like Node.js).

Performance

So far llhttp outperforms http_parser:

	input size	bandwidth	reqs/sec	time
llhttp	8192.00 mb	1777.24 mb/s	3583799.39 req/sec	4.61 s
http_parser	8192.00 mb	694.66 mb/s	1406180.33 req/sec	11.79 s

llhttp is faster by approximately **156%**.

Maintenance

llhttp project has about 1400 lines of TypeScript code describing the parser itself and around 450 lines of C code and headers providing the helper methods. The whole http_parser is implemented in approximately 2500 lines of C, and 436 lines of headers.

All optimizations and multi-character matching in llhttp are generated automatically, and thus doesn't add any extra maintenance cost. On the contrary, most of http_parser's code is hand-optimized and unrolled. Instead describing "how" it should parse the HTTP requests/responses, a maintainer should implement the new features in http_parser cautiously, considering possible performance degradation and manually optimizing the new code.

Verification

The state machine graph is encoded explicitly in llhttp. The llparse automatically checks the graph for absence of loops and correct reporting of the input ranges (spans) like header names and values. In the future, additional checks could be performed to get even stricter verification of the llhttp.

Usage

```c
#include "llhttp.h"

llhttp_t parser;
llhttp_settings_t settings;

/* Initialize user callbacks and settings */
llhttp_settings_init(&settings);

/* Set user callback */
settings.on_message_complete = handle_on_message_complete;

/* Initialize the parser in HTTP_BOTH mode, meaning that it will select between
 * HTTP_REQUEST and HTTP_RESPONSE parsing automatically while reading the first
 * input.
 */
llhttp_init(&parser, HTTP_BOTH, &settings);

/* Parse request! */
const char* request = "GET / HTTP/1.1\r\n\r\n";
int request_len = strlen(request);

enum llhttp_errno err = llhttp_execute(&parser, request, request_len);
if (err == HPE_OK) {
  /* Successfully parsed! */
} else {
  fprintf(stderr, "Parse error: %s %s\n", llhttp_errno_name(err),
          parser.reason);
}
```

For more information on API usage, please refer to src/native/api.h.

Bindings to other languages

- Python: pallas/pyllhttp
- Ruby: metabahn/llhttp

Using with CMake

If you want to use this library in a CMake project you can use the snippet below.

```
FetchContent_Declare(llhttp
  URL "https://github.com/nodejs/llhttp/releases/download/v6.0.4/llhttp-release-v6.0.4.tar.gz")  # Using version 6.0.4

FetchContent_MakeAvailable(llhttp)
```

```
target_link_libraries(${EXAMPLE_PROJECT_NAME} ${PROJECT_LIBRARIES} llhttp
${PROJECT_NAME})
```

LICENSE