1. http.

2. File Structure.

3. Main Program.

```
⟨ include files 5 ⟩
⟨ declarations of functions 7 ⟩
⟨ type declarations 8 ⟩
⟨ local functions 27 ⟩
int main(int argc, char *const *argv)
{
  struct MHD_Daemon *d;
  assert(MHD_is_feature_supported(MHD_FEATURE_MESSAGES));
  if (argc ≠ 2) {
    printf("%s␣PORT\n", argv[0]);
    return 1;
  }
  unsigned int flags = MHD_USE_THREAD_PER_CONNECTION;
  flags |= MHD_USE_INTERNAL_POLLING_THREAD;
  flags |= MHD_USE_ERROR_LOG;
  d = MHD_start_daemon(flags,
      atoi(argv[1]),
      ⟨ accept policy callback option 9 ⟩
      ⟨ http request callback option 10 ⟩
      ⟨ http options 11 ⟩
      ⟨ logging options 13 ⟩
      MHD_OPTION_END);
  if (d ≡ Λ) return 1;
  (void) getc(stdin);
  MHD_stop_daemon(d);
  return 0;
}
```

4. library

```
⟨ dummy.c 4 ⟩ ≡
  ⟨ include files 5 ⟩
  ⟨ declarations of functions 7 ⟩
  ⟨ library helper functions 26 ⟩
  ⟨ library functions 14 ⟩
```

5. ⟨ include files 5 ⟩ ≡

```
#include "platform.h"
#include <microhttpd.h>
#include <assert.h>
#include <stdbool.h>
```

This code is used in sections 3 and 4.

6. ⟨ initialize request local data 6 ⟩ ≡

```
if (&aptr ≠ *ptr) {    /* do never respond on first call */
  *ptr = &aptr;
  return MHD_YES;
}
```

7. declarations.

⟨ declarations of functions 7 ⟩ ≡

```
enum MHD_Result cb_request(void *cls, struct MHD_Connection *connection, const char *url, const
      char *method, const char *version, const char *upload_data, size_t *upload_data_size, void
      **ptr);
void logger(void *cls, const char *fm, va_list ap);
```

This code is used in sections 3 and 4.

8.

⟨ type declarations 8 ⟩ ≡

```
typedef struct _Request *Request;
```

This code is used in section 3.

9.

⟨ accept policy callback option 9 ⟩ ≡

```
Λ, Λ ,
```

This code is used in section 3.

10.

⟨ http request callback option 10 ⟩ ≡

```
&cb_request, Λ ,
```

This code is used in section 3.

11.

⟨ http options 11 ⟩ ≡

```
MHD_OPTION_CONNECTION_TIMEOUT, 256 ,
```

This code is used in section 3.

12. Define HTTPS related options. The key and a certificate needs to be set.

⟨ https specific options 12 ⟩ ≡

```
MHD_OPTION_HTTPS_MEM_KEY, key_pem, MHD_OPTION_HTTPS_MEM_CERT, cert_pem ,
```

13. ⟨ logging options 13 ⟩ ≡

```
MHD_OPTION_EXTERNAL_LOGGER, logger, &argv ,
```

This code is used in section 3.

14. ⟨ library functions 14 ⟩ ≡

```
void logger(void *cls, const char *fm, va_list ap)
{
  fprintf(stderr, "!!!!!␣");
  vfprintf(stderr, fm, ap);
  fprintf(stderr, "\n");
}
```

See also sections 18, 19, 21, 24, and 25.

This code is used in section 4.

15. processing.

16.

```
⟨try open file 16⟩ ≡
  FILE *file = fopen(&url[1], "rb");
  struct stat buf;
  if (Λ ≠ file) {
    int fd = fileno(file);
    if (−1 ≡ fd) {
      fclose(file);
      file = Λ;
    }
    else if ((0 ≠ fstat(fd, &buf)) ∨ (¬S_ISREG(buf.st_mode))) {
        /* not a regular file, refuse to serve */
      fclose(file);
      file = Λ;
    }
  }
```

This code is used in section 25.

17. respond with data in file by using callbacks for data and for cleanup.

```
⟨respond page from file content 17⟩ ≡
  status_code = MHD_HTTP_OK;
  response = MHD_create_response_from_callback(buf.st_size, 32 * 1024,    /* 32k size */
  &file_reader, file, &file_free_callback);
```

This code is used in section 25.

18. file callback

```
⟨library functions 14⟩ +≡
  static ssize_t file_reader(void *cls, uint64_t pos, char *buf, size_t max)
  {
    FILE *file = cls;
    (void) fseek(file, pos, SEEK_SET);
    return fread(buf, 1, max, file);
  }
```

19. file cleanup callback

```
⟨library functions 14⟩ +≡
  static void file_free_callback(void *cls)
  {
    fclose((FILE *) cls);
  }
```

20.

```
⟨respond static page 20⟩ ≡
  response = MHD_create_response_from_buffer(strlen(page), (void *) page, MHD_RESPMEM_PERSISTENT);
```

This code is used in section 25.

21. ⟨library functions 14⟩ +≡

```
enum MHD_Result print_key_value(void *cls, enum MHD_ValueKind kind, const char *key, const
        char *value)
{
  fprintf(stderr, "***␣%d:%s:%s\n", kind, key, value);
  return MHD_YES;
}
```

22. ⟨check for allowed method 22⟩ ≡

```
if ((0 ≠ strcmp(method, MHD_HTTP_METHOD_GET)) ∧ (0 ≠ strcmp(method, MHD_HTTP_METHOD_HEAD)))
  return MHD_NO;    /* unexpected method */
```

23. ⟨log request info 23⟩ ≡

```
fprintf(stderr, "ECHO␣url:%s\n␣method:%s\n", url, method);
fprintf(stderr, "␣␣␣upload␣data␣size:␣%d\n", *upload_data_size);
MHD_get_connection_values(connection,
    MHD_HEADER_KIND | MHD_COOKIE_KIND | MHD_POSTDATA_KIND | MHD_FOOTER_KIND, print_key_value, Λ);
```

This code is used in section 25.

24. ⟨library functions 14⟩ +≡

```
enum MHD_Result post_iterator(void *cls, enum MHD_ValueKind kind, const char *key, const
        char *filename, const char *content_type, const char *transfer_encoding, const char
        *data, uint64_t off, size_t size)
{
  struct Request *request = cls;
  fprintf(stderr, "###␣%s\n", key);
  return MHD_YES;
}
```

25. ⟨ library functions 14 ⟩ +≡

```
enum MHD_Result cb_request(void *cls, struct MHD_Connection *connection, const char *url, const
        char *method, const char *version, const char *upload_data, size_t *upload_data_size, void
        **ptr)
{
  static char *page = "{\"data\":1}";
  static int aptr;
  struct MHD_Response *response = Λ;
  int ret;
  unsigned int status_code = MHD_HTTP_NOT_IMPLEMENTED;
  ⟨ log request info 23 ⟩
  if ((0 ≡ strcmp(method, MHD_HTTP_METHOD_POST))) {
    fprintf(stderr, "Upload␣data␣size:␣%d\n", *upload_data_size);
    if (*upload_data_size ≡ 0) return MHD_YES;
    else {
      fprintf(stderr, "CONTENT:␣");
      for (int i = 0; i < *upload_data_size; i++) {
        fprintf(stderr, "%02x␣", upload_data[i]);
      }
      for (int i = 0; i < *upload_data_size; i++) {
        fprintf(stderr, "%c", upload_data[i]);
      }
      fprintf(stderr, "␣%p\n", response);
      const char *xpage = "XXX";
      if (false) response = MHD_create_response_from_buffer(*upload_data_size, (void *)
            upload_data, MHD_RESPMEM_MUST_COPY);
      else response = MHD_create_response_from_buffer(strlen(xpage), (void *) xpage,
            MHD_RESPMEM_MUST_COPY);
      MHD_add_response_header(response, MHD_HTTP_HEADER_CONTENT_ENCODING, "application/json");
      if (false) *upload_data_size = 0;
      status_code = MHD_HTTP_OK;
      ret = MHD_queue_response(connection, status_code, response);
      fprintf(stderr, "x␣queued␣response␣%d␣->␣%d\n", status_code, ret);
      MHD_destroy_response(response);
      return MHD_YES;
    }
  }
  if (response ≡ Λ) {
    ⟨ try open file 16 ⟩
    if (file) {
      ⟨ respond page from file content 17 ⟩
    }
  }
  if (response ≡ Λ) {
    ⟨ respond static page 20 ⟩
  }
  fprintf(stderr, "response␣%p\n", response);
  ret = MHD_queue_response(connection, status_code, response);
  fprintf(stderr, "queued␣response␣%d␣->␣%d\n", status_code, ret);
  MHD_destroy_response(response);
  return ret;
}
```

26. ⟨library helper functions 26⟩ ≡ /* empty */

This code is used in section 4.

27. ⟨local functions 27⟩ ≡ /* empty */

This code is used in section 3.

28. INDEX.

HTTP