# MyHTTP Roadmap & Checklist (Step-by-Step)

October 16, 2025

## Overview

A practical, incremental path to build the multithreaded HTTP/1.1 file server. Each phase ends with a clear "Definition of Done" and small test commands you can run immediately.

## 1 HTTP Methods & Semantics (MVP)

This server handles **GET**, **POST**, **PUT**, **PATCH**, and **DELETE**. Chunked requests are not supported in the MVP; a valid `Content-Length` is required for methods with bodies. Unless noted, behavior follows HTTP/1.1 with a pragmatic file-server interpretation.

### Common Rules

- **Host required**. Missing `Host` → 400.
- **Docroot jail**: Percent-decode path, join to docroot, `realpath()`, and reject traversal/symlink escapes (400/403).
- **Bodies**: For POST/PUT/PATCH, require `Content-Length`. If missing/invalid → 411/400.
- **Limits**: Enforce `MAX_BODY`; if exceeded → 413.
- **Expect: 100-continue**: If present, send `HTTP/1.1 100 Continue` before reading body.
- **Keep-Alive**: Persistent by default; honor `Connection: close`. Optional cap of 100 requests/connection.

### GET (safe, idempotent)

- Serve regular files with `200 OK`.
- Directories:
  - Missing trailing "/" → `301 Moved Permanently` to add "/".
  - Serve `index.html` if present; else generate simple HTML listing.
- Errors: outside docroot → 403/400; not found → 404.

### POST (create new resource; non-idempotent)

- If target resolves to an *existing directory* (or ends with "/"), create a new file (e.g., `upload-<ts>.bin`) and write the body → `201 Created` and `Location`.
- If target resolves to a *non-existent file path*, create it and write the body → `201 Created`.
- If target resolves to an *existing file*, return `409 Conflict` (use PUT to replace).

### PUT (create-or-replace; idempotent)

- Target must be a file path under docroot. Create if absent, otherwise *replace* entire contents.
- Status: `201 Created` if new; `200 OK` if replaced.
- If target is a directory → 409 Conflict.

### PATCH (partial update; MVP = replace-if-exists)

- MVP behavior: treat as "replace only if target exists". If exists, overwrite with the body → `200 OK`; if not, `409 Conflict`.
- Rationale: simple baseline while reserving future content-type specific partial semantics (e.g., JSON Merge Patch).

### DELETE (remove resource; idempotent)

- If target is a regular file under docroot, unlink it: `200 OK` (or `204 No Content`).
- Directories: reject with `403 Forbidden` (or `409 Conflict`) in MVP.
- Not found → `404 Not Found`.

### Status Codes (summary)

- **2xx**: 200 OK, 201 Created, 204 No Content.
- **3xx**: 301 Moved Permanently (dir redirect).
- **4xx**: 400 Bad Request, 403 Forbidden, 404 Not Found, 405 Method Not Allowed, 409 Conflict, 411 Length Required, 413 Payload Too Large.
- **5xx**: 500 Internal Server Error.

## 2 Phase 0 — Workspace Bootstrapping (30–45 min)

### 0.1 Repo layout

```
MyHTTP/
  Makefile
  src/
    main.c
    http_parse.c  http_parse.h
    fs.c          fs.h
    log.c         log.h
```

**0.2 Makefile (minimal)**: compile all `.c` in `src/` with `-std=c11 -Wall -Wextra -O2`.

**0.3 Main flags**: parse `-p`, `-d`. Default: port 8080, root ".".

**0.4 Open listen socket**: `socket, setsockopt(SO_REUSEADDR), bind, listen`.

**0.5 Definition of Done**: binary runs and prints "listening on 0.0.0.0:8080".

## 3 Phase 1 — Single-Connection Echo (15–30 min)

**1.1 Accept one client**: `accept4(..., SOCK_CLOEXEC)`.

**1.2 Blocking read/write loop**: read bytes, immediately write back (temporary).

**1.3 Definition of Done**: `nc 127.0.0.1 8080` echoes back text.

# 4 Phase 2 — Minimal HTTP Parser + Routing (GET/POST/PUT/-PATCH/DELETE) (1–2 h)

**2.1 Parser skeleton** (`http_parse.c`): parse request line (METHOD, TARGET, VERSION), then headers until CRLFCRLF.

**2.2 Limits**: max start-line 4KiB, headers 16KiB; reject on overflow (400).

**2.3 Routing (MVP semantics)**:

- **GET**: Map path under docroot; serve file with `200 OK`. If directory:
  - If missing trailing "/", return `301 Moved Permanently` to "/".
  - If `index.html` exists, serve it; else generate an HTML listing.
- **POST**: If target is an existing directory (or ends with "/"), create a new file (e.g., `upload-<ts>.bin`) and write the body → `201 Created` + `Location`. If target is a new file path, create and write → `201 Created`. If target is an existing file, `409 Conflict`.
- **PUT**: Create-or-replace the target file. New file → `201 Created`; existing file → `200 OK`. If target is a directory, `409 Conflict`.
- **PATCH**: Replace-only-if-exists (MVP). If file exists, overwrite with body → `200 OK`; else `409 Conflict`.
- **DELETE**: If the target is a regular file under docroot, unlink it → `200 OK` (or `204`). If not found, `404`. Directories rejected (`403`/`409`) in MVP.

**2.4 Bodies**: For POST/PUT/PATCH, require `Content-Length`; enforce `MAX_BODY`; respect `Expect: 100-continue`.

**2.5 Version checks**: if VERSION != HTTP/1.1 → 400.

**2.6 Definition of Done**:

- `curl -v http://127.0.0.1:8080/` serves files or listings (GET).
- `curl -v -X POST --data 'hi' http://127.0.0.1:8080/uploads/` creates a file (201).
- `curl -v -X PUT --data-binary @main.c http://127.0.0.1:8080/x.c` returns 201 or 200.
- `curl -v -X PATCH --data 'new' http://127.0.0.1:8080/x.c` returns 200 if `x.c` exists; 409 otherwise.
- `curl -v -X DELETE http://127.0.0.1:8080/x.c` returns 200/204 or 404.

# 5 Phase 3 — Map Target to Files (1–2 h)

**3.1 Path normalization** (`fs.c`): percent-decode safe chars, reject "..", build absolute path under ROOT, `realpath()` both ROOT and target; ensure target has ROOT prefix.

**3.2 Directories**: if directory and missing trailing slash → 301 redirect to "/.../".

**3.3 Index**: if directory and has `index.html`, serve it; else generate simple HTML listing.

**3.4 MIME**: basic table by extension, default `application/octet-stream`.

**3.5 Definition of Done**: static files are served; bad paths 404; dir redirect works.

# 6 Phase 4 — Keep-Alive + Connection Loop (45–60 min)

**4.1 Connection loop**: after finishing a response, attempt to parse the next request from the same socket.

**4.2 Timeouts**: `SO_RCVTIMEO` or `poll()` with 5s idle limit.

**4.3 Connection header**: default persistent; close on `Connection: close` or parse error.

**4.4 Cap**: optional max 100 requests/connection.

**4.5 Definition of Done**: `curl -v --keepalive-time 2 http://127.0.0.1:8080/` reuses TCP.

# 7 Phase 5 — Thread Pool & Work Queue (2–3 h)

**5.1 Introduce workq**

```
struct job { int client_fd; struct sockaddr_storage peer; };

struct workq {
  struct job *ring; size_t cap, head, tail, count;
  pthread_mutex_t mtx;
  pthread_cond_t not_empty, not_full;
};
```

**5.2 APIs**: `workq_init(cap)`, `enqueue(job)`, `dequeue()`, `workq_destroy()`.

**5.3 Acceptor thread**: `accept4()` then `enqueue()` (blocks when full; natural backpressure).

**5.4 Workers (N)**: `dequeue()` and run the full connection lifecycle (keep-alive loop).

**5.5 Per-thread buffers**: allocate request buffer & path buffer once per thread to avoid malloc churn.

**5.6 Definition of Done**: with `-w N`, concurrent requests are served in parallel.

# 8 Phase 6 — Logging, Errors, Hardening (1–2 h)

**6.1 Logging**: one line per request (ISO timestamp, peer IP, method, target, status, bytes, ms, user-agent).

**6.2 Error pages**: small HTML bodies for 400/404/405/413/500.

**6.3 Resource limits**: header size cap; idle timeout; sanitize directory listings (HTML-escape).

**6.4 Definition of Done**: malformed requests yield 400 with a body; logs look consistent under load.

## 9 Phase 7 — Functional Tests

- `curl -v /does-not-exist` (404), nested dirs, large files.
- Long URLs; encoded spaces; unicode.
- Symlink inside root; symlink escape attempt (must fail).

## 10 Phase 8 — Load & Stability Tests

- **Concurrency**: `for i in {1..200}; do curl -s http://127.0.0.1:8080/ & done; wait`
- **Backpressure**: small queue capacity; ensure acceptor blocks when full.
- **FD limits**: `ulimit -n 256`; observe graceful handling near limits.

## 11 Phase 9 — Polish (optional)

- Directory listing UX; size/mtime; inline CSS.
- `Last-Modified` / `If-Modified-Since`.
- Byte-range (206) single-range support.

## Mini Implementation Order (Files & Functions)

**1.** `main.c`: parse flags, open socket, single accept/handle.

**2.** `http_parse.c`: `myhttp_parse_request()`, `myhttp_find_header()`, limits.

**3.** `fs.c`: `safe_join_and_realpath()`, `is_dir()`, `serve_file()`, `render_listing()`.

**4.** `log.c`: `log_request()`, `ts_iso8601()`.

**5.** Thread pool: `workq.c`, `acceptor.c`, `worker.c`.

### Worker Loop Skeleton

```
for (;;) {
  ssize_t n = recv(client, buf + used, sizeof(buf)-used, 0);
  if (n <= 0) break;
  used += n;
  int parsed = myhttp_parse_request(buf, used, &req);
  if (parsed < 0) { send_400(client); break; }
  if (parsed == 0) continue; // need more bytes
  handle_request(client, &req); // may read body for POST/PUT/PATCH
  compact_or_reset_buffer(buf, &used, parsed); // keepalive: keep unread bytes
}
```