HTTP Server Code Review - Answers

Candidate Name

April 6, 2025

1 Architecture and Design Answers

1. **Threading Model:**
   - *Listener thread*: Accepts new connections and distributes them to worker threads using round-robin. Runs in a loop while `running_` is true.
   - *Worker threads*: Each has its own epoll instance to handle I/O events for assigned connections. Process both read (EPOLLIN) and write (EPOLLOUT) events.
2. **epoll Advantages:**
   - `select()/poll()` scale O(n) with connections, while epoll scales O(1)
   - epoll uses edge-triggered mode which avoids repeated notifications
   - Better performance with large numbers of connections (10k+)
   - Kernel maintains ready list instead of scanning all fds
3. **Round-Robin Distribution:**
   - Pros: Simple, no synchronization needed between workers
   - Cons: Can lead to imbalance if connections have uneven workloads
   - Alternatives: Least connections, weighted round-robin, or work stealing queues
4. **EventData Purpose:**

- Carries buffer state (cursor/length for partial reads/writes)
- Preserves context between epoll events (request/response cycle)
- Contains file descriptor and buffer for I/O operations
- Lifetime managed between EPOLLIN → EPOLLOUT transitions

2 Implementation Details Answers

5. `HandleEpollEvent` Lifecycle:
   - EPOLLIN: Reads request → Parses → Generates response → Switches to EPOLLOUT
   - EPOLLOUT: Writes response → Returns to EPOLLIN or closes connection
   - Memory Management: Allocates new EventData for each state transition to prevent use-after-free
6. **Level-Triggered Changes:**
   - Remove EPOLLET flag from event registration
   - Must handle repeated notifications for same data
   - Simpler but less efficient (more syscalls)
7. **Error Handling:**
   - Close Conditions: EOF (recv=0), errors (EPOLLHUP/EPOLLERR), send failures
   - EPOLLHUP=hangup (remote closed), EPOLLERR=error (local problem)
   - Both indicate unrecoverable connection states
8. `control_epoll_event` Race Conditions:
   - Potential FD reuse between delete/add
   - Current solution: Serialized through single-threaded event processing
   - Better: Could use thread-safe FD tracking

3 Performance and Optimization Answers

9. **Exponential Backoff:**
   - Used when accept() or epoll_wait() return no events
   - Prevents CPU spinning during idle periods
   - Improvement: Adaptive sleeping based on load metrics
10. **Copy Elimination:**
    - Current copies:
      (a) recv() → request buffer
      (b) response string → send buffer
    - Optimization: Use scatter/gather I/O (writev) or response buffers directly
11. **Thread Pool Limitations:**
    - Fixed size can't adapt to load spikes
    - Dynamic scaling: Monitor queue depth, add/remove workers
    - Challenge: Epoll FDs aren't thread-safe to share

4 Advanced Questions Answers

12. **HTTPS Support:**
    - Changes Needed:
      - SSL_accept/SSL_read/SSL_write wrappers
      - TLS handshake state machine
      - Certificate management
    - Performance Impact: 10x slower due to crypto ops
13. **Routing Enhancements:**
    - Path Parameters: Use regex matching with capture groups
    - Middleware: Chain of responsibility pattern with handler stack

14. **Timeout Handling:**
    - Idle: Timerfd + epoll with connection timestamp checks
    - Slow requests: Per-operation deadlines with SO_RCVTIMEO/SO_SNDTIMEO

Code Improvements

- **Memory:** Use memory pools for EventData allocations
- **Logging:** Add connection lifecycle tracing
- **Stats:** Track request rates/latencies per worker
- **HTTP/2:** Frame-based protocol support