Distributed Systems Server Demo © — Project Specification

1. Objectives

Build a 3-tier toy RPC system in C:

- Tier 1: Client
- Tier 2: Server (multiple implementations; focus of demo)
- Tier 3: Database (toy math service)

The goal is to demonstrate and compare:

1. Single-threaded blocking server
2. Thread-per-connection server
3. Single-threaded `select()` state machine
4. Event-library-driven server (e.g., libevent)

All versions:

- Use TCP
- Use a fresh TCP connection per request (client→server and server→db)
- Use identical wire protocol
- Share common RPC/message code

The client and database remain constant across experiments. Only the server implementation changes.

---

2. Repository Structure

Top-level:

- `Makefile`
- `common/`
- `client/`
- `db/`
- `server_sync/`
- `server_threaded/`
- `server_select/`
- `server_event/`

2.1 common/

Contains:

- Wire encoding/decoding
- Message definitions
- Framing logic
- Socket utilities
- Shared configuration constants

Builds into a static library (e.g., `libcommon.a`).

2.2 One Binary per Directory

Each subdirectory builds exactly one executable:

- `client/client`
- `db/db`
- `server_sync/server`
- `server_threaded/server`
- `server_select/server`
- `server_event/server`

All link against `libcommon.a`.

---

3. Build Requirements

- Compiler: `clang`
- Language standard: C11
- Strict warnings enabled and treated as errors
- No GNU-only extensions unless justified
- Optional sanitizer build via make variable

Default target builds all binaries.

---

4. Transport and Framing

All communication uses TCP stream sockets.

No assumption is permitted that:

- A single `recv()` returns a full message.
- A single `send()` writes all bytes.

All messages use length-prefixed framing:

1. `int32 total_len_be`
2. `int32 msg_type_be`
3. fields...

`total_len_be` is the number of bytes after itself.

All integers:

- 32-bit signed
- Network byte order

Strings:

- `int32 len_be`
- Followed by raw bytes (not NUL-terminated)

A maximum message size must be enforced (configurable constant).

---

5. Message Model

5.1 Common Types

- `i32` — signed 32-bit integer
- `msg_type_t` — enum of message kinds
- `op_t` — arithmetic operation enum (ADD, SUB, MUL, DIV)

5.2 Client → Server Request

Fields after `msg_type`:

1. `i32 request_id`
2. `i32 op`
3. `i32 a`
4. `i32 b`
5. `string key` (may be empty)

`request_id` is echoed in the response.

5.3 Server → Client Response

Fields:

1. `i32 request_id`
2. `string error` (empty means success)
3. `i32 result` (valid only if no error)

5.4 Server → DB Request

Fields:

1. `i32 request_id`
2. `i32 op`
3. `i32 a`
4. `i32 b`

5.5 DB → Server Response

Fields:

1. `i32 request_id`
2. `string error`
3. `i32 result`

Division by zero must produce a non-empty error string.

---

6. Common Library Responsibilities

6.1 Wire Layer

Responsibilities:

- Encode/decode 32-bit integers
- Encode/decode strings
- Maintain offset within buffers
- Bounds checking

6.2 Message Layer

Responsibilities:

- Pack typed structs into framed buffers
- Unpack buffers into typed structs
- Validate lengths and field counts

6.3 Framing Layer

Responsibilities:

- Incremental read state machine
- Maintain:
  - read buffer
  - expected frame length
  - current offset
- Detect complete frame

Conceptual connection state includes:

- File descriptor
- Read buffer and offsets
- Write buffer and offsets
- Parsed message object (when available)
- Logical phase/state

6.4 Socket Utilities

Responsibilities:

- Create listening socket
- Accept connections
- Connect to remote host
- Set non-blocking mode

---

7. Client Specification

7.1 Behavior

For each request:

1. Open TCP connection to server
2. Serialize request
3. Optionally send in two parts with delay
4. Receive response
5. Print timing metrics
6. Close connection

7.2 Options

- `--n N` number of requests
- `--fork N` spawn N processes
- `--slow-send` split request into chunks with random sleep
- `--annotate` print decoded wire format

7.3 Timing Output

Per request, print:

- Time spent sending (including artificial delay)
- Time waiting for response

---

8. Database Server Specification

8.1 Behavior

Loop:

1. Accept connection
2. Read exactly one request
3. Optionally sleep (random delay)
4. Compute result
5. Optionally split response write with delay
6. Close connection

8.2 Options

- `--delay-ms MIN MAX`
- `--split-send`
- `--annotate`

The DB server may remain single-threaded for simplicity.

---

9. Server Variants (Tier 2)

All server versions implement identical external semantics.

Common logical flow per client request:

1. Accept client connection
2. Read framed client request
3. Open connection to DB
4. Send DB request
5. Read DB response
6. Construct client response
7. Send response
8. Close both connections

Optional mode: two DB calls per request.

9.1 Version 1: Single-Threaded Blocking

Model:

- One accept loop
- Blocking `accept`, `recv`, `send`, `connect`

Properties:

- Fully serialized handling
- Idle while waiting on DB or slow client
- No shared-state concurrency issues

Demonstrates head-of-line blocking.

9.2 Version 2: Thread-per-Connection

Model:

- Main thread accepts
- Each accepted socket handled by new thread

Shared server state (if any) must be protected by mutex.

Demonstrates:

- Concurrency
- Synchronization costs
- Thread overhead

9.3 Version 3: select()-Driven State Machine

Model:

- Single-threaded
- All sockets non-blocking
- One `select()` loop

Each connection maintains explicit state:

States include:

- Reading client frame
- Connecting to DB
- Writing DB request
- Reading DB response
- Writing client response
- Closing

No mutexes required.

Demonstrates:

- Readiness-based multiplexing
- Explicit state machines
- Complexity of manual event management

9.4 Version 4: Event Library (libevent/libev)

Model:

- Event base
- Read/write callbacks
- Timer callbacks

Connection state stored in per-connection context struct.

Semantics identical to select version.

Demonstrates:

- Cleaner event abstraction
- Reduced boilerplate
- Similar architectural model

---

10. Shared State (Optional Demonstration)

To justify mutexes in threaded version:

Optional server-side cache:

- Map from string key → last result
- Accessed by all threads

Threaded server must guard map with mutex.

select/event versions require no locking.

---

11. Configuration Constants

Defined in `common/config.h`:

- Default ports (client-server, server-db)
- Max message size
- Default backlog
- Default timeouts

---

12. Pedagogical Demonstrations

1. Partial reads/writes break naive assumptions.
2. Blocking servers waste time waiting on I/O.
3. Thread-per-connection removes serialization but adds synchronization.
4. select() avoids threads but requires explicit state machines.
5. Event libraries abstract readiness but preserve same core model.
6. Network-bound workloads expose scalability limits.

---

13. Non-Goals

- Production-grade RPC
- Persistent connections
- TLS
- Binary compatibility guarantees
- High-performance memory allocators

The system exists purely to expose concurrency and I/O model tradeoffs in C.