

A) EPOLLIN – Incoming events (reading)

1. Handling Incoming Connections:

- The server listens for new client requests and monitors the socket using EPOLLIN to detect when data is ready to be read.

2. Accepting New Clients:

- When a connection request arrives, the server accepts it and creates a new socket to communicate with the client.

3. Reading Client Data:

- The server reads a specific amount of data from the client in chunks to ensure efficient processing and avoids reading too much at once.

4. Storing Data in Buffers:

- Any data received from a client is stored in the corresponding client's buffer to be processed later or sent back as a response.

B) Processing Requests

1. Iterate Through All Clients:

- Loop through each connected client to check the status of their requests.

2. Check for Complete Request Headers:

- Verify if the full HTTP header has been received in the client's buffer.

3. Parse the Header:

- If the header is complete, parse it to extract the request details.

4. Assign the Request to a Server:

- Determine which server instance should handle the request based on the parsed information.

5. Route the Request:

- Perform routing within the assigned server to find the appropriate handler for the request.

6. Handle the Request Body:

- Check if the request includes a body:
 - If so, determine whether more body data needs to be received.
 - For chunked transfer encoding, process each chunk as it arrives; otherwise, handle the body as-is.

7. Process the Request:

- Once the body is fully received (or if there's no body), proceed to process the request and generate a response.

A) EPOLLOUT – Outgoing events (writing)

1. Check Writable Sockets:

- Loop through the sockets flagged as ready for writing by EPOLLOUT.

2. Confirm Client Readiness:

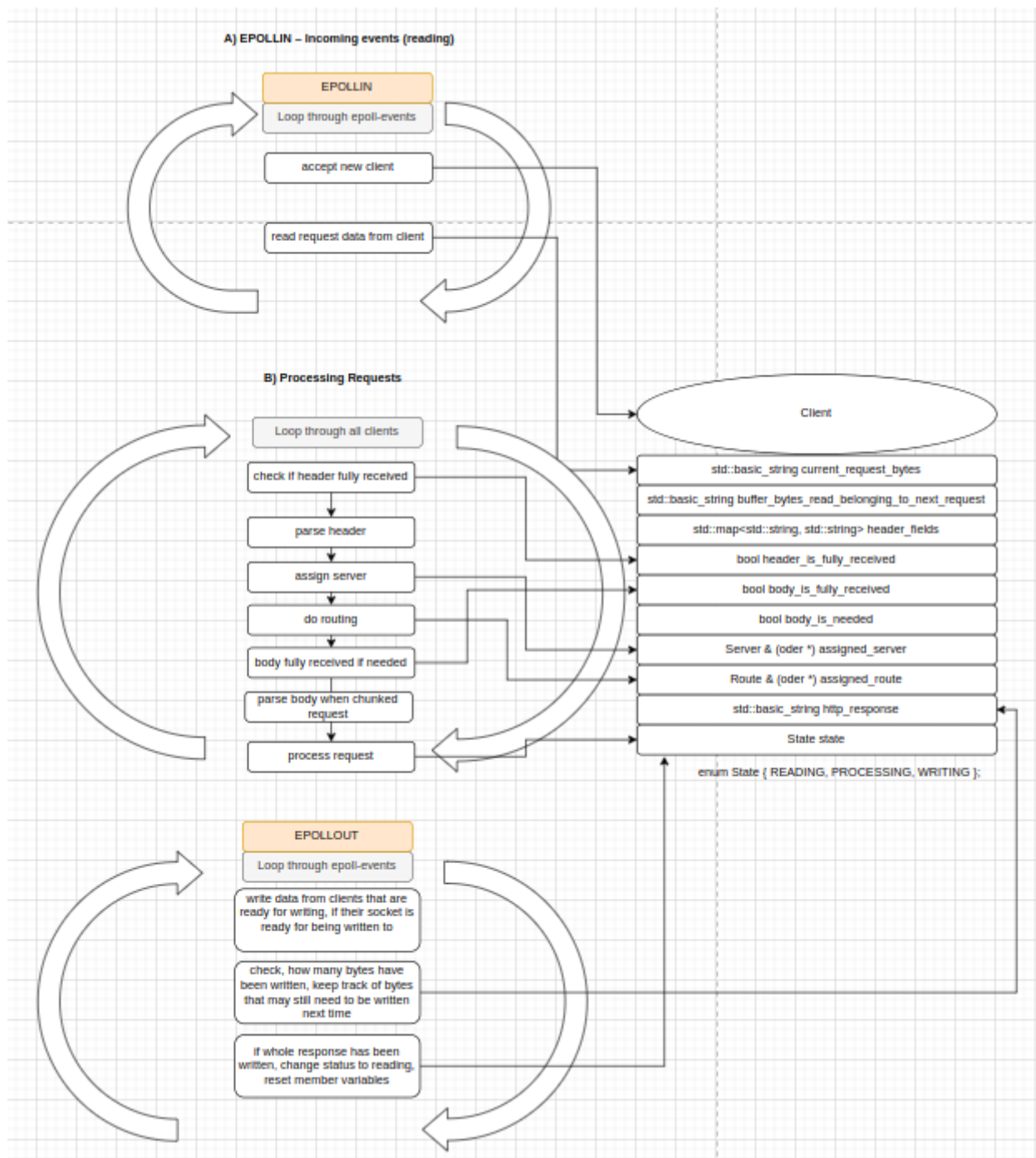
- Verify if the corresponding client has finished processing the request and is prepared to send a response.

3. Write Data to the Socket:

- Send the available response bytes from the client's buffer to the socket.

4. Track Written Bytes:

- Keep track of how many bytes were successfully written and remove those bytes from the response buffer, so the remaining data can be sent during the next writable cycle.




How Does Chunked Request Work?

- Similar to chunked responses, the request body is sent in chunks.
- Each chunk is preceded by its size in hexadecimal, followed by the chunk data, and ends with `\r\n`.
- The final chunk has a size of `0` and signals the end of the request body.

Example of a Chunked Request

makefile

 Copy code

```
POST /upload HTTP/1.1
Host: example.com
Transfer-Encoding: chunked
Content-Type: application/octet-stream

5\r\n
Hello\r\n
6\r\n
 World\r\n
0\r\n
\r\n
```

In this example:

- The client sends "Hello" as the first chunk (5 bytes) and " World" as the second chunk (6 bytes).
- The final chunk (`0`) signals the end of the data.