Objective:

Enhance the SIP server to handle standard SIP call flows, focusing on managing call setup and teardown; states from the initial INVITE to the final 200 OK response to the BYE request, including interim states.

Background:

In SIP communication, establishing a call involves a sequence of messages and state transitions, starting with an INVITE request. Intermediate responses, such as 180 Ringing, indicate the call's progress and are crucial for a standard-compliant SIP implementation.

Requirements:

State Machine Design for Call Setup:

Design a detailed state machine to handle the SIP call setup process, encompassing:

- Reception of the INVITE request.
- Transmission and handling of interim requests/responses (e.g., 100 Trying, 180 Ringing).
- Completion of the call setup with a 200 OK response. And then finally successful teardown.
- The state machine should clearly define state transitions, actions, and events for each step in the call setup and teardown process.

Implementation of the State Machine:

- Implement the state machine in the SIP server, ensuring it can manage and transition through call setup states accurately.
- First step would be to identify these states. You can implement `**TWO**` of your *choice* out of the total identified states in detail.
- Leave `Todos` for the remaining states. Make sure your code is extendable.

Documentation:

- Briefly document the state machine design.

Assumptions:

you can extend the following list of assumptions

- UEs do not require registration.
- No Authentication required.
- Only successful cases require implementation for call-setup and teardown.
- No media needs to be handled.

Where to Start

- The `main.c::handle_new_message()`, where you will update the routing logic to pass on the message to the thread.

- The `process_sip_messages()` where you will add your processing logic.

- You are free to add new functions/structs as required per your design.