Question 4: Multi-Client Chat Room Using Named Pipes(FIFOs)

This is a client-server chat room implementation in C using named pipes (FIFOs) for inter-process communication. The system allows multiple clients to join chat rooms and exchange messages.

System Requirements and Features

Core Requirements (Persian)

1. **اعلام ورود و خروج کاربران به اتاق**
    - Server broadcasts join/leave notifications
    - Shows online member count
    - Implemented via 'J' and 'L' message types
2. **رساندن پیام هر کاربر به بقیه کاربران**
    - Messages broadcast to all other room members
    - Implemented in `send_message_all()` function
    - Excludes sender from broadcast
3. **دریافت پیام ها به صورت زنده و بدون تاخیر در client**
    - Real-time message reception using threads
    - Non-blocking FIFO operations
    - Immediate console updates
4. **پشتیبانی از وجود کاربرانی با نام یکسان**
    - Unique internal IDs using PID
    - Display names can be identical
    - Internal ID format: `username_pid`
5. **امکان دریافت ورودی از کنسول همزمان با دریافت و پردازش اطلاعات**
    - Concurrent input/output handling
    - Separate thread for message reception
    - Console input remains responsive
6. **آزادسازی منابع تخصیص یافته**
    - Proper cleanup of memory and FIFOs
    - Signal handlers for graceful termination
    - Resource cleanup on client disconnect
7. **ادامه فعالیت اتاق در صورت خروج همه افراد**
    - Room persists after all clients leave
    - New clients can join existing rooms

- Server maintains room state

Message Structure

```
typedef struct {
    char type;              // Message type (J:Join, M:Message, L:Leave
S:Server)
    char display_name[32];  // User's display name
    char internal_id[64];   // Internal user ID
    char content[MAX_MSG];  // Message content
    char client_fifo[128];  // Client's FIFO path
    char room_name[32];     // Chat room name
} Message;
```

Communication Protocol

Message Types

1. Join Message (J)
    - Sent when a client connects
    - Triggers welcome message and member count update
    - Broadcasts join notification to other clients
2. Chat Message (M)
    - Regular chat messages
    - Broadcast to all clients except sender
    - Includes sender identification
3. Leave Message (L)
    - Sent during client disconnection
    - Triggers resource cleanup
    - Notifies other clients
4. Server Message (S)
    - Used for server shutdown notification
    - Forces clients to disconnect

Protocol Flow

1. Client Connection Process

```
Client → Server: Join Message (J)
Server → Client: Welcome Message
Server → Others: Join Notification
```

2. Message Exchange

```
Client → Server: Chat Message (M)
Server → Others: Message Broadcast
```

3. Client Disconnection

```
Client → Server: Leave Message (L)
Server → Others: Leave Notification
Server: Resource Cleanup
```

4. Server Shutdown

```
Server → All Clients: Server Message (S)
Server: Cleanup and Exit
Clients: Cleanup and Exit
```

Implementation Details

Server Implementation

1. **Client Management**

```c
typedef struct Client {
    char display_name[32];
    char internal_id[64];
    char fifo_path[128];
    int fd;
    struct Client *next;
} Client;
```

2. Message Broadcasting
    - Efficient message routing
    - Exclude sender from broadcasts
    - Handle client disconnections during broadcast
3. **Resource Management**
    - Proper FIFO cleanup in `cleanup()`
    - Memory deallocation for client structures
    - File descriptor management

Client Implementation

1. **Message Reception**
    - Separate thread (`receive_messages`)
    - Non-blocking FIFO reading
    - Console update handling
2. **Input Handling**
    - Concurrent input processing
    - Message formatting and sending
    - Known limitation: Console overwrite during typing

Known Limitations

- Console overwrite during message reception (noted in requirements)
- No message history persistence
- Basic console interface
- No message encryption

Error Handling

1. **Server-side**
    - Unexpected client disconnections
    - FIFO creation failures
    - Client connection tracking
2. **Client-side**
    - Server disconnection handling
    - FIFO creation/opening errors
    - Signal handling for clean exit

Resource Management

1. **Memory**
    - Dynamic allocation for client structures
    - Proper deallocation in cleanup
2. **FIFOs**
    - Creation and cleanup
    - Proper file descriptor management
    - Signal handler cleanup
3. **Process Management**
    - Thread cleanup
    - Signal handling
    - Resource deallocation

Core Functions Analysis

Server-Side Functions

create_user

```
void create_user(const char *display_name, const char *internal_id, const
char *fifo_path)
```

Purpose: Creates a new client node and adds it to the linked list of active clients. Implementation Details:

- Allocates memory for new client structure
- Copies user identification data
- Opens client's FIFO in non-blocking write mode
- Adds client to front of linked list
- Handles FIFO opening failures gracefully

remove_user

```
void remove_user(const char *internal_id)
```

Purpose: Removes a client from the active clients list and cleans up resources. Key Operations:

- Traverses client linked list
- Closes client's file descriptor
- Frees allocated memory
- Maintains list integrity during removal
- Handles both head and middle node removal

send_message_all

```
void send_message_all(const Message *msg, const char *sender_id, int
is_welcome)
```

Purpose: Broadcasts messages to all relevant clients. Implementation:

- Handles welcome messages separately (sent only to new user)
- Excludes sender from regular broadcasts
- Detects disconnected clients during send
- Removes clients on pipe errors
- Uses non-blocking writes to prevent hanging

get_online_members

```
int get_online_members()
```

Purpose: Counts current active clients. Implementation:

- Traverses client linked list
- Returns total count of connected clients
- Used for welcome messages and statistics

cleanup

```
void cleanup()
```

Purpose: Performs system shutdown and resource cleanup. Operations:

- Sends shutdown notification to all clients
- Closes all client file descriptors
- Removes all FIFO files
- Frees all allocated memory
- Closes server FIFOs

console_input_handler

```
void *console_input_handler(void *arg)
```

Purpose: Handles server console commands. Features:

- Runs in separate thread
- Processes 'close' command
- Enables graceful server shutdown
- Maintains server interactivity

Client-Side Functions

get_message

```
void *get_message(void *arg)
```

Purpose: Handles incoming message reception. Implementation:

- Runs in dedicated thread
- Uses poll for non-blocking reads
- Handles different message types
- Updates console display

- Detects server disconnection

print_message

```
void print_message(const char *name, const char *content, int is_you)
```

Purpose: Formats and displays chat messages. Features:

- Clears current line
- Differentiates between own and others' messages
- Maintains input prompt
- Handles console formatting

clear_line

```
void clear_line()
```

Purpose: Manages console output formatting. Implementation:

- Uses ANSI escape codes
- Clears current line
- Maintains clean console appearance
- Ensures proper message display

cleanup (Client)

```
void cleanup()
```

Purpose: Handles client-side resource cleanup. Operations:

- Sends leave message to server
- Closes file descriptors
- Removes client FIFO
- Terminates receive thread
- Ensures clean exit

Function Interactions

Message Flow

1. Client Input → Server

```
Main Loop → write() → server FIFO
```

2. Server Processing

```
poll() → read() → send_message_all() → client FIFOs
```

3. Client Reception

```
get_message() → read() → print_message() → console
```

Resource Management Flow

1. Client Connection

```
main() → create FIFOs → create_user() → send welcome
```

2. Client Disconnection

```
cleanup() → send leave → remove_user() → free resources
```

Error Handling Flow

1. Server Disconnection

```
get_message() → detect error → cleanup() → exit
```

2. Client Disconnection

```
send_message_all() → detect error → remove_user()
```

Critical Path Analysis

Client Startup

1. Command line argument validation
2. FIFO creation and opening
3. Thread creation
4. Join message transmission
5. Welcome message reception

Message Processing

1. Input reception
2. Message structure population
3. Server transmission
4. Broadcast processing
5. Console update

Shutdown Sequence

1. Signal reception/command input
2. Notification broadcast
3. Resource cleanup
4. Thread termination
5. Process exit

Thread Management

Server Threads

1. Main Thread:
    - Message processing
    - Client management
    - FIFO operations
2. Console Thread:
    - Command processing
    - Shutdown handling

Client Threads

1. Main Thread:
    - User input
    - Message sending
    - Resource management
2. Receive Thread:
    - Message reception
    - Console updates
    - Server status monitoring