Bài tập thực hành

Môn: TH Lập Trình Mạng

Chương 6: I/O UDP sockets

1. MỤC TIÊU

Mục tiêu của bài thực hành này là giúp sinh viên hiểu sâu hơn về lập trình socket UDP bằng cách triển khai một trò chơi chat nhiều người chơi có mã hóa. Sinh viên cần thực hiện các công việc sau:

- Sử dụng socket UDP (socket(), sendto(), recvfrom()).
- Xử lý nhiều client và các thông điệp.
- Triển khai xác minh server bằng cách sử dụng hàm memcmp().
- Sử dụng hàm connect() để tối ưu hóa giao tiếp UDP.
- Triển khai timeout cho các thông điệp.
- Thêm mã hóa đơn giản cho việc trao đổi dữ liệu giữa client và server.

2. YÊU CẦU

- Kiến thức cơ bản về lập trình C
- Kiến thức cơ bản về Mạng máy tính
- Máy tính cài đặt Hệ điều hành Linux (khuyến khích distro Ubuntu)
- Sử dụng các hàm trong bộ thư viện liên quan lập trình mạng

3. BÀI THỰC HÀNH

Tổng quan về ứng dụng:

Trong bài thực hành này, sinh viên sẽ triển khai một trò chơi chat nhiều người chơi, trong đó server xử lý nhiều client. Việc giao tiếp sẽ được mã hóa bằng cách sử dụng một thuật toán XOR đơn giản, và client sẽ xác minh danh tính của server bằng cách sử dụng hàm `memcmp()`.

- Server: Gửi các thông điệp đã mã hóa đến nhiều client, nhận phản hồi từ họ và xử lý chúng.
- Client: Kết nối đến server, nhận các thông điệp đã mã hóa, giải mã chúng, gửi lại phản hồi và xác minh rằng thông điệp đến từ server đúng bằng cách sử dụng `memcmp()`.

Phần 1: Thiết lập UDP Sockets với mã hóa

1.1: Chương trình Server (Xử lý nhiều client, các hàm cơ bản, mã hóa)

1. Tạo một server UDP lắng nghe kết nối từ nhiều client trên một địa chỉ IP và cổng cụ thể.
2. Server gửi các thông điệp đã mã hóa đến các client và nhận phản hồi của họ.
3. Mỗi client nhận được một thông điệp khác nhau, và server phản hồi với một thông báo xác nhận.

Cần giải quyết các vấn đề sau:

- Xử lý nhiều client với các thông điệp khác nhau.
- Mã hóa và giải mã các thông điệp bằng cách sử dụng thuật toán XOR đơn giản.
- Triển khai timeout cho thông điệp bằng cách sử dụng select().

Hướng dẫn:

- Sử dụng socket() để tạo socket UDP cho server.

```
31      // Create socket
32      server_socket = socket(AF_INET, SOCK_DGRAM, 0);
33      if (server_socket < 0) {
34          perror("Socket creation failed");
35          exit(1);
36      }
```

Nếu socket không tạo được sẽ in ra lỗi và thoát chương trình.

- Sử dụng bind() để liên kết socket với một địa chỉ IP và cổng.

```
44      // Bind socket
45      if (bind(server_socket, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
46          perror("Bind failed");
47          exit(1);
48      }
```

Nếu bind thất bại, server sẽ thoát ra.

- Sử dụng recvfrom() và sendto() để giao tiếp với các client.

```
79              // Use recvfrom to receive data and the address of the client
80              int recv_len = recvfrom(server_socket, buffer, MAX_BUFFER, 0,
81                                      (struct sockaddr*)&client_addr, &client_len);
```

*Nhận dữ liệu từ client, lưu vào **buffer** và địa chỉ client lưu vào **client_addr.***

- Sử dụng thuật toán XOR để mã hóa và giải mã các thông điệp.

```c
void xor_cipher(char *data, char key) {
    for (int i = 0; data[i] != '\0'; i++) {
        data[i] ^= key;
    }
}
```

*Hàm lặp từng ký tự trong chuỗi và XOR với khóa **key**. Khi thực hiện XOR lần thứ 2 với cùng 1 khóa sẽ trả về giá trị gốc nên sử dụng hàm này để giải mã và mã hóa các thông điệp.*

- Sử dụng select() để xử lý timeout cho các thông điệp.

```c
fd_set readfds;
struct timeval tv;

FD_ZERO(&readfds);
FD_SET(server_socket, &readfds);

// Set timeout to 5 seconds
tv.tv_sec = 5;
tv.tv_usec = 0;

int activity = select(server_socket + 1, &readfds, NULL, NULL, &tv);

if (activity < 0) {
    perror("Select error");
    continue;
}

if (activity == 0) {
    printf("Timeout occurred. No data after 5 seconds.\n");
    continue;
}
```

*Hàm **select()** sẽ chờ khi có dữ liệu hoặc timeout:*

+ *Nếu có dữ liệu, sẽ trả về số lượng file descriptors sẵn sàng để đọc.*
+ *Nếu không có dữ liệu, sẽ trả về 0 và hiển thị timeout.*

1.2: Chương trình Client (Xác minh địa chỉ server bằng memcmp(), mã hóa)

1. Client kết nối đến server, nhận thông điệp mã hóa, giải mã nó và gửi lại câu trả lời.

2. Client xác minh danh tính của server bằng cách sử dụng memcmp().

Cần giải quyết các vấn đề sau:

- Giải mã thông điệp bằng thuật toán XOR.
- Xác minh rằng thông điệp đến từ server mong đợi.
- Xử lý timeout cho thông điệp bằng cách sử dụng select().

Hướng dẫn:

- Sử dụng socket() để tạo socket UDP cho client.

```
24      // Create socket
25      client_socket = socket(AF_INET, SOCK_DGRAM, 0);
26      if (client_socket < 0) {
27          perror("Socket creation failed");
28          exit(1);
29      }
```

*Tương tự phía **server***

- Sử dụng connect() để đơn giản hóa giao tiếp.

```
37      // Connect to server
38      if (connect(client_socket, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
39          perror("Connection failed");
40          exit(1);
41      }
```

*Dùng để kết nối socket với địa chỉ và cổng của server. Do UDP không yêu cầu kết nối thực sự như TCP nên sử dụng **connect()** để đơn giản hóa giao tiếp giữa client và server (không cần chỉ định địa chỉ đích khi gửi, nhận dữ liệu).*

- Sử dụng recvfrom() và sendto() để giao tiếp.
***recvfrom()** và **sendto()** được sử dụng khi không kết nối socket trước (không dùng **connect()**), phù hợp khi server/client không duy trì một kết nối cố định. Vì đã sử dụng **connect()** nên ở đây sẽ dùng **recv()** và **send()** để giao tiếp:*

```
77                  // Use send() since we have already connected
78                  if (send(client_socket, buffer, strlen(buffer), 0) < 0) {
79                      perror("Send failed");
80                      continue;
81                  }
82                  printf("Sent encrypted message to server\n");
```

*Hàm **send()** gửi dữ liệu với **buffer** là dữ liệu đã mã hóa.*

```
88          // Use recv() since we have already connected
89          int bytes_received = recv(client_socket, buffer, MAX_BUFFER, 0);
90
91          if (bytes_received > 0) {
92              buffer[bytes_received] = '\0';
93
94              // Decrypt and display the message
95              xor_cipher(buffer, XOR_KEY);
96              printf("Server response: %s\n", buffer);
97          }
98          else if (bytes_received < 0) {
99              perror("Receive failed");
100         }
```

*Hàm **recv()** nhận dữ liệu qua socket và lưu vào **buffer**. Giải mã **buffer** với khóa **XOR_KEY**, nếu khớp thì xác minh được thông điệp đến từ server đúng.*

- Sử dụng memcmp() để so sánh địa chỉ server.

```
82          // Verify server with memcmp()
83          if (memcmp(&server_addr.sin_addr, &server_ip, sizeof(server_ip)) == 0) {
84              printf("Server address verified: %s\n", buffer);
85          } else {
86              printf("Warning: Unverified server address\n");
87          }
```

*Hàm **memcmp()** so sánh nội dung địa chỉ server mà client nhận được với địa chỉ server mong muốn để đảm bảo tính toàn vẹn của kết nối.*

- Triển khai mã hóa đơn giản bằng XOR.

```
13  void xor_cipher(char *data, char key) {
14      for (int i = 0; data[i] != '\0'; i++) {
15          data[i] ^= key;
16      }
17  }
```

*Tương tự phía **server***

```
73              // Encrypt the message before sending
74              strcpy(buffer, input);
75              xor_cipher(buffer, XOR_KEY);
```

*Mã hóa thông điệp trước khi gửi bằng khóa **XOR_KEY** được định nghĩa ở đầu.*

```
94              // Decrypt and display the message
95              xor_cipher(buffer, XOR_KEY);
96              printf("Server response: %s\n", buffer);
```

*Giải mã và hiển thị thông điệp nhận từ **server.***

Kết quả:

- *Server:*

```
phuc215118@sau:~/network/lab6$ ./server
Server started. Listening...
Timeout occurred. No data after 5 seconds.
Timeout occurred. No data after 5 seconds.
Timeout occurred. No data after 5 seconds.
Timeout occurred. No data after 5 seconds.
New client connected. ID: 1, IP: 127.0.0.1, Port: 50686
Received from client 1: client 1 send message
Timeout occurred. No data after 5 seconds.
New client connected. ID: 2, IP: 127.0.0.1, Port: 49147
Received from client 2: client 2 send message
Timeout occurred. No data after 5 seconds.
Timeout occurred. No data after 5 seconds.
^C
```

- *Client 1:*

```
phuc215118@sau:~/network/lab6$ ./client
Connected to server. Type your messages (type 'exit' to quit):
client 1 send message
Sent encrypted message to server
Server response: Client 1 sent: client 1 send message
exit
Closing...
```

- *Client 2:*

```
phuc215118@sau:~/network/lab6$ ./client
Connected to server. Type your messages (type 'exit' to quit):
client 2 send message
Sent encrypted message to server
Server response: Client 2 sent: client 2 send message
exit
Closing...
```

Phần 2: Cải tiến

2.1: Nhiệm vụ bổ sung

- Thêm tính năng xử lý nhiều client trong server.

```
86              // Find or add a new client
87              int client_index = -1;
88              for (int i = 0; i < client_count; i++) {
89                  if (client_addr.sin_addr.s_addr == clients[i].addr.sin_addr.s_addr &&
90                      client_addr.sin_port == clients[i].addr.sin_port) {
91                      client_index = i;
92                      break;
93                  }
94              }
95
96              if (client_index == -1 && client_count < MAX_CLIENTS) {
97                  client_index = client_count;
98                  clients[client_index].addr = client_addr;
99                  clients[client_index].id = client_count + 1;
100                 client_count++;
101                 printf("New client connected. ID: %d, IP: %s, Port: %d\n",
102                        clients[client_index].id,
103                        inet_ntoa(client_addr.sin_addr),
104                        ntohs(client_addr.sin_port));
105             }
```

*Trong file **server**, tạo danh sách client. Mỗi khi nhận thông điệp từ client, server sẽ kiểm tra địa chỉ đã tồn tại trong danh sách chưa. Nếu chưa, sẽ thêm vào danh sách, bao gồm địa chỉ và ID của client. Điều này cho phép server theo dõi nhiều client khác nhau, phân biệt dựa trên địa chỉ IP và cổng của client.*

- Cải thiện cơ chế mã hóa để an toàn hơn.

```
15  void xor_cipher(char *data, size_t data_len, const char *key) {
16      size_t key_len = strlen(key);
17      for (size_t i = 0; i < data_len; i++) {
18          data[i] ^= key[i % key_len];
19      }
20  }
```

*Cung cấp 1 khóa chuỗi **XOR_KEY** để cải thiện cơ chế mã hóa nếu không có khóa. Xác định độ dài của khóa để thực hiện vòng lặp qua từng ký tự dữ liệu, XOR từng ký tự dữ liệu với dữ liệu tương ứng trong khóa.*

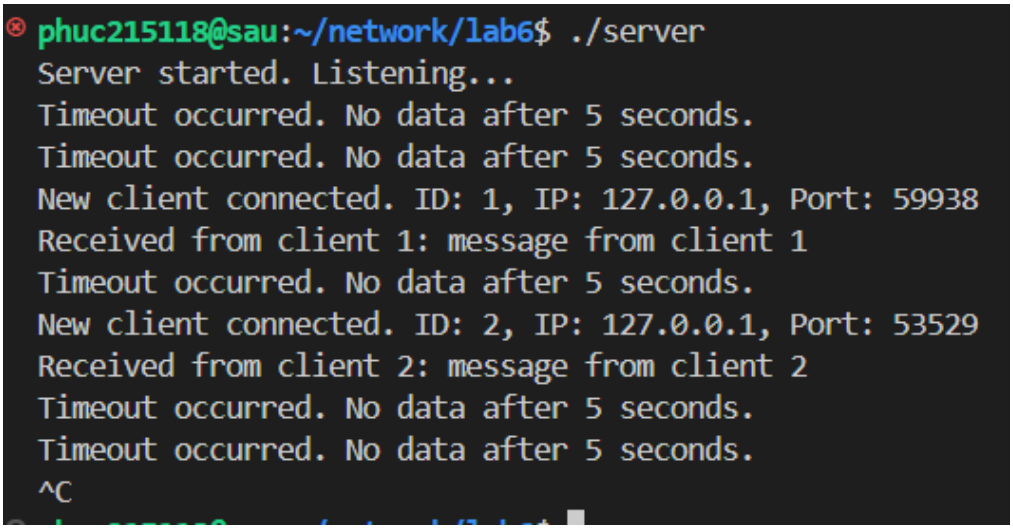
- Triển khai xử lý lỗi tốt hơn cho timeout của thông điệp và phản hồi từ server.

```c
int send_with_timeout(int sock, const char *msg, size_t len, int flags,
                      struct sockaddr *dest_addr, socklen_t addrlen, int timeout_sec) {
    fd_set writefds;
    struct timeval tv;
    int retries = 0;

    while (retries < MAX_RETRIES) {
        FD_ZERO(&writefds);
        FD_SET(sock, &writefds);

        tv.tv_sec = timeout_sec;
        tv.tv_usec = 0;

        int ready = select(sock + 1, NULL, &writefds, NULL, &tv);

        if (ready == -1) {
            if (errno == EINTR) continue;  // Interrupted, try again
            perror("select() error");
            return -1;
        } else if (ready == 0) {
            printf("Send timeout occurred. Retrying... (%d/%d)\n", retries + 1, MAX_RETRIES);
            retries++;
            continue;
        }

        if (FD_ISSET(sock, &writefds)) {
            return sendto(sock, msg, len, flags, dest_addr, addrlen);
        }
    }

    printf("Max retries reached. Message not sent.\n");
    return -1;
}
```

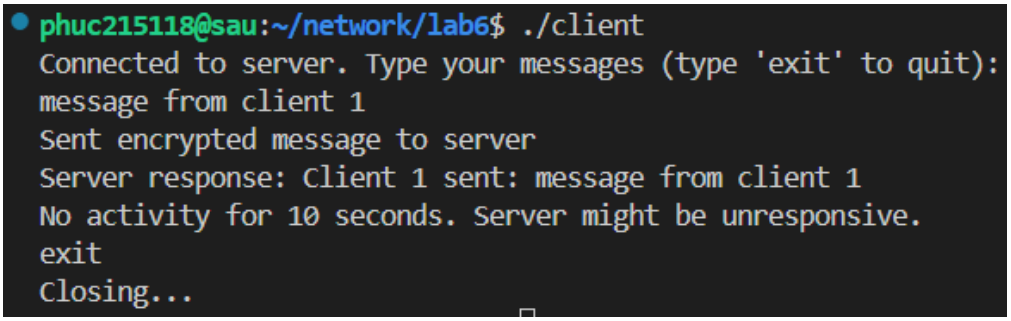
*Hàm **send_with_timeout** dùng để gửi thông điệp qua socket thử lại nhiều lần. Nếu không thể gửi trong khoảng thời gian quy định, hàm sẽ gửi lại nhiều lần (tối đa là **MAX_RETRIES**), đảm bảo việc gửi dữ liệu không bị treo khi có lỗi mạng hoặc server không phản hồi.*

Kết quả:

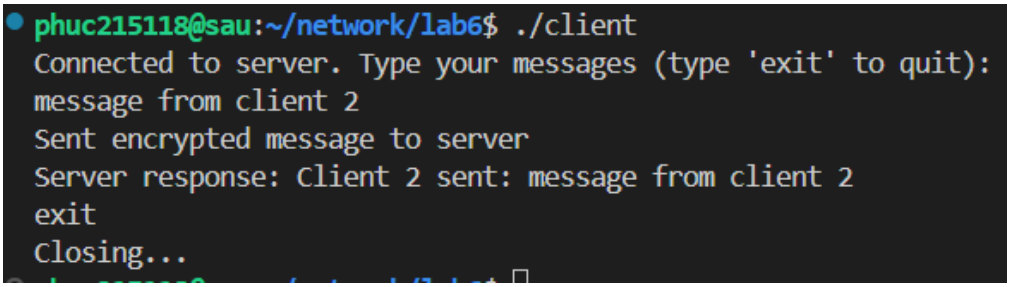
- *Server:*



- *Client 1:*



- *Client 2:*



Yêu cầu

Mỗi sinh viên phải nộp:

- Giải thích ngắn gọn về mã nguồn đã viết.
- Ảnh chụp màn hình hoặc nhật ký của việc giao tiếp giữa client và server.
- Thảo luận về cách triển khai mã hóa, xác minh server bằng memcmp(), và sử dụng connect().

Chú ý:

- Sinh viên phải nộp cả mã nguồn của server và client.
- Một báo cáo bao gồm giải thích và ảnh chụp màn hình của kết quả chương trình.