Báo cáo: Hệ thống Chat đơn giản sử dụng Socket trong Linux

1. Tổng quan hệ thống

1.1 Kiến trúc

- Mô hình Client-Server
- Sử dụng TCP Socket để giao tiếp
- Đa luồng xử lý nhiều client đồng thời bằng select()

1.2 Công nghệ sử dụng

- Ngôn ngữ: C
- IPC: TCP Socket
- File I/O để lưu trữ user
- Thư viện: sys/socket.h, sys/select.h,...

2. Chi tiết thiết kế

2.1 Cấu trúc dữ liệu

```
typedef struct
{
int type;
char username[MAX_USERNAME];
char password[MAX_PASSWORD];
char message[BUFFER_SIZE];
char target[MAX_USERNAME];
} Message;

typedef struct
{
char username[MAX_USERNAME];
char password[MAX_PASSWORD];
} User;
```

PROF

2.2 Các chức năng chính

Đăng ký (/register)

1. Client gửi username/password lên server
2. Server lưu vào file users.dat
3. Trả về thông báo thành công/thất bại

Đăng nhập (/login)

1. Client gửi thông tin đăng nhập

2. Server kiểm tra trong users.dat
3. Nếu hợp lệ, đánh dấu client đã xác thực

Chat chung (/all)

1. Client gửi tin nhắn broadcast
2. Server forward tới tất cả client đã xác thực

Chat riêng (/username)

1. Client chỉ định người nhận và nội dung
2. Server chỉ gửi tới client được chỉ định

Gửi file tin nhắn (/input)

1. Client đọc file theo dòng
2. Gửi từng dòng như tin nhắn riêng/chung
3. Có delay giữa các tin để tránh nghẽn

3. Chi tiết IPC Socket

3.1 Khởi tạo kết nối

```
// Server
server_fd = socket(AF_INET, SOCK_STREAM, 0);
bind(server_fd, ...);
listen(server_fd, 3);

// Client
sock_fd = socket(AF_INET, SOCK_STREAM, 0);
connect(sock_fd, ...);
```

PROF

3.2 Xử lý đa tiến trình

- Sử dụng select() để theo dõi nhiều socket
- Server xử lý các event:
  - Socket mới kết nối
  - Socket gửi dữ liệu
  - Socket ngắt kết nối

```
while (1)
{
  FD_ZERO(&read_fds);
  FD_SET(server_fd, &read_fds);
  max_sd = server_fd;
```

```
    for (int i = 0; i < client_count; i++)
    {
      int sd = clients[i].socket;
      FD_SET(sd, &read_fds);
      max_sd = (sd > max_sd) ? sd : max_sd;
    }

    select(max_sd + 1, &read_fds, NULL, NULL, NULL);

    if (FD_ISSET(server_fd, &read_fds))
    {
      int new_socket = accept(server_fd, NULL, NULL);
      clients[client_count].socket = new_socket;
      clients[client_count].authenticated = 0;
      client_count++;
      printf("New client connected, socket fd: %d\n", new_socket);
    }

    for (int i = 0; i < client_count; i++)
    {
      int sd = clients[i].socket;

      if (FD_ISSET(sd, &read_fds))
      {
        Message msg;
        int valread = read(sd, &msg, sizeof(Message));

        if (valread == 0)
        {
          printf("Client disconnected, socket fd: %d\n", sd);
          close(sd);
          for (int j = i; j < client_count - 1; j++)
          {
            clients[j] = clients[j + 1];
          }
          client_count--;
          continue;
        }

        switch (msg.type)
        {
        case MSG_REGISTER:
        {
          User new_user;
          strcpy(new_user.username, msg.username);
          strcpy(new_user.password, msg.password);
          save_user(&new_user);
          send(sd, "Registration successful", 22, 0);
          printf("User %s registered\n", msg.username);
          break;
        }
        case MSG_LOGIN:
        {
```

```
          if (check_user(msg.username, msg.password))
          {
            strcpy(clients[i].username, msg.username);
            clients[i].authenticated = 1;
            send(sd, "Login successful", 16, 0);
            printf("User %s logged in\n", msg.username);
          }
          else
          {
            send(sd, "Login failed", 11, 0);
            printf("Login failed for user %s\n", msg.username);
          }
          break;
        }
        case MSG_BROADCAST:
          if (clients[i].authenticated)
          {
            broadcast_message(clients[i].username, msg.message);
          }
          break;
        case MSG_PRIVATE:
          if (clients[i].authenticated)
          {
            private_message(clients[i].username, msg.target,
  msg.message);
          }
          break;
        }
      }
    }
  }
```

- Client xử lý:
  - Input từ bàn phím
  - Dữ liệu từ server

PROF

```
  while (1)
{
  FD_ZERO(&read_fds);
  FD_SET(STDIN_FILENO, &read_fds);
  FD_SET(sock_fd, &read_fds);

  select(sock_fd + 1, &read_fds, NULL, NULL, NULL);

  if (FD_ISSET(STDIN_FILENO, &read_fds))
  {
    fgets(buffer, BUFFER_SIZE, stdin);
    buffer[strcspn(buffer, "\n")] = 0;
    handle_command(buffer);
  }
```

```
  if (FD_ISSET(sock_fd, &read_fds))
  {
    int valread = read(sock_fd, buffer, BUFFER_SIZE);
    if (valread == 0)
    {
      printf("Server disconnected\n");
      return 0;
    }

    time_t now = time(NULL);
    struct tm *t = localtime(&now);

    char timestamp[32];
    strftime(timestamp, sizeof(timestamp), "[%H:%M:%S - %d/%m/%Y]", t);

    buffer[valread] = 0;
    printf("%s %s\n", timestamp, buffer);
  }
}
```

3.3 Truyền nhận dữ liệu

```
  // Gửi
send(socket, &msg, sizeof(Message), 0);

// Nhận
read(socket, &msg, sizeof(Message));
```