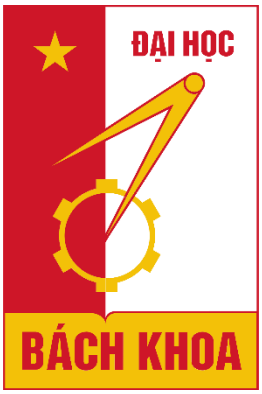
ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



BÁO CÁO BÀI TẬP LỚN
MÔN LẬP TRÌNH MẠNG

Đề tài: Xây dựng ứng dụng trực tuyến "Ai là triệu phú"

Nhóm sinh viên thực hiện **Trịnh Hữu An – 20225593**
Phạm Quốc Minh – 20225743

Giảng viên hướng dẫn **TS Đặng Tuấn Linh**

Hà Nội, ngày 18 tháng 01 năm 2025

Mục lục

Chương 1: Giới thiệu đề tài

1. Sơ lược về đề tài

Trong kỷ nguyên số hiện nay, công nghệ thông tin và truyền thông đã đóng vai trò quan trọng trong mọi lĩnh vực của cuộc sống, đặc biệt là trong lĩnh vực giáo dục và giải trí. Dự án "Ai Là Triệu Phú" là một ứng dụng lập trình mạng được phát triển nhằm tái hiện lại trò chơi truyền hình nổi tiếng "Ai Là Triệu Phú" dưới dạng một nền tảng trực tuyến, mang lại trải nghiệm tương tác thú vị cho người chơi.

Mục tiêu của dự án:

- **Xây dựng hệ thống client-server:** Tạo ra một hệ thống mạng ổn định, cho phép nhiều người chơi tham gia vào trò chơi cùng một lúc thông qua mạng internet.

- **Quản lý dữ liệu câu hỏi:** Thiết kế cơ sở dữ liệu chứa các câu hỏi đa dạng về nhiều lĩnh vực khác nhau, đảm bảo tính công bằng và thách thức cho người chơi.

- **Tính năng tương tác:** Cho phép người chơi trả lời câu hỏi, sử dụng quyền trợ giúp như gọi điện cho bạn bè, loại trừ hai phương án sai, và hỏi ý kiến khán giả một cách nhanh chóng và hiệu quả.

- **Bảo mật và bảo vệ dữ liệu:** Đảm bảo an toàn thông tin người dùng.

- **Ý nghĩa của dự án:** Dự án "Ai Là Triệu Phú" không chỉ là một trò chơi giải trí mà còn là công cụ học tập hiệu quả, giúp người chơi mở rộng kiến thức và kỹ năng tư duy logic. Bên cạnh đó, việc triển khai dự án này sẽ giúp các thành viên trong nhóm phát triển nâng cao kỹ năng lập trình mạng, quản lý cơ sở dữ liệu, thiết kế giao diện người dùng và xử lý các vấn đề bảo mật trong môi trường mạng.

Chương 2: Cơ sở lý thuyết

Dưới đây là một số cơ sở lý thuyết và nền tảng mà nhóm chúng em áp dụng để thực hiện ứng dụng của mình:

1. Kiến thức cơ bản bằng lập trình socket, truyền thông bằng TCP

- **Socket** là giao diện lập trình ứng dụng mạng được dùng để truyền và nhận dữ liệu trên internet. Giữa hai chương trình chạy trên mạng cần có một liên kết giao tiếp hai chiều, hay còn gọi là two-way communication để kết nối 2 process trò chuyện với nhau. Điểm cuối (endpoint) của liên kết này được gọi là socket.
    - Socket định danh ứng dụng mà dữ liệu sẽ được gửi tới thông qua sự ràng buộc với một cổng port để tiến hành kết nối giữa client và server.

    - Ưu điểm của socket: tương thích với hầu hết các hệ điều hành, từ Windows, Linux cho đến Mac OS X… Ngoài ra, socket còn có thể kết hợp được với rất nhiều ngôn ngữ lập trình như: C, C++, Java, Python…người dùng cũng có thể chạy cùng một lúc nhiều socket liên tục, giúp nâng cao hiệu suất làm việc, cũng như tiết kiệm thêm nhiều thời gian và công sức hơn.

- **Truyền thông bằng TCP**:
    - Tạo luồng dữ liệu hai chiều, đáng tin cậy, có trình tự và không trùng lặp, dữ liệu chỉ được gửi/nhận khi có đã có liên kết. Trong java sử dụng class socket để tạo và sử dụng đối tượng socket

    - Cần có liên kết 2 chiều trước khi server và client có thể trao đổi thông điệp với nhau.

    - Ban đầu, phía server tạo socket được ràng buộc với một cổng (port number) để chờ nhận yêu cầu từ phía client.

    - Tiếp đến phía client yêu cầu server bằng cách tạo một Socket TCP trên máy kèm với địa chỉ IP và port number của tiến trình tương ứng trên máy server. Khi client tạo Socket, client TCP tạo liên kết với server TCP và chờ chấp nhận kết nối từ server.

- TCP cung cấp dịch vụ truyền dòng tin cậy và có thứ tự giữa client và server, giữa máy chủ và máy nhận chỉ có 1 địa chỉ IP duy nhất. Thêm vào đó, mỗi thông điệp truyền đi đều có xác nhận trả về.

2. Kiến thức về lập trình đa luồng, thread, xử lý đồng bộ

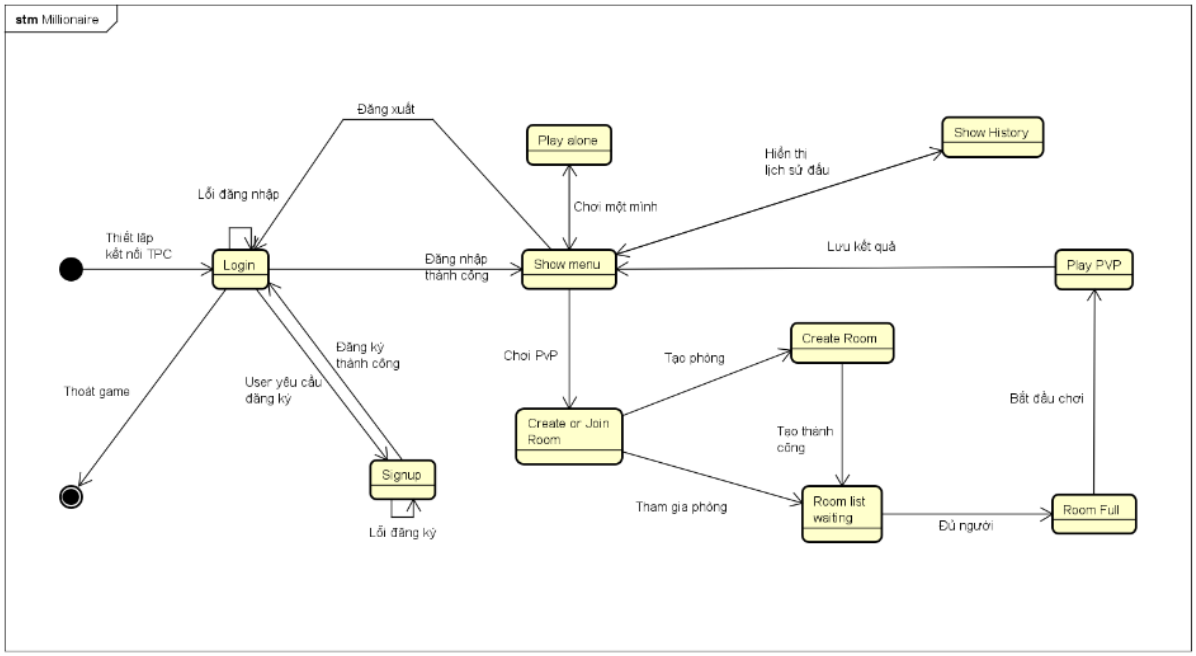
- **Luồng** (Thread) cơ bản là một tiến trình con (sub-process). Trong C, một luồng là một đơn vị xử lý nhỏ nhất của máy tính có thể thực hiện một công việc riêng biệt, tương tự như tiến trình nhưng nhẹ hơn và chia sẻ cùng không gian địa chỉ với các luồng khác trong cùng một tiến trình.

- **Đa luồng** (Multi-threading) là một tiến trình thực hiện nhiều luồng đồng thời. Trong C, việc sử dụng đa luồng giúp ứng dụng chạy nhanh và hiệu quả hơn bằng cách tận dụng tối đa tài nguyên của hệ thống, đặc biệt là trên các hệ thống đa lõi.

- **Đồng bộ hóa** (Synchronization) trong môi trường đa luồng là rất quan trọng để tránh các vấn đề như xung đột truy cập tài nguyên chung. Trong C, việc đồng bộ hóa thường được thực hiện thông qua các mutex và các cơ chế đồng bộ khác được cung cấp bởi thư viện POSIX threads (pthread).

Chương 3: Thiết kế giao thức

Ứng dụng "Ai Là Triệu Phú" sẽ được xây dựng theo mô hình **Multi Client-Server:**



Sơ đồ State Machine của ứng dụng

Dưới đây là chi tiết cụ thể về kiến trúc mạng được thực hiện cho từng server:

1. Server

Server thực hiện gửi và nhận thông điệp với các client (Multiclient) thông qua giao thức truyền thông TCP để đảm bảo thông tin được truyền đi một cách đầy đủ và chính xác nhất. Sẽ có 2 dạng thông điệp là ServerMessage (thông điệp từ server trả về cho một hoặc nhiều client) và ClientMessage (thông điệp từ một client gửi lên server). Mỗi dạng thông điệp này sẽ có các loại khác nhau (sử dụng enum để liệt kê và phân biệt) tương ứng với việc thực hiện các chức năng khác nhau. Dưới đây là bảng liệt kê các loại thông điệp:

```
enum msg_type
{
  DISCONNECT,
  LOGIN,
  LOGIN_SUCCESS,
  LOGGED_IN,
  WRONG_PASSWORD,
```

```
  ACCOUNT_NOT_EXIST,
  ACCOUNT_BLOCKED,
  SIGNUP,
  ACCOUNT_EXIST,
  SIGNUP_SUCCESS,
  CHANGE_PASSWORD,
  SAME_OLD_PASSWORD,
  CHANGE_PASSWORD_SUCCESS,
  PLAY_ALONE,
  STOP_GAME,
  QUESTION,
  CHOICE_ANSWER,
  CORRECT_ANSWER,
  WIN,
  LOSE,
  OVER_TIME,
  LOGOUT,
  FIFTY_FIFTY,
  CALL_PHONE,
  CHANGE_QUESTION,
  ASK_AUDIENCE,
  HISTORY,
  HISTORY_PVP,
  PLAY_PVP,
  FOUND_PLAYER,
  ENTERED_ROOM,
  WAIT_OTHER_PLAYER,
  NOT_FOUND_PLAYER,
  OTHER_PLAYER_IS_PLAYING,
  WIN_PVP,
  LOSE_PVP,
  VIEW_ONLINE_PLAYERS,
  DRAW
};

enum login_status
{
  AUTH,
  UN_AUTH
};
```

Cấu trúc bản tin: (header: kiểu thông điệp, loại dữ liệu, độ dài thông điệp và payload chứa dữ liệu)

```
typedef struct _message
{
  enum msg_type type;
```

```
  char data_type[25];
  int length;
  char value[BUFF_SIZE];
} Message;
```

Như vậy, mỗi khi có phản hồi hay yêu cầu từ client, server sẽ biết được client muốn gì thông qua kiểu thông điệp, từ đó xử lý yêu cầu và trả lại kết quả cho người dùng.

Chương 4: Phân chia công việc và cách thức triển khai

Tính năng	An	Minh	Điểm
Xử lý truyền dòng	X		1

Cơ chế vào ra socket trên server	**X**		**2**
Đăng kí và quản lí tài khoản	**X**		**2**
Đăng nhập và quản lí phiên đăng nhập, đổi mật khẩu	**X**		**2**
Chế độ chơi đơn	**X**		**2**
Thiết lập các quyền trợ giúp và logic	**X**		**3**
Thiết lập cơ chế, luật chơi, quản lí phiên trò chơi xác định thắng thua chế độ chơi đơn	**X**		**5**
Mã hóa mật khẩu người dùng	**X**		**1**
Lưu thông tin ván đấu	**X**		**2**
Tạo và tìm phòng cho PVP		**X**	**3**
Hiển thị danh sách người chơi online		**X**	**1**
Xử lý đồng bộ câu hỏi PVP		**X**	**3**
Lưu thông tin ván đấu PVP		**X**	**2**
Xử lý xác định thắng thua PVP		**X**	**2**
Đồng bộ thời gian thi đấu giữa hai người		**X**	**4**
Tổng Điểm	**19**	**15**	**34**

Trịnh Hữu An:

1. Xử lý truyền dòng

Các bước xử lý truyền dòng:

1. **Nhận dữ liệu từ client (dòng hoặc gói tin):** Dữ liệu có thể được nhận dưới dạng từng dòng, chuỗi ký tự. Sử dụng hàm như recv() để nhận dữ liệu từ client.
2. **Xử lý dữ liệu nhận được (dòng hoặc gói tin):** Sau khi nhận dữ liệu, ta cần xử lý dữ liệu đó, có thể là chuyển đổi, phân tích cú pháp hoặc thực hiện các tác vụ khác dựa trên nội dung dữ liệu.
3. **Gửi phản hồi cho client (dòng hoặc gói tin):** Sau khi xử lý, gửi phản hồi về cho client. Dữ liệu gửi có thể là một chuỗi thông báo, kết quả xử lý hoặc thông tin khác.

```
while ((recvBytes = recv(conn_fd, &msg, sizeof(msg), 0)) > 0)
{
  if (msg.type == DISCONNECT)
  {
    recvBytes = 0;
    break;
  }

  switch (cli->login_status)
  {
  case AUTH:
    switch (msg.type)
    {
  case CHANGE_PASSWORD:
    re = change_password(cli->login_account, msg.value);

    char response_message[256];

    if (re == SAME_OLD_PASSWORD) {
        snprintf(response_message, sizeof(response_message), "Mật khẩu mới trùng với mật khẩu cũ!");
        printf("[%d] Mật khẩu mới trùng với mật khẩu cũ!\n", conn_fd);
    }
    else if (re == CHANGE_PASSWORD_SUCCESS) {
        snprintf(response_message, sizeof(response_message), "Thay đổi mật khẩu thành công!");
        printf("[%d] Thay đổi mật khẩu thành công!\n", conn_fd);
    }
    else if (re == CHANGE_PASSWORD_SUCCESS) {
        snprintf(response_message, sizeof(response_message), "Thay đổi mật khẩu thành công!");
        printf("[%d] Thay đổi mật khẩu thành công!\n", conn_fd);
    }
    strncpy(msg.value, response_message, sizeof(msg.value) - 1);
    msg.type = re;
    send(conn_fd, &msg, sizeof(msg), 0);
    break;
```

2. Cơ chế vào ra socket trên server

Cơ chế vào ra (I/O) trên socket trong lập trình mạng TCP/IP sử dụng các hàm hệ thống để thực hiện việc giao tiếp giữa các máy tính qua mạng. Quá trình này có thể chia thành hai phần chính: **Gửi dữ liệu (Send)** và **Nhận dữ liệu (Recv)**. Sau đây là phân tích cơ chế vào ra này:

Cơ chế vào (Send)

Khi một client hoặc server muốn gửi dữ liệu qua mạng, nó sử dụng hàm như send() để truyền tải thông tin từ bộ nhớ của chương trình đến một kết nối socket. Các bước cơ bản là:

Tạo socket

Trước khi gửi dữ liệu, một socket phải được tạo và thiết lập kết nối. Đối với server, socket này phải được "liên kết" (bind) với một địa chỉ và cổng và sau đó "lắng nghe" kết nối đến từ client. Đối với client, socket sẽ được tạo và kết nối tới server.

```
if ((listen_fd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
{
  perror("\nError: ");
  exit(-1);
}


bzero(&server, sizeof(server));
server.sin_family = AF_INET;
server.sin_port = htons(PORT);
server.sin_addr.s_addr = htonl(INADDR_ANY);


if (setsockopt(listen_fd, SOL_SOCKET, SO_REUSEADDR, (char *)&opt, sizeof(opt)) < 0)
{
  perror("setsockopt() failed");
  close(listen_fd);
  exit(-1);
}

if (bind(listen_fd, (struct sockaddr *)&server, sizeof(server)) == -1)
{
  perror("\nError: ");
  close(listen_fd);
  exit(-1);
}
```

Gửi dữ liệu (send())

Hàm send() là một trong những hàm sử dụng để gửi dữ liệu qua socket. Cấu trúc của hàm này như sau:

*ssize_t send(int sockfd, const void *buf, size_t len, int flags);*

- sockfd: Socket đã được kết nối với một remote endpoint (ví dụ: kết nối tới server).
- buf: Con trỏ đến vùng nhớ chứa dữ liệu cần gửi.
- len: Số byte dữ liệu cần gửi.
- flags: Các tùy chọn điều khiển hành vi của send(), như MSG_NOSIGNAL để tránh gửi tín hiệu khi socket bị đóng.

Khi send() được gọi, nó sẽ gửi dữ liệu qua kết nối. Dữ liệu sẽ được chia thành các gói (packets) và gửi qua giao thức TCP, đảm bảo độ tin cậy (tái truyền khi mất gói).

Buffering (Đệm)

Trước khi dữ liệu thực sự được gửi qua mạng, nó có thể được lưu trong bộ đệm của kernel (hệ điều hành). Bộ đệm này giúp giảm thiểu số lần truy xuất mạng và cải thiện hiệu suất. Dữ liệu trong bộ đệm sẽ được gửi đi khi có đủ hoặc khi kết nối đã được đóng.

Cơ chế ra (Recv)

Khi một client hoặc server muốn nhận dữ liệu từ một socket đã kết nối, nó sẽ sử dụng hàm như recv()

Nhận dữ liệu (recv())

Hàm recv() được sử dụng để nhận dữ liệu từ socket, với cấu trúc như sau:

*ssize_t recv(int sockfd, void *buf, size_t len, int flags);*

- sockfd: Socket mà bạn muốn nhận dữ liệu.
- buf: Con trỏ đến vùng nhớ nơi dữ liệu sẽ được lưu trữ.
- len: Số byte tối đa có thể nhận được.
- flags: Các tùy chọn để điều khiển hành vi của recv().

Khi recv() được gọi, nó sẽ chờ đợi cho đến khi có dữ liệu đến trên socket. Nếu dữ liệu không có sẵn ngay lập tức, hàm này sẽ chặn (block) và đợi. Tuy nhiên, cũng có thể sử dụng các tùy chọn như MSG_DONTWAIT để thực hiện nhận dữ liệu không đồng bộ (non-blocking).

Blocking

- **Blocking I/O**: Mặc định, các hàm recv() và send() sẽ chặn (block) chương trình cho đến khi chúng hoàn thành. Điều này có nghĩa là nếu không có dữ liệu đến, hàm recv() sẽ chờ đợi cho đến khi có.

Timeouts

Trong trường hợp không có dữ liệu trong một khoảng thời gian nhất định, các chương trình có thể thiết lập **timeout** để tránh việc chương trình bị "treo" vĩnh viễn.

```c
void *thread_start(void *client_fd)
{
  pthread_detach(pthread_self());

  int recvBytes, re;
  Message msg;
  int conn_fd = *((int *)client_fd);
    free(client_fd);

    // Thiết lập timeout cho socket
    struct timeval timeout;
    timeout.tv_sec = TIME_OUT;
    timeout.tv_usec = 0;

    if (setsockopt(conn_fd, SOL_SOCKET, SO_RCVTIMEO, (char *)&timeout, sizeof(timeout)) < 0) {
        perror("Error setting socket timeout");
        close(conn_fd);
        pthread_exit(NULL);
    }

    Client *cli = head_client;
```

Quản lý lỗi

Khi hoạt động I/O gặp phải sự cố (ví dụ: mất kết nối, lỗi truyền tải), hệ điều hành sẽ trả về mã lỗi. Các lỗi phổ biến khi làm việc với socket bao gồm:

- **EAGAIN/EWOULDBLOCK**: Có thể xảy ra khi socket không có dữ liệu hoặc không thể gửi được dữ liệu ngay lập tức trong chế độ non-blocking.

```c
  // Kiểm tra lý do thoát khỏi vòng lặp
   if (recvBytes <= 0) {
        if (errno == EWOULDBLOCK || errno == EAGAIN) {
            printf("[%d]: Timeout: Không có phản hồi nào trong %d giây\n", conn_fd, TIME_OUT);
        } else if (recvBytes == 0) {
            printf("[%d]: Client ngắt kết nối\n", conn_fd);
        } else {
            perror("Recv error");
        }
        close(conn_fd);
        delete_client(conn_fd);
    }

  pthread_exit(NULL);
```

3. Đăng ký và quản lý tài khoản

Client truyền vào 2 tham số là tên tài khoản và mật khẩu rồi gửi đến server:

```
int signup(char username[], char password[]) {

    Message msg;
    msg.type = SIGNUP;
    strcpy(msg.data_type, "string");
    strcpy(msg.value, username);
    strcat(msg.value, " ");
    strcat(msg.value, password);
    msg.length = strlen(msg.value);
    if (send(sockfd, &msg, sizeof(Message), 0) < 0) {
        printf("Gửi dữ liệu không thành công");
    }

    recvBytes = recv(sockfd, &msg, sizeof(Message), 0);
    if (recvBytes == 0) {
        printf("Server đã ngắt kết nối\n");
        close(sockfd);
        exit(0);
    } else if (recvBytes < 0) {
        printf("Nhận dữ liệu không thành công");
    }

    return msg.type;
}
```

Sau khi nhận thông tin từ client, server xử lí yêu cầu bằng cách truy cập vào database, khóa mutex tránh tương tranh tài nguyên (2 người dùng cùng đăng kí 1 tên tài khoản 1 lúc). Sau đó, kiểm tra xem có bị trùng username không. Nếu trùng thông báo lỗi, ngược lại đăng kí thành công.

```
    pthread_mutex_lock(&mutex);

    sprintf(query, "SELECT * FROM account WHERE username = '%s'", username);
    execute_query(query);
    res = mysql_use_result(conn);
    if ((row = mysql_fetch_row(res)) == NULL)
    {
        mysql_free_result(res);
        sprintf(query, "INSERT INTO account(username, password, status) VALUES('%s', '%s', 1)", username, hashed_pas
        execute_query(query);
        res = mysql_use_result(conn);
        re = SIGNUP_SUCCESS;
    }
    else
        re = ACCOUNT_EXIST;

    mysql_free_result(res);

    pthread_mutex_unlock(&mutex);

    return re;
}
```

4. Đăng nhập và quản lí phiên đăng nhập, đổi mật khẩu

Tương tự, khi đăng nhập, người dùng cũng truyền vào 2 tham số là username và password rồi gửi cho server:

```
int login(char username[], char password[]) {
    Message msg;
    msg.type = LOGIN;
    strcpy(msg.data_type, "string");
    strcpy(msg.value, username);
    strcat(msg.value, " ");
    strcat(msg.value, password);
    msg.length = strlen(msg.value);
    if (send(sockfd, &msg, sizeof(Message), 0) < 0) {
        printf("Gửi dữ liệu không thành công");
    }

    recvBytes = recv(sockfd, &msg, sizeof(Message), 0);
    if (recvBytes == 0) {
        printf("Server đã ngắt kết nối\n");
        close(sockfd);
        exit(0);
    } else if (recvBytes < 0) {
        printf("Nhận dữ liệu không thành công");
    }

    return msg.type;
}
```

Sau đó, server tách chuỗi nhận được thành 2 tham số:

```
    strcpy(username, strtok(msg_data, " "));
    strcpy(password, strtok(NULL, " "));

    while (cli != NULL && cli->conn_fd != conn_fd)
        cli = cli->next;
```

Tiếp theo, truy cập vào cơ sở dữ liệu để kiểm tra tính chính xác, có các trường hợp xảy ra như: Tài khoản không tồn tại, sai mật khẩu, tài khoản bị khóa, tài khoản đang đăng nhập ở nơi khác, đăng nhập thành công

```
    sprintf(query, "SELECT * FROM account WHERE username = '%s'", username);
    execute_query(query);
    res = mysql_use_result(conn);

    if ((row = mysql_fetch_row(res)) == NULL) {
        re = ACCOUNT_NOT_EXIST;
    } else {
        if (strcmp(row[2], hashed_password) == 0) {
            if (strcmp(row[3], "1") == 0) {
                while (tmp != NULL) {
                    if (strcmp(tmp->login_account, username) == 0 && tmp->login_status == AUTH) {
                        re = LOGGED_IN;
                        break;
                    }
                    tmp = tmp->next;
                }
                if (re != LOGGED_IN) {
                    strcpy(cli->login_account, username);
                    cli->login_status = AUTH;
                    re = LOGIN_SUCCESS;
                }
            } else {
                re = ACCOUNT_BLOCKED;
            }
```

```
            }
        } else {
            re = WRONG_PASSWORD;
        }
    }

    mysql_free_result(res);
    return re;
}
```

Khi đổi mật khẩu, client sẽ gửi mật khẩu mới đến server:

```
int change_password(char password[]){
  Message msg;
  msg.type = CHANGE_PASSWORD;
  strcpy(msg.data_type, "string");
  strcat(msg.value, password);
  msg.length = strlen(msg.value);
  if (send(sockfd, &msg, sizeof(Message), 0) < 0)
  {
    printf("Gửi dữ liệu không thành công");
  }

  if (recv(sockfd, &msg, sizeof(Message), 0) < 0)
  {
    printf("Nhận dữ liệu không thành công");
  }

  return msg.type;
}
```

Sau đó server truy cập vào cơ sở dữ liệu và thực hiện logic nghiệp vụ:

```
    sprintf(query, "SELECT * FROM account WHERE username = '%s'", username);
    execute_query(query);
    res = mysql_use_result(conn);

    if ((row = mysql_fetch_row(res)) != NULL) {
        strncpy(current_hashed_password, row[2], SHA256_DIGEST_LENGTH * 2 + 1);
        if (strcmp(current_hashed_password, hashed_password) == 0) {
            re = SAME_OLD_PASSWORD;
        } else {
            mysql_free_result(res);
            sprintf(query, "UPDATE account SET password = '%s' WHERE username = '%s'", hashed_password, username);
            execute_query(query);
            res = mysql_use_result(conn);

            re = CHANGE_PASSWORD_SUCCESS;
        }
    } else {
        re = ACCOUNT_NOT_EXIST;
    }

    mysql_free_result(res);
    return re;
}
```

5. Chế độ chơi đơn

Khi người chơi chọn chơi đơn, server sẽ random lấy 15 câu hỏi bất kì tương ứng với 15 level khác nhau ở trong cơ sở dữ liệu:

```
int handle_play_alone(int conn_fd, char username[])
{
  Message msg;
  Question questions = get_questions();
  char str[2048];
```

Sau đó chờ phản hồi từ client, người chơi có thể có các lựa chọn sau:

```
  printf("\t\t'0': Xin dừng cuộc chơi\n");
  printf("\t\t'1 -> 4': Trả lời câu hỏi\n");
  printf("\t\t'5': Trợ giúp 50/50\n");
  printf("\t\t'6': Trợ giúp gọi điện thoại cho người thân\n");
  printf("\t\t'7': Trợ giúp đổi câu hỏi\n");
  printf("\t\t'8': Trợ giúp hỏi ý kiến khán giả\n");
```

Khi người chơi gửi đáp án hoặc yêu cầu, server sẽ căn cứ vào thông điệp để xử lí logic:

```
  switch (msg.type)
  {
  case STOP_GAME:
    handle_play_game(msg, conn_fd, &questions, level, id, username);
    return 0;
  case CHOICE_ANSWER:
    id = questions.id[level-1];
    re = handle_play_game(msg, conn_fd, &questions, level, id, username);
    if(re == 0) continue;
    return 0;
  case FIFTY_FIFTY:
  case CALL_PHONE:
  case CHANGE_QUESTION:
    id = questions.id[level-1];
    handle_play_game(msg, conn_fd, &questions, level, id, username);
    level--;
    goto initQuestion;
  case ASK_AUDIENCE:
    id = questions.id[level-1];
    handle_play_game(msg, conn_fd, &questions, level, id, username);
    level--;
  default:
    break;
  }
```

6. Thiết lập các quyền trợ giúp và logic

Quyền trợ giúp 50/50: Server sẽ lấy 1 đáp án đúng và random 1 đáp án sai trả về cho người dùng:

```
int fifty_fifty(Question q, int level, int answers[2]) {
    srand(time(0));
    int correct_answer = q.answer[level - 1];
    int incorrect_answer;
    do {
        incorrect_answer = rand() % 4 + 1;
    } while (incorrect_answer == correct_answer);
    answers[0] = correct_answer;
    answers[1] = incorrect_answer;
    return 1;
}
```

Quyền trợ giúp gọi điện thoại cho người thân: Server trả về phương án đúng cho người dùng:

```
int call_phone(Question q, int level){
  srand(time(0));
  return q.answer[level - 1];
}
```

Quyền trợ giúp đổi câu hỏi: Random 1 câu khác trong cơ sở dữ liệu có cùng mức level với câu hỏi hiện tại:

```c
int change_question(Question *q, int level, int id) {
  MYSQL_RES *res;
  MYSQL_ROW row;

  char query[1000];
  sprintf(query,
          "SELECT question, a, b, c, d, answer, reward, id , sum_a, sum_b, sum_c, sum_d "
          "FROM questions "
          "WHERE level = %d AND id != %d "
          "ORDER BY RAND() LIMIT 1",
          level, id);
  execute_query(query);
  res = mysql_store_result(conn);
  if((row = mysql_fetch_row(res)) != NULL){
    strcpy(q->question[level - 1], row[0]);
    strcpy(q->a[level - 1], row[1]);
    strcpy(q->b[level - 1], row[2]);
    strcpy(q->c[level - 1], row[3]);
    strcpy(q->d[level - 1], row[4]);
    q->answer[level - 1] = atoi(row[5]);
    q->reward[level - 1] = atoi(row[6]);
    q->id[level-1] = atoi(row[7]);
    q->sum_a[level-1] = atoi(row[8]);
    q->sum_b[level-1] = atoi(row[9]);
    q->sum_c[level-1] = atoi(row[10]);
    q->sum_d[level-1] = atoi(row[11]);
    mysql_free_result(res);
  }
```

Quyền trợ giúp hỏi ý kiến khán giả: Server sẽ thu thập thông tin các câu trả lời kể cả đúng sai từ những người dùng khác và tính toán %:

```c
int ask_audience(const Question *q, int level, int sum[4]) {
    sum[0] = q->sum_a[level - 1];
    sum[1] = q->sum_b[level - 1];
    sum[2] = q->sum_c[level - 1];
    sum[3] = q->sum_d[level - 1];
    return 1;
}
```

```c
    case ASK_AUDIENCE:
      printf("[%d]: Client yêu cầu trợ giúp hỏi ý kiến khán giả cho câu hỏi %d\n", conn_fd, level);
      int sum[4];
      ask_audience(questions, level, sum);
      msg.type = ASK_AUDIENCE;
      int sum_answer = sum[0] + sum[1] + sum[2] + sum[3];
      float sum_1 = (float)sum[0] / sum_answer * 100;
      float sum_2 = (float)sum[1] / sum_answer * 100;
      float sum_3 = (float)sum[2] / sum_answer * 100;
      float sum_4 = (float)sum[3] / sum_answer * 100;
      snprintf(msg.value, sizeof(msg.value),
          "Tỷ lệ chọn phương án 1 là: %.2f%%\n"
          "Tỷ lệ chọn phương án 2 là: %.2f%%\n"
          "Tỷ lệ chọn phương án 3 là: %.2f%%\n"
          "Tỷ lệ chọn phương án 4 là: %.2f%%\n", sum_1, sum_2, sum_3, sum_4);
      send(conn_fd, &msg, sizeof(msg), 0);
      break;
```

7. Thiết lập cơ chế, luật chơi, quản lí phiên trò chơi xác định thắng thua (chơi đơn)

Cơ chế luật chơi: Có ba mốc câu hỏi 5, 10, 15. Trả lời đúng sẽ được đi tiếp, trả lời sai sẽ nhận được số tiền thưởng ở mốc gần nhất đã trả lời được. Nếu dừng cuộc chơi thì sẽ nhận được số tiền thưởng câu hỏi trước đó. Sau khi nhận thông điệp từ người chơi, kiểm tra logic đáp án hoặc quyền lựa chọn và đưa ra thông báo (thắng/thua) cùng số tiền thưởng:

```
case CHOICE_ANSWER:
  answer = atoi(strtok(msg.value, "|"));
  if (answer == 0){
    msg.type = STOP_GAME;
    insert_history(username, level);
  if(level <= 1){
    sprintf(str, "Đáp án: %d\nSố tiền thưởng của bạn: 0", questions->answer[level - 1]);
    strcpy(msg.value, str);
    send(conn_fd, &msg, sizeof(msg), 0);
    printf("[%d]: Xin dừng cuộc chơi!\n", conn_fd);
    break;
  }
  else {
    sprintf(str, "Đáp án: %d\nSố tiền thưởng của bạn: %d", questions->answer[level - 1], questions->reward[lev
    strcpy(msg.value, str);
    send(conn_fd, &msg, sizeof(msg), 0);
    printf("[%d]: Xin dừng cuộc chơi!\n", conn_fd);
    break;
  }
  }
```

```
  else if (questions->answer[level - 1] == answer)
  {
    update_answer_sum(id, answer);
    sprintf(str, "Đáp án: %d\nSố tiền thưởng của bạn: %d", questions->answer[level - 1], questions->reward[
    strcpy(msg.value, str);
    if (level == 15)
    {
      msg.type = WIN;
      insert_history(username, level);
      send(conn_fd, &msg, sizeof(msg), 0);
      printf("[%d]: WIN!\n", conn_fd);
    }
    else{
      msg.type = CORRECT_ANSWER;
      send(conn_fd, &msg, sizeof(msg), 0);
      printf("[%d]: Trả lời đúng câu hỏi %d\n", conn_fd, level );
      return 0;
    }
  }
```

```
else
{
  update_answer_sum(id, answer);
  msg.type = LOSE;
  insert_history(username, level);
  if (level <= 5) {
  sprintf(str, "Đáp án: %d\nSố tiền thưởng của bạn: 0", questions->answer[level - 1]);
  strcpy(msg.value, str);
  send(conn_fd, &msg, sizeof(msg), 0);
  printf("[%d]: LOSE\n", conn_fd);
  break;
  } else if (level <= 10) {
  sprintf(str, "Đáp án: %d\nSố tiền thưởng của bạn: 2000", questions->answer[level - 1]);
  strcpy(msg.value, str);
  send(conn_fd, &msg, sizeof(msg), 0);
  printf("[%d]: LOSE\n", conn_fd);
  break;
  } else {
  sprintf(str, "Đáp án: %d\nSố tiền thưởng của bạn: 22000", questions->answer[level - 1]);
  strcpy(msg.value, str);
  send(conn_fd, &msg, sizeof(msg), 0);
  printf("[%d]: LOSE\n", conn_fd);
  break;
  }
}
```

8. Mã hóa mật khẩu người dùng

Sau khi nhận tham số là mật khẩu từ người dùng, có thể là đăng kí hoặc đăng nhập. Server sẽ mã hóa/ kiểm tra tính xác thực của mật khẩu. Hàm hash_user_password trong đoạn mã sử dụng thuật toán băm SHA-256 để mã hóa mật khẩu người dùng thành một chuỗi băm cố định. Đây là cách mà mật khẩu thường được xử lý để bảo vệ trong các ứng dụng bảo mật:

```
void hash_user_password(const char *password, char *hashed_password) {
    unsigned char hash[SHA256_DIGEST_LENGTH];
    SHA256((unsigned char *)password, strlen(password), hash);

    for (int i = 0; i < SHA256_DIGEST_LENGTH; i++) {
        sprintf(hashed_password + (i * 2), "%02x", hash[i]);
    }
    hashed_password[SHA256_DIGEST_LENGTH * 2] = '\0';
}
```

```
    // Mã hóa mật khẩu người dùng nhập vào
    hash_user_password(password, hashed_password);
```

9. Lưu thông tin ván đấu

Sau khi người chơi thắng /thua /bỏ cuộc, lưu số câu trả lời đúng và tiền thưởng vào cơ sở dữ liệu:

```
void insert_history(char username[], int level) {
    char query[500];
    pthread_mutex_lock(&mutex);
    snprintf(query, sizeof(query), "INSERT INTO history (username, correct_answers) VALUES ('%s', %d)", username, le
    if (mysql_query(conn, query)) {
        fprintf(stderr, "Lỗi khi chèn vào cơ sở dữ liệu: %s\n", mysql_error(conn));
    }
    pthread_mutex_unlock(&mutex);
}
```

```
CREATE TABLE `history` (
  `id` int NOT NULL AUTO_INCREMENT,
  `username` varchar(50) NOT NULL,
  `correct_answers` int NOT NULL,
  `play_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Phạm Quốc Minh

10. Tạo và tìm phòng PVP

1.1 Khởi tạo cấu trúc phòng:

```
typedef struct _room
{
  int room_id;
  int client_fd[2];              // set default all = 0
  int status;                    // 0: no player enter, 1: waiting, 2:
playing, 3: end
  int play_status[2];            // 0: playing, 1: out of room, 2: lose,
3: win, 4: draw
  int index_current_question;    // start from 1 to 15
  Question questions;
  int reward[2];                 // set default all = 0
  int is_answered[2];
  char player_name[2][BUFF_SIZE];
  struct _room *next;
} Room;
Room *head_room = NULL;
int current_id_room = 0;
```

Phòng sẽ được chia rõ theo **room_id** và mảng **play_status[2]** sẽ bao gồm người chơi đầu tiên tạo phòng là chủ phòng và người chơi thứ 2 là thành viên.

1.2 Hàm tạo phòng mới theo một danh sách liên kết đơn:

```
Room *add_room()
{
  Room *new = new_room();
  if (head_room == NULL)
    head_room = new; // if linked list is empty
  else
  {
    Room *tmp = head_room; // assign head to p
    while (tmp->next != NULL)
      tmp = tmp->next; // traverse the list until the last node
    tmp->next = new;   // Point the previous last node to the new node
created.
  }
  return new;
}
```

Trong đó hàm **new_room()** là hàm khởi tạo các giá trị mặc định của phòng PVP.

Tương tự với thêm phòng thì xóa phòng cũng tương tự:

```
void delete_room(int room_id)
{
  Room *tmp = head_room;
  Room *prev = NULL;
  while (tmp != NULL)
  {
    if (tmp->room_id == room_id)
    {
      if (prev == NULL)
        head_room = tmp->next;
      else
        prev->next = tmp->next;
      free(tmp);
      return;
    }
    prev = tmp;
    tmp = tmp->next;
  }
}

Room *find_room_is_blank_or_waiting()
{
  Room *tmp = head_room;
  while (tmp != NULL)
  {
    if (tmp->status == 0 || tmp->status == 1)
      return tmp;
    tmp = tmp->next;
  }
  return NULL;
}

int add_client_to_room(int conn_fd, Room *room)
{
  if (room->client_fd[0] == 0)
  {
    room->client_fd[0] = conn_fd;
    room->status = 1;
    return 0;
  }
  else if (room->client_fd[1] == 0)
  {
    room->client_fd[1] = conn_fd;
```

```
    room->status = 2;
    return 1;
  }
  return -1;
}
```

1.3 Tìm phòng cho người chơi

Đầu tiên nếu kiểm tra client chưa có phòng chơi PVP thì bắt đầu tạo phòng và chờ người chơi khác tham gia:

```
if (room == NULL)
  {
    is_found = 0;
    room = add_room();

    add_client_to_room(conn_fd, room);
    send_question = 1;
    start = time(NULL);
    seconds = 10; // end loop after this time has elapsed
    endwait = start + seconds;

    msg.type = WAIT_OTHER_PLAYER;
    strcpy(msg.value, "Đang tìm kiếm người chơi khác, xin chờ chút...\n");
    if (send(conn_fd, &msg, sizeof(msg), 0) < 0)
    {
      perror("Send error!");
      delete_client(conn_fd);
      return 0;
    }

    printf("Find opponent for room %d...\n", room->room_id);
```

Nếu quá 10s thì sẽ không tìm người chơi nữa và xóa phòng. Nếu tìm được người chơi khác thì thêm client đó vào phòng. Sau đó cài đặt vị trí trong room là 0 là chủ phòng, vị trí của thành viên là 1.

```
while (start < endwait)
    {
      if (room->client_fd[1] != 0)
      {
        is_found = 1;
        index_in_room = 0;
        index_doi_thu_in_room = 1;
        break;
      }
      sleep(1); // sleep 1s.
      // start = time(NULL);
```

```
      // printf("loop time is: %s", ctime(&start));
      printf("Finding opponent for room %d...\n", room->room_id);
    }
    // printf("end time is: %s", ctime(&endwait));
    printf("Found opponent for room %d...\n", room->room_id);
  }
  else
  {
    is_found = 1;
    add_client_to_room(conn_fd, room);
    send_question = 0;
    index_in_room = 1;
    index_doi_thu_in_room = 0;
  }

  if (is_found == 1)
  {
    msg.type = FOUND_PLAYER;
    strcpy(msg.value, "Đã tìm thấy người chơi khác, bắt đầu trò chơi!");

    if (send(conn_fd, &msg, sizeof(msg), 0) < 0)
    {
      perror("Send error!");
      delete_client(conn_fd);
      return 0;
    }

    room->play_status[index_in_room] = 0;
    msg.type = ENTERED_ROOM;

    sprintf(str, "%d", room->room_id);
    strcpy(msg.value, str);

    if (send(conn_fd, &msg, sizeof(msg), 0) < 0)
    {
      perror("Send error!");
      delete_client(conn_fd);
      return 0;
    }

    int room_id_current = room->room_id;
    strcpy(room->player_name[index_in_room],username);
```

11. Hiển thị danh sách người chơi online

```
void send_online_players(int conn_fd) {
    pthread_mutex_lock(&client_mutex);
    Client *current = head_client;
    char response[BUFF_SIZE] = "Danh sách người chơi đang online:\n";
    int count = 0;
    while (current != NULL) {
        count++;
        if (current->login_status == AUTH) {
            char count_str[20];
            sprintf(count_str, "%d. ", count);
            strcat(response, count_str);
            strcat(response, current->login_account);
            strcat(response, "\n");
        }
        current = current->next;
    }

    pthread_mutex_unlock(&client_mutex);

    Message msg;
    msg.type = HISTORY;
    strcpy(msg.data_type, "string");
    strcpy(msg.value, response);
    msg.length = strlen(response);

    if (send(conn_fd, &msg, sizeof(msg), 0) < 0) {
        perror("Gửi danh sách người chơi online thất bại");
    }
    return;
}
```

Hàm sử dụng tạo ra một khóa mutex tại thời điểm kiểm tra để khóa tài nguyên dùng chung để để tránh tình trạng "race condition" khi nhiều thread cùng thao tác trên danh sách client.
Sau đó hàm duyệt các client đang có trạng thái **AUTH** trên server rồi gộp lại trả kết quả về client theo message type là **HISTORY**.
Khi thiết kế tính năng này, do lúc đầu chưa bổ sung khóa mutex, khi nhiều client cùng lúc muốn xem danh sách người chơi online thì sẽ bị lỗi do đúng lúc 1 client đăng xuất hoặc có thêm một client mới đăng nhập.

12. Xử lý đồng bộ câu hỏi PVP.

Do lúc đầu thiết kế vẫn dùng hàm get_question() cho cả 2 client mà không đồng bộ nên khi PVP gây ra mỗi người có bộ câu hỏi khác nhau. Vậy nên em thiết kế để cho lấy câu hỏi từ chủ phòng và gửi cho cả 2 client trong phòng đó.

```
if(send_question){
      while (room->index_current_question < 15)
      {
        room->is_answered[index_in_room] = 0;
        room->is_answered[index_doi_thu_in_room] = 0;
        msg.type = QUESTION;
        snprintf(str, sizeof(str), "Câu %d: %s\n",
            room->index_current_question + 1,
            room->questions.question[room->index_current_question]);
        strcpy(msg.value, str);

        snprintf(str, sizeof(str),
"1. %.500s\n2. %.500s\n3. %.500s\n4. %.500s\n",
            room->questions.a[room->index_current_question],
            room->questions.b[room->index_current_question],
            room->questions.c[room->index_current_question],
            room->questions.d[room->index_current_question]);
        strcat(msg.value, str);

        send(room->client_fd[0], &msg, sizeof(msg), 0);
        send(room->client_fd[1], &msg, sizeof(msg), 0);
        room->index_current_question++;
```

Trong đó 2 hàm **send(room->client_fd[0], &msg, sizeof(msg), 0); send(room->client_fd[1], &msg, sizeof(msg), 0);** gửi câu hỏi đồng thời cho 2 client.

13. Xử lý xác định thắng thua PVP

Cơ chế xử lý thắng thua theo cơ chế thời gian trả lời:

- Nếu cả 2 người chơi đều trả lời đúng thì trả lời tiếp câu hỏi tiếp theo.
- Nếu 1 người đúng và 1 người sai câu hỏi đó thì xác định người chơi đúng thắng cuộc.
- Nếu cả 2 trả lời sai hoặc tới câu hỏi số 15 đúng cả 2 thì xác định ván đấu hòa.
- Nếu cả 2 quá thời gian trả lời thì xác định ván đấu hòa và đều xử trả lời sai câu hỏi đó. Vậy nếu 1 người trả lời đúng và người còn lại không trả lời thì người trả lời đúng thắng, còn người trả lời sai thì xử hòa.

Do code quá dài nên em sẽ không đưa vào báo cáo

14. Lưu thông tin ván đấu PVP

```
void insert_history_pvp(char username[], char opponent[], char result[]) {
    char query[500];
    pthread_mutex_lock(&mutex);

    snprintf(query, sizeof(query), "INSERT INTO history_pvp (username,
opponent, result) VALUES ('%s', '%s', '%s')", username, opponent, result);
    if (mysql_query(conn, query)) {
        fprintf(stderr, "Lỗi khi chèn vào cơ sở dữ liệu: %s\n",
mysql_error(conn));
    }
    pthread_mutex_unlock(&mutex);
}
```

Sau khi xử thắng thua thì sẽ dùng câu lệnh SQL để lưu vào CSDL. Ở đây cũng dùng lock mutex để tránh xung đột tài nguyên.

```
void get_history_pvp_by_username(char user_name[], int conn_fd) {
    char query[500];
    snprintf(query, sizeof(query),
        "SELECT username, result, opponent, DATE_ADD(play_time, INTERVAL 7
HOUR) AS adjusted_play_time "
        "FROM history_pvp WHERE username = '%s'", user_name);

    if (mysql_query(conn, query)) {
        fprintf(stderr, "Lỗi khi thực hiện truy vấn: %s\n",
mysql_error(conn));
        return;
    }

    MYSQL_RES *result = mysql_store_result(conn);
    if (result == NULL) {
        fprintf(stderr, "Lỗi khi lấy kết quả truy vấn: %s\n",
mysql_error(conn));
        return;
    }

    int num_fields = mysql_num_fields(result);
    if (num_fields == 0) {
        printf("Không tìm thấy lịch sử cho username: %s\n", user_name);
        mysql_free_result(result);
        return;
    }

    char response[BUFF_SIZE] = "";
    MYSQL_ROW row;
    while ((row = mysql_fetch_row(result))) {
        char row_result[256];
```

```
        snprintf(row_result, sizeof(row_result), "Username: %s %s
Username: %s || Time: %s\n",
                 row[0] ? row[0] : "NULL",
                 row[1] ? row[1] : "NULL",
                 row[2] ? row[2] : "NULL",
                 row[3] ? row[3] : "NULL");

        if (strlen(response) + strlen(row_result) < sizeof(response)) {
            strncat(response, row_result, sizeof(response) -
strlen(response) - 1);
        } else {
            printf("Dữ liệu quá lớn, không thể thêm vào.\n");
            break;
        }
    }

    Message msg;
    msg.type = HISTORY;
    strcpy(msg.data_type, "string");
    msg.length = strlen(response);
    strcpy(msg.value, response);

    ssize_t bytes_sent = send(conn_fd, &msg, sizeof(msg), 0);
    if (bytes_sent == -1) {
        perror("Gửi lịch sử thất bại");
    }
    mysql_free_result(result);
}
```

Khi người chơi cần xem lại lịch sử đấu thì cũng sử dụng hàm này để thực hiện câu lệnh SQL để truy vấn các ván đấu PVP đã đấu trong cơ sở dữ liệu rồi server gửi cho Client dưới dạng message là HISTORY.

15. Đồng bộ thời gian thi đấu PVP

Trong lúc thiết kế chế độ PVP, em nhận ra khi 1 người chơi trả lời thì lại có thể trả lời được tiếp câu hỏi tiếp theo. Tuy nhiên sẽ có vấn đề khi người chơi đó trả lời sai hoặc đến câu hỏi cuối từ rất sớm sẽ xảy ra trường hợp đợi người chơi còn lại trả lời rất lâu, hoặc xử lý kết quả trả về ngay lập tức khá phức tạp và khó khăn.

Vậy nên em quyết định xử lý đồng bộ thời gian theo từng câu hỏi. Khi cả 2 đều trả lời câu hỏi thì mới được chuyển qua câu hỏi tiếp theo để trả lời.

Giải pháp là chuyển từ room PVP chỉ có trạng trái qua xác định ai là chủ phòng ai là thành viên. Mọi thao tác liên quan tới phòng đều phải thông qua chủ phòng từ:

- Tạo phòng
- Xóa phòng
- Tạo bộ câu hỏi
- Gửi kết quả thắng thua
- Giới hạn thời gian trả lời từ thời điểm gửi câu hỏi
- Nhận câu hỏi
- Trả lời câu hỏi
- Nhận kết quả ván đấu

Các thao tác được cho phép ở client của thành viên là:

- Nhận câu hỏi
- Trả lời câu hỏi
- Nhận kết quả ván đấu

Do đó khi gửi kết quả về cho 2 client, thành viên sẽ được cho rời khỏi phòng trước. Sau đó mới thực hiện các thao tác xóa phòng tránh trường hợp khi thoát phòng thì xóa phòng luôn, client còn lại sẽ bị lỗi con trỏ tới phòng hiện tại rồi trỏ ra phòng NULL.

Cơ chế đồng bộ có liên quan chặt chẽ tới cơ chế xử lý thắng thua PVP, phần giải thích sẽ tương tự.

Phần đồng bộ này đã khiến em phải xóa tất cả cơ chế cũ đã thiết kế ban đầu của chế độ PVP và xây dựng lại từ đầu cho tới khi chạy ổn định là 2 tuần.

Do code phần này quá dài nên em sẽ không đưa vào báo cáo

Chương 5: Kết luận

Tổng kết

(% được tính dựa trên khối lượng công việc thực tế khi thực hiện dự án):

- Trịnh Hữu An: 19 điểm (55,9%)
- Phạm Quốc Minh: 15 điểm (44,1%)

Thông tin thêm

- Link Github: https://github.com/Carbon2301/Millionaire.

Báo cáo của nhóm em đến đây là hết, nhóm em xin cảm ơn thầy vì đã hướng dẫn để nhóm em có được kiến thức xây dựng nên dự án này. Chúc thầy một năm mới vui vẻ, bình an ạ!