매칭 로직 구조

먼저 게임서버가 로비와 연결이 되려면 조건이 필요합니다.

1. 로비서버가 켜져 있을 것
2. Ingame으로 들어가고 싶은 유저가 있을 것

2번 같은 경우는 플레이어가 Match Start버튼을 눌렀을 때입니다.

그 다음으로 게임서버가 켜지려면 조건이 필요합니다.

1. Ingame으로 들어가고 싶지만 로비서버가 관리하는 게임서버 중 Ready중인 게임서버가 없을 경우

이 조건을 만족하면 해당 게임서버의 Port번호를 부여하고 명령인수로 넣어서 게임서버를 가동하게 됩니다.

<로비서버 코드>

```
case CL_PACKET_MATCH_REQUEST: {
    cl_packet_match_request* packet = reinterpret_cast<cl_packet_match_request*>(p);
    bool bAllServerNotActivate = true;
    short AmountOfTryMatchingPlayer = 1;

    if (-1 != packet->amount)   //ai일경우 packet이 -1이 아니다.
        AmountOfTryMatchingPlayer = packet->amount;

    for (auto& server : servers) {
        GameServer* gameserver = reinterpret_cast<GameServer*>(server);
        gameserver->state_lock.lock();
        if (gameserver->_state == Server::STATE::ST_MATHCING)
        {
            gameserver->state_lock.unlock();
            bAllServerNotActivate = false;  //게임서버가 1개라도 켜져있으므로 false
            bool res = gameserver->Match(object->_id, AmountOfTryMatchingPlayer);
            if (!res)
            {
                cout << "매칭실패 매칭 재시도₩n";
                Timer_Event instq;
                instq.player_id = object->_id;
                instq.object_id = AmountOfTryMatchingPlayer;
                instq.type = Timer_Event::TIMER_TYPE::TYPE_MATCH_REQUEST;
                instq.exec_time = chrono::system_clock::now() + 1000ms;

                timer_queue.push(instq);
            }
            else {
                cout << "매칭성공₩n";
            }

        }
        else {
            gameserver->state_lock.unlock();
        }
    }

    //아무 게임서버도 켜져있지 않으면 새로 하나 켜줌.
    if (bAllServerNotActivate)
    {

        int newid = Generate_ServerId();
        Server* server = servers[newid];    //state == USING
        server->_id = newid;
        GameServer* gameserver = reinterpret_cast<GameServer*>(server);
        gameserver->mServerPort = 4000 + newid; //포트넘버
        wchar_t tmp[20];
        _itow_s(gameserver->mServerPort, tmp, 10);
        ShellExecute(NULL, TEXT("open"), TEXT("../FPP_Server₩₩x64₩₩Release₩₩FPP_Server.exe"), tmp, NULL, SW_SHOW);

        cout << "서버 열린곳이 없음. 매칭 재시도₩n";
        {
            Timer_Event instq;
            instq.player_id = object->_id;
            instq.object_id = AmountOfTryMatchingPlayer;
            instq.type = Timer_Event::TIMER_TYPE::TYPE_MATCH_REQUEST;
            instq.exec_time = chrono::system_clock::now() + 1000ms;

            timer_queue.push(instq);
        }
    }

    break;
}
```

<게임서버 코드>

ShellExecute함수를 통해 켜진 게임서버는 전달받은 Port번호로 bind listen합니다.

```
int main(int argc, char* argv[])
{

    wcout.imbue(locale("korean"));
    WSADATA WSAData;
    WSAStartup(MAKEWORD(2, 2), &WSAData);
    short server_port = -1;
    if(argc >1)
        server_port = atoi(argv[1]);

    s_socket = WSASocket(AF_INET, SOCK_STREAM, IPPROTO_TCP, 0, 0, WSA_FLAG_OVERLAPPED);
    SOCKADDR_IN server_addr;
    ZeroMemory(&server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    if (server_port != -1)
        server_addr.sin_port = htons(server_port);
    else
        server_addr.sin_port = htons(GAMESERVER_PORT);
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    bind(s_socket, reinterpret_cast<sockaddr*>(&server_addr), sizeof(server_addr));
    listen(s_socket, SOMAXCONN);
    //cout <<   ntohs(server_addr.sin_port) << endl;
    hiocp = CreateIoCompletionPort(INVALID_HANDLE_VALUE, NULL, NULL, 0);
    CreateIoCompletionPort(reinterpret_cast<HANDLE>(s_socket), hiocp, 0, 0);
```

로비서버와 연결한 후 로비서버로 자신의 포트번호(정확히는 로비서버가 부여해준 포트번호)를 담아 인증절차를 거칩니다.

```
//-------------
mServer = new Server();
mServer->_socket = WSASocketW(AF_INET, SOCK_STREAM, IPPROTO_TCP, NULL, 0, WSA_FLAG_OVERLAPPED);
ZeroMemory(&mServer->server_addr, sizeof(mServer->server_addr));
mServer->server_addr.sin_family = AF_INET;
mServer->server_addr.sin_port = htons(LOBBYSERVER_PORT);
inet_pton(AF_INET, "127.0.0.1", &mServer->server_addr.sin_addr);
mServer->wsa_ex_recv.getWsaBuf().buf = reinterpret_cast<char*>(mServer->wsa_ex_recv.getBuf());
mServer->wsa_ex_recv.getWsaBuf().len = BUFSIZE;
mServer->wsa_ex_recv.setCmd(CMD_SERVER_RECV);
ZeroMemory(&mServer->wsa_ex_recv.getWsaOver(), sizeof(mServer->wsa_ex_recv.getWsaOver()));
CreateIoCompletionPort(reinterpret_cast<HANDLE>(mServer->_socket), hiocp, 1, 0);

int rt = connect(mServer->_socket, reinterpret_cast<sockaddr*>(&mServer->server_addr), sizeof(mServer->server_addr));
if (SOCKET_ERROR == rt)
{
    std::cout << "connet Error :";
    int err_num = WSAGetLastError();
    error_display(err_num);
    system("pause");
    //exit(0);
    closesocket(mServer->_socket);
    return false;
}
```

```
gl_packet_login packet;
packet.size = sizeof(packet);
packet.type = GL_PACKET_LOGIN;
packet.port = server_port;
mServer->sendPacket(&packet, sizeof(packet));


//-----------------------------------------
```

<로비서버 코드>

게임서버로부터 해당 패킷을 받게 된다면 해당 게임서버를 로비서버에서 관리할 수 있게 오브젝트 풀에 넣고 Match가능한 상태로 활성화시켜줍니다.

```
case GL_PACKET_LOGIN: {
    //게임서버는 초기에 player로 등록되지만, login 패킷을 보냄으로써
    //비로소 게임서버 오브젝트 풀에 등록되게 된다.
    //따라서 character에서 필요한 소켓은 복사해오도록 한다.
    //이후 character는 비워주게 된다.
    // 하지만 io에 등록된 key값(CompletionKey)는 바꿀수가 있나? 없는 것 같다.
    //그럼 process_packet자체에서 사용하고 있는 client_id와 object[client_id]의
    //sync가 맞지 않게 될 것이다. ex) 게임서버로 key가 1만큼 밀리게 되면
    // object 0 번째의 key는 1번이 될테지만
    // 여기서 client_id로 오는것 또한 1이 와서
    // object[0]이 아닌 object[1]을 참조하게 되는것
    // 이것에 대한 해결방법은 ?
    // 그냥 무시하고 key를 보내고, 서버와 관련된놈은 어차피 여기서 한번 더 캐스팅 해서 없앤다?
    // 이거 나쁘지않은 idea인듯. 실행해보자.
    // completion key가 단순히 전달만 해준다고 가정했을 때의 이야기이다.
    // 윗 아이디어는 어차피 workerThread에서 key값을 통해 prev_size를 얻어내야 해서 말이 안되는 방법.
    // iocp_key를 사용하지 않고, wsaoverlapped 확장 구조체에 key값을 넣어서 사용하자.
    // 이를 위해 서버전용 wsa_over_only_server를 만들고, serverID값을 따로 넣는다.
    // 결과는 성공적.

    gl_packet_login* packet = reinterpret_cast<gl_packet_login*>(p);
    Player* character = reinterpret_cast<Player*>(object);

    for (auto& server : servers)
    {
        GameServer* gameserver = reinterpret_cast<GameServer*>(server);
        if (gameserver->mServerPort == packet->port)
        {
            server->_socket = character->_socket;
            server->_prev_size = 0;
            server->wsa_server_recv.getWsaBuf().buf = reinterpret_cast<char*>(server->wsa_server_recv.getBuf());
            server->wsa_server_recv.getWsaBuf().len = BUFSIZE;
            server->wsa_server_recv.setID(server->_id);
            server->wsa_server_recv.setCmd(CMD_SERVER_RECV);    //이제 서버관련 패킷은 따로 처리 해준다.
            ZeroMemory(&server->wsa_server_recv.getWsaOver(), sizeof(server->wsa_server_recv.getWsaOver()));


            server->recvPacket();
            server->state_lock.lock();
            server->_state = Server::STATE::ST_MATHCING;
            server->state_lock.unlock();
            //server assign end

            //clear character
            character->state_lock.lock();
            character->_state = Player::STATE::ST_FREE;
            character->state_lock.unlock();
            character->_socket = NULL;
            //clear character end
            lg_packet_login_ok spacket;
            memset(&spacket, 0, sizeof(spacket));
            spacket.size = sizeof(spacket);
            spacket.type = LG_PACKET_LOGIN_OK;
            server->sendPacket(&spacket, sizeof(spacket));
        }
    }



    break;
}
```