



# Adaptable Online Game Server Design

Jintaek Seo\* , Member, KIICE

Game Development Track of International College, Dongseo University, Busan 47011, Rep. of Korea

## Abstract

This paper discusses how to design a game server that is scalable, adaptable, and re-buildable with components. Furthermore, it explains how various implementation issues were resolved. To support adaptability, the server comprises three layers: network, user, and database. To ensure independence between the layers, each layer was designed to communicate with each other only via message queues. In this architecture, each layer can have an arbitrary number of threads; thus, scalability is guaranteed for each layer. The network layer uses input/output completion ports (IOCP), which shows the best performance on the Windows platform, it can handle up to 5,000 simultaneous connections on a typical entry-level computer, despite being built with a single-threaded user layer. To completely separate the database from the game server, the SQL code was not directly embedded in the database layer.

**Index Terms:** IOCP, Layered Server Architecture, Multi-threaded Architecture, Online Game Server

## I. INTRODUCTION

During the design and implementation of a game server, it is important to create and maintain a reusable code. Servers with desired functions can be configured by assembling reusable server components as blocks. A reusable component means the combination of functions and classes that make up a specific feature of the server. Reuse of a component requires a standard environment, wherein the component operates accurately.

The basic structure of the game server can be designed such that the cost of the components, which make up the server is minimized. Different games have several similar components irrespective of their genre; therefore, it can be useful to design a common game server structure. Constructing components that can be used by most servers can reduce the server building cost. Additionally, the components can be configured such that they can be easily extended when the devices are required to be extended in parallel.

In this study, a scalable server architecture that minimizes the cost of the components being used for the server is proposed and implemented.

## II. RELATED WORKS

Modern massively multiplayer online game servers capable of handling a large number of connections have nearly identical cell or grid structures [1]. Furthermore, their performances are constantly improving owing to lock-free containers [2]. The design of the game server facilitate communication between different servers. Moreover, the server configuration should make the server adaptable to meet the needs of various game clients [3, 4]. This study proposes a design method based on components and layers so that a game server can be configured to satisfy various requirements of the game clients.

In a real-world scenario, several users are simultaneously connected to an online game. Access information of each


Received 16 March 2020, Revised 10 June 2020, Accepted 15 June 2020

\*Corresponding Author Jintaek Seo (E-mail: [jintaeks@dongseo.ac.kr](mailto:jintaeks@dongseo.ac.kr), Tel: +82-51-320-2955)

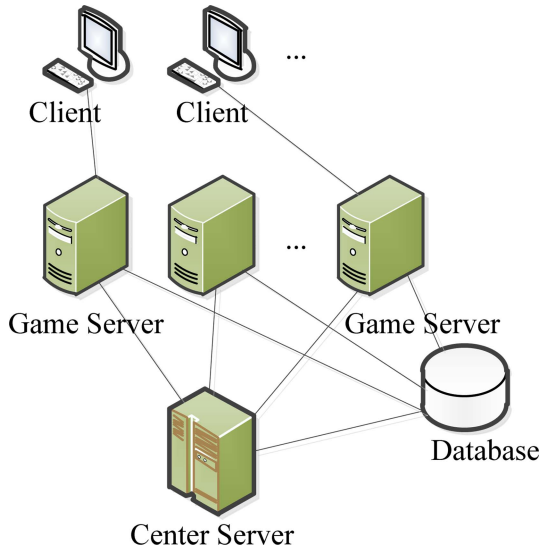
Game Track, International College, Dongseo University, Busan, 47011, Rep. of Korea.

Open Access <https://doi.org/10.6109/jicce.2020.18.2.82>

print ISSN: 2234-8255 online ISSN: 2234-8883

 This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (<http://creativecommons.org/licenses/by-nc/3.0/>) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

Copyright © The Korea Institute of Information and Communication Engineering



**Fig. 1.** Clients connected to game servers. One center server manages multiple game servers. All servers are connected to a database.

user is managed in the database. However, a single game server cannot process all the users managed by the database. Therefore, multiple game servers connect to one database. In this configuration, users should be able to chat or set up a party with other users, even if they are connected to different game servers. Therefore, a server that connects multiple game servers is required [5, 6]. Such a server is termed as the “center server” in this paper.

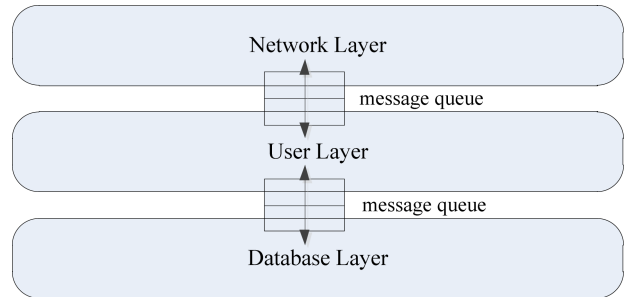
In this study, an architecture is proposed that is applicable to all types of game servers configured in the structure of a typical online game server as shown in Fig. 1.

### III. LAYERED ARCHITECTURE OF GAME SERVER

The game server receives packets from multiple clients such as mobiles, desktops, and web applications [7]. It maintains the information of all connected clients [7]. It maintains the information of all connected clients, processes game-related requests, and handles processing between the clients when necessary. Additionally, it stores the clients' game progress information in the database. This server must be capable of handling a DoS attack [8].

Based on its core functions, a game server can comprise the following three layers: the network, user and database layers. The network layer facilitates packet exchange with clients; the user layer processes user information; and the database layer processes database requests. Each layer must be independently extensible and maintainable; therefore, the structure design should minimize dependency between layers.

This can be achieved by a server design involving three layers as shown in Fig. 2. To reduce dependency between the



**Fig. 2.** Layered architecture of game server.

layers, they are prohibited from directly communicating with each other; they can communicate only through message queues.

#### A. Network Layer

The input/output completion ports (IOCP) is the most preferred way to send and receive packets using sockets on the Windows platform [9]. The socket's accept function is a blocking call; therefore, it is necessary to process this functionality in a thread. If the connection handling function of a socket creates and returns a socket for a new connection, it will be associated with the IOCP. The I/O for such a socket can be notified via the IOCP. To receive IOCP notification, a IOCP worker thread calls the IOCP event wait function for the registered port of IOCP. Each IOCP worker thread finds the client information for the completion of I/O, constructs the packet by deserializing the packet for each I/O, and pushes the constructed packet into the queue of the corresponding user.

In certain cases, when the game server needs to communicate to the center server, it directly sends the packets to the center server.

#### B. User Layer

The user layer can be designed to be either single- or multi-threaded, depending on the operating mode of the game. A component operating in a single thread environment can be built first; then, by creating a user distribution manager, a scalable user layer can be built. In this study, a single-threaded user layer was implemented.

In the user layer, each user has its own message queue. However, in the multi-threaded environment, a message queue may be shared for communication between groups of users within the user layer. The user layer has a loop that processes user requests. It operates on an individual user basis. Furthermore, a method must be designed to obtain the user information of other users connected to other servers when requested by a user. For database processing, the user communicates through the database message queue.

### C. Database Layer

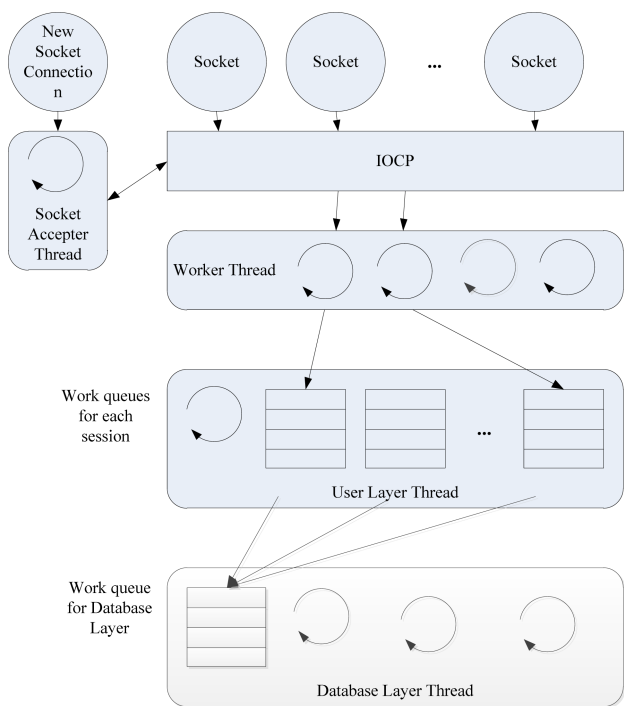
The database layer is responsible for processing messages intended for the database. For maximum performance, multiple database related message processing threads run simultaneously, where each thread's messages are independent of the other threads' messages. All dependencies between the messages are handled at the user layer; therefore, there is no dependency between the messages at the database layer. This configuration guarantees the scalability of the database layer.

To guarantee independence between the game server and database, all SQL routines that modify database tables operate only in the database; the game server only calls a corresponding stored procedure. Fig. 3. illustrates the interaction between these three layers.

### D. Key Components and Implementation Issues

#### 1) Thread classes

Class KThread is generally defined for the operation of multiple threads. This class can be inherited to implement a specific thread. In our study, for the network layer socket connections, we define the class: KSocketAcceptorThread. For the IOCP working thread in the network layer, we define the class: KIocpWorkerThread. We define the class: KUser-



**Fig. 3.** Layered game server structure with components: Each socket connection has a corresponding IOCP. IOCP worker threads process I/O data. The user layer thread creates a queue for each connection. There is one thread in the user layer. Although there are several threads in the database layer, there is only one queue for the database related message.

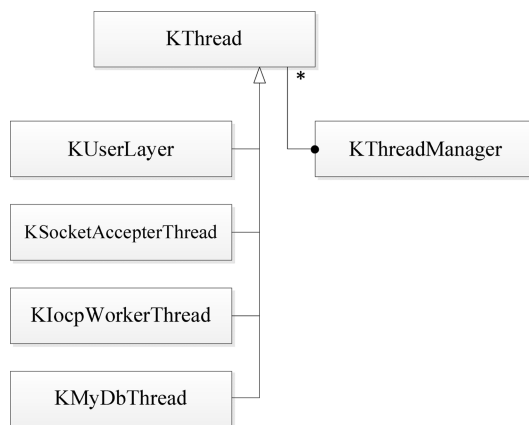
Thread to handle the processing in the user layer. Furthermore, we define the class: KMyDbThread that processes database messages at the database layer. Each thread has reference information about the thread manager that manages it. If the thread is not managed by the manager, the reference to KThreadManager is set as NULL.

Given that only queues are shared between threads, there are no synchronization issues for global data. Fig. 4 illustrates the class hierarchy of the thread classes.

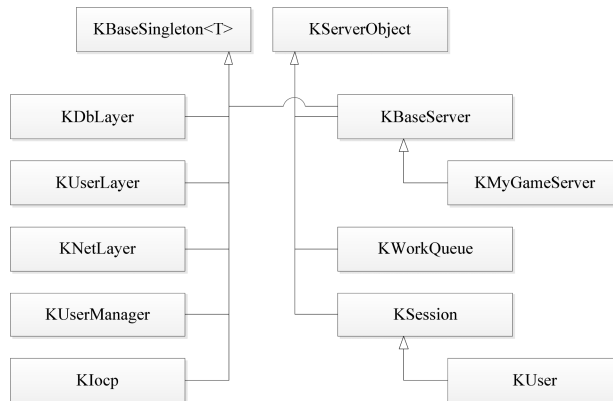
#### 2) Layer classes

There is only one instance for each layer. We implemented the KNetLayer, KUserLayer, and KDbLayer layers by inheriting the classes from KBaseSingleton for the functionality of a singleton. An instance of class KUser is created for each client connection, class KUser inherits from class KSession because it has the socket connection information. If multiple threads are used for the user layer, users belonging to the same thread group are managed by one KUserManager.

This structural information is managed by a server object of class KBaseServer. Class KIocp that is responsible for the



**Fig. 4.** Class hierarchy of threads in game server.



**Fig. 5.** Class hierarchy of the game server layers.

IOCP function is accessed only at the network layer. KIoCP is a singleton class; however in our implementation, another layer cannot access the instance of this class.

The queue for message communication in each thread is implemented as a class KWorkQueue, and each instance that requires message communication has a work queue. Fig. 5 illustrates the class hierarchy of these classes.

**3) Packet design issues**

Each layer communicates only via messages; therefore, we must define multiple messages for a specific task. The server wraps the packet in the form of a message that is then sent/received to/from a layer or another server or database.

Table 1 lists the machines that can send or receive messages.

Packets that request database processing are only used in the server; it starts with DB to distinguish from normal network layer packet that starts with GS.

The message name has the following format:

2Byte Sender + 2Byte Receiver + Message Name

Table 2 lists possible messages sent and received between machines.

For example, the message sent by the client to the game server to update the user's health power (HP) is CLGS\_UPDATE\_HP\_REQ.

The header of the message data includes the IDs of the sending and receiving layers. Therefore, when the server processes a message, it determines the target layer and enqueues the message to the target layer's receive queue.

**Table 1.** 2-Character Machine Type IDs.

Sender type	Packet prefix ID
Client	CL
Center Server	CS
Game Server	Network Layer User Layer Database Layer
	GS N/A DB
Database	N/A

**Table 2.** 4-Character Sender Receiver ID.

Sender	Receiver						
	CL	CS	GS	User-Layer	DB	Data-base	
CL	CLCL	N/A	CLGS	N/A	N/A	N/A	
CS	N/A	N/A	CSGS	N/A	N/A	N/A	
GS	GSCL	GSCS	GSGS	N/A	DB	N/A	
User Layer	N/A	N/A	N/A	N/A	DB	N/A	
DB	N/A	N/A	DB	DB	N/A	SP Call	
Database	N/A	N/A	N/A	N/A	SP Ret		

For example, to update user's HP, the following messages are required:

- ✓ CLGS\_UPDATE\_HP\_REQ : Client requests HP update
- ✓ DB\_UPDATE\_HP\_REQ : User layer enqueues it to DB layer
- ✓ DB\_UPDATE\_HP\_ACK : Database response
- ✓ GSCL\_UPDATE\_HP\_ACK : User layer responds to client

Messages that require a response specify a REQ/ACK at the end of the message name. For packets that only intend to inform, a "NOT" is specified at the end of the name.

**4) Database issues**

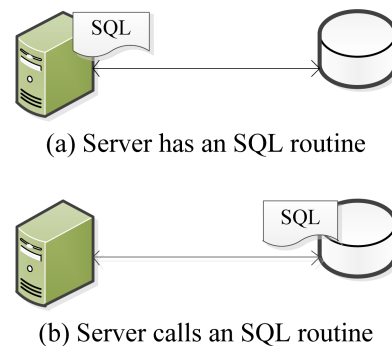
We built a database using Microsoft SQL Server. The SQL used to retrieve information from the database can be implemented either on the game server or in the database.

The problem with implementing SQL on a game server is that server and the database development are not independent. The database must be maintainable independent of the server.

Therefore, the SQL routine to obtain the user's information should be present in the database; the game server should only request and receive the result according to the SQL interface. This is depicted in Fig. 6. To achieve this, for each stored procedure in the database, we must generate the appropriate corresponding class source to be used by the game server. These tasks have specific rules; therefore, we have created a tool that generates classes from SQL; this avoids errors.

**5) Operator's issues**

Online games usually have regular weekly checks. However, stopping the operation of a game server should be done with caution because all users get disconnected. In certain cases, it is necessary to activate the game server's specific features, observe statistical metrics, etc. without stopping the game server.



**Fig. 6.** Game server can host SQL routines; however, to separate server and database development, the server should be configured only to call stored procedures.

**Table 3.** Game Server Test Environment

Item	Details
CPU	Intel Core i7-6, 500U
Core Speed	3,000 MHz
# of Cores	2
Memory Type	DDR3
Memory Size	8 GB
# of IOCP Worker Threads	4
# of User Threads	1
# of DB Worker Threads	4
# of Socket Acceptor Threads	1
Maximum # of Robots	5,000 clients

**Table 4.** Test Results

# of Clients	Memory (Unit: MB)	CPU(%)
1,000	4,196	81
2,000	4,697	81
3,000	5,202	82
4,000	5,708	83
5,000	6,310	83

To this end, the main thread of the game server must have the features to export functions that the operator can call and execute from the server console. We added an operator console to the main thread. A built-in scripting language allows the operator to control certain server functions [10].

#### IV. IMPLEMENTATION AND TEST

We measured the performance of the game server on a typical entry-level computer. Table 3 presents the server test environment.

Table 4 presents the server test result for multiple clients. We first created a test client that connects to the server and then performed a load test. The server, implemented on the Windows platform, provided reliable performance up to 5,000 client connections. Every time the number of client connections grows, the memory requirement increases linearly; however, the CPU usage does not change significantly. Fig. 7. and Fig. 8. show online games developed by KOG. These games use the server architecture proposed in this paper.

#### V. CONCLUSION

In this study, we divided the basic components of the game server into three layers. We implemented the required functions in each layer and the send and receive information



**Fig. 7.** Elsword developed by KOG uses the server architecture proposed in this paper.



**Fig. 8.** Ultimate Race developed by KOG uses the server architecture proposed in this paper.

through message queues so that each layer can be extended independently. This implementation demonstrated that the game server can handle up to 5,000 concurrent connections even though the user layer is implemented as a single thread.

The implemented server has a task-parallel architecture. However, if we apply data-parallel architecture to the user layer, the server architecture can be made comparatively more flexible. As future research, we will apply data parallelism to improve server performance.

#### APPENDIX

##### A. Online game based on the proposed game server architecture

Elsword (Korean: 엘소드) is a free-to-play, 2.5D action MMORPG developed by the South Korean company: KOG Studios. It features real-time action gameplay and includes both player vs. environment and player vs. player modes.

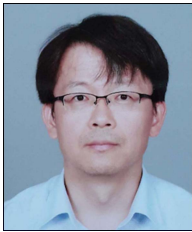
Ultimate Race is an MMORPG style racing game developed by KOG.

## ACKNOWLEDGEMENT

We would like to thank Editage ([www.editage.co.kr](http://www.editage.co.kr)) for English language editing and KOG([www.kog.co.kr](http://www.kog.co.kr)) for game posters.

## REFERENCES

- [ 1 ] B. V. D. Bossche, T. Verdickt, B. D. Vleeschauer, S. Desmet, S. D. Mulder, F. D. Turck, B. Dhoedt and P. Demeester, "A platform for dynamic microcell redeployment in massively multiplayer online games," in *NOSSDAV '06 Proceedings of the 2006 International Workshop on Network and Operating Systems*, Nov., 2006. DOI: 10.1145/1378191.1378195.
- [ 2 ] K. Raaen, H. Espeland, H. K. Stensland, A. Petlund, P. Halvorsen and C. Griwodz, "A demonstration of a lockless, relaxed atomicity state parallel game server (LEARS)," in *NetGames '11 Proceedings of the 10th Annual Workshop on Network and Systems Support for Games*, Oct., 2011. DOI: 10.1109/NetGames.2011.6080994.
- [ 3 ] F. Glinka, A. Ploss and S. Gorlatch, "RTF: a real-time framework for developing scalable multiplayer online games," in *Proceedings of the 6th ACM SIGCOMM Workshop on Network and System Support for Games(NetGames '07)*, pp. 81-86, Melbourne, Australia, Sep. 2007. DOI: 10.1145/1326257.1326272.
- [ 4 ] A. Ploss, F. Glinka, S. Gorlatch and J. Müller-Iken, "Towards a high-level design approach for multi-server online games," in *Proceedings of the 8th International Conference on Intelligent Games and Simulation (GAMEON '07)*, pp. 10-17, Bologna, Italy, Nov. 2007.
- [ 5 ] J. Brun, F. Safaei, and P. Boustead, "Server topology considerations in online games," in *Proceedings of the 4th ACM Network and System Support for Games (NetGames)*, (Singapore), Oct. 2006. DOI: 10.1145/1230040.1230094.
- [ 6 ] K. W. Lee, B. J. Ko, and S. Calo, "Adaptive server selection for large scale interactive online games," in *Proceedings of NOSSDAV*, (Kinsale, County Cork, Ireland), Jun. 2004. DOI: 10.1016/j.comnet.2005.04.006.
- [ 7 ] M. H. Choi and I. Y. Moon, "Development of branch processing system using WebAssembly and JavaScript," *Journal of Information and Communication Convergence Engineering*, vol. 17, no. 4, pp. 234-238, Dec. 2019. DOI: 10.6109/jicce.2019.17.4.234.
- [ 8 ] K. Rim and D. Lim, "DoS attack control design of IoT system for 5G era," *Journal of Information and Communication Convergence Engineering*, vol. 16, no. 2, pp. 93-98, Jun. 2018. DOI: 10.6109/jicce.2018.16.2.93.
- [ 9 ] A. Jones and J. Ohlund, *Network Programming for Microsoft Windows*, 2nd ed, Microsoft Press, 2002.
- [10] W. Celes, L. H. de Figueiredo et al, "Binding C/C++ objects to lua," in *Game Programming Gems 6*, M. Dickheiser, Ed., pp. 341-355, Charles River Media, Rockland, Mass, USA, 2006.



### Jintaek Seo

received his B.E. and M.S. degrees in 1996 from Kyungpook National University, Daegu, Korea. He completed his Ph.D. course from Kyungpook National University. He co-founded a game company after graduation in 2000 and worked with the same for 16 years before moving to Dongseo University, Busan, Korea. His research interests include game server, game design, 3D game technology, AI, and deep neural networks.