Design and Implementation of Transfer Buffer Sharing Technique for Efficient Massive Data Transfer

Dae-Soo Cho, *Member, KIMICS*

Abstract—It is required that a server which communicates with various client simultaneously should have an efficient data transfer model. In Windows® environment, the server was generally developed based on IOCP model. Developing the IOCP model, the server generally has one data transfer buffer per client. If the server divides a larger data than the transfer buffer into several fragments, there used to be a problem in sending it to a client, because there is a conflict in a data transfer buffer. That is, CPU requests one data-fragment transfer, then it will request the next data-fragment transfer successively before completing the previous request, owing to the property of overlapped IO model. In this paper, we proposed the transfer buffer sharing technique to solve the conflicting problem. The experimental result shows that the performance of data transfer was enhanced by 39% maximally.

Index Terms—IOCP, Buffer Sharing, Massive Data Transfer, Client-Server System, Healthcare Monitoring

I. INTRODUCTION

It is required that a server which should support real time communication with various client simultaneously should have an efficient data transfer model. In windows environment, the server was generally developed based on IOCP model. IOCP, one of the most powerful I/O model, is based on asynchronous (non-blocked), overlapped IO. In IOCP, the server issues the read/write operation asynchronously, and the operation will be completed later. That is, after CPU issues the read/write operation, it is not blocked to wait the completion of the operation. The throughput of CPU, therefore, is higher than in the synchronous I/O [1] [2].

Developing the IOCP model, the server generally has one data transfer buffer per client, and this buffer must remain valid until the I/O operation completes. If the server divides a larger data than the transfer buffer into several fragments, there used to be a problem in sending it to a client, because there is a conflict in a data transfer buffer. That is, CPU issues a write operation for a data-fragment, and then it will successively issue another write operation for the next data-fragment before completing the previous write operation.

In this paper, we proposed the transfer buffer sharing technique to solve the conflicting problem. The experimental result shows that the performance of data transfer was enhanced by 39% maximally. It is expected for the server to transfer massive data efficiently.

The rest of the paper is organized as follows: In section II, we will overview the IO completion port. In section III, we will explain the problem and solution briefly, and then we will show the experimental result in section IV. Finally, we will conclude by giving directions for future work in section V.

II. Overview of IOCP I/O Model

Generally, compared to the speed of CPU, I/O devices are considered extremely slow. In synchronous I/O depicted in Fig. 1.(a), the throughput of CPU, therefore, is very low due to the delay time. This means that the CPU spends almost all of its time idle waiting for I/O operations to complete.

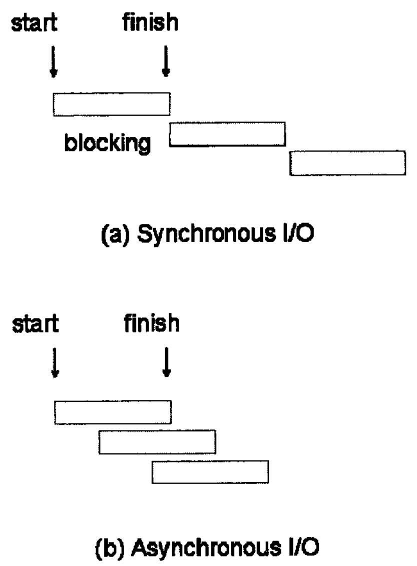


Fig. 1 Two different I/O models: Synchronous I/O model VS Asynchronous I/O model

Asynchronous I/O (see Fig. 1.(b)) is used to improve throughput, latency, and/or responsiveness. An asynchronous I/O request returns immediately, leaving the I/O call pending. At some point of time, the result of the I/O asynchronous call must be synchronized with the

Manuscript received March 11, 2008; revised August 5, 2008. Dae-Soo Cho is with the Division of Computer and Information, Dongseo University, Busan, 617-716, Korea (Tel: +82-51-320-1897, Fax: +82-51-312-2329, Email: dscho@dongseo.ac.kr)

main thread. This can be done in different ways. The synchronization can be performed by using event, asynchronous procedure calls(APC), or IOCP.

IO Completion Port (IOCP) was introduced in Windows to suite the needs of an architecture which could best fit in a server application [3]. The server application should be able to serve multiple clients with a limited number of threads, which constitute the thread pool. The number of these limited threads depends upon the number of processors. An IOCP object is associated with several I/O objects (i.e. sockets) that support pending asynchronous I/O calls. A thread that has access to an IOCP can be suspended until a pending asynchronous I/O call is finished [4] [5].

A completion port is a queue into which the operating system puts notifications of completed overlapped I/O requests. Once the operation completes, a notification is sent to a worker thread that can process the result. A socket may be associated with a completion port at any point after creation [6].

As shown in Fig. 2, while working with IOCP, you have to deal with three things, (1) associating a socket to the completion port, (2) making the asynchronous I/O call, and (3) synchronization with the thread. To get the result from the asynchronous I/O call and to know, for example, which client has made the I/O request, you have to pass two parameters: the CompletionKey parameter, and the OVERLAPPED structure.

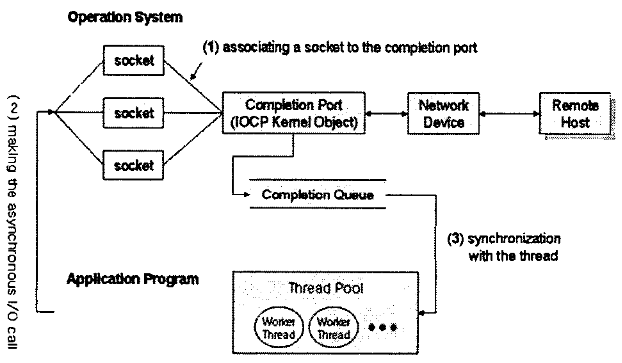

Operation System
(1) associating a socket to the completion port
socket
socket
socket
Completion Port (IOCP Kernel Object)
Network Device
Remote Host
Completion Queue
(3) synchronization with the thread
Application Program
Thread Pool
Worker Thread
Worker Thread
(2) making the asynchronous I/O call

Fig. 2 I/O operation with IOCP

By using IOCP, we can overcome the "one-thread-per-client" problem. It is commonly known that the performance decreases heavily if the software does not run on a true multiprocessor machine. Threads are system resources that are neither unlimited nor cheap. IOCP provides a way to have a few (I/O worker) threads handle multiple clients' input/output "fairly". The threads are suspended, and don't use the CPU cycles until there is something to do.

III. Massive Data Transfer

A. Problem Definition

The OVERLAPPED structure parameter is commonly used to pass the memory buffer that is used by the asynchronous I/O call. Generally, the "ONE-buffer-per-client" is used in client/server communication by using IOCP and this buffer allocation method works well except when a server is willing to send a larger data than the transfer buffer. Sending the larger data, the server should divide it into several fragments which are fit for buffer size. In the asynchronous I/O, the CPU issues a write operation for a data-fragment, and then it will successively issue another write operation for the next data-fragment before completing the previous write operation. Therefore, a data-fragment could overlap in the buffer with the previous data-fragment which is not sent completely yet (as shown in Fig. 3).

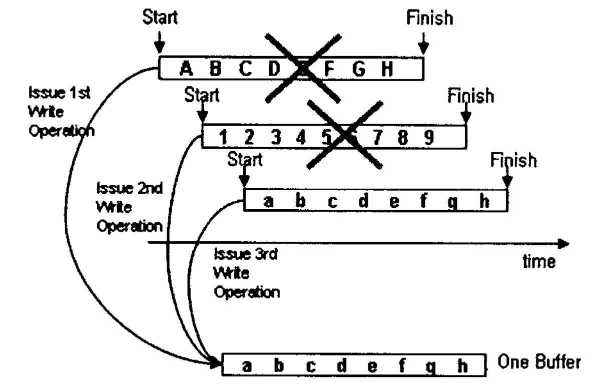


Fig. 3 Transfer Buffer Conflict in the Asynchronous I/O

B. Simple Solution with One Buffer: OB

To simply solve the buffer conflict problem in "one-buffer-per-client" environment, the server application issues the write operation just for 1st data-fragment of the large data, and then the write operation for the next data-fragment could be issued by the worker thread. This worker thread is activated by IOCP after the previous write operation is completed. That means the buffer is not empty-status. We call this method as OB ("One Buffer").

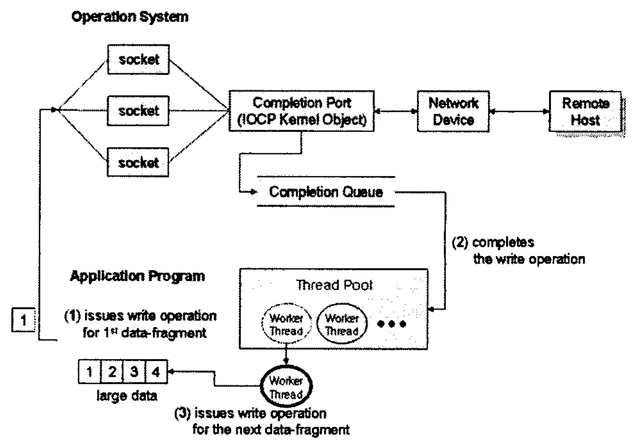

Operation System
socket
socket
socket
Completion Port (IOCP Kernel Object)
Network Device
Remote Host
Completion Queue
(2) completes the write operation
Application Program
Thread Pool
Worker Thread
Worker Thread
1
(1) issues write operation for 1st data-fragment
1 2 3 4
large data
Worker Thread
(3) issues write operation for the next data-fragment

Fig. 4 Simple Solution for Buffer Conflict Problem in "one-buffer-per-client" Environment

C. Buffer Sharing Solution: BS

In this paper, we have proposed another solution for the buffer conflict problem which is to assign buffers as many as the number of data-fragment. This means that a client might have several buffers if needed. Because each data-fragment is written in different buffer, there is no conflict any more as shown in Fig. 5.

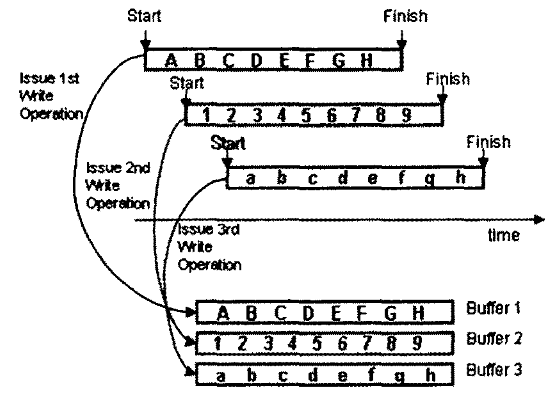


Fig. 5 Example: Three Buffers for One Client

These buffers could be *(1) dynamically allocated*, or *(2) statically reused* from a buffer pool which is consisted of buffers which have been allocated. In general, to allocate and free memory is expensive, therefore we should reuse buffers. We call this method as BS ("Buffer Sharing"). Fig. 6 shows the memory allocation and de-allocation model for one-buffer-per-client and buffer sharing. Buffer sharing means that the buffers in a pool are not dedicated by some specific clients, but could be used by any client.

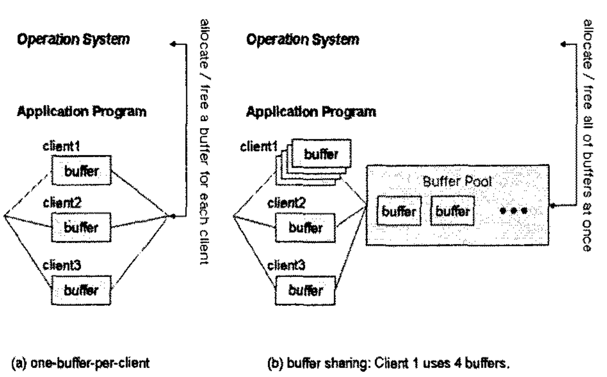

Operation System
Application Program
client1
buffer
client2
client3
allocate / free a buffer for each client
Buffer Pool
allocate / free all of buffers at once
(a) one-buffer-per-client
(b) buffer sharing: Client 1 uses 4 buffers.

Fig. 6 Memory Allocation & De-allocation Model

There certainly occurs a problem in one-buffer model. Because the server does not communicated with all of the clients concurrently, a lot of buffers allocated by clients that are not communicated, but just connected to the server are frequently not used for a while. In this case, due to the limit of available memory resources of the server, the number of client of one buffer model is less than that of buffer sharing model.

IV. EXPERIMENTAL EVALUATION

A. Test Environment

In this paper, we have evaluated the massive data transfer speed in order to compare BS proposed in this paper with OB. The Test environment for evaluation is as follows:

- CPU: Intel Pentium 4 2.80GHz
- RAM: Samsung DDR 1.75GB
- O/S: Windows XP Professional SP 2
- Buffer Size: 4Kbyte
- Number of IOCP Worker Thread: 4

The parameters for performance evaluations are as follows. The size of transfer data ranges from 10 to 1000 times larger than the size of transfer buffer (4KB). That is, we have evaluated the performance of data transfer with massive data of which size is maximally 1000 times larger than the buffer size.

- The size of transfer data: 10*4, 100*4, 1000*4(KB)
- The number of client: 10, 100
- The size of buffer pool: 1000*4(KB)

B. Evaluation Results

The massive data transfer speed evaluated in this paper is calculated as data size divided by elapsed time for the client to receive from the 1st data-segment to the last data-segment. Fig. 7 shows the evaluation results. The y-axis means the ratio of data transfer speed of BS over data transfer speed of OB. Ten hundred percent means that two methods have equal performance. The ratio value above 100% means that BS method has better performance than OB method. The evaluation results range from 98% to 105.9%.

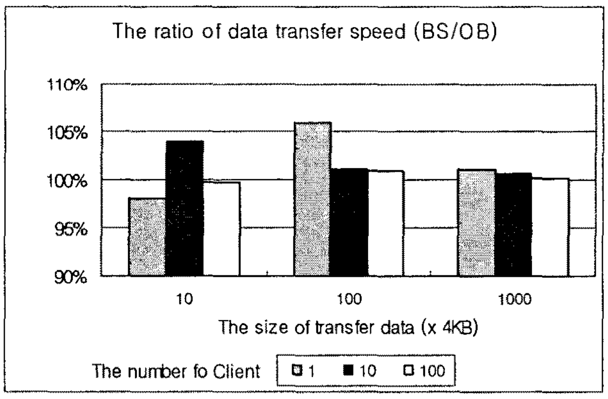


Fig. 7 The Ratio of Data Transfer Speed: Evaluation results are dependent on network environment. The Server and clients are different machines.

There is a few difference between OB method and BS method, because the data transfer speed itself depends on network environment. Because the buffer sharing make the throughput of CPU higher, another performance measure is required. In this paper, therefore, the ratio of throughput is evaluated by loading the server and the clients at the same machines. The experimental results depicted in Fig. 8 demonstrate that the proposed buffer sharing method performs maximally 39% better than the one buffer method.

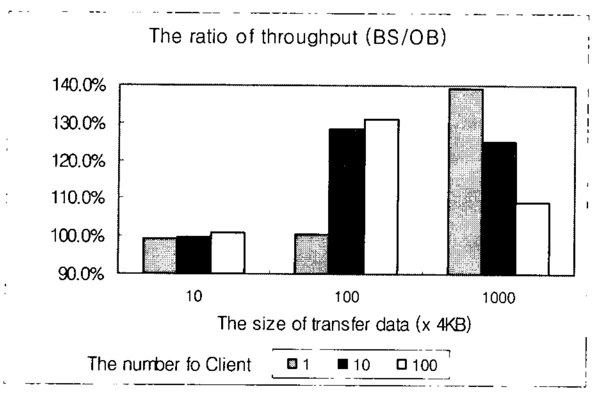


Fig. 8 The Ratio of Throughput: Evaluation results are independent of network environment. The Server and clients run at the same machines.

V. CONCLUSIONS

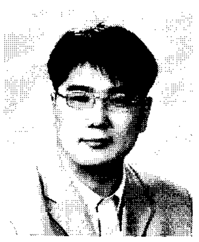
In this paper, we have proposed transfer buffer sharing method in order to solve the buffer conflict problem that arises on transferring massive data while using IOCP I/O model. Experimental tests show that buffer sharing method performs better than the previous method using one buffer per client. It is expected that this buffer sharing method would be applicable to develop the server with higher throughput. As future work, we should revise the cost model and perform the experimental tests with varying parameters, in order to decide the optimal size of buffer pool according to the number of client and the size of data.

ACKNOWLEDGMENT

This research was supported by a grant (08KLSGC05) from Cutting-edge Urban Development - Korean Land Spatialization Research Project funded by Ministry of Land, Transport and Maritime Affairs and by the Dongseo University research grant.

REFERENCES

[1] J. Richter and J. D. Clark, Programming Server-Side Application for Windows 2000, Microsoft Press, 2000
[2] Anthony Jone and Jim Ohlund, Network Programming for Microsoft Windows, Microsoft Press, pp.227-271, 1999
[3] Dinesh Ahuja, "IOCompletion Port Technique and Asynchoronos I/O Operartion," Available: http://www.codeproject.com/internet/IOCompletionPort.asp, 2005
[4] Amin Gholiha, "A simple IOCP Server/Client Class," Available: http://www.codeproject.com/internet/iocp_server_client.asp, 2006
[5] Anthony Jones and Amol Deshpande, "Windows Sockets 2.0: Write Scalable Winsock Apps Using Completion Ports," Available: http://msdn.microsoft.com/msdnmag/issues/1000/winsock/, 2005
[6] Len Holgate, "A reusable, high performance, socket server class - Part 1-6," Available: http://www.codeproject.com/internet/JBSocketServer1.asp, 2002
[7] Oz Ben Eliezer, "Writing scalable server applications using IOCP," Available: http://www.codeproject.com/internet/iocp.asp, 2001



Dae-Soo Cho
received the B.S., M.S., and Ph.D. degrees in the Computer Engineering from Pusan National University in 1995, 1997, 2001, respectively. He was on the technical staff in ETRI from 2001 to 2004. In 2004, he joined a faculty member of the Division of Computer and Information in Dongseo University. His research interests include GIS, spatio-temporal databases, LBS, and stream data processing.