Design and Performance Analysis of Windows-based Serial-Socket Bridge

Liu Xin, Wei Zhiqiang

College of Information Science and Engineering, Ocean University of China, No.238 Songling Road, Qingdao, China

liuxinouc@126.com, weizhiqiangouc@126.com

Wei Zhiqiang

Keywords: Serial-Socket Bridge, IOCP, CPU utilization, Serial-to-USB

Abstract: Serial port is an ancient hardware port but difficult to be replaced. There are still many special devices in the use of it. However, In the Microsoft Windows, the serial port programming is becoming increasingly difficult. This article not only describes the Serial-Socket Bridge software structure of the Windows operating system, but also analyses the relevant hardware and software structure, and pointed out the possible bottlenecks. The software will map the serial port operation to the socket port operation, to replace the Serial-to-RJ45 Converter hardware, and will not take up the valuable physical network port resources. We tested the processor usage on a variety of different baud rates on an Intel i5 processor's computer. We have found that high baud rate serial transmission (more than 115200) with a certain time interval is more efficient than continuous low baud transmission. In addition, the operating environment is also important for runtime reliability.

1. Introduction

In the field of server clustering and distributed computing, Socket is the most popular interface. Because of both Windows, Linux, iOS, their Socket interface is compatible, the use of Socket connection services can be easily configured in any kind of computer hardware. Many distributed software system interface is Socket form. However, there are some special hardware devices, such as ECG signal acquisition modules, multi-biological-parameter data acquisition modules, and other products, almost all of them have only one kind of interface form - the serial port. So how to transmit data between the Socket and the serial devices?

Serial port programming include the configuration of baud rate and work mode, stream encoder and decoder, multi-threaded programming, Overlapped IO and other relatively advanced software development technology. And debugging it requires some relevant hardware knowledge. So far, both in Windows and Linux, no matter what kind of programming language, there is no simple way to deal with serial communication. There is a special device called a Serial-to-RJ45 converter. This device is used to convert the Serial port to an Ethernet port. Its function is proportional to the price, the cheap module is very limited in function, and the high-performance ones is quite expensive. In addition, it will take up a physical network port. For most computers, there is only one network port, if the user in the use of this device at the same time needs to access the Internet, its needs to use a HUB to expand physical network ports.

An ideal solution is to run a Serial-to-Socket translation service on Windows. This service will take up some CPU resources and memory bus bandwidth but can provide flexible and efficient

Copyright © 2017, the Authors. Published by Atlantis Press.
This is an open access article under the CC BY-NC license (http://creativecommons.org/licenses/by-nc/4.0/).

Socket service interface[1-4]. More importantly, you can also do data preprocessing work directly in this service, thereby reducing the burden of data communications. Some researchers have already done this work, such as Zhang Hui-ming and his collaborators, they realizes remote control to the multi-media class-room equipment without network central control device[5-7]. The service software in this paper uses the Windows Completion Port, which is currently known as the most efficient Socket communication mechanism[8-12]. This paper put forward the hardware and software principles related to the Serial-Socket Bridge, and the working efficiency of the Bridge is discussed.

2. Block Diagram of the Serial-Socket Bridge

2.1. Relevant Hardware Diagram

More and more computers replace the serial port with USB. Today, most microcomputers have no serial port. Therefore, we expand the serial ports through USB-HUB and Serial-to-USB devices. In general, one USB-HOST interface can expand up to 127 independent serial ports. However, in actual use, we do not install more than 10 serial devices on one computer, because the USB-HUB power supply capacity is limited, connect too much Serial-to-USB will cause the USB system unstable. The whole Serial-Socket Bridge application environment was shown in Fig1.

In the serial communication system design, we must first assess the flux of data on the serial port. Some serial devices can support baud rates up to 921600. But in the actual equipment rarely use such a high baud rate. Serial data flux calculation equation shows below:

$$\text{EBPS} = \text{data bits flux} * \text{baud-rate} / (\text{START} + \text{DB} + \text{CHECK} + \text{STOP}) \tag{1}$$

EBPS : effective byte per second
START : start bits size
DB : data bits size
CHECK : odd-even-check bits size
STOP : stop bits size

When the serial communication format is baud rate 115200, 8 bit data bits, 1 stop bit, no parity, the number of bytes per second can be transmitted:

$$\text{EBPS} = 115200 / (1 + 8 + 0 + 1) = 11520 \text{ (bytes per second)} \tag{2}$$

Firstly, data is generated from the device and is converted to USB data stream via the Serial-to-USB. USB using differential signal lines, receive and send cannot be carried out at the same time, so they are half-duplex working. USB session using polling work mechanism, the USB-Host starts a polling instructions every 1ms, the USB-Devices receive the instructions and must response during the specified time. Serial port work in duplex mode, receive and send can be carried out at the same time. In general, in the lower computer, the serial port baud rate will not exceed 115200. That is 11.5K bytes per second data traffic. While USB1.1 (now known as USB 2.0 Full Speed) can achieve a theoretical rate of 1.2M bytes per second. In theory, the speed of USB is 100 times more than the serial speed. But in actual using the USB part will lose data. The reason is that USB design principles do not guarantee that each communication will be able to succeed. If the USB transfer error, then only wait until the next cycle (1ms later) will be retransmitted, at this time if the Serial-to-USB buffer overflow, it will cause an irreversible error. Serial to USB hardware was shown in Fig2.

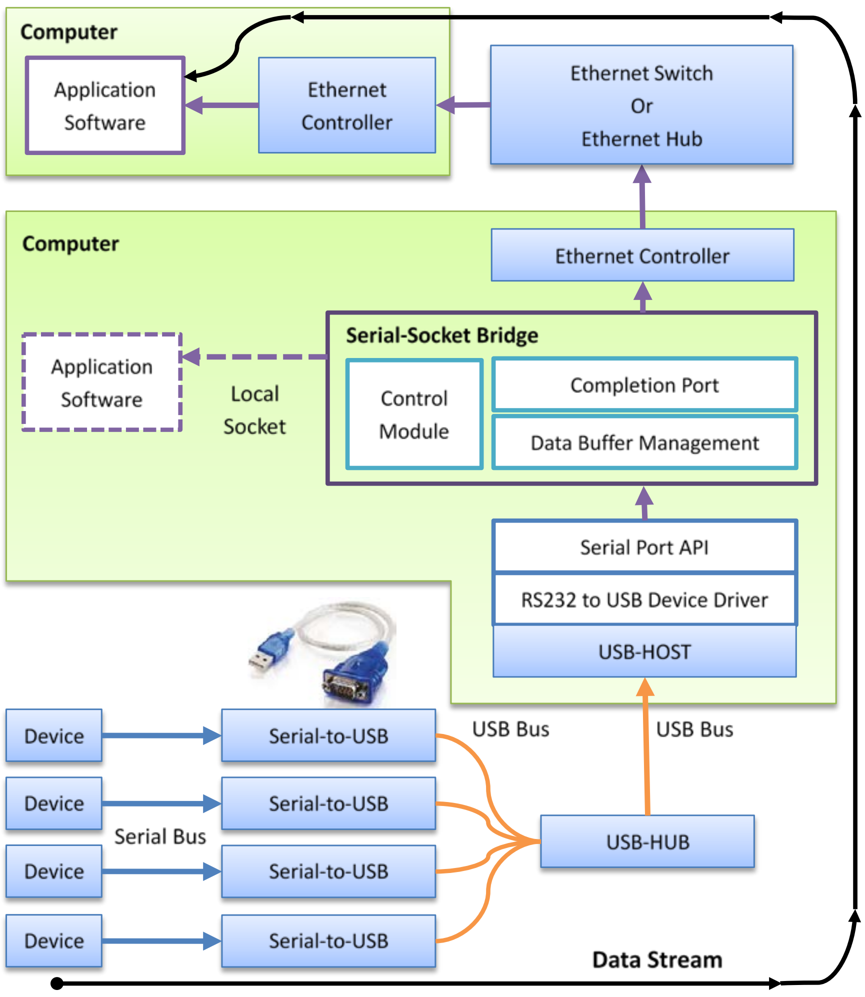


Figure 1 Serial-Socket Bridge and related hardware and software environment.

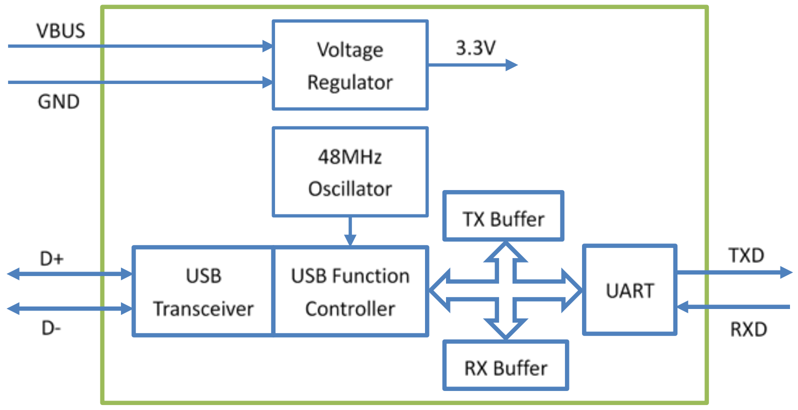


Figure 2 Hardware Principle of the Serial-to-USB.

Generally, the lower computer does not require the host computer instructions timely. If a downstream data error occurs, it would cause 1ms delay. This case will not cause serious impact. But when the lower computer sends data to the host computer if the host computer failed to read the RX buffer in time, the RX buffer would overflow, it will cause data loss. The error probability of

Serial-to-USB is closely related to the probability of the USB error occurrence and the RX buffer size, as shown in the equation below.

$$\text{BPMS (byte count per millisecond)} = \text{EBPS} / 1000 \tag{3}$$

$$P(\,data\ loss\,) = \begin{cases} 1, & BPMS > RX\ buffer\ size \\ x, & x = P(\,usb\ error) \text{ ^ } CEILING(\dfrac{BPMS}{RX\ buffer\ size}) \end{cases} \tag{4}$$

When the amount of serial data per millisecond exceeds the RX buffer size, the RX buffer is overflowed before the USB-Host starts the polling action. So RX buffer size must be larger than the 1ms data flow requirements, otherwise, shrunk the serial baud rate or adopt the intermittent communication mode. The larger the RX buffer, the more times to allow the USB communication to retry. In general, the hardware makes seldom errors, and the probability that a USB error immediately after another USB error is very low. So an RX buffer as large enough to accommodate 2ms serial data can work properly for a long time.

If the application is running on the local computer, access the data through the Local Socket, then the access speed will be very fast, almost with no delay. The Local Socket is just a logical conversion layer, it handles the data diagram header and the big-endian or little-endian problems.

If the application is running on another machine, then the data will be sent to the Ethernet Controller. Data stream pass through the Switch or HUB come to another computer's Ethernet Controller. First, we should ensure that the performance of the network is fast enough. This is usually not a problem, even if the speed of an average Ethernet Controller is more than 100 times faster than the serial communication. In a properly configured network system, the Switch or the Ethernet Controller hardware will not cause performance bottlenecks.

The Ethernet Controller is directly connected to the PCIE bus or AHB bus (Advanced High-performance Bus). PCIE or AHB as a parallel bus, the real-time response performance is far more than the USB bus. Although the RX FIFO and TX FIFO has only 256 bytes, but this is really enough. If the data transmission bottleneck is indeed caused by Ethernet, then the only solution is to upgrade the hardware, such as faster network cards and faster Switches.

2.2. Principle of Software

The core mechanism of the software consists of three types of collaborative threads, as shown in Fig3.

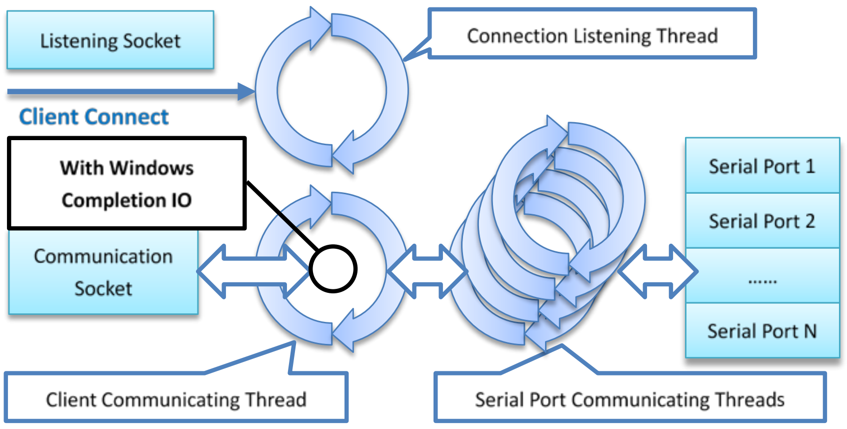

Listening Socket
Connection Listening Thread
Client Connect
With Windows Completion IO
Communication Socket
Serial Port 1
Serial Port 2
......
Serial Port N
Client Communicating Thread
Serial Port Communicating Threads

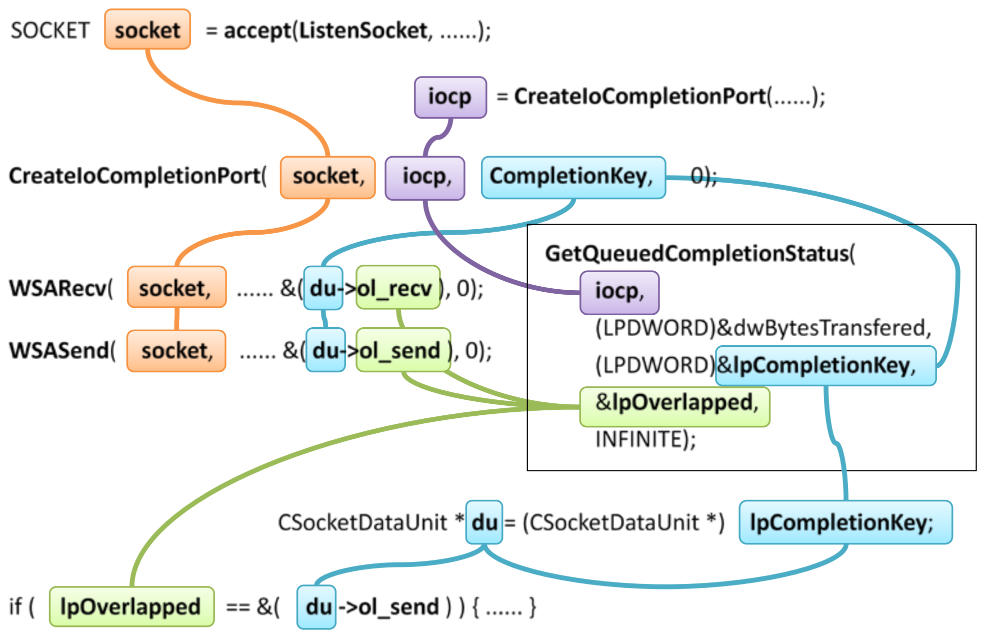
Figure 3 Key Structure of the Software

These three types of threads are responsible for listening to the Client connection, handling the Client communication and handling the Serial Port communication. When the program starts, it only creates one thread which is responsible for listening Client connection. When the Client connects to the server, the Windows System creates a new Socket to communicate with the Client, and then we start a new thread to handle the communication processing of the new Socket. The program will open the pointed serial port based on the requirements of the Client. One Serial Port Communicating Thread is created as the serial port opening. The system needs to prepare one Serial Port Communicating Thread for each opened serial port. When Client Communication Thread receives a request from the Client, it forwards the request to the corresponding port's Communicating thread. When the serial data returns, it is transmitted in the opposite direction.

Mentioned in Windows C/S programming, a perfect solution is the I/O Completion Port (IOCP). IOCP will make full use of the Windows kernel for I/O scheduling. It is the best network communication model for C/S communication works. The traditional solution, such as the thread pool, for each client connection, is assigned one processing thread. With many threads, the thread context switching will take a lot of processor time. Microsoft made the IOCP is to solve this "one-thread-per-client" shortcomings, it makes full use of kernel object scheduling, using only a small number of threads to deal with all the client communication, eliminating the unnecessary thread context switch.

In short, we can understand the IOCP as an agent. When we appoint a "receive" or "send" task to the IOCP, we can immediately return and to do other things. When the "receive" or "send" task is completed, the agent will give us a notice. Then we can do the same thing with the next task.

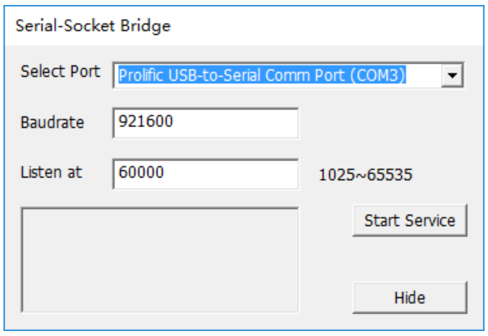
The use of the Windows IOCP is not complicated, but the relevant code debugging is really difficult. Which involves several confusing type casts, as well as the function name and the function-deed does not matter. Perhaps Microsoft has to do so because Windows must maintain some sort of forward compatibility. The internal relations of the key functions are shown in Fig4.



```
SOCKET socket = accept(ListenSocket, ......);

iocp = CreateIoCompletionPort(......);

CreateIoCompletionPort( socket, iocp, CompletionKey, 0);

WSARecv( socket, ...... &(du->ol_recv), 0);

WSASend( socket, ...... &(du->ol_send), 0);

GetQueuedCompletionStatus(
    iocp,
    (LPDWORD)&dwBytesTransfered,
    (LPDWORD)&lpCompletionKey,
    &lpOverlapped,
    INFINITE);

CSocketDataUnit * du = (CSocketDataUnit *) lpCompletionKey;

if ( lpOverlapped == &( du->ol_send ) ) { ...... }
```

Figure 4 Windows IOCP Related Functions and Their Intrinsic Links.

Through the above steps, we establish the key parts of the Serial-Socket Bridge. This part plays a decisive role in the software's performance and reliability. Finally, we created a simple user’s interface for the software, as shown in Fig5. It can be minimized to an icon on the system tray.


Serial-Socket Bridge

Select Port: Prolific USB-to-Serial Comm Port (COM3)

Baudrate: 921600

Listen at: 60000 1025~65535

Figure 5 User Interface.

3. Performance Evaluation and Conclusion

We tested the Serial-Socket Bridge service on a laptop computer with an Intel i5-3317U processor and 4G memory. As the computer has no serial port, we use a PL2303 converter to expand a serial port. PL2303 compatible with USB2.0 communication protocol, with 256 bytes send buffer and 256 bytes receive buffer.

We use a single-chip system to produce the source data, we set its serial port to a fixed baud rate and send data continuously. The receiving-end is a TCP Client software. We use the Windows built-in Task Manager to see the processor usage. In the case of different serial port baud rate, CPU utilization as shown in Table 1.

Table 1 Baud Rate and CPU Usage Rate.

No.	Baud Rate	CPU Usage Rate
1	9600	7%
2	14400	9%
3	19200	11%
4	38400	21%
5	57600	30%
6	115200	53%
7	230400	69%

As can be seen from Table 1 when the baud rate is lower than 115200, the processor utilization and baud rate is a basically linear relationship between. When the baud rate increased to 230400, the processor utilization rate of increase has slowed down. We tested the 921600 baud rate with a processor utilization of 72%, only a little bit higher than the 230400 baud rate.

By checking the datagram, we found that baud rate is less than or equal to 115200, no error occurred. The data loss will occur in the situation with baud rate more than 230400. Why does the data loss occur in only about 70% processor utilization? We speculate that serial IO is the bottleneck. The PL2303 converter has only 256 bytes of the receive buffer. When the serial port works on 921600 baud rate, passing one byte (1 start bit, 8 data bits, no parity, and 1 stop bit) only takes 0.01 milliseconds. Fill 256 bytes buffer need only 2.8 milliseconds. This time is greater than the USB device rotation gap (1 millisecond). As long as only one error occurred in the USB communication process, it is enough to exceed this time. On this occasion, although the processor has spare processing power, but the USB link has been fully loaded.

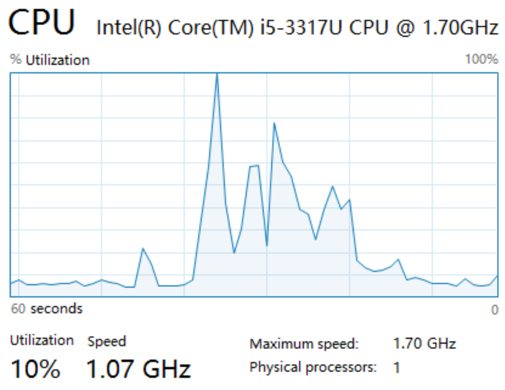
This loss of data can also be eliminated if gaps are inserted between the datagrams. At 230400 baud rate, we set the length of the datagram to 22 bytes (0.96ms) and insert a 0.2ms interval between datagrams after that the data loss has never happened. In this case, the data transfer efficiency reached the performance of 170% to 180% of 115200 baud rate. Therefore, the use of high baud rate with intervals can improve the serial transmission efficiency. Then we gradually increase the length of the interval and test the processor utilization. The test results are shown in Table 2.

Table 2 Datagram Interval and CPU Usage Rate.

No.	Interval	CPU Usage Rate
1	0.2ms	66%
2	0.4ms	61%
3	0.6ms	56%
4	0.8ms	46%
5	1ms	35%
6	2ms	25%
7	3ms	18%
8	4ms	13%

As can be seen from the Table 2, there is a turning point in processor utilization when the datagram interval exceeds 1 millisecond. Since the data has no lost, the burden of the USB process should be within the normal range. We believe that this phenomenon reflects the actual efficiency of the CPU-memory subsystem. In the case of the datagram interval greater than 1 millisecond, the CPU-memory subsystem can be mitigated and more processor time can be shared with other programs.

In addition, in the test, we noticed that the anti-virus software made a significant impact. Processor utilization suddenly rose as shown in Fig6. When we turn off anti-virus software this phenomenon does not appear again.



Utilization 10%
Speed 1.07 GHz
Maximum speed: 1.70 GHz
Physical processors: 1

Figure 6 The Influence of Anti-Virus Software on the System.

Finally, we also tested the Socket Client on the impact of processor utilization. In the serial baud rate of 115,200 continuous transmission case, Socket communication burden is about 5% constantly, never exceed 10%. This shows that the Socket communication can easily cope with the dataflow transfer in this degrees.

4. Conclusion

According to the above test and analysis, we make the following recommendations:

- 1. Turn off anti-virus software and other interfere functions.
- 2. If the serial communication baud rate reaches 115200, try to limit only one serial device connection. If you need to connect more than one serial devices, you should consider reducing the serial baud rate or inserting intervals into the datagram.

- 3. If the serial communication baud rate exceeds 115200, you must insert intervals into the datagram. The size of each datagram package should be smaller than the physical buffer size of the Serial-to-USB. And the interval between two datagram packages should be more than 1 millisecond.
- 4. The use of the advanced Serial-to-USB Adapter may be able to further enhance the serial port communication performance.

Acknowledgment

This work is supported by Major Projects of Shandong (No.2015ZDZX05002), Key Research and Development Project of Shandong (No.2016ZDJS09A01) and Aoshan Science and Technology Innovation Program of QNLM (2016ASK07).

References

[1] Zhu, Q. S. (2012) Adaptor between Ethernet and serial ports based on uC/OS and LwIP, In Advanced Materials Research, 433, 4038-4041.

[2] Wallmark, O., L. Jin, and S. Norrga (2016) Serial Communication Based Distributed Control of the Stacked Polyphase Bridges Converter. In 8th IET International Conference on Power Electronics, Machines and Drive, 1-5.

[3] Wen, Nai Ning, S. F. Gong, and Y. J. Wang (2013) The Implementation of Serial Communication CSerial the Port Class, Proceedings of the 2012 2nd International Conference on Computer and Information Application, 169-171.

[4] Wang Qi, and Wang Dongjie (2015) Design of Serial Interface/Ethernet Converter Based on TCP/IP, Industrial Control Computer, 11, 8-9.

[5] ZHANG Hui-ming, LEI Zhi-hua (2009) Design of Communication Software between Network Interface and Serial Interface Based on Windows Service. Microcomputer Information, 25.12, 114-115.

[6] Wei, Yi Hu, and L. Chen (2015) Serial port communication based on LabVIEW-VISA, Electronic Design Engineering, 23(24), 129-131.

[7] Wang, Bin, Y. Jiang, and W. Xu (2011) Design of Module of Network Communication Applied in PLC Device Management, Microcomputer Applications, 27(2), 21-27.

[8] Sinha, Sharad, and T. Srikanthan (2014) IP-enabled C/C++ based high level synthesis: a step towards better designer productivity and design performance, International Journal of Reconfigurable Computing, 2014, 1-17.

[9] Saito, Takeshi, et al (2016) Significance of imaging modalities for preoperative evaluation of the pancreaticobiliary system in surgery for pediatric choledochal cyst, J Hepatobiliary Pancreat Sci, 23(6), 347-352.

[10] Yu, Y. E., and F. U. Yu (2011) IOCP Technology-based Server Program Design, Computer Knowledge and Technology, 7(12), 2844-2845.

[11] Liu, Zhenyang (2013) Software Design Based on High-powered Server of IOCP, Computer and Digital Engineering, 41(7), 1154-1156.

[12] Cheng, Song Tao, and X. X. Liu (2012) Design and Implementation of Server Model Based on IOCP, Computer Programming Skills and Maintenance, 20, 74-76.