An Inter-subsystem Data Transfer Mechanism Based on A New Computer Architecture with Single CPU and Dual Bus

Wang Wei
College of Information Engineering
Qingdao University, QDU
Qingdao, China
pattonwei@126.com

Shao Fengjing
College of Information Engineering
Qingdao University, QDU
Qingdao, China

***Abstract*—Focusing on the architecture characteristics of the new computer architecture with high-security sCPU-dBUS, this paper designs and implements an Inter-subsystem data transfer mechanism based on the new high-security operating system with internal networking structure netOS-I which is only used by the sCPU-dBUS. The data transfer mechanism mainly contains Inter-subsystem data transfer interface, Inter-subsystem data transfer protocol and operation and management of transit cache memory.**

Keywords-computer architecture; operating system; data transfer; private protocol; memory management

I. INTRODUCTION

With the development of computer application, especially the comprehensive development of informationization on network, the network security is becoming more and more important. A series of network security technologies have been used to protect the computer security, such as computer virus scan technology, intrusion detection technology [1,2], software or hardware encryption technology [3,4], etc.

The new computer architecture sCPU-dBUS (single CPU and dual bus security architecture [5]) provides a method that takes advantage of computer architecture to protect computer security. The sCPU-dBUS has a single CPU and two independent high-speed system buses (local bus and network bus). All the network devices and other devices mounted on the network bus form into a network subsystem which connects with the Internet. All the storage devices and other devices mounted on the local bus form into a local subsystem which is isolated from the Internet. A bus bridge [6] of the sCPU-dBUS controls the connection between CPU and the two high-speed system buses to make sure that the CPU only connect with one bus at any time. Therefore, the network intrusions can only damage the network subsystem, but can't enter the local subsystem where user stores data.

As a result of the particularity of the sCPU-dBUS, a special operating system named netOS-I is applied according to an Embedded System Developing Platform Hardware Abstraction Layer [7]. The netOS-I is a high-security operating system with internal networking structure. It uses dual-kernel mechanism which contains local kernel and network kernel to manage two subsystems, and uses dual-kernel switch mechanism to resolve the problem of sharing a single CPU between two subsystems. The relationship between the netOS-I and the sCPU-dBUS is shown in Figure 1.

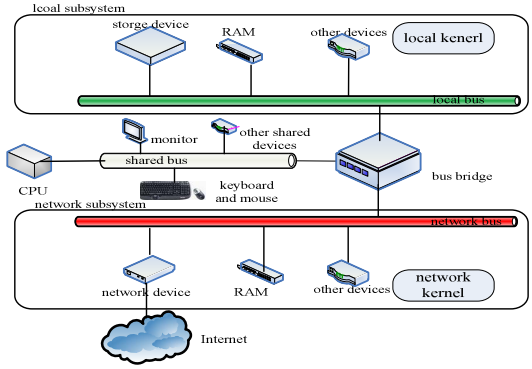

Figure 1. The netOS-I based on sCPU-dBUS

The sCPU-dBUS combined with the netOS-I can effectively prevent viruses, trojans, and other network intrusions from invading local subsystem, thus user's data is effectively protected from being stolen or damaged. But at the same time, it also prevents the data that user gets from Internet in the network subsystem from entering the local subsystem, and the data that user wants to upload to Internet in local subsystem from entering the network subsystem. Currently there are many secure communication technologies in bus systems [8], but none of them can be used in the sCPU-dBUS. In order to communicate between two subsystems securely, this paper designs and implements an inter-subsystem data transfer mechanism. With the data transfer mechanism, data can be transferred between the subsystems and the network intrusion can still be isolated from the local subsystem.

II. THE BASIC THEORY OF THE INTER-SUBSYSTEM DATA TRANSFER MECHANISM

The two subsystems of the sCPU-dBUS are managed respectively by two kernels, that is, network kernel manages network subsystem and local kernel manages local subsystem. In each subsystem, the kernel has a data transfer daemon to handle

978-0-7695-3881-5/09 $26.00 © 2009 IEEE
DOI 10.1109/WCSE.2009.742

the inter-subsystem data transfer. The data transfer daemon is the data transfer interface between the subsystems.

The data transfer daemon communicates with the other processes in the same subsystem via single. The two data transfer daemons in different subsystems communicate with each other via message. Because only one subsystem can use the CPU resource at any time, i.e., only one kernel is active at the same time, the two data transfer daemons can't directly communicate with each other via message. Therefore, they must use the transit cache mounted on the shared bus of the sCPU-dBUS to transfer messages indirectly. The process of inter-subsystem data transfer is shown in Figure 2.

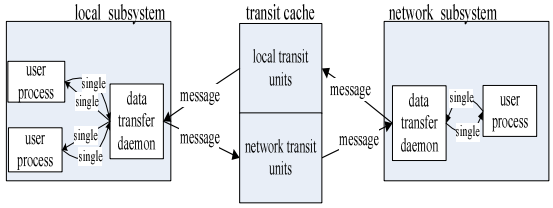

Figure 2. The process of inter-subsystem data transfer

III. INTER-SUBSYSTEM DATA TRANSFER PROTOCOL

In order to prevent the network intrusions from entering local subsystem from network subsystem via transit cache, a private protocol named inter-subsystem data transfer protocol is used to control the inter-subsystem data transfer. Because this protocol is a connection-oriented protocol, the inter-subsystem data transfer process is divided into three phases: creating connection, transferring data, releasing connection [9].

A. Creating connection

In a data transfer connection, the data transfer daemon whose responsibility is to send data is called sender daemon, and the one whose responsibility is to receive data is called receiver daemon. For creating a data transfer connection, the sender daemon will check the source address of data transfer and then send the connection request message. In the message head, the synchronization bit SYN is 1 and the message sequence SEQ is 0. In the figure 3, "SYN=1,SEQ=0" means the above description, and the other 0 bits in the message head aren't shown.

After receiving connection request message, the receiver daemon will check if the destination address exists in its subsystem. If it exists, the receiver daemon sends connection success confirmation message in which synchronization bit SYN is 1, acknowledgement bit ACK is 1 and message sequence SEQ is 1. If it doesn't exist, the receiver daemon sends connection failure message in which synchronization bit SYN is 1, acknowledgement bit ACK is 1 and message sequence SEQ is 0. The process of creating connection successfully is shown in Figure 3.

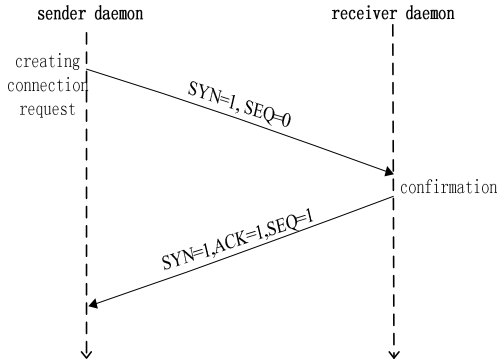

Figure 3. The process of creating connection successfully

B. Transfer data

If the connection is created successfully, the data transfer process will enter the phase of transferring data. In this phase, sender daemon sends data message. The receiver daemon handles data message and send back confirmation message. In the message head, the data bit DAT is 1 and message sequence SEN is x. In the confirmation message head, the data bit DAT is 1, acknowledgement bit is 1 and the message sequence SEN is x+1. The process of transferring data is shown in Figure 4.

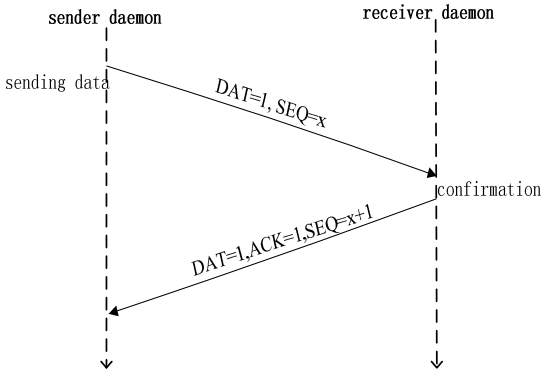

Figure 4. The process of transferring data

C. Releasing connection

After all the data have been transferred, the sender daemon will send release connection message. If the receiver daemon confirms that all the data has been accepted, it will send confirmation message back to agree with releasing connection. In the release connection message, end bit END is 1, message sequence is y. in the confirmation message, the end bit END is 1 and the message sequence is y+1. The process of releasing connection is shown in Figure 5.

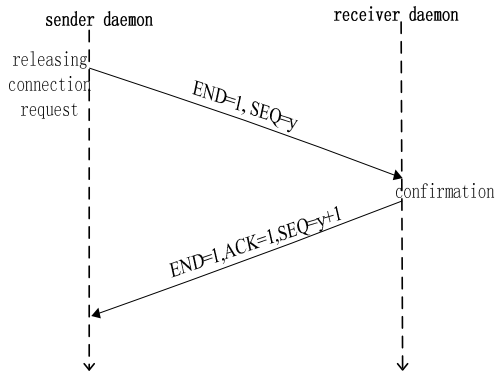

Figure 5. The process of releasing connection

IV. TRANSIT CACHE MEMORY

The transit cache mounted on the shared bus is managed and operated by the two data transfer daemons. And it plays a role of bridge in the communication between the data transfer daemons which must transfer data indirectly through the transit cache.

A. Transit cache structure

The transit cache is divided into three parts: transit cache state block, transit head area and transit block area. In the process of transit cache initialization, transit heads are set from the end of transit cache state block which is in the low-end of cache area and their corresponding transit blocks are divided in the high-end of cache area at the same time. All the transit blocks form the transit block area, while all the transit heads form the transit block area. Every transit head links its corresponding transit block to form a transit unit. There are two kinds of transit units: network transit unit which is made up of a network transit head and a network transit block, and local transit unit which is made up of a local transit head and a local transit block. The transit cache structure is shown in Figure 6.

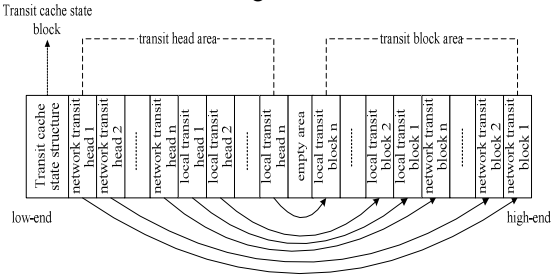

Figure 6. transit cache structure

The transit head area is made up of network transit heads and local transit heads. The two kinds of heads have the same structure that is shown in the table 1. The transit head is used to descript attributes of the transit unit, store the message head and link all the same type of transit units to be a doubly linked circular list. All the network transit units are linked to be a network doubly linked circular list and all the local transit heads are linked to be a local doubly linked circular list.

TABLE I. TABLE 1 TRANSIT HEAD STRUCTURE

field name	field introduction
trans_idle	idle marker, it indicates if the transit unit is used.
trans_num	transit unit number, it is the transit unit ID.
mes_head	message head, it is used to store the message head.
trans_prev	transit head forward pointer, it points to the transit head before itself.
trans_next	transit head backward pointer, it points to the transit head after itself.
trans_block	transit block pointer, it points to the corresponding block pointer.

The transit block area only contains two kinds of blocks: network transit block and local transit block. All the blocks are linked to corresponding transit heads. The function of transit block is to store the data of message.

The transit cache state block contains a transit cache state structure which mainly has three fields: network linked list head pointer, local linked list head pointer and residual connection count. The network linked list head pointer points to the first idle transit unit of the network doubly linked circular list. It is used when network data transfer daemon receives message or local data transfer daemon sends message. The local linked list head pointer points to the first idle transit unit of the local doubly linked circular list. It is used when local data transfer daemon receives message or network data transfer daemon send message. The residual connection count indicates that how many data transfer connections the transit cache can still support to create. It is used for the data transfer daemon to create a data transfer connection.

B. The operation and management of transit cache

There are two kinds of transit units in a doubly linked circular list of transit units (both the network doubly linked circular list and the local doubly linked circular list): idle transit unit and non-idle transit unit. In the beginning, every transit unit in the list is the idle transit unit. When an idle transit unit is needed, it will be obtained from the head of the list, become a non-idle transit unit, and then be inserted into the tail of the list. Therefore, the idle transit units are in the front of the list while the non-idle transit units are in the end of the list. The linked list head pointer (both the network list head pointer and the local list head pointer) in the transit state structure points to the first idle transit head of the list. The doubly linked circular list of transit units is shown in the figure 7:

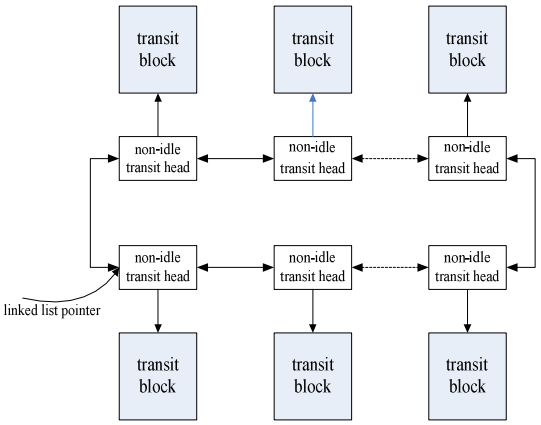

Figure 7. The doubly linked circular list of transit units

A data transfer connection has a pair of transit unites (a network transit unit and a local transit unit) which have the same ID to transfer message. The network transit unit is used by the network data transfer daemon to receive the message that the local data transfer daemon sends. Similarly, the local transit unit is used by the local data transfer daemon to receive the message that the network data transfer daemon sends. These two units are released after the connection is released. Residual connection count in the transit state structure indicates how many pairs of idle units still can be used.

Before creating a connection, data transfer daemon first check the residual connection count to find if there is a pair of idle units which can be used for the new data transfer connection. If there is, it will find an idle network transit unit by the network transit head list pointer, modify the idle marker of the unit to make it turn into a non-idle unit, and then insert it into the end of the list. After that the data transfer daemon traverses the local doubly linked circular list to find the idle local transit unit whose number is same to the network transit unit's, and then modify the it's idle marker and insert it into the end of the list. All the messages of the connection are transferred via these two transit units.

If the data transfer daemon needs to send a message in a connection, it should traverse the doubly linked circular list of target subsystem from the end to find the transit unit that belongs to this connection, and then put the message into the unit. When the data transfer daemon wants to receive message in a connection, it will traverse the doubly linked circular list of itself subsystem from the end to find the transit unit which belongs to the connection, and then get the message from the unit.

After data transfer daemon releases a connection, it restores the idle marker of the pairs of the transit units that belong to the connection, and inserts them into the head of their doubly linked circular list to release them.

V. CONCLUSION

In the inter-subsystem data transfer mechanism, the data transfer protocol ensures that the data transfer mechanism can't be used by network intrusion, and the way to operate and manage transit cache memory can make many connections transfer data between subsystems at the same time. Therefore with the inter-subsystem data transfer mechanism, the local subsystem and the network subsystem can communicate with each other safely and efficiently.

ACKNOWLEDGMENT

Many people have made invaluable contributions, both directly and indirectly to my research. I would like to express my warmest gratitude to Sun Rencheng, for his instructive suggestions and valuable comments on the writing of this thesis. Without his invaluable help and generous encouragement, the present thesis would not have been accomplished.

REFERENCES

[1] Abdoul Karim Ganame, Julien Bourgeois, Renaud Bidou and Francois Spies, "A global security architecture for intrusion detection on computer networks," Computer & Security, Vol. 27, pp. 30-47, March 2008.

[2] Ben Rexworthy, "Intrusion detections systems – an outmoded network protection model," Network Security, Vol. 2009, pp. 17-19, June 2009.

[3] Niansheng Liu, Donghui Guo, Security analysis of Public-key Encryption Scheme Based on Neural Networks and Its Implementing. International Conference on Computational Intelligence and Security, pp. 1327-1330, 2006.

[4] Prayag Narulaa, Sanjay Kumar Dhurandhera, Sudip Misrab and Isaac Woungangc, "Security in mobile ad-hoc networks using soft encryption and trust-based multi-path routing," Computer Communications, Vol. 31, pp. 760-769, March 2008.

[5] Fengjing Shao, Rencheng Sun, Kegang Diao, Xiaopeng Wang, "Hardware encryption technology complies with encryption regulations," In Proc. Fifth IEEE International Symposium on Embedded Computing, Beijing, China, Oct, 2008.

[6] Tiedong Wang, Fengjing Shao, Rencheng Sun, He Huang. "A hardware implement of bus bridge based on single CPU and dual bus architecture," International Symposium on Computer Science and Computational Technology, Shanghai, China, December 2008.

[7] Le Zhang, Fengjing Shao, Rencheng Sun, "Design and realization of embedded system developing platform hardware abstraction layer," Journal of Qingdao University Engineering & Technology Edition, Vol.21, pp. 20-25, Mar, 2007.

[8] Sascha Mühlbach, Sebastian Wallner. "Secure communication in microcomputer bus systems for embedded devices," Journal of Systems Architecture, Vol. 54, pp. 1065-1076, November 2008.

[9] W.Richard Stevens. TCP/IP Illustrated Volume 1: The protocols. US: Addison-Wesley Professional, 1993.