ГОДИШНИК НА СОФИЙСКИЯ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ“

ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА
Книга 2 — Приложна математика и информатика
Том 89, 1995

ANNUAIRE DE L'UNIVERSITE DE SOFIA „ST. KLIMENT OHRIDSKI“

FACULTE DE MATHEMATIQUES ET INFORMATIQUE
Livre 2 — Mathématiques Appliquée et Informatique
Tome 89, 1995

COOPERATION OF CLIENT ROUTINES IN CLIENT-SERVER NETWORK ARCHITECTURE WITHOUT USING OF SPECIAL MONITOR ROUTINE ON SERVER

PETER DIMOV

A method for communication between client routines without monitoring by a special routine, working on a server, is proposed. The base of the method is a message transferring engine, but not in the classical form. The method is oriented to data sharing between computers in a client-server architecture network. All data are stored on a PC disk space and there is no need to store data on the server. The server is used only for files lock and unlock purposes and its disk capacity is used only as an intermediate storage. The method ensures higher information security level than the traditionally used methods. Communication between computers allows to develop applications for cooperative work and documents routing. In a more global aspect the described engine is applicable in both single and multiprogram environments.

Keywords: message transferring engine, network architecture, communicating routines.
1991/95 Mathematics Subject Classification: 94-99.

1. INTRODUCTION

Let us use the term "*server*" to denote the main node of the network, responsible for sharing resources between users. Let us use the term "*client*" to denote workstations on the computer network. Then we would use the terms "*client routine*" and "*server routine*" to denote routines, working on user workstations and server platform. In this way we will describe the typical features of the client-server architecture computer network.

"*Monitor*" denotes a user-written routine, which must take control over user routines (client routines) and is used on client-server architecture network. The method proposed does not use any monitor.

The method requires the server routine to be able to:

— share resources — lock/unlock files or part of their contents;

— store temporary data on server disk storage as intermediate storage.

The operations of sharing the resources are non-interruptible and the server routine cannot affect logically the cross-user communications. Then we will have to discuss only client-routine relationships.

To describe the method, we will use the term "*protocol*" to denote the sequential *actions*, undertaken by the client routines, and the corresponding *states* the client routines may reach during the process of message transfer.

The method serves the following "*protocol*" (the states are shown after the description of every action):

(i) the "*sender*" sends a message to the "*receiver*" ("W");

(ii) the "*receiver*" accepts the *message* sent by the "*sender*", takes some actions and sends an *answer* back to the "*sender*" ("C");

(iii) the "*sender*" accepts the *answer* sent by the "*receiver*" ("M").

It is possible anyone of the two communicating user routines (each of both users) to take the role of the "sender" and consequently the other user routine must take the role of the "receiver".

Because the *answer* does not have only the role of an *acknowledgment*, this protocol is provided to realize the interchange of a wide range of information — every transferred portion of information (message) can have the form of a *request*, *query*, *command*. The type of the message depends on its form. The answer from the receiver can be a *request*, *query*, *command* as a nearer result or a complete document.

The protocol described above facilitates the transfer of any message from the side of the sender and the reply on the side of the receiver after some processing (if needed). Because each of the communicating user routines can take the role of the sender, the protocol is the base for creating channels between the user routines in both the simple and the duplex modes.

2. A METHOD FOR MESSAGE INTERCHANGE

To transfer messages between two users (client routines) it is necessary to:

— establish a connection between the client routines;

— provide resources for the message transfer and sharing.

We will use, whenever it is possible, the term "*user*" instead of "*client routine*", because users communicate through client routines. Every user must identify itself by a user identifier and must place a request for communication in the form of CSB (Communication Sign-on Block), labelled by the same user identifier. Every record in this block must point to the identifier of the other user, with which the CSB owner wants to communicate. Each client routine must also check the existence of a CSB for each user, marked by a record in its own CSB. A connection between two client routines is considered established when:

(i) CSBs labelled by two user identifiers exist at the same time;

(ii) a record of one CSB points to the corresponding CSB and vice versa.

When (ii) is not satisfied, it means that the CSB points to different (maybe already connected) users.

All resources may be realized as files. In this case the network server must only lock and unlock files (or parts of the files' content) in response to client routines requests.

To provide synchronization between two communicating client routines, a "*post-box*" must be created. In this "post-box", realized as a file too, the information must be recorded as follows:

(i) field "*State*" contains the code of the current state, which the client routines can reach ("W", "C" or "M");

(ii) field "*Identifier*" must point to the identifier of the user, that must receive a message or an answer;

(iii) field "*Message/answer*".

The states shown above are common of the two communicating client routines. It is enough to store only the codes "W", "C" and "M" about the three states, which the user routines can reach. Field "Identifier" points (except when field "State" contains "W") to the user (the client routine), which *must be activated* by the message/answer transfer initiator to receive a message or an answer. To cause changes in the other client-routine state (the client routine to be activated to receive a message or an answer), the client routine, which wants to send a message/answer must:

(i) store into the field "State" a new state code;

(ii) store into the field "Identifier" the user identifier of the corresponding client routine to be activated.

Each client routine must check the "post-box" to determine the state (the states are common of both client routines, as described above).

All read/write operations with a "post-box" information are possible only after the post-box file has been locked. When one of the client routines locks the "post-box" file, the other client routine at this time must wait (or take actions, which must not affect the "post-box" content) until the "post-box" can be locked again. When one of the communicating client routines cannot find its own user identifier stored in the field "Identifier", that routine must immediately unlock the file (except when the file "State" contains the code "W" and the routine wants to send a message). This is possible when there are differences in the speed of the two client routines' execution. In this case immediate unlocking resolves the problem and makes the method independent of the relative speed of the communicating routines' execution.

Now we can describe the protocol used as follows:

(i) the "sender" writes a message to the "receiver" into the field "Message/answer", stores the identifier of the "receiver" in the field "Identifier" and changes the state to "C";

(ii) "receiver" accepts the message sent by the "sender", takes some actions, writes an answer in the field "Message/answer" back to the "sender", stores the identifier of the "sender" in the field "Identifier" and changes the state to "M";

(iii) the "sender" accepts the answer sent by the "receiver" and changes the state to "W".

A client routine must create resources (excluding CSB), e.g. a client routine, executing on the computer of the user with a higher identifier (in lexicographical order). On disconnection, resources must be released in the order of:

(i) disconnection-initiator acting as a "sender" sends the message "quit" to the other client routine (the "receiver");

(ii) the "receiver" releases the corresponding record in its own CSB and returns an answer back to the "sender";

(iii) the "sender" accepts the answer sent by the "receiver" and makes sure that the "receiver" will no more use the shared resources, then the latter releases, in his own CSB, the corresponding record.

Every user (client) routine control flow can be proposed as a graph. The nodes correspond to the states a routine can involve. The arrows are used for signals, which cause changes in the routine state. Fig. 1 displays a graph of two client routines presented by their states and corresponding signals according to

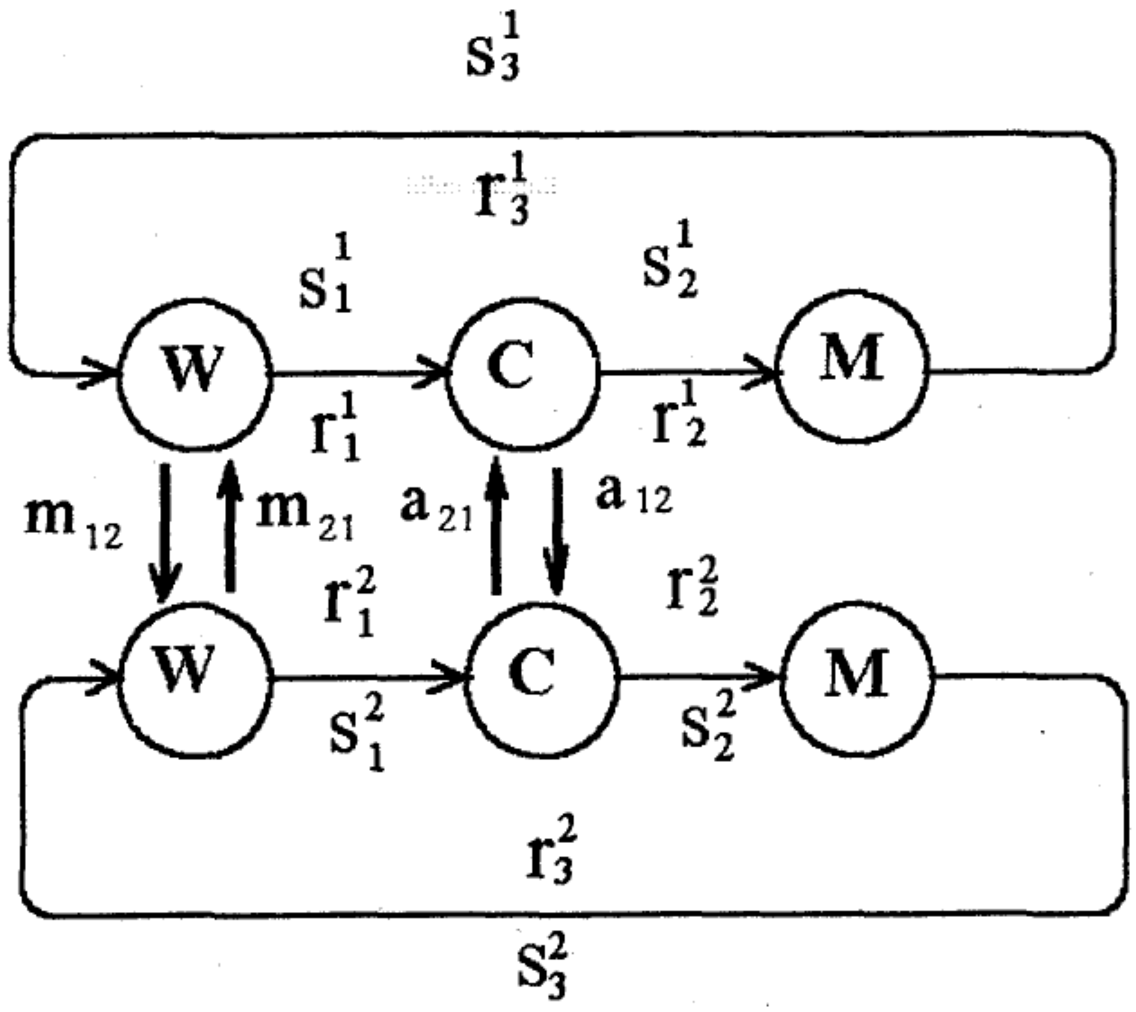

Fig. 1. Graph of communicating user routines

the described method. States "W", "C" and "M" are common to both routines (processes).

Denotifications:

s_1^i: the "sender" sends a message to the "receiver" and changes the state to "C";

s_2^i: the "receiver" changes the state to "M";

s_3^i: the "sender" changes the state to "W";

r_1^j: the "sender" changes the state to "C";

r_2^j: the "receiver" sends an answer to the "sender" and changes the state to "M";

r_3^j: the "sender" changes the state to "W";

m_{ij}: a message from the sender;

a_{ji} : an answer from the "receiver".

3. APPLICATION OF THE METHOD

The method is applied by the author for work with MSDOS — for providing an environment as a remote service facility in the client-server architecture network. To execute MSDOS — a command or a file on a *remote* computer, the command must be transferred to the other client routine, working on the *remote* computer, as a message. To return the messages, produced at the execution time, a file for the messages is used as a temporary disk space on the server. The steps are listed below:

— sending the command, entered by the user, to the client routine, working on the remote computer;

— receiving the message by the remote client routine, executing a command on the remote computer, recording the messages produced in the file for messages and returning an answer to the command initiator via the "post-box";

— receiving the answer, sent by the remote client routine, and typing on the screen the messages, stored into the file for messages.

File-transfer commands are realized as multipartitioned commands, which run on both computers sequentially. Synchronization is provided by the message-transfer method, described above. The steps are listed below:

— To get files from the remote PC disk space:

- in state "W" the command is to be sent to the remote client routine (remote computer);
- in state "C" the remote client routine copies the files into a temporary directory on the server and transfers the answer to the file-transfer initiator;
- in state "M" the initiator receives the answer, displays the messages from the file for messages, and copies the files from the temporary directory on the server into his own disk space.

— To put files into the remote computer disk space:

- in state "W" the initiator copies the files into a temporary directory on the server and transfers the command to the remote client routine;
- in state "C" the remote client routine copies the files from the temporary directory on the server into his own disk space, then sends an answer to the file-transfer initiator;
- in state "M" the initiator receives an answer from *the remote client routine* and displays the messages from the file for messages.

It is possible for each user to take the initiative for a command execution or file transfer.

4. ADVANTAGES OF THE METHOD

Data security. There is no need to store data for permanent use on the server disk space. Documents can be stored partially on the user's disk space. Each side can access only the part of the information, provided for his (her) own use. High security level is reached when the messages, transferred between the users (the user routines) are encoded. The resource identifiers can be generated as words and/or numbers and they are accessible in the communicating routines at the time of the connection established by them and put in the corresponding GSB records.

Unauthorized access prevention. The user access is protected by a password. Passwords are not registered in the file of common access. The user can change his (her) password. The user must provide secured access to his (her) own data.

Groupwork. The method can be used for transferring data between people, working in a group. In this case the method provides fast access to the information, supported by each of the members of the group.

Coordination. The method provides coordination both between the client routines and the people working in the group.

Documents routing. Documents routing depends on the needs of the organization.

Electronic mail. It is easy to send a message or a document to any group member.

Establishing connections. It is possible to establish connections between every two users. In this way every user can communicate at every time with somebody else. The method provides opportunities for simultaneous connections between the users.

Server machinery requirements. The server can have enough disk space installed, but the latter can be smaller than the space used when the data is stored entirely on the server. There are no special requirements to the server processors.

Workstations requirements. Workstations have usually enough disk space installed. Software can be stored on the server when needed. More space is needed to complete entire documents.

Relationships between applications, based on this method and traditionally written applications. The method, described in this paper, does not exclude the sharing of data files and the application of software on the network server.

5. CONCLUSION

The applications of the method and any further developments are independent of the network and the server types. There is no need to create or use any server monitor routines. All the changes in one application do not affect the other applications. It is not necessary to reconfigure and recompile the server and network software.

The method developed by the author is applicable in cooperative work on computer network, designed on the base of a client-server architecture.

REFERENCES

1. Hoare, C. A. R. Communicating Sequential Processes. Prentice-Hall, London, 1985.
2. Denev, J. Finite State Processes in CSP. In: *Discrete Mathematics and Applications*, Blagoevgrad, 1993.

Received on September 15, 1996
Revised on November 10, 1997

South-West University
Blagoevgrad, Bulgaria