Description

Method for synchronization of programs on different computers in a network

Computer systems are increasingly being networked to one another via communications devices. In some applications, particular attention has to be paid to the synchronization of the data processing systems which are interconnected in a network.

In the process, the type of synchronization may be subject to various requirements. On the one hand, the synchronism required of the computers in a network may be predominantly of a time nature. In this case, if possible, the computers should operate in parallel, that is to say the processing of the instructions in a program which is running should as far as possible always be at the same processing state at any point in time. Thus, if possible, the computers should all be processing the same processing sequence at all times.

Going beyond pure time parallelity, the synchronism of computers in a network may be subject to the additional requirement that the respective processing sequences in the computers should also, if possible, have the same meaning content in terms of data technology. This means that the computers in a network should also achieve result values which are as identical as possible, with approximately concurrent instruction processing based on initial values which are as identical as possible. In such a case, the comparison of original values and selected current processing values from the individual programs on the computers is advantageous in the process of time synchronization.

In practical technical applications, owing to the widely different types of disturbance influences, it is impossible for the instruction processing in different computers that are networked to one another in a network to maintain time and/or logical-content synchronism over a lengthy time period. In fact, as a rule, special technical measures have to be carried out cyclically in order to maintain or reproduce the synchronism of the instruction processing actions in the computers.

This is particularly important if the computers in a network are intended to form a so-called high-availability or fail-safe system. Examples of this are so-called "one of two" or "two of two" systems.

In the case of a "one of two" system, for example, two connected computers with identical programs are intended to process the same original data, such as measurements for example, subject to the boundary condition of high computer system availability. It is necessary to ensure regularly that the processing states of the two computers do not diverge too severely in the medium term owing to processing speeds which, under some circumstances, are also slightly different. Furthermore, it is necessary to ensure regularly that both computers have matching processing results. In the situation in which the two computers are used, for example, for controlling a technical process which requires high availability, and one of the two computers fails, this is a precondition for the other computer to be able to continue to control the technical process virtually without any discontinuity. Thus, in the case of such a "one of two" system, the two computers involved must be synchronized in time by means of special synchronization measures. Furthermore, their current processing contents must be regularly checked for equivalence.

The conditions in a so-called "two of two" computer system are also very similar to those for a "one of two" computer

system. In this case, two connected computers with identical programs are intended to process the same original data, such as measurements for example, subject to the boundary condition of high processing reliability. However, if a sudden non-equivalence, which cannot be corrected, in the processing results is detected in this case, then the desired result reliability is no longer provided. Thus, in the case of safety-relevant processing, the user programs on both computers must be brought to a safe state, for example the stop state. Further processing of the safety-relevant process is no longer possible since it is impossible to decide which program on which of the two computers had the still "correct" processing results at the moment when the non-equivalence suddenly occurred.

Comparable boundary conditions also exist in a so-called "two of three" system. In this system, the requirements for high system availability on the one hand and high data processing reliability on the other hand can be satisfied at the same time. In such a case, three connected computers can process the same original data, using identical programs. Once again, it is necessary to ensure at regular time intervals that the processing states of the three computers have not diverged too severely and that, in particular, each computer is producing the same processing results. If any non-equivalences are detected during a comparison of current processing results, then one computer may be regarded as being defective, and must be excluded from the network if its current processing result suddenly and permanently deviates from the matching processing results from the two other computers. The system as maintains both processing reliability and availability if one of the three computers fails. This process is also called majority decision. Following such a situation, the original "three of two" system then, in practice, reverts to a "two of two" system or a "one of two" system depending on whether the two

remaining computers in the system are intended to continue processing the data for the respective programs with high result reliability or with high availability.

In the case of computer systems of the above type, measures which, as a rule, are regularly repeated must be used to ensure and/or reproduce the time parallelity of the internal processing sequence of the instructions. In the process, the measures which are required for such synchronization are intended to have as little adverse effect as possible on normal operation of each computer involved and, at least, to run as quickly as possible.

The invention is thus based on the object of specifying a method, which is as simple and robust as possible and, in particular, consumes as little processing time as possible, for synchronization of at least two programs, in particular user programs, on computers which are connected to one another.

The object is achieved by the synchronization method specified in claim 1. Further, advantageous embodiments of the invention are specified in the dependent claims.

The invention will be explained in more detail with reference to exemplary embodiments which are illustrated in the figures that are described briefly below and in which:

FIG 1: shows, by way of example, two computers which form a network by means of a communications link which is set up via a databus,

FIG 2: shows, by way of example, two computers which interchange data by means of so-called broadcasts via a databus,

FIG 3: shows a first timing diagram, on which the sequence of the synchronization method according to the invention is explained using the example of two computers in a network,

which computers can form a so-called "one of two" or "two of two" computer system, and

FIG 4: shows a second timing diagram, in which the sequence of the synchronization method according to the invention is explained using the example of three computers in a network, which computers can form a so-called "two of three" computer system.

Figure 1 shows a first example of a communications link between two data processing systems, called computers in the following text, for short. In this case, a first computer A, which can be operated via a keyboard and a screen 4, is connected to a databus 1 via a first coupling 3, and a second computer B, which can be operated via a keyboard and a screen 6, is connected to the databus 1 via a second coupling 6. A permanent communications link 8 is set up between these two computers, and is indicated in Figure 1 by a dashed line 8. Such a permanent, connection-oriented communications link can be used to interchange data continuously and directly between the two computers. Each computer is set up to send data just to the other computer and to receive data from it.

Figure 2 shows a further example of a communications link between the two data processing systems A and B. In this case, no permanent communications link is set up between the two computers. Instead of this, this communication is based on the so-called "broadcast principle". In this case, the computer A, for example, when required feeds a message 10, that is to say a so-called broadcast, into the databus 1. All the other computers which are connected to the databus 1, in the example in Figure 1 the computer B and possibly other computers that are not illustrated in Figure 1, are in an operating mode which allows data reception via the databus 1. All the other computers first of all receive the

broadcast fed into the databus from computer A and then check whether the data in the broadcast are addressed to them or have contents that can be evaluated.

The synchronization method according to the invention can be used without any limitation at least for both types of connection illustrated in Figures 1, 2, and it is intended to explain this method in detail in the following text, using the example in Figure 3.

In the timing diagram in Figure 3, a general time axis 14 which timing marks T1 ...T11 is shown on the left-hand side. These times represent the starting points of processing states of the two computers A and B, which occur during the sequence of the synchronization method according to the invention. These processing states are represented in the form of function boxes on a time axis 15, located to the right of the general time axis 14, for the computer A and on a time axis 15, located on the extreme right of Figure 3, for the computer B. In principle, the time axes 15, 16 for the computers A, B symbolize the sequence and the progress in the processing of instructions in programs which are running, in particular in user programs, which are called task A for the computer A, and task B for the computer B. The processing states required for the synchronization method according to the invention are inserted into the natural sequence of processing the instructions for tasks A and B and thus each result in an at least temporary interruption in the processing sequence of the command instructions for tasks A, B. The method according to the invention has the advantage that the additional processing states which are caused by the synchronization and are inserted into the sequence of the individual tasks are as short as possible.

The two tasks running on the computers A, B are thus interrupted only for as long as is absolutely necessary for the purpose of mutual synchronization.

In the case of the example illustrated in Figure 3, it is assumed that, while a task is running in one computer, processing

states occur, that is to say, for example, during the processing of the instructions in the user program which is running in the computer, an instruction occurs as a result of which a request is made for a synchronization process to run between the two computers. For example, on the time axis 15 for the computer A, these are the times T2 = syncpoint X and T11 = syncpoint X+1. In a corresponding manner, times T6 = syncpoint Y and T11 = syncpoint Y+1 which each initiate synchronization are shown on the time axis 16 for computer B. In order to simplify the explanation of the invention, it is first of all additionally assumed, by way of example, that the tasks A and B running in the computers A and B each comprise matching chains of command instructions, that is to say that, in consequence, the same user programs are being processed by the computers A, B. This means that, for example, the processing states associated with the synchronization times [lacuna] and T6 = syncpoint Y have identical contents in the respective tasks A, B. However, in the example in Figure 3, the processing of these tasks on the computers A and B has taken place at different times, that is to say the processing is no longer being carried out approximately at the same time or synchronously. This can be seen in Figure 3 from the fact that the syncpoint X in task A at the time T2 occurs earlier than the corresponding syncpoint Y in task B at the time T6.

The synchronization process which is initiated at the time T2 is thus necessary. This is thus initiated, according to the invention, in that that computer in the network, for example the computer A in Figure 3, as the first to reach a synchronization point during the processing of the corresponding task, that is to say during instruction processing reaches an instruction which requests synchronization with another computer or the other computers in the network [lacuna].

In this case, at the time T2, the further processing of the instructions in the user program which is running is interrupted first of all by the computer A, and at least one set of synchronization data for at least one other computer in the network is fed into the data

interface. In the example in Figure 15, this is represented by the processing block 17, which is denoted "send syncdata x", and has an arrow 35 originating from it.

According to the invention, immediately after feeding in the synchronization data set, the computer which initiates the synchronization switches directly to a receive state, that is to say to an operating mode which allows the reception of corresponding data sets from programs on other computers in the network. Particularly in the course of feeding in the synchronization data set from the program of the transmitting computer, no reception of acknowledgment messages from the programs on receiving computers in the network is expected for completion of the feeding-in process. Instead of this, after feeding in the synchronization data set, a changeover is immediately made to a data receive state, and the further processing of the current task is interrupted until the arrival, in particular, of a corresponding synchronization data set from a program of another computer in the network. In Figure 3, this is symbolized by the processing block 19 denoted by "receive wait" on the time axis 15 of computer A.

In order to reduce the amount of data to be interchanged and to be compared during synchronization, so-called signature formation can be used to considerably reduce the amount of data to be transferred. When a signature is being formed, a data set is reduced by a type of compression to

a uniquely identifying tag which may be of a very much smaller size than the data set.

During this processing block 19, which is denoted by "receive wait" and may also be regarded as a so-called "wait state with readiness to receive", the processing of the instructions in the user program "task B" continues on the computer B and, at the time T6, reaches a corresponding synchronization time, which is denoted by "syncpoint Y". According to the invention, the computer B now also interrupts the processing of the command instructions in the user program which is running, and feeds a synchronization data set into the databus. This is symbolized in Figure 3 by the processing block 20, which is denoted by "send syncdata Y", on the time axis 16 of computer B and an arrow 36 originating from it. As soon as the synchronization data set has been fed in, the computer B preferably switches to a state for receiving data. Since the computer A (which is the other computer in this example) in the network has already fed in its synchronization data set at the time T2, no "wait" phase occurs after computer B has switched over to the "receive" operating state, but the computer A can immediately receive the synchronization data set syncdata x which is already present. This is symbolized in Figure 3, at the time T8, by the processing block 26, which is denoted by "receive/receive syncdata x", on the time axis 16 for computer B.

On the other hand, once the transmission time represented by the arrow 36 in Figure 3 has passed, the synchronization data set which is sent by the computer B is immediately received by the computer A, which is locked in the "receive" operating mode, which is symbolized by the processing block 22, which reception is denoted by "receive syncdata y", on the time axis 15 of computer A. This thus results in its "wait" state ending. On the basis of the transmission duration, which is relatively short as a rule, of the

synchronization data set "syncdata y", this reception likewise starts approximately at the time T8. Synchronism between the processing states of the two computers A and B has now been achieved approximately, and further instruction processing thus takes place virtually concurrently, so that the desired synchronization aim is achieved.

Both computers advantageously each carry out an analysis of the respectively received synchronization data set, at the time T10. For the computer A, this is represented on the time axis 15 by the processing block 24 "processing sync data y", and for the computer B, this is represented on the time axis 16 by the processing block 28 "processing sync data x". These two processing activities also take place virtually concurrently.

The contents of the respective synchronization data set are irrelevant for the synchronization method according to the invention. In the simplest case, the synchronization data sets syncdata x and syncdata y transmitted backwards and forwards may each comprise, for example, a single bit whose function is to act as a time marker. Even this allows time synchronization of the processing states of computers to be achieved using the method according to the invention. Furthermore, the synchronization data sets may, of course, contain any desired additional information which can be used, for example, for matching the processing results from the two computers. In the example in Figure 3, for example, depending on the comparison result from a comparison of the synchronization data set syncdata x from computer A with the synchronization data set syncdata y from computer B in the processing block 24 ("processing syndata y") it is either possible to initiate continuation of the processing of the command instructions in the interrupted task A ("return to / ... : task A"), in

particular if a match of the two synchronization data sets has been found [lacuna]. On the other hand, particularly if there is non-equivalence, the

further processing can be interrupted or not started again ("... / stop : task A"). In the same way, by comparing the synchronization data set syncdata y from computer B with the synchronization data set syncdata x received from computer A in the processing block 28 ("processing syncdata x"), it is possible to initiate either a continuation of the processing of the command instructions in the interrupted task B ("return to / ... : task B"), or to finally terminate further processing ("... / stop : task B").

The method according to the invention has the particular advantage that the synchronization can be carried out particularly quickly, since the computers in the network which feed synchronization data into the data interface are not required to produce and wait for any acknowledgments, for example of correct reception of this data by another computer in the network. Instead of this, another computer in the network makes a virtual "response" by itself feeding synchronization data into the data interface. In this way, very effective synchronization can be achieved with minimal time losses and minimal additional load on the data transmission path between for computers in a network, on the basis of the additionally interchanged synchronization data sets. In the example in Figure 3, this synchronization which has been achieved or reproduced is represented by in each case one further synchronization time "syncpoint x+1" or "syncpoint y+1" which occurs at the time t11, virtually concurrently, both in the processing of the command instructions for task A on the time axis 15 and in the processing of the command instructions for task B on the time axis 16.

According to a further embodiment of the invention, after a computer has fed in at least one set of synchronization data for at least one other computer in the network, a timeout is advantageously started by said first computer and predetermined command instructions are carried out by said one computer in

the situation when it has not been possible for at least one other computer to receive any set of synchronization data by the end of the timeout, in particular a message is generated or normal instruction processing is continued. This measure is used, in contrast, to ensure that, if a fault occurs in another computer in the network, a computer requesting synchronization is not permanently, and under some circumstances in vain, waiting for the arrival of a synchronization data set from this other computer. If a computer which has requested synchronization by feeding in its own synchronization data set has not been able to receive a synchronization data set from another computer by the time it reaches the end of the predetermined timeout, then the requesting computer can initiate suitable measures. It can, for example, terminate the synchronization attempt with other computers in the network, and continue the processing (which has been suspended for the duration of the synchronization attempt) of the instructions of the respective user program which is currently running.

This embodiment of the invention has already been illustrated in the example in Figure 3. In this case, on the one hand, computer A starts such a timeout in the processing block 18 "start timeout A" at the time T3 immediately after feeding in the synchronization data set "send syncdata x". Since it has nevertheless been possible to receive another synchronization data set in good time, in the illustrated example syncdata y from computer B at the time T8 after a certain waiting time "wait" has elapsed, the timeout is reset again on reaching the time T10 in the processing block 23 "reset timeout A". In the same way, such a timeout is started by computer B in the processing block 25 "start timeout B" at the time T7 immediately after feeding in the synchronization data set "send syncdata y".

Since, as has already been explained above, computer B can immediately receive the synchronization data set syncdata x, which is already present, from computer A in the processing block 26,

the timeout immediately following this in the processing block 27 "reset timeout B", can already be reset.

The sequence of the synchronization method according to the invention will be explained with reference to the example of a second timing diagram, which is illustrated in Figure 4, for three computers in a network. The synchronization method according to the invention can also be applied, without any limitation, to a greater number of computers which are interconnected in a network. In the example in Figure 4, the sequence of a user program "task A" is shown in the left-hand region in a first processing block for computer A and, in particular, a synchronization routine "sync routine A" constructed according to the invention, the sequence of a user program "task B" is shown in the central region of Figure 4 in a second processing block for computer B, and, in particular, a synchronization routine "sync routine B" constructed according to the invention and, finally, the sequence of a user program "task C" is shown in the right-hand region of Figure 4 in a third processing block for computer C and, in particular, a synchronization routine "sync routine C" constructed according to the invention. In particular, the computers are coupled to one another in a network via a data network. In one application, the user programs task A, task B and task C may comprise identical command instructions, so that the network illustrated in Figure 4 then represents a computer network system which has already been explained above, a so-called "two out of three" system.

During normal instruction processing, computers A, B and C each reach a program item in which synchronization with the other computers in the network is requested and initiated. In the processing blocks in Figure 4, these are the program item "syncpoint X" in the

sequence of "task A", the program item "syncpoint Y" in the sequence of "task B", and the program item "syncpoint Z" in the sequence of "task C". Each of these program items is respectively followed by

5

an instruction routine entitled "write syncdata x", "write syncdata y" or "write syncdata z", respectively. This results in the processing of the respective task A, task B or task C being interrupted and, if required, temporary data are buffer-stored for subsequent continuation of program processing and a synchronization data set "syncdata x", "syncdata y" or "syncdata z", respectively, is generated by the corresponding computer A, B or C for the respective synchronization routine and for feeding into the data interface.

The synchronization routine "sync routine A" from computer A comprises the following command instructions in the example in Figure 4:

```
1.  send C,x
2.  send B,x
3.  set timeout A
4.  receive B,y
5.  wait
6.  receive C,z
7.  wait
8.  reset timeout A
9.  bool = x eq y eq z
```

The synchronization routine "sync routine B" from computer B comprises the following command instructions in the example in Figure 4:

```
10. send A,y
11. send C,y
12. set timeout B
13. receive C,z
14. wait
15. receive A,x
16. wait
17. reset timeout B
18. bool = x eq y eq z
```

The synchronization routine "sync routine C" from computer C comprises the following command instructions in the example in Figure 4:

```
19. send B,z
20. send A,z
21. set timeout C
22. receive A,x
23. wait
24. receive B,y
25. wait
26. reset timeout C
27. bool = x eq y eq z
```

The meanings of the individual command instructions 1 to 28 in the synchronization routines are explained as follows:

1. Computer A feeds a synchronization data set x, which is addressed to the computer C in the network, into the data interface (see the dashed arrow with the reference symbol 29).

2. Computer A feeds a synchronization data set x, which is addressed to the computer B in the network, into the data interface (see the solid arrow with the reference symbol 32).

3. Computer A starts a timeout A, within which it waits to receive the synchronization messages y, z from the other computers B, C in the network.

4. Computer A switches to a receive state and interrupts further instruction processing.

5. Computer A receives a synchronization data set y which is addressed to it and has been fed in by computer B

(see the solid arrow with the reference symbol 31).

6. Computer A remains in the receive state and continues to interrupt instruction processing.

7. Computer A receives a synchronization data set z which is addressed to it and has been fed in by the computer C (see the dash-dotted arrow with the reference symbol 30).

8. Computer A resets the timeout A before reaching the predetermined timeout point since, in the meantime, it has been possible to receive the expected synchronization data sets from computers B and C in good time.

9. Computer A carries out processing operations on its own synchronization data set x and on the received synchronization data sets y, z. In particular, the contents of the synchronization data sets x, y, z are checked to see if they are identical. If the desired result is obtained from the processing operations, then the further processing of task a is continued. Otherwise, the instruction processing of task A on computer A remains interrupted, and application-dependent fault handling can be initiated.

10. Computer B feeds a synchronization data set y, which is addressed to the computer A in the network, into the data interface (see the solid arrow with the reference symbol 31).

11. Computer B feeds a synchronization data set y, which is addressed to the computer C in the network, into the data interface (see the solid arrow with the reference symbol 34).

5

12. Computer B starts a timeout B, within which it waits to receive the synchronization messages x, z from the other computers A, C in the network.

13. Computer B switches to a receive state and interrupts further instruction processing.

14. Computer B receives a synchronization data set z which is addressed to it and has been fed in by the computer C (see the solid arrow with the reference symbol 33).

15. Computer B remains in the receive state and continues to interrupt instruction processing.

16. Computer B receives a synchronization data set x which is addressed to it and has been fed in by the computer A (see the solid arrow with the reference symbol 32).

17. Computer B resets the timeout B before reaching the predetermined timeout point since, in the meantime, it has been possible to receive the expected synchronization data sets from computers A and C in good time.

18. Computer B carries out processing operations on its own synchronization data set y and on the received synchronization data sets x, z. In particular, the contents of the synchronization data sets x, y, z are checked to see if they are identical. If the desired result is obtained from the processing operations, then the further processing of task b is continued.

Otherwise, the instruction processing of task B on computer B remains interrupted, and application-dependent fault handling can be initiated.

19. Computer C feeds a synchronization data set z, which is addressed to the computer B in the network, into the data interface (see the solid arrow with the reference symbol 33).

20. Computer C feeds a synchronization data set z, which is addressed to the computer A in the network, into the data interface (see the dash-dotted arrow with the reference symbol 30).

21. Computer C starts a timeout C, within which it waits to receive the synchronization messages x, y from the other computers A, B in the network.

22. Computer C switches to a receive state and interrupts further instruction processing.

23. Computer C receives a synchronization data set x which is addressed to it and has been fed in by the computer A (see the dashed arrow with the reference symbol 29).

24. Computer C remains in the receive state and continues to interrupt instruction processing.

25. Computer C receives a synchronization data set y which is addressed to it and has been fed in by computer B (see the dashed arrow with the reference symbol 29).

27. Computer C resets the timeout C before reaching the predetermined timeout point since, in the meantime, it has been possible to receive the expected synchronization data sets from computers A and B in good time.

28. Computer C carries out processing operations on its own synchronization data set z and on the received synchronization data sets x, y.

In particular, the contents of the synchronization data sets x, y, z are checked to see if they are identical. If the desired result is obtained from the processing operations, then the further processing of task c is continued. Otherwise, the instruction processing of task C on computer C remains interrupted, and application-dependent fault handling can be initiated.

The exemplary embodiment illustrated in Figure 4 already takes account, according to a further embodiment of the invention, of the fact that, in the situation when more than two computers are connected or can be connected via the data interface, each computer A, B or C feeds in in each case one set of synchronization data addressed to the other computers in the network, and this computer waits, in the opposite sequence to that previously used for feeding in, to receive a corresponding set of synchronization data from the other computers in the network. Thus, the computer A in Figure 4, for example, feeds in in each case one synchronization data set x (send C,x and send B,x) which is addressed to the computer C and to the computer B, respectively, in the network. In the opposite sequence to this transmission sequence, computer A then waits to receive the synchronization data sets y and z from the computers B and C (receive B,y and receive C,z).

The synchronization method according to the invention can also be used without any change if the tasks of the connected computers do not comprise identical sequences of command instructions but are set up merely to interchange data in specific program items, or even if only one computer is set up to receive data from the other computer.

Patent Claims

1. A method for synchronization of programs on at least two computers (A, B, C) in a network, which are connected or can be connected to one another via a data interface (1, 3, 6), in which case one computer (A) in the network

a) interrupts the further processing of the instructions in a program (task A) that is running when, during the processing, an instruction (syncpoint X,...) occurs which requests synchronization of the program (task A) in the computer (A) to at least one program (task B, task C) on at least one other computer (B, C) in the network,

b) feeds into the data interface (1, 3, 6) at least one set of synchronization data (syncdata x,...) for the at least one other computer (C, D) in the network,

c) after feeding in the at least one set of synchronization data (syncdata x,...), switches to the receive state (receive), and

d) interrupts the further processing of the instructions in the program (task A) which is running (wait) until the computer (A) has received, via the data interface (1, 3, 6), at least one set of synchronization data (syncdata y, syncdata z) from the programs (task B, task C) on at least one other computer (B, C) in the network.

2. The method as claimed in claim 1, in which case

a) after feeding in at least one set of synchronization data (syncdata x,...) for at least one other computer in the network, one computer (A, B, C) in the network starts an internal timeout (timeout A,...), and

b) in the situation when no set of synchronization data has been received from at least one other computer (A, B, C), the one computer (A, B, C) carries out predetermined command instructions, in particular generates a message and/or finally interrupts the processing of the instructions in the program (task A) which is running, or

c) otherwise, the one computer (A, B, C) resets the internal timeout (timeout A) and continues to process the instructions in the program (task A,...) which is running.

3. The method as claimed in one of the preceding claims, in which case, in the situation when more than two computers are connected or can be connected via the data interface,

a) one computer (A,...) in the network in each case feeds in at least one set of synchronization data (send 3,x, send 2,x,...), which is directed at in each case one other computer (B, C) in the network, and

b) this computer (A,...) waits (receive 2,x, receive 3,x,...) for the reception of a corresponding set of synchronization data from the other computers in the network, in the opposite sequence to that previously used for feeding in.

GR 96 P 3368 P

Abstract

Method for synchronization of programs on different computers in a network

In the case of the method, the processing of instructions in a program in one computer is interrupted when, during the processing, an instruction occurs which requests synchronization with a program on another computer. A set of synchronization data for a partner program on at least one other computer is then fed into the data interface from each program involved, a change is made to a receive state and the processing of the instructions in a program is interrupted until a set of synchronization data has been received from programs in the other computers.

FIG 1

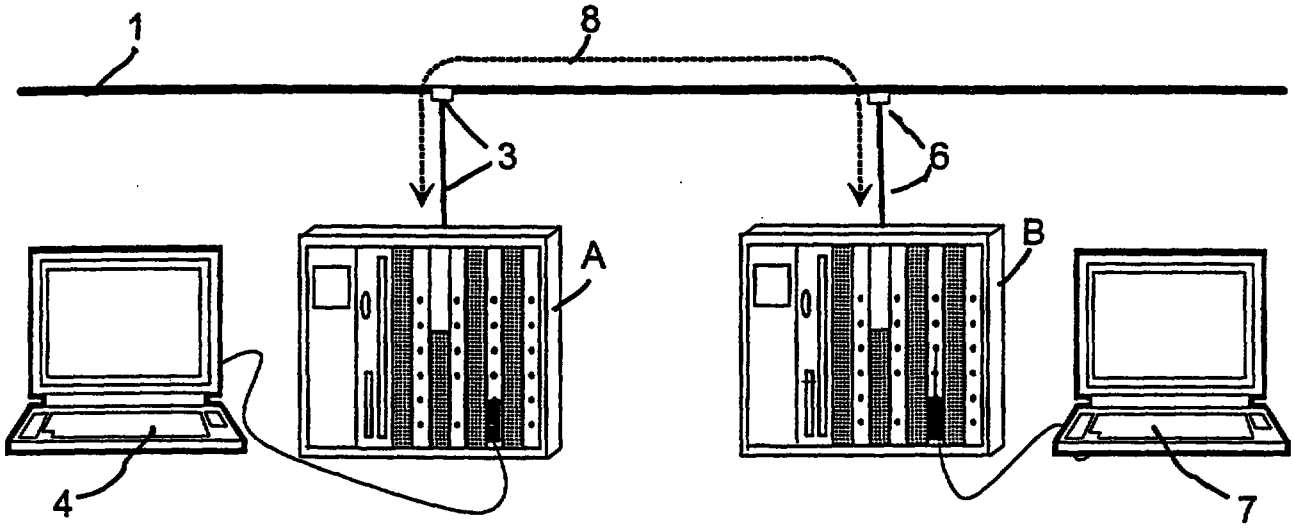


Figure 1

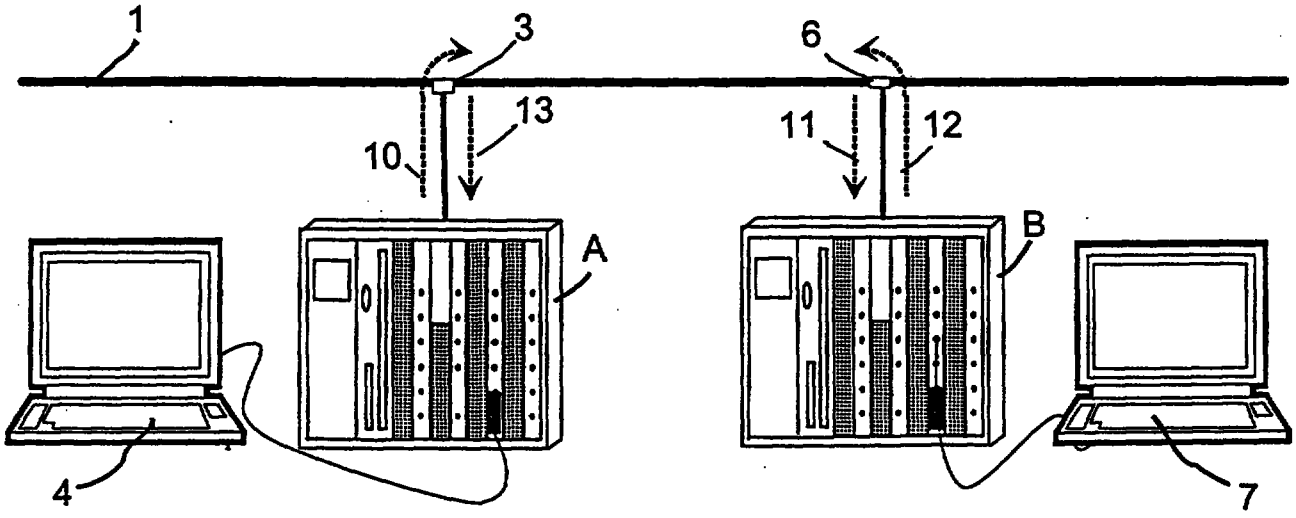


Figure 2

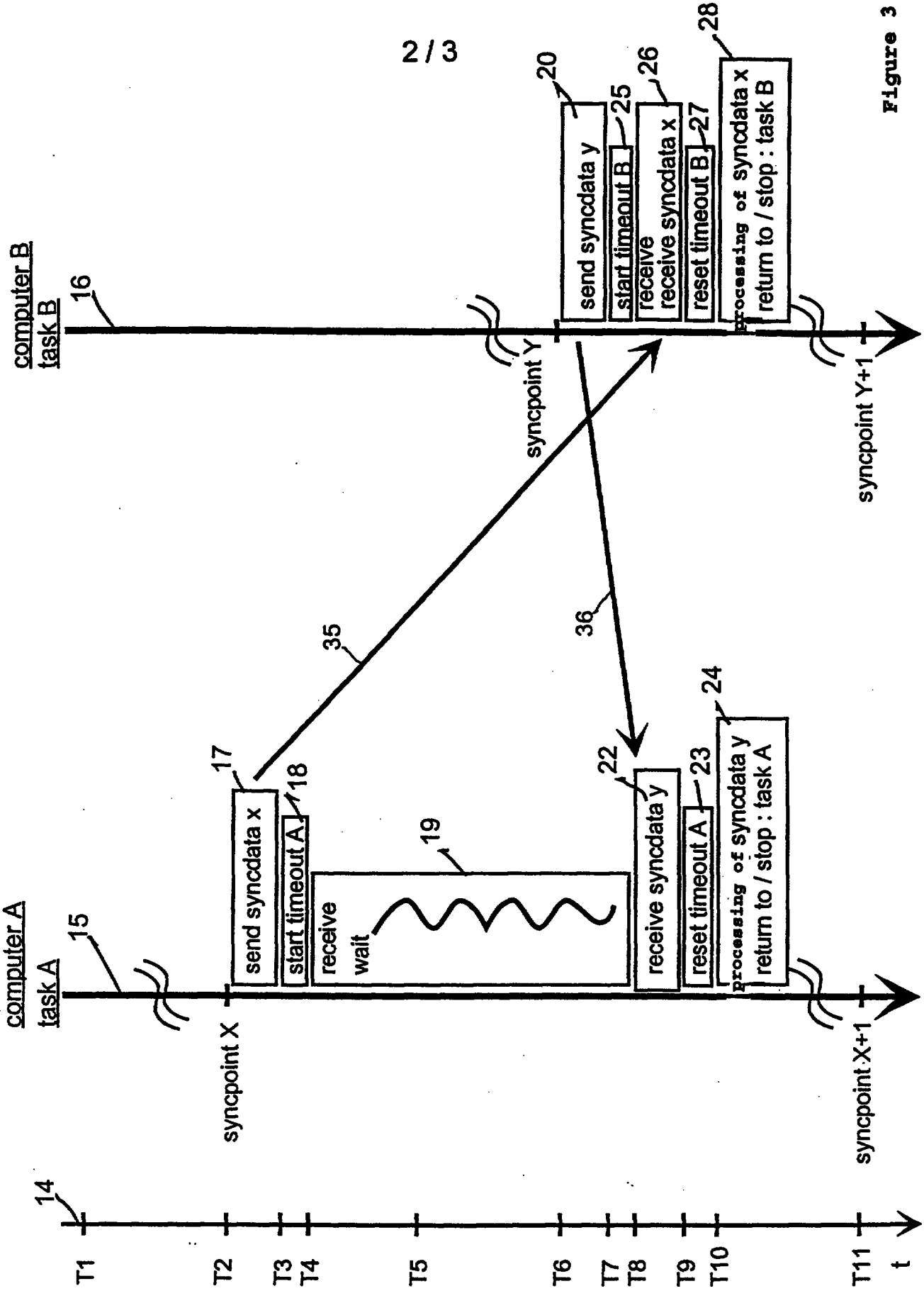

computer A
task A
15
send syncdata x
start timeout A
receive
wait
19
receive syncdata y
reset timeout A
processing of syncdata y
return to / stop : task A
syncpoint X
syncpoint X+1
17
18
22
23
24
35
36
computer B
task B
16
send syncdata y
start timeout B
receive
receive syncdata x
reset timeout B
processing of syncdata x
return to / stop : task B
syncpoint Y
syncpoint Y+1
20
25
26
27
28
14
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
T11
t

Figure 3

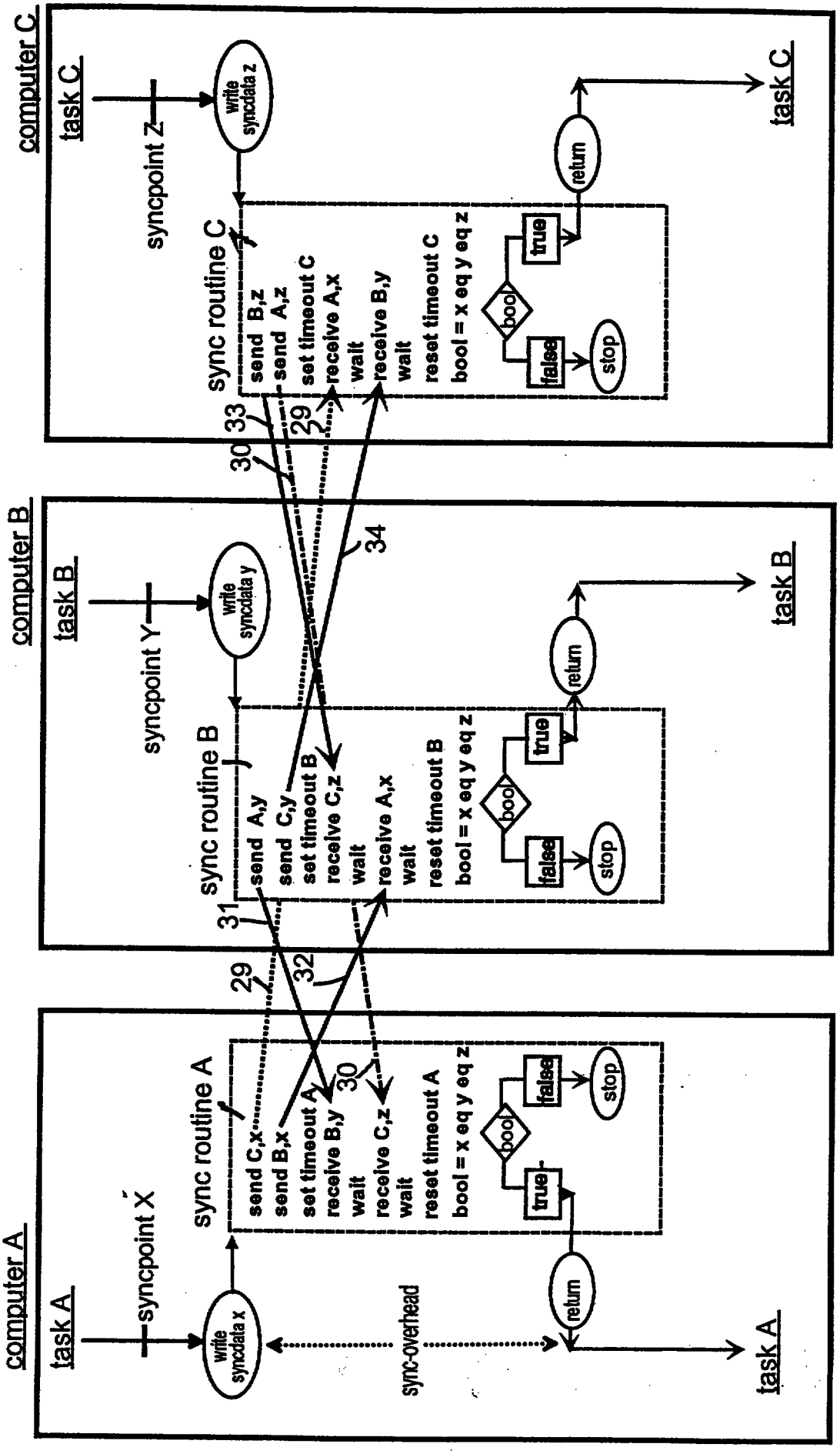

computer A
task A
syncpoint X
write syncdata x
sync routine A
send C,x
send B,x
set timeout A
receive B,y
wait
receive C,z
wait
reset timeout A
bool = x eq y eq z
bool
true
false
stop
return
sync-overhead
computer B
task B
syncpoint Y
write syncdata y
sync routine B
send A,y
send C,y
set timeout B
receive C,z
wait
receive A,x
wait
reset timeout B
bool = x eq y eq z
computer C
task C
syncpoint Z
write syncdata z
sync routine C
send B,z
send A,z
set timeout C
receive A,x
wait
receive B,y
wait
reset timeout C
bool = x eq y eq z
29
30
31
32
33
34

Figure 4