PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : G06F 9/46, 11/00 — A1

(11) International Publication Number: WO 94/18621

(43) International Publication Date: 18 August 1994 (18.08.94)

(21) International Application Number: PCT/SE94/00079

(22) International Filing Date: 2 February 1994 (02.02.94)

(30) Priority Data:
9300431-5 10 February 1993 (10.02.93) SE

(71) Applicant: TELEFONAKTIEBOLAGET LM ERICSSON [SE/SE]; S-126 25 Stockholm (SE).

(72) Inventor: MÅLÖY, Jon, Paul; Himlabacken 9, S-170 74 Solna (SE).

(74) Agents: ROSENQUIST, Per, Olof et al.; Bergenstråhle & Lindvall AB, P.O. Box 17704, S-118 93 Stockholm (SE).

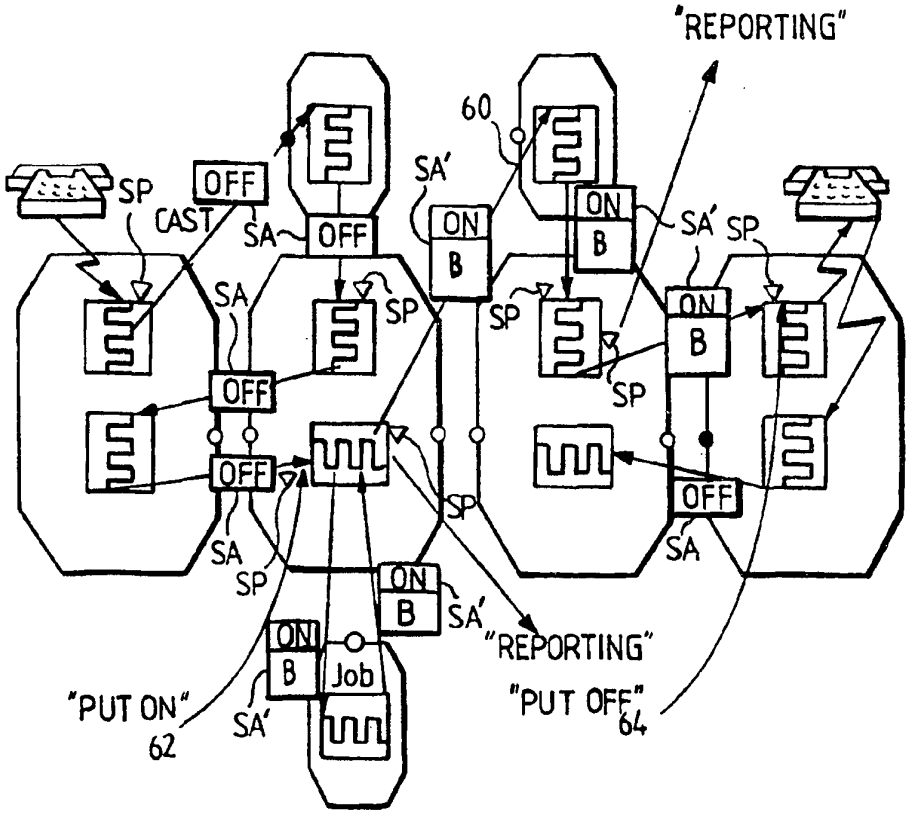
(81) Designated States: AU, BR, CA, CN, FI, JP, KR, NO, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: A METHOD AND A SYSTEM IN A DISTRIBUTED OPERATING SYSTEM

(57) Abstract

For establishing two-way communication links between processes in a distributed operative system, the processes are provided with ports through which communication between the processes is performed. The processes and the ports make possible for the operative system to keep a check on processes having links and to use these links also if the process per se is terminated, and to discover an error in the process and terminate it. For enabling the operative system to be able to transmit via the links information regarding process or computer drop out and thus be able to propagate this information through the whole chain of linked processes, and to report this information to applications executed in the linked processes in order to enable for these to undertake application specific measures, a code is used which is called at link abortion and communication errors. The function of this code includes terminating an erroneous process and reporting the error to an error handling code. The first mentioned code is always executing in a process to which an error has been reported.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	**GB**	United Kingdom	**MR**	Mauritania
AU	Australia	**GE**	Georgia	**MW**	Malawi
BB	Barbados	**GN**	Guinea	**NE**	Niger
BE	Belgium	**GR**	Greece	**NL**	Netherlands
BF	Burkina Faso	**HU**	Hungary	**NO**	Norway
BG	Bulgaria	**IE**	Ireland	**NZ**	New Zealand
BJ	Benin	**IT**	Italy	**PL**	Poland
BR	Brazil	**JP**	Japan	**PT**	Portugal
BY	Belarus	**KE**	Kenya	**RO**	Romania
CA	Canada	**KG**	Kyrgystan	**RU**	Russian Federation
CF	Central African Republic	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CG	Congo			**SE**	Sweden
CH	Switzerland	**KR**	Republic of Korea	**SI**	Slovenia
CI	Côte d'Ivoire	**KZ**	Kazakhstan	**SK**	Slovakia
CM	Cameroon	**LI**	Liechtenstein	**SN**	Senegal
CN	China	**LK**	Sri Lanka	**TD**	Chad
CS	Czechoslovakia	**LU**	Luxembourg	**TG**	Togo
CZ	Czech Republic	**LV**	Latvia	**TJ**	Tajikistan
DE	Germany	**MC**	Monaco	**TT**	Trinidad and Tobago
DK	Denmark	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**US**	United States of America
FI	Finland	**ML**	Mali	**UZ**	Uzbekistan
FR	France	**MN**	Mongolia	**VN**	Viet Nam
GA	Gabon				

A method and a system in a distributed operating system.

Technical field of the invention

The present invention generally relates to handling processes and related resources in a distributed operating system.

With a process, in the present connection also called context, is here meant a resource in an operating system which needs to be used by a job for enabling it to execute program code in the process. The process provides the job with several indispensable resources, such as its own program counter, its own memory space, and its own set of processor registers. The process synchronizes jobs by only allowing one job at a time to execute.

By job is here meant, more generally, a phenomenon which is directed towards a process, so that a method in an object owned by the process is executed. A job can create new jobs directed towards other processes or to the own process.

Description of related art.

US 3 905 023 illustrates and describes a system including a multiple level operating system. The system is characterized as very big and extraordinary complicated. The reliability of the system hardware is secured by the capacity of the multiple level operating system to reconfigurate the system modules dynamically and automatically in a suitable way. In all main modules of the system there are error detecting and error reporting circuits providing the operating system with information for performing error analyses and dynamic reconfiguration of the system resources. The memory modules are provided with "single bit" error correcting ability independently of the operating system. The operating system may be regarded as including a basic level and N consecutive levels. The basic level is defined as the core of the operating system. A process in each level of the operating system is

responsible for the processes creating on the nearmost higher level and not for any other ones. The operating system controls the system resources dynamically and plans job or tasks in a multiple program mixture. It reassigns resources, starts jobs and supervises their execution.

GB 2 079 997 relates to a distributed computer system with a plurality of systems connected to each other. Each one of the systems has a plurality of mutually connected elements. The systems include redundant elements with a distributed operating system
for operating, error supervision and reconfiguration of functions while using vertical addressing. When an error is detected, the error is verified, the erronous element is isolated and its task is assigned to another unoccupied element. If no other element should be available the system is reconfigurated for enabling deteriorated operation while using the available elements.

In US 4 933 936 there is described a distributed computer system providing for flexible error tolerance. A distributed operating system is resident in all computers. Each computer is controlled by a resident copy of a common operating system.

Summary of the invention.

In a computer it is desired that communication errors, or errors caused by erroneous programs, shall be able to be handled by the operating system of the computer in such a way that it is kept intact and that other programs and calls will not be affected by the error. An error shall at worst involve controlled disengagement of the chain of linked processes, or calls, where the error occurred. The effects of the error shall be completely isolated with respect to this call. In other words recovery of an arisen error must not include greater consequences for the system than those caused by the error itself.

A first object of the invention is to enable, in a distributed operating system, demounting of a chain of linked processes while returning as many memory and hardware resources as possible to the system.

A second object of the invention is to enable isolation of errors and limit their consequences, including the consequences of the recovery measures, only to the transaction/call in question, and thus if possible avoid computer restarts and influence on other calls.

A third object is to enable tracing of errors, irrespective of where these appear in the system.

A fourth object is to enble, in connection with system updating, type marking of certain activities for being able to control the execution of these towards the desired program ware version.

Generally according to the invention, for establishing two-way communication links between processes in a distributed operative system, the processes are provided with ports through which communication between the processes is performed. The processes and the ports make possible for the operative system to keep a check on processes having links and to use these links also if the process per se is terminated, and to discover an error in the process and terminate it. For enabling the operative system to be able to transmit via the links information regarding process or computer drop out and thus be able to propagate this information through the whole chain of linked processes, and to report this information to applications executed in the linked processes in order to enable for these to undertake application specific measures, a code is used which is called at link abortion and communication errors. The function of this code includes terminating an erroneous process and reporting the error to an error handling code. The first mentioned code is always executing in a process to which an error has been reported.

More particularly, a method according to the invention, for handling resources in a distributed operating system

comprises the steps of

providing two-way communication links between said processes and using said operating system

for keeping up with processes having links, and

using said links also if a process having links is terminated,

detecting an error in a process and terminating it,

transferring information via said links regarding process or computer failure, and propagating this information through a whole chain of linked processes, and

reporting this information to applications executed in said linked processes for enabling these to perform application specific recoveries.

A system according to the invention comprises code means including

first means for providing two-way communication links between processes,

second means for enabling said operating system to keep up with processes having links, and to use said links also if a process is terminated,

third means for enabling said operating system, or in certain cases a process itself, to detect an error in a process and terminate it,

fourth means for enabling said operating system to transfer failure information via said links regarding process or computer failure, and to propagate this failure information through a whole chain of linked processes, and

fifth means for enabling said operating system to report said failure information to applications executed in the linked processes in order to enable for these to perform application specific recoveries.

Brief description of the drawings.

The invention will now be described more closely with reference to embodiments schematically shown on the attached drawings, on which

Figure 1 illustrates an exemple of an activity in the form of a chain of jobs in a distributed operating system,

Figure 2 illustrates exemples of an activity group formed by several such activities,

Figure 3 illustrates how resources can belong to

an activity for a shorter or longer time,

Figure 4 shows a link representation view of an activity,

Figure 5 is intended to illustrate that consequences of an error in an activity may be isolated to the activity itself,

Figures 6 a-d illustrate how disengagement of an activity may be performed when an error has appeared in a process,

Figure 7 illustrates system upgrading,

Figure 8 illustrates the design of an error chasing system in the activity according to Figure 1,

Figure 9 shows actors performing at the appearance of an error situation in a process,

Figures 10-13 illustrate the handling of process local errors,

Figures 14-16 illustrate the handling of communcation errors,

Figures 17-19 illustrate the handling of errors in other processes,

Figure 20 in a table sums up error cases described with reference to Figures 6-19.

Detailed description of preferred embodiments.

In the different Figures the same reference characters are used for illustrating the same or similar elements.

In the description below and on the drawings, expressions familiar to the man of the art relating to messages and communication may be used, as well as pseudo syntax expressions of a certain type. To the extent that they are not explained below, it is presumed that the man of the art does not require any closer explanation or translation of these expressions and syntax, respectively.

The concept of activity used below is used for defining a chain of jobs created in an operating system as a result of an independent external or internal event, plus the sum of the resources used by the chain during its execution.

Figure 1 shows a "log" of such a job chain which as an example is illustrated as arisen due to events in a telephone exchange between two telephone subscribers A and B. More particularly the Figure shows an activity in the form of a chain of jobs, and three of the types of resource an activity can attach to itself, viz. processes, ports and subscriber equipment. More particularly, processes are designated 2, jobs 4.n, ports 6, and subscriber equipment 8 in the Figure.

The arrows relate to different messages in the job chain, such as an asynchronous communication message 10, also called "cast" message, and synchronous messages in the form of call and reply messages 12 and 14. More particularly, with asynchronous messages are here meant such messages which a process sends and can continue its execution without waiting for response, whereas in the case of synchronous messages the process is locked until a reply has arrived. Each new "cast" message results in a new job, such as 4.2, which then very well can exist simultanously with the job 4.1, which has created it. The call 12 also results in a new job 4.6, whereas the calling job 4.5 is temporarily suspended. Not until the new job 4.6 has stopped executing and has sent a reply message 14, the suspended job 4.5 may continue.

With an "independent" external event is meant an event not directed to any activity in the system. If the A-subscriber lifts the telephone receiver is this an independent event starting a new activity. If the B-subscriber lifts the receiver it is not an independent event, since it is directed towards a call under erection and thereby towards an existing activity. If the A or B subscriber puts on the receiver the same is true.

Most internal events are not independent. If e.g. a debiting pulse is received this is the result of the fact that an activity has ordered a periodic time supervision, and has thus created a temporarily resting "timeout" job. This is included in the activity. Certain internal events should however be regarded as independent. This may apply to such as start of test activities from a test generator

or triggered absolute time supervisions (of the type waking up, start of routine tests).

It is not necessarily so that a job in the chain directly has to have arisen in another job or a call via the communication mechanisms in the operating system. It may e.g. happen that, during a certain space of time, there is no job within the activity, either executing or waiting in some queue. In such cases it is only the link picture, which will be described more closely below, that defines the activity. If now a new job is started from some of the resources which exclusively belong to the activity, e.g. the line circuit of the B subscriber, also this job 4.10 belongs to the activity.

Referring to Figure 2, if an operator or a third part C wishes to be connected into the speech, the distinction between "independent" and "dependent" will be somewhat more difficult. It is true that the event is directed to an existing activity 20, but it results at first in the creation of a new activity 22. These two activities will then form an "activity group", shown schematically in Figure 2, by the job chains "meeting" in the same resource, i.e. in the half call process 24 of A. It should however be observed that the fact that two activities share a resource is not a sufficient criterium for allowing that they shall form an activity group. Many activities (calls) shall of course share the access processes without being included in the same recovery domain for that reason.

A criterium good enough is presumably that activities sharing dynamic processes form an activity group, whereas those sharing static processes do not. Static processes are considered to be robust enough to be able to stand that an activity is recovered without this affecting the other ones sharing the process.

As is schematically illustrated in Figure 3 the activity, during its lifetime, attaches different resources for shorter or longer time. A job 25 beginning to execute attaches e.g. always a resource 26 of the type process. In many cases, e.g. static start processes, the process is

released directly when the job terminates, but it may also be attached to the activity for a longer time, e.g. by there being created a port 28 to the process, so that new calls from the same activity can arrive at a later point of time, as is indicated at 30 and 32, and which e.g. may imply that a new process 34 is attached or disconnected, respectively.

One important type of resource that the activity usually attaches is communication ports which belong to the communication mechanisms of the operating system. All ports belong to a process and each port has a reference to an opposite port. By linking together ports the activity may thus create a link picture according to Figure 4, which keeps together the "owner" processes of the ports 6. In that way the activity may attach a process also during the time in which it has no job which shall be executed in it. Please observe, however, that this "attaching" does not imply any exclusive access to the process.

It is important to notice that a link picture is only something existing in the form of its nodes and links. Thus, there is no central or even distributed co-ordinating function which has knowledge of the extension and existance of the link pictures. The only knowledge of a link picture existing in the system is the limited information existing in each port (a node knows its immediate neighbours in the link picture).

The ports 6 are also usable for indirectly attaching such resources that are administrated in a process to an activity. In the program executing in the process "Access A" in Figure 4 there is an internal reference between the port 6, that has contact with the hardware of the subscriber A, and the port 6 that directly belongs to the link picture. Such "internal" connections may be needed when it is not desirable to terminate the current process together with the rest of the link picture. Typically, static processes are expected to survive disengagement of a link picture (c.f Figure 6).

Of course there are a number of other types of resources which may be attached to an activity during the

existance thereof, but it is always the ports and the link picture which make it possible to keep together all these resources.

Due to the fact that resources and jobs belonging to an activity are kept together there is formed a new type of "domain" in the system. As illustrated in Figure 5 this domain 40 "traverses" all the computers 42, 44, 46 and 48 involved in the call, but are on the other hand well delimited within each computer. With support of the right type of mechanisms this domain 40, i.e. the activity, may to great advantage be used as a recovery domain.

If it is possible to limit the consequences and extension of an error to keep within the activity, and simultaneously accomplish that all occupied resources are released, it is then possible, at worst, to disconnect the call controlled by the activity, whereas all other calls remain untouched.

This is in great contrast to methods according to the stand of the art, where the smallest recovery domain is the separate computer. In case of more serious errors in a call the standard measure is to restart the computer, with the consequence that all calls belonging to that computer must be disconnected.

In case of a serious error appearing in one of the processes in the link picture, the normal measure is to disconnect the whole call, i.e. the activity, in a way that no "call rests" remain. If the ambition is only this, it is possible to perform this by means of the operating system itself. Release of busy resources may however be more flexible and faster if the application contains code which can handle the release. Figures 6a-d illustrate the typical view when a call is disconnected due to error. In these Figures the erronous process is designated 50, static processes 52, and dynamic processes 54. In the example shown, the chain of events extends through three steps, viz. according to Figures 6a, 6b and 6c, respectively, and results in the condition shown in Figure 6d where only the static processes 52 remain. More particularly, every process always first sends an interruption message 56,

called "ConnectionAbort" out over its links before it terminates itself according to arrows 58. For the last mentioned step the designation "ContextTerminate" is used.

An activity may also operate as a client for system updating. All, or parts, of the activity may be directed towards executing a specific version of program ware. If e.g. a new version of a program has been installed it is possible to create during a time "test activities" which use this program version, whereas "normal" activities still are controlled towards the old version. Later it is possible to choose to also control new "normal" activities towards the new program ware.

This requires that the activity is associated with an "activity attribute". The attribute must include a field with information about the type of activity. This attribute must follow in all messages, jobs, time supervisions and job creating resources included in the activity.

The "area of interest" of the system updating in the activity is the job chain and the job creating resources (e.g. access processes and ports) i.e. the parts of the activity which may contain a system updating attribute. The link picture is not of interest or visible from the point of view of system updating.

Figure 7 more in detail illustrates the performance of system updating. In this Figure contexts A, B, C, D, E, E', F, F', G are shown. In one each of these contexts programs execute, which for the sake of simplicity may be assumed to have the same designation as the corresponding context. There is only one program version in the contexts A-D and G, the programs A,D and G being assumed to be of an old version, and the programs B and C of a new version. Each of the programs E and F exist in two different versions, which execute in E and E' and F and F', respectively.

During a certain phase of the system updating e.g. all "normal traffic" proceeds towards an "old" program version, i.e. contexts E and F, and all "test traffic" towards "new" program version, i.e. the contexts E'and F'. The shift between the two versions according to this system

is illustrated by means of arrows E" and F", which are indicated as movable. Running of test traffic is thus shown in the Figure. If only one program version exists, all traffic will necessarily be controlled towards this, which thus is true for contexts A-D and G. The rectangles UA with the text "TEST" included in the Figure indicate the above mentioned system updating attribute included in the activity.

The communication service of the operating system knows the program versions which are available and controls the calls according to existing "directing rules". It should be emphasised that the "rules" which are used according to Figure 7 only are a simplified example.

When it is necessary to trace errors the activity can also be used as carrier of tracing information. The activity attribute therefore includes a field indicating if the tracing is activated or not, and some "visibility attributes", for indicating which type of events (e.g. each message sending) that are to be "viewed" during the tracing. A tracing identity is also included. Attribute and tracing identity may indirectly, ordered by an operator, be changed wherever and whenever during the execution of the activity. If the tracing is on, the activity attaches a resource in the form of a tracing information buffer. This also follows the activity and is available in all computors where the activity executes.

In Figure 8 a started tracing in the activity according to Figure 1 is marked with a thicker line 60. The above mentioned tracing attribute is indicated by rectangles SA, the text "OFF" and "ON", respectively, indicating that the attribute is "off" and "on", respectively. The view of the tracing system of the activity is still more limited than the one for the system updating. The interest is only directed to parts of the job chain, viz. the parts following after the tracing attribute has been "put on" at 62 and up to the point (the points) 64 where it is put off again. This part 60 of the job chain may be called an execution thread. Within the execution thread it is furthermore only certain events which are of

interest to be seen. The tracing attribute changes its size in the moment it is changed. In the position "on", which appears in five cases at SA', the attribute contains a buffer B with tracing information. In the position "off" no such buffer is needed.

The tracing attributes may be read and changed in certain "tracing points", which are located in well defined points along the extension of the job chain. Some of these tracing points have been marked as an exmple in Figure 8 as triangles SP. A tracing point is a code which is always called in case of events in the activity. The tracing point is able to read, during this call, the contents of the tracing buffer and decide, from its "visibility attribute", if the event shall be reported, i.e. be visible to the tracing operator, or not.

Examples of visibility attributes which can exist are: "Report the contents in each message which is being sent", whereupon the tracing point located in each port takes care of this being done, or "Report the identity to each job being created" resulting in one tracing point in each process creating such a report.

In order that the tracing points shall be able to both report events and also change tracing attributes it is required that they have an interface to an operator, i.e. a man. How this communication is performed does not form part of the invention, but it may be elucidating to see which type of information that passes the interface.

A typical order to be given by an operator to a tracing point is "put on the tracing attributes in all execution threads passing and put in a visibility attribute with the meaning 'report message sendings' in the buffer of the tracing attributes".

A typical report to be given by a tracing point to the operator is "A message with the identity XX and contents xyz was just sent from port No. ABC to port No. DEX".

The link picture or further resources are not of interest from the point of view of tracing.

The present invention is based on the following

conditions:

- All computers directly involved in the activity must work with an operating system which supports the mechanisms which are required for carrying through the invention. Computers not having such operating system must only exist as usable "resources" controlled from the activity.

- The communication mechanisms of the operating system are expected to have advanced means for error detection, and possibility of reporting errors to the users, which is known per se.

- The required extensions of the communication mechanisms of the operating system must not affect the executing and sending capacity more than superficially.

- The system and its hardware components are assumed to be so robust that recovery measures become relatively rare. Frequent and massive recoveries would seriously affect the availability of the system.

The invention is not concerned with

- how static processes recover after context failure,

- support, if any, for recovering failured or partly failured activities - all recovery, going beyond the functionality to disconnect the activity and return the execution resources must be performed by the application itself,

- some mechanism for returning used resources except execution resources of the type ports and contexts.

Below a description will be given of the architecture and the principles on which the invention is based. In turn, actors in case of error situations, handling of process local errors, handling of communication errors, and errors in other processes will be treated.

Actors in case of error situations.

These are codes in a machine interpretable language which may be known per se, e.g. compiled from the programming language C++, and which can be executed in case of appearance of different types of error situations. In the below used names of the actors in question appears in

some cases a syllable "Exception". This syllable is included for particularly indicating that the actor in question is executed in connection with some type of abnormal event, i.e. an exceptional event.

- "ErrorHandler"

This is the error handler of the operative system. In Figure 9 66 designates a faulty process and 68 an associated executive core. A neighbour process and the associated executed core are designated 70 and 72, respectively. The processes 66 and 70 communicate, indicated at 74, with each other via ports 76 and 78, respectively.

"ErrorHandler", which is indicated at 80 and 82, respectively, has as its task to receive error indications from the processer hardware and the executive core, as well as from the applications themselves, which are indicated at 84 and 86, respectively, in Figure 9. In case of such indications "ErrorHandler" can sometimes actively intervene and control the recovery, sometimes only keep statistics over the number of errors. "ErrorHandler" is reached only by means of two calls: via the call "UserException" 88 from the application 84, and the call "reportError" 90 from the parts of the core functions executing in supervisor mode. The errors indicated are then stated in parameters following the respective calls. "UserException" is a call to be used when an error shall be reported. As a parameter in connection with this call an error code and textual error information, if any, is stated.

All error codes to "ErrorHandler" following with the call "UserException" and "reportError" will be supplemented with available error information, i.e. normally an error code and a short textual description of the error.

- "PortExceptionHandler" 92

This is a specialized exception handler of the communication mechanisms of the operative system, which is called in case of link abort and communication errors. Its immediate recovery measure is to terminate the process in question and report the error to "ErrorHandler". The

handler can however be rewritten or further specialized by the application designer so as to enable a more qualified recovery. This exception handler executes always in the process to which the error has been reported.

As regards error calls to "PortExceptionHandler", it is a name of the code which is executed in case of the exception call "handleException" in a function "Port" and its specializations, which will be described more closely below.

- "ApplicationExceptionHandler" 94

This is the specialized exception handler of the application which is called in cases where the application is allowed to get back the control after the detection of an error. Default recovery measure is to return all resources and terminate the process in question. The handler may, however, be further specialized by the application designer, so that more qualified recovery can be done. This exception handler always executes in the process where the error has appeared.

"ApplicationExceptionHandler" is the name of a code executed after the call "UserException". "ApplicationExceptionHandler" does not handle communication errors, but only process local executing errors.

- "Context" = process

Among other things, "Context" will also keep a check on which ports are attached to it. When a process gets instructions to terminate, either it is a normal or abnormal termination, it can very fast point to the ports which will be without an owner and order these to terminate themselves and their links.

A call to "Context" is "terminateProcess". This takes away the process in question, and also envolved therein is that all these remaining ports shall be taken away.

- "Port"

In connection with error handling a port has several tasks:

1) To receive "delete" and while performing this send out "ConnectionAbort", arrow 96, to the port, if any,

78, to which it is linked.

2) To receive error indications from other ports or from "MainGate" 98 and call "PortExceptionHandler" 92, arrow 100, with information regarding the error.

Regarding error indications to "Port" the following applies:

1) Send a message of the type "ReturnedMessage" including available error information to the port. The port will then call "PortExceptionHandler" with an error code.

2) Send a message of the type "connectionAbort" including available error information to the port. The port will then call "PortExceptionHandler" with the error code "connectionError".

3) The call "connectionAbort" gives the information to the port that the port to which it is linked does not exist any longer. This has the same importance and effect as the message "connectionAbort".

- "MainGate" 98

This "port" handles some specific errors which have to be taken care of by the communication mechanisms of the operating system. Among other things it must be able to receive and handle wrongly addressed messages, as there is no destination port which can handle this. When such a message arrives, it generates a message of the type "ReturnedMessage" towards the sending port. "MainGate" is not connected to any process.

- "Computer Execution Capability Control" - "COECC" 102

"COECC" has as its task to know the status of all other computers belonging to the subnet. In the case of error handling it has only one task, namely to find ports having links towards ports in a failured computer and thereafter call these with "connectionAbort". A message "stateChange" gives the information that a computer in the subnet has changed its status.

- "Application" 84 = 86

The expression "Application" is used in a wide sense, i.e. all users of the communication mechanisms described here. In many cases it can discover errors

itself, and report and even handle these.

- "Kernel" 104 = 68 = 72

By "Kernel" is meant the executive core. It reports errors to "ErrorHandler". "Kernel" among other things includes certain parts of the commucation mechanisms of the operating system, namely "MainGate" and "Port", since the error handling of these includes executing on the user process and reporting errors therefrom. "COECC" is also a part of "Kernel", but is drawn separately, since its functionality has a specific relevance in case of error detection.

No specific error calls to "Kernel" exist. In cases where "Kernel" acts in error situations it has only an active role.

Below a number of error handling situations will now be described with reference to drawing Figures 11-19. With respect to their general contents these drawing Figures correspond to Figure 9 and have the same reference characters as in this Figure for designating similar functions and phenomenon. The figures appearing within brackets in the drawing figures in question indicate numbers of order for the function steps appearing in the respective Figures.

Handling of processor local errors.

- Execution errors in the application, detection by a component or the execution core. Reference is made to Figure 10.

Errors of this type can be such as addressing beyond a permitted area, division with zero, overflow, loops etc.

The error results in an (often hardware) interruption that causes the current core function 104 to send via "reportError" (1) an error indication 90 to the "ErrorHandler" 80 of the operating system. In case of such errors the process is always judged as unreliable, and "ErrorHandler" therefore sends "terminateProcess" (2) to the process which in turn sends "delete" (3) to the ports which are left. These in turn send "ConnectionAbort" (4) over their links. If it is the question of a static process

"ErrorHandler" then creates a new process of the same type and calls the start routine of the same.

- Execution errors in the application detected by the application 84 itself. Reference is made to Figure 11.

If the application program 84 detects that some serious error has occurred during the execution it takes the initiative itself to call (1) "ErrorHandler" 90, as usual via "UserException" 88. This time the process is judged as "reliable" since it is capable of detecting and reporting the error itself. "Kernel" 104 therefore has the possibility of letting the control return to the specialized "ApplicationExceptionHandler" 94 of the application. Default measure for this should nevertheless consist in terminating the process with "TerminateProcess" (2), whereupon everything proceeds as in the present case with "delete" (3) and "ConnectionAbort" (4).

- Error in case of system call. Reference is made to Figure 12 and 13.

If a serious error is detected by the core 68 during a system call 105 the return value from the core will indicate this (1) according to Figure 12. An "Exception" 106 is thrown (2) to the application so that the "ApplicationExceptionHandler" 94 itself of the application program can take care of the error (3). After this the case enters that just described with reference to Figure 11, with "terminateProcess" (4), "delete" (5) and "ConnectionAbort" (6) with termination, if any, (8) of the process.

In case of certain errors the core 104 can, however, directly draw the conclusion that the process is unreliable. In such cases the core reports (1) instead directly to "ErrorHandler" 80, according to Figure 13, and this then terminates the process (2). The continuation is the same as in Figure 12.

<u>Handling communication errors.</u>

- Lost message.

If a message of the type "Call" or "Reply" has been lost this will be detected by a time supervision of the original calling part being released. In the case

"Call-Reply" it is the calling port that orders time supervision, and when this is released the relevant error code is returned as a reply to the call "Call". The continuation will be exactly the same as for failed system calls, as has been described earlier with reference to Figures 12 and 13.

If the lost message is a "Cast" it is instead the calling application itself which orders the time supervision. When this is released the calling part is in the same situation as in the case already described with reference to Figure 11.

With reference to Figure 14 lost messages can also be detected by sequence enumeration. For e.g. "Call", "Cast" and "Reply" the following appears. In case of a two-way link all messages sent over this will be sequence enumerated, so that the receiver can detect gaps in the enumeration. The following can happen. The calling part sends a sequence enumerated message, which is lost on its way (1). The calling part sends its next message (2), the sequence number of which is incremented with one. The receiving port 76 detects the gap in the enumeration and sends a message to the calling part 78 of the type "ReturnMessage" (3) with information regarding the missing number. The port 78 first calls "ErrorHandler" 82 (4) and then "PortExceptionHandler" 92 with an error code "LostMessage" (5), whereupon "PortExceptionHandler" makes some form of recovery.

- Wrongly addressed message. Reference is made to Figure 15.

A message 130 (1) which for some reason includes an erronous destination address (a portname not published, an old port reference or similar) will appear in "MainGate" 98. This then sends a message (2) of the type "ReturnedMessage" to the port 78 of the sender. The port 78 first calls "ErrorHandler" 82 (3) and then "PortExceptionHandler" 92 (4) with the error code "PortNotAvailable". Thereafter the case can be brought back to those earlier described.

- Disconnected contact.

If the contact 74 to another computer is broken this may be detected in two ways:

1) Reference is made to Figure 16. An emitted message will not arrive. Instead it will appear in "MainGate" 98 of the computer to which it has arrived (1). As in the former case this will send a "ReturnedMessage" to the sender port 78 (2), whereupon the case can be brought back to the former one described with reference to Figure 15, although with another error code, namely "ComputerNotAvailable".

2) The link supervision of the sender port detects that the destination can no longer be reached, and calls with "reportError" to the "ExceptionHandler" (not shown). Thereafter the case will be the same as case (1).

Errors in other processes.

- Failured process in own or other computer. Reference is made to Figure 17.

When a process 66 fails (i.e. is terminated by "ErrorHandler"), but the computer, on which it was executing still is intact, all its linked ports, such as 76, will send out "ConnectionAbort" (1) over its links. This results in a call with an error code (2), first to "ErrorHandler" and then to "PortExceptionHandler" 92 in the receiver process 70, which performs default or a specified recovery.

- Failed computer in own subnet. Reference is made to Figure 18.

If a computer in the own subnet fails, "COECC" 102 will very soon be informed about that with "StateChange" (1). "COECC" will then find out the ports having links directed towards this computer, and calls these with "ComputerNotAvailable" (N). Each port then calls "ErrorHandler" and its own "PortExceptionHandler" with "ComputerNotAvailable". Thereafter the course of events proceeds analogously with other errors of the same type.

- Failed computer in another subnet. Reference is made to Figure 19.

If a computer in another subnet fails, "COECC" will not be informed. The disappearance of the computer

will be detected either by no message arriving or by the link supervision of the operating system. The case is therefore in practice the same as the case described earlier with reference to Figure 16, and is detected and treated the same way.

-Loops in other processes.

Infinite program loops are detected in two ways:

1) "Kernel" detects the loop and releases the same chain of events as described with reference to Figure 10.

2) The time supervision in the calling process releases. The case then passes into the case "Lost messages" as described above, c.f Figure 14.

The error cases described above i.a. with reference to Figure 11-19, are also summed up in Figure 20. The table contains the abbreviation IPC, which refers to the communication mechanisms of the operating system.

In the above description of different error cases with reference to the drawings no closer description in detail has been given of software and hardware, to be used, or of how the described functions and processes are performed in practice, since it is pre-supposed to be clear to the man of the art how the invention shall be practiced guided by the description and the drawings. The invention may also be used in known operating systems and does not presuppose any special hardware.

Claims

1. A method in a distributed operating system for enabling demounting of a chain of linked processes, wherein

with a process is meant a resource in said operating system in which applications may be executed, and which must be used by a job for enabling execution of program code in the process,

with said job is meant a phenomenon which is directed to said process in order that a method in an object owned by the process should be executed, a job also being able to create new jobs directed to other processes or towards an own process, and wherein

said process provides the job with resources and synchronizes jobs by only allowing one job at a time to execute,

said method comprising the steps of

providing two-way communication links between said processes and using said operating system

for keeping up with processes having links, and using said links also if a process having links is terminated,

detecting an error in a process and terminating it,

transferring information via said links regarding process or computer failure, and propagating this information through a whole chain of linked processes, and

reporting this information to applications executed in said linked processes for enabling these to perform application specific recoveries.

2. The method according to claim 1, comprising the further step of using error tracing attributes to be sent from job to job in a chain of linked processes.

3. The method according to claim 2, comprising

allowing change of value of said attributes at any time during the execution of a job chain.

4. The method according to claim 2 or 3, comprising

providing a said tracing attribute with a tracing

buffer able to store information regarding events in said system traced by said attribute, and

creating by means of said information a log over said events.

5. The method according to any of the preceding claims, comprising

providing in said system a system updating function,

providing system updating information attributes able to carry information internal to said updating function regarding type of traffic carrying on in the system,

tranferring said information attributes in a chain of jobs for enabling determination of version of a specific program to be executed at an execution occasion.

6. A system for enabling in a distributed operating system demounting of a chain of linked processes, wherein

with a process is meant a resource in said operating system in which applications may be executed, and which must be used by a job for enabling execution of program code in the process,

with said job is meant a phenomenon which is directed to said process in order that a method in an object owned by the process should be executed, a job also being able to create new jobs directed to other processes or towards an own process, and wherein

said process provides the job with resources and synchronizes jobs by only allowing one job at a time to execute,

comprising code means including

first means for providing two-way communication links between processes,

second means for enabling said operating system to keep up with processes having links, and to use said links also if a process is terminated,

third means for enabling said operating system, or in certain cases a process itself, to detect an error in a process and terminate it,

fourth means for enabling said operating system to transfer failure information via said links regarding process or computer failure, and to propagate this failure information through a whole chain of linked processes, and

fifth means for enabling said operating system to report said failure information to applications executed in the linked processes in order to enable for these to perform application specific recoveries.

7. The system according to claim 6, comprising

communicating mechanisms,

process ports included in said first means, via which communication between processes may be performed,

communication ports not connected to a process for handling specific errors which have to be dealt with by said communication mechanisms of the operating system,

said process ports having in connection with error handling the task of

receiving a "delete" instruction relating to its own process, and while executing this instruction, sending a link disconnecting message, to any port, to which it is linked,

receiving error indications from other process ports and from any communication port, and calling code in said communication mechanisms for transferring information thereto regarding the error in such an error indication.

8. The system according to claim 7, wherein said second means include processes ports.

9. The system according to claim 7 or 8, wherein said third means include the operating system and processes.

10. The system according to any of claims 7-9, wherein said code is an exception handling code included in said fourth means and always executing in a process, to which an error is reported, and has the function of

being called in case of link abort and communication errors,

terminating an erronous process, and

reporting the error to an error handling code.

11. The system according to any of claims 7-9,

including error tracing attributes to be sent from job to job in a chain of linked processes.

12. The system according to claim 11, comprising means for allowing change of value of said tracing attributes at any time during the execution of a job chain.

13. The system according to claim 10 or 11, wherein a said tracing attribute is associated with a tracing buffer for storing information regarding events in said system when traced by said attribute, said system comprising means for creating by means of said information a log over said events.

14. The system according to any of claims 6-13, comprising

a system updating function,

system updating information attributes for carrying information internal to said updating function regarding type of traffic carrying on in the system,

means tranferring said information attributes in a chain of jobs for enabling determination of the version of a specific program to be executed at an execution occasion.

Fig.1

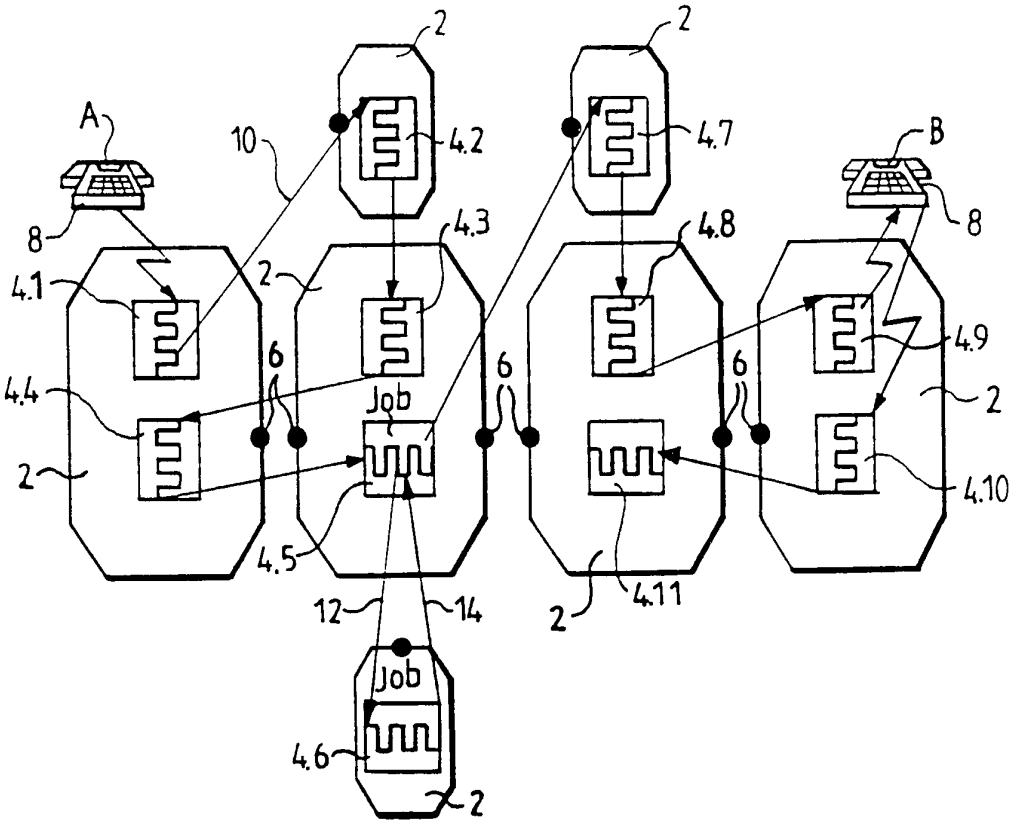

A
B
C
2
4.1
4.2
4.3
4.4
4.5
4.6
4.7
4.8
4.9
4.10
4.11
6
8
10
12
14
Job
Job

Fig. 2

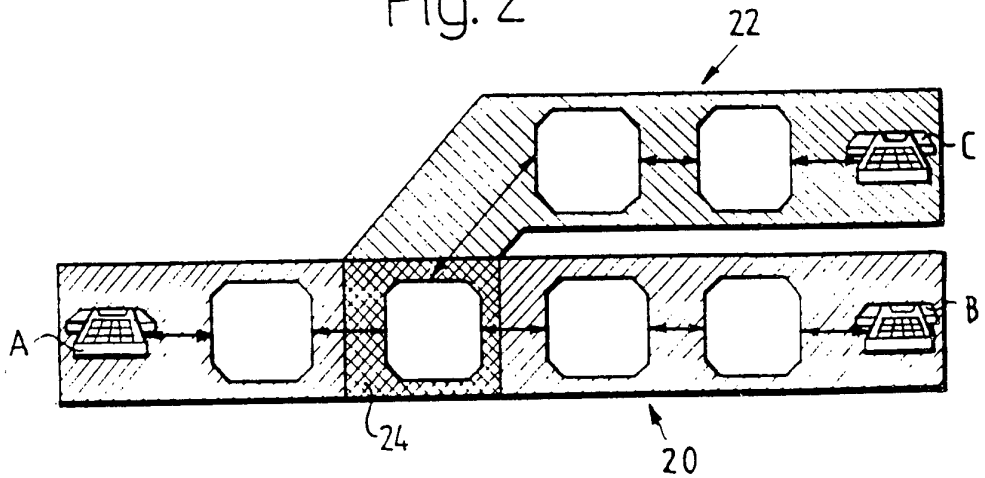

22
A
B
C
20
24

Fig. 3

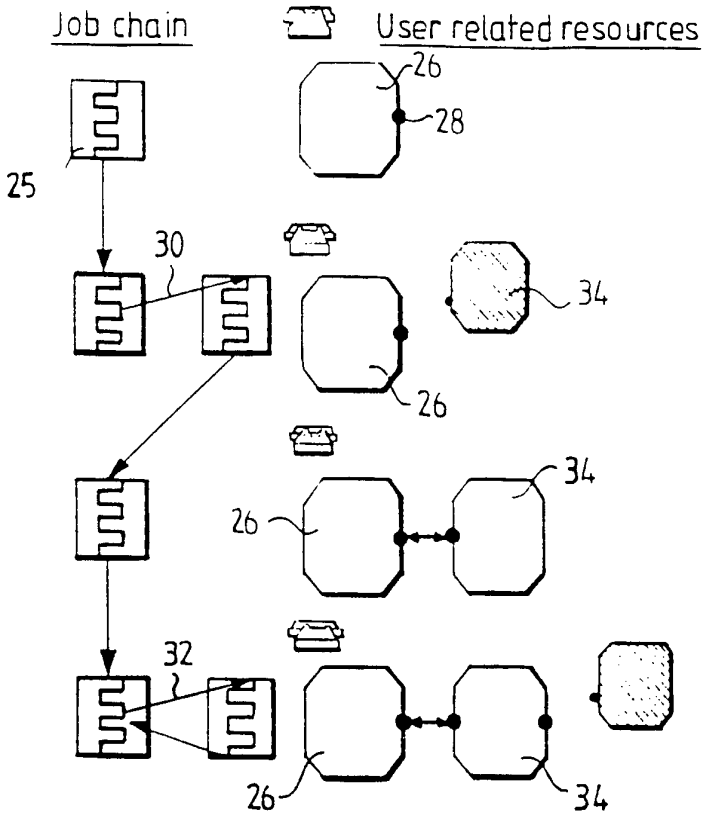


Fig. 4

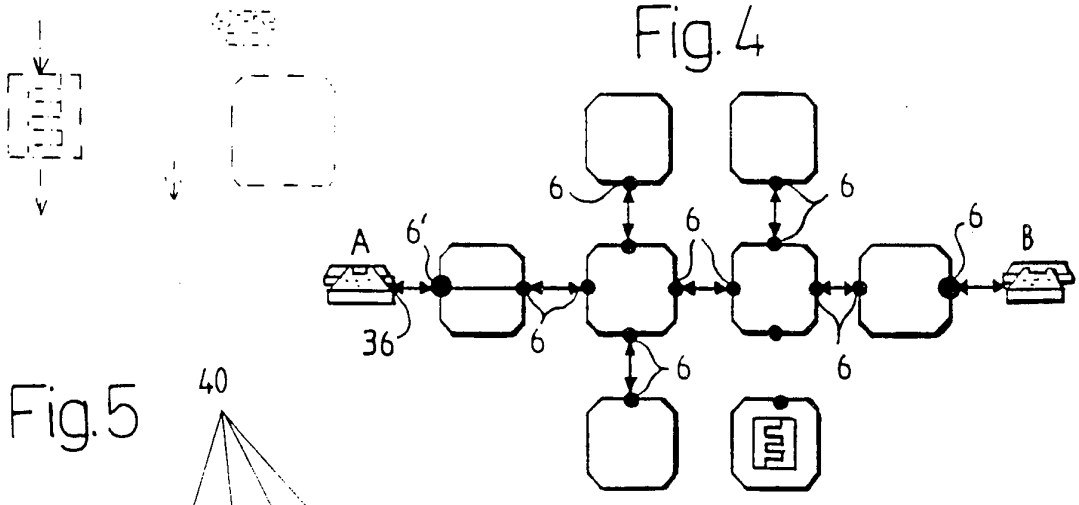


Fig. 5

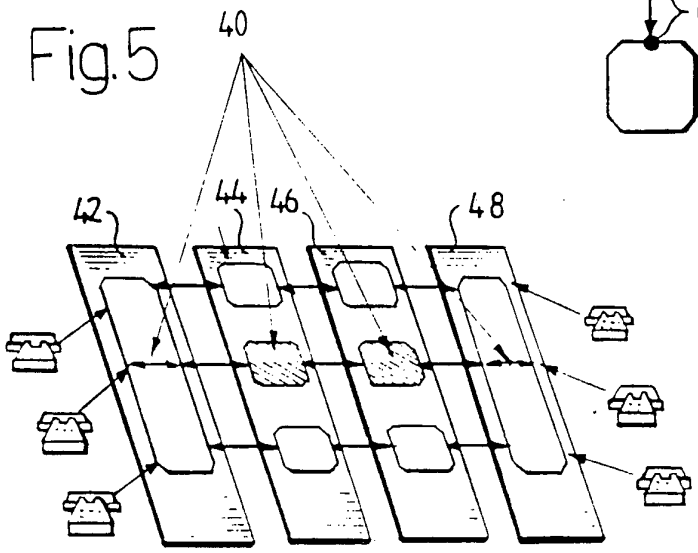


Fig. 6a

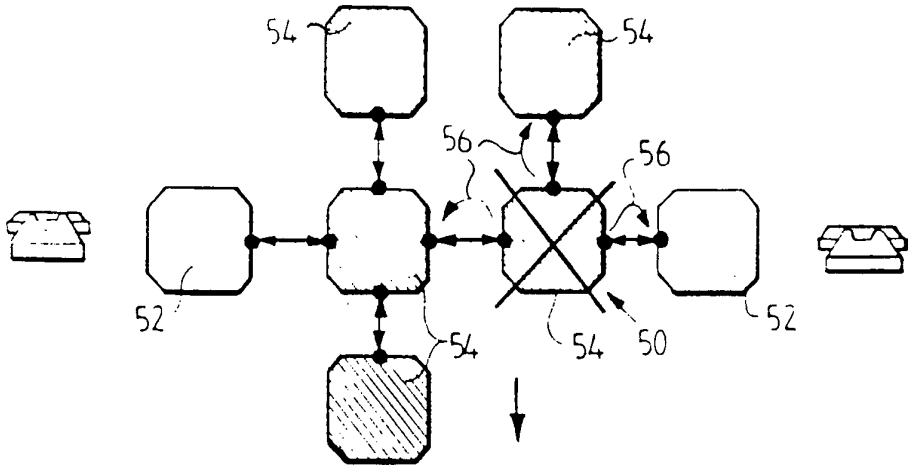


Fig.6b

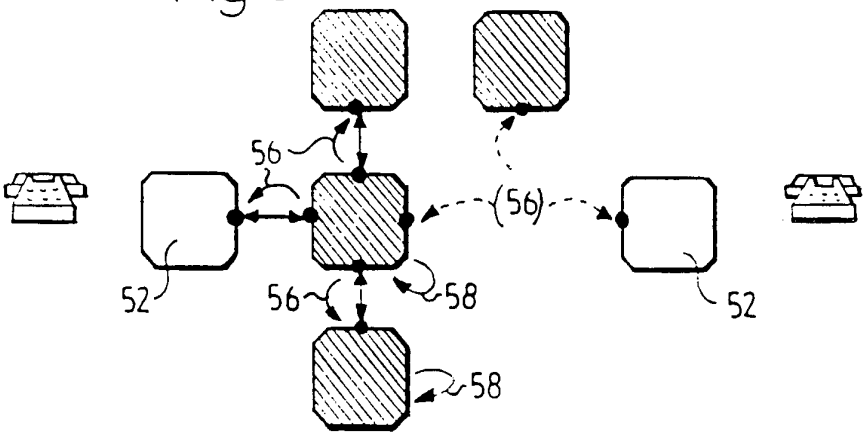


Fig.6c

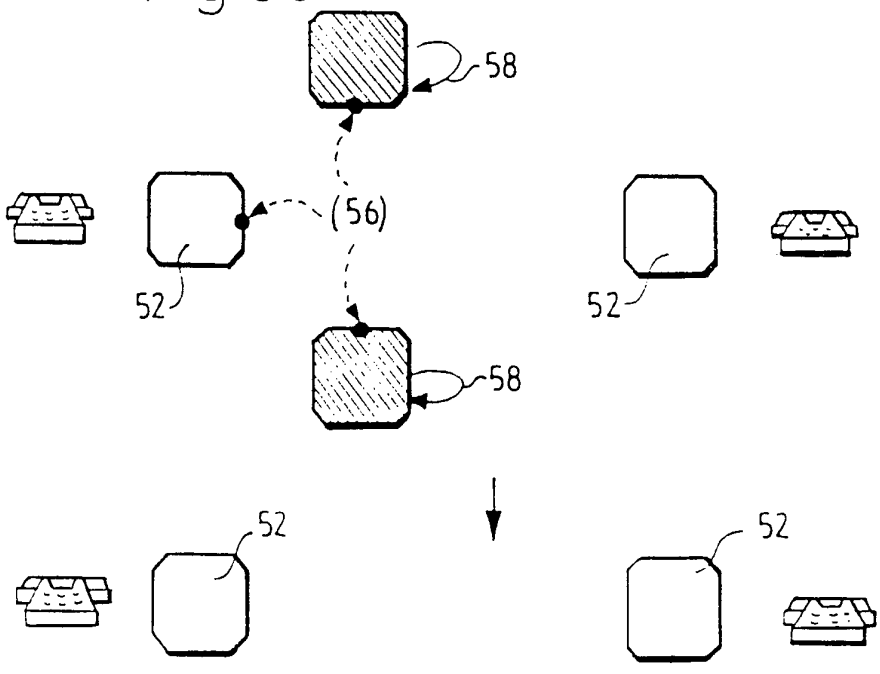


Fig.7

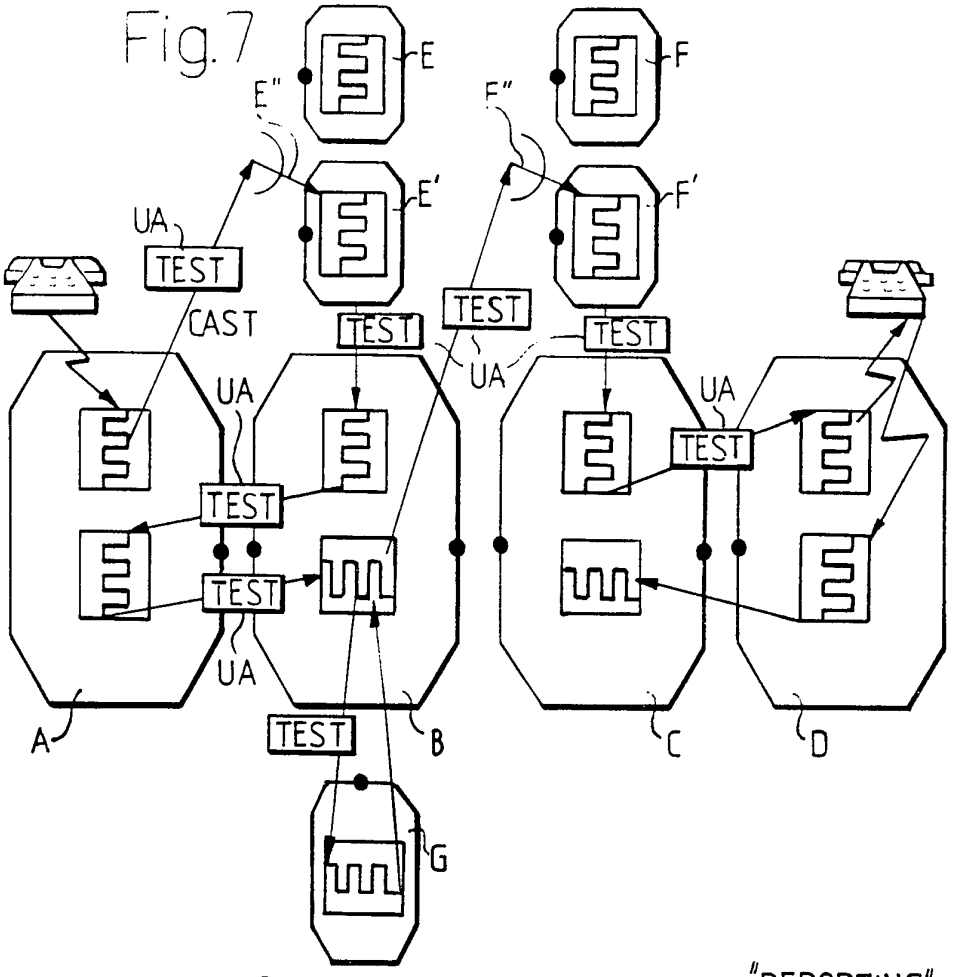

E
E"
E'
F
F"
F'
UA
TEST
CAST
TEST
TEST
UA
TEST
UA
UA
TEST
TEST
TEST
UA
TEST
A
B
C
D
G

Fig.8

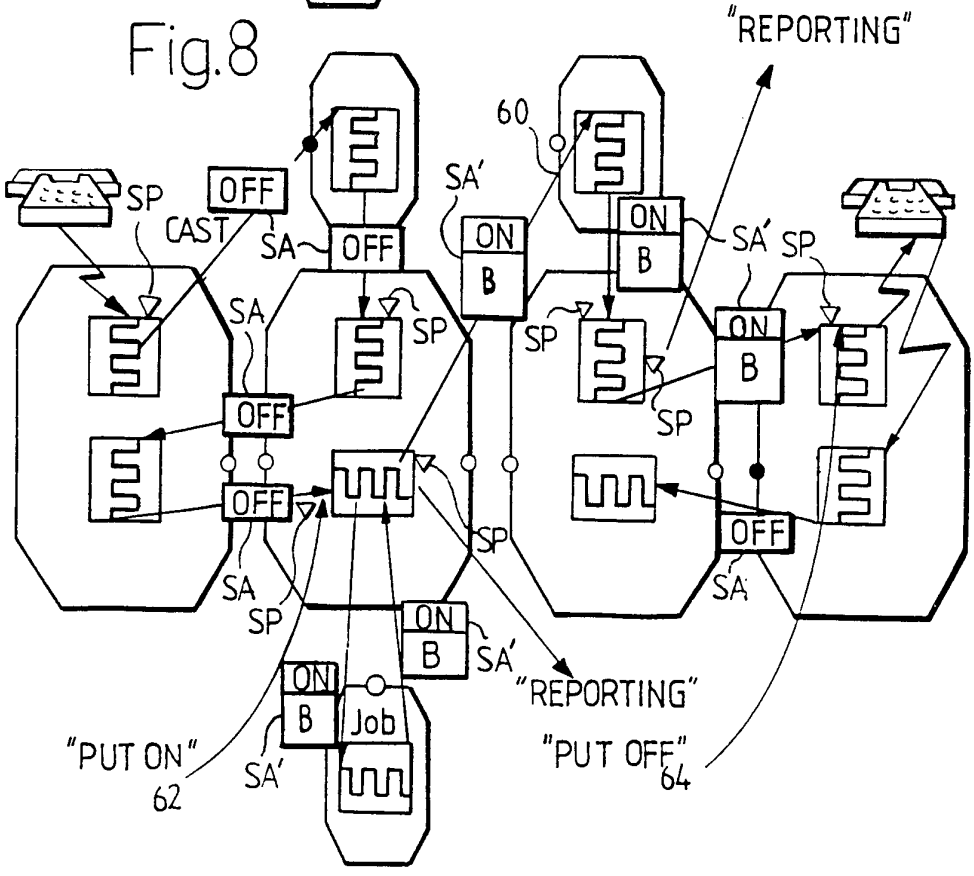

"REPORTING"
60
SP
OFF
CAST
SA
OFF
SA'
ON
B
ON
B
SA'
SP
SA
SP
SP
ON
B
OFF
SP
OFF
SP
OFF
SA
SP
SA
ON
B
SA'
"REPORTING"
ON
B
Job
"PUT ON"
62
SA'
"PUT OFF"
64

Fig. 9

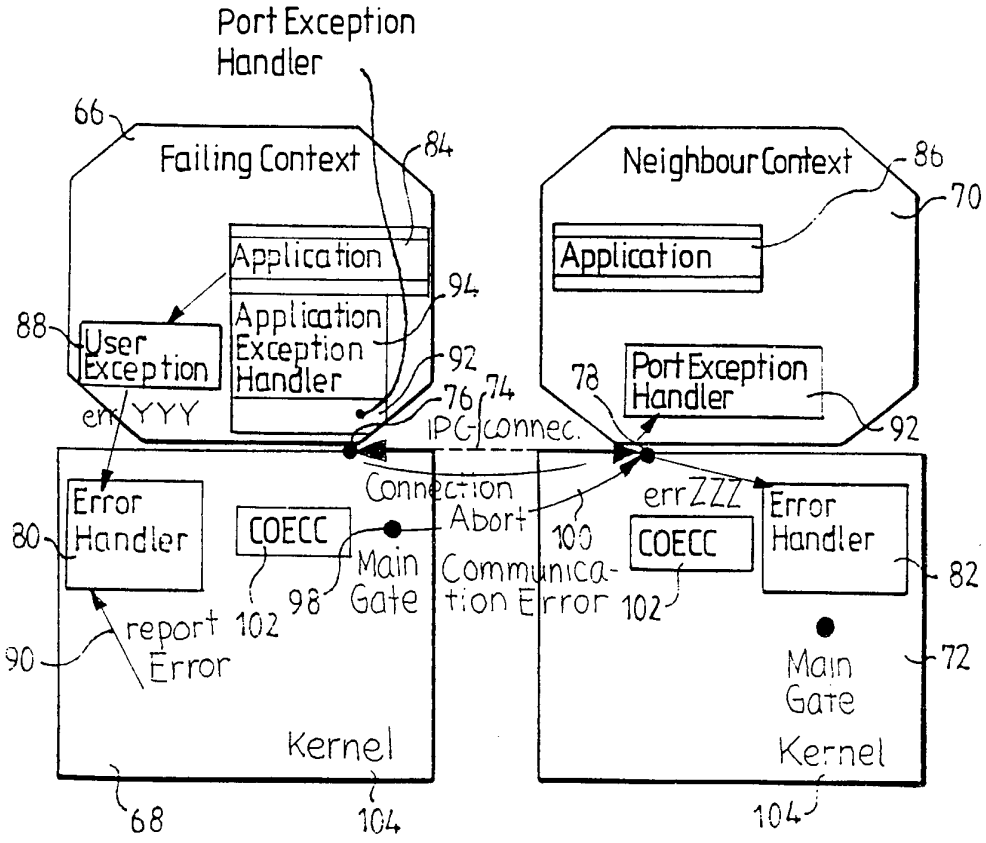

Port Exception Handler
66
Failing Context
84
Application
94
Application Exception Handler
88
User Exception
92
76
74
IPC-connec.
errYYY
Connection Abort
Error Handler
80
COECC
Main Gate
98
102
report Error
90
Kernel
68
104
Neighbour Context
86
70
Application
78
Port Exception Handler
92
100
errZZZ
COECC
102
Error Handler
82
Main Gate
72
Kernel
104

Fig. 10

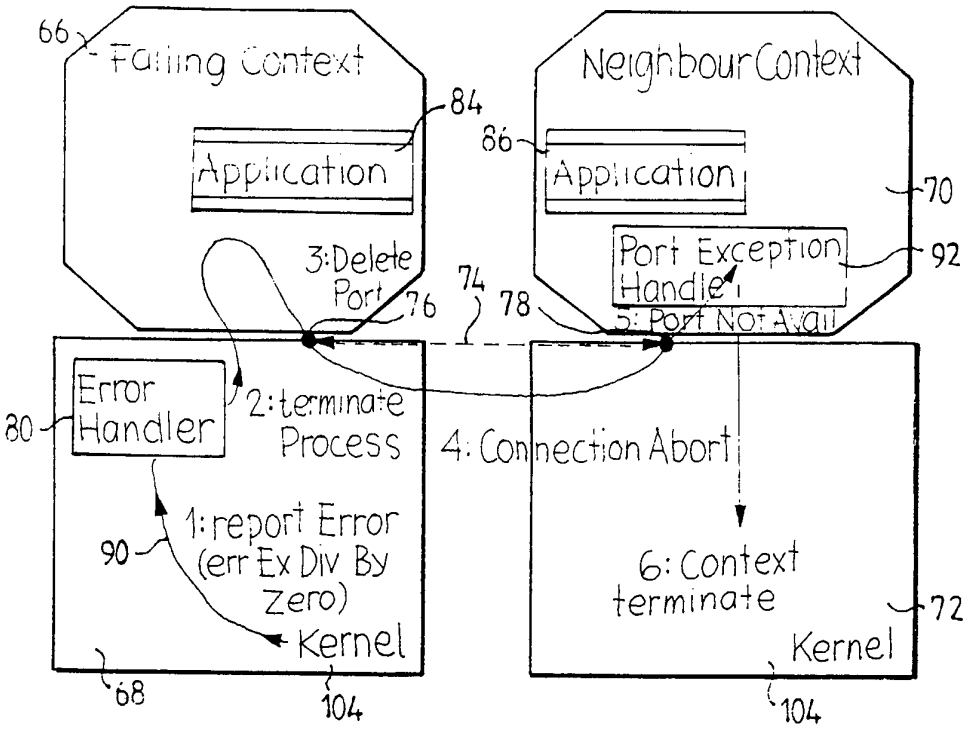

66
Failing Context
84
Application
86
Application
Neighbour Context
70
Port Exception Handler
92
3:Delete Port
76
74
78
5: Port Not Avail
Error Handler
80
2:terminate Process
4: Connection Abort
1:report Error (err Ex Div By Zero)
90
Kernel
6: Context terminate
72
Kernel
68
104
104

Fig. 11

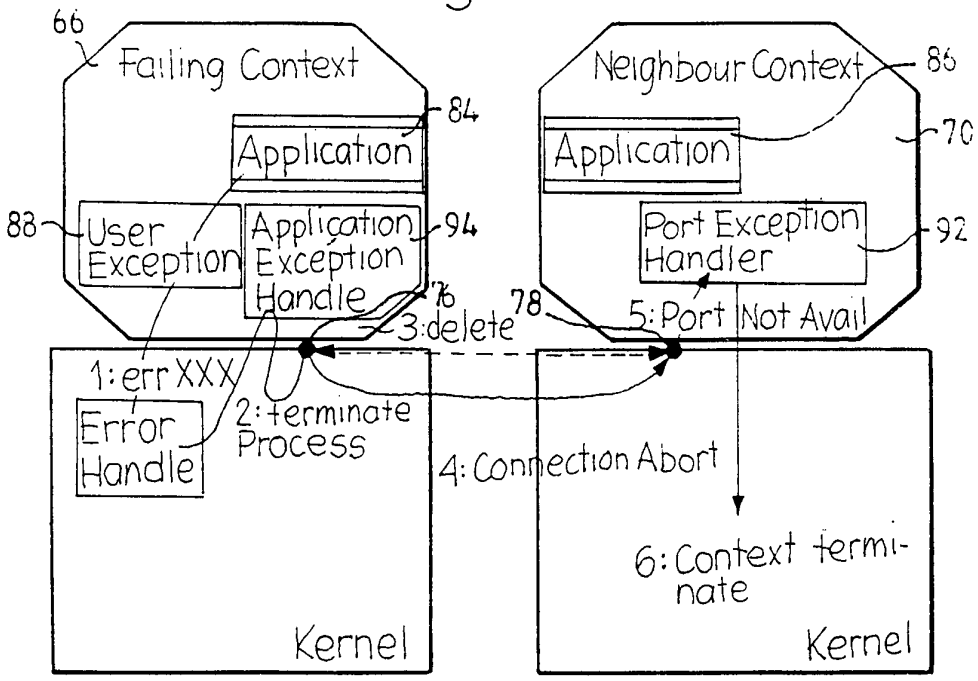

66
Failing Context
84
Application
86
Application
Neighbour Context
70
88
User Exception
Application Exception Handle
94
Port Exception Handler
92
76
78
3:delete
5:Port Not Avail
1:err XXX
Error Handle
2:terminate Process
4: Connection Abort
Kernel
6: Context terminate
Kernel

Fig. 12

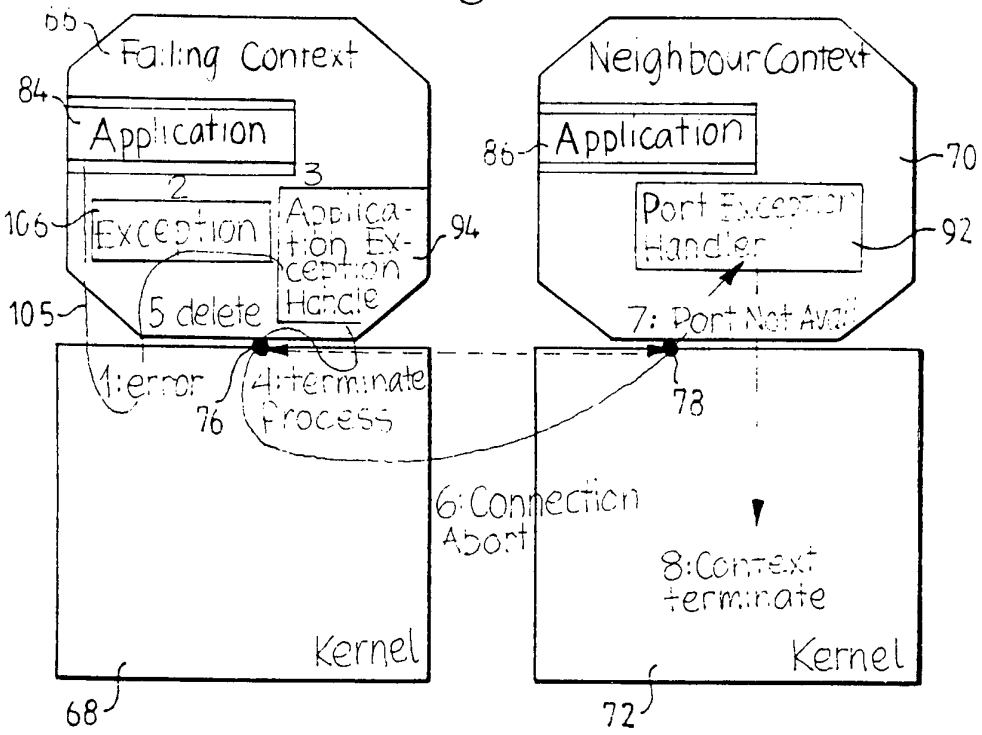

66
Failing Context
84
Application
2
3
106
EXCEPTION
Applica-tion Ex-ception Handle
94
105
5 delete
1:error
76
4:terminate Process
Neighbour Context
86
Application
70
Port Exception Handler
92
7: Port Not Avail
78
6: Connection Abort
8: Context terminate
Kernel
68
Kernel
72

Fig. 13

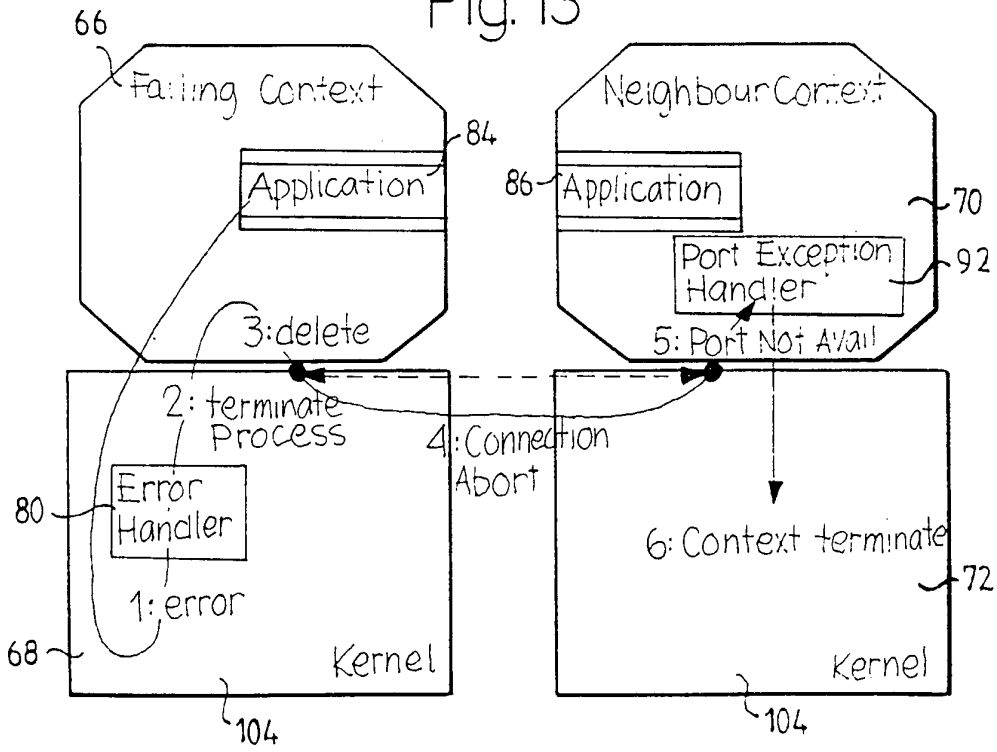

66
Failing Context
84
Application
3:delete
2: terminate Process
Error Handler
80
1: error
68
Kernel
104
Neighbour Context
86
Application
70
Port Exception Handler
92
5: Port Not Avail
4: Connection Abort
6: Context terminate
72
Kernel
104

Fig. 14

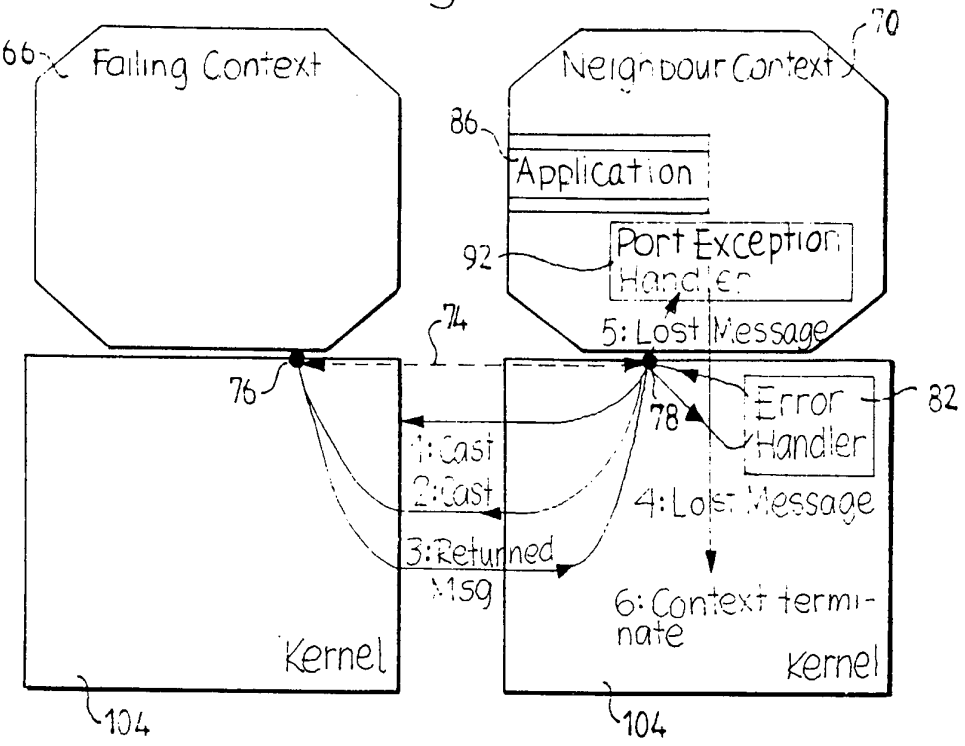

66
Failing Context
Neighbour Context
70
86
Application
92
Port Exception Handler
5: Lost Message
74
76
78
Error Handler
82
1: Cast
2: Cast
4: Lost Message
3: Returned Msg
6: Context terminate
Kernel
Kernel
104
104

Fig. 15

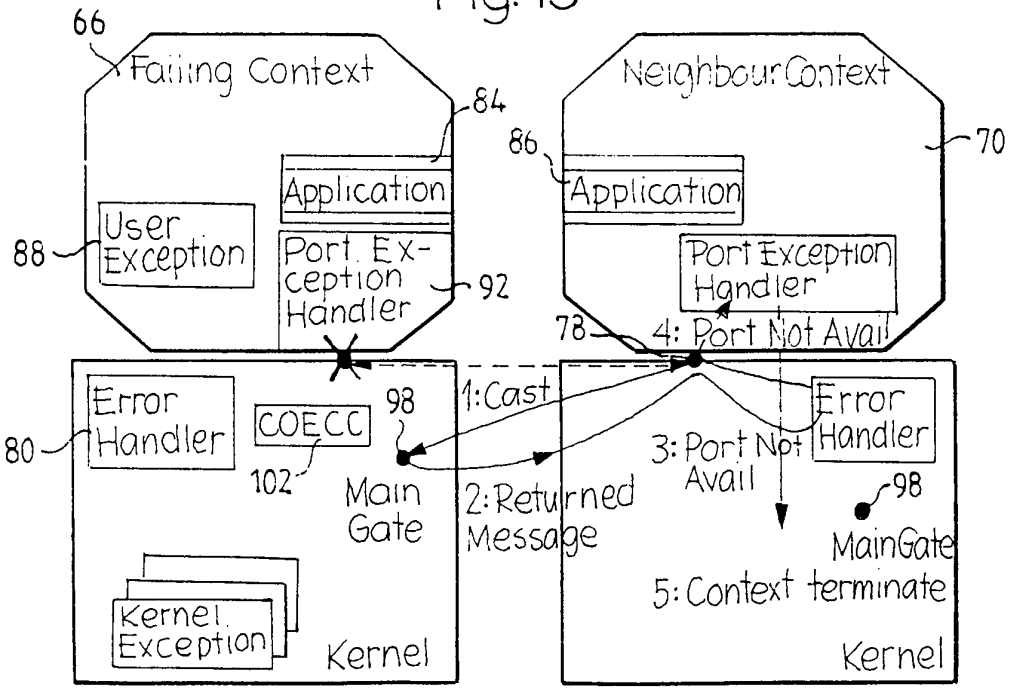

66
Failing Context
84
Application
User Exception
88
Port Exception Handler
92
Neighbour Context
70
86
Application
Port Exception Handler
78
4: Port Not Avail
Error Handler
80
COE.CC
102
98
Main Gate
1: Cast
2: Returned Message
3: Port Not Avail
Error Handler
98
MainGate
5: Context terminate
Kernel Exception
Kernel
Kernel

Fig. 16

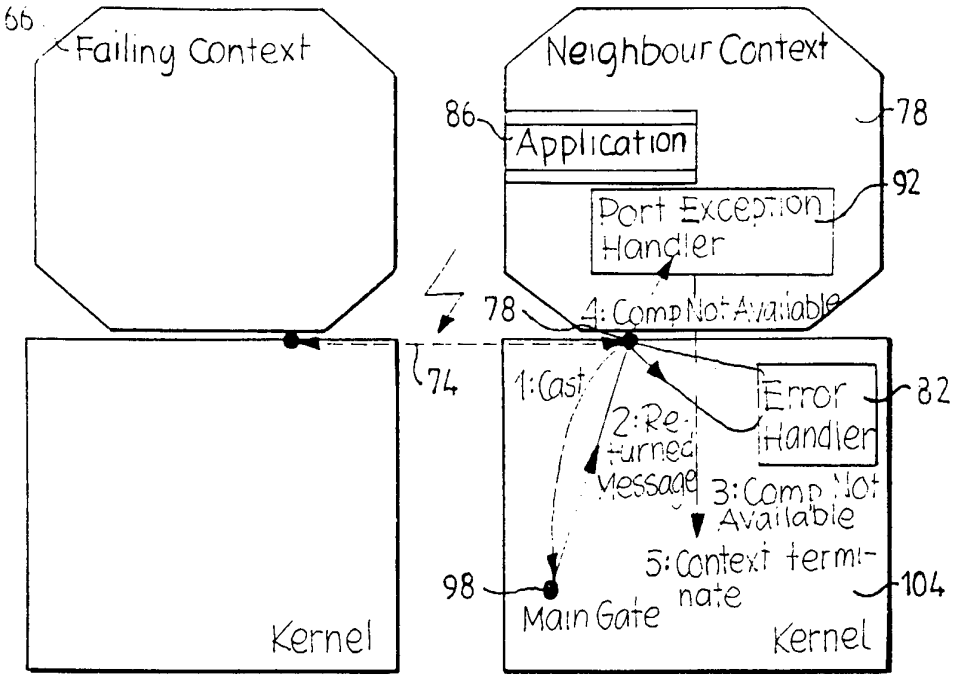

66
Failing Context
Neighbour Context
86
Application
78
92
Port Exception Handler
78
4: Comp Not Available
74
1: Cast
2: Returned Message
Error Handler
82
3: Comp Not Available
5: Context terminate
104
98
Main Gate
Kernel
Kernel

Fig. 17

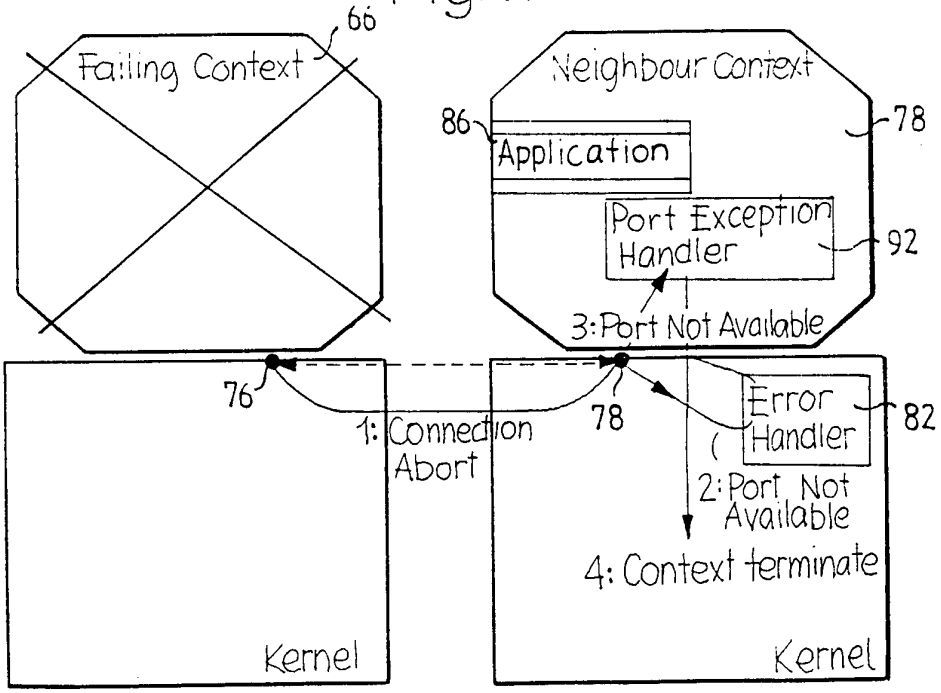

66
Failing Context
Neighbour Context
86
Application
78
Port Exception Handler
92
3: Port Not Available
76
1: Connection Abort
78
Error Handler
82
2: Port Not Available
4: Context terminate
Kernel
Kernel

Fig. 18

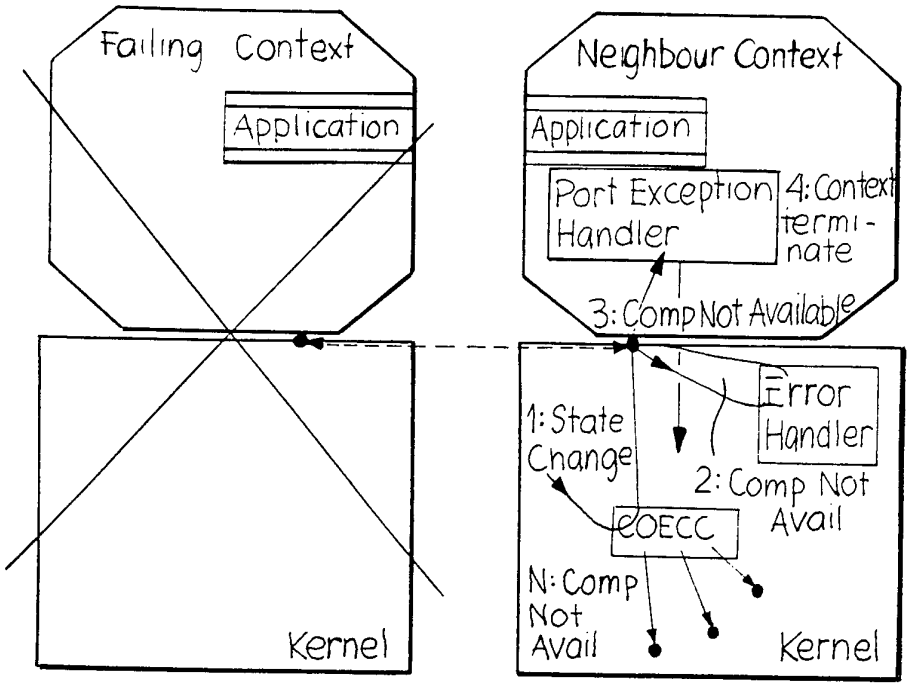

Failing Context
Application
Kernel
Neighbour Context
Application
Port Exception Handler
4: Context terminate
3: Comp Not Available
Error Handler
1: State Change
2: Comp Not Avail
COECC
N: Comp Not Avail
Kernel

Fig. 19

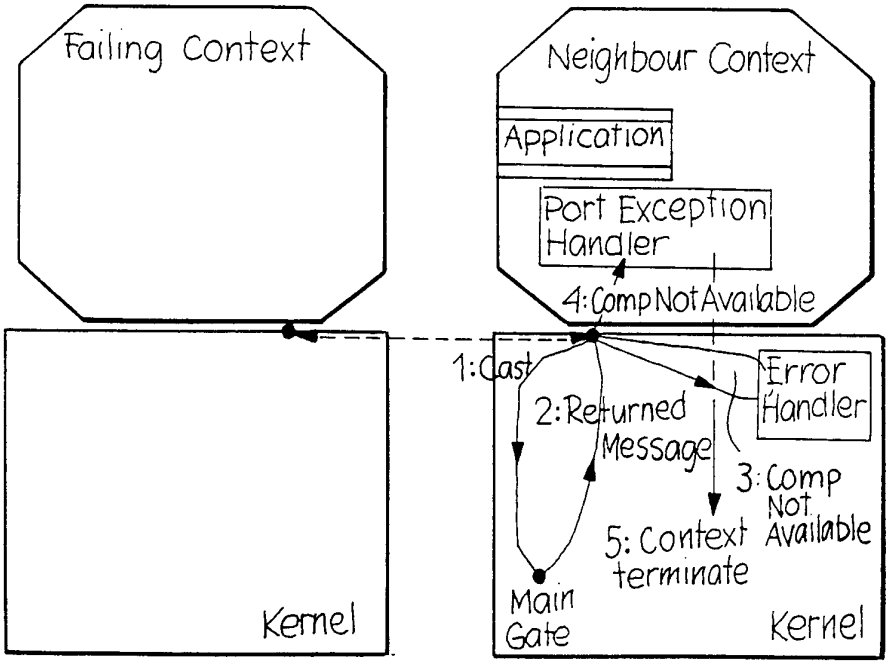

Failing Context
Kernel
Neighbour Context
Application
Port Exception Handler
4: Comp Not Available
1: Cast
2: Returned Message
Error Handler
3: Comp Not Available
5: Context terminate
Main Gate
Kernel

Fig. 20

Type of errror	Errors	Error detection	Error code to Port-Exeption-Handler	Error code to Error-Handler	Default recovery measure
Context local	Executing errors	Component/ core	———	Error specific	Terminate context
		Applic-ation	———	Error specific for current applic-ation	Terminate context
	Errors in case of system calls	Core	———	Error specific	Terminate context
Communi-cation errors	Lost message	Time release in IPC	replyNot Received	replyNot Received	Terminate context
		Time Release in the applic-ation	———	Error specific for current applic-ation	———
		Sequence enumera-	message Lost	message Lost	Terminate context
	Wrongly adressed message	Return message	remotePort NotAvail-able	remotePort NotAvail-able	Terminate context
	Dis-connected contact	Return message	remoteCom-puter Not Available	remoteCom-puter Not Available	Terminate context
		Link super-vision in IPC	remoteCom-puter Not Available	remoteCom-puter Not Available	Terminate context

Fig. 20

Type of error	Errors	Error detection	Error code to Port-Exeption-Handler	Error code to Error-Handler	Default recovery measure
Context external errors	Failured context	Link abort	remotePort NotAvail-able	remotePort NotAvail-able	Terminate context
	Failured local computer	Indication from COECC	remoteCom-puter Not Available	remoteCom-puter Not Available	Terminate context
	Failured computer in another subnet	Return message	remoteCom-puter Not Available	remoteCom-puter Not Available	Terminate context
		IPC link super-vision	remoteCom-puter Not Available	remoteCom-puter Not Available	Terminate context
	Infinite loop in a called context	Time release	replyNot received	replyNot received	Terminate context
		Link abort	remotePort NotAvail-able	remotePort NotAvail-able/Loop	Terminate context

INTERNATIONAL SEARCH REPORT

International application No.
PCT/SE 94/00079

A. CLASSIFICATION OF SUBJECT MATTER

IPC : G06F 9/46, G06F 11/00
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC : G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

SE,DK,FI,NO classes as above

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

DIALOG: CLAIMS, WPI, JAPIO, INSPEC

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US, A, 5165018 (GABOR SIMOR), 17 November 1992 (17.11.92), column 4, line 45 - column 5, line 4; column 12, line 37 - column 14, line 2 --	1-14
A	US, A, 5117352 (LOUIS H. FALEK), 26 May 1992 (26.05.92), column 1, line 46 - line 61 --	1-14
A	US, A, 5129080 (DONALD M. SMITH), 7 July 1992 (07.07.92), column 2, line 3 - line 38 --	1-14
A	US, A, 3905023 (FRANK JOSEPH PERPIGLIA), 9 Sept 1975 (09.09.75), abstract --	1-14

[X] Further documents are listed in the continuation of Box C. [X] See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" erlier document but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance: the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance: the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
5 May 1994	10 -05- 1994
Name and mailing address of the ISA/ Swedish Patent Office Box 5055, S-102 42 STOCKHOLM Facsimile No. +46 8 666 02 86	Authorized officer Katarina Fredriksson Telephone No. +46 8 782 25 00

INTERNATIONAL SEARCH REPORT

International application No.
PCT/SE 94/00079

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	GB, A, 2079997 (RAYTHEON COMPANY), 27 January 1982 (27.01.82)	1-14
	--	
A,P	EP, A2, 0539130 (INTERNATIONAL BUSINESS MACHINES CORPORATION), 28 April 1993 (28.04.93), abstract	1-14
	--	

INTERNATIONAL SEARCH REPORT

Information on patent family members

16/04/94

International application No.
PCT/SE 94/00079

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US-A- 5165018	17/11/92	CA-A- 1292323	19/11/91
		EP-A- 0274406	13/07/88
US-A- 5117352	26/05/92	DE-A- 4033336	25/04/91
		FR-A- 2655168	31/05/91
		GB-A,B- 2237130	24/04/91
		JP-A- 3194647	26/08/91
US-A- 5129080	07/07/92	CA-C- 2053344	29/03/94
		EP-A- 0481231	22/04/92
US-A- 3905023	09/09/75	BE-A- 818364	02/12/74
		CA-A- 1029131	04/04/78
		CH-A- 574646	15/04/76
		DE-A,C- 2437200	27/02/75
		FR-A,B- 2295486	16/07/76
		GB-A- 1454198	27/10/76
		JP-C- 1254410	12/03/85
		JP-A- 50073541	17/06/75
		JP-B- 59014776	06/04/84
		NL-A- 7410212	18/02/75
GB-A- 2079997	27/01/82	CA-A- 1176337	16/10/84
		DE-A,C- 3127349	16/09/82
		FR-A,B- 2486682	15/01/82
		GB-A,B- 2166271	30/04/86
		JP-C- 1460620	28/09/88
		JP-A- 57050064	24/03/82
		JP-B- 63006894	12/02/88
		NL-A- 8103136	01/02/82
		US-A- 4412281	25/10/83
EP-A2- 0539130	28/04/93	GB-A- 2260835	28/04/93