Computer Science Journal of Moldova, vol.**1**, no.3(3), 1993

# Distributed operating systems for multiprocessors and networks

E.Trakhtengherts

## 1 Introduction

An idea of parallel computational, that makes it possible to raise computers capacity and organize data exchange among the sources and destinations of information, has led to multiprocessors and networks with distributed resources appearance. They were called "distributed systems", i.e. the systems with resources distributed in space. Distributed systems gained wast ground.

Not only essential expansion of software potentialities but also a new understanding of the computational process were necessary to implement concurrent calculations. The computational process came to be considered as a totality of asynchronous processes but not as a set of consecutive actions.

Asynchronous concurrent processes may be implemented in three types of computing systems.

The first type is represented by computing complexes with common memory for a few processors. Those are well-known systems of the type BURROUGHS, ELBRUS, GRAY X-PM et al.

So, the computer GRAY X-MP includes two or four highly productive processors, a common main memory, an input-output system and external devices.

Complexes with separate memory are of the second type. They are represented by a group of processors possessing local main memories connected by a fast communicative network. The IBM corporation's computing complex ICAP/3090, for example, is the system of the type mentioned.

This is a very powerful system combining multiprocessors IBM 3090 and vector processors in various conjunctions. Connection is realized with the help of a fast bus through which data exchange takes place. Besides local memories in IBM 3090 the complexes have also a great common memory of about several hundred Mb, the direct access to the latter occuring either through special channels (in ICAP/3090, model 300) or through a common bus (in ICAP/3090,model 400). In addition, every processor has access to the whole of the disk memory.

Recently the transputers are being more and more widely spread, which the computing complexes of the second type are made of. The main reasons causing transputer efficiency as to its productivity and implementational simplicity are reasonable selection of the instruction set and the possibility of multiprocessing systems creation put in transputers.

These transputers are often called reduced instruction-set computers (RISC). An idea of an instruction-set selection is that any RISC-instruction is to be fulfilled at a clock cycle. The advantages of RISC-processors are indeed used in transputers. But the latter include usually a small number of non-RISC instructions , related to the control and message transmission.

Any transputer may be used as a monocrystal system containing a processor and a volume of main memory. In such a form they are often used in embedded systems.

To combine two transputers it is enough to connect them with a bus, because some special means for message exchange called links are built in the transputer crystal. Existing transputers have four links each. Links are a very important mechanism for systems with multiple data and instructions flows creation. Links are compatible in different types of transputers, so the latter may be unified in one system. We can choose link speed independently of processor speed, as the links of the same crystal can work with different speeds. There is no standard configuration suggested, when combining transputers; they may have quite diversed linkage topologies.

One of the most widespread 32-bit transputers is T800 which has an arithmetic unit operating on floating point numbers and achives

the speed of ten millions RISC-operations and one million float point operations.

A transputer is intended to work with its own main memory. Data exchange among the transputers working in parallel is realized through message movement. Message exchange among adjacent transputers is carried out in hardware.

To built the systems possessing maximal concurrency a topology of connections named "hypercube" is widely used.

A hypercube node contains a processor unit, a processor main memory and communication facilities. Each node of hypercube of power $n$ is connected with other $(n-1)$ nodes. All the memories are local, and remote nodes interact by means messages transmitted through intermediate nodes. Since messages are passing over a lot of nodes, transmition delay is a serious problem for the systems with hypercube architecture. The Ncube and Intel Scientific Computers companies actively take their bearing on this architecture . The latest system of the Intel company, ISPC/2, is built on the base of the 32-bit microprocessor Intel 80386 and may contain 8–128 nodes. Each node may have a main memory 1–16 Mb and a cash-memory up to 64 Kb. In each node a linkage module with eight duplex serial channels is installed, possessing a throughput of 2,8 Mb/sec in one direction. MicroVax or Sun-3 may be used as an input/output processor.

The most widespread type of multiprocessing systems with separate memory is computer networks. The networks become as usual as, for example, telephone, mail, telegraph, so there is no need to deal with their organization. It should be noticed only that multiprocessing systems with separate memory are also a sort of a network.

To organize computational process in multiprocessors and networks an extension of the operating system functions was needed. Due to this a new class of operating systems has appeared called "distributed operating systems".

The latter control resources spread all over the computer or network in a distributed (decentrelized) or concentrated (centrelized) manner.

Processors, channels, external devices, files, tasks, jobs etc. are usually reffered to as resources.

The principal characteristics of the distributed operating systems are the control of several simultaneous processes, the presence of message exchange and of synchronizing aids for these processes.

The operating systems organization in multiprocessors with common main memory has a number of substantial distinctions in contrast to their organization in complexes with separate memory and in networks. The principles of operating systems construction for these two kinds of multiprocessors are considered below.

# 2 Operating systems for multiprocessors with common memory

## 2.1 Operating system structure

When describing the operating systems it is convenient to structurize them by the levels of hierarchy. Such structurizing pursues usually two aims:

a) to determine the effectiveness level of program implementation: the lower the level the more effective are to be the programs of the level;

b) to simplify program interfaces interaction and determination: programs of the higher levels have to use services of the lower level programs.

The hierarchical structures of different operating systems are distinguished from each other. The hierarchical structure depends on the system purpose, ideology and on the hardware used for implementation.

The structure of the operating systems in multiprocessors with common memory is close to the traditional structure of the monoprocessing operating systems. Table 1 shows, as an example, a monoprocessing operating system structurized by levels for the computer MULREG with common memory.

In accordance with structurizing principles the low level of the operating system cannot use the services of higher levels. That is why the programs of the $7^{\text{th}}$ level must be resident. The programs of the $6^{\text{th}}$ level realize the exchange on the physical level and must be also resident. The programs of the remaining levels may be resident or may be inputted from external devices, depending on the demands of the system and the possibilities of its computational resources.

On the 7-th level – the level of processes – a process control program realizes process initialization, halt and synchronization.

The 6-th level – the external devices management – is rather traditional. Its characteristic property is that the concurrent processers may adress the same external device simultaneously. In this case they are put in the queue attended by a discipline given in advance.

| | | | |
|---|---|---|---|
| 1 | Applied programs | | |
| 2 | User's interface | | |
| 3 | Loader and Linkage Editor | | System subroutine Control |
| 4 | File control | Scheduling | Exclusive situation control |
| 5 | System memory management | | |
| 6 | External devices management | | External devices management |
| 7 | Process control | | Process control |
| | The 1-st processor | | The 2-nd processor |

Table 1

Other levels are purely traditional. Their functions are clear from Table 1 and need no comments.

We must notice that all processes of the levels 1–5 may run on either of the processors. This increases the system reliability, because the failure of one of the processors does not lead to the failure of the operating system as a whole. So, the operating system routines binding to the processors shown in Table 1 is not fixed.

Organizing the concurrent processes interaction is a very complex task, greatly affecting the system productivity. The time intervals spent on the processes interaction may differ many times for the same computer depending on the operating system algorithms.

So, for the microprocessor based on the Motorola 6800 using WME-BUS buses two operating systems are considered the best ones, VERSAdos and PDOS. About one million of measurements of the time spent on interaction in both systems were made. It turned out that in PDOS the interaction is fulfilled almost 21 times faster than in VERSAdos.

Distributed operating systems differ from those for monoprocessing computers in realization of the following three elements of the asynchronous concurrent processes interaction: initiation, completion and synchronization.

## 2.2 Initiation and completion of concurrent processes

In multiprocessors with common memory the task to be solved may consist of a few sections (processes) running in parallel. Inside every such section all the instructions are executed consecutively. During execution every process may initiate (under corresponding request to the operating system) one or more processes which will run in parallel with the first process if the computational resources allow it. Processes created in such a way may generate in their turn other processes etc.

The function of process generation is in many respects the same as the function of task generation but is being fulfilled faster in most cases.

At the moment of the task initiation the operating system may, although not obligatory, expreme the task as the only (initial) process possessing all the task attributes. This initial process may be represented by an organizing program, controlling all other processes of the

task without the use of the operating system mainly, but it may be simply the head process addressing the operating system for the resources needed, if new processes generation is necessary.

When initiating a process, it may be given some factual parameters which are to be processed. Therefore, the process initiation is semantically close to the subroutine call (although substantially more complex). That is why the linguistic constructs for the process initiation are like the linguistic constructs for the subroutine call. The same may be said about the process termination.

In some systems the process created may obey the master process or to the other processes of the higher level. Obeying means that the control does not re-enter into the calling process from the process created until the work with all the other obeying objects (files, other processes etc.) is finished.

After initiation the process is functioning independently, synchronizing its actions and iteracting with other processes by its own initiative or by the initiative of other processes.

To control each created process in the interval of its existence a process control block is created which contains all the information necessary for the process functioning and, in particular: the process state word; entry address; an address of the queue of the non-served demands for synchronization ; an address of the area keeping the process registers during interrupts; pointers connecting the given process control block with other such blocks of the same task; addresses of the lists of processes, files and other objects obeying the given process; the priority of the process; execution time; the readiness bits (is not ready means waiting for some event termination); information necessary for re-entry into the calling process or subroutine etc.

A queue of processes ready to run and a queue of processes waiting for some events completion are created for the control blocks. The queues may be created inside the tasks or may be common for all of the tasks. The queues are arranged in a mode accepted when generating the operating system or by the programmer's instruction. The modes may be the following: first in – first out, corresponding to the priorities and others. In the cases when the mode of the selection from the queue

is given by the linguistic construct, for example, by the widely used FORK-instruction, the operating system must garantee parallel (if the resourses allow it) or parallel-consecutive execution of the processes whose identifiers are listed in the linguistic construct.

As soon as one of the task processes is finished and a processor given to the task is released, the first block is being chosen from the queue of the process control blocks ready to run, and the processor released changes over to the performance of the process for the block chosen. If the task has no processes ready to run, the processor is given to the other task by the operating system.

If the process waits for the occurence of an event, it is put in the queue to the synchronization block (see 1.3), and its own control block is transferred to the queue of the processes waiting for the occurence of different events.

Readiness of the external device, sending the answer to the demand of one of the parallel processes, a process or device call etc. are the examples of such events.

There may be different interrelations among processes and subroutines. A subroutine may be included in the process or identified with the process. In some systems there are more complex logical relations among the processes and subroutines. On the one hand, since any task is represented by a set of parallel processes, any subroutine performed when solving the task, belongs to one or another process as a component; on the other hand, any operation on a process (the process generation including) may be fulfilled only within the scope of some subroutine. Hence, we can talk about the process obeying the subroutine generating this process.

A process when being created may obey either a subroutine generating it (by default), or any pointed subroutine of a higher level.

The case when a process obeys a subroutine is similar to the case of a process obeying a process.

There exist two ways of the program performance termination:

- a subroutine is completed normally when executing the sequence of instructions organizing return to the calling subroutine from the current subroutine;

- a subroutine is completed under abnormal conditions when processing the interruption.

At the normal completion of a subroutine a check is made first of all, if there exist the processes obeying the subroutine. If there are such processes, then the completing subroutine switches to the state of waiting up to the moment when all the processes obeying to are abandoned.

On the process exit from the state of waiting, the memory occupied by other objects obeying the subprogram in view is released, and an analysis is made whether the subprogram which has returned control is in the separate load module. If this is the case and if all the subprograms of the load module given completed their work, then the main memory segment occupied by the load segment is released; only after that the control is returned to the calling subprogram. If the control is returned to the head subprogram, the task as a whole is considered completed.

There are three ways of the process completion:

- the process is completed if the highest level subprogram of this process is completed (the normal completion of the process);
- an emergency completion, in some cases;
- completion on the special demand to the operating system (forced process completion).

In either case, at the process completion the block controlling this process and all the references to it are cancelled.

On the forced process completion the operating system halts the task solving and destroys the following data: the process indicated as the parameter (if the process is not indicated, then the current process is taken off); the processes obeying subprograms of the given process, if any; the processes obeying the subprograms of the obeying processes, if there are any, and so on. Besides the processes all other objects obeying the subprograms of the processes are destroyed, and all the segments of the main memory used for these subprograms to function are released.

The processors released change over to the other processes performance, if there are processes ready to run on the processor in this task. Otherwise the processors are redistributed by the operating system.

In the case of the forced completion of the head process of the task the latter, as a whole, is considered completed.

## 2.3 Concurrent processes synchronization

In the multiprocessors with a multiple flow of instructions a mechanism of events is widely used for process synchronization, as in the multiprogramming monoprocessors. It appeared when the first multiprogramming systems were created where input/output of information was realized in parallel with calculations.

The ideology of the mechanism of events is similar to the widely spread semaphors ideology. They may be implemented with the help of the well-known instructions of the type WAIT and POST.

Macroinstruction WAIT switches the process to the state of waiting for the completion of the needed number of events. Each event is represented by the separate event control block (ECB). In each ECB a check is made if the related event has occured. If so, then the counter of events expected in the block of demands is being diminished by 1 and then the check of the next ECB follows. If the event did not happen, it is flagged as an expected one and the next ECB is checked. If the counter of events expected is equal to 0 as a result of looking through all the ECB, then the control is returned to the point in the task (or process) program where the macroinstruction WAIT occured. Otherwize the rest of the events expected are placed in the block of demands, and the task or the process are switched to the state of waiting.

To take the task or the process out of the state of waiting the macro- instruction POST is used. The address of ECB representing the event and the event completion code are given as the operands of the macroinstruction POST. If an event was fullfiled, the counter of the events expected is diminished by 1. If the number received is not equal to 0, then the task or the process remains in the state of waiting;

if it is equal to 0 then the task or the process is taken out of the state of waiting, switches to the state of readiness and proceeds to run when the processor is given to it.

Of course, other disciplines are also possible, for example, the discipline when a task or a process is being taken out of the state of waiting for the occurrance of only one of the set of events indicated etc.

# 3 Operating systems for multiprocessors with separate memory and for networks

Multiprocessors with separate memory and multiple flows of instructions and data are created on the principle of local networks with strongly simplified protocols. Because of this, operating systems for such complexes and networks may be considered jointly, excluding the hierarchical system of protocols characteristic only of networks.

The main component of a distributed program (the program running on the multiprocessor or in the net) is usually a process, performed asynchronously and in the definite sense independently of other processes. It is nameed differently in the systems: agent, task, actor etc.

Every process is running on a computer which is a local network node under the control of its own local computing system. A local computing system proceeds under the control of the distributed computing system which distributes all resourses and organizes exchange among processors.

Operating systems for the local computing networks may be either created on the base of ready monocomputing operating system (for example, in the Ethernet and Token-ring nets) or designed anew as a single whole (for example, nets CHORUS).

The following variants of the distributed operating systems structure are possible:

I. Every computer in the local net may fulfill all the functions of the distributed operating system, i.e. to keep in its main memory the whole of the resident part and to have access to any non-resident

part of the distributed operating system kept on the external storage media.

II. Copies of the programs implementing the most usable functions of the distributed operating system are present in all computers in the net, and in one computer only(or in a few of them) – for the functions rarely used.

III. Each computer performs a definite set of functions of a distributed operating system, and these functions may differ in various computers and may overlap.

The distinguished feature of operating systems for multiprocessors with separate memory and networks in comparison with the ones for computers with the common memory is a layer of operating systems, which garantees the data exchange among computers.

The principal distinction of distributed operating systems in networks and multiprocessors from the ones in the monoprocessors is new functions appearance. The latter include: initiation and completion of the user's concurrent processes, message exchange among them and their synchronization. Let's consider these functions.

### 3.1 Asynchronous concurrent processes interaction

Asynchronous concurrent processes in the systems with separate memory (networks and transputer complexes) may interact only through messages.

The effectiveness of the message exchange for the same hardware depends on the type of exchange protocol and operating system. Table 2 shows the time of exchange with the same 2-pares of messages between two processors under different protocols and different operating systems control is shown. The experiment was carried out on two computers Sun 3/75s, connected by means of Ethernet network.

| Protocol | Operating system | Execution time (msec) |
|---|---|---|
| UDP | $x$-kernel | 2.0 |
| | UNIX | 5.4 |
| | Spite | 7.5 |
| TCP | $x$-kernel | 3.3 |
| | UNIX | 6.1 |
| | Mach | 11.0 |

Table 2

UDP (User Datagram Protocol) – the standard protocol for the Internet network, which allows an applied program running on one computer to send a message to an applied program running on the other computer.
TCP (Transmission Control Protocol) – the standard protocol of the transport level for the Internet network, which garantees reliable duplex (two-side) data flow transmission from one process to another.

The processes interaction consists in initiating some processes by others, in synchronizing and completing the processes.

A process may be initiated by a message transport to a local operating system of another network node. Interaction related to a message transport and to processing it by the processor is often called "remote procedure call". This name is connected with the fact that the sender is calling a procedure which in principle may be performed on a remote computer.

Linguistic construction of the remote procedure call is usually analogous to the traditional (sequential) procedure call.

A remote procedure call is performed in the following way: input arguments are sent to the corresponding addressee, and the calling process is delayed until the procedure called is fulfilled and results are transferred to the calling process in the form of output parameters. Therefore a remote procedure call is one of the synchronizing media.

There exist at least two approaches to a remote procedure description. At the first approach a remote procedure is commonly described as a procedure in consequtive languages. For example,

REMOTE PROCEDURE <procedure_name> (<parameters_list>)
    <procedure_body>
END.

Such a procedure is run as a process. This process waits for the calling process message, performs the procedure body and returns output parameters.

At the second approach a remote procedure is considered as an operation which may be placed at any place in the process. In the ADA language this operation may be written, for example, in the form:

ACCEPT <procedure_name> (<parameters_list>) –
    <procedure_body>.

If the call-operation is ready to start earlier than the called process is ready to accept it, then the calling process performance is delayed until the called process accepts the call.

If the reception-operator is ready to the receipt earlier than the relative call has appeared, then on its appearance this call may be processed immediately, but the calling process will be delayed till the answer is produced. In such a way the process synchronization called "randevu" is achived.

Processes completion is realized, so as in the traditional monoprocessing systems, through the message, that the process performance is stopped, sent to all interacting processes. The process may be stopped also by the message sent from the interacting process.

In the most cases each node in local networks is a monoprocessor. So the task control inside the node is usual, taking into consideration message exchange between tasks. In this case the task may be viewed as a process. Its performance is initiated by the local operating system, or after receiving a message from other nodes or tasks, or through the

instruction of the user or device to communicate with an object (for example, through the signal of the initiating transducer).

There exist two basically different approaches in the synchronizing methods. At the first approach beating is used – that is the subscription of the moments of time in which messages may be transported from one node to another in the network. By the moment of exchange all the messages specified for the given beat are to be prepared for transport, and all the addressee are to be ready to receipt. This approach found its application in specialized networks only.

At the second approach the asynchronous sending of the control messages is made, which initiates, completes and synchronizes concurrent processes. In such cases, any synchronizing operation always leads to intercomputer exchanges requiring relatively much time.

It follows that the number of intermachine exchanges needed to implement each operation of synchronization is to be minimized to increase effectiveness of the synchronizing mechanisms. Therefore all the information related to any operation of synchronization is to be situated in the main memory in the form of a number of continuous sections,as small as possible, which is achieved, for example, by special blocks creation named transfer control blocks (TCB) or exchangers.

The synchronizing methods based on the mechanism of events in the systems with separate memory, as in the systems with common memory, differ in many details which depend on both hardware and the operating systems destination and ideology. But all of them have a common peculiarity of using message transport for synchronization. The mechanism of synchronization uses three,not two operations. Besides the operations WAIT and POST discussed above (the latter is named ACCEPT in some systems), one more operation is needed to synchronize events in the systems with separate memory, which is the operation of message transport named usually PASS or SEND. The PASS-operation passes the demand on the asynchronous operation perfomance from the process, fulfilling the operation PASS, to the process addressed in the call, and switches the latter in the active state, if it is waiting for this call. When the operation PASS completes its work, the control is returned to the calling process.

To implement a call or message transport a special transport control block is created, which contains: the indentifier of the process to which this call or message are passed; call code determining the asynchronous operation called (from the number of operations performed by the addressed process); the address of the factual parameters list (if they exist); return address including the identifier of the process which passed the call; return code to transfer a message about the asynchronous operation completion. In the same block the fields of links are contained to include it in the lists of subordination and in the queues.

On getting control the PASS-operation checks if the addressed process is able to accept the call passed (i.e. if it is in the state of waiting for a call of asynchronous operation with the transmitted call code). If this is the case, then the addressed process is taken out of the state of waiting and gets the transport control block and a list of factual parameters of the call.

If the addressed process cannot accept the call transmitted then the corresponding transfer control block is put in the queue to the given process for serving. The heading of the queue for the asynchronous operations serving is in the control block representing the process.

Transfer control blocks are put in queue for to the accepting process for serving in the order of their coming in. Asynchronous operation calls may be passed to one accepting process from several calls belonging both to the same task and to other tasks without taking into account whether these processes are situated in the same physical memory or not. Asynchronous operation calls receipt is garanteed for every process in the system.

At the moment when accepting process is able to receive an asynchronous operation new call, it has to perform WAIT-operation with indication of the call code for an asynchronous operation to be served. If at the moment there is at least one transfer control block with the code given in the queue, then the first of these blocks is excluded from the queue and transmitted to the process. Then, if there is transfer control block with the given call code, the accepting process is switched to the state of waiting.

There are also variants of WAIT-operation for the transfer control block with any code receit from the list of call codes or for the first transfer control block receipt from the queue for serving without the analysis of the call code.

On completion of the asynchronous operation call processing, the accepting process informs the process which passed the call about this using POST-operation.

Owing to the mechanism described above the synchronization of the receipt and transfer of the messages SEND, RECEIVE and the like, may be made. So, the synchronizing methods in local networks are based on the message exchange.

Each message or a remote procedure call defines some actions beginning, i.e. synchronizes the sender and the addressee actions. Finally, the synchronizing methods play a particular role in the operations with distributed data bases.

Information exchange in networks demands some linguistic media and serious system support.

At least two quite independent sides take part in the exchange, as it was mentioned above: the sender and addressee. The addressee may be occupied when receiving a message and unable to accept it at once, the sender has then to delay its process performance and to make sure that its message is received by the addressee, and in the most cases it has to receive the answer with the results of its message processed by the addressee. Thus some elements of synchronization are always present in the information exchange among asynchronous concurrent processes in local networks. Message transfer always includes at least two operations: an operation in the process sending a message and an operation in the process accepting the message. Therefore in the message transport system it is inportant: who is the partner(process, module, task etc.); how the connection is established (statically or dynamically); the method of data transport (using parameters, by an assignment operation etc.); what is the object of addressing (process name, procedure name, entry name, port name etc.); conditions of message receipt (unconditional, from the definite source only, depending on the accepting process etc.); where the answer is contained (in the

reception comfirmation, in the results of the message processing) etc. Of course, it is far away from all these factors to be reflected in the linguistic constructs. They are defined partly by the system means, but in any event they find their reflection in the programming system.

Operations of sending and accepting messages contain correspondingly reserved words of a language, the sender and addressee addresses and a list of variables whose values are transferred from one process to another. Some examples of the simplest linguistic constructs for operations of message sending and accepting are given below:

SEND <variable_list> TO <addressee–acceptor>,

where <variable_list> contains values of variables or data arrays at the moment of message transfer, and <addressee–acceptor> defines the addressee of the message and the operation sending the message

RECEIVE <variable_list> FROM <addressee–acceptor>

where <variable_list> does not demand any commentary, and <addressee – acceptor> is the message sender address and the operation-destination.

The SEND-operation transforms the message into the form corresponding to the protocol of information transfer along communication channels adopted in the distributed system, but the operation RECEIVE carries out transformation of the message received to the form which conforms to the given node demands.

Conditions may be imposed on the receipt, for example, with the help of the operation WHEN $\mathcal{L}$. Then the message will be accepted only in the case if the value of the logical variable $\mathcal{L}$ is true.

When interacting the processes may explicitly refer to one another. Explicit reference means that any applyed or system process transferring information must indicate explicity the process-addressee name.

Such method of addressing gives rise to definite complications when introducing new processes with new names into the system, but it is simple enough for implementation, therefore it is used in many systems, for example, in DECNET, VAX/VMS, in ADA language etc.

To escape explicit identification, the communication intermediates, or ports (buffers in the main memory with names), are introduced in some systems. Ports introduction allows one to indicate port names only in programs. Messages may be transported from one port of a process to one or a few ports, depending on the addressee number. Ports may be "two-way ports", i.e. may both accept and send messages. For example, the CHORUS operating system may create, open, close and cancel ports by means of operations of the type:

**CREATE PORT** (port_name) – to create a port, i.e. to allocate memory sector for a buffer (port), to enter its identifier into the operating system catalog etc.

**OPEN PORT** (port_name) – to open a port, i.e. to make it accessible for the process

**CLOSE PORT** (port_name) – to close a port, i.e. to make it unaccessible for the process

**DESTROY PORT** (port_name) – to cancel a port.

Distinguishing of port opening and closing operations allows one to open ports with the same name in two different processes consecutively. The possibility to create and to cancel ports in the process of computation makes it possible to realize dynamic system configuration and to economize on the main memory. Nevertheless, in many systems static port creation is used, because dynamic port creation and cancelling make port and data transfer control complicated.

In distributed systems it is important to provide maximal productivity of intermachine data transfer channels. The channel characteristics in many respects define the distributed system time reaction, the measure of concurrency and the other important parameters in many respects. Therefore the broadcasting way of information transfer is spread in distributed systems by which information from one source may came in a few receivers at once. Notice, that dynamic port creation and cancelling complicate the broadcasting transfer implementation.

## 3.2 Network protocols

In the process of impetuous network software progress two tendencies have appeared: software structuring (the definite hierarchy creation) and standardization of different kinds of algorithms and data transfer conventions.

To achive the aims mentioned considerable efforts were applied and certain success was achieved. International standardization organization (ISO) proposed an hierarchical structure of network software development and worked out partly the standards of system conventions, formats and algorithms for every level of the hierarchy. This structure and protocols were called "Standard model for open systems interaction". The term "open" means that all the devices and processes of the system (networks) interact according to some set of standards and are open therefore to interaction with other networks. This approach is due to the necessity to link in one system a great variety of technical devices and programs used in networks.

It should be noticed that hierarchical structuring is not something unusual in programming. It is traditional for program complexes development. Hierarchical structuring makes independent program design, local modification and/or replacement of the programs possible. In networks it allows one also to organize processes interaction relative to the levels of hierarchy, providing flexible networks construction.

A new notion, protocol, is introduced for networks software. It is a special convention about formats, algorithms and methods of synchronization of the instructions and data transfer and receipt in networks. Protocols define the order of different divices and network programs interaction. So the protocols are partitioned by the levels of hierarchy.

Each level of a protocol implements a definite set of data transfer functions in networks. It is very important to notice that every level of protocol, except the most upper one, provides functioning of the nearby upper level.

In the standard model seven levels of hierarchy were specified. The number of levels is not some mystical number. Since networks had been created long before the standard model appeared, there are networks

with more or less than seven levels. Network protocols are divided into seven levels due to purely pragmatic considerations. Such division proved to be the most convenient one and became the standard that all the net designers are aiming at. The delivers of networks and data transfer media are strongly oriented on the standard model of the open system interaction. Many firms began to produce apparatus implementing protocols of "the standard model for open systems interaction".

In different existing local networks various protocols and services are used. Different partitioning on levels is carried out in them. In many of the nets it is close to "the standard model for the open systems interaction". As it was already mentioned the idea of hierarchical structuring is traditional for program complexes development.

It is neccessary to note that the collection of protocols and services on every level is superfluous. Therefore in every real network some subset of it is being chosen.

Moreover, the protocol functions of different levels of standard model may overlap. In the most of real networks the functions on each level are chosen so as not to overlap.

Let's consider hierarchical structure of the standard model of opened systems interaction.

The highest, 7-th level, called applied level, does not give services to the other levels. Its main task is to provide various forms of interconnection for applied processes belonging to any computer or to some of them. They may be the following: text messages exchange management; file control, access and transfer media; means for task transfer and manipulation, virtual terminal management, means for transact processing etc.

The 6-th level is the representing level. It transforms data structures into a standard form used in the network and vice versa. Its main purpose is to provide applied processes independence of the differences in the forms of data representation.

For example, some means may be created on this level making it possible to use Japanese writings (each symbol on this level is represented by 16 bits). The aim of the level introduction is to define a

unified protocol which could permit either message syntax independing of whether or not the latter is standard. If the systems exchanging information are aware in advance about the message syntax and semantics then only exchange with the name of the given syntax and semantics will suffice for them. In other words, the level garantees that the data the devices exchange, will come in on the upper, applied level or on the end-user's divices in the understandable form. This makes it possible to use different collections of data formats in different sets of hardware without sacrifice of mutual understanding.

The 5-th, perfomance level is intended for organizing and carrying out the dialog among applied processes. It organizes two-way simultaneous data exchange and two-way by-queue one, sets so-called main and secondary synchronization points.

If two applied processes in the traditional computer are to exchange information, this may be realized according to the form of exchange set in advance. Function of the performance level is to set such kind of link. It is called usually a performance.

The 4-th, transport level, provides transparent, reliable data transport from one end-node to another end-node making the above situated levels free of the task on reliable and economical data transfer. Finding and, in some cases, correcting errors in packages, multiplexing a few transport links onto one network link, fighting with overloads etc. are among its functions.

The 3-rd, network level, provides route organization, virtual links support and independence of the levels situated above on the data transfer methods and the functions of data retranslating and routing. It masks all the peculiarities of real data transport means off the transport level. The network level provides connection among different networks. Note that in small local networks the network level fulfills only the last function, and as in such networks a common transferring environment is often used, there are no transite nodes and therefore there is no necessity of routing.

The 2-nd, channel level, is intended for block (package) transfer through the physical network connections. It provides means for setting, supporting and disconnecting the channel links among network

objects. An important function of the channel level is to find and correct errors in channel links. On this level data selection occurs in the local networks with the common transfer environment, i.e. the routine for sampling the data blocks received by the systems through the addresses of destination.

The 1-st, physical level provides mechanical, electrical, functional and procedural standards for systems conjugation by use of physical data transfer means.

It must be noted that partitioning on levels by itself is only an element of standardization. Protocols are developed for every level. In many cases a few protocols are developed for every level.

It is by no means in all industrial networks that the software is divided into the enumerated seven levels. But this is the standard, approached by the network designers at present time.

The delivers of systems, network means and data transfer means are strongly oriented on the standard model of the open systems interaction. Many of the firms began to produce the repeates by the first level protocols, the bridges by the second level protocols, the routing means by the third level protocols and the servers by the upper level protocols.

### 3.3 Networks integration

Quite recently networks were a poorly investigated territory. Those being encroached on it were making this at their own risk. Now it is considerably investigated and strict standards appeared for it. And the world of networks became by no means simpler, but only more regulated. But the main point is that the role of network has changed. The integration of computers in a network and networks integration opens great new possibilities in computers use, but demands new hardware and software application as well.

An enlargement or partitioning of the network is made with the help of servers (they are called also gateway or bridges).

A server allows:

- to increase the network length and the number of nodes in it;

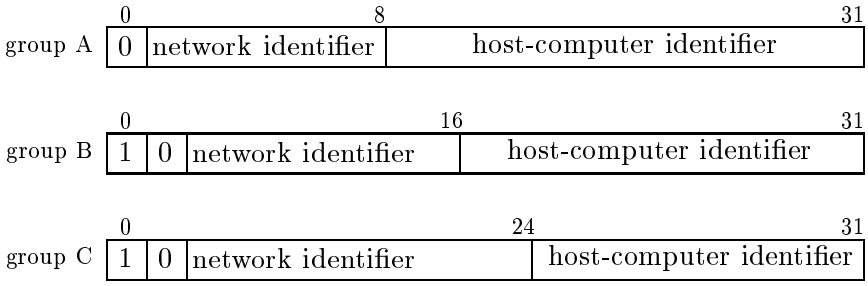
- to decrease traffic in every net segment due to data flow filtration.

Let's consider networks integration by the example of the Internet network.

Internet's servers are subclassified into central and local ones. Central servers are under the Internet Operational Centre control. The number of them is not great. Central servers keep information about all existing addresses of Internet and all information exchanges passing through them. Exchange among the central sluices occurs following the special sluice-sluice protocol.

Local server control is done by different control systems of independent networks responsible for communication withinf them (see Fig. 1).

In Internet an address includes a network identifier and a host-computer identifier. There exist three groups of networks with the following address formats:




So, depending on the network class the number of networks supported and the number of host-computers may vary. Internet falls into the class B.

A central server assures connection with one or more central sluices and, as a rule, with a few local ones (Fig. 1). Sluices classification on central and local ones helps to reduce the volume of service information circulating in the network, because only the central servers keep information about the whole network. The central server protocol includes three main parts:

1. **Protocol of the access to a neighbour.** One central server is permitted "to ask " another central server to be its neighbour, i.e. to establish direct communication between a pair of central servers.

2. **Achievability protocol.** It ascribes to a server a check of its neighbour's normal operation and achievability.

3. **Network achievability messages.** These are messages permitting to set new routes among a central server and its neighbours.

Central server protocols are "interurban" protocols, they provide information transfer among neighbouring sluices, which may be hundreds of kilometers apart and may exchange information, because they may exercise monitoring over the information exchange rate.

One of the peculiarities of Internet is that the central server message may be passed to a neighbouring central and to a local server only, so that it goes through one passage exactly between servers and the time of the message existence in the channel of communication is not too long.

As a consequence, there is no need in distance metrics use in the messages concerning the changes of message transfer routes. Indication of the route length reinforces the route existence only. So, the Internet's communication protocol is not a routing aid but only an aid for defining the addressee achievability. This is because the Internet's subnetwork topology is a tree with a central server as a vertex. Therefore there are the following restrictions for a protocol:

- the overall information achievability is violated in Internet if the central server has failed

- a central server may be aware of only one way only to an independent system, although there may exist a few physical ways

- a protocol has no cocern to the load balance among several servers to an independent system.
  It must be done by the independent system itself which may be connected with a few central servers.

- difficulties may arise when the physical channel between a central server and an independent system fails, because only one of the central servers may choose a new shortest way and declare it along the network.

The protocol restrictions listed above are important and may demand the revision of the protocol.

Local servers of an independent network are exchanging information about routes. After the routes within the local network are set, the network is considered created and declares itself to other networks through the central server it is connected with. There is no single protocol of exchange among local servers, because in different networks too many various topologies and apparatus are used. Nevertheless the most spread protocol of routing is Routing Information Protocol (RIP). We note that a server may use various protocols simultaneously. So, a central server may perform information exchange with another central sluice by one protocol and with an independent network by the other. RIP was initially designed at Berkley University, California, to provide local network functioning. The protocol was gaining wide acceptance because of the operating system 4.X BCD Unix popularity, where it is used.

## Conclusion

So, a distributed operating system differs from an operating system for monoprocessor in that in every moment it controls not one but a few asynchronous processes of the user performed simultaneously and on different processors.

Stormy development of these operating systems, especially for networks, is connected with the comparatively simple assembly of a network because of strict standardization of the hardware used and system software conventions (protocols), with the relative ease of data transfer, low cost, possibility to use remote expensive computing resources etc.

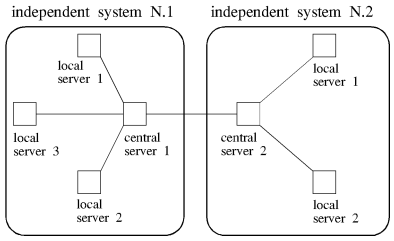



Figure 1

# References

[1] Clemonti E., Logan D., Saarinem J. ICAP/3090: parallel processing for large-scale scientific and engineering problems. IBM system journal vol. 27, N. 4, 1988, pp. 474-509

[2] Crimsdall C.H.R. Distributed operating systems for transputers. Microprocessors and microsystems vol. 13, N. 2, 1989, pp. 79-88

[3] Day Y.D., Zimmerman H. The OSI reference model//Proc. IEEE. Dec. 1983, vol. 71, N. 12, pp. 1334-1340

[4] ISO. Basic reference model for Open Systems Interconnection. ISO 7498, 1983

[5] Knowles A., Kantenev T. Message passing in a transputer system. Microprocessors and microsystems vol. 12, N. 2, 1989, pp. 113-124

[6] Mullender S.J., Rossum G., Tanenbaum A., Renesse R., Staveren H. A distributed operating system for 1990 s. Computer vol. 23, N. 5, May 1990, pp. 44-53

[7] Trakhtengherts E.A. Software for concurrent processes. Moscow: Nauka, 1987 (Russian)

[8] Trakhtengherts E.A. Network protocols . Automation and remote control, No.12, 1990 (Russian)

[9] Wexler R., Prior D. Solving problems with transputers: background and experience. Microprocessors and microsystems vol. 13, N. 12, 1989, pp. 67-69

[10] Zambre R. Design considerations for extended local area networks. Proceeding of the 10 international conference on computer communications. New Delhi, 4-9 November 1990, pp. 432-442

[11] Zimmerman H., Gullemont M., Morriset G., Banino J.-S. CHORUS: a communication and proceeding architecture for disrtibuted systems. Rapport de recherche N 328, 1989, INRIA PARIS, FRANCE

Prof. E.Trakhtengherts
Institute of Control Sciences,
65 Profsoiuznaia str.,
Moscow, 117806, Russia

Received April 18, 1993