A. Yu. Burtsev, L. B. Ryzhyk, Yu. O. Timoshenko

# OPERATING SYSTEM ARCHITECTURE BASED ON DISTRIBUTED OBJECTS

## Introduction

Today the computing power of the networks of workstations has become comparable to that of modern supercomputers. The practical implementation of high-performance distributed computations obviously requires a special software platform. At present, this platform is usually implemented by application-level parallel programming libraries, providing software developers with distributed communication and synchronization facilities [1,2]. Our claim is that the problem can be addressed in full measure by an operating system providing processes with transparent access to all resources of the distributed system. Such operating system would allow users and application programmers to think of the computer network as a single multiprocessor with common memory.

This paper presents the architectural design of a distributed operating system aimed to make the computing potential of the network immediately available to application programmer.

## Fundamental Concepts

The proposed operating system architecture, called E1, is based on the following fundamental concepts:

**Single system image support.** The single system image abstraction implies that all distributed system resources are uniformly accessible from any of its nodes. For users and software developers such system looks and feels like a single virtual computer. Its distributed nature is transparent to applications.

**Efficient access to resources.** In a distributed environment execution threads and resources they use can be located in different nodes. However, efficient access to resources can occur only locally. E1 enables access locality by making an object physically distributed, i.e. storing a complete or partial copy of its state in several network nodes. The internal organization of an object allows minimizing interactions with remote nodes during execution of operations on objects, thus neutralizing the influence of communication latencies on overall system performance.

**Load balancing.** The E1 operating system supports load balancing, i.e. dynamic distribution of computational payload between system nodes.

**Support for redundancy mechanisms.** E1 allows applications to transparently use distributed algorithms for redundant data storage and execution.

**Component model support.** E1 has a component architecture and supports component-oriented software development paradigm. Both operating system services and application software are developed within the framework of a single E1 component model. The component model provides the following services:

- global naming service;
- protected object interaction facilities;
- late binding mechanism;
- persistent objects service.

## Distributed objects

The E1 architecture is based on the abstraction of distributed object. Distributed object is an object, globally accessible from all nodes of distributed system. The important E1 properties like efficient support for reliable distributed computing and persistence are based on the corresponding properties of distributed objects.

**Object** is an encapsulated abstraction, including state information and well defined access protocols.

The **E1 distributed object** is an object having the following additional properties:

1.**Accessibility through interfaces.** A distributed object exposes one or more well defined interfaces, consisting of methods. The object can be accessed by means of method invocations only.

2.**Global accessibility.** Each object has a unique identifier associated with it. Any other object possessing this identifier can invoke the given object's methods from any node of the system, provided that it has sufficient capabilities.

3.**Ability to replicate.** The object's state is physically distributed among its replicas. Object replicas can exist in several nodes and must exist in all nodes where object's methods are invoked. Replication is a process of distribution of object's state among its replicas and replica synchronization, aimed to support integrity of the object.

4.**Persistence.** Persistence is the ability of an object to exist for unlimited time, irrespectively of whether a system functions continuously. The object is at any time accessible through its interfaces.

The E1 distributed objects are composed of ***local objects*** (Figure 1). A local object is constrained to one node of a distributed system. Like distributed objects, local objects are accessible through interfaces.

In a trivial case when the distributed object has only one replica, it is identified with a single local object, ***semantics object***. Semantics object contains the distributed object state, exposes the distributed object interfaces and implements its functionality.

If the distributed object has several replicas, a copy of the semantics object is placed in each node, where

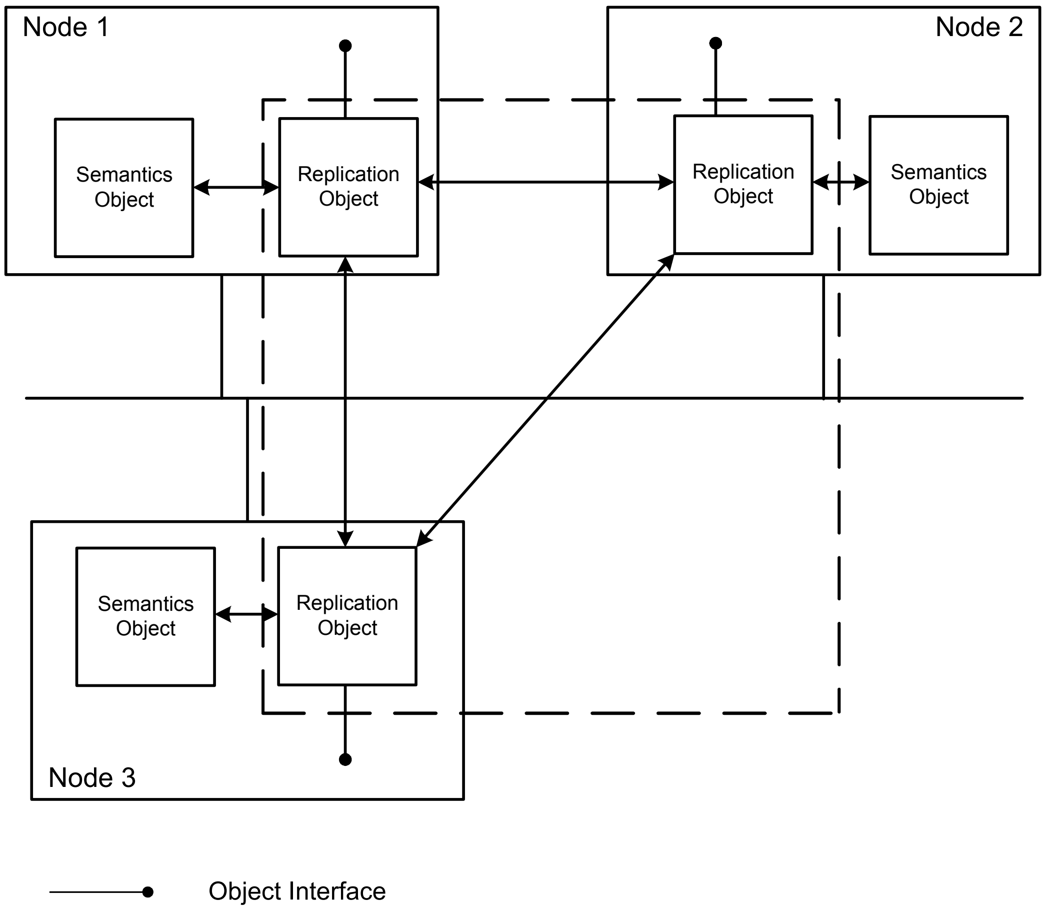

Figure. 1. Distributed Object

the distributed object is represented. The distributed object integrity is maintained by *replication objects*, complementing the semantics objects in each node. Replication objects implement the distributed object replication protocol. The distributed object interfaces are locally exposed by replication objects. While processing the distributed object invocation, replication object can refer to the semantics object to execute necessary operations over the local object state, as well as communicate with remote replication objects to perform synchronization and remote execution of operations.

## Local objects

A local object (Figure 2) consists of *interface section* and *data section*. The interface section contains references to object's method tables. The data section contains object's private data that represents its state.

## Class objects

Classes of local objects in E1 are described by objects of the special type – *class objects*. Encapsulation of class properties by objects allows implementing dynamic class loading. Class objects provide the following functionality:

- expose methods for creating and destroying instances of the given class;
- store interface implementations.

## Replication

The E1 distributed object architecture follows the principle of policy and mechanism separation that was first proposed by the developers of Hydra operating system [3]. The operating system provides the replication mechanism while specific replication strategies are implemented by replication objects. This architecture allows applying to each object the most efficient replication algorithm that takes into account its semantics.

Below we briefly describe several widely used classes of replication algorithms.

**Client/server replication.** Client/server is a trivial replication strategy. A single copy of the object state is maintained by a *server* replica. Other replicas are *clients*. All client invocations are forwarded to the server. This strategy is in most cases inefficient, since it does not provide local access to resources. Another disadvantage is low reliability due to centralized access to objects.

**Master/slave replication** is an extension of client/server strategy. Each replica stores a copy of an object state. One replica is assigned as *primary*. Read operations are executed locally in each node. Modifications are forwarded to the primary replica, which executes the required operations and updates all other replicas. For this purpose the new object state or information about state changes is broadcasted to all secondary replicas.

**Active replication.** Each replica stores a copy of an object state. Both read operations and modifications are performed locally in each node. To ensure replica consistency modifications are broadcasted to all replicas.

**Copy invalidation.** This strategy was proposed by Li [4] as a distributed shared memory coherence algorithm. It provides strong consistency of an object, i.e., each read operation returns the value written by the last modification.

**Release consistency.** This strategy relies on two synchronization primitives: *acquire* and *release*. The *acquire* operation returns a copy of the object for exclusive use. *Release* operation finalizes the set of operations on the object. This strategy requires the developer to explicitly indicate the beginning and the end of each critical section. Therefore, it is most appropriate for objects, which require mandatory access synchronization irrespectively of whether they are used in local or distributed environment. For example, an object representing a shared memory region can expose ***acquire*** and ***release*** operations for access synchronization.

**Migration.** Migration in E1 refers to the transfer of object replica between nodes. Migration is not an independent replication strategy. It is used in conjunction with other strategies to improve the efficiency of access to resources by means of load balancing.

## Architectural overview

The E1 operating system consists of a microkernel and a set of distributed objects acting at the user level. The microkernel supports a minimal set of primitives that are necessary for operating system construction, such as: address spaces, threads, IPC and interrupts dispatching. All operating system and application functionality is implemented by objects.

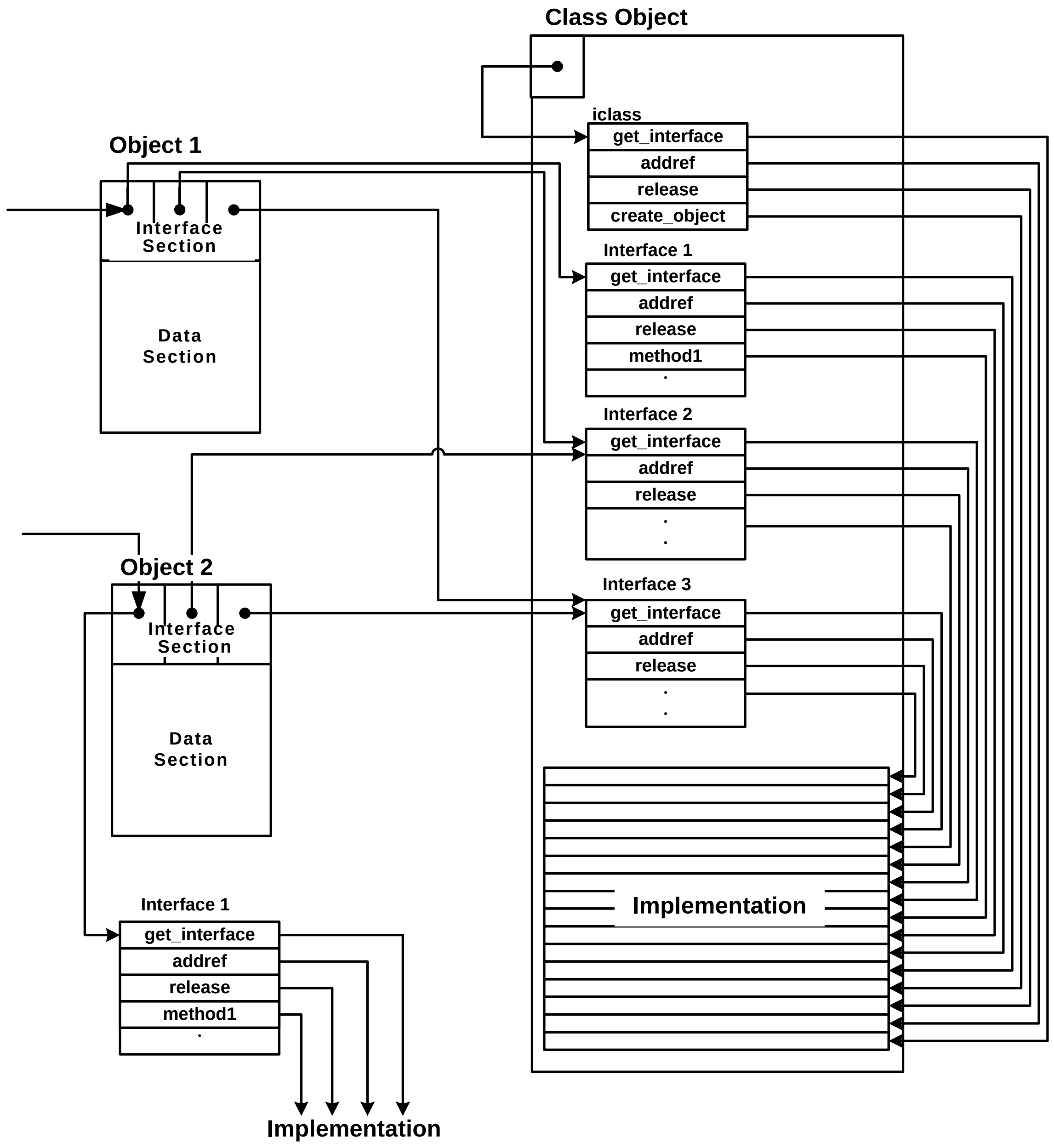

Figure 2. Local Object

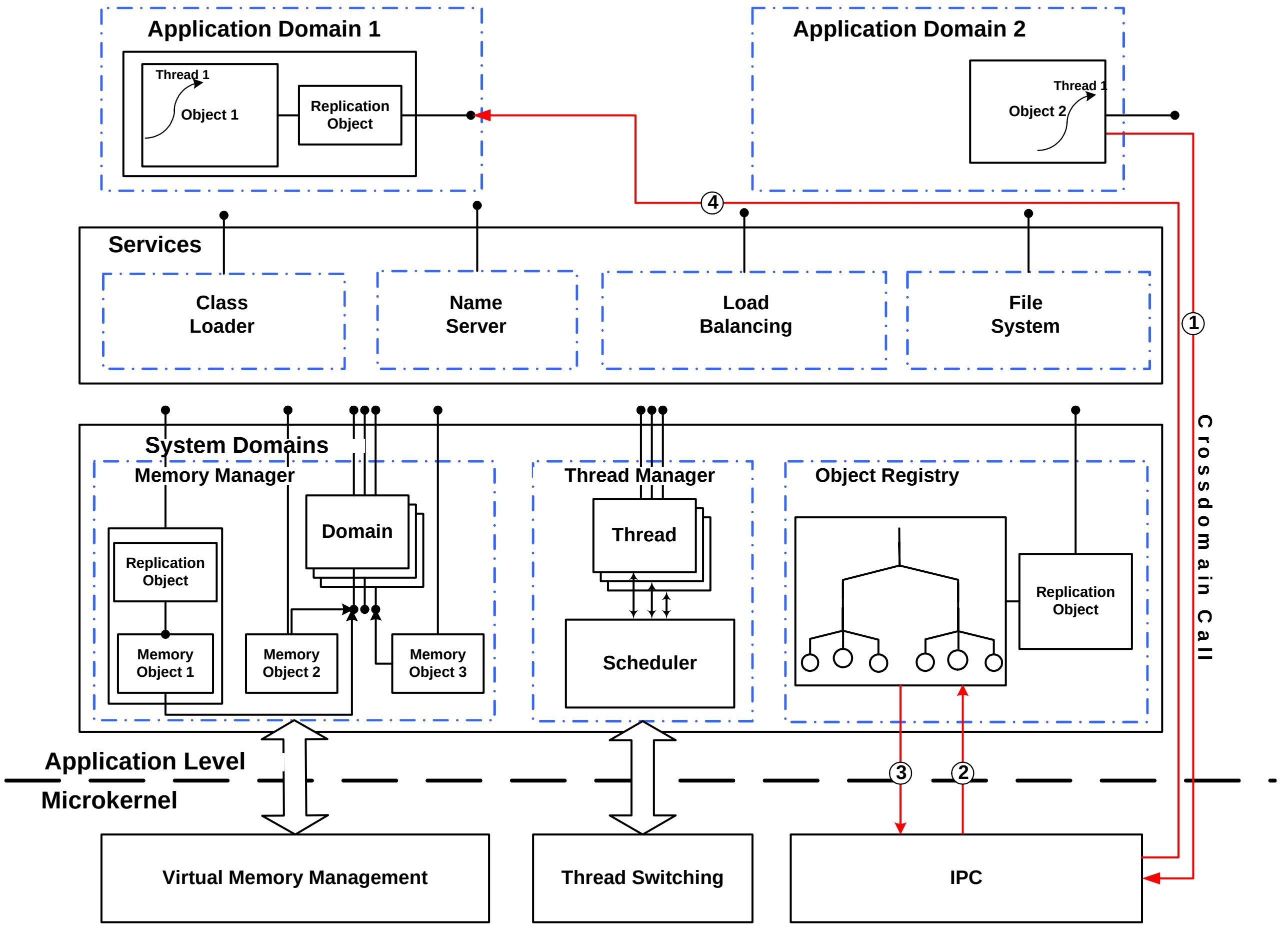

Figure 3. E1 Architecture

## Protection domains

All objects in E1 are located within protection domains. Domain boundaries are shown by dashed lines on Figure 3.

**Protection domain** is an object that isolates one or more objects from other objects in the system. Domains implement object protection and ensure that objects can communicate only by means of method invocations. Implementation of domains varies for different hardware architectures. For instance, on Intel x86 [5] architecture domains will be implemented as separate address spaces. In contrast, on 64-bit processors (MIPS [6], IA-64 [7]) a single address-space memory model will be used; and a domain will be represented by a set of virtual memory regions accessible to threads running within the given domain.

A common domain must be used by intensively communicating objects which jointly implement some functionality. For instance, each of the three basic E1 subsystems: *Memory Manager*, *Thread Manager* and *Object Registry*, uses a separate domain.

It is necessary to distinguish between domains as objects and domains as resources. These entities are subjects to the following relations:

- the domain-object controls the domain-resource;
- the domain-resource can be accessed through the domain-object;
- the domain-resource is a part of domain-object's state.

## Crossdomain calls

Objects located in different domains interact by means of crossdomain calls.

**Crossdomain call** is a protected invocation of an interface of an object located beyond the boundaries of the calling object's domain.

A sequence of steps involved in crossdomain call is shown by arrows on Figure 3. The calling object (*Object2*) transfers control to the microkernel through a special system call and supplies the following information: an identifier of the object to be invoked (*Object1*), required interface and method identifiers, and invocation arguments.

In order to complete the invocation, the kernel requires the following additional information:

- whether *Object2* is authorized to invoke the indicated method of *Object1;*
- the identifier of the domain containing *Object1* and the required entry point address.

This information is stored in a special system object – *Object Registry.* The microkernel interacts with Object Registry to obtain the information about *Object1's* interfaces and to validate the caller's rights to perform the operation.

If the invoked object does not have a local replica yet, Registry initiates its replication to the local node.

Finally, the microkernel transfers control to *Object1* to execute the call. The return from invocation also occurs through kernel.

## Access control

E1 uses a *diminished take-grant* access control model [8] proposed by Shapiro as a generalization of the classical *take-grant* capability model. This model provides a number of properties that can't be implemented in the access control list based systems.

**Least privilege.** It is possible to grant to each subject only those capabilities that it requires.

**Selective Access Right Delegation.** A subject that possesses certain capabilities is able to selectively delegate those capabilities to other subjects.

**Rights Transfer Control.** A subject should be able to receive additional authority only if that authority is granted via an explicitly authorized channel.

**Endogenous Verification.** It must be possible to verify from within the system itself that certain restrictions on rights transfer and information flow are met.

In the E1 access control model capabilities belong to distributed objects. Each capability describes owner's rights to invoke certain interfaces of another object.

In each E1 node Object Registry stores objects' capability lists and enforces access control policy. It exposes *take* and *grant* methods that constitute the only way to pass capabilities between objects. During each crossdomain call Registry verifies whether the calling object possesses a capability to invoke the specified interface of the target object.

Registry itself is a distributed object. Its replication strategy is responsible for maintaining consistent information about objects and capabilities in different nodes.

An object can obtain a capability for some other object, which does not have a replica in the local node. On the first invocation of such non-existent object Registry turns to Global Naming Service to locate target object's contact points and initiate its replication to the local node.

## Threads

In E1 objects are passive entities, i.e. an object does not have any threads permanently associated with it. Instead, E1 supports a migrating threads model [9] in which during a crossdomain call execution of the calling thread is transferred to the target object. As shown in [10], migrating threads are more appropriate for object-oriented environment than traditional static threads. Static threads result in significant overhead for the systems consisting of interacting objects of medium granularity, since at least one thread must be associated with each object for dispatching incoming messages.

## Memory management

The E1 memory management system is based on the concept of ***virtual memory objects*** introduced by Mach [11]. All used virtual memory of a domain consists of continuous page regions – virtual memory objects. Each memory object can be mapped to one or more domains in different nodes. Memory objects control the mapping of virtual memory pages into external memory. Main memory is viewed as a high performance cache for the data in external storage. Associated with each memory object is a pager object, supporting the exchange of pages between external and main memory.

Efficient distributed shared memory support is an important component of distributed operating system. In E1 shared memory is implemented by memory objects. Memory object is a unit of memory sharing. Memory coherence algorithm is implemented as memory object’s replication strategy. By using different replication strategies the distributed shared memory parameters, such as consistency guarantees and efficiency of access, can be adjusted to specific tasks. By default, the copy invalidation strategy, providing strong consistency, is used.

## Class Loader

Classes of local objects in E1 are described by objects of the special type – Class objects. Class object stores interface implementations, assigned to instances of the given class upon their creation. All local objects in the system are created by appropriate Class objects. Before creating an object in a certain domain, an instance of the corresponding class must be loaded to this domain. The loading of Class objects to domains is performed by a special system service – *Class Loader*.

## Conclusions

The E1 operating system can be used as an operating system for clusters of workstations. Currently most of the computing clusters are using Linux operating system plus PVM parallel programming library as their software platform. Linux was not designed to support parallel distributed computing. In contrast, E1 provides the necessary set of primitives on the operating system level rather than on library level, which will allow to significantly improve performance and reliability of a distributed system.

E1 can also be used as an operating system for distributed data processing systems, e.g., distributed CAD systems. In order to support decentralized processing of huge amount of data, these systems require resources to be highly available across the network. E1 encourages developing these systems within the framework of its component model. The replication mechanism will then enable efficient access to resources in distributed environment, and persistence of objects will improve reliability of the system and provide developers with a simple programming model.

In general, the E1 architecture is rather universal and can be used in a variety of distributed environments including even corporate networks of personal computers. E1 can also be used in heterogeneous environments, consisting of high-performance servers sharing their resources with low-end clients, like personal or portable computers and handhelds.

А.Ю. Бурцев, Л.Б. Рыжик, Ю.А. Тимошенко

АРХИТЕКТУРА ОПЕРАЦИОННОЙ СИСТЕМЫ НА ОСНОВЕ РАСПРЕДЕЛЕННЫХ ОБЪЕКТОВ

Предложена архитектура операционной системы E1, управляющей ресурсами ЭВМ, объединенных сетью, для наиболее полного использования вычислительного потенциала. Архитектура E1 основана на абстракции распределенного объекта, что позволяет реализовать эффективный и надежный доступ к ресурсам вычислительного комплекса из всех его узлов, скрывая при этом распределенную природу комплекса.

A.Y. Burtsev, L.B. Ryzhyk, Y.A. Timoshenko

OPERATING SYSTEM ARCHITECTURE BASED ON DISTRIBUTED OBJECTS

The paper presents architecture of E1 operating system, aimed to control resources of a computer network. The goal of the operating system design is to use the computing potential of the network in full measure. The E1 architecture is based on the abstraction of the distributed object, suggested by the authors. The distributed objects allow implementing efficient and reliable access to all resources of a distributed system, while hiding their distributed nature.

1. *Sunderam V.S* PVM: A Framework for Parallel Distributed Computing // Journal of Concurrency: Practice and Experience. – 1990. – December. – P. 315–339.
2. *MPI*: A Message-passing interface standard // International J. Supercomputing Applications. – 1994.
3. *Wulf W.A., Levin R., Harbison S.P.* HYDRA/C.mmp An Experimental Computer System. – N.-Y.: McGraw-Hill, 1981.
4. *Li K.* Shared Virtual Memory on Loosely Coupled Multiprocessors: PhD thesis. – 1986. – Yale.
5. *Intel* Corporation. i486 Processor Programmer's Reference Manual. – 1990.
6. *Kane G., Heinrich J.* MIPS RISC Architecture. – New Jersey: Prentice Hall, 1992.
7. *Intel Corporation.* IA-64 Architecture Software Developer's Manual. Volume 2: IA-64 System Architecture, Revision 1.1. – 2000.
8. *Shapiro J., Smith J., Farber D.* EROS: A Fast Capability System // 17th ACM Symposium on Operating System Principles (SOSP'99). – Charleston, USA, 1999.
9. *Ford B., Lepreau J.* Evolving Mach 3.0 to use migrating threads // Technical Report UUCS-93-022. – University of Utah, 1993.
10. *Ford B., Lepreau J.* Microkernels Should Support Passive Objects // Proc. of I-WOOOS'93.
11. *Tevanian A.* Architecture-Independent Virtual Memory Management for Parallel and Distributed Environments: The Mach Approach: PhD thesis. – Carnegie Mellon University, Department of Computer Science, 1987.